Instituto Tecnológico de Acapulco

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ACAPULCO

DESARROLLO DE UN SISTEMA DE INFORMACIÓN WEB PARA EL SEGUIMIENTO Y CONTROL DEL MANTENIMIENTO DE INFRAESTRUCTURA Y EQUIPO

Caso: Instituto Tecnológico de San Marcos

TITULACIÓN INTEGRAL

TESIS PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:
MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:
ING. ROGELIO RAMÍREZ SILVA

DIRECTOR:
M.C. FRANCISCO JAVIER GUTIÉRREZ MATA

CODIRECTOR:
M.I.D.S. ALMA DELIA DE JESÚS ISLAO

ACAPULCO, GRO., DICIEMBRE 2020.

El presente trabajo de tesis fue desarrollado en la *División de Estudios de Posgrado e Investigación del Instituto Tecnológico de* Acapulco, perteneciente al Programa Nacional de Posgrado de Calidad (PNPC-CONACyT).

Con domicilio para recibir y oír notificaciones en AV. Instituto Tecnológico de Acapulco s/n, Crucero del Cayaco, Acapulco, Guerrero, México. C.P. 39905.

Becario:	Rogelio Ramírez Silva.
CVU:	928497.
Núm. de apoyo:	711616.
Grado:	Maestría.

DESCARGO DE RESPONSABILIDAD INSTITUCIONAL

Quien suscribe declara que el presente documento titulado "DESARROLLO DE UN SISTEMA DE INFORMACIÓN WEB PARA EL SEGUIMIENTO Y CONTROL DEL MANTENIMIENTO DE INFRAESTRUCTURA Y EQUIPO" es un trabajo propio y original, el cual no ha sido utilizado anteriormente en institución alguna para propósitos de evaluación, publicación y/o obtención de algún grado académico.

Además, se adelanta que se han recogido todas las fuentes de información utilizadas, las cuales han sido citadas en la sección de referencias bibliográfica de este trabajo.

Acapulco, Gro; a 12 de diciembre de 2020.

ROGELIO RAMÍREZ SILVA

Nombre y Firma

CARTA DE CESIÓN DE DERECHOS DE AUTOR

El que suscribe: ROGELIO RAMÍREZ SILVA, autor del trabajo escrito de evaluación profesional en la opción de Tesis Profesional de Maestría con el título "DESARROLLO DE UN SISTEMA DE INFORMACIÓN WEB PARA EL SEGUIMIENTO Y CONTROL DEL MANTENIMIENTO DE INFRAESTRUCTURA Y EQUIPO", por medio de la presente con fundamento en lo dispuesto en los artículos 5, 18, 24, 25, 27, 30, 32 y 148 de la Ley Federal de Derechos de Autor, así como los numerales 2.15.5 de los lineamientos para la Operación de los Estudios de Posgrado; manifiesto mi autoría y originalidad de la obra mencionada que se presentó en la División de Estudios de Posgrado e Investigación, para ser evaluada con el fin de obtener el Título Profesional de Maestro en Sistemas Computacionales. Así mismo expreso mi conformidad de ceder los derechos de reproducción, difusión y circulación de esta obra, en forma NO EXCLUSIVA, al Tecnológico Nacional de México campus Acapulco; se podrá realizar a nivel nacional e internacional, de manera parcial o total a través de cualquier medio de información que sea susceptible para ello, en una o varias ocasiones, así como en cualquier soporte documental, todo ello siempre y cuando sus fines sean académicos, humanísticos, tecnológicos, históricos, artísticos, sociales, científicos u otra manifestación de la cultura.

Entendiendo que dicha cesión no genera obligación alguna para el Tecnológico Nacional de México campus Acapulco y que podrá o no ejercer los derechos cedidos. Por lo que el autor da su consentimiento para la publicación de su trabajo escrito de evaluación profesional.

Se firma presente en la ciudad de Acapulco de Juárez, estado de Guerrero a los 12 días del mes de diciembre de 2020.

ROGELIO RAMÍREZ SILVA
NOMBRE Y FIRMA

Instituto Tecnológico de Acapulco
División de Estudios de Posgrado e Investigación

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Acapulco, Gro; a 7 de diciembre de 2020.

AUTORIZACIÓN DE IMPRESIÓN DE TESIS

Los abajo firmantes, miembros de la comisión revisora de tesis designada por la División de Estudios de Posgrado e Investigación del Tecnológico Nacional de México campus Acapulco para la evaluación de la tesis del alumno **Rogelio Ramírez Silva**, manifiestan que después de haber revisado su tesis: "**Desarrollo de un sistema de información web para el seguimiento y control del mantenimiento de infraestructura y equipo**" desarrollada bajo la dirección del DIRECTOR, y el CO-DIRECTOR, el trabajo se **ACEPTA** para proceder a su impresión.

A T E N T A M E N T E

M.C. Francisco Javier Gutiérrez Mata
Cédula Profesional: 11429189

M.I.D.S. Alma Delia de Jesús Islao
Cédula Profesional: 8604126

M.I.T. Eloy Cadena Mendoza
Cédula Profesional: 5181429

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE ACAPULCO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Enterado

Dr. Eduardo de la Cruz Gámez
Coordinador de la Maestría en Sistemas Computacionales

e-mail de contacto: depi_acapulco@tecnm.mx
Teléfonos: (744) 4429010

Instituto Tecnológico de Acapulco
División de Estudios de Posgrado e Investigación

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Acapulco Gro., 8/Diciembre/2020

NO. OFICIO: DEPI-211/2020

ASUNTO:
AUTORIZACIÓN DE
IMPRESIÓN DE TESIS PROFESIONAL

C. ROGELIO RAMÍREZ SILVA

De acuerdo al reglamento de los Institutos Tecnológicos, dependiente de la Secretaría de Educación Pública y habiendo cumplido con todos los requisitos normativos respecto a su trabajo para titulación, Opción Titulación Tesis Profesional, con el proyecto titulado: DESARROLLO DE UN SISTEMA DE INFORMACIÓN WEB PARA EL SEGUIMIENTO Y CONTROL DEL MANTENIMIENTO DE INFRAESTRUCTURA Y EQUIPO. Se **CONCEDE** la **AUTORIZACIÓN** para que proceda a la impresión del mismo.

Sin otro particular por el momento, me es grato quedar de usted.

A T E N T A M E N T E "
Educación Tecnológica con Compromiso Social"

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE ACAPULCO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUARDO DE LA CRUZ GÁMEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

C.c.p. Expediente

EDG/stv

Av. Instituto Tecnológico s/n Crucero del Cayaco C.P. 39905
e-mail de contacto: depi_acapulco@tecnm.mx
Teléfonos: (744) 4429010 al 19 ext. 121
www.it-acapulco.edu.mx

Descargo de responsabilidad Institucional

El que suscribe declara que el presente documento titulado “Desarrollo de un sistema de información web para el seguimiento y control del mantenimiento de infraestructura y equipo” es un trabajo propio y original, el cuál no ha sido utilizado anteriormente en institución alguna para propósitos de evaluación, publicación y/o obtención de algún grado académico.

Además, se han recogido todas las fuentes de información utilizadas, las cuales han sido citadas en la sección de referencias bibliográfica de este trabajo.

12 de diciembre de 2020

Nombre: Ing. Rogelio Ramírez Silva

Fecha y firma

Agradecimientos

Agradezco antes que nada a Dios, por las bendiciones que derrama sobre mí,
Cada día de mi vida, gracias mi Dios por darme
la paciencia y entendimiento para realizar
este proyecto.

A mi familia, que siempre están al pendiente de mí, gracias mamá
por tus palabras de aliento, gracias papá por creer en mí.

Al Consejo Nacional de Ciencia y Tecnología por haberme apoyado económicamente
para materializar este proyecto, que me dio más experiencia y
conocimientos.

A mi director de tesis, el M.C. Francisco Javier Gutiérrez Mata por todas
las oportunidades brindadas desde que lo conozco, por sus aportes,
consejos y apoyo para la realización de este proyecto.

Al departamento de Posgrado e Investigación del Instituto Tecnológico de Acapulco, y
a todos los maestros que día con día me enseñaron
a ser mejor profesionista.

Dedicatoria

A todos mis amigos, José Raúl, Alejandro, José Alberto y Ángel Rayo, por el apoyo y compañía brindada en esta etapa de nuestras vidas, en el departamento de Posgrado e Investigación del Instituto Tecnológico de Acapulco.

A mi amiga gato gordo (Crisol) por la orientación y compañía durante mi estancia en la MSC.

Al Ing. Hugo Rojas por compartir sus conocimientos en materia de desarrollo de software.

A Yuridia Salas Ramírez por su apoyo y paciencia durante mi estancia en la maestría.

Resumen

En el presente documento se describe el desarrollo de un sistema web que permite controlar y dar seguimiento a las anomalías encontradas en los edificios y/o equipos del Instituto Tecnológico de San Marcos, basados en el procedimiento proporcionado por el **TecNM**, para el mantenimiento preventivo y correctivo de infraestructuras y equipos.

El sistema propuesto aborda una problemática actual que consiste en la necesidad de consultar información detallada del estado actual de las instalaciones del Tecnológico de San Marcos, la necesidad surge a partir de que no se cuenta con una plataforma que centralice la información de los mantenimientos, debido a eso la perdida de información por accidentes o descuidos es muy frecuente. Por tal motivo este trabajo de tesis propone el desarrollo de un sistema web que haciendo uso de las tecnologías de programación del marco de desarrollo ASP.NET Core MVC, y metodologías utilizadas para el desarrollo de software den solución a una gran parte de la problemática.

La implementación del sistema propuesto, permitirá que los departamentos que hagan uso del sistema, puedan gestionar de una mejor manera las anomalías reportadas cada semestre, obteniendo beneficios, tales como, búsquedas más rápidas de información, orden en los documentos generados por el procedimiento y una mejor toma de decisiones.

Abstract

This document describes the development of a web system that allows controlling and monitoring the anomalies found in the buildings and / or equipment of the Instituto Tecnológico de San Marcos, based on the procedure provided by the TecNM, for preventive maintenance and corrective infrastructure and equipment.

The proposed system addresses a current problem that consists of the need to consult detailed information on the current state of the facilities of the Instituto Tecnológico de San Marcos, the need arises from the fact that there is no platform to centralize the maintenance information, due to Therefore, the loss of information due to accidents or oversights is very frequent. For this reason, this thesis work proposes the development of a web system that, using the programming technologies of the ASP.NET Core MVC development framework, and software development methodologies provide a solution to a large part of the problem.

The implementation of the proposed system will allow the departments that make use of the system to better manage the anomalies reported each semester, obtaining benefits, such as faster information searches, order in the documents generated by the procedure and a better decision making.

Índice de contenido

Descargo de responsabilidad Institucional ... vii
Agradecimientos ... viii
Dedicatoria ... ix
Resumen ... x
Abstract ... xi
Capítulo 1 Introducción ... 1
1.1 Estado del arte ... 1
1.2 Planteamiento del Problema ... 3
1.3 Objetivo General ... 4
1.4 Objetivos Específicos ... 4
1.5 Hipótesis ... 4
1.6 Justificación ... 4
1.7 Alcances y Limitaciones ... 5
1.7.1 Alcances ... 5
1.7.2 Limitaciones ... 5
Capítulo 2 Marco teórico ... 7
2.1 Ingeniería del software ... 7
2.2 Modelo de desarrollo del software ... 7
2.2.1 Scrum ... 8
2.2.2 WSDM (*Web Design Method*) ... 10
2.2.3 SOHDM (*Scenario-Based Object-Oriented Hypermedia Design Methodology*) 11
2.2.4 WAE (*Web Application Extension*) ... 11
2.2.5 IWeb (Ingeniería Web) ... 11
2.2.6 OOHDM (*Object Oriented Hypermedia Design Methodology*) ... 11
2.2 Lenguaje Unificado de Modelado ... 15
2.2.1 Casos de uso ... 15
2.2.2 Diagrama de clase ... 18
2.2.3 Diagrama de estado ... 20
2.2.4 Diagrama de navegación ... 22
2.3 Arquitectura del software ... 23
2.3.1 Arquitectura de tres capas ... 24
2.3.2 Arquitectura de repositorio ... 25

2.4 Herramientas de desarrollo del software ... 26
2.4.1 ASP.NET Core 2 ... 26
2.4.2 Lenguaje C# ... 27
2.4.3 Entity Framework Core ... 28
2.4.4 Linq (*Language Integrated Query*) ... 29
2.4.5 Patrón de Repositorio (*Pattern Repository*) ... 30
2.4.6 Patrón Unidad de Trabajo (*Pattern Unit Of Work*) ... 31
2.4.7 PostgreSQL ... 32
2.4.8 Angular 7 ... 33
2.4.9 Bootstrap ... 33
2.4.10 Visual Studio ... 33
2.4.11 AdminLTE2 ... 34
Capítulo 3 Análisis y Diseño ... 35
3.1 Análisis ... 35
3.1.1 Historias de usuario ... 35
3.1.2 Modelado de negocios ... 37
3.1.3 Modelado de casos de uso ... 39
3.2 Diseño ... 43
3.2.1 Arquitectura de la aplicación ... 44
3.2.2 Diagrama de clases ... 46
3.2.3 Diagrama de estado ... 53
3.2.4 Diagramas de navegación ... 54
3.2.5 Diagrama de despliegue ... 56
Capítulo 4 Implementación ... 58
4.1 Configuración requerida para el desarrollo de la aplicación ... 59
4.2 Instalación de ASP.NET Core 2 ... 60
4.3 Instalación de Angular 9 ... 64
4.4 Sprint 1: Implementación de la capa de acceso a datos ... 68
4.4.1 Instalación de los paquetes en el proyecto Soportec ... 69
4.4.2 Creación de la capa de acceso a datos ... 72
4.5 Sprint 2: Implementación del módulo de usuarios ... 81
4.5.1 Back-End: Implementación del controlador para la gestión de usuarios de la aplicación ... 82
4.5.2 Back-End: Implementación del controlador para el inicio de sesión de los usuarios de la aplicación ... 85

4.5.2 Front-End: Formularios para el registro de los usuarios de la aplicación 87
4.5.3 Front-End: Formulario utilizado para el inicio de sesión de los usuarios de la aplicación 89
4.6 Sprint 3: Implementación del módulo de catálogos 91
4.6.1 Back-End: Implementación de las funcionalidades para la gestión de catálogos 92
4.6.2 Front-End: Implementación del formulario para la creación de registros en un catálogo 99
4.7 Sprint 4: Implementación del módulo de listas de verificación 101
4.7.1 Configurar la aplicación para persistir la información haciendo uso de los controladores 103
4.7.2 Back-End: Implementación de las funcionalidades para la gestión de las listas de verificación 104
4.7.3 Back-End: Implementación de las funcionalidades para la gestión de anomalías encontradas en las infraestructuras y equipos 108
4.7.4 Front-End: Implementación de las vistas para mostrar las listas de verificación creadas 112
4.7.5 Front-End: Implementación del formulario para la creación de una lista de verificación 114
4.7.6 Front-End: Implementación del formulario para el registro de una anomalía .. 115
4.8 Sprint 5: Implementación del módulo de planes de trabajo 117
4.8.1 Back-End: Configuración del controlador del módulo para acceder a la información de la base de datos 118
4.8.2 Back-End: Implementación de las funcionalidades para la gestión de planes de trabajo 119
4.8.3 Back-End: Implementación de los formularios para la gestión de los planes de trabajo 123
4.8.4 Implementación del formulario para la generación de formato pdf del plan de trabajo 127
Capítulo 5 Resultados 130
5.1 Inicializando la aplicación Soportec 130
5.2 Caso de estudio 1: Gestión de usuarios (HU-1) 131
5.3 Caso de estudio 2: Gestión de catálogo (HU-2) 135
5.4 Caso de estudio 3: Gestión de listas de verificación (HU-4, HU-5 y HU-6) 139
5.5 Caso de estudio 4: Gestión de planes de trabajo (HU-7, HU-8, HU-9 y HU-10) 148
Capítulo 6 Conclusiones 155
Referencias 156

Índice de ilustraciones

Ilustración 2.1 Modelo de desarrollo utilizando Scrum 9
Ilustración 2.2 Fases de la metodología OOHDM 12
Ilustración 2.3 Metodología adaptada para el desarrollo del sistema. 14
Ilustración 2.4 Casos de uso Administración de usuarios (Neustadt, 2005). 16
Ilustración 2.5 Plantilla para describir un caso de uso (Neustadt, 2005). 17
Ilustración 2.6 Casos de uso con simbología más utilizada en el diseño de casos de uso (James Rumbaugh, 2000). 17
Ilustración 2.7 Clase caballo 19
Ilustración 2.8 Clases relacionadas 20
Ilustración 2.9 Multiplicidad entre clases. 20
Ilustración 2.10 Diagrama de estado de un horno de microondas (Sommerville, 2011). 21
Ilustración 2.11Diagrama de navegación (München, 2010). 22
Ilustración 2.12 Nombres de los estereotipos y su simbología. 23
Ilustración 2.13 Arquitectura de aplicación en 3 capas con el patrón MVC (Sommerville, 2011). 25
Ilustración 2.14 Arquitectura de repositorio para un IDE. 26
Ilustración 2.15 Entity Framework Core en contexto. 28
Ilustración 2.16 Enfoque databasefirst. 29
Ilustración 2.17 Enfoque codefirst. 29
Ilustración 2.18 Patrón Repositorio (Microsoft, 2013). 31
Ilustración 2.19 Unidad de Trabajo (Elaboración propia). 32
Ilustración 3.1 Modelado de negocios de procedimiento de mantenimiento del Instituto Tecnológico de San Marcos. 38
Ilustración 3.2 Actores involucrados 39
Ilustración 3.3 Casos de uso para la gestión de verificaciones de infraestructuras y equipos. 41
Ilustración 3.4Casos de uso para la gestión del plan de trabajo. 41
Ilustración 3.5Casos de uso para la gestión de catálogo. 43
Ilustración 3.6 Arquitectura lógica de la herramienta SOPORTEC. 45
Ilustración 3.7 Diagrama de clases de SOPORTEC. 47
Ilustración 3.8 Diagrama de clases del patrón Repository. 52
Ilustración 3.9 Interfaces que dictan los métodos que serán heredados en cada clase del modelo de dominio. 53
Ilustración 3.10 Diagrama de estado de una anomalía (incidencia). 54
Ilustración 3.11 Menú de navegación de un usuario con el rol de Auxiliar. 55
Ilustración 3.12 Menú de navegación de un usuario con el rol de Administrador. 55
Ilustración 3.13 Menú de navegación de un usuario con el rol de Coordinador. 56
Ilustración 3.14 Diagrama de despliegue de soportec. 56
Ilustración 4.1Página oficial para la descarga del SDK de ASP.NET Core 2.2 (2019). 60
Ilustración 4.2 Proceso de instalación del SDK de ASP.NET Core 2. 61
Ilustración 4.3 Ventana de instalación finalizada de SDK de ASP.NET Core 2. 61
Ilustración 4.4 Asistente de Visual Studio para crear un proyecto. 62

Ilustración 4.5 Ventana para seleccionar el tipo de proyecto a crear. 62
Ilustración 4.6 Ventana para proporcionar nombre y ubicación del proyecto..................... 63
Ilustración 4.7 Ventana para la selección de la plantilla de la aplicación. 63
Ilustración 4.8 Página oficial para descargar el Motor de NodeJS (2019). 64
Ilustración 4.9 Asistente de instalación de NodeJs.. 65
Ilustración 4.10 Opciones se instalación de NodeJS (2019). 65
Ilustración 4.11 Instalación de NodeJS en proceso. 66
Ilustración 4.12 Instalación de NodeJS terminada. 66
Ilustración 4.13 Comando para la instalación de Angular CLI. 67
Ilustración 4.14 Instalando Angular CLI. 67
Ilustración 4.15 Proceso de instalación de Angular CLI terminado. 67
Ilustración 4.16 Creando proyecto en Angular CLI. 68
Ilustración 4.17 Proceso de instalación de librerías a un proyecto. 70
Ilustración 4.18 Administrador de paquetes NuGet (Visual Studio 2019). 71
Ilustración 4.19 Ventana de confirmación para instalar librerías. 71
Ilustración 4.20 Proceso para agregar un nuevo proyecto a la solución. 72
Ilustración 4.21 Asistente de Visual Studio para crear un nuevo proyecto. 73
Ilustración 4.22 Solución que contiene los proyectos Soportec.DAL 73
Ilustración 4.23 Clases de modelo de dominio de la aplicación. 74
Ilustración 4.24 Configuración de una clase para traducirla a tabla en la base de datos. 74
Ilustración 4.25 Referencia de librerías y clases. 74
Ilustración 4.26 Configuración de la relación entre clases. 75
Ilustración 4.27 Configuración de la clase contexto de datos. 75
Ilustración 4.28 Propiedades que serán convertidas a tablas en la base de datos. 76
Ilustración 4.29 Cadena de conexión a la base de datos de la aplicación. 76
Ilustración 4.30 Configuración de la cadena de conexión en Startup. 76
Ilustración 4.31 Interfaz IRepository para el acceso a datos. 77
Ilustración 4.32 Implementación de la clase Repository. 78
Ilustración 4.33 implementación de las propiedades de UnitOfWork (Unidad de trabajo). 78
Ilustración 4.34 Implementación de los métodos de UnitOfWork. 79
Ilustración 4.35 Migración de clases de la aplicación. 79
Ilustración 4.36 Traduciendo lenguaje orientado a objeto a instrucciones SQL. 79
Ilustración 4.37 Instrucción para migrar las clases a una base de datos. 79
Ilustración 4.38 Log de migraciones. 80
Ilustración 4.39 Comparación de las clases con las tablas en la base de daos después de la migración. 80
Ilustración 4.40 Implementación del controlador UsuarioController. 82
Ilustración 4.41 Configuración de la inyección de dependencias y servicios. 83
Ilustración 4.42 Método Post para crear y actualiza usuarios. 84
Ilustración 4.43 Mapeador creado para transformar los tipos de datos. 85
Ilustración 4.44 Modelos de datos que representa al usuario almacenado en la base de datos. 85
Ilustración 4.45 Método para login de usuario. 86
Ilustración 4.46 Método para crear un token. 87
Ilustración 4.47 Formulario para registrar usuarios. 87
Ilustración 4.48 Función save que se comunica con el back-end para persistir información. 88

Ilustración 4.49 Código HTML que muestra los usuarios registrados. 89
Ilustración 4.50 Método que llena la tabla de usuarios. 89
Ilustración 4.51Códido HTML para el formulario de Login. 90
Ilustración 4.52 Lógica del lado del cliente del formulario de login. 90
Ilustración 4.53 Clase de dominio que almacena Áreas. 92
Ilustración 4.54 Clase que almacena problemas más comunes. 93
Ilustración 4.55 Clase que almacena información básica de proveedores. 93
Ilustración 4.56 Clase almacena información de los servicios realizados. 93
Ilustración 4.57 Configuración del AreaController. 94
Ilustración 4.58 Método para obtener todos los departamentos. 95
Ilustración 4.59 Método para buscar usuario por Id. 96
Ilustración 4.60 Método para crear un nuevo registro. 97
Ilustración 4.61 Método para actualizar un área existente. 97
Ilustración 4.62 Método para consultar áreas paginadas. 98
Ilustración 4.63 Método para eliminar un registro. 98
Ilustración 4.64 Formulario HTML para la creación de un área. 99
Ilustración 4.65 Lógica del formulario para crear registro. 100
Ilustración 4.66 Botón para crear un registro con el formulario. 100
Ilustración 4.67 Lógica del front-end para actualizar registros. 101
Ilustración 4.68 Configurando Repositorios. 103
Ilustración 4.69 Configuración del controlador Revisión. 103
Ilustración 4.70 Método para consultar todas las listas de verificación. 104
Ilustración 4.71 Método que obtiene información paginada. 104
Ilustración 4.72 Obtener un registro con el identificador único de la clase que lo implementa. 105
Ilustración 4.73 Insertando un registro en el controlador RevisionController. 106
Ilustración 4.74 Clase que almacena datos de las revisiones. 106
Ilustración 4.75 Método para actualizar listas de verificación. 107
Ilustración 4.76 Método para la búsqueda de una revisión por periodo. 107
Ilustración 4.77 Método para eliminar registros. 108
Ilustración 4.78 Clase Ticket que permite almacena una anomalía. 109
Ilustración 4.79 Método que permite registrar una anomalía. 110
Ilustración 4.80 Método que permite actualizar una anomalía 110
Ilustración 4.81 Método que permite obtener un conjunto de anomalías. 111
Ilustración 4.82 Método que permite buscar una anomalía. 111
Ilustración 4.83 Código HTML utilizado para mostrar en pantalla una lista de verificación. 112
Ilustración 4.84 Código HTML que muestra las opciones que se pueden ejecutar en una verificación. 113
Ilustración 4.85 Método utilizado para extraer las verificaciones del back-end. 113
Ilustración 4.86 Lógica para carga un registro a actualizar. 114
Ilustración 4.87 Formulario para la creación y actualización de listas de verificaciones. 114
Ilustración 4.88 Método para enviar los datos del formulario al back-end. 115
Ilustración 4.89 Formulario para registrar una anomalía. 116
Ilustración 4.90 Lógica del formulario del registro de una anomalía. 116
Ilustración 4.91 Controlador plan de trabajo. 118
Ilustración 4.92 Modelo de dominio del plan de trabajos. 118

Ilustración 4.93 Método para registrar un plan de trabajo. 119
Ilustración 4.94 Registro de un plan de trabajo. 120
Ilustración 4.95 Actualización de un plan de trabajo. 120
Ilustración 4.96 Método para buscar un plan por id. 121
Ilustración 4.97 obtener planes de trabajo. 121
Ilustración 4.98 Operación interna de la paginación. 122
Ilustración 4.99 Método para buscar plan por periodo. 122
Ilustración 4.100 Código HTML para listar los planes de trabajo. 123
Ilustración 4.101 Método para cambiar de estado un plan de trabajo. 124
Ilustración 4.102 Método para obtener los registros en la tabla. 124
Ilustración 4.103 Formulario para realizar una búsqueda por periodo. 125
Ilustración 4.104 Método que realiza búsqueda. 125
Ilustración 4.105 Botón para crear un nuevo plan de trabajo. 125
Ilustración 4.106 Método para agregar un nuevo plan 126
Ilustración 4.107 Código HTML del formulario para agregar planes de trabajo. 126
Ilustración 4.108 Método que agrega un plan de trabajo. 126
Ilustración 4.109 Método para editar un plan de trabajo. 127
Ilustración 4.110 Cabecera del formato pdf de los planes de trabajo. 127
Ilustración 4.111 Título del formato pdf del plan de trabajo. 128
Ilustración 4.112 Cuerpo del formato pdf. 128
Ilustración 4.113 Botón para descargar el pdf del plan de trabajo. 128
Ilustración 4.114 Lógica del lado del frontend para descargar el pdf. 129
Ilustración 5.1 Aplicación Soportec en Visual Studio. 130
Ilustración 5.2 Salida de depuración y compilación del proyecto Soportec. 130
Ilustración 5.3 Inicio de sesión de la aplicación Soportec. 131
Ilustración 5.4 Icono del navegador Firefox Mozilla. 132
Ilustración 5.5 Barra de direcciones del navegador Firefox Mozilla. 132
Ilustración 5.6 Opción de la aplicación para ir a la ventana de usuarios. 132
Ilustración 5.7 Botón para abrir ventana y registrar usuario. 133
Ilustración 5.8 Ventana para registrar usuario. 133
Ilustración 5.9 Usuarios registrados en la aplicación. 133
Ilustración 5.10 Usuario ingresando sus credenciales para iniciar sesión. 134
Ilustración 5.11 Inicio de sesión en la aplicación Soportec. 134
Ilustración 5.12 Consulta del usuario registrado. 135
Ilustración 5.13 Icono del navegador Firefox Mozilla. 136
Ilustración 5.14 Barra de navegación de Firefox. 136
Ilustración 5.15 Inicio de sesión del usuario Rogelio Ramírez. 136
Ilustración 5.16 Menú de la plataforma Soportec. 137
Ilustración 5.17 Botón para agregar un nuevo departamento. 137
Ilustración 5.18 Formulario para registrar un nuevo departamento. 138
Ilustración 5.19 Listado de departamentos registrados. 138
Ilustración 5.20 Tabla area con los registros que contiene. 139
Ilustración 5.21 Icono del navegador Firefox Mozilla. 140
Ilustración 5.22 Barra de navegación de Firefox. 140
Ilustración 5.23 Inicio de sesión del usuario Rogelio Ramírez. 140
Ilustración 5.24 Menú de la plataforma Soportec. 141
Ilustración 5.25 Botón para agregar una lista de verificación. 141

Ilustración 5.26 Formulario para registrar una nueva lista de verificación. 142
Ilustración 5.27 Listas de registros de las verificaciones. 142
Ilustración 5.28 Formulario para editar una verificación. 142
Ilustración 5.29 Formato pdf de la lista de verificación. 143
Ilustración 5.30 Lista de anomalías. 143
Ilustración 5.31 Formulario para registrar una anomalía. 144
Ilustración 5.32 Anomalía insertada en la lista de verificación. 144
Ilustración 5.33 Anomalía detallada. 145
Ilustración 5.34 Programación de orden de trabajo. 145
Ilustración 5.35 Detalle de la orden de trabajo. 146
Ilustración 5.36 Alerta indicando que se actualizó el detalle de la anomalía. 147
Ilustración 5.37 Formato pdf de la lista de verificación con dos registros. 147
Ilustración 5.38 Persistencia de la información de la verificación. 147
Ilustración 5.39 Persistencia de la información de una anomalía. 147
Ilustración 5.40 Icono del navegador Firefox Mozilla. 149
Ilustración 5.41 Barra de navegación de Firefox. 149
Ilustración 5.42 Inicio de sesión del usuario Rogelio Ramírez. 149
Ilustración 5.43 Menú de la plataforma Soportec. 149
Ilustración 5.44 Opción para agregar un nuevo plan de trabajo. 150
Ilustración 5.45 Lista de planes de trabajo. 150
Ilustración 5.46 Buscando plan por periodo. 151
Ilustración 5.47 Planes de trabajo cerrados. 151
Ilustración 5.48 Asignando a una anomalía un plan de trabajo activo. 152
Ilustración 5.49 Fallo sin asignación de planes de trabajo. 152
Ilustración 5.50 Pdf generado del plan de trabajo. 153
Ilustración 5.51 Descarga del formato pdf con información de los fallos. 153
Ilustración 5.52 Información almacenada en la tabla que almacena los planes de trabajo. 153

Índice de tablas

Tabla 2.1 Productos y formalismos de la metodología OOHDM. 13
Tabla 2.2 Comparación de requisitos en el entorno web contemplados en las (Ríos, 2018). 13
Tabla 2.3 Simbología de casos de uso (Elaboración propia). 17
Tabla 2.4 Estado y estímulos para el ejemplo de la ilustración 10. 21
Tabla 2.5 Proveedores de bases de datos y los paquetes NuGet para EF Core. 29
Tabla 3.1 Product Backlog proporcionado por el product owner del sistema. 36
Tabla 3.2 Clase para la gestión de planes de trabajo. 47
Tabla 3.3 Clase para la gestión de Usuarios. 48
Tabla 3.4 Clase para la gestión de Departamentos. 48
Tabla 3.5 Clase para la gestión de revisiones. 49
Tabla 3.6 Clase para la gestión de la relación de un plan con la incidencia. 49
Tabla 3.7 Clase para la gestión de tickets (incidencias). 50
Tabla 3.8 Clase para la gestión de los proveedores. 51
Tabla 3.9 Clase para la gestión de servicios. 51
Tabla 3.10 Clase para la gestión de problemas. 51
Tabla 4.1 Planeación de los Sprints para el desarrollo de la aplicación. 58
Tabla 4.2 Programas instalados para desarrollo de la aplicación. 59
Tabla 4.3 Requerimientos de hardware, programas y utilidades para el despliegue de la aplicación.. 59
Tabla 4.4 Planeación del Sprint 1. 68
Tabla 4.5 Paquetes necesarios para la implementación de la Capa de Acceso a datos (DAL). 70
Tabla 4.6 Planeación del Sprint 2. 81
Tabla 4.7 Planeación del Sprint 3. 91
Tabla 4.8 Planeación del Sprint 4. 101
Tabla 4.9 Planeación del Sprint 5. 117

Capítulo 1 Introducción

1.1 Estado del arte

Debido a los avances tecnológicos en los últimos años y al impacto que han tenido en la sociedad, muchas organizaciones han adoptado un nuevo modelo para realizar sus funciones cotidianas. La incorporación de nuevas tecnologías dentro de una organización trae consigo numerosas ventajas como la automatización de procesos, simplificación de tareas, disminución de costos de operación y/o producción, disminución en el tiempo de entrega del producto o agilización de servicios, incremento en la productividad, entre muchas otras cosas que la hacen más competitiva. Estas ventajas competitivas pueden lograrse con la ayuda de un sistema de cómputo, el cual es prescindible en organizaciones de tamaño considerable. Estos sistemas son capaces de gestionar peticiones específicas hechas por los usuarios (como consultar información en una Base de Datos, realizar operaciones matemáticas de gran escala o acceder a la información de una empresa de forma remota, entre otras) y responder de manera casi inmediata, dejando la posibilidad abierta para el continuo crecimiento de una organización.

En la actualidad, estos sistemas están disponibles para cualquier empresa, gracias a la constante evolución de los dispositivos de cómputo que día a día se vuelven más baratos, y a las tecnologías web que cada día se van desarrollando con mejores características que las hacen una herramienta sencilla de usar y de fácil acceso, un ejemplo de este tipo de herramientas son los sistemas que gestionan las solicitudes de servicio, administran los procedimientos de mantenimiento preventivo y correctivo de diferentes infraestructuras y/o equipos. Se ha ido creando varios tipos de software que garantizan una mejor gestión de los procesos de mantenimiento y disminución de los tiempos de respuesta entre las solicitudes de servicio y las actividades de mantenimiento.

Algunos de los antecedentes relacionados con el presente trabajo de tesis son los mencionados a continuación.

El primero es el articulo con el nombre "***Sistema automatizado para la gestión del mantenimiento de equipos (módulos administración y solicitud de servicio)"*** publicado en diciembre de 2015 en la revista Ciencias Técnicas Agropecuarias, por la ingeniera Yanelis Suárez Fragas en la Universidad Agraria de La Habana en la Facultad de Ciencias Técnicas, el artículo trata acerca del software llamado SGMANTE el cual fue desarrollado para apoyar el proceso de gestión de mantenimiento de equipos, dicho software cuenta con cinco módulos implementados y en este artículo solo se habla del desarrollo de dos módulos, el primero es el módulo de administración, el cual abarca todo el control de usuarios del sistema, así como los privilegios de los mismos para acceder al sistema. El módulo solicitudes de servicio es el encargado de gestionar todas las solicitudes que hacen las distintas áreas que solicitan algún tipo de mantenimiento llámese preventivo o correctivo.
También se analizan diferentes sistemas automatizados que se utilizan para gestionar el mantenimiento de distintos equipos en la industria. Los autores del artículo resaltan que la automatización trae como resultado un incremento en el rendimiento y control del activo fijo con el que cuenta una empresa o institución.

Se mencionan algunas herramientas que son muy útiles en el desarrollo de este tipo de software, las herramientas mencionadas son el Lenguaje Unificado de Modelado (UML) el cual sirve para hacer modelos de software. También se menciona una herramienta para el diseño y esquematización de los artefactos de software, es decir, para el diseño de casos de uso y diagramas de lógica del negocio. El programa que los autores desarrollaron es un programa está diseñado para dar mantenimiento solo a equipo de agricultura (Fragas, 2015).

El artículo con el nombre "***Sistema automatizado para la gestión del mantenimiento de equipos (módulos patrimonio y órdenes de trabajo)"*** es el segundo de dos artículo publicado el 13 de diciembre de 2015 por la ingeniera Yanelis Suárez Fragas, en la revista Ciencias Técnicas Agropecuarias en la Universidad Agraria de La Habana, en la Facultad de Ciencias Técnicas, según el autor del artículo, los principales sistemas de mantenimiento y reparación se han desarrollado para organizar, ejecutar y responder a las exigencias trayendo consigo grandes beneficios tanto técnicos como económicos a las instituciones. Hacer mantenimiento implica estar acorde con nuevos desarrollos tecnológicos y nuevos retos para los sectores industrial, comercial, servicio y agrario. Estos retos están asociados con la necesidad de optimizar la eficiencia y eficacia en la prestación de los servicios y el mejoramiento de la calidad. En este artículo se habla del desarrollo de dos módulos para implementarlos en el software SGMANTE, el primer módulo se llama módulo patrimonio es el que controla todo el equipamiento, es decir, es el que da de alta los equipos a los cuales se les dará mantenimiento. El segundo módulo se llama órdenes de trabajo que es el que gestiona todas las órdenes que se hagan por cada solicitud de mantenimiento de servicio. En este mismo artículo los autores muestran diagramas de caso de uso que ellos hicieron para explicar todas las funciones que tendrán los módulos desarrollados. El autor concluye que se diseñaron los módulos correspondientes tomando como guía la metodología de desarrollo de software RUP, debido a los diagramas que se generan como apoyo para el desarrollo de la aplicación (Fragas, 2015).

También es importante mencionar la tesis titulado ***"Estudio comparativo entre las metodologías ágiles y las metodologías tradicionales para la gestión de proyectos software"*** fue publicado en julio de 2015 por Manuel José García Rodríguez en la Universidad de Oviedo, la tesis trata sobre el análisis de las diferentes metodologías para el desarrollo de software que existen, en ella se abordan las metodologías tradicionales y las metodologías agiles. El autor hace un análisis de once metodologías dentro de las cuales está, una muy utilizada actualmente llamada Scrum, la cual se adapta a todo tipo de proyectos, no solo de ingeniería de software. El autor concluyó que la mejor metodología de software que se puede utilizar para el desarrollo de un proyecto, principalmente dependerá de la naturaleza del proyecto, es decir, si para desarrollar un proyecto no se cuenta con tiempo y además el proyecto se desarrolla con no más de seis personas, se recomienda que se utilice una metodología tradicional, pero si el proyecto que se pretende desarrollar está más orientado a la producción en masa de software, el autor recomienda que la metodología que se puede utilizar es Scrum (Rodríguez, 2015).

1.2 Planteamiento del Problema

En el Instituto Tecnológico de San Marcos existen dos departamentos los cuales son Centro de Cómputo, Recursos Materiales y servicios, los cuales se encargan de dar mantenimiento y soporte técnico a los equipos de aire acondicionado, equipos de cómputo, impresoras, redes eléctricas, instalación de equipos multimedia, también se encargan de hacer trabajos de herrería, carpintería, plomería, instalaciones eléctricas, reparación de lámparas, mantenimiento de aulas y muchos otros trabajos que son necesarios realizar para que las áreas del tecnológico lleven a cabo sus actividades de una manera eficiente.

Para que las actividades mencionadas anteriormente pueda realizarlas el departamento de Recursos y Servicios, el jefe del departamento que solicita un trabajo debe realizar una solicitud para dicho trabajo. Otro departamento que se mencionó es el Centro de Cómputo, el cual es el departamento encargado de atender a todas las áreas del Tecnológico que soliciten algún tipo de mantenimiento, de tipo preventivo o correctivo en los equipos de cómputo que utilizan los docentes y administrativos de dicha institución para desarrollar sus actividades diarias de trabajo, además el Centro de cómputo realiza otras actividades como la verificación de equipo al inicio de cada semestre, esto con el fin de saber cada seis meses el estado de los equipos, también el departamento de servicios hace un recorrido por todo el Tecnológico verificando el estado de las instalaciones de toda la institución, el centro de cómputo revisa todos los equipos de cómputo, y el departamento de Recursos Materiales y Servicios, todos los aíres acondicionados, puertas, infraestructura, instalaciones eléctricas, pintura de paredes y sillas , en ese recorrido cada departamento verifica el estado en el que se encuentran las instalaciones o equipos que les corresponde, y si existe algún problema con algún equipo de cómputo o de aire acondicionado se procede a repararlo en ese momento y si por algún motivo ese equipo no se puede reparar en el instante, se programa esa actividad para realizarla posteriormente, es decir, se anota el fallo y si se reparó o no el equipo, para posteriormente programarle una fecha en la que se atenderá dicho fallo, la fecha se proporciona dependiendo de los días disponibles en la agenda de trabajo.

Los dos departamentos se encargan de manejar una gran cantidad de información de las órdenes de trabajo que son solicitadas por el personal de la institución, toda esta información actualmente es recabada y archivada en un formato de registro el cual posteriormente se almacena en carpetas que están clasificados por semestre. La forma en que actualmente se está almacenando la información es de una manera tradicional utilizando formatos en papel y llenados a computadora en editores texto simple.

También es bueno mencionar que han existido perdidas de solicitudes de mantenimiento y de órdenes de trabajo, lo cuan como consecuencia no se realiza el trabajo que se solicitó y, por lo tanto, las actividades de otros departamentos son detenidas. Cabe mencionar que para entregar una solicitud de mantenimiento el encargado del departamento que lo solicita debe de ir físicamente hasta el departamento al cual quiere solicitar el mantenimiento de un equipo o infraestructura.

En ocasiones cuando se realizan auditorías a dichos departamentos, es necesario e importante tener toda la información a la mano en tiempo y forma o cuando se quiere

consultar si algún departamento hizo una petición de servicio, se tiene que buscar entre todas las carpetas que se tienen, en ocasiones este proceso suele tardar debido a que muchas veces la información no está guardada en orden o los formatos se pierden, este tipo de tareas requieren de tiempo para poder tener la información disponible. Si a todo esto se le agrega la cantidad de papel que se usa, es demasiada para una institución que está dentro de normas y estándares ISO14000 que es un conjunto de normas que tratan sobre el aspecto del ambiente, es decir, socialmente responsables con el medio ambiente y que sus procesos o desarrollo de actividades no deben consumir una gran cantidad de papel como lo hacen actualmente.

1.3 Objetivo General

Desarrollar un sistema de información web para el seguimiento y control del mantenimiento de infraestructura y equipo para el Instituto Tecnológico de San Marcos.

1.4 Objetivos Específicos

- Recopilar los requerimientos de las necesidades del software que requiere el Instituto Tecnológico de San Marcos.
- Seleccionar las herramientas de desarrollo y programación para la construcción del sistema propuesto.
- Diseñar y desarrollar los módulos que cumplan con los requerimientos proporcionados por el propietario (Instituto Tecnológico de San Marcos) del software.
- Demostrar a través de casos de estudio la funcionalidad de los módulos diseñados a partir de los requerimientos proporcionados por el propietario del software.

1.5 Hipótesis

El desarrollo del sistema de información web para el seguimiento y control del mantenimiento de infraestructuras y equipos brindará información detallada de las anomalías reportadas en los edificios y/o equipos del Instituto Tecnológico de San Marcos, facilitando al jefe del departamento que hagan uso del sistema, la centralización y obtención de las ordenes de trabajo de las anomalías reportadas.

1.6 Justificación

En la actualidad, es necesario que el manejo de la información sea procesada y almacenada de una forma efectiva para agilizar los procesos de verificación de infraestructura, generación de un plan de trabajo y así lograr un verdadero control de estas actividades, la idea principal es ejecutar las actividades con el menor esfuerzo humano posible y en la menor cantidad de tiempo posible. Las tecnologías que han surgido recientemente en materia de desarrollo web ofrecen grandes oportunidades para el desarrollo y avance de una institución educativa que está al servicio del público en general y que aporta profesionistas a diversos sectores empresariales.

Por lo tanto, la implementación de una plataforma web resuelve la ausencia de un sistema que permita controlar y dar seguimiento al mantenimiento de las infraestructuras y equipos con los que cuenta el Instituto Tecnológico de San Marcos, que para brindar un servicio de calidad sus instalaciones deben estar a la altura de cualquier institución del país que forma parte del Tecnológico Nacional de México y esto se logra manteniendo todas sus instalaciones en orden, bien cuidadas con los mantenimientos adecuados.
Con este sistema se podrán gestionar correctamente los mantenimientos de cada edificio, equipo de cómputo, servidores, puertas, ventanas, subestaciones eléctricas y todo tipo de producto o artículo que necesite un mantenimiento.

1.7 Alcances y Limitaciones

1.7.1 Alcances

- El sistema automatiza los procesos más importantes que se llevan a cabo en el Instituto Tecnológico de San Marcos en el procedimiento de mantenimiento de infraestructuras y equipos.

- El sistema almacenará la información de las solicitudes de mantenimiento preventivo, así como la información de los planes de trabajo generadas por los departamentos de Centro de Cómputo y, Recursos Materiales y Servicios que dan soporte en el Instituto Tecnológico de San Marcos.

1.7.2 Limitaciones

- El uso de este sistema no garantiza que el departamento de Centro de Cómputo, Recursos Materiales y Servicios dejen de usar papel al cien por ciento en sus actividades diarias.

- El sistema no administrará la parte operativa del Centro de Cómputo, Recursos Materiales y Servicios.

- El sistema no controla ni da seguimiento al proceso de solicitudes de mantenimiento correctivo.

Capítulo 2 Marco teórico

En este segundo capítulo de tesis se abordan los fundamentos teóricos que refuerzan el desarrollo del sistema. Se presentarán cada uno de los tópicos en el orden conforme se fue haciendo uso de cada uno ellos en el desarrollo del proyecto.

El contenido de este capítulo se encuentra dividido en cinco secciones, en la primera sección se menciona de que trata el desarrollo de software, en la segunda sección se describen las metodologías que fueron consideradas para el desarrollo del sistema y un marco de trabajo que dicta los pasos a seguir entre el cliente y los desarrolladores, en la tercera sección se describen los diagramas que apoyan y sustentan el análisis y diseño del sistema, en la cuarta sección se tratan los patrones arquitectónicos que se implementaran en la aplicación, en la quinta y última sección se describen los conceptos que son necesarios para el desarrollo del software, además de los **frameworks**, lenguajes de programación y sistema gestor de base de datos indispensable para persistir la información con la que trabajará la aplicación.

A continuación, como primer paso en el desarrollo de la aplicación es saber que disciplina se ocupa de fundamentar el análisis, diseño y hasta la implementación de software.

2.1 Ingeniería del software

La ingeniería de software es una disciplina que se interesa por todas las etapas por las que pasa la producción del software, es decir, desde el análisis, pasando por el diseño hasta las pruebas del software. A continuación, se citan otras definiciones por reconocidos autores:

- La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software (Sommerville, 2011).
- La Ingeniería del Software incluye la aplicación práctica del conocimiento científico en el diseño y construcción de los programas y la documentación requerida para su desarrollo, operación y mantenimiento (Boehm, 1976).
- La ingeniería de software está formada por un proceso, un conjunto de métodos (prácticas) y un arreglo de herramientas que permite a los profesionales elaborar software de cómputo de alta calidad (Roger S. Pressman, 2010).

Desarrollar software no es un trabajo sencillo, debido a que se deben realizar procesos muy detallados apoyados por técnicas y metodologías planteadas y probadas. En la siguiente sección se tratan las metodologías más comunes en el desarrollo de aplicaciones web.

2.2 Modelo de desarrollo del software

El primer paso en el desarrollo de una aplicación consiste en la elección de una metodología de desarrollo que nos dicte la secuencia de pasos a seguir para la implementación del software a desarrollar, desde la obtención de requerimientos, hasta la fase final, siendo esta última las pruebas de software en algunas metodologías.

El principal problema de desarrollar aplicaciones que no cuentan con buenos atributos de calidad siendo uno de los más importantes la robustez y escalabilidad, es el no hacer uso de metodologías que brindan una secuencia de pasos para la construcción del software, también es importante mencionar que el constante avance de la tecnología implica el aumento de la complejidad en diseñar, desarrollar, mantener y gestionar estos sistemas de información.

Debido a esto, los sistemas son construidos en lapsos de tiempo muy largos, que resultan costosos, incluso más de lo planeado en el presupuesto inicial, el resultado del software muchas veces no es lo que el cliente espera, la calidad se ve afectada y los proyectos quedan incompletos. Sin el apoyo de herramientas de trabajo adecuadas y haciendo uso de métodos o técnicas que no van de acuerdo con el problema que se necesita resolver, da como resultado el desarrollo de aplicaciones que no pasan las pruebas a las cuales la somete el propietario del software. A raíz de esto se han ido creando enfoques disciplinados, sistemáticos y metodologías donde se tienen en cuenta aspectos específicos del desarrollo de software.

Es importante mencionar que existe una variedad de metodologías que son utilizadas en el desarrollo de software, dichas metodologías dependerán de la naturaleza del desarrollo que se realizará. Actualmente existen diferentes grupos de metodologías por mencionar algunas, existen las tradicionales, agiles y web, donde las tradicionales son poco usadas y las agiles tiene un auge debido a que su naturaleza permiten hacer entregas de porciones del software en poco tiempo. Es importante mencionar que se puede crear una metodología diferente obteniendo lo mejor de cada una de las existentes, en otras palabras, se puede hacer una combinación de las tradicionales, agiles y obtener lo mejor de la metodología web.

Para el desarrollo del presente proyecto del cual trata esta tesis se hará uso del marco de trabajo Scrum el cual es un framework que permite orientar al cliente del software y al equipo que lo desarrolla, dictando una serie de pasos que los involucrados deben de seguir.

2.2.1 Scrum

El método Scrum permite hacer entregas del producto interactivamente, es decir, aproximadamente cada dos o tres semanadas se hace la entrega de una porción del software, también llamado incremento. Los que hacen uso de Scrum son equipos de desarrollo experimentados y comprometidos con la metodología, una sola persona puede hacer uso de Scrum, pero se debe de tener en cuenta que la misma persona tomará el papel de cada uno de los actores en Scrum. Se dice que es una metodología rigurosa porque desde que inicia un proyecto el equipo de desarrollo y todos los participantes se comprometen con la realización de juntas, participación en actividades y documentación de las actividades realizadas cada día, así como adoptar valores, principios y utilizar buenas prácticas.

Algunas definiciones proporcionadas por autores reconocidos dicen lo siguiente:

- El método de Scrum es un método ágil que ofrece un marco de referencia para la administración del proyecto. Se centra alrededor de un conjunto de Sprints, que son periodos fijos cuando se desarrolla un incremento de sistema (Sommerville, 2011).

- Scrum proviene de un juego que tiene lugar durante un partido de rugby, es un método de desarrollo ágil el cual es congruente con el manifiesto ágil, algunas de sus actividades son: requerimientos, análisis, diseño, evaluación y entrega (Roger S. Pressman, 2010).

Existen tres roles importantes en la metodología Scrum:

- **Propietario del producto (product owner):** Es la persona dueña o encargada del producto software que brinda toda la información necesaria para el desarrollo que lleva acabo el equipo de desarrollo. Una de sus funciones es otorgar los requerimientos y priorizarlos, es decir, indicar que funcionalidades deben ser entregadas, otra de sus funciones es aprobar el trabajo y los resultados en cada interacción, también puede rechazar el sprint o re-priorizar ciertos requerimientos, sin olvidar que es quien da la visión del producto.

- **Equipo de desarrollo (team):** Es un grupo de personas que se encarga de hacer análisis, diseño, codificación y realizar pruebas del producto. Una de sus actividades principales es decidir la cantidad de requerimientos que entran en un sprint (interacción). Una vez que el equipo tiene los requerimientos que se van a desarrollar, se organizan y cada uno desarrollara ciertas tareas según las habilidades de cada persona del grupo, es decir, son auto-organizados y multifuncionales debido a que no solo están sujetos a realizar una sola tarea.

- **Scrum master:** Es la persona encargada de velar porque se cumpla cada uno de los pasos que dicta Scrum, es su responsabilidad el éxito o fracaso de un proyecto o en su debido caso el éxito o fracaso de la entrega de una interacción. Se puede ver como el árbitro entre el propietario del software y los desarrolladores.

Es importante mencionar que Scrum no solo se usa en el desarrollo de software, sino que también en la arquitectura, ingeniería automotriz, para organizar personas o producción de productos.

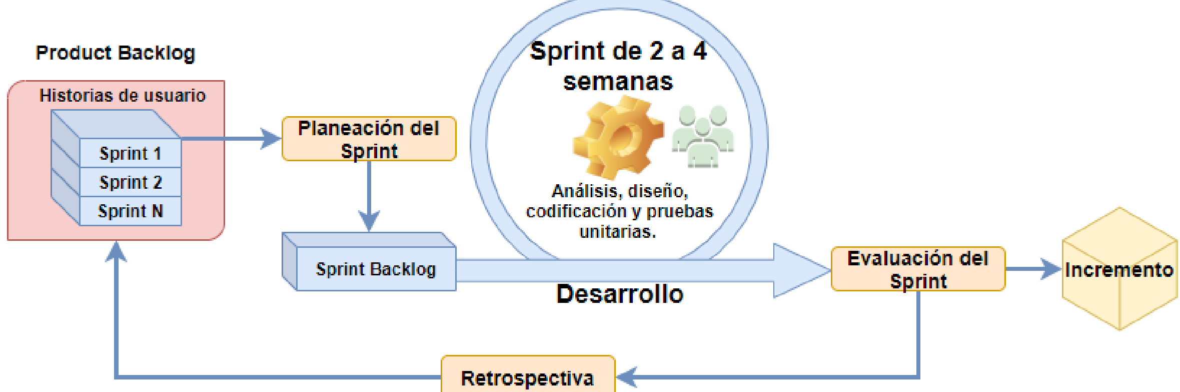


Ilustración 2.1 Modelo de desarrollo utilizando Scrum

En la ilustración 2.1 se muestra el ciclo de desarrollo con el modelo Scrum que a continuación de describen.

Proceso Scrum

Las historias de usuario ordenadas (**Product Backlog**) son descripciones utilizando el lenguaje común para expresar los requerimientos que un cliente proporciona a un equipo de desarrollo. El conjunto de historias de usuario, son priorizadas por el cliente del software (**product owner)** quien indica que historias deben realizase primero.

El proceso comienza cuando el cliente proporciona al Scrum master las historias de usuario priorizadas, y en donde el equipo de desarrollo se reúne y hacen la planeación del Sprint dando como resultado las tareas en detalle que se deben de realizar para alcanzar el objetivo (**Sprint Backlog)** el Sprint es el corazón de Scrum es una iteración con un tiempo que va de una a cuatro semanadas. Es importante mencionar que diariamente el equipo de desarrollo se reúne para analizar las tareas realizadas y por realizar. En el tiempo que dura el Sprint se lleva a cabo ciertas tareas que en el desarrollo de software son análisis, diseño, codificación y pruebas unitarias, o cualquier otro tipo de pruebas, esto dependerá del equipo de desarrollo. Una vez que termina el Sprint se realiza una revisión del Sprint en el cual se evalúan los resultados del Sprint. Posteriormente se realiza una reunión (retrospectiva) para analizar lo que se ha hecho bien o mal durante el Sprint, no es otra cosa que ver que se aprendió de ese Sprint.

La importancia de usar la metodología ágil Scrum radica en que a diferencia de las metodologías tradicionales el product owner no tiene que esperar mucho tiempo para ver su proyecto funcionando, en Scrum el proyecto se va desarrollando por partes y se le entregan dichas porciones del software al usuario para que trabaje. Otro veneficio es que si el proyecto en algún momento se cancela se pierde solo lo invertido en lo que se ha desarrollado en ese Sprint. En la gestión del cambio se hace demasiado eficiente, debido a que cualquier cambio que se realice en algún Sprint se puede ver reflejado en el próximo Sprint o cuando mucho en un mes.

Como se mencionó anteriormente Scrum solo es una metodología que me dicta los pasos que se deben de realizar para interactuar con el cliente y el equipo de desarrollo para hacer una entrega fácil e interactiva del producto, pero nunca dice que pasos se deben de ejecutar propios del desarrollo de software, como que diagramas se deben de realizar en qué momento se debe de codificar, que paso va primero, si el diseño, el análisis o codificación, es por eso que a continuación se mencionan un conjunto de metodologías que se consideraron para el desarrollo del software, dichas metodologías son propias del desarrollo web.

2.2.2 WSDM (*Web Design Method*)

Es una metodología para aplicaciones web, hoy en día las aplicaciones deben desarrollarse en un lapso corto de tiempo siguiendo su estructura semántica del contenido y funcionalidad. Es por esto, que se considera apropiada para aplicaciones web. Sin embargo, no es recomendada para la gestión de proyectos, para lo cual se debe utilizar una metodología adicional que facilite el ciclo de vida del software (Valle, 2010).

2.2.3 SOHDM (*Scenario-Based Object-Oriented Hypermedia Design Methodology*)

Es una metodología orientada a objetos en hipermedia que desarrolla diseños en escenarios o panoramas. Además, permite capturar las necesidades del sistema proponiendo el uso de escenario. **SOHDM** parte de un diagrama donde se identifican las entidades externas capaces de comunicarse con el sistema, es una metodología muy parecida a la metodología **OOHDM** diferenciadas por la utilización de escenarios.

2.2.4 WAE (*Web Application Extension*)

La **WAE** es una extensión de **UML**, que no se enfoca en el paradigma orientado a objetos si no en los elementos web. WAE incorpora algunos conceptos como JavaScript y Form. En esta metodología se cubre el lado tanto del servidor (páginas del servidor) como el cliente (ActiveX, applet Java). Sin embargo, los conceptos orientados a objetos no están suficientemente preocupados por la extensión. Se utiliza una notación de clase en el diagrama de clase para representar una página HTML. WAE se centra en la tecnología de la página de secuencias de comandos, como ASP y JSP (Ríos, 2018).

2.2.5 IWeb (Ingeniería Web)

Sirve para modelar aplicaciones web, y presta una especial atención a la sistematización y personalización. Provee de perfiles **UML**, metamodelos, un proceso de desarrollo dirigido por modelos, y herramientas de soporte para el diseño sistemático de aplicaciones web con herramientas como ***ArgoUWE*** y ***MagicUWE***. Utiliza notación basada en UML 2.0 de OMG para aplicaciones web en general y para aplicaciones adaptativas en particular.

La metodología consta de seis modelos:

- Modelo de ***casos de uso,*** para capturar los requisitos del sistema.
- Modelo ***conceptual,*** para el contenido el cual es el modelo del dominio.
- Modelo de ***usuario***, modelo de navegación que incluye modelos estáticos y dinámicos.
- Modelo de ***estructura de presentación***, modelo de flujo de presentación.
- Modelo ***abstracto de interfaz de usuario*** y modelo de ciclo de vida del objeto.
- Modelo de ***adaptación***.

2.2.6 OOHDM (*Object Oriented Hypermedia Design Methodology*)

La metodología ***OOHDM*** cuyas siglas en español son Método de Diseño e Hipermedia Orientado a Objetos. Este método toma como punto de partida el modelo de clases obtenido durante la primera fase del desarrollo de software denominado modelo conceptual, además permite modelar aplicaciones de grandes tamaños o con grandes volúmenes de información y pueden ser usados en diversos tipos de aplicaciones navegables, sitios web, sistemas de información o presentaciones multimedia.

OOHDM es una metodología orientada a objetos que propone un proceso de desarrollo de cinco fases donde se combinan notaciones graficas UML con otras propias de la metodología. Cuando internet no era accesible para todas las personas OOHDM solo se utilizaba para aplicaciones hipermedia, pero gracias al auge de internet en la actualidad se adoptó dicha metodología para el desarrollo de aplicaciones hipermedias orientadas a la web, como por ejemplo sitios de comercio electrónico, motores de búsqueda y entretenimiento.

Esta metodología permite desarrollar aplicaciones Web a partir de la utilización de modelos especializados como: ***conceptual***, ***navegación*** e ***interfaz de usuario*** teniendo como objetivo simplificar y hacer más eficaz el diseño de aplicaciones.

Las fases de esta metodología según los autores (Molina Ríos, 2018) y (Zea Ordóñez, 2018) son cinco, como se puede apreciar en la ilustración 2.2.

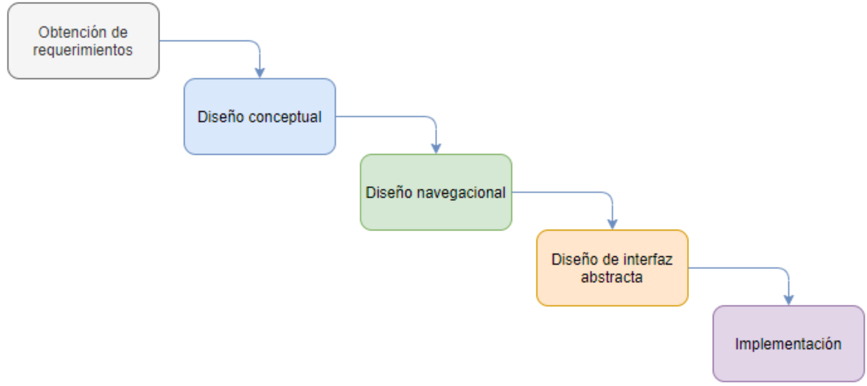


Ilustración 2.2 Fases de la metodología OOHDM

Algunos autores mencionan que son cuatro fases, esto debido a que no incluyen la fase de obtención de requerimientos pues dan por hecho que es una fase que se debe de realizar y la incluyen en la fase de diseño conceptual.

A continuación, se describirán las cinco etapas de la metodología **OOHDM**.

- ***Obtención de requerimientos***: se plantea la obtención de requerimientos de manera cuidadosa, entonces es muy importante conocer los actores y tareas que se deben modelar en los casos de uso.
- ***Diseño conceptual***: se representa el modelo conceptual a través del modelamiento de diagramas de clases, relaciones y subsistemas, enfocándose en el dominio semántico dejando de lado a los actores y tareas.
- ***Diseño navegacional***: representa los diferentes caminos que puede ejecutar la aplicación dependiendo del tipo de usuario. Es decir, brinda un contexto navegacional capaz de realizar acciones a través de enlaces, vínculos o índices que están relacionados dentro de la aplicación web dependiendo del perfil de usuario para mostrar sus vistas correspondientes.

- ***Diseño de interfaz abstracta***: es ejecutada después del diseño navegacional, donde es necesario especificar las interfaces de usuario que se visualizaran en la aplicación

web. Dentro de este modelo se pueden identificar dos subtareas tales como el diseño estructural y el diseño de comportamiento.

- ***Implementación***: es implementar la aplicación web independientemente de la plataforma que será utilizada. Esta fase también es conocida como puesta en marcha ya que es a partir de aquí en donde los usuarios empiezan a utilizar y sacar provecho al sistema elaborado, a través de un navegador web y conexión a internet.

Es importante mencionar que el desarrollador y diseñador son los encargados de la parte técnica del sistema y su apariencia final, mientras que el cliente verifica que funcione correctamente como lo ha solicitado en la primera fase o etapa (***Requerimientos***).

Tabla 2.1 Productos y formalismos de la metodología OOHDM.

ETAPAS	PRODUCTOS	FORMALISMOS
Obtención de requerimientos	Casos de uso (actores, escenarios)	Plantillas del formato del documento, Diagramas de Interacción de Usuario (UIDs).
Diseño conceptual	Clases, subsistemas, relaciones, atributos.	Modelos Orientados a Objetos
Diseño navegacional	Nodos, enlaces, estructuras de acceso, contextos, navegacionales, transformaciones de navegación.	Vistas Orientadas a Objetos, Cartas de navegación orientadas a objetos, Clases de Contexto.
Diseño de interfaz Abstracta	Objetos de la interfaz abstracta respuestas a eventos externos, transformaciones de la interfaz.	Vistas abstractas de Datos (ADV), Diagramas de Configuración, Cartas de navegación de los ADVs.
Implementación	Aplicación en funcionamiento	Los soportados por el entorno.

A continuación, se muestra una serie de comparativas de cada una de las metodologías con diferentes criterios que abarcan aspectos de requisitos, desarrollo, diseño, y calidad. En la tabla 2.2, se muestra una comparación de los requisitos que contempla cada metodología estudiada.

Tabla 2.2 Comparación de requisitos en el entorno web contemplados en las (Ríos, 2018).

REQUERIMIENTOS	METODOLOGÍAS					
	WSDM	SOHDM	OODHM	UWE	WAE	IWEB
Datos	X	X	X	X	X	X
Interfaz de usuario		X	X	X	X	X
Navegacionales			X	X	X	X
Personalización	X		X			
Transaccionales		X		X		

No funcionales	X	X	X	X	X	X

Analizando la tabla 2.2, considerando que las metodologías se encuentran en orden cronológico, se puede evidenciar que en un principio las metodologías solo se centraban en los datos y la interfaz que se le proporcionaba al usuario final, mientras que en las más actuales se resalta la importancia de tratar los requisitos de personalización, navegación, transaccionales y no funcionales. Es importante mencionar que la metodología OOHDM y UWE son las que contemplan la mayoría de los requisitos en el entorno Web.

Como se mencionó al principio del capítulo para desarrollar un software no necesariamente se debe adoptar una metodología al cien por ciento de principio a fin. De las metodologías mencionas en las secciones anteriores de este capítulo se hará uso de Scrum principalmente como base para organizar las entregas del producto, así también es importante recalcar que se hará uso de las fases obtención de requerimientos, diseño conceptual, diseño navegacional e implementación de la metodología OOHDM cada vez que se trabaje con una porción del producto que se deba entregar en el lapso acordado con el cliente (ver ilustración 2.3). El desarrollo será apoyado como ya se mencionó al principio del capítulo por diagramas UML diseñados en las fases de diseño conceptual OOHDM, así como el diseño navegación de para proporcionar una visión más general de las opciones con las que contará la aplicación. En la siguiente sección se tratarán a mayor detalle los diagramas que serán utilizados en el desarrollo del proyecto.

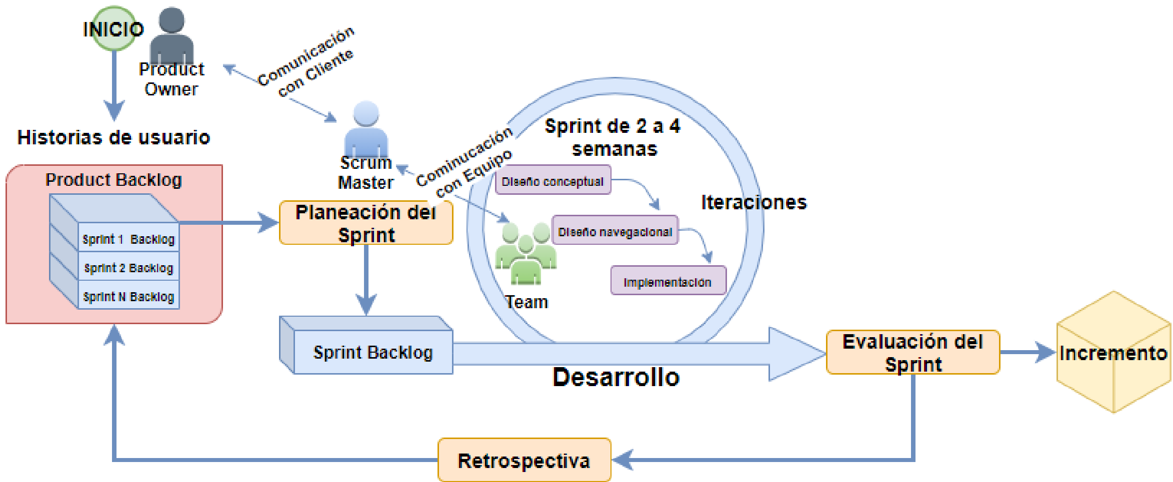


Ilustración 2.3 Metodología adaptada para el desarrollo del sistema.

En la ilustración 2.3 se puede observar la metodología adaptada en la cual se usa Scrum como base, combinada con **OOHDM**. El flujo de la metodología es el siguiente, inicia con una junta entre el **product owner** y el **Scrum mater** en donde el product owner entrega las **historias de usuario** priorizadas, posteriormente el Scrum Master y el **Team** se reúnen con todos los involucrados y planean el Sprint en donde se decide que requerimientos van a realizarse primero y el equipo de desarrollo indica que tiempo va a dedicar a cada Sprint, esta decisión la tomaran en base a su experiencia, una vez que se acuerda ese punto conforman un **Sprint Backlog** de donde el equipo ya independientemente se ponen de acuerdo que elemento del equipo va a realizar ciertas tareas, ya que se ponen de acuerdo el Scrum Master indica el inicio del primer Sprint y el equipo comienza a trabajar, el primer Sprint se entregará dependiendo del tiempo que le indicaron al product owner comúnmente son de 3 a 4 semanas, en cada iteración se tomará de la metodología **OOHDM** tres fases que son

diseño conceptual, diseño navegacional e implementación, se debe observar que la metodología web elegida **OOHDM** se realiza cada vez que se realiza un SPRINT es decir, el proceso es iterativo y tiene sentido debido a que cada que se ejecuta un Sprint, se está desarrollando software es decir, se diseñan modelos **UML**, diagramas navegacionales, se codifica, se implementa y se prueba el software. Posteriormente ya que realiza todas las fases propias del desarrollo del software se realiza una evaluación del Sprint en donde se revisa con el product owner si el Sprint cumplió con el objetivo propuesto durante la planeación de dicho Sprint, si el Sprint se cumple se entrega un incremento del software y el equipo realiza una junta de retrospectiva en donde platican sobre las experiencias adquiridas durante el **Sprint** que terminó.

En la siguiente sección se tratan los tipos de diagramas UML que serán implementados en la fase de diseño conceptual y diseño navegacional explicados anteriormente en esta sección como apoyo para el desarrollo del software.

2.2 Lenguaje Unificado de Modelado

El modelado de sistemas se utiliza para visualizar, especificar, construir y documentar los artefactos de un sistema de software. Tal como los arquitectos de edificios crean planos para que los use un grupo de personas para construir una casa, los arquitectos de software crean diagramas de UML para ayudar a los desarrolladores de software a construir el software (Sommerville, 2011).

Grady Booch, Jim Rumbaugh e Ivar Jacobson desarrollaron el UML a mediados de los años noventa con ayuda de la realimentación de la comunidad de desarrollo de software. El UML es una fusión de algunas notaciones de modelado que competían entre sí y que se usaban en la industria del software en la que se desarrolló y que aun muchas de ellas siguen vigentes. A pesar de que ha pasado por diferentes versiones hasta la versión más actual siendo esta la dos, la cual proporciona trece diferentes diagramas para su uso en modelado de software. En esta sección solo se describirán a mayo detalle los diagramas que se utilizarán para el diseño del proyecto de este trabajo de tesis, los cuales son: diagrama de clases, casos de uso y estados.

- Diagramas de ***caso de uso*** exponen las interacciones entre un sistema y su entorno.
- Diagramas de ***clase*** revelan las clases de objeto en el sistema y las asociaciones entre estas clases.
- Diagramas de ***estado*** los cuales explican cómo reacciona el sistema frente a eventos internos y externos.

A continuación, se fundamentan cada uno de los diagramas mencionados anteriormente y que serán utilizados para el diseño de la aplicación propuesta en el objetivo de la tesis, es importante mencionar que en el orden que se muestran es como serán diseños.

2.2.1 Casos de uso

El modelado de casos de uso se utiliza ampliamente para apoyar la adquisición de requerimientos. Un caso de uso ayuda a determinar la funcionalidad y características del

software desde la perspectiva del usuario, el cual describe la manera en que un usuario interactúa con el sistema, definiendo los pasos requeridos para lograr una meta específica.

En un diagrama de casos de uso estos se muestran como óvalos. Los actores se conectan mediante líneas continuas a los casos de uso. Donde los detalles de los casos de uso se incluyen en el diagrama, y en vez de ello, necesita almacenarse por separado. También es importante mencionar que se pueden colocar dentro en un rectángulo, pero los actores no. Este rectángulo es un recordatorio visual de las fronteras del sistema y de que los actores están afuera del sistema.

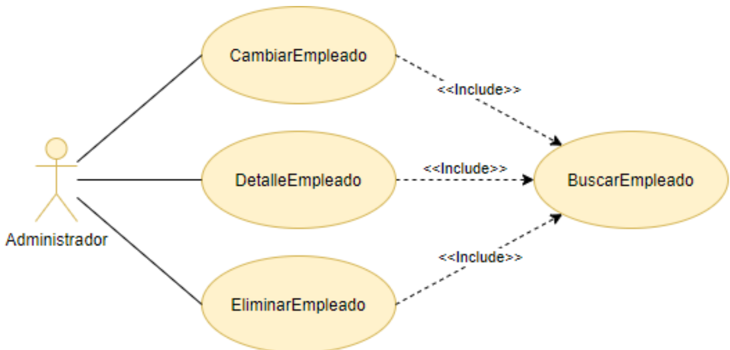


Ilustración 2.4 Casos de uso Administración de usuarios (Neustadt, 2005).

Como se observa en la ilustración 2.4, se presenta un caso de uso de un sistema de administración de personal; en el que un administrador podrá realizar ciertas operaciones en el sistema final, como buscar empleados, cambiar datos de un empleado, eliminar empleados y verDetalleEmpleado.

Observe que las líneas que van del actor en este caso un administrador son líneas continuas sin punta de flecha, esto es debido a que la asociación entre un actor y un caso de uso es un tipo de línea sin interrupción, se recomienda que los casos de uso siempre sean nombrados con acciones.

Todos los casos de uso deben ser descritos, aunque no es un estándar UML para la especificación de casos de uso. Sin embargo, en la ilustración 2.5 se muestra una plantilla que se puede utilizar, hay plantillas más complejas, pero esto dependerá de la complejidad del sistema a desarrollar.

- Nombre del caso de uso. En esta sección de la plantilla se le da un nombre al caso de uso.
- Identificador del caso de uso. En esta sección se enumera el caso de uso, como una manera de identificación única de un caso de uso.
- Descripción. En esta sección brevemente se describe el caso de uso.
- Actores involucrados. Se escriben los actores involucrados en el caso de uso.
- Precondiciones del sistema. En esta sección se listan las condiciones que debe cumplir antes de iniciar el flujo normal del caso de uso.
- Flujo normal del caso de uso. En esta sección se escribe el flujo ideal en que se debe ejecutar la acción que se describe.

- Condiciones cuando finaliza. Acción que el sistema debe cumplir una vez que finalizo el caso de uso.
- Flujo alterativo. En esta parte se deben escribir las acciones que debe ejecutar el sistema en caso de que el flujo normal del caso de uso no se ejecute como debería de ser.

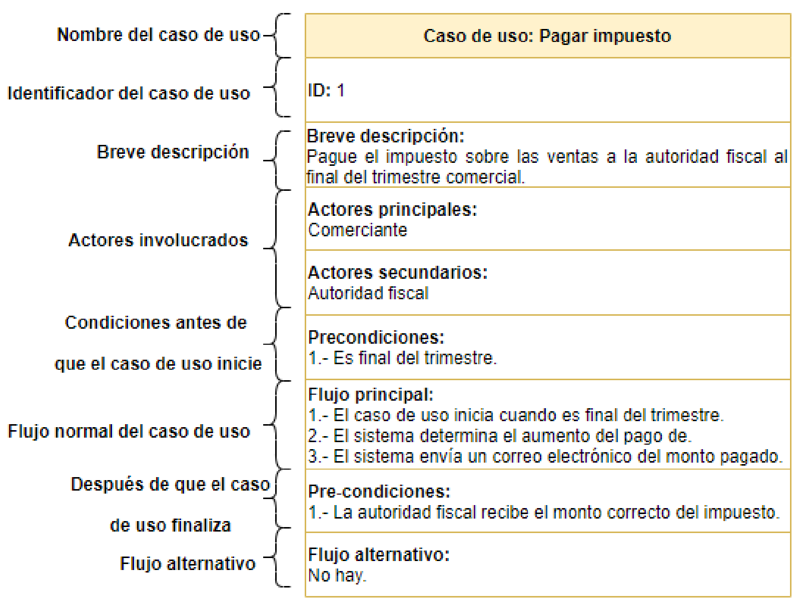


Ilustración 2.5 Plantilla para describir un caso de uso (Neustadt, 2005).

Para diseñar casos de uso se deben tener conocimiento de la simbología que se utiliza para un correcto diseño.

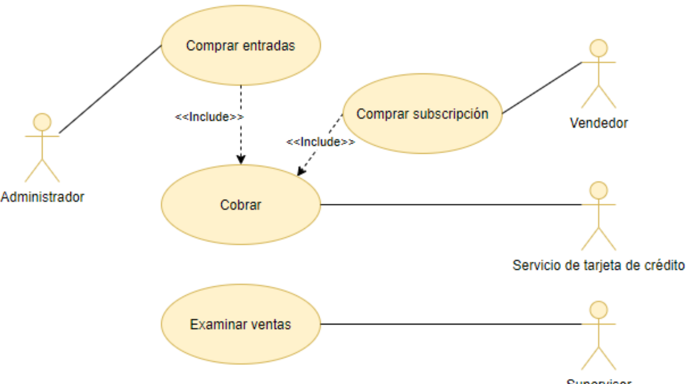


Ilustración 2.6 Casos de uso con simbología más utilizada en el diseño de casos de uso (James Rumbaugh, 2000).

En la ilustración 2.6 se puede observar un diagrama de casos de uso en el cual se hace uso de la simbología más común en el diseño de diagramas.

Tabla 2.3 Simbología de casos de uso (Elaboración propia).

Relación	Símbolo	Significado
Comunicación	______________________	Para conectar un actor con un caso de uso se utiliza una línea sin puntas de flecha.

Incluye	<<Include>>	Un caso de uso contiene un comportamiento común para más de un caso de uso. La flecha apunta al caso de uso común.
Extiende	<<extend>>	Un caso de uso distinto maneja las excepciones del caso de uso básico. La flecha apunta del caso de uso extendido al básico.
Generaliza		Una “cosa” de UML es más general que otra “cosa”. La flecha apunta a la “cosa” general.

La simbología mostrada en la tabla 2.4 es la más utilizada en el diseño de casos de uso, aunque parece básica, a partir de ella se pueden diseñar sistemas complejos.

2.2.2 Diagrama de clase

El modelado basado en clases representa los objetos que manipulará el sistema, las operaciones, también llamadas métodos o servicios que se aplicarán a los objetos para efectuar la manipulación, las relaciones entre los objetos y las colaboraciones que tienen lugar entre las clases definidas.

Para modelar clases, incluidos sus atributos, operaciones, relaciones y asociaciones con otras clases, el UML proporciona un diagrama de clase, que aporta una visión estática o de estructura de un sistema, sin mostrar la naturaleza dinámica de las comunicaciones entre los objetos de las clases.

Los elementos principales de un diagrama de clase son cajas, que son los elementos utilizados para representar clases e interfaces. Cada caja se divide en partes horizontales. La parte superior contiene el nombre de la clase. La sección media menciona sus atributos. Un atributo es algo que un objeto de dicha clase conoce o puede proporcionar todo el tiempo. Por lo general, los atributos se implementan como campos de la clase. Podrían ser valores que la clase puede calcular a partir de sus variables o valores instancia y que puede obtener de otros objetos de los cuales está compuesto. La tercera sección del diagrama de clase contiene las operaciones o comportamientos de la clase. Una operación es lo que pueden hacer los objetos de la clase. Comúnmente se implementa como un método de la clase.

En la ilustración 2.7, se puede observar una clase **Caballo** que modela caballos de pura sangre. Muestra tres atributos: madre, padre y año de nacimiento. El diagrama también muestra tres operaciones: obtener edad actual, obtener padre y obtener madre.

Cada atributo puede tener un nombre, un tipo y un nivel de visibilidad. El tipo y la visibilidad son opcionales. El tipo de dato que es almacenado sigue al nombre y se separa de

él mediante dos puntos. El nivel de seguridad de los atributos de la clase se indica mediante un signo menos, un signo de gato o un signo más, lo cual indica respectivamente, la visibilidad privada, protegida o pública. En la ilustración 2.7, todos los atributos tienen visibilidad privada, como se indica mediante el signo menos que los antecede. También es posible especificar que un atributo es estático o de clase, subrayándolo. Cada operación puede desplegarse con un nivel de visibilidad, parámetros con nombres, y un tipo de retorno.

Caballo

- CumpleAnio: Int
- Madre: Caballo
- Padre: Caballo

+ getEdad(): Int
+ getMadre(): Caballo
+ getPadre(): Caballo

Ilustración 2.7 Clase caballo

Una clase o un método abstractos se indica con el uso de cursivas en el nombre del diagrama de clase. Una interfaz se indica con la frase "***<<interface>>***" el cual se llama estereotipo, como se observa en la interfaz ***Propietario*** en la ilustración 2.8. Una interfaz también puede representarse gráficamente mediante un círculo hueco.

Los diagramas de clase también pueden mostrar relaciones entre clases. Una clase que sea una subclase de otra clase se conecta con ella mediante una flecha con una línea sólida y con una punta triangular hueca. La flecha apunta de la subclase a la superclase. En UML, tal relación se llama generalización. Como se observa en la ilustración 2.8, la clase ***PuraSangre*** y ***Cuarto***, se muestran como subclases de la clase abstracta ***Caballo***. Una flecha con una línea punteada indica implementación de una interfaz. En UML tal relación se llama realización, como se muestra en la ilustración 2.8, la clase ***Caballo*** implementa o realiza la interfaz ***Propietario***.

Una ***asociación*** entre dos clases significa que existe una relación estructural entre ellas. Las asociaciones se representan mediante líneas sólidas. Una asociación tiene muchas partes opcionales. Puede etiquetarse, así como cada una de sus terminaciones, para indicar el papel de cada clase en la asociación.

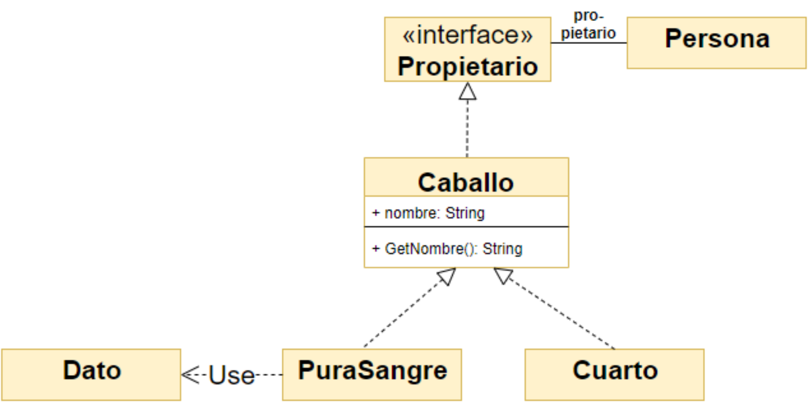


Ilustración 2.8 Clases relacionadas

Otra de las características importantes en una clase es la multiplicidad de un extremo de una asociación significa el número de objetos de dicha clase que se asocia con la otra clase. Una multiplicidad se especifica mediante un entero no negativo o mediante un rango de enteros, como se muestra en la ilustración 2.9. Una multiplicidad especificada por un asterisco indicando muchos, así como 0 o 1 objetos en dicho extremo de la asociación.

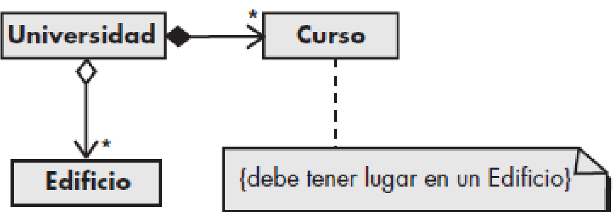


Ilustración 2.9 Multiplicidad entre clases.

En la ilustración 2.9 se observa la clase Universidad la cual se puede interpretar que una universidad puede tener muchos cursos y un curso es impartido en la universidad, también se puede interpretar de la ilustración que una universidad tiene muchos edificios pero que un edificio pertenece a una universidad. La multiplicidad es muy útil debido a que da una perspectiva de cómo debe ser la relación entre dichas clases y como estas deben ser implementadas utilizando lenguajes de programación, es por eso que este diagrama es muy utilizado para modelar software debido a que te indica como y cuantos objetos deben de ser creados y como se relacionan entre sí.

2.2.3 Diagrama de estado

El modelado dirigido por un evento muestra cómo responde un sistema a eventos externos e internos. Se basa en la suposición de que un sistema tiene un número finito de estados y que los eventos pueden causar una transición de un estado a otro. El UML soporta modelado basado en eventos usando diagramas de estado, que se fundamentaron en gráficos de estado (Harel, 1988). Los diagramas de estado muestran eventos del sistema que causan transiciones de un estado a otro.

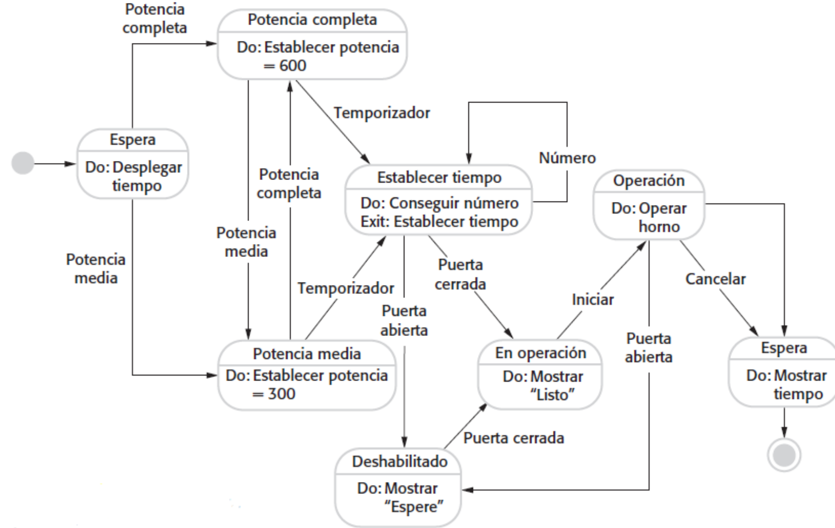


Ilustración 2.10 Diagrama de estado de un horno de microondas (Sommerville, 2011).

El diagrama de la ilustración 2.10, representa los estados de un horno de microondas de una manera sencilla, el cual tiene un interruptor para seleccionar potencia completa o media, un teclado numérico para ingresar el tiempo de cocción, un botón de iniciar-detener y una pantalla alfanumérica.

Se supone que la secuencia de acciones al usar el horno de microondas es:

- Seleccionar el nivel de potencia (ya sea media o completa)
- Ingresar el tiempo de cocción con el teclado numérico.
- Presionar "Iniciar", y la comida se cocina durante el tiempo dado.

En los diagramas de estado UML, los rectángulos redondeados representan estados del sistema. Pueden incluir una breve descripción de las acciones que se tomarán en dicho estado. Las flechas etiquetadas representan estímulos que fuerzan una transición de un estado a otro. Puede indicar los estados inicial y final usando círculos rellenos, como en los diagramas de actividad.

Tabla 2.4 Estado y estímulos para el ejemplo de la ilustración 10.

Estado	Descripción
Esperar	El horno espera la entrada. La pantalla indica el tiempo actual.
Potencia media	La potencia del horno se establece en 300 watts. La pantalla muestra "Potencia media".
Potencia completa	La potencia del horno se establece en 600 watts. La pantalla muestra "Potencia completa".
Establecer tiempo	El tiempo de cocción se establece al valor de entrada del usuario. La pantalla indica el tiempo de cocción seleccionado y se actualiza conforme se establece el tiempo.
Deshabilitado	La operación del horno se deshabilita por cuestiones de seguridad. La luz interior del horno está encendida. La pantalla indica "No está listo".

Habilitado	Se habilita la operación del horno. La luz interior del horno está apagada. La pantalla muestra “Listo para cocinar”.
Operación	Horno en operación. La luz interior del horno está encendida. La pantalla muestra la cuenta descendente del temporizador. Al completar la cocción, suena el timbre durante cinco segundos. La luz del horno está encendida. La pantalla muestra “Cocción completa” mientras suena el timbre.
Estímulo	Descripción
Potencia media	El usuario oprime el botón de potencia media.
Potencia completa	El usuario oprime el botón de potencia completa.
Temporizador	El usuario oprime uno de los botones del temporizador.
Número	El usuario oprime una tecla numérica.
Puerta abierta	El interruptor de la puerta del horno no está cerrado.
Puerta cerrada	El interruptor de la puerta del horno está cerrado.
Iniciar	El usuario oprime el botón Iniciar.
Cancelar	El usuario oprime el botón Cancelar.

El diagrama de estados fue explicado brevemente en esta sección debido a que se hará uso de un diagrama en la sección de **análisis** y **diseño** de este trabajo de tesis, para entender los estados por los que pasa un documento de solicitud de mantenimiento. En la siguiente sección se tratará otro tipo de diagrama utilizado en el desarrollo web de aplicaciones.

2.2.4 Diagrama de navegación

El diseño de aplicaciones Se apoya de los diagramas UML para representar interacciones del cliente con el usuario, estructuras de datos, estados y procesos internos que a simple vista no pueden ser observados en el desarrollo de un sistema, existe una metodología llamada ***OOHDM*** que permite desarrollar específicamente aplicaciones orientados a la web, dentro de la metodología existe una fase que es el diseño de diagramas que ayudan a entender las opciones posibles por las cuales un usuario final (usuario que usa el software) podrá navegar e ingresar a diferentes secciones dentro de la aplicación dependiendo del rol con el que fue registrado en el sistema.

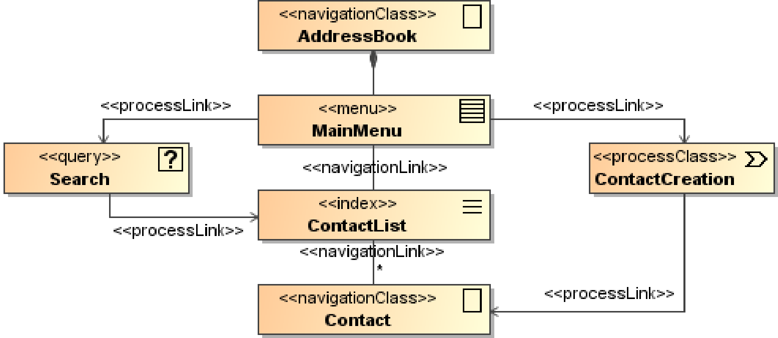


Ilustración 2.11Diagrama de navegación (München, 2010).

La ilustración 2.11 muestra un diagrama de navegación utilizado en el desarrollo de aplicaciones web (***OOHDM***).

La simbología es muy sencilla de entender, debido a que tiene como objetivo expresar con un nivel de abstracción muy alto para que los usuarios finales del software puedan entender.

clase de navegación
menú
índice
pregunta
visita guiada
clase de proceso
nodo externo

Ilustración 2.12 Nombres de los estereotipos y su simbología.

En la ilustración 2.12, se observa un conjunto de símbolos, los cuales significan lo siguiente:

- ***Clase de navegación***, este símbolo se utiliza para representar una página con contenido.
- ***Índice***, este símbolo representa un listado de elementos en una página.
- ***Visita guiada***, es una manera de indicar la ayuda para conocer la página.
- ***Nodo externo***, indica un sitio externo al actual.
- ***Menú***, simboliza una página con diferentes opciones para navegar en la página, este es muy importante debido a que es uno de los más usados en el diseño.
- ***Pregunta***, este símbolo se utiliza cuando requieres hacer una búsqueda en el sitio o en una base de datos.
- ***Clase de proceso***, este símbolo indica un proceso, es decir, cuando se realiza una consulta u operación de inserción, actualización, eliminación en una base de datos desde la interfaz gráfica de la página.

Una vez descritos los tipos de diagramas que se emplearán en este trabajo para el desarrollo del proyecto, es importante mencionar que toda aplicación web debe tener bases sólidas para cumplir con buenos atributos de calidad, además del buen diseño se debe desarrollar sobre una arquitectura escalable, es decir, que le permita crecer si en un futuro se le tienen que implementar otros módulos. A continuación, se describen dos diseños arquitectónicos que se utilizarán en el desarrollo de este trabajo.

2.3 Arquitectura del software

La arquitectura del software se interesa por entender cómo debe organizarse un sistema y cómo tiene que diseñarse la estructura global de ese sistema. En el modelo del proceso de desarrollo de software, el diseño arquitectónico es una de las primeras etapas en el proceso de diseño del software. Es el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos. La salida del proceso de diseño arquitectónico consiste en un modelo arquitectónico que describe la forma en que se organiza el sistema como un conjunto de componentes en comunicación (Sommerville, 2011).

La arquitectura de software de un sistema es el conjunto de estructuras necesarias para razonar sobre el sistema. Comprende elementos de software, relaciones entre ellos, y propiedades de ambos (Bass, Clements y Kazman, 2012).

Un concepto fundamental en el diseño de la arquitectura del software, son los patrones, que son soluciones conceptuales a problemas recurrentes a la hora de diseñar. Durante el proceso de diseño tiende a cambiar el tipo de patrones utilizados. Al diseñar desde cero un sistema, la clase de patrón con el que inicia son los estilos arquitectónicos o también las arquitecturas de referencia (Maceda, 2016).

Existen diferentes estilos arquitectónicos de software que permiten estructurar un sistema de software, en este apartado se tratarán dos arquitecturas que por la naturaleza de la aplicación a desarrollar son las que se utilizarán para implementarlas en el proceso de desarrollo.

2.3.1 Arquitectura de tres capas

Las nociones de separación e independencia son fundamentales para el diseño arquitectónico porque permiten localizar cambios o fallos dependiendo de la naturaleza de estos, de una forma rápida. El patrón MVC separa componentes de un sistema, permitiéndoles cambiar de forma independiente, es decir, si se agrega una nueva vista o funcionalidad de algún modulo que forma parte de un sistema con este patrón se hace sin modificación alguna a los datos subyacentes en el modelo. El patrón de arquitectura en capas es otra forma de lograr separación e independencia, la funcionalidad del sistema está organizada en capas separadas, y cada una se apoya sólo en las facilidades y los servicios ofrecidos por la capa inmediatamente debajo de ella.

Este enfoque en capas soporta el desarrollo incremental de sistemas, debido a que conforme se desarrolla una capa, algunos de los servicios proporcionados por esta capa deben quedar a disposición de los usuarios. La arquitectura también es cambiable y portátil. En tanto su interfaz no varíe, una capa puede sustituirse por otra equivalente. Cuando las interfaces de capa cambian o se agregan nuevas facilidades a una capa, sólo resulta afectada la capa adyacente. A medida que los sistemas en capas localizan dependencias de máquina en capas más internas, se facilita el ofrecimiento de implementaciones multiplataforma de un sistema de aplicación. Sólo las capas más internas dependientes de la máquina deben reimplantarse para considerar las facilidades de un sistema operativo o base de datos diferentes.

Como se observa en la ilustración 2.13, existe una separación de responsabilidades haciendo uso de la arquitectura en tres capas combinado con el patrón MVC. La razón por la cual se eligió esta arquitectura es por su facilidad para integrarle o quitarle módulos sin alterar la funcionalidad de la aplicación, debido a que permite crear módulos separados para almacenar las vista que son las que se presentan al usuario final, también se almacenan por separado los modelos de dominio que son donde se almacena la lógica de negocios de la aplicación permitiendo en cualquier momento exportar el módulo de dominio a otra aplicación reutilizando código para futuras implementaciones, la capa del controlador que es

la que se expone a las peticiones de los navegadores para recibir información externa.

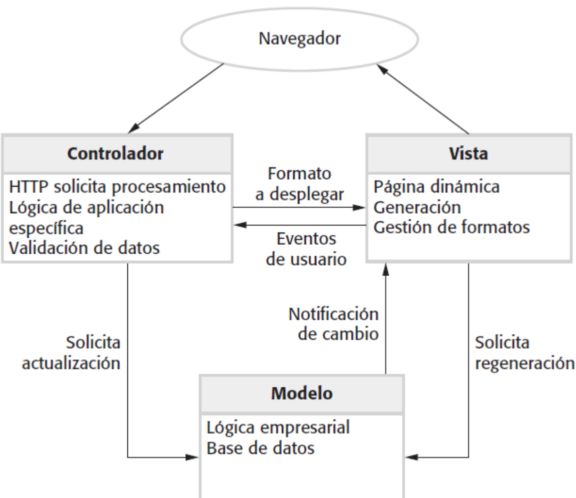


Ilustración 2.13 Arquitectura de aplicación en 3 capas con el patrón MVC (Sommerville, 2011).

Como se puede observar en la ilustración 2.13 todo comienza por una petición desde el navegador que llega a través del protocolo HTTP al controlador solicitado, una vez que el controlador recibe la petición el controlador comienza a buscar la información solicitada en la capa de modelos de datos donde se encuentra la lógica empresarial del negocio, es en esta capa donde se puede hacer una comunicación con la base de datos para insertar, extraer, actualizar o eliminar algún recurso, una vez que se realizó la operación la información es devuelta a una vista para ser mostrado a través de las vista al usuario, es importante mencionar que de esta arquitectura existen muchas variaciones y según la variación la información es devuelta a la vista al controlador o enviar la información a otro recursos externo.

2.3.2 Arquitectura de repositorio

El patrón de repositorio describe cómo comparte datos un conjunto de componentes en interacción. La mayoría de los sistemas que usan grandes cantidades de datos se organizan sobre una base de datos o un repositorio compartido. Este modelo es adecuado para aplicaciones en las que un componente genere datos y otro los consuma. Este tipo de sistema incluyen sistemas de comando y control, sistemas de información administrativa, sistemas CAD y entornos de desarrollo interactivo para software.

Organizar herramientas alrededor de un repositorio es una forma eficiente de compartir grandes cantidades de datos. No hay necesidad de transmitir explícitamente datos de un componente a otro. Sin embargo, los componentes deben operar en torno a un modelo de repositorio de datos acordado. Inevitablemente, éste es un compromiso entre las necesidades específicas de cada herramienta y sería difícil o imposible integrar nuevos componentes, si sus modelos de datos no se ajustan al esquema acordado. En la práctica, llega a ser complicado distribuir el repositorio sobre un número de máquinas. Aunque es posible distribuir un repositorio lógicamente centralizado, puede haber problemas con la redundancia e inconsistencia de los datos.

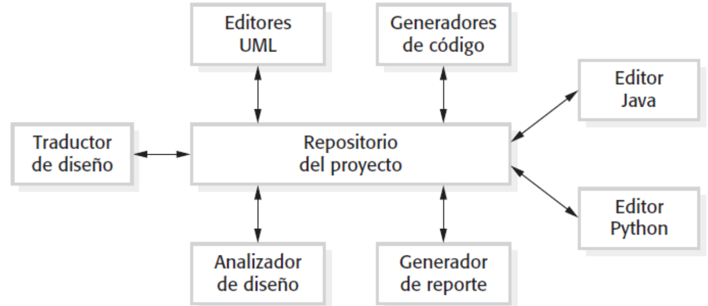


Ilustración 2.14 Arquitectura de repositorio para un IDE.

Como se puede observar en la ilustración 2.14, diferentes componentes están interactuando con un proyecto, en este caso los componentes son editores de diferentes lenguajes de programación que no tienen nada en común pero que pueden consumir información del repositorio sin ningún problema. Este tipo de arquitectura es muy utilizada en donde se tiene que compartir datos y diferentes usuarios que programan en diferentes lenguajes puedan consumirla sin ningún problema. Es ese uno de los principales motivos por los cuales se eligió esta arquitectura para el desarrollo del proyecto de tesis, porque sin importar en que lenguaje se programe el proyecto, la información se podrá consumir con una aplicación web, de dispositivo móvil o incluso de escritorio. Es relevante mencionar que a este tipo de patrones conceptuales en la práctica del desarrollo web son llamadas ***API-REST***, la cual es una aplicación ejecutándose en un servidor compartiendo información pero con la condición de que si el api es privada se le pedirá cumplir con un contrato, es decir, se debe estar registrado en una base de datos que es consultada cada vez que se quiere obtener información y la información será entregada dependiendo de si se tienen los permisos necesarios para consumir dicha información.

Después de haber documentado las arquitecturas de software, en la siguiente sección se describirán las herramientas de desarrollo de software que se utilizarán para la construcción de la aplicación propuesta en el objetivo de la tesis.

2.4 Herramientas de desarrollo del software

Las aplicaciones web son desarrolladas con diversas tecnologías que pueden ser divididas en dos grandes grupos, **backend** y **frontend**, dichas tecnologías tienen un contexto de ejecución, una es ejecutada del lado del cliente y la otra del lado del servidor. A continuación, se describirán los framework que se utilizarán para el desarrollo del proyecto.

2.4.1 ASP.NET Core 2

ASP.NET Core MVC es un marco de desarrollo de aplicaciones web de Microsoft que combina la efectividad y el orden de la arquitectura de modelo-vista-controlador (MVC), ideas y técnicas de desarrollo ágil, y las mejores partes de la plataforma .NET. Se introdujo en 2002, en un momento en que Microsoft deseaba proteger una posición dominante en el desarrollo tradicional de aplicaciones de escritorio y veía a Internet como una amenaza.

La versión ASP.NET Core MVC 2 se centra en la consolidación, trabajando a través de algunos de los cambios de herramientas y plataformas que se introdujeron en versiones anteriores. ASP.NET Core MVC 2 requiere .NET Core 2, que tiene una superficie API muy expandida y ahora es compatible con distribuciones de Linux. Los cambios útiles incluyen un nuevo sistema de meta paquete, que simplifica la administración de paquetes **NuGet**, un nuevo sistema de configuración para ASP.NET Core y soporte para Entity Framework Core.

Arquitectura MVC

ASP.NET Core MVC sigue un patrón llamado **model-view-controller**, que guía la forma de una aplicación web ASP.NET y las interacciones entre los componentes que contiene.

- La interacción del usuario con una aplicación que se adhiere al patrón MVC sigue un ciclo natural: el usuario toma una acción y, en respuesta, la aplicación cambia su modelo de datos y ofrece una vista actualizada al usuario. Entonces el ciclo se repite. Este es un ajuste conveniente para aplicaciones web entregadas como una serie de solicitudes y respuestas HTTP.

- Las aplicaciones web requieren combinar varias tecnologías (bases de datos, HTML y código ejecutable, generalmente divididas en un conjunto de niveles o capas. Los patrones que surgen de estas combinaciones se asignan naturalmente a los conceptos en el patrón MVC. ASP.NET Core MVC implementa el patrón MVC y, al hacerlo, proporciona una separación de preocupaciones mucho mejor en comparación con los formularios web.

Es importante mencionar que a diferencia de las plataformas de desarrollo web de Microsoft anteriores, puede descargar el código fuente de ASP.NET Core MVC e incluso modificar y compilar su propia versión (Freeman, Pro ASP.NET Core MVC 2, 2017).

Se decide utilizar ASP.NET Core como la tecnología para formar parte del núcleo de la aplicación a desarrollar, su curva de aprendizaje que permite aprender nuevas características del lenguaje en poco tiempo, la documentación tan variada que existe en los medios de información, así como la disponibilidad de información en los foros y portales en la web. Otra de las razones es que es un lenguaje de distribución libre que puede ser utilizado para el desarrollo de aplicaciones sin costo alguno.

2.4.2 Lenguaje C#

C# es un lenguaje orientado a objetos seguro y elegante que permite a los desarrolladores construir un amplio rango de aplicaciones seguras y robustas que se ejecutan sobre .NET Framework (que incluye entre otras cosas la biblioteca básica de .NET y el compilador C#) junto con otros componentes de desarrollo, como ASP.NET (formularios web y servicios web) y ADO.NET, forman un paquete de desarrollo denominado Microsoft Visual Studio que podemos utilizar para crear aplicaciones Windows tradicionales (aplicaciones de escritorio que muestren una interfaz gráfica al usuario) y aplicaciones para la Web. Para ello,

este paquete proporciona un editor de código avanzado, diseñadores de interfaces de usuario apropiados, depurador integrado y muchas otras utilidades para facilitar un desarrollo rápido de aplicaciones (Sierra, 2013).

2.4.3 Entity Framework Core

Entity Framework Core, también conocido como EFCore, es un paquete de mapeo relacional de objetos (ORM) producido por Microsoft que permite que las aplicaciones .NET Core almacenen datos en bases de datos relacionales. Entity Framework permite a los desarrolladores crear aplicaciones que acceden a bases de datos elevando el nivel de abstracción, del nivel lógico relacional al nivel conceptual. Con este nivel de abstracción superior, Entity Framework admite código que es independiente de cualquier motor de almacenamiento de datos o esquema relacional determinados. Pues bien, utilizando LINQ, concretamente el proveedor LINQ, es posible consultar las entidades que definen el modelo conceptual de Entity Framework. Durante el diseño de la aplicación, se asigna el modelo de datos relacional de una base de datos a un modelo de objetos expresado en el lenguaje de programación, para después realizar las consultas sobre el modelo de objetos. Estas consultas son convertidas por Entity Framework a SQL y enviadas a la base de datos para su ejecución. Cuando la base de datos devuelva los resultados, Entity Framework los vuelve a convertir en objetos expresados en el propio lenguaje de programación utilizado (ver ilustración 2.15).

Hay una gran diferencia entre Entity Framework y LINQ. LINQ es un lenguaje de consulta integrado que se puede ejecutar sobre varias fuentes por medio de los distintos proveedores de **LINQ** desarrollados hasta la fecha. Estas fuentes y los proveedores correspondientes son:

- DataSet: LINQ to DataSet.
- XML: LINQ to XML.
- Objetos de memoria: LINQ to Objects.
- Bases de datos relacionales: LINQ to SQL y Entity Framework.

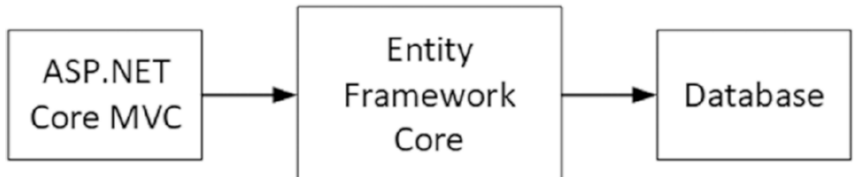


Ilustración 2.15 Entity Framework Core en contexto.

Entity Framework soporta dos enfoques de desarrollo, los cuales son:

- **Database First.** Entity Framework Core API crea la base de datos y las tablas utilizando la migración basada en las convenciones y la configuración proporcionadas en sus clases de dominio. Este enfoque es útil en el diseño dirigido por dominio, como se observa en la ilustración 2.16.

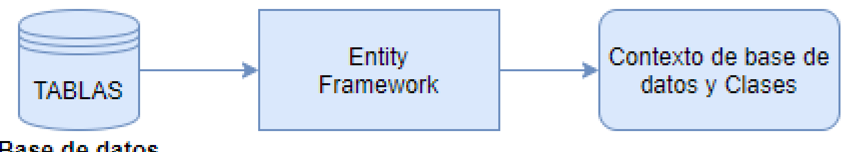


Ilustración 2.16 Enfoque databasefirst.

- **Code First.** Entity Framework Core API crea el dominio y las clases de contexto basadas en su base de datos existente usando los comandos de Entity Framework Core, observar ilustración 2.17. Esto tiene un soporte limitado en EF Core, ya que no es compatible con el diseñador visual.

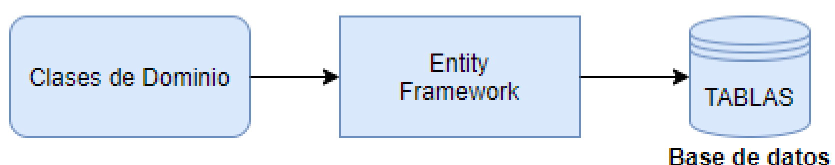


Ilustración 2.17 Enfoque codefirst.

Proveedores de bases de datos de Entity Framework Core:

Tabla 2.5 Proveedores de bases de datos y los paquetes NuGet para EF Core.

Bases de datos	Paquetes NuGet
SQL Server	Microsoft.EntityFrameworkCore.SqlServer
MySQL	MySql.Data.EntityFrameworkCore
PostgreSQL	Npgsql.EntityFrameworkCore.PostgreSQL
SQLite	Microsoft.EntityFrameworkCore.SQLite
SQL Compact	EntityFrameworkCore.SqlServerCompact40
In-Memory	Microsoft.EntityFrameworkCore.InMemory

2.4.4 Linq (*Language Integrated Query*)

LINQ (Language Integrated Query) fue inicialmente soportado por .NET Framework 3.0 con la finalidad de ofrecer la posibilidad de expresar las operaciones de consulta en el propio lenguaje C# y no como literales de cadena pertenecientes a otro lenguaje incrustados en el código de aplicación o como procedimientos almacenados. Finalmente, es con la versión 3.5 de .NET Framework, espacios de nombres System.Linq y System.Data.Linq, cuando LINQ queda totalmente integrado en este marco de trabajo, junto con otras bibliotecas como WPF, WCF, WF o ASP.NET AJAX, haciendo realidad la implementación de aplicaciones que contengan única y exclusivamente código .NET.

LINQ es una combinación de extensiones al lenguaje y bibliotecas de código administrado que permite expresar de manera uniforme consultas sobre colecciones de datos de diversa procedencia utilizando recursos del propio lenguaje de programación; por ejemplo, sobre objetos en memoria, sobre bases de datos relacionales o sobre documentos XML, entre otros. Los elementos básicos sobre los que se construyeron estas nuevas extensiones son los siguientes:

- Declaración implícita de variables locales.
- Matrices de tipos definidos de forma implícita.
- Tipos anónimos.
- Propiedades auto implementadas.
- Iniciadores de objetos y colecciones.
- Métodos extensores.
- Expresiones lambda.
- Operadores de consulta.
- Árboles de expresiones lambda.

La característica que tiene Linq llamada de tipos anónimos es lo que permite implementar patrones de diseño, específicamente el patrón repositorio, el cual con una sola clase de tipo anónima se pueden crear objetos comunes a todo tipo de clase que la herede. Es por este motivo que se eligió el marco de desarrollo de.NET Core 2 para desarrollar el proyecto que ocupa esta tesis.

2.4.5 Patrón de Repositorio (*Pattern Repository*)

Los patrones de repositorio y unidad de trabajo están destinados a crear una capa de abstracción entre la capa de acceso a datos y la capa de lógica de negocios de una aplicación. La implementación de estos patrones puede ayudar a aislar su aplicación de los cambios en el almacén de datos y se puede facilitar las pruebas unitarias automatizadas o el desarrollo basado en pruebas.

Como se observa en la ilustración 2.18, una aplicación sin el uso del patrón repositorio tiene un acceso directo a la base de datos lo cual pareciera que está bien, pero en realidad la lógica de acceso a datos es muy grande, tanto que por cada clase que se implemente en la aplicación se tienen que realizar las operaciones por cada clase. Sin embargo, haciendo uso del patrón repositorio, el trabajo de crear clases se facilita debido a que se crea una clase con las operaciones comunes de cada clase y se heredan en la clase que las ocupe, de esta forma se reutiliza y se reduce código que al final de cuentas se resume en un rendimiento muy alto en las aplicaciones.

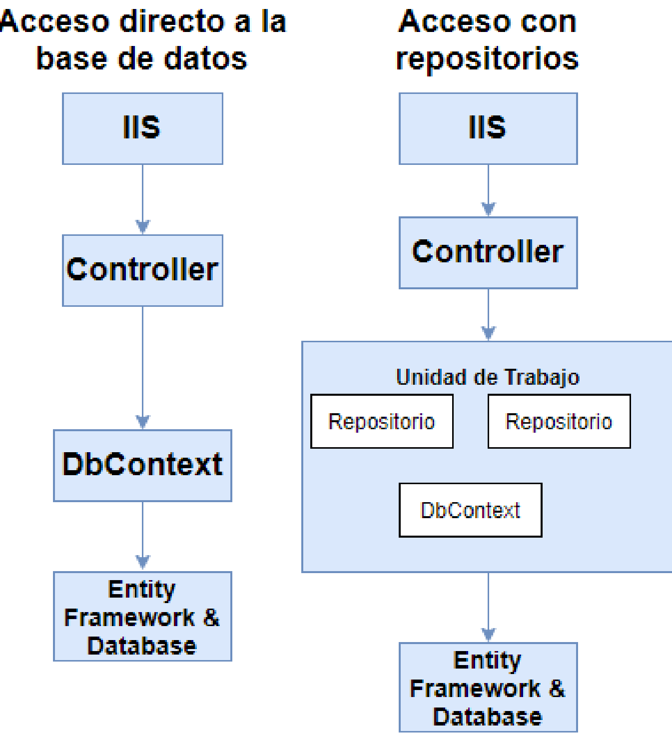


Ilustración 2.18 Patrón Repositorio (Microsoft, 2013)

Un repositorio no es más que una clase definida para una entidad, con todas las operaciones posibles de la base de datos. Por ejemplo, un repositorio para una entidad empleado tendrá las operaciones CRUD básicas y cualquier otra operación posible relacionada con la entidad empleado. El patrón de repositorio se puede implementar de dos maneras:

- Un repositorio por entidad (no genérico). Este tipo de implementación implica el uso de una clase de repositorio para cada entidad. Por ejemplo, si tiene dos entidades, Empleado y Cliente, cada entidad tendrá su propio repositorio.

- Repositorio genérico. Un repositorio genérico es el que se puede usar para todas las entidades. En otras palabras, puede usarse para Empleado o Cliente o cualquier otra entidad, heredando la misma clase.

2.4.6 Patrón Unidad de Trabajo (*Pattern Unit Of Work*)

La unidad de trabajo tiene un propósito: asegurarse que cuando se usen múltiples repositorios, compartan un solo contexto de base de datos. De esa manera, cuando se completa una unidad de trabajo, se puede llamar a un mismo método en esa instancia del contexto y asegurarse de que todos los cambios relacionados se coordinarán.

El patrón de unidad de trabajo se usa para agrupar una o más operaciones (generalmente operaciones CRUD de base de datos) en una sola transacción o unidad de trabajo para que todas las operaciones pasen o fallen como una sola.

Se puede decir que, para una acción específica del usuario, por ejemplo, reservar en un sitio web, todas las transacciones como insertar, actualizar, eliminar, se realizan en una sola transacción, en lugar de realizar múltiples transacciones en la base de datos. Esto significa que una unidad de trabajo involucra operaciones de inserción, actualización, eliminación, todo en una sola transacción para que todas las operaciones pasen o fallen como una sola, como se observa en la ilustración 2.19.

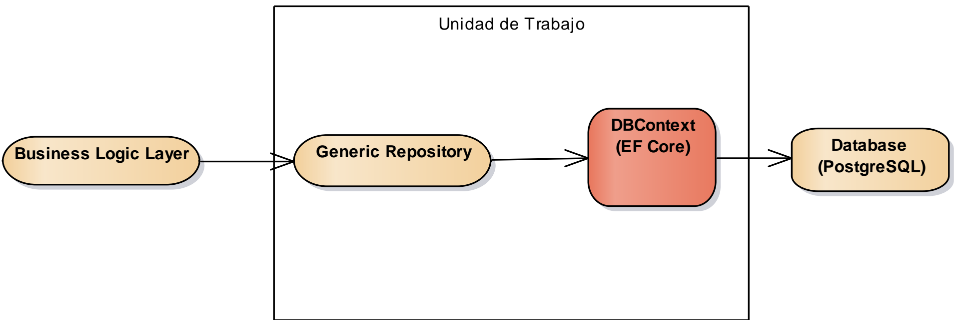


Ilustración 2.19 Unidad de Trabajo (Elaboración propia).

El uso de este patrón de diseño asegura la consistencia e integridad de los datos al realizar una operación costosa en cuanto a tiempo y proceso. Usar la unidad de trabajo garantiza que la información sea almacenada y consultada sin problemas.

2.4.7 PostgreSQL

PostgreSQL es un sistema de gestión de base de datos relacional de objetos (ORDBMS), desarrollado en el Departamento de Informática de la Universidad de California en Berkeley. PostgreSQL fue pionero en muchos conceptos que solo estuvieron disponibles en algunos sistemas de bases de datos comerciales, es un descendiente de código abierto. Admite una gran parte del estándar SQL y ofrece muchas características modernas:

- Consultas complejas
- Llaves extranjeras
- Disparadores
- Vistas actualizables
- Integridad transaccional
- Control de concurrencia en múltiples versiones

PostgreSQL puede ser extendido por el usuario de muchas maneras, por ejemplo, agregando nuevos:

- Tipos de datos
- Funciones
- Operadores
- Funciones agregadas
- Métodos de índice
- Lenguajes de procedimiento

Y debido a la licencia liberal, PostgreSQL puede ser utilizado, modificado y distribuido por cualquier persona de forma gratuita para cualquier propósito, ya sea privado, comercial o académico, y es aquí donde radica el porqué de hacer uso de PostgreSQL como almacen de datos para el sistema a desarrollar.

2.4.8 Angular 7

Angular es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo, Vista, Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript que en conjunto forma un componente, ya que angular tiene un principio y es que toda página es un componente angular (Google, 2016).

2.4.9 Bootstrap

Bootstrap es un kit de herramientas de código abierto para desarrollar con HTML, CSS y JavaScript. Permite realizar rápidamente prototipos de sus ideas o crear toda una aplicación con variables y mixinsSass, sistema de cuadrícula sensible, componentes precompilados extensos y complementos potentes creados en jQuery. Se pueden crear proyectos receptivos para dispositivos móviles en la web con la biblioteca de componentes frontend más popular del mundo (Twitter, 2011).

Para realizar funciones como la manipulación del Modelo de Objetos del Documento (DOM) o validaciones en formularios con JavaScript estas acciones conllevan en ocasiones un tiempo considerable, es por eso que Bootstrap te permite ahorrar el tiempo que consumen realizar esas tareas, este framework es sencillo y con una curva de aprendizaje muy simple, permitiendo darle estilos a todo un sistema completo y complejo.

2.4.10 Visual Studio

Visual Studio es una plataforma que ayuda a editar, depurar y crear código de aplicaciones de diferentes lenguajes. El entorno de desarrollo integrado es un programa rico en funcionalidades que se pueden usar en muchas áreas del desarrollo del software. Además del editor enriquecido con compiladores, analizadores de sintaxis, Visual Studio incluye un gestor de paquetes muy amplio con diferentes librerías que pueden ser descargadas. Es importante mencionar que está disponible para varios sistemas operativos siento uno de los principales Windows y MacOS, actualmente tiene una funcionalidad muy utilizada y es que se le integro una librería que permite hacer cambios y subir código para el control de versiones en la plataforma de GitHub, además que se pueden hacer despliegues de aplicaciones con Azure desde el IDE facilitando la puesta en producción de las aplicaciones que se desarrollan.

Se opta por hacer uso de este IDE debido a la fuerte integración que existe con el lenguaje utilizado para el desarrollo del proyecto propuesto, contando con todas las características mencionadas anteriormente.

2.4.11 AdminLTE2

Las plantillas son maquetaciones diseñadas por terceros, esto ayuda a facilitar un desarrollo web. Haciendo uso de tecnologías ya existentes es muy fácil desarrollar paneles administrativos, los cuales son los más comunes. AdminLTE2 es una plantilla que hace uso de **Bootstrap**, **HTML**, **CSS** y **JavaScript**. Comúnmente las plantillas son desarrollos genéricos, es decir, se desarrollan de tal manera que puedan implementarse con cualquier tecnología Backend.

Existen tres versiones de AdminLTE2, se decide utilizar esta versión debido a la facilidad de uso y a la fácil adaptación con el framework que se utiliza como base del desarrollo de la aplicación. La plantillad AdminLTE2 cuenta con una variedad de formularios, botones, cuadros y pantallas para ingresar datos que lo único que se tiene que hacer es organizar dichas plantillas, pues todo está prediseñado, es decir, un desarrollador de páginas web ya realizo el trabajo de maquetación web evitando así invertir más tiempo en el proyecto.

Capítulo 3 Análisis y Diseño

Como se mencionó en el capítulo anterior, la metodología planteada para el desarrollo del sistema es OOHDM o Metodología de Diseño de Hipermedia Orientada a Objetos, apoyada de Scrum como un marco de trabajo que dicta los pasos a seguir en cómo se deben de obtener los requerimientos del sistema y como se deben de hacer las entregas del producto terminado, es importante recalcar que el marco de trabajo Scrum nos permite hacer entregas del producto en periodos cortos de tiempo.

En este apartado de análisis y diseño se han seleccionado diagramas propios de la metodología propuesta OOHDM, que permiten una abstracción de alto nivel, facilitando la integración en el desarrollo del sistema propuesto.

La sección de análisis inicia con las **historias de usuarios** proporcionadas, después se explica el diagrama de modelado de negocios del sistema, el cual permitirá una visión amplia del sistema a desarrollar, remarcando los procesos en los cuales interviene el trabajo a automatizar en esta tesis, se han modelado diagramas específicos de estas actividades a partir de los cuales es posible crear modelos de casos de uso para cada proceso.

El apartado de diseño inicia con una visión general de alto nivel en el que es posible identificar a los usuarios que interactuarán con el sistema. Los diagramas que se diseñan son de casos de uso, clases, estados y navegación, siendo este último diagrama propio de la metodología OOHDM, dicho diagrama nos permite visualizar gráficamente las opciones que cada usuario puede ejecutar dependiendo del rol con que sean registrados. Al final de este capítulo se describe el diagrama de despliegue, en el que se visualizan los componentes físicos mínimos y las tecnologías utilizadas con que debe de cumplir el sistema para una correcta puesta en producción de la aplicación.

3.1 Análisis

Como se mencionó en el capítulo 2 para desarrollar un software se necesita hacer uso de una metodología que dicte los pasos a seguir en las entregas puntuales del software, en este trabajo de tesis para el desarrollo de la herramienta se eligió Scrum y la primera etapa que dicta esta metodología es la obtención de los requerimientos funcionales (**historias de usuario**), los cuales se muestran en la siguiente sección.

3.1.1 Historias de usuario

Las historias de usuarios con las que debe de cumplir la aplicación son proporcionadas por el propietario del proyecto en **Scrum** llamado **product owner**. Este conjunto de funcionalidades deberá ser priorizadas, es decir, se deben de seleccionar las que el propietario del software decida que tengan más importancia en el momento para llevarse a cabo. Lo descrito anteriormente se realiza iterativamente (**Sprints**) con todos los requerimientos hasta que ya no existan más historias de usuario por realizar. La lista de historias de usuarios tiene como nombre **product backlog**.

Product backlog

Tabla 3.1 Product Backlog proporcionado por el product owner del sistema.

ID	Requerimiento	Criterio de validación
HU-1	El usuario con el rol de **administrador** y **coordinador de servicio** pueden gestionar los usuarios del sistema.	Se pueden crear, actualizar, consultar y eliminar un usuario como mínimo.
HU-2	Únicamente el usuario con el rol de **coordinador de servicio** y **administrador** podrá agregar información al catálogo de la aplicación.	El catálogo puede ser manipulado como mínimo por el administrador del sistema.
HU-3	El usuario con el rol de **auxiliar** podrá consultar información del catálogo.	El usuario con el rol del sistema puede ver todos los registros del catálogo.
HU-4	El usuario con el rol de **coordinador de servicio** puede crear listas de verificación.	Se puede consultar la lista de verificación creada.
HU-5	Los usuarios con el rol de **coordinador de servicio**, **administrador** y **auxiliar** pueden consultar las listas de verificación creadas.	Como mínimo puede ser consultada una lista, mostrando la información de esta.
HU-6	El usuario con el rol de **coordinador de servicio** y **auxiliar** pueden registrar anomalías de los equipos e infraestructuras en la lista de verificación.	Las anomalías registradas son mostradas en una tabla en la pantalla del sistema.
HU-7	El usuario con el rol de **coordinador de servicio** puede gestionar planes de trabajo.	La información es almacenada en la base de datos y mostrada en pantalla.
HU-8	El usuario con el rol de **coordinador de servicio** o **administrador** puede programar en el plan de trabajo las fechas en que se resolverán las anomalías encontradas en la verificación de infraestructuras y equipos.	El sistema muestra un formulario en el cual se puede asignar la fecha en que se resolverá la anomalía programada.
HU-9	El usuario con el rol de administrador puede consultar los planes de trabajos creados.	Se muestra una tabla con la información del plan de trabajo creado.
HU-10	El usuario con el rol de **coordinador de servicio** puede generar un pdf con las tareas asignadas a un plan de trabajo.	Se muestra el pdf con las tareas asignadas al plan de trabajo.

Las historias de usuario mostradas anteriormente se detallan en las siguientes secciones haciendo uso de diagramas, conceptualmente cada requerimiento funcional se expresa en un caso de uso o en su defecto en un conjunto de casos de uso.

La tabla 3.1 muestra el **product backlog** priorizado donde cada requerimiento cuenta con un criterio de validación que se debe de cumplir al evaluar el Sprint según el marco de trabajo Scrum.

3.1.2 Modelado de negocios

El modelado de negocios muestra una visión general de los procesos que se llevan a cabo en el Instituto Tecnológico de San Marcos, en este diagrama se visualiza en donde la aplicación web ayudará a automatizar operaciones en el procedimiento de mantenimiento de infraestructuras y equipo.

En el diagrama que se muestra en la ilustración 3.1 se observan los procesos que realiza el departamento de **centro de cómputo**, **recursos materiales y servicios**. Solo los procesos que están en azul son donde la aplicación web automatizará las operaciones que ocurren en los departamentos mencionados.

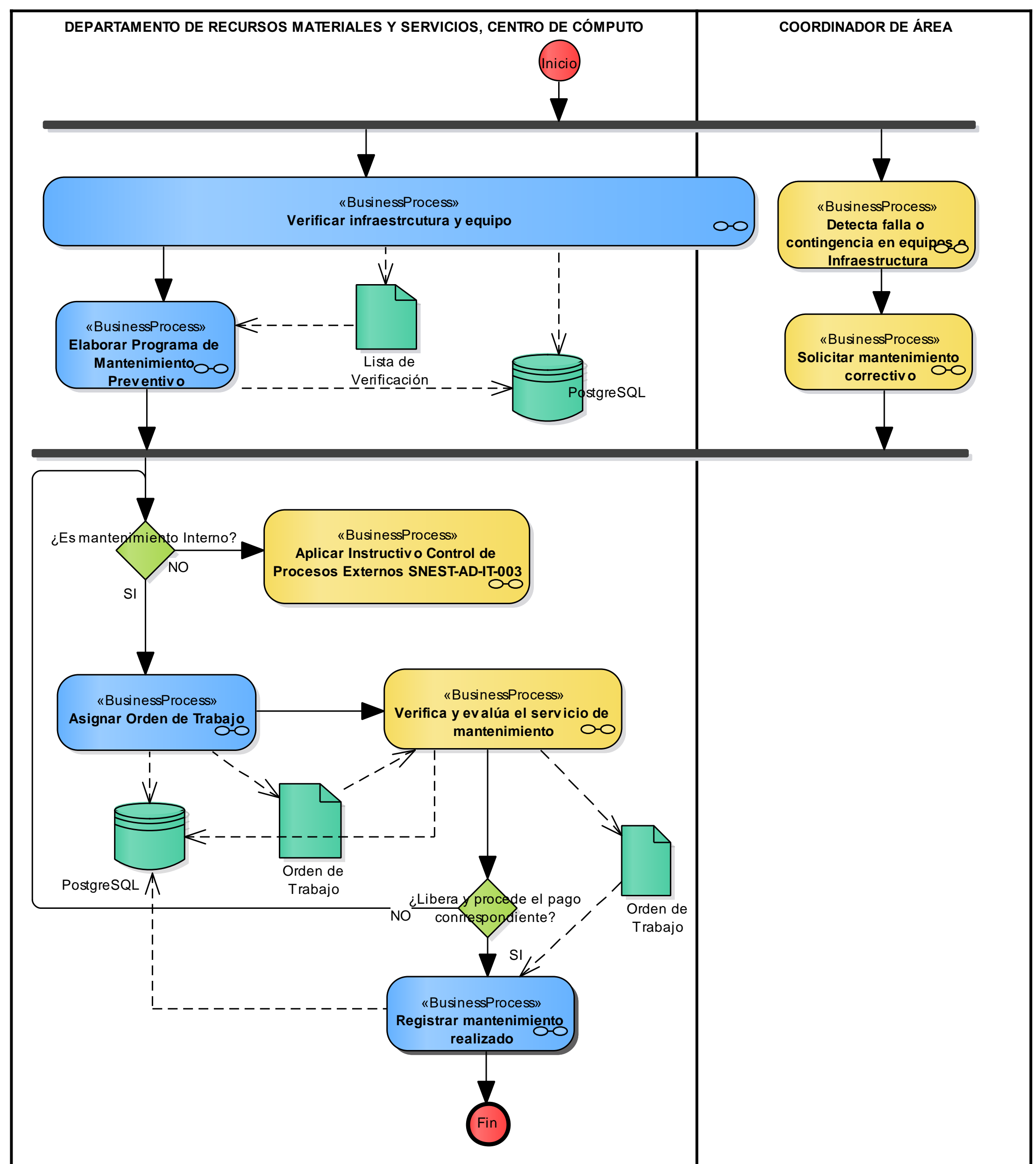


Ilustración 3.1 Modelado de negocios de procedimiento de mantenimiento del Instituto Tecnológico de San Marcos.

Los carriles hacen alusión a las áreas o departamentos involucrados en las operaciones. A continuación, se explica brevemente en que consiste cada proceso del modelado de negocio.

- *Detección de fallas o contingencias en equipos o infraestructuras*: En este proceso cada **jefe de departamento** o docente puede crear solicitudes de mantenimiento correctivo.
- *Asignar orden de trabajo*: En este proceso el jefe de departamento realiza órdenes de trabajo para cada solicitud de mantenimiento correctivo o preventivo que son solicitadas.

- *Verificación infraestructura y equipo*: Proceso en el cual se anotan las **anomalías** que son encontradas en la **infraestructura y equipos**.
- *Elaborar programa de mantenimiento preventivo*: En este proceso se **realiza** un programa de las actividades a realizar durante el semestre en curso, **a partir** del proceso de **verificación de infraestructura y equipo**, el cual consiste en asignar y describir las actividades que se realizarán en una fecha indicada por el jefe de departamento que brinda el mantenimiento.
- *Verifica y evalúa el servicio de mantenimiento*: En este proceso la persona que solicitó un mantenimiento firma la orden de trabajo que libera el trabajo que se realizó.

A partir del modelado de negocios que describen las actividades del proceso de mantenimiento de infraestructura y equipo, se obtuvo el diagrama de casos de uso en el cual se muestra la interacción de los usuarios con los procesos que se realizarán en el desarrollo del sistema, cada caso de uso tiene asociado un actor del negocio, los cuales están directamente relacionados con cada caso de uso, es decir, son las personas que interactuarán con el sistema directamente, cada caso de uso son las funciones que cada uno podrá realizar en el sistema, dependiendo del rol con el que cuente el usuario que inicia sesión en la aplicación.

3.1.3 Modelado de casos de uso

Para el modelado de casos de uso, se describe los casos de uso más relevantes del sistema, apegados estrictamente a los procesos del modelado de negocios que se explicó en la sección anterior. En la ilustración 3.2 se puede observar a tres actores de los modelos de casos de uso, **Auxiliar**, **Coordinador de Servicio** y **Administrador**.

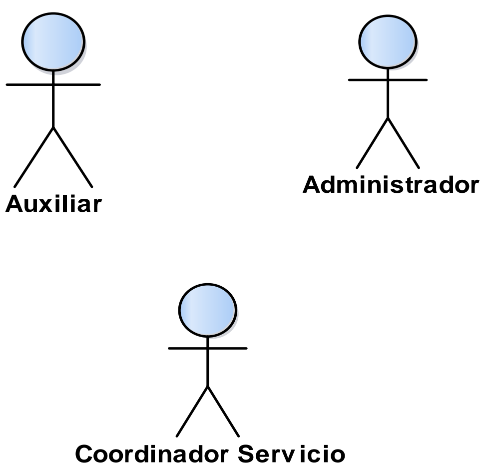


Ilustración 3.2 Actores involucrados

El **Coordinador de Servicio** es cada una de las personas que están a cargo de algún departamento que brinda servicios en el Tecnológico de San Marcos. Por ejemplo:

- Departamento de Centro de Cómputo.
- Departamento de Recursos Materiales y Servicios.

El **Administrador** es el jefe de las personas que tienen a cargo un departamento que brinda servicios, por ejemplo: Centro de Cómputo, Recursos Materiales y Servicios, es decir, es el jefe inmediato de los departamentos mencionados.

El **Auxiliar** de servicio es cada una de las personas que tienen el cargo de técnico, secretaria/o u auxiliar de departamento, en los departamentos que dan soporte.

Descripción de cada caso de uso:

- Gestionar lista de verificación
  1. El coordinador de servicio inicia la creación de una lista de verificación.
  2. La aplicación proporciona un formulario para que el coordinador de servicio ingrese los datos de la lista.
  3. El coordinador de servicio ingresa los datos solicitados y guarda la información haciendo clic en el botón guardar.
  4. La aplicación muestra un mensaje que indica que la información se registró con éxito.

- Consultar lista de verificación
  1. El coordinador de servicio solicita a la aplicación la información al hacer clic en una verificación.
  2. La aplicación recibe un identificador que corresponde a la verificación seleccionada para hacer una búsqueda y devolver la información solicitada.

- Gestionar anomalías
  1. El coordinador de servicio solicita a la aplicación una verificación en específico.
  2. La aplicación muestra la verificación que se solicitó.
  3. El coordinador selecciona la opción de agregar anomalía y la aplicación muestra un formulario.
  4. El coordinador ingresa los datos solicitados en el formulario y hace clic en la opción guardar.
  5. La aplicación muestra un mensaje indicando que los datos fueron ingresados con éxito.

- Generar orden de trabajo
  1. El coordinador solicita a la aplicación una verificación de alguna infraestructura o equipo.
  2. La aplicación responde mostrando la información de dicha verificación.
  3. El coordinador hace clic en la opción generar orden de trabajo.
  4. La aplicación muestra un formulario para ingresar datos que se solicitan.
  5. El coordinador ingresa los datos que se solicitan para la creación de la orden de trabajo.
  6. El coordinador hace clic en la opción guardar orden de trabajo y la aplicación responde mostrando un mensaje indicando que los datos fueron guardados con éxito.

La ilustración 3.3 muestra los casos de uso para gestionar las **listas de verificaciones de infraestructuras y equipos**.

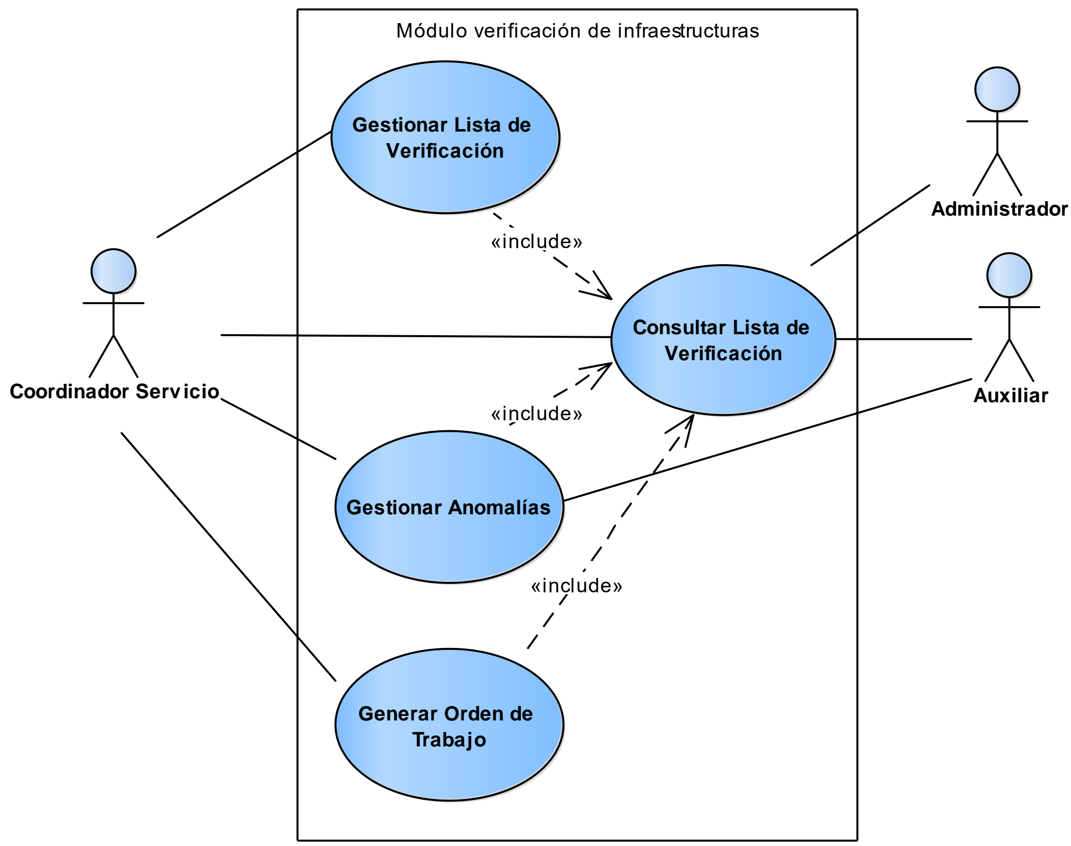


Ilustración 3.3 Casos de uso para la gestión de verificaciones de infraestructuras y equipos.

La ilustración 3.4 muestra los casos de uso para gestionar el **plan de mantenimiento de equipos e infraestructuras**.

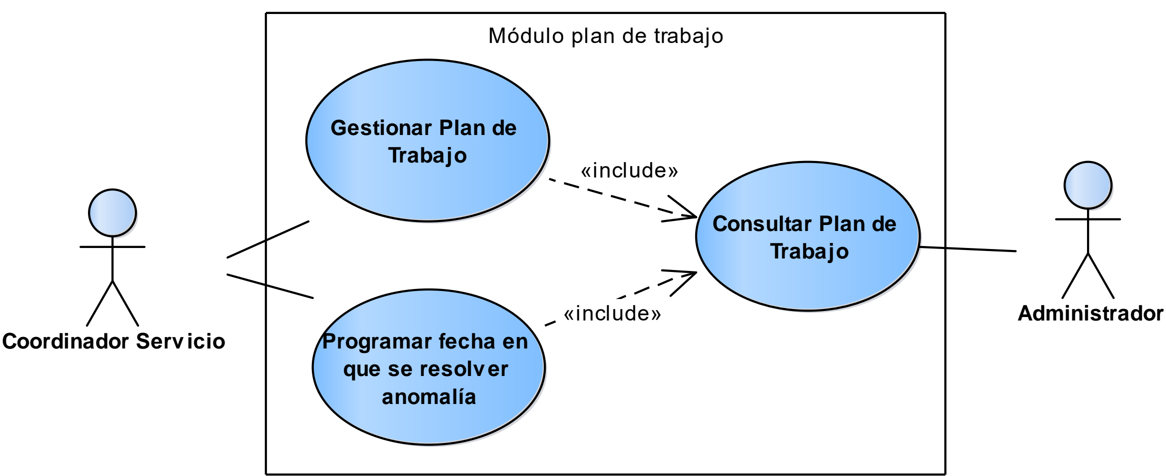


Ilustración 3.4Casos de uso para la gestión del plan de trabajo.

Descripción de casos de uso:

- Gestionar plan de trabajo
    1. El coordinador de servicio solicita a la aplicación iniciar el proceso de registro de un plan de trabajo.
    2. La aplicación proporciona un formulario para que el coordinador introduzca los datos para la creación del plan.
    3. El coordinador de servicio ingresa los datos al formulario y guarda la información.
    4. La aplicación muestra un mensaje indicando que el plan de trabajo se creó con éxito.

- Consultar plan de trabajo
    1. El coordinador de servicio solicita a la aplicación la información al hacer clic en un plan de trabajo.
    2. La aplicación recibe un identificador que corresponde al plan de trabajo seleccionado para hacer una búsqueda y devolver la información solicitada.

- Programar fechas en el plan de trabajo
    1. El coordinador hace clic en la opción plan de trabajo y la aplicación le muestra un conjunto de opciones.
    2. El coordinador hace clic en la opción programar fecha de mantenimiento preventivo y una vez que se indicó la fecha.
    3. La aplicación muestra un mensaje indicando que se actualizó la fecha con éxito.

La ilustración 3.5 muestra los casos de uso para gestionar el módulo **catálogo**, con los cuales cuenta la aplicación, los catálogos por el momento son tres:

- **Proveedores:** En este catálogo se almacenan todos los proveedores externos que brindan apoyo al Instituto Tecnológico para realizar algún tipo de trabajo que no se logró resolver internamente por los técnicos e ingenieros de la institución.

- **Servicios:** Este catálogo almacena los tipos de servicios que brindan los departamentos que dan algún tipo de soporte, es decir, Jardinería, Herrería, Plomería, Albañilería, Obra civil, entre muchos otros.

- **Problemas:** Este catálogo almacena los conceptos, es decir, los tipos de problemas más comunes que se presentan en un equipo o infraestructura, esto con el fin de que el usuario no tenga que escribir, sino más bien solo elija el problema que presenta su equipo.

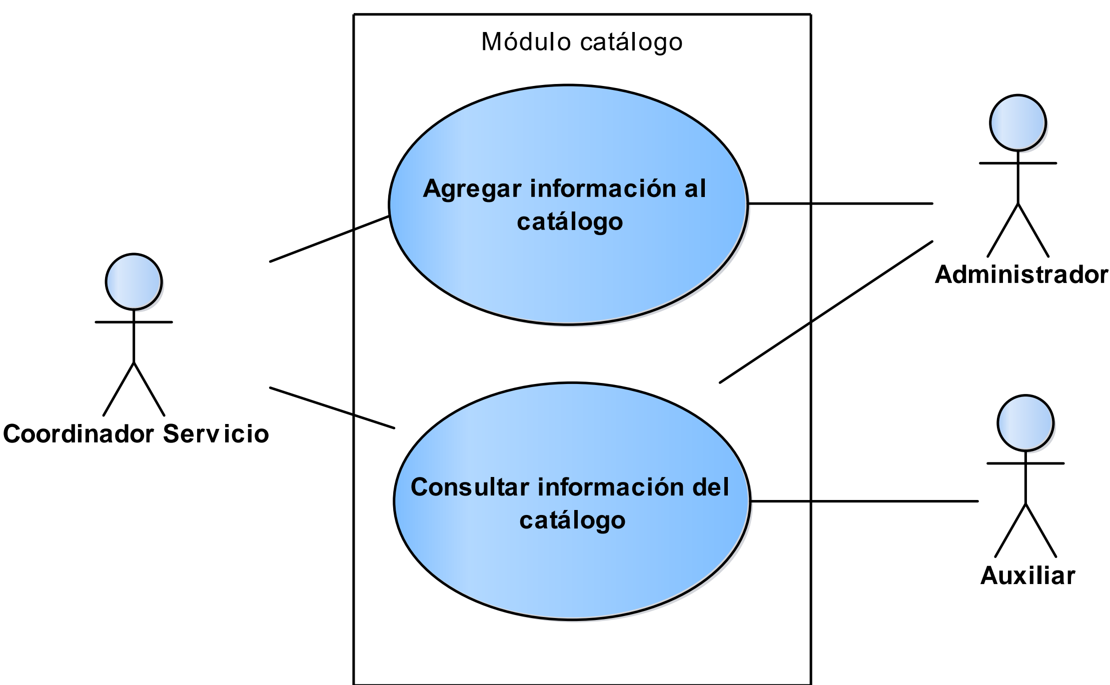


Ilustración 3.5Casos de uso para la gestión de catálogo.

Descripción de casos de uso:

- Agregar información al catálogo
  1. El administrador solicita a la aplicación iniciar el proceso de agregar un registro al catálogo.
  2. La aplicación muestra un formulario para que el administrador o coordinador de servicio introduzcan los datos.
  3. El administrador ingresa los datos al formulario y hace clic en la opción guardar para que la aplicación guarde el registro.
  4. La aplicación muestra un mensaje indicando que el registro se realizó con éxito.

- Consultar información del catálogo
  1. El administrador hace clic en la opción catálogo.
  2. La aplicación muestra una ventana en la que se muestran todos los registros del catálogo.
  3. El administrador ingresa el nombre de registro que quiere encontrar.
  4. La aplicación realiza la búsqueda y responde mostrando la información solicitada por el administrador.

Como se mencionó anteriormente, los diagramas de clases, estado y navegación darán soporte al desarrollo de la aplicación web, en la siguiente sección se describe cada una de ellos, así como la arquitectura lógica de la misma.

3.2 Diseño

En este apartado se presentan tres diagramas, el primero describe la arquitectura lógica de la aplicación, el segundo describe las clases que almacenan la información en tiempo de ejecución, y por último se explica el diagrama de despliegue con la estructura mínima requerida que conforma la aplicación, en el cual se consideran los componentes físicos y sus conexiones.

3.2.1 Arquitectura de la aplicación

Dentro de esta sección se explicará la arquitectura utilizada que da soporte a la aplicación web. En la ilustración 3.6, se puede observar la arquitectura lógica del proyecto, el cual está dividido en capas y a su vez en niveles, es decir, es de tres capas y dos niveles.

Nivel de aplicación: El nivel de aplicación es donde se encuentra toda la estructura de la API REST de ASP.NET CORE 2, el cual controla la petición de servicios solicitados. También es el nivel donde se encuentra el sistema gestor de base de datos **PostgreSQL**.

Nivel de presentación: El nivel de presentación es la parte del Frontend, es decir, la parte que está visible y expuesta al usuario haciendo uso de HTML, CSS, BOOTSTRAP y demás tecnologías que fueron mencionados en el capítulo 2 como el framework Angular.

Las capas que conforman la aplicación son las partes en que fue dividida, separar responsabilidades es una buena práctica, es decir, separar las tareas en módulos que posteriormente son unidos a través de referencias es muy útil, debido a que se estructura mejor el código, haciéndolo reutilizable. A continuación, se describirá cada capa, debido a que cada una de esas capas realiza tareas en específico.

Capa de acceso a datos: Esta capa es donde se ejecuta la base de datos en memoria, es decir, los datos son cargados utilizando un ORM (mapeador de datos), el cual permite utilizar los modelos de datos para crear registros en PostgreSQL, para posteriormente realizar una migración a la base de datos, por lo que, si se requiere actualizar la base de datos haciendo una inserción, actualización, eliminación, se realiza utilizando un lenguaje de programación nativo al lenguaje en el que se está programando la aplicación en este caso es **C#**, haciendo uso de la librería ***EntityFramework***, y un conjunto de librerías que son llamadas en tiempo de ejecución para realizar dicha tarea.

Capa de lógica de negocios: Esta capa permite codificar las reglas de negocio de la institución, con las cuales se debe de cumplir, esto se realiza en el controlador de la **API REST**. Los controladores en esta capa son los métodos que responden a las peticiones del usuario haciendo uso del navegador. Los modelos no son otra cosa que las clases que permiten almacenar y estructurar la información en tiempo de ejecución de la aplicación. También se utilizan para generar la base de datos a partir de las clases programadas en **C#** desde **ASP.NET CORE 2**, este enfoque nos permite un rápido levantamiento de un servicio web orientado a servicios.

Capa de presentación: La capa de presentación es donde se visualizan las ventanas que el usuario final tiene como apoyo para interactuar con la información almacenada en el sistema.

Como se puede observar en la ilustración 3.6 se utilizará el framework **Angular** para realizar la lógica de validación del lado del cliente y estructurar la información proveniente de la base de datos y la que va hacia la base de datos, proporcionada por el usuario, esto con el fin de que el usuario tenga una interacción agradable con el sistema.

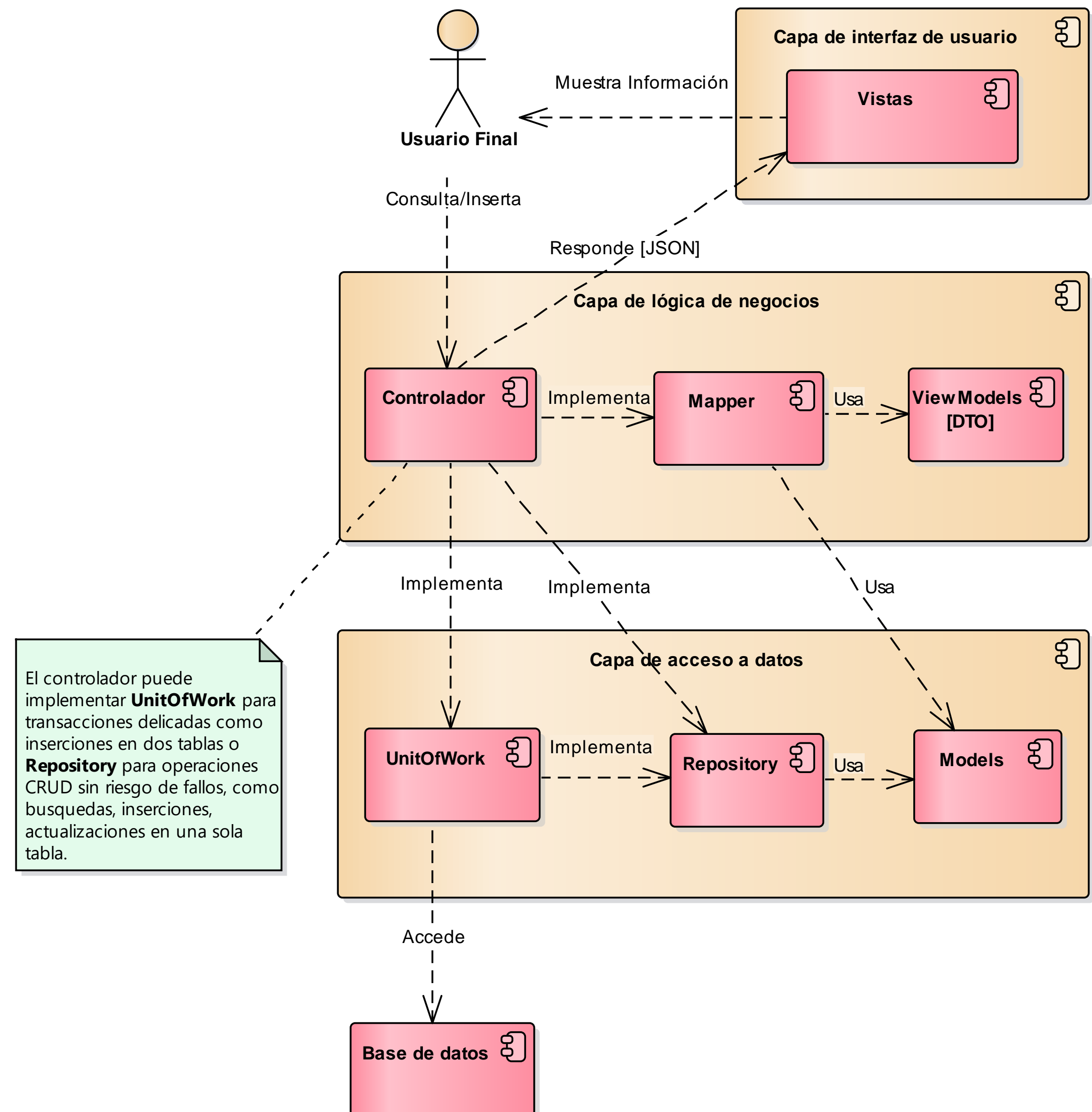


Ilustración 3.6 Arquitectura lógica de la herramienta SOPORTEC.

A continuación, se explica brevemente el flujo que sigue una petición realizada por un usuario a la aplicación. El **usuario** realiza una petición haciendo uso de la vista de la aplicación pero que inicia por el controlador, si es una solicitud para insertar información, la información que llega al controlador hace uso de los **ViewModels,** los cuales son objetos que se crean para almacenar la información con la que interactúa directamente el usuario, esto se hace con la finalidad de que el usuario no tenga contacto directo con los **Modelos** de datos que son los que representan fielmente las estructuras de las tablas en la base de datos. Cuando se hace una inserción o consulta los **Mapper** son métodos que se encargan de transformar los **ViewModels** en modelos de la lógica de negocios, es decir, no se permite

que el usuario final tenga contacto directamente con las entidades de la lógica de negocios, esto con la finalidad de que no puedan ser modificados cuando el usuario no tenga idea de la relación que hay entre dichos modelos. En la capa de acceso a datos se encuentran las entidades que son el núcleo de la aplicación, en la capa de acceso a datos se encuentran las **Entidades** y los repositorios que son un conjunto de métodos que se agrupan para que cada entidad pueda hacer uso de ellos y a su vez la unidad de trabajo (**UnitOfWork**) pueda implementar cada una de las operaciones que se heredan del patrón repositorio (**Repository**). La unidad de trabajo es un patrón que se utiliza para centralizar todas las operaciones posibles que se puedan realizar sobre una entidad, **Repository** dictará a través de la herencia que método implementará **UnitOfWork,** es decir, que en el flujo normal de la vida de una petición en la aplicación el controlador manda a llamar a **UnitOfWork** si requiere hacer una inserción, actualización, lectura o eliminación. No se necesita en cada **Entidad** declarar cada uno de dichos métodos para cada clase que representa la lógica de negocios. Esto permitirá agilizar el escalado del proyecto, es decir, si se agregará una nueva clase (porque así lo requiera el negocio), normalmente seria agregar la entidad, agregarle todos sus métodos y hacer esa llamada en el controlador, con el uso de UnitOfWork el panorama cambia, ya que, solo se agrega la entidad y por herencia de Repository todos los métodos los hereda de dicha clase. La información que se obtiene de la base de datos se formatea utilizando nuevamente los Mappers y en vez de enviar los modelos originales de la lógica de negocios se envían los ViewModels que fueron creados para eso.

En la sección siguiente se explicarán los diagramas de clases que conforma la capa de lógica de negocio mencionada anteriormente en la ilustración 3.6. Es importante remarcar como se mencionó en el capítulo 2, el diagrama de clases nos permite almacenar la información en tiempo de ejecución de la aplicación, cuando se introduce información, así como, cuando se extrae de la base de datos.

3.2.2 Diagrama de clases

Se elaboró el diagrama de clases que contempla los procesos analizados en secciones anteriores, dichos procesos son: plan de trabajo, orden de trabajo, y verificación de infraestructuras. Se colocaron en las clases solo los atributos y funciones que tienen una relevancia importante para el modelado, esto con la finalidad de que sean legibles (ver ilustración 3.7).

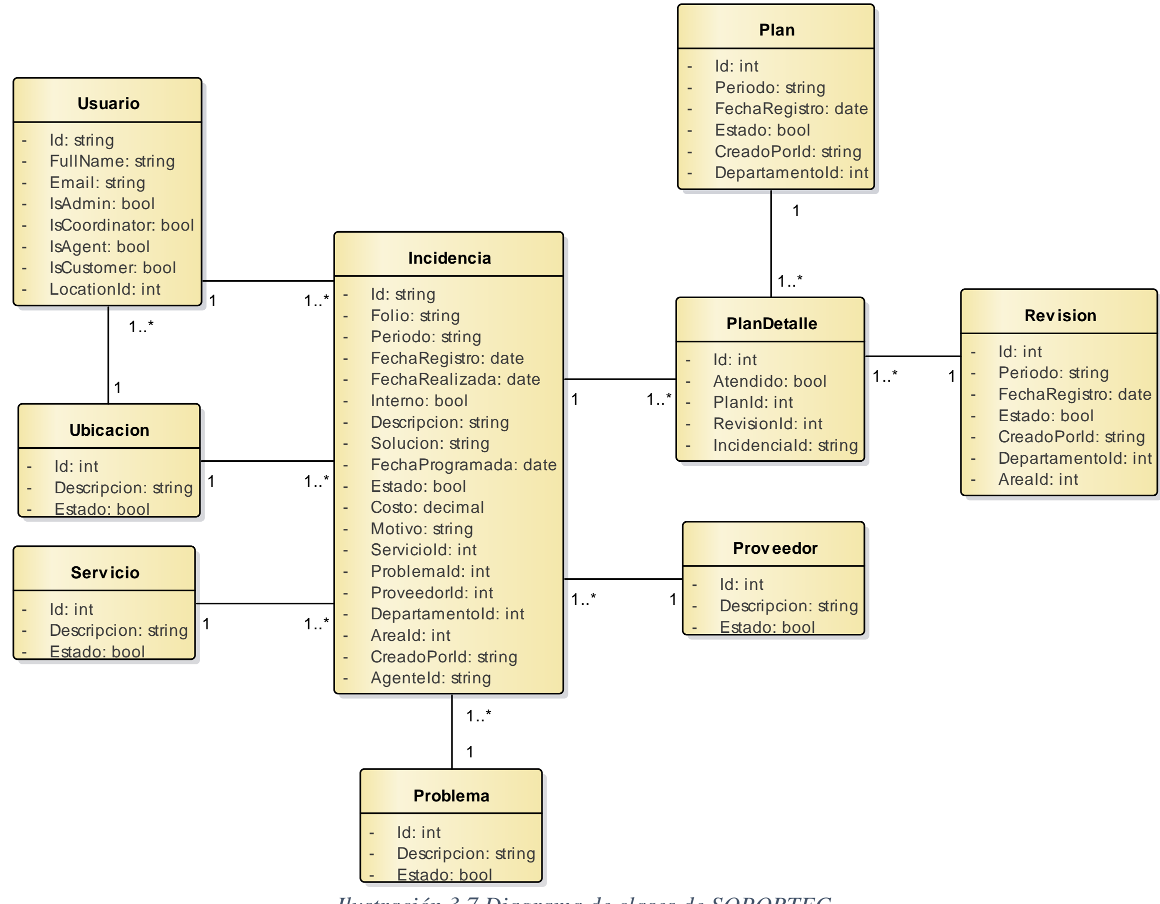


Ilustración 3.7 Diagrama de clases de SOPORTEC.

Descripción de las clases:

- **Plan**

Esta clase almacena la información de los planes que se crean cada semestre, esta clase se crea debido que hay información que se debe almacenar de un plan de trabajo.

Tabla 3.2 Clase para la gestión de planes de trabajo.

Atributos	Tipo de dato	Descripción de atributo
Id	Entero	Permite almacenar un identificador con el cual se identificará el plan.
Periodo	Cadena	Almacena el periodo en el que se creó el plan de trabajo, esto con la finalidad de facilitar las búsquedas por periodos. Los cuales pueden ser dos posibles valores: ENE-JUN o AGO-DIC seguido del año en que se genera dicho plan.
FechaRegistro	Date	Este campo almacena la fecha en que fue creado el plan de trabajo.
Estado	Boolean	Este campo almacena el estado del plan de trabajo, es decir, si se encuentra en estado ACTIVO o INACTIVO.

CreadoPorId	Cadena	Este campo almacena el identificador del usuario que creó el plan de trabajo.
DepartamentoId	Entero	Almacena el identificador del departamento que creó el plan de trabajo.

- **Usuario**

Esta clase almacena la información de los usuarios que se autentican en la aplicación y que hacen uso de ella.

Tabla 3.3 Clase para la gestión de Usuarios.

Atributos	Tipo de dato	Descripción de atributo
Id	Cadena	Permite almacenar un identificador con el cual se identificará el usuario.
FullName	Cadena	Almacena el nombre del usuario para que este pueda ser identificado o ser llamada de alguna manera.
Email	Cadena	Almacena el correo del usuario, es importante debido a que cualquier notificación se hará a este medio de comunicación.
IsAdmin	Boolean	Almacena la fecha en que fue creado el usuario.
IsCoordinator	Boolean	Almacena el correo encriptado del usuario, esto como una medida de seguridad.
IsAgent	Boolean	Almacena el teléfono del usuario.
IsCustomer	Boolean	Almacena el identificador del rol que le corresponde a dicho usuario, esto con la finalidad de que cada usuario que es registrado tenga ciertos permisos en la aplicación.
LocationId	Entero	Este campo almacena el identificador del departamento al cual pertenece el usuario registrado.

- **Ubicacion**

Esta clase almacena la información de los departamentos que hacen uso de la aplicación, se almacena los departamentos que dan soporte, así como los que reciben dicho soporte, son distinguidos por el campo llamado servicio.

Tabla 3.4 Clase para la gestión de Departamentos.

Atributos	Tipo de dato	Descripción de atributo
Id	Entero	Permite almacenar un identificador con el cual se identificará el departamento.
Descripcion	Cadena	Almacena el nombre del departamento para que este pueda ser identificado o ser llamado de alguna manera.

Servicio	Booleano	Almacena dos posibles valores, es decir, si el departamento es un departamento que brinde soporte o no.

- **Revision**

Esta clase almacena la información de las verificaciones de infraestructura y equipo, es decir, almacena el encabezado de un documento que indica quien creo el documento y a qué área se le realizará la verificación, así como la fecha de creación y estado del mismo.

Tabla 3.5 Clase para la gestión de revisiones.

Atributos	Tipo de dato	Descripción de atributo
Id	Entero	Permite almacenar un identificador con el cual se identificará el plan.
Periodo	Cadena	Almacena el periodo en el que se creó el plan de trabajo, esto con la finalidad de facilitar las búsquedas por periodos. Los cuales pueden ser dos posibles valores: ENE-JUN o AGO-DIC seguido del año en que se genera dicho plan.
FechaRegistro	Date	Este campo almacena la fecha en que fue creado el plan de trabajo.
Estado	Boolean	Este campo almacena el estado del plan de trabajo, es decir, si se encuentra en estado ACTIVO o INACTIVO.
CreadoPorId	Cadena	Este campo almacena el identificador del usuario que creó el plan de trabajo.
DepartamentoId	Entero	Almacena el identificador del departamento que creó el plan de trabajo.
AreaId	Entero	Almacena el identificador del departamento al cual se le hará una verificación de infraestructura y equipo.

- **PlanDetalle**

Esta clase almacena la información que relaciona un plan de trabajo, una verificación y una incidencia reportada, esto es así, debido a que en una verificación se reportan incidencias que posteriormente estarán en un plan de trabajo para resolverse según la fecha que se proporcione cuando se programen todas las anomalías encontradas en las verificaciones.

Tabla 3.6 Clase para la gestión de la relación de un plan con la incidencia.

Atributos	Tipo de dato	Descripción de atributo
Id	Cadena	Permite almacenar un identificador para hacer único al detalle del plan.
Atendido	Booleano	Almacena dos posibles valores, es decir, si la incidencia asignada a una verificación fue resuelta en el momento que se reporta.

PlanId	Entero	Este atributo almacena la información del plan que le corresponde a dicha incidencia y revisión.
RevisionId	Fecha	Almacena el identificador de la revisión que le corresponde a dicho detalle.
IncidenciaId	Fecha	Almacena el identificador de la incidencia que le corresponde al detalle del plan.

- **Incidencia**

Esta clase almacena información de las incidencias que se reportan para un mantenimiento preventivo, también se podría almacenar información para un mantenimiento correctivo.

Tabla 3.7 Clase para la gestión de tickets (incidencias).

Atributos	Tipo de dato	Descripción de atributo
Id	Cadena	Permite almacenar un identificador para hacer único a la incidencia reportada.
Folio	Cadena	Almacena el número de folio que se le asigna a una incidencia.
Periodo	Cadena	Almacena el periodo en el que fue registrada la incidencia.
FechaRegistro	Fecha	Este campo almacena la fecha en que se registró la incidencia.
FechaRealizada	Fecha	Este campo almacena la fecha en que se llevo a cabo la reparación del problema reportado.
Interno	Booleano	Almacena dos posibles valores, es decir, si el mantenimiento es interno o es externo.
Descripcion	Cadena	Almacena la descripción del problema reportado.
Solucion	Cadena	Almacena una descripción de la solución del problema reportado.
FechaProgramada	Fecha	Este campo almacena la fecha en que se realizará el trabajo que dará solución al problema reportado.
Estado	Boolean	Este campo almacena el estado de la incidencia.
Costo	Numero	Este campo almacena el costo que conlleva realizar el problema reportado.
Motivo	Cadena	Este atributo almacena la información del motivo por el cual el mantenimiento fue reprogramado.
ServicioId	Entero	Almacena el identificador del servicio que se brinda.
ProblemaId	Entero	Almacena el identificador del problema que se reportó.
ProveedorId	Entero	Almacena el identificador del proveedor que dará soporte, puede ser el Tecnológico o cualquier otra empresa externa.

DepartamentoId	Numero	Este campo almacena el identificador del departamento que dará soporte al problema reportado.
AreaId	Numero	Este campo almacena el identificador del departamento que reporta alguna incidencia.
CreadoPorId	Cadena	Este campo almacena el identificador del usuario que reporta la incidencia.
AgenteId	Cadena	Este campo almacena el identificador del agente que dará solución al problema reportado.

- **Proveedor**

Esta clase almacena información de los proveedores que brindan servicio a la institución.

Tabla 3.8 Clase para la gestión de los proveedores.

Atributos	Tipo de dato	Descripción de atributo
Id	Entero	Permite almacenar el identificador primario del proveedor.
Descripcion	Cadena	Almacena el nombre del proveedor para identificarlo de una forma rápida.
Estado	Boolean	Este campo almacena el estado del registro, es decir, si está activo será igual a true, y si esta desactivado el registro tendrá un valor false.

- **Servicio**

Esta clase almacena información de los servicios más comunes realizados en la institución.

Tabla 3.9 Clase para la gestión de servicios.

Atributos	Tipo de dato	Descripción de atributo
Id	Entero	Permite almacenar el identificador primario del Servicio.
Descripcion	Cadena	Almacena el nombre del servicio más comunes realizados.
Estado	Boolean	Este campo almacena el estado del registro, es decir, si está activo será igual a true, y si esta desactivado el registro tendrá un valor false.

- **Problema**

Esta clase almacena información de los problemas más comunes que se reportan en las solicitudes.

Tabla 3.10 Clase para la gestión de problemas.

Atributos	Tipo de dato	Descripción de atributo

Id	Entero	Permite almacenar el identificador primario del Problema.
Descripcion	Cadena	Almacena el nombre de los problemas más comunes reportados en la institución.
Estado	Boolean	Este campo almacena el estado del registro, es decir, si está activo será igual a true, y si esta desactivado el registro tendrá un valor false.

Como se observa en la ilustración 3.7 las clases en el diagrama no cuentan con operaciones para crear, consultar, eliminar y actualizar, esto es porque dicha lógica se realiza según la arquitectura de la aplicación (ver ilustración 3.6) en la capa de acceso a datos, específicamente en los componentes llamados **Repository,** que es donde se programa la lógica de cómo debe operar la aplicación, en base a las **historias de usuario** proporcionados por el cliente (**product owner).**

Diagrama de clases del patrón de diseño Repository

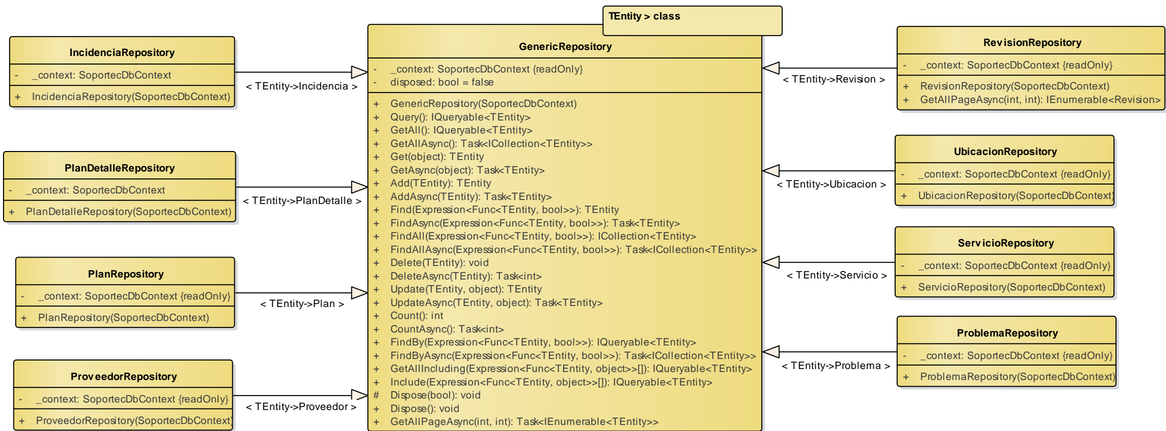


Ilustración 3.8 Diagrama de clases del patrón Repository.

En la ilustración 3.8 se puede observar las clases que permiten persistir la información, en el diagrama de la ilustración 3.7 no contenían métodos que realizaran las operaciones básicas de crear, actualizar, leer, eliminar y esto es debido a que se implementa en la aplicación patrones que permiten reutilizar código. La interface **IRepository** contiene un grupo de firmas (métodos) que el programador considera básicas en cada una de las clases de la lógica de negocios, la clase **Repository** hereda de la interfaz **IRepository** (ver ilustración 2.9) dichas firmas y las implementa, es decir, que en esta clase llamada **Repository** se implementa la lógica de crear, actualizar, leer y eliminar de cada una de las clases del negocio, es importante mencionar que para que esto funcione, la interface **IRepository** y la clase **Repository** deben ser genéricas.

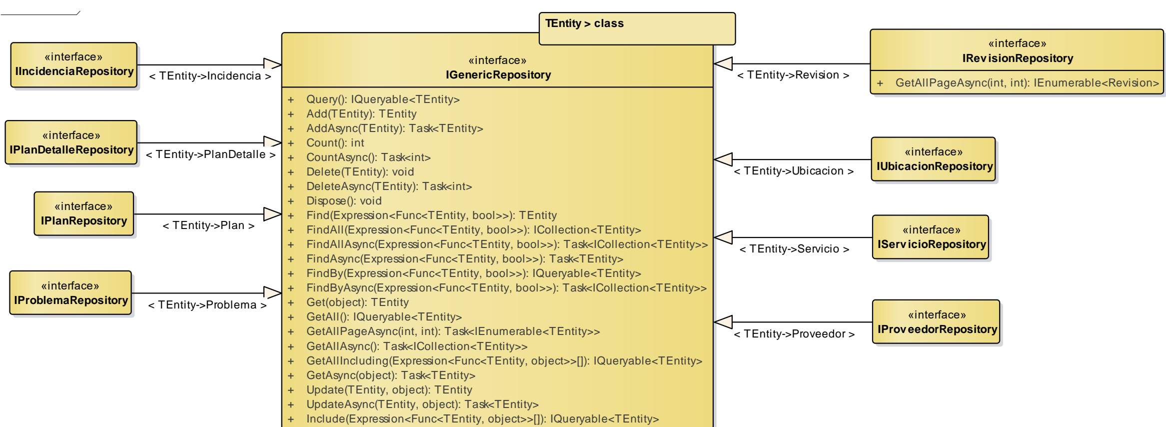


Ilustración 3.9 Interfaces que dictan los métodos que serán heredados en cada clase del modelo de dominio.

La ilustración 3.9 muestra las firmas que cada clase podrá ejecutar, en sí la aplicación no contiene lógica fuerte en la cual se hagan operaciones como transacciones de dinero, operaciones en tiempo real o algún tipo de información delicada que requiera de tiempos de respuestas muy cortos, en otras palabras, la aplicación califica como una aplicación CRUD en la que solo se hacen inserciones, actualizaciones, consultas de información pasando parámetros como fechas, cantidades o algún termino de búsqueda, es por eso que como se puede observar en las clases, las firmas que se implementan son las más comunes utilizadas en una aplicación, si se requiriera de algún tipo de lógica más compleja en el futuro esta lógica podría ser escrita en la clase que implementa dicha interface, es decir, suponiendo que en algún momento el dueño del producto (cliente) se lo ocurriera enviar notificaciones en tiempo real de cuando una incidencia es resulta, esta lógica podría ser escrita en el repositorio de la clase **Incidencia** en la interface **IIncidenciaRepository** se crearía una firma llamada Notify (string message) y la clase que implemente dicha interface exigirá que se implemente dicho método, es ahí donde se escribirá la lógica cumpliendo así un buen principio de diseño de los muy mencionados principios **SOLID**, el cual dice que cada clase debe de tener una única responsabilidad. El patrón utilizado para la gestión de la información en este proyecto cumple con dichos principios, si se necesitara agregar más reglas de negocio a la aplicación de ninguna manera afectaría a los métodos ya implementados, dicho en otras palabras, la aplicación se vuelve escalable.

3.2.3 Diagrama de estado

Los diagramas de estado son utilizados en modelado de software (como se explicó en el capítulo 2) específicamente donde se generan documentos, esto con el fin de ver el estado que va tomando los documentos que se generan. En esta sección se hablará del diagrama de estados de un documento (solicitud de mantenimiento), a partir de que se crea dicha solicitud hasta cuando se crea una orden de trabajo y se cierra dicha orden de trabajo.

En el diagrama mostrado en la ilustración 3.10 se puede ver cuatro estados, los cuales son los siguientes: **Nuevo**, **Proceso**, **Cerrado** y **Cancelado**.

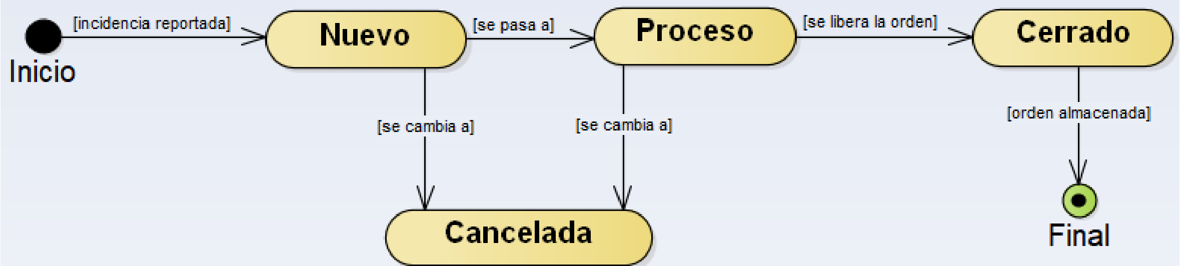


Ilustración 3.10 Diagrama de estado de una anomalía (incidencia).

A continuación, se describen cada uno de los estados que se muestran en el diagrama de la ilustración 3.9.

- *Nuevo***.** La solicitud es nueva, cuando se envía y los jefes de departamento evalúan si cuentan con el material y herramientas necesarias para llevar a cabo el trabajo.
- *Proceso***.** La solicitud adquiere el estado proceso cuando el jefe de departamento que brinda el servicio evaluó que cuenta con los materiales necesarios para realizar el trabajo solicitado.
- *Cerrado*. La solicitud está en cerrado cuando ya se liberó el trabajo, es decir, las persona a las que se les realizó el trabajo, firmaron la orden de trabajo indicando que se solucionó el problema que solicitaron fuera atendido.
- *Cancelada*. La solicitud puede estar en cancelada cuando se decide que el ya no se atenderá una incidencia o que alguien solucionó el problema sin la necesidad del técnico.

Los estados mostrados anteriormente son por los que pasará cada solicitud que se crea y se le dé seguimiento, es importante mencionar que en el diagrama de estado que se muestra en la ilustración 3.10, se incluyen los procesos que se llevan a cabo para una orden de trabajo, debido a que por cada solicitud creada se le crea una orden de trabajo.

3.2.4 Diagramas de navegación

En esta sección se explican los diagramas de navegación los cuales son una vista general de las rutas posibles que tendrá el sistema, estas rutas son opciones en las cuales el usuario que ingrese al sistema dependiendo del rol con el que fue registrado, podrá ejecutar en algún determinado momento.

Como se muestra en la ilustración 3.11 se puede observar que el menú es para un usuario con el tipo de rol **Auxiliar**, este usuario es el que tiene más limitaciones ya que este solo puede ver información de él mismo, iniciar sesión, cerrar sesión, consultar ordenes de trabajo que debe realizar, buscar ordenes de trabajo, mostrar información de las verificaciones, registrar anomalías en el proceso de verificación, es decir, cuando se van a realizar revisiones a los departamentos con los que cuenta la institución.

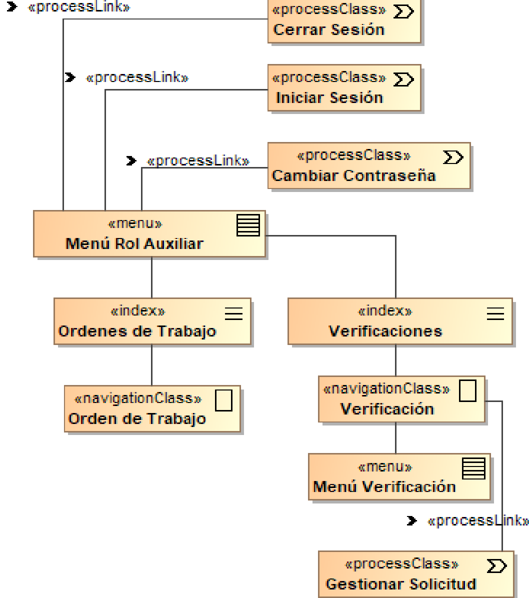


Ilustración 3.11 Menú de navegación de un usuario con el rol de Auxiliar.

En la ilustración 3.12, se puede observar el diagrama de navegación del usuario con el rol de administrador del sistema, quien solo puede dentro de sus funciones más fuertes crear usuarios, ver reportes de todos los módulos que tiene el sistema, sin embargo, el usuario no puede ejecutar ciertas tareas, debido a que no le corresponde dentro de sus funciones, por ejemplo: aprobar órdenes de trabajo o verificaciones de infraestructura.

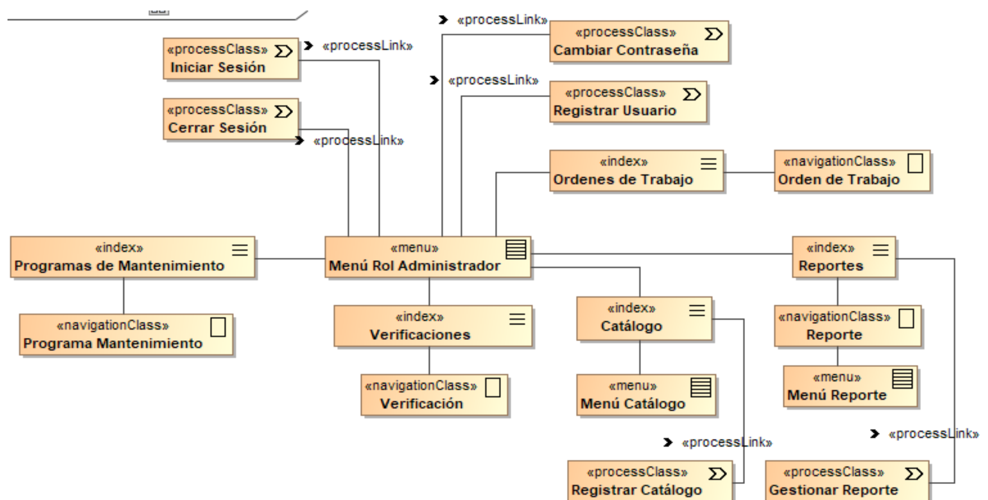


Ilustración 3.12 Menú de navegación de un usuario con el rol de Administrador.

El diagrama mostrado en la ilustración 3.13, es el diagrama de navegación de los usuarios registrados con el rol de **Coordinador de Servicios,** el cual es el jefe de departamento que realiza mantenimientos a los equipos y/o edificios de la institución. Como se puede observar el usuario con este tipo de rol puede ejecutar todas las opciones posibles configuradas en el sistema, como gestionar solicitudes, aprobar ordenes de trabajo, gestionar listas de verificaciones de infraestructuras y/o equipos, consultar información, generar reportes, gestionar usuarios y mantenimientos.

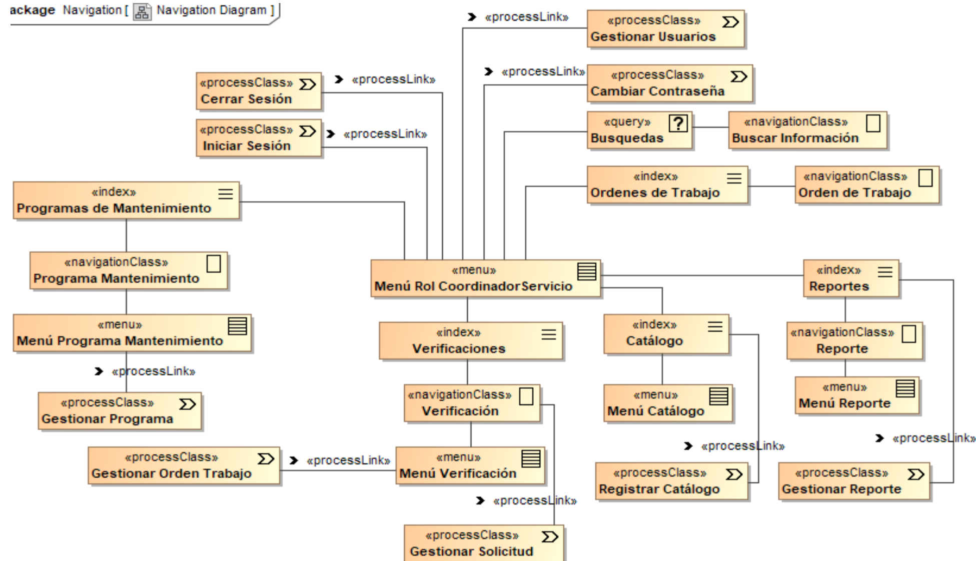


Ilustración 3.13 Menú de navegación de un usuario con el rol de Coordinador.

Los diagramas mostrados en esta sección darán soporte al desarrollo de las vistas del sistema, ofreciendo una visión general del sistema, detallando y delimitando las tareas que cada usuario podrá realizar.

3.2.5 Diagrama de despliegue

El diagrama de despliegue permite visualizar la estructura (componentes físicos y librerías) necesaria para que la aplicación a desarrollar pueda ser desplegada en producción. En la ilustración 3.14 se muestran las herramientas que serán utilizadas en el desarrollo de la aplicación, así como la forma en que interactúan.

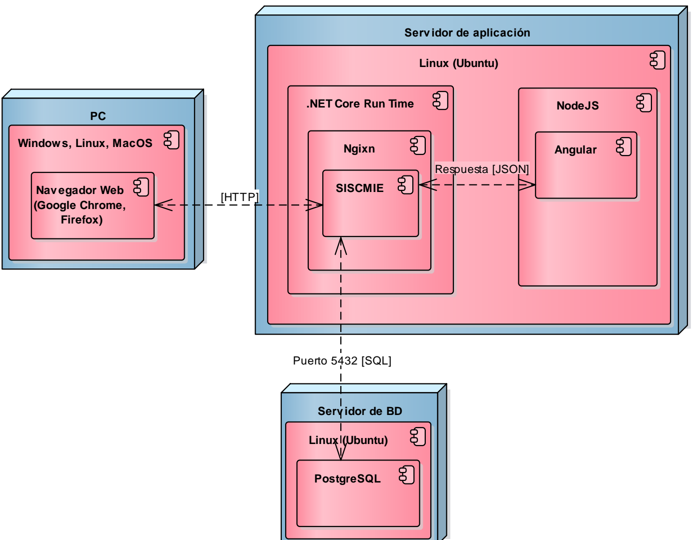


Ilustración 3.14 Diagrama de despliegue de soportec.

Como se muestra en la ilustración 3.14 están involucradas varias tecnologías para el desarrollo del sistema, la descripción de cada tecnología es la siguiente:

- ***Navegador web***: Navegador web Google Chrome o Firefox que se utiliza para consultar el sistema.
- ***Servidor de aplicación***: El servidor de aplicación contiene un sistema operativo Linux con una distribución Ubuntu 19, en la cual se tiene instalado las librerías de .NET Core para la ejecución y puesta en producción de la aplicación. A si mismo tiene instalado un servidor web Ngixn.
- ***Aplicación ANGULAR 7***: Aplicación angular que ayudará al usuario a interactuar con la base de datos y que para desplegarse en producción requiere de las librerías de NodeJS.
- ***Servidor de base de datos***: El servidor de base de datos tiene instalado el sistema operativo Linux con la distribución Ubuntu 19. Dicha distribución tiene instalada una copia del sistema gestor de base de datos relacional **PostgreSQL 6.5**, el motivo de instalar esta versión de **PostgreSQL** es que es una versión muy recomendada por ser estable.

En este capítulo se han detallado los diagramas que darán soporte al desarrollo de la aplicación, el diagrama de la ilustración 3.1 sección **modelado de negocio** contiene los procesos que se automatizarán con el desarrollo de la aplicación, todos los diagramas generados en esta sección fueron diseñados respetando cada uno de los procesos del negocio. En el siguiente capítulo se describirá el proceso de desarrollo del proyecto presentado en esta tesis, haciendo uso de las tecnologías descritas en el capítulo 2 con base al análisis y diseño realizado durante este capítulo.

Capítulo 4 Implementación

Como se mencionó en el capítulo 2 el marco de trabajo utilizado para desarrollar la aplicación es Scrum, la entrega del producto se hará en incrementos, es decir, en pequeños módulos. Específicamente el desarrollo de la aplicación será en 5 Sprints (ver tabla 4.1), donde el primer Sprint se utiliza para explicar la configuración básica de la aplicación, es decir, la implementación de la capa de acceso a datos. En el segundo Sprint se implementa la creación, autenticación y autorización de usuarios de la aplicación, en el tercer Sprint se presenta la implementación del catálogo con el que contará la aplicación. En el cuarto Sprint se detalla la implementación de la creación de listas de verificación, así como su consulta y las operaciones relacionadas con la misma, por último, en el quinto y último Sprint se implementa la creación del plan de trabajo de las anomalías que se deben de resolver según la información almacenada en las listas de verificación.

Tabla 4.1 Planeación de los Sprints para el desarrollo de la aplicación.

Sprint	Historia de usuario	Fecha de inicio	Fecha de finalización	Duración del Sprint (semanas)
1		01-05-2019	31-05-2019	4
2	HU1	01-06-2019	30-06-2019	4
3	HU2, HU3	01-07-2019	07-08-2019	6
4	HU4, HU5 y HU6	08-08-2019	07-10-2019	10
5	HU7, HU8, HU9 y HU10	08-10-2019	20-12-2020	10
Duración total del desarrollo de la aplicación				8 meses y 2 semanas

En este capítulo se explica el desarrollo de la aplicación la cual se diseñó y analizó en el capítulo anterior. El **product owner**, el **equipo de desarrollo** y el **Scrum master** hicieron reuniones para priorizar y planificar los **Sprints**. En cada junta se presentaba el **product backlog** y el **product owner** indicaba que historias de usuario quería primero (mayor a menor relevancia) una vez que se tenía el **Sprint Backlog** el **equipo de desarrollo** estableció la duración de cada **Sprint** con base a las historias de usuarios presentadas, es decir, considerando la complejidad de cada tarea es el tiempo que los desarrolladores le proporcionaban a dicho **Sprint.**

En la tabla 4.1 se puede observar que el primer Sprint no cuenta con historias de usuario asignadas, esto es debido a que ese Sprint es dedicado únicamente a la instalación y configuración de los entornos de desarrollo, así como la preparación del proyecto base y programación de la capa de acceso a datos de las cuales el **product owner** no es consiente, pero para los desarrolladores si, y es por eso que se decide asignar un tiempo a dichas tareas que son indispensables.

Antes de iniciar la explicación de los Sprints es necesario presentar las configuraciones básicas del proyecto, es decir, la configuración necesaria para el desarrollo de la aplicación.

4.1 Configuración requerida para el desarrollo de la aplicación

En la práctica las tecnologías con que se desarrolla una aplicación se dividen en dos grandes intereses: **backend** y **frontend**. El backend es todo aquello que se ejecuta del lado del servidor de aplicación y el frontend es todo aquello que se ejecuta del lado del cliente, en este caso, el navegador web.

En la tabla 4.2 se describen las tecnologías **backend** que fueron instaladas para el desarrollo de la aplicación.

Tabla 4.2 Programas instalados para desarrollo de la aplicación.

<table>
<tr><th>Programa</th><th>Versión</th><th colspan="2">Sitio web de descarga</th></tr>
<tr><td>Visual Studio Community</td><td>2019 v4.8.03752</td><td colspan="2">https://visualstudio.microsoft.com/es/</td></tr>
<tr><td>NodeJS</td><td>v13.8.0</td><td colspan="2">https://nodejs.org/es/download/</td></tr>
<tr><td>PostgreSQL</td><td>V9.5</td><td colspan="2">https://www.postgresql.org/download/windows/</td></tr>
<tr><td>PgAdmin</td><td>V3.0</td><td colspan="2">https://www.pgadmin.org/download/pgadmin-4-windows/</td></tr>
<tr><td>Firefox Mozilla</td><td>v78.0</td><td colspan="2">https://www.mozilla.org/es-MX/firefox/new/</td></tr>
<tr><td>Angular CLI</td><td>v9.0</td><td colspan="2">https://www.npmjs.com/package/@angular/cli</td></tr>
<tr><th colspan="4">Características de la PC de desarrollo</th></tr>
<tr><td>S.O</td><td>Procesador</td><td>Memoria RAM</td><td>Disco Duro</td></tr>
<tr><td>Windows 10 de 64bits</td><td>Intel Core i5 8th generación</td><td>8GB</td><td>1TB</td></tr>
</table>

Como se mencionó en el capítulo 3 para que la aplicación sea ejecutada con éxito debe de cumplir con algunos requerimientos mínimos de hardware y software, en la tabla 4.3 se muestran los requerimientos necesarios para la puesta en producción de la aplicación.

Tabla 4.3 Requerimientos de hardware, programas y utilidades para el despliegue de la aplicación..

SERVIDOR DE APLICACIÓN
Software

Programa	Versión	Sitio de descarga	Año de descarga
.NET Core runtime	2.2	https://docs.microsoft.com/es-es/dotnet/core/install/linux-package-manager-ubuntu-1910	2019
Ngixn	18	http://nginx.org/en/linux_packages.html#Ubuntu	2019
PostgreSQL	9.5	https://www.postgresql.org/download/windows/	2019
NodeJS		https://nodejs.org/es/download/	2019
Angular CLI	7	https://www.npmjs.com/package/@angular/cli	2019

Como se observa en la tabla 4.3 solo se utilizan librerías que no son de un tamaño considerablemente grande, más bien se instala el núcleo de cada una de ellas, pues como el proyecto ya está compilado y testeado cuando se despliega, no es necesario tener un gran número de librerías instaladas en el servidor de producción.

Antes de iniciar el primer Sprint se mostrará el proceso de la instalación del SDK para la creación de aplicaciones web con el marco de desarrollo ASP.NET Core MVC.

4.2 Instalación de ASP.NET Core 2

Como en todo desarrollo sucede se debe de contar con el entorno adecuado para el desarrollo de una aplicación, es por eso que como se mencionó en el capítulo 2 el entorno de desarrollo que permitirá hacer uso de las librerías necesarias para realizar con éxito la aplicación que se desarrollará es Visual Studio Community 2019. El primer paso para descargar las librerías que brindan el soporte para desarrollo web se encuentran en la siguiente página oficial de Microsoft **https://dotnet.microsoft.com/download/dotnet-core/2.2** (ver ilustración)

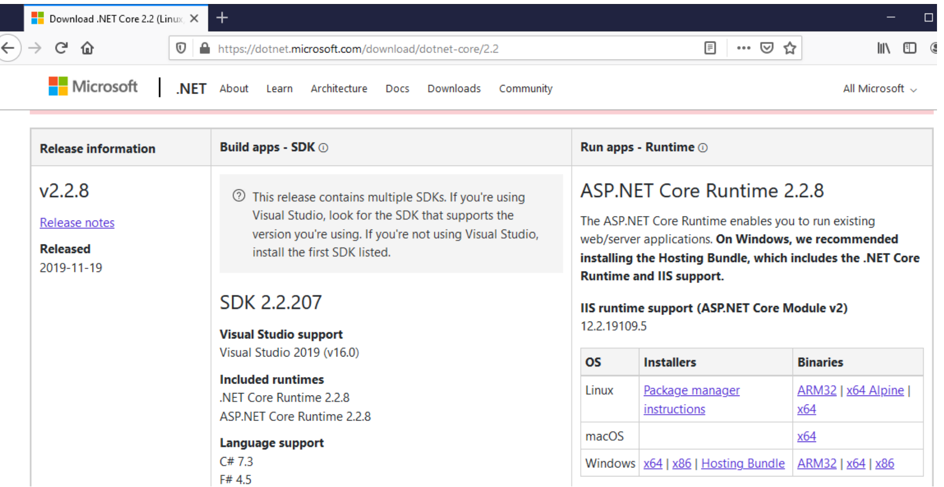


Ilustración 4.1Página oficial para la descarga del SDK de ASP.NET Core 2.2 (2019).

En la ilustración 4.1 se observan tres secciones para descargar el paquete de librerías, para el desarrollo de aplicaciones en ASP.NET Core es necesario descargarlo e instalarlo. En la

página oficial se encuentran las librerías de desarrollo y las de producción, es decir, cuando se despliegue en producción la aplicación solo se instalará el run time .NET Core. Como se mencionó en el capítulo dos una de las razones por las que se eligió ASP.NET Core es porque se puede ejecutar en diversas plataformas (Windows, Linux, MacOS). Para el desarrollo del presente proyecto se descargó la versión del SDK para Windows (ver ilustración 4.2).

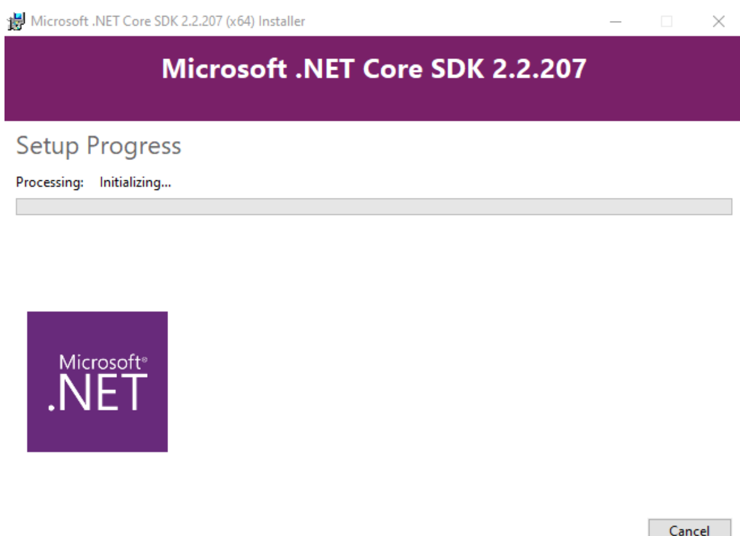


Ilustración 4.2 Proceso de instalación del SDK de ASP.NET Core 2.

Una vez que la instalación inició no se configura nada, solo es esperar hasta que se muestre la ventana de la ilustración 4.3.

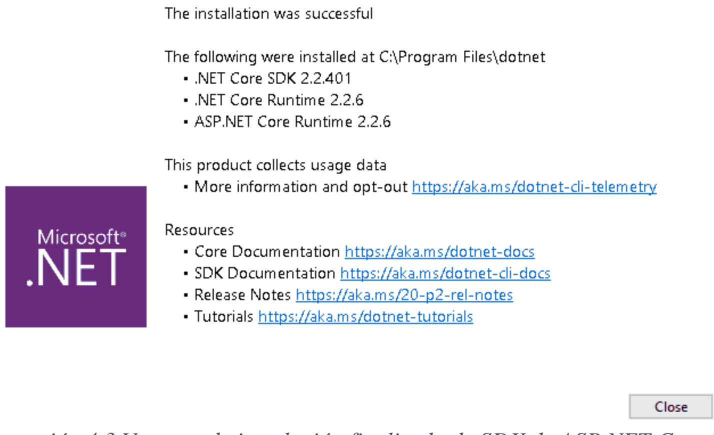


Ilustración 4.3 Ventana de instalación finalizada de SDK de ASP.NET Core 2.

Cuando termina la instalación del SDK, para verificar que se instaló correctamente se abre Visual Studio 2019 y se verifica buscando la librería SDK ASP.NET Core 2.

A continuación, se describe el proceso para la creación de la solución en Visual Studio que contendrá toda las librerías, clases y lógica de negocios que son propias del sistema a desarrollar. Cuando se ejecuta Visual Studio se muestran diferentes opciones para crear un proyecto, librería o extensiones (ver ilustración 4.4) para desarrollar la aplicación propuesta se creó un nuevo proyecto, para ello se seleccionó la opción **Crear Proyecto** (ver indicador 1 de la ilustración 4.4).

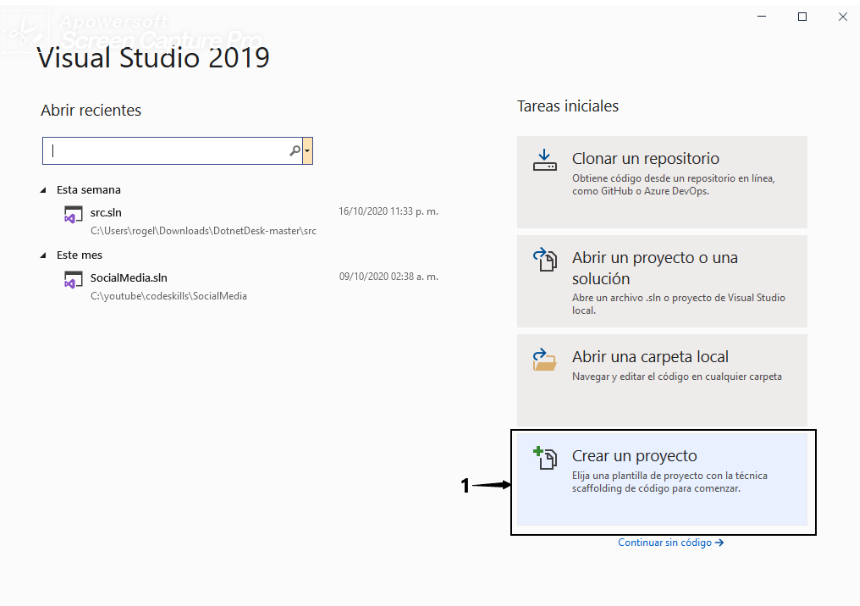


Ilustración 4.4 Asistente de Visual Studio para crear un proyecto.

Después de seleccionar la opción de crear proyecto, el asistente muestra una ventana con diversas opciones que permiten crear diferentes tipos de proyectos, librerías y extensiones, para el desarrollo de la plataforma se usará **aplicación web ASP.NET Core** (ver indicador 1 de ilustración 4.5).

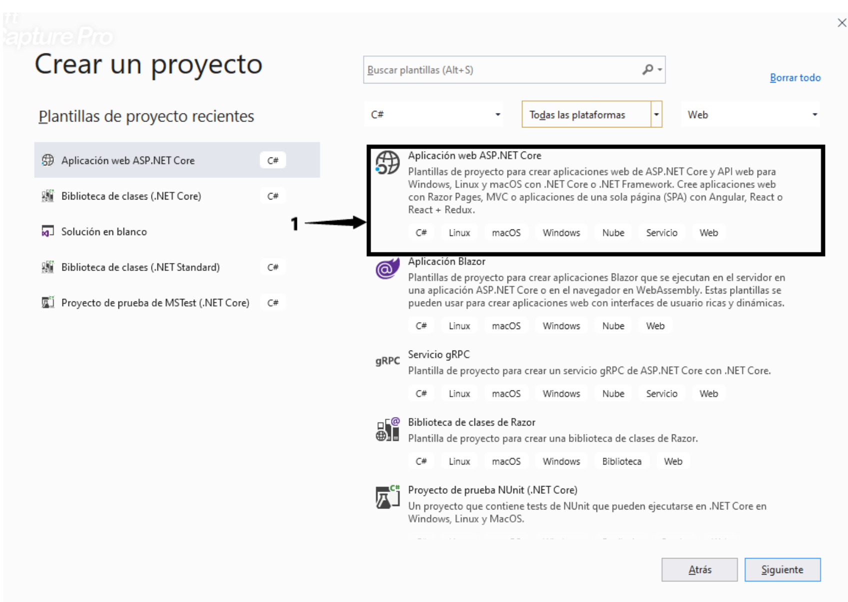


Ilustración 4.5 Ventana para seleccionar el tipo de proyecto a crear.

Después de seleccionar el tipo de proyecto, es necesario dar un nombre y ubicación al proyecto que se está creando (ver indicador 1) y posteriormente seleccionar el botón **Crear** (ver indicador 2) como se observa en la ilustración 4.6

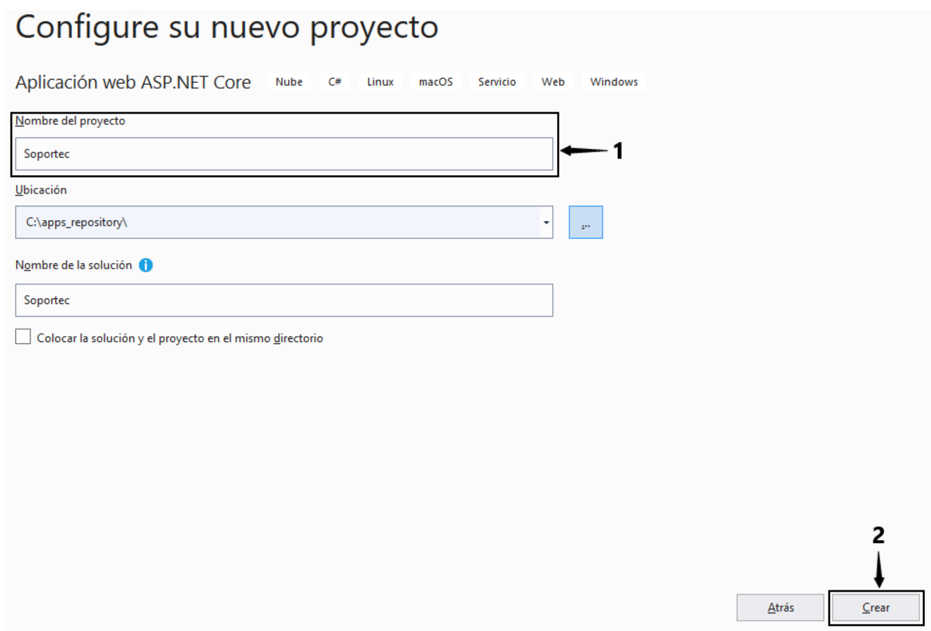


Ilustración 4.6 Ventana para proporcionar nombre y ubicación del proyecto.

Como último paso en la creación de un proyecto se selecciona la plataforma .NET Core y la versión del SDK con el que se quiere programar la aplicación (ver indicador 1). Visual Studio cuenta con una variedad de plantillas para desarrollar aplicaciones, para desarrollar la aplicación se selecciona la plantilla **API** (ver indicador 2) esta plantilla proporciona una configuración básica de controladores, modelos y vistas (ver ilustración 4.7), con esto al seleccionar el botón **Crear** (indicador 3), Visual Studio prepara el entorno de trabajo para el desarrollo de la plataforma.

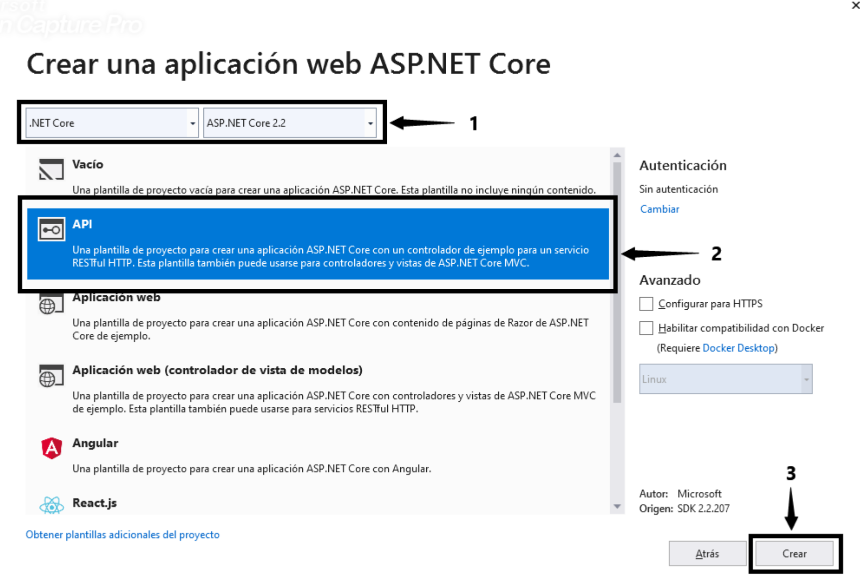


Ilustración 4.7 Ventana para la selección de la plantilla de la aplicación.

Posteriormente a la instalación del SDK para el desarrollo de la aplicación propuesta con ASP.NET Core apoyado del lenguaje C# y la creación del proyecto que contendrá la solución

disponible para comenzar a trabajar, se empieza a interpretar los modelos diseñados en el capítulo 3, es decir, se crean los modelos de datos (clases) apoyados por los diagramas de clases. Es en este capítulo donde todo se junta como una sola unidad y se comienzan a implementar todo lo planeado en los Sprints con ayuda de diagramas de casos de uso, diagramas de clase con las herramientas propuestas.

4.3 Instalación de Angular 9

Otra tecnología de la que también se hace uso para el desarrollo del proyecto y de la cual se habla en el capítulo 2 es Angular, que se utiliza para renderizar las vistas con la información almacenada en la base de datos PostgreSQL utilizada para persistir la información. Angular es el ***framework*** que se encargará de obtener la información de la lógica de negocios del lado del servidor y renderizarla en el navegador web. Antes de instalar angular se tiene que descargar el instalador de NodeJS que es el motor que utiliza angular para trabajar. El primer paso para trabajar con angular es acceder a la página oficial de NodeJS **https://nodejs.org/es/download/** (ver ilustración 4.8).

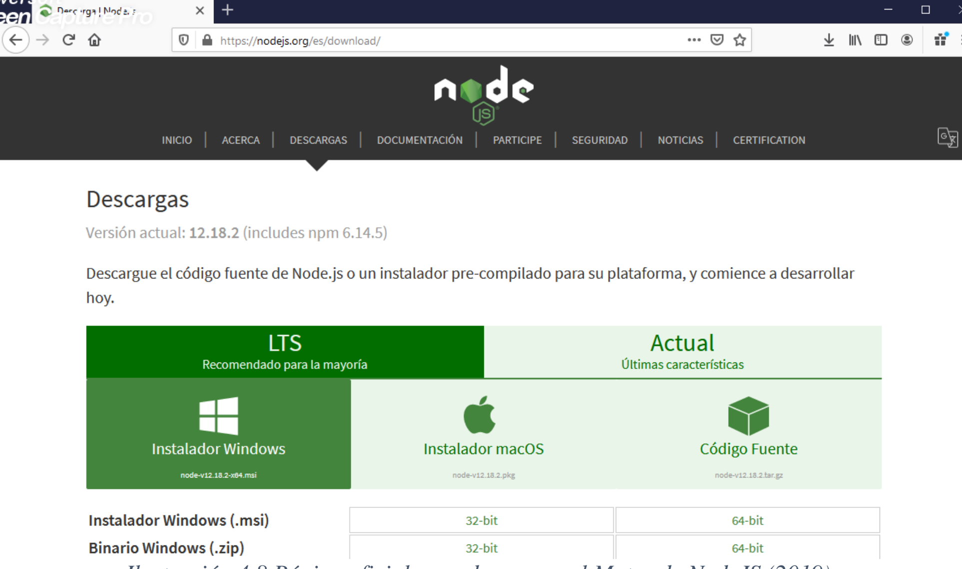
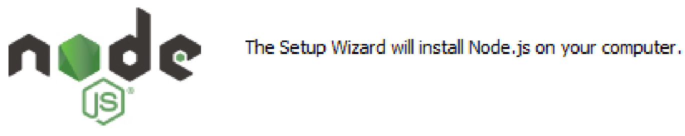


Ilustración 4.8 Página oficial para descargar el Motor de NodeJS (2019).

En la ilustración 4.8 se observan las opciones de descarga, para el desarrollo de la aplicación se descargará la versión de Windows para arquitectura de 64bits la versión **12.18.2LTS** que es la versión más estable hasta el momento de la descarga del motor de NodeJS, es importante mencionar que para la puesta en producción de la aplicación se descarga la versión estable hasta ese momento de puesta en producción.



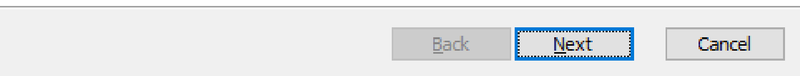


Ilustración 4.9 Asistente de instalación de NodeJs.

En la ilustración 4.9 se observa que para iniciar el proceso de instalación de NodeJS se selecciona el botón Siguiente (Next) y el proceso comenzará.

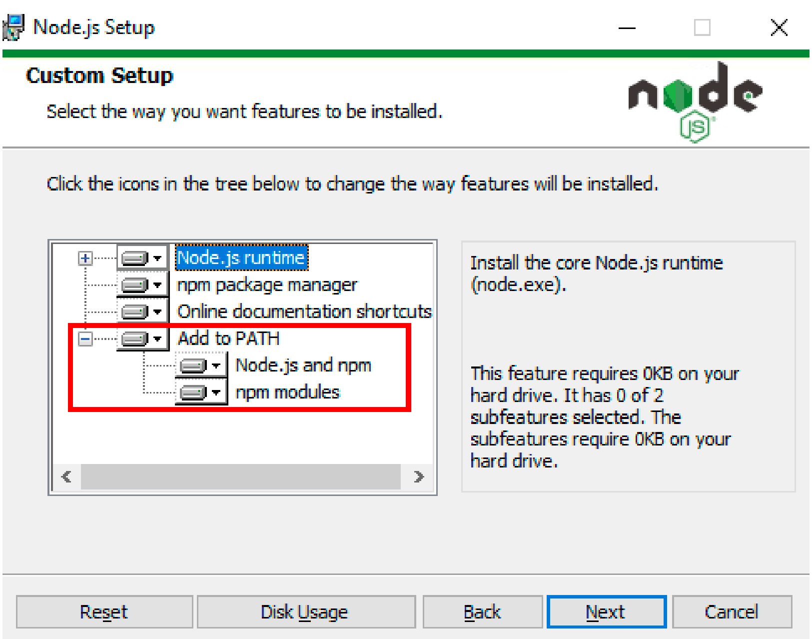


Ilustración 4.10 Opciones se instalación de NodeJS (2019).

En la ilustración 4.10 se observa que el asistente de instalación muestra las posibles configuraciones que se pueden realizar al instalar el motor de NodeJS, en este caso se seleccionará la opción **Node.js** y **npm** y módulos de npm. Cuando sea instalado NodeJS para la puesta en producción de la aplicación se instalará solo el Node.js runtime debido a que solo se necesitaran el núcleo del motor y no las librerías de compilación y depuración del motor.

En la ilustración 4.11 se observa que está siendo instalado, el proceso no tarda más de diez minutos.

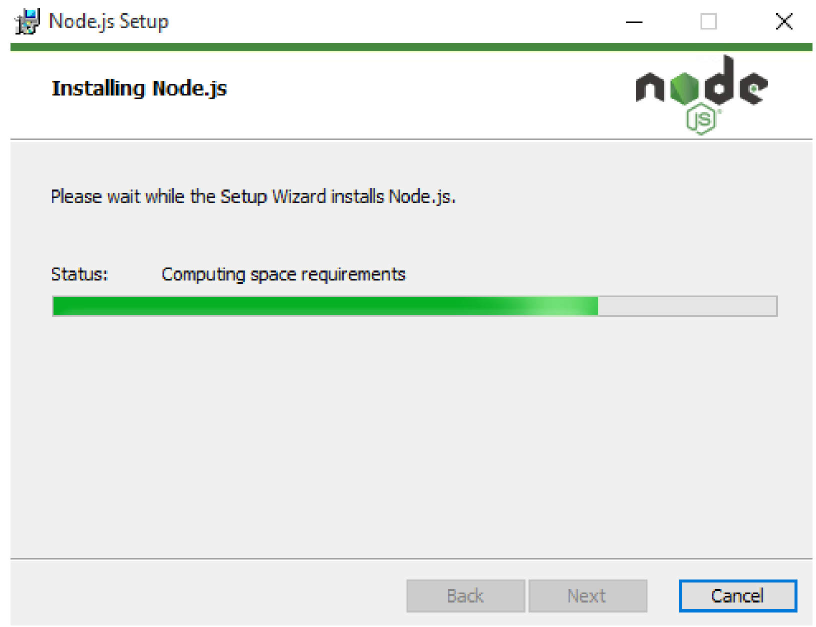


Ilustración 4.11 Instalación de NodeJS en proceso.

Al terminar la instalación de NodeJS como se observa en la ilustración 4.12, aún queda un paso por realizar y es comprobar que se instaló correctamente.

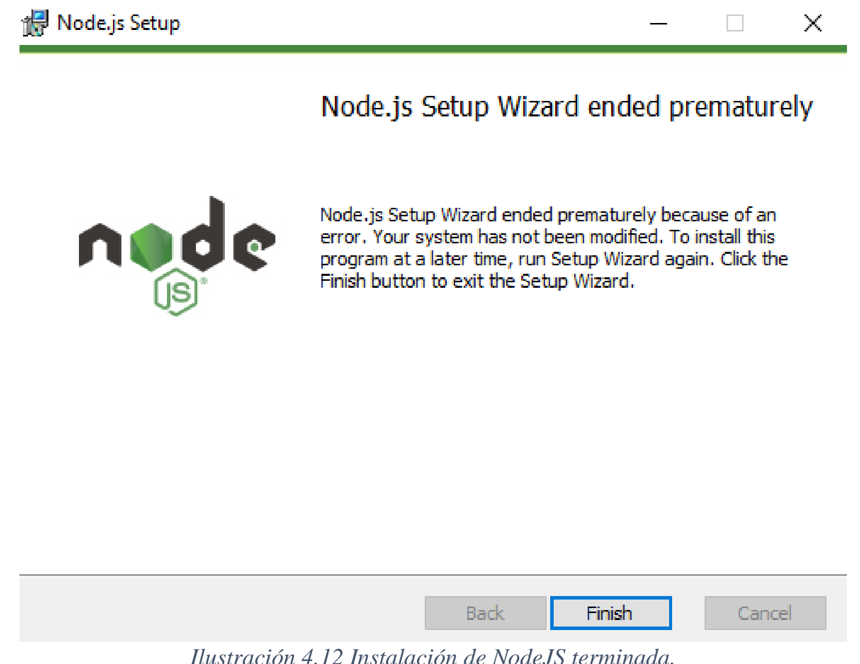


Ilustración 4.12 Instalación de NodeJS terminada.

Para comprobar que realmente se instaló el motor sin problemas, se tiene que abrir una ventana de comando de Windows (CMD) y teclear el comando **node -v** para verificar que versión fue instalada. Si la ventana de comando te muestra un mensaje que no reconoce el

comando es debido a que no se instaló adecuadamente. Después de que se comprobó que la instalación se realizó con éxito el siguiente paso a realizar es instalar el framework de Angular, debido a que NodeJS solo es el motor (núcleo) sobre el cual se ejecuta Angular. Lo que se tiene que hacer es abrir una ventana CMD de Windows y teclear los comandos que se muestran en la ilustración 4.13.

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\rogelio>npm install -g @angular/cli
```

Ilustración 4.13 Comando para la instalación de Angular CLI.

En el momento que se presiona la tecla enter para comenzar a instalar los paquetes la ventana de comando va indicando el porcentaje de instalación como se observa en la ilustración 4.14.

```
npm
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\rogelio>npm install -g @angular/cli
[█................] | fetchMetadata: sill resolveWithNewModule socks@2.3.3 checking installable status
```

Ilustración 4.14 Instalando Angular CLI.

Al finalizar la instalación se muestra el mensaje que se observa en la ilustración 4.15, incluso muestra un mensaje que indica que se detecta una nueva versión del gestor de paquetes de NodeJS, si se quiere actualizar solo se debe introducir el siguiente comando en la consola **npm install -g**.

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\rogelio>npm install -g @angular/cli
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
C:\Users\rogelio\AppData\Roaming\npm\ng -> C:\Users\rogelio\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng

> @angular/cli@10.0.2 postinstall C:\Users\rogelio\AppData\Roaming\npm\node_modules\@angular\cli
> node ./bin/postinstall/script.js

(node:12016) ExperimentalWarning: Conditional exports is an experimental feature. This feature could change at any time
+ @angular/cli@10.0.2
added 29 packages from 16 contributors, removed 9 packages, updated 64 packages and moved 1 package in 88.257s

      New minor version of npm available! 6.13.6 -> 6.14.6
   Changelog: https://github.com/npm/cli/releases/tag/v6.14.6
            Run npm install -g npm to update!
```

Ilustración 4.15 Proceso de instalación de Angular CLI terminado.

A continuación, se describe el proceso para la creación de un proyecto en Angular, proyecto que se utilizará como solución de la aplicación propuesta de la cual trata este tema de tesis, se debe recordar que Angular es muy independiente de ASP.NET Core 2, angular será una aplicación independiente que realizará peticiones al **API-REST** (ASP.NET Core) donde la API-REST responderá enviándole información al cliente que la solicite,

ANGULAR tendrá el trabajo de renderizar páginas web con información recibida de esas peticiones.

Para crear un proyecto en angular se debe abrir la terminal de comando de Windows (CMD), introducir el comando que se observa en la ilustración 4.16.

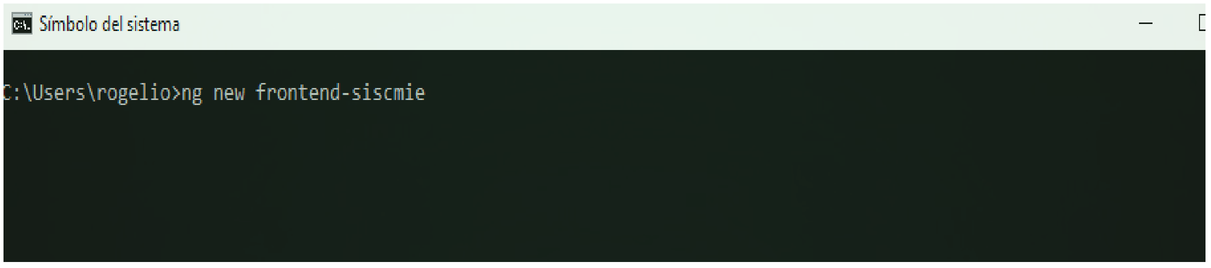


Ilustración 4.16 Creando proyecto en Angular CLI.

La palabra **ng new** es propia de la librería de Angular que indica crear un proyecto, la palabra **frontend-siscmie** es el nombre de la carpeta que contiene al proyecto. Una vez que se hace enter el proyecto se comienza a crear, la creación de un proyecto en Angular en promedio tarda 2 minutos debido a la cantidad de librerías que tiene que instalar.

Posteriormente a la instalación del SDK de ASP.NET Core, motor de NodeJS y el framework Angular CLI con sus debidos proyectos cada uno creado se procede a realizar el trabajo de codificación de la capa de acceso a datos y los módulos de los que se hablaron en el análisis y diseño del capítulo 3. A continuación se comenzará con el primer Sprint planeado.

4.4 Sprint 1: Implementación de la capa de acceso a datos

En la tabla 4.4 se observa la planeación del Sprint 1, en el presente Sprint no se realizará ninguna historia de usuario proporcionada por el **product owner** presentadas en el capítulo 3. En este Sprint se describirá la creación de la capa de acceso a datos para la gestión y manipulación de los datos de la aplicación propuesta. Los servicios proporcionados por esta capa serán utilizados durante el desarrollo de los módulos que abarcan las historias de usuario de la aplicación, debido a que todos los módulos que se implementarán de acuerdo con el ciclo de vida de cada petición que se realice a la aplicación, la capa de acceso a datos es con la que interactúa en todo momento la aplicación, pues facilita el trabajo de manipular la información.

Tabla 4.4 Planeación del Sprint 1.

Sprint 1		
Objetivo: El objetivo del Sprint es que los programadores puedan acceder fácilmente a la información de la base de datos haciendo uso del patrón repositorio genérico y unidad de trabajo.		
Fecha de inicio	Fecha de finalización	N° de semanas
01-05-2019	31-05-2019	4

Tareas a realizar en el Sprint 1	Fecha de inicio	Fecha de finalización	Horas de trabajo invertidas
Creación de los modelos de dominio (clases).	01-05-2019	15-05-2019	60
Creación del contexto de datos.	16-05-2019	20-05-2019	28
Implementación del patrón de repositorio genérico (IRepository y Repository).	21-05-2019	26-05-2019	25
Implementación de la unidad de trabajo (UnitOfWork).	27-05-2019	31-05-2019	15
Total de horas de trabajo invertidas			128

La capa de acceso a datos es de suma importancia en el desarrollo del proyecto, ya que en dicha capa se encuentran los modelos de dominio que en la arquitectura MVC vienen siendo los modelos, dichos modelos para el desarrollo del presente proyecto fueron creados en la sección de análisis y diseño del capítulo 3. Los modelos son utilizados para representar la información en tiempo de ejecución de la aplicación, es decir, cuando la información se persiste se encuentra en una base de datos almacenada en un disco duro, pero cuando la aplicación se encuentra en ejecución y esta necesita cargar información en memoria, toda la información debe conservar su estructura y es ahí en donde entran los modelos que permiten hacer uso de la programación orientada a objeto y almacenar esa información en la memoria de la computadora sin perder su estructura, haciendo una comparación más comprensible, la información almacenada en una tabla de la base de datos corresponden a modelos. El desarrollo clásico de aplicaciones dice que primero se debe de crear la base de datos y posteriormente a eso se deben de crear las clases. Con **Entity Framework** no sucede así gracias al enfoque llamado **CodeFirst** como se mencionó en el capítulo 3, sucede al revés primero se crean las clases y haciendo uso de comando se realiza un proceso llamado migración que consiste en que los modelos son convertidos a instrucciones SQL y dichas instrucciones SQL son ejecutadas en el manejador de base de datos elegido, en este caso **PostgreSQL**. Las migraciones son una funcionalidad muy novedosa con la que ASP.NET Core cuenta, se pueden actualizar los modelos y actualizar la migración las veces que sean necesarias, es decir, si en algún momento la relación de los modelos cambia, se hace una nueva migración y la relación de la base de datos también se actualiza.

Antes de continuar se debe hacer mención que para utilizar el enfoque **CodeFirst** se deben instalar librerías propias de Microsoft en el proyecto que fue creado con Visual Studio (***Soportec***) en la sección anterior (***Instalación de .NET Core SDK 2.2***).

4.4.1 Instalación de los paquetes en el proyecto Soportec

Los paquetes que fueron instalados en el proyecto ***Soportec*** para la implementación de la capa de acceso a datos se describen en la tabla 4.5.

Tabla 4.5 Paquetes necesarios para la implementación de la Capa de Acceso a datos (DAL).

Nombre del paquete	Uso	Versión
Npgsql.EntityFrameworkCore.PostgreSQL	Este paquete proporciona librerías que facilitan la migración de los modelos a la base de datos relacional. Como su nombre lo indica contiene librerías de **EntityFrameworkCore.**	2.2.4
Npgsql	Este paquete contiene librerías que ayudan a convertir los modelos a instrucciones SQL específicamente del sistema gestor de base de datos **PostgreSQL.**	2.2.4
Microsoft.EntityFrameworkCore. Tools	Este paquete permite interpretar el comando **add-migration** y **update-database** que son los que Visual Studio interpreta y ejecuta la migración.	3.0.2

A continuación, se describe el proceso de instalación de un paquete en la solución (***Soportec.Api***) (ver indicador 1) del proyecto en Visual Studio. Para instalar la librería a un proyecto, es necesario hacer clic derecho en la instancia del proyecto (ver indicador 2) y después aparecerá un cuadro de opciones, donde se tiene que elegir la opción **Administrador de paquetes NuGet** (ver indicador 3) (ver figura 4.17).

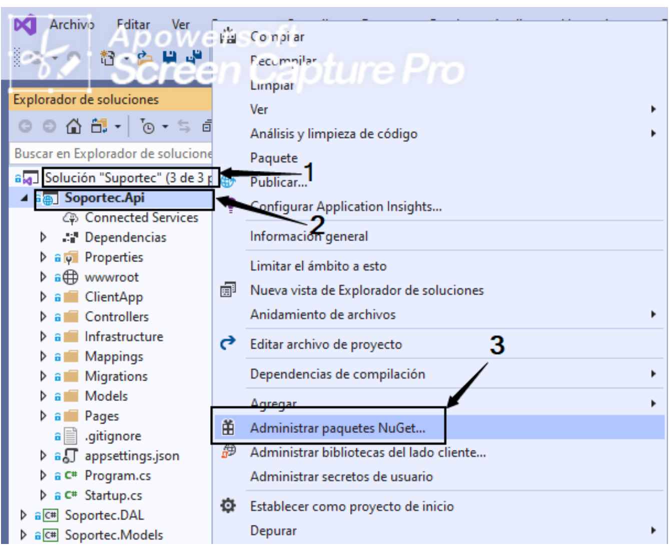


Ilustración 4.17 Proceso de instalación de librerías a un proyecto.

Después de seleccionar la opción **Administrador de paquetes NuGet,** aparecerá una ventana que permite buscar las librerías que se requieren instalar como se muestra en la ilustración 4.18. Lo primero que se tiene que hacer es buscar la librería deseada (ver indicador

1), si el administrador de paquetes encuentra la librería buscada (***Npgsql***), al seleccionar la librería (ver indicador 2) en la parte derecha de la ventana de ***NuGets*** se muestra otro formulario que muestra el nombre completo de la librería (ver indicador 3) y una lista de versiones con las que cuenta el paquete a instalar (ver indicador 4) y por último se hace clic en el botón **Instalar** (ver indicador 5).

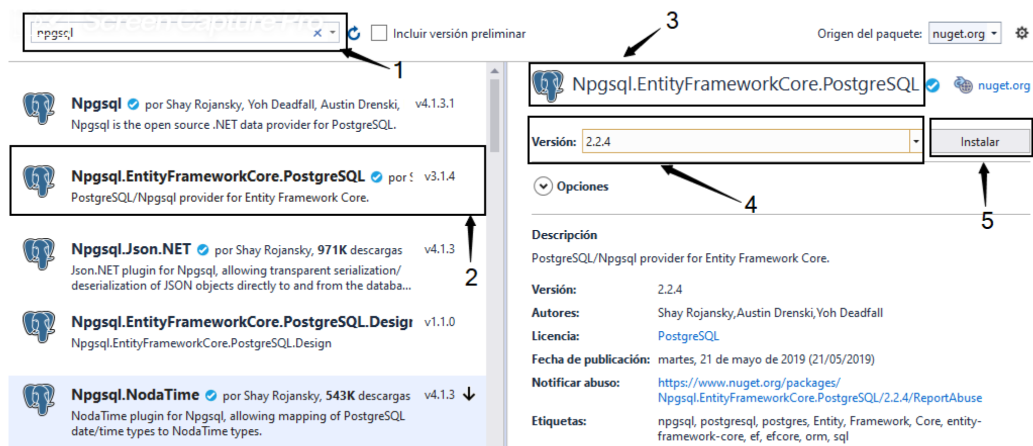


Ilustración 4.18 Administrador de paquetes NuGet (Visual Studio 2019).

Posterior al haber seleccionado el botón **Instalar**, aparecerá una ventana (ver figura 4.19) donde está indicando las librerías que serán instaladas (ve indicando 1). Al seleccionar el botón **Aceptar** (ver indicador 2) se aceptar los términos de uso de las librerías.

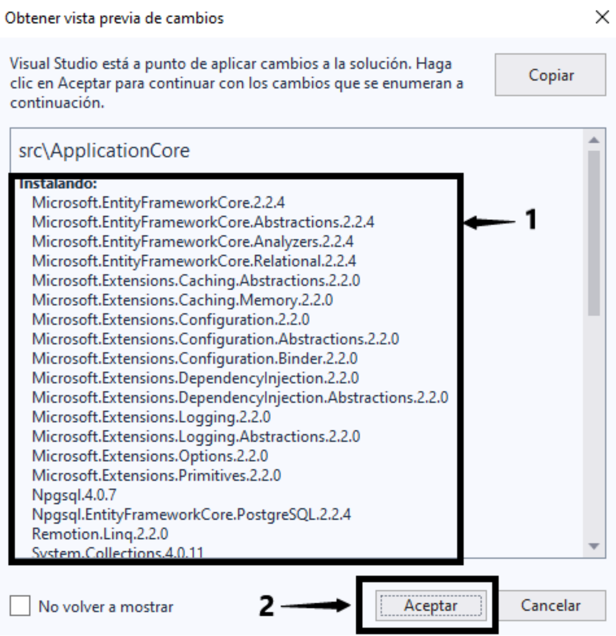


Ilustración 4.19 Ventana de confirmación para instalar librerías.

Este mismo proceso se lleva a cabo para casa una de las librerías que son instaladas, ya instalados las librerías el proyecto ***Soportec*** queda listo para el desarrollo de los módulos.

Para persistir y consultar información de la base de datos utilizada en el desarrollo del proyecto, es indispensable de acuerdo con lo propuesto en el capítulo 3, la creación de una capa que abstraiga la manipulación de los datos almacenados en la base de datos.

4.4.2 Creación de la capa de acceso a datos

Las buenas prácticas de programación dicen que en el desarrollo de una aplicación se deben de separar las responsabilidades según la naturaleza de estas, por lo tanto, para la creación de la capa de acceso a datos se crea un nuevo proyecto de tipo librería **.net core** dentro de la solución ***Soportec***, que a través de referencias entre proyecto se podrá acceder a la librería creada. Para agregar una instancia a la solución es necesario hacer clic derecho (ver indicador 1), entonces se mostrará un cuadro de opciones en el cual se selecciona la opción **Agregar** (ver indicador 2), posteriormente **Nuevo proyecto** (ver indicador 3) (ver ilustración 4.20).

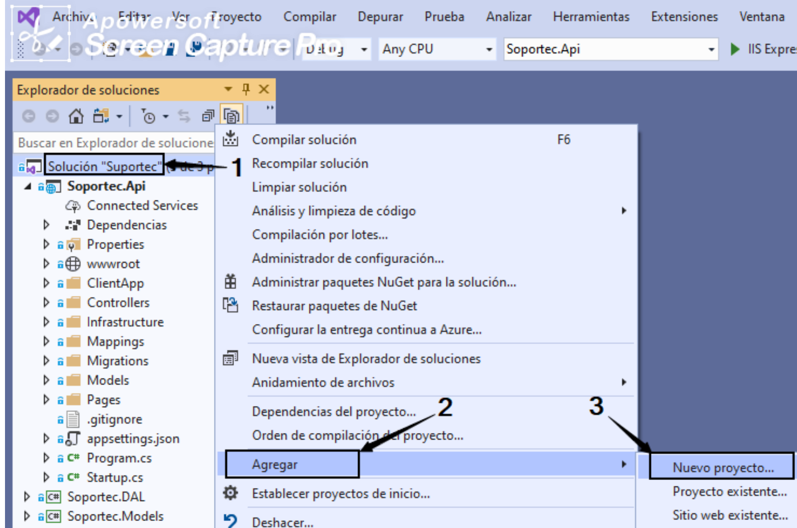


Ilustración 4.20 Proceso para agregar un nuevo proyecto a la solución.

En el momento de seleccionar la opción **Nuevo proyecto**, aparecerá una ventana que proporciona diferentes tipos de proyectos y librerías. Para la creación de la capa de acceso a datos es necesario crear una **biblioteca de clases .NET Core** (ver indicador 1) (ver ilustración 4.21) el proceso es similar para la creación de un proyecto como se explicó anteriormente, cuando se creó la instancia del proyecto ***Soportec.Api***.

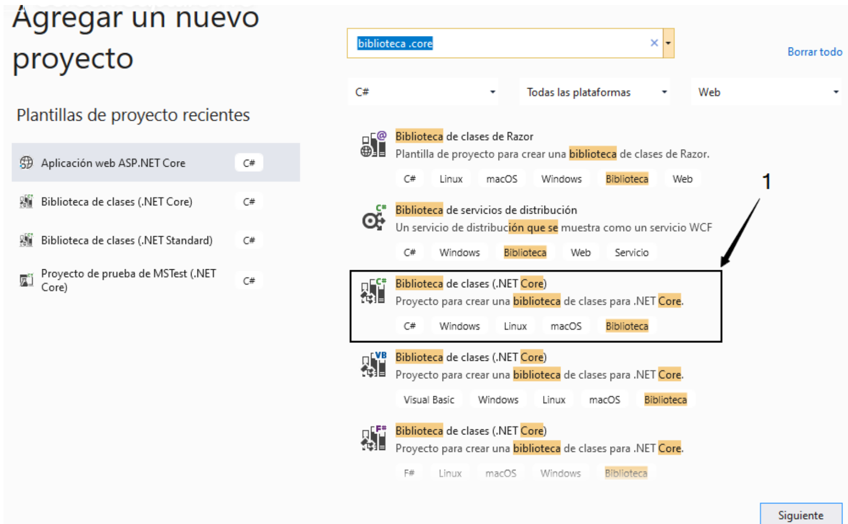


Ilustración 4.21 Asistente de Visual Studio para crear un nuevo proyecto.

El proyecto que contendrá la lógica de la capa de acceso a datos fue nombrado como ***Soportec.DAL*** (ver indicador 1) se muestra en la ilustración 4.22.

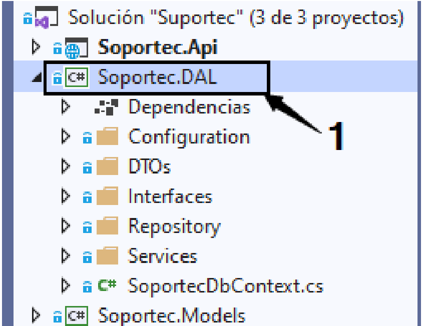


Ilustración 4.22 Solución que contiene los proyectos Soportec.DAL

En el proyecto ***Soportec.DAL*** se crearon las clases que representa el modelo de dominio que fueron presentadas en el capítulo 3 en la sección de diseño. Como se puede observar en la ilustración 4.23 se encuentran todas las clases que conforman la aplicación a desarrollar (ver indicador 1) en la parte derecha de la imagen se observa una clase llamada ***Servicio*** que contiene dos propiedades (ver indicador 2), donde el primero es el atributo que identifica como único un registro y el segundo atributo el nombre, que traducido a un contexto de base de datos, los atributo se les llama campos y a la clase tabla, lo mismo sucede para todas las demás clases.

```
Solución "Soportec" (3 de 3 proyectos)
  Soportec.Api
  Soportec.DAL
  Soportec.Models
    Dependencias
    Enum
    Area.cs
    Plan.cs
    Problema.cs
    Proveedor.cs
    Revision.cs
    Servicio.cs
    Ticket.cs
    Usuario.cs
```

```
3  namespace Soportec.Models
4  {
5      public class Servicio
6      {
7          public int Id { get; set; }
8          public string Descripcion { get; set; }
9          public bool Estado { get; set; }
10
11         public ICollection<Ticket> Tickets { get; set; }
12     }
13 }
```

Ilustración 4.23 Clases de modelo de dominio de la aplicación.

Haciendo uso del lenguaje C# se crean los modelos de datos para posteriormente crear la base de datos, y para realizar esa tarea una clase tiene que ser representada como una tabla y sus atributos como campos. En la ilustración 4.24 se muestra la configuración de la clase ***Servicio***, donde la clase es pasada como parámetro en un método (ver indicador 1) para acceder a sus propiedades, posteriormente se le asigna un nombre (ver indicador 2) que es como será llamada la clase en la base de datos y como toda tabla debe de tener un campo primario para asegurar la integridad de cada registro es por eso que se configura (ver indicador 3) como ***HasKey***, por último se le dice que el mismo campo debe de ser único y no debe repetirse y eso configura usando la palabra reservada ***IsRequired*** (ver indicador 4).

```
6  {
7      public class ServicioConfig
8      {
9          public static void SetEntityBuilder(EntityTypeBuilder<Servicio> entityBuilder)
10         {
11             entityBuilder.ToTable("tab_servicios");
12
13             entityBuilder.HasKey(x => x.servicioId);
14             entityBuilder.Property(x => x.servicioId).IsRequired();
15         }
16     }
17 }
```

Ilustración 4.24 Configuración de una clase para traducirla a tabla en la base de datos.

Para hacer uso de cada una de las palabras reservadas descritas previamente se deben llamar las librerías instaladas anteriormente (**sección de instalación de paquetes NuGet**) como muestra la ilustración 4.25.

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Support.Api.DataAccess.Core.Entities;
```

Ilustración 4.25 Referencia de librerías y clases.

Existen modelos de datos que se relacionan uno con otros, tal es el caso de la clase ***Usuario***, que según a los modelos diseñado en el capítulo 3 tiene una relación con la clase ***Rol***. En la ilustración 4.26 se muestra la clase de configuración de la clase usuario que su tabla en la base de datos será llamada ***tab_usuarios*** (ver indicador 1), su campo primario se llamará ***usuarioId*** (ver indicador 2) y será único, es decir, el valor de dicho campo no se debe de repetir en los registros de los otros campos que contenga.

```
7    public class UsuarioConfig
8    {
         1 referencia | 0 excepciones
9        public static void SetEntityBuilder(EntityTypeBuilder<Usuario> entityBuilder)
10       {
11           entityBuilder.ToTable("tab_usuarios");
12
13           entityBuilder.HasKey(x => x.usuarioId);
14           entityBuilder.Property(x => x.usuarioId).IsRequired();
15
16           entityBuilder.HasOne(x => x.rol).WithMany(x => x.usuarios).HasForeignKey(x => x.rolId);
17           //entityBuilder.HasOne(x => x.Departamento).WithMany(x => x.Usuario).HasForeignKey(x =>
18       }
19   }
```

Ilustración 4.26 Configuración de la relación entre clases.

Según la configuración mostrada en la ilustración 4.26 un ***Usuario*** puede tener un rol y a su vez un rol le puede pertenecer a muchos usuarios y la llave foránea de esa relación será el campo ***rolId*** que se encuentra en la clase ***Usuario***.

La configuración explicada anteriormente se realiza para cada clase y relación que exista entre cada una de las clases de acuerdo con el diagrama de clases diseñado en la sección de análisis y diseño del capítulo 3.

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Soportec.DAL.Configuration;
using Soportec.Models;

namespace Soportec.DAL
{
    25 referencias
    public class SoportecDbContext : IdentityDbContext<Usuario>
    {
        0 referencias
        public SoportecDbContext(DbContextOptions<SoportecDbContext> options)
            : base(options) { }
        0 referencias
        protected override void OnModelCreating(ModelBuilder mb)
        {
            mb.ApplyConfiguration(new AreaConfig());
            mb.ApplyConfiguration(new TicketConfig());
            mb.ApplyConfiguration(new PlanConfig());
            mb.ApplyConfiguration(new ProblemaConfig());
            mb.ApplyConfiguration(new ProveedorConfig());
            mb.ApplyConfiguration(new ServicioConfig());
            mb.ApplyConfiguration(new RevisionConfig());

            base.OnModelCreating(mb);
        }
```

Ilustración 4.27 Configuración de la clase contexto de datos.

Después de configurar y relacionar todas las clases de la aplicación el siguiente paso es mandar a llamar las configuraciones en la clase ***SoportecDBContext*** (*clase que se comunica directa con la base datos*). El contexto de datos debe heredar de una clase de la librería ***EntityFramework*** llamada ***DbContext*** (ver indicador 1), debido a que esta clase contiene los métodos necesarios para migrar todas las clases a la base de datos. Dentro de la clase SoportecDbContext se declara un método llamado ***OnModelCreating*** (ver indicador 2) este método cuando se ejecuta la migración es el primero que es llamado para identificar las configuraciones que se les realizaron a cada una de las clase (ver indicador 3) y que se deben

de migrar, por último como se observa en la ilustración 4.27 (ver indicador 4) haciendo uso de la herencia se llama al método base de la clase ***DbContext*** y manda las configuraciones para que sean ejecutadas mientras se ejecuta la migración.

```
2 referencias
public DbSet<Ticket> Tickets { get; set; }
1 referencia
public DbSet<Revision> Revisiones { get; set; }
0 referencias
public DbSet<Area> Ubicaciones { get; set; }
0 referencias
public DbSet<Problema> Problemas { get; set; }
0 referencias
public DbSet<Proveedor> Proveedores { get; set; }
0 referencias
public DbSet<Servicio> Servicios { get; set; }
1 referencia
public DbSet<Plan> Planes { get; set; }
1 referencia
public DbSet<Usuario> Usuarios { get; set; }
```

Ilustración 4.28 Propiedades que serán convertidas a tablas en la base de datos.

En la ilustración 4.28 se muestran en ***SoportecDBContext*** algunas de las propiedades con el tipo de clase (ver indicador 1) que serán migradas y convertidas a tablas en la base de datos ***PostgreSQL***.

Antes de lograr hacer una migración se debe de configurar la cadena de conexión a la base de datos (ver indicador 1) que se encuentra en el archivo ***appsettings.json***, para comunicar con el proveedor de conexión de datos ***Npgsql*** el cual es un proveedor para el acceso de datos a la base de datos ***PostgreSQL***. En la ilustración 4.29 se puede observar dos cadenas de conexión (ver indicador 2) a la base de datos, en la cadena llamada ***DataBaseDesarrollo*** es la cadena que se utiliza para el servidor de base de datos utilizado para el desarrollo de la aplicación y la cadena llamada ***DataBaseProduccion*** es utilizada para probar la puesta en producción de la aplicación en ***Azure Microsoft***.

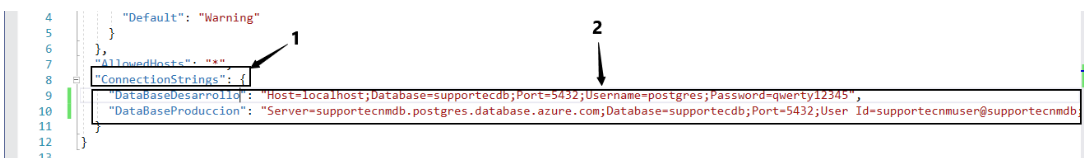


Ilustración 4.29 Cadena de conexión a la base de datos de la aplicación.

Posteriormente a la configuración a la cadena de conexión a la base de datos, en el archivo ***Startup*** del proyecto ***Soportec.Api*** se configura el servicio para que cada vez que se realice una petición a la aplicación se creé la conexión a la base de datos. En la ilustración 4.30 se observa que haciendo uso del contexto de datos (ver indicador 1) se le indica la cadena a la base de datos que le corresponde (ver indicador 2).

```
// Connection Strings
services.AddDbContext<SoportecDbContext>(options =>
    options.UseNpgsql(Configuration.GetConnectionString("DataBaseDesarrollo"),
        assembly => assembly.MigrationsAssembly("Soportec.Api"))
);
```

Ilustración 4.30 Configuración de la cadena de conexión en Startup.

Lo interesante de la capa de acceso a datos se encuentra en la interfaz ***IRepository***, en la clase ***Repository*** y la clase ***UnitOfWork*** (*Unidad de trabajo*). Como se mencionó en el capítulo 3 las clases que no tenían métodos para realiza tareas como agregar, actualizar, consultar y eliminar era debido a que se hacía uso de patrones de diseño que facilitaban la tarea y hacían eficiente el acceso los datos, haciendo uso de la interfaz ***IRepository,*** donde dentro de dicha interfaz (ver indicador 1) se encuentran las firmas (*métodos*) comunes que toda clase como mínimo debe de tener. En la ilustración 4.30 se muestra la interfaz y los métodos, para el desarrollo de la aplicación se utilizan los métodos más comunes que son ***Crear, Actualizar, Consultar y Eliminar***, sin embargo, se implementaron más de veinte (ver indicador 2), esto con la finalidad de hacer pruebas para el desarrollo de futuros módulos de la aplicación.

```
{
    public interface IRepository<TEntity> where TEntity : class
    {
        IQueryable<TEntity> Query();
        TEntity Add(TEntity entity);
        Task<TEntity> AddAsync(TEntity entity);
        int Count();
        Task<int> CountAsync();
        void Delete(TEntity entity);
        Task<int> DeleteAsync(TEntity entity);
        void Dispose();
        TEntity Find(Expression<Func<TEntity, bool>> match);
        ICollection<TEntity> FindAll(Expression<Func<TEntity, bool>> match);
        Task<ICollection<TEntity>> FindAllAsync(Expression<Func<TEntity, bool>> match);
        Task<TEntity> FindAsync(Expression<Func<TEntity, bool>> match);
        IQueryable<TEntity> FindBy(Expression<Func<TEntity, bool>> predicate);
        Task<ICollection<TEntity>> FindByAsync(Expression<Func<TEntity, bool>> predicate);
        Task<TEntity> Get(object id);
        IQueryable<TEntity> GetAll();
        Task<ICollection<TEntity>> GetAllPaged(int count, int page);
        Task<ICollection<TEntity>> GetAllAsync();
        IQueryable<TEntity> GetAllIncluding(params Expression<Func<TEntity, object>>[] includeProperties);
```

Ilustración 4.31 Interfaz IRepository para el acceso a datos.

En la interfaz ***IRepository*** (ver indicador 1) (ilustración 4.31) solo se declaran métodos sin cuerpo, es decir, no se implementa el código. Por lo que en la ilustración 4.31 se muestra la implementación de la clase ***Repository*** que hereda de la interfaz ***IRepository*** (ver indicador 2), es decir, fuerza a tener que implementar cada uno de los métodos heredados tal es el caso del método ***Query*** (ver indicador 3) que se encarga de extraer de la base de datos todos los registro de una tabla y convertirlos en una lista, o el método ***GetAll*** (ver indicador 4) que también extrae todos los datos de una tabla pero sin ser convertidos en una lista, parecieran tener la misma funcionalidad pero en realidad tiene diferentes usos.

```
public class Repository<TEntity> : IRepository<TEntity> where TEntity : class
{
    private readonly ISupportDBContext _context;

    public Repository(ISupportDBContext context)
    {
        this._context = context;
    }

    public IQueryable<TEntity> Query()
    {
        return _context.Set<TEntity>().AsQueryable();
    }

    public IQueryable<TEntity> GetAll()
    {
        return _context.Set<TEntity>();
    }
```

Ilustración 4.32 Implementación de la clase Repository.

Después de la implementación de la clase ***Repository***, se implementa la clase ***UnitOfWork*** (ver ilustración 4.32) que se expone directamente cada vez que un ***Controlador*** hace una petición a la base de datos de acuerdo con la **arquitectura MVC**. Se puede observar en la ilustración 4.32 que para hacer uso de las clases y contexto de datos se debe de llamar antes a los espacios (ver indicador 2) de nombre donde se encuentra cada una de las clases e interfaces que se implementan en la clase (ver indicador 3).

```
using Support.Api.DataAccess.Core.Entities;
using Support.Api.DataAccess.Core.Interfaces;
using Support.Api.DataAccess.Core.Repository;
using System;

namespace Support.Api.DataAccess.Core
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly SupportDBContext _context;

        private IRepository<Producto> _productos { get; set; }
        private IRepository<Documento> _documentos { get; set; }
        private IRepository<Departamento> _departamentos { get; set; }
        private IRepository<SolicitudDetalle> _solicitudDetalles { get; set; }
        private IRepository<Problema> _problemas { get; set; }
        private IRepository<Proveedor> _proveedores { get; set; }
        private IRepository<Servicio> _servicios { get; set; }
        private IRepository<Usuario> _usuarios { get; set; }
```

Ilustración 4.33 implementación de las propiedades de UnitOfWork (Unidad de trabajo).

Dentro de la misma clase ***UnitOfWork*** se declara un constructor que permite hacer el llamado por ***Inyección de dependencia*** (ver indicador 1) para que le sea otorgado un objeto del tipo ***SoportecDBContext*** para manipular cada uno de los método implementados en ***Repository*** (ver indicador 2), como se observa (indicador 5) el objeto obtenido a través del constructor se le pasa al objeto de tipo ***Repository*** (ver indicador 4) para crear una instancia y asignársela al objeto ***_servicios*** (ver indicador 3), una vez realizado este proceso ***UnitOfWork*** podrá hacer uso de todos los métodos declarados en la interface ***IRepository*** e implementados en la clase ***Repository***.

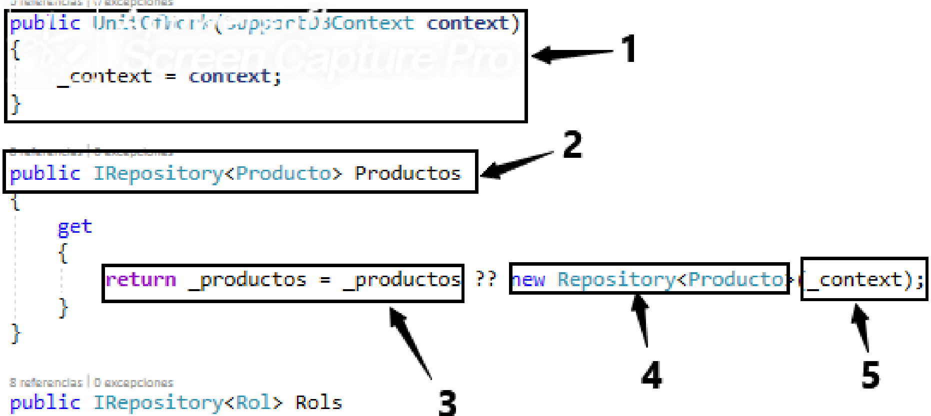


Ilustración 4.34 Implementación de los métodos de UnitOfWork.

Una vez que toda la configuración de clases esta realizada se procede hacer la migración desde la consola del ***Administrador de paquetes NuGet*** (ver indicador 4), donde se tiene que elegir la ubicación de donde se encuentra la capa que tiene la clase ***SoportecDBContext*** (ver indicador 1), después se procede a teclear el comando **add-migration** (ver indicador 2) y posteriormente a darle un nombre (ver indicador 3) a dicha migración para identificarla de alguna manera por si en el futuro se desea borrar una migración en específico.

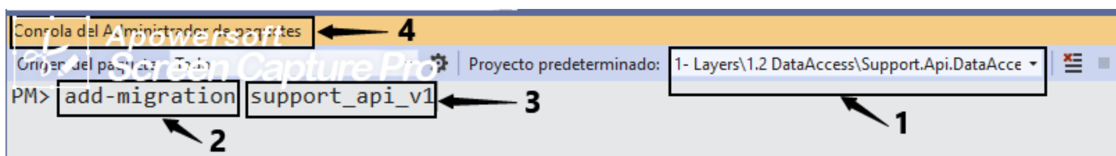


Ilustración 4.35 Migración de clases de la aplicación.

Después de ingresar los comandos necesarios, las librerías de ***Npgsql.EntityFrameworkCore.PostgreSQL*** comienzan a hacer una especie de traducción de lenguaje orientado a objeto a instrucciones ***SQL*** del manejador ***PostgreSQL***. Una vez que termina la traducción a lenguaje SQL se muestra un mensaje como el que se observa en la ilustración 4.36, que indica que la operación fue realizar y que puede posteriormente eliminarla.

```
PM> add-migration support_api
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.2.6-servicing-10079 initialized 'SupportDBContext' using provider 'Npgsql.EntityFrameworkCore.PostgreSQL'
with options: None
To undo this action, use Remove-Migration.
```

Ilustración 4.36 Traduciendo lenguaje orientado a objeto a instrucciones SQL.

Posteriormente el siguiente paso es migrar las instrucciones SQL al gestor de base de datos, esto se realiza introduciendo el comando ***update-database*** como se muestra en la ilustración 4.37 (ver indicador 1).

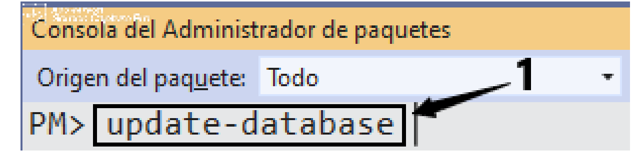


Ilustración 4.37 Instrucción para migrar las clases a una base de datos.

Después de la migración en el proyecto ***Soportec.DAL*** (ver indicador 1) se crea una carpeta con el nombre ***Migrations***, esta carpeta contiene cada una de las migraciones (ver indicador 2) que se realizan en el proyecto, la carpeta puede ser eliminada y el mismo proceso se puede repetir nuevamente sin ningún problema.

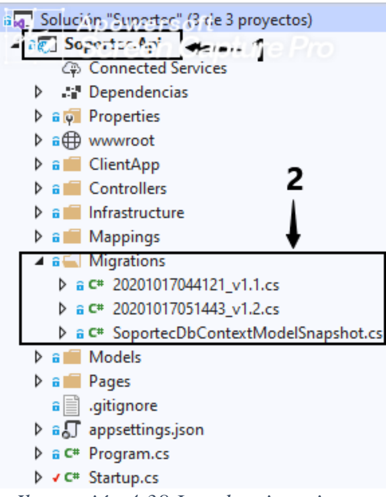


Ilustración 4.38 Log de migraciones.

En la ilustración 4.39 se muestra la comparación de las clases y sus respectivas tablas, del lado izquierdo de la imagen se muestras las tablas de la base de datos (ver indicador 1) y del lado derecho se muestran las clases que le corresponde a cada tabla (ver ilustración 2). A partir del momento en que se genera la migración y se verifica que las tablas fueron creadas en la base de datos el proyecto se encuentra listo para comenzar a hacer peticiones a la base de datos desde la clase ***UnitOfWork*** o ***IRepository***, lo único que se tiene que hacer es crear una instancia de dichas clases y acceder a las clases (tablas en la base de datos) utilizando la nomenclatura propiedad punto como comúnmente se hace en programación orientada a objetos, una vez que se crea el objeto de la unidad de trabajo se accede a la clase deseada y al método que se requiera ejecutar y sería todo lo que se tiene que hacer para realizar una operación en la base de datos.

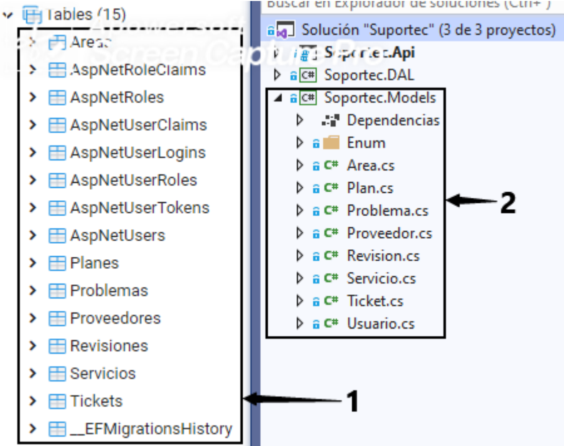


Ilustración 4.39 Comparación de las clases con las tablas en la base de daos después de la migración.

Es importante mencionar que, a partir de este punto, en futuras secciones ya no se hablará más de tablas de la base de datos, debido a que como ya se mencionó todas las operaciones que se realicen con la clase ***Repository*** automáticamente se reflejaran los cambios en la base de datos con la que ésta enlazada la aplicación y en la cual se llevaron a cabo las migraciones mencionadas en la sección anterior. En los Sprints que faltan por explicar solo se hará referencia al repositorio de la clase a la que se le requiera hacer un CRUD (***Creación***, ***Actualización***, ***Consulta*** o ***Eliminación***) de información debido a que la clase está relacionada con la tabla que le corresponde en el gestor de base de datos, es por eso que se decidió dedicarle este primer Sprint a la explicación de dicha capa, es decir, en las implementaciones de las módulos de catálogos, listas de verificaciones y planes de trabajo que se explicaran enseguida, solo se hablará de la implementación de los controladores que es donde se codifica la lógica del negocio de cada módulo y donde se hace uso de los repositorios de cada clase para acceder a la base de datos (ver capítulo 3, arquitectura de la aplicación).

4.5 Sprint 2: Implementación del módulo de usuarios

En la tabla 4.6 se presenta la planeación del Sprint 2, donde cada historia de usuario se desglosa en tareas que tiene un tiempo estimado para su realización, acordado por el equipo ***Scrum***, así como las fechas de inicio y finalización del Sprint.

Tabla 4.6 Planeación del Sprint 2.

<table>
<tr><th colspan="5">Sprint 2</th></tr>
<tr><td colspan="5">Objetivo: El objetivo del Sprint es que el administrador pueda registrar usuarios y los usuarios creados puedan ingresar en el sistema de acuerdo con el rol con el que fueron registrados.</td></tr>
<tr><td colspan="2">Fecha de inicio</td><td colspan="2">Fecha de finalización</td><td>N° de semanas</td></tr>
<tr><td colspan="2">01-06-2019</td><td colspan="2">30-06-2019</td><td>4</td></tr>
<tr><th>Historias de usuario</th><th>Tareas del Sprint 2</th><th>Fecha de inicio</th><th>Fecha de finalización</th><th>Horas de trabajo invertidas</th></tr>
<tr><td rowspan="3">HU1</td><td>Implementar el controlador para la gestión de usuarios de la aplicación.</td><td>01-06-2019</td><td>10-06-2019</td><td>40</td></tr>
<tr><td>Implementar la funcionalidad que valide si el usuario está registrado y activo para ingresar a la aplicación.</td><td>11-06-2019</td><td>20-06-2019</td><td>42</td></tr>
<tr><td>Implementar la vista para que el administrador pueda registrar a los usuarios en la aplicación.</td><td>21-06-2019</td><td>30-07-2019</td><td>125</td></tr>
<tr><td colspan="4">Total de horas de trabajo invertidas</td><td>207</td></tr>
</table>

Cada una de las tareas realizadas en la historia de usuario serán presentadas en dos fases **Back-End** y **Front-End**.

4.5.1 Back-End: Implementación del controlador para la gestión de usuarios de la aplicación

En el capítulo 2 se mencionó la arquitectura utilizada para el desarrollo de la aplicación, así como las tecnologías para desarrollar el proyecto, la cual es ASP.NET MVC Core 2, en donde el ciclo de vida de una petición realizada por un cliente comienza en los controladores que son los componentes que almacenan la lógica de negocio. Lo primero que se realizó después de la migración de las clases a tablas en la base de datos (ver Sprint 1), se creó un controlador llamado ***UsuarioController*** (ver ilustración 4.40) el cual es un componente que es accedido usando la ruta **http://localhost:5000/api/usuario**, esta es la ruta en un ambiente de desarrollo, ya que en producción ***localhost:5000*** será sustituido por el dominio donde se encuentre la aplicación seguida por la ruta del controlador.

La creación de usuario se realiza desde una aplicación Angular, es decir, la información será envidada desde una aplicación Back-End expuesta como una api-rest (ver indicador 1 en la ilustración 4.40), el controlador (ver indicador 2) hereda de la clase ***Controller*** de los paquetes del núcleo de ASP.NET Core que se representa en forma de clase que declara una propiedad de solo lectura (ver indicador 3) de tipo ***IUsuarioRepository*** que es utilizado para ejecutar operaciones en la base de datos explicado en la sección anterior ***Capa de Acceso a datos*** *(proyecto Soportec.DAL).* Como toda clase de C# cuenta con un constructor, el controlador ***UsuarioController*** no es la excepción (ver indicador 4) haciendo uso de un mecanismo llamada ***Inyección de dependencias*** el controlador pide le sea otorgada una instancia del tipo ***IUsuarioRepository*** para crear, actualizar, consultar o eliminar un usuario en la base de datos, en este caso ***Crear un usuario,*** esa referencia le es entregada y asignada a la propiedad ***_usuario*** (ver indicador 4), a partir de este paso en adelante los métodos que se implementen para crear, actualizar u obtener un usuario por id, se harán utilizando el objeto ***_usuario***, así como la propiedad ***_userManager*** que sigue el mismo principio de inyección de dependencia.

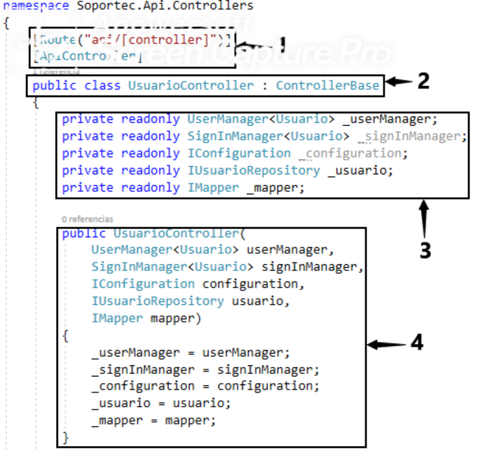


Ilustración 4.40 Implementación del controlador UsuarioController.

La configuración para que le sea proporcionado un objeto a través de inyección de dependencias pasando una interfaz del tipo de objeto que se necesite implementar, se hace en el archivo ***Startup*** en una aplicación asp.net core, específicamente en el método ***ConfigurationService*** como se observa en la ilustración 4.41.

Como se puede observar en la ilustración 4.41 el método ***ConfigurationService*** se encuentra dentro de la clase Startup que como ya se mencionó es una clase de configuración que asp.net core utiliza para configurar repositorios y servicios que se utilizarán durante todo el ciclo de vida de una petición a un controlador. En la imagen se puede observar que se configura el servicio de ***AutoMapper*** (ver indicador 1), servicio que permite implementar el mapeo de objeto de un tipo de clase de dominio a un tipo de clase DTO (recordar termino, pues se verá en futuros Sprints), otro de los servicios que se configuran es la cadena de conexión para las migraciones de clases a la base de datos (ver indicador 2) e ***Identity*** (servicio proporcionado por Microsoft para la gestión de usuarios) (ver indicador 3), el servicio que consiste en la inyección de dependencias es en el que se muestran los repositorios con los nombres de cada una de las clases del modelo de dominio, cada vez que un controlador en el constructor declare una interfaz de tipo ***IUsuarioRepository*** el contenedor de inyección de dependencias automáticamente le proporcionará un objeto creado del tipo ***UsuarioRepository***, y así con cada una de las clases que estén configurada para ser inyectadas (ver indicador 4), otro servicio que se configura es el de un servidor ***SMTP*** que se utilizará para enviar correos (ver indicador 5) a los usuarios con una contraseña aleatoria que se genera cuando son registrados por el administrador de la aplicación y por último se configura un servicio para que cuando se ejecute la aplicación en desarrollo o producción se mande a llamar las vistas HTML y CSS que forman parte de la aplicación (parte del front-end) (ver indicador 6).

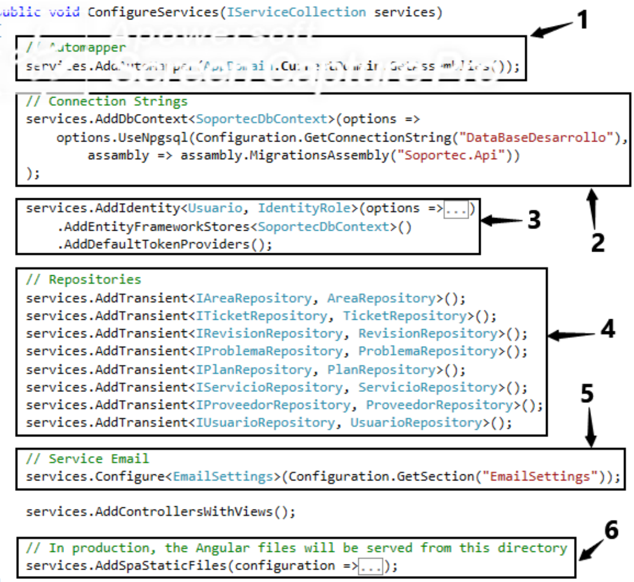


Ilustración 4.41 Configuración de la inyección de dependencias y servicios.

Después de la configuración de la clase Startup como ya se mencionó, además de la creación del controlador para la gestión de los usuarios, se creó el primer método que permite registrar y actualizar un usuario (ver ilustración 4.42)

En la ilustración 4.42 se muestra el método utilizado para crear y actualizar un usuario en la aplicación, el método recibe un parámetro de tipo ***UsuarioDto*** (ver indicador 1) que son los datos enviados desde el formulario (front-end) en formato **Json**, una vez que los datos son recibidos se hace una validación para verificar si la propiedad usuarioId viene vacía o cuanta con un dato (ver indicador 2), si la propiedad está vacía, es decir, no trae información, se entiende que se requiere crear un usuario, entonces los datos de la propiedad ***usuarioDto*** se aginan a una nueva variable llamada ***user***, posteriormente se genera una contraseña aleatoria y se crea el usuario (ver indicador 3), una vez que se crea el usuario se valida si se creó correctamente y se envía una notificación al usuario que lo creó (ver indicador 4), si el usuario no se puede crear, se envía un mensaje al usuario con la leyenda ***La creación del usuario falló*** (ver indicador 5), por último, si el usuario en los datos enviados desde el formulario contiene un valor el campo usuarioId, se hace una búsqueda del usuario (ver indicador 6) y los datos obtenidos de la búsqueda se sustituyen por los enviado en el formulario y se ejecuta la instrucción ***_userManager.updateAsync(user)*** la cual hace una actualización del usuario (ver indicador 7).

```
[HttpPost]
0 referencias
public async Task<IActionResult> Post([FromBody] UsuarioDto usuarioDto)   <-- 1
{
    if (string.IsNullOrEmpty(usuarioDto.Id))   <-- 2
    {
        try
        {
            var user = new Usuario{...};   <-- 3
            // Generar password aleatorio y enviarla por correo
            var randomPassword = new Random().Next(0, 999999);
            var result = await _userManager.CreateAsync(user, randomPassword.ToString());
            if (result.Succeeded)   <-- 4
            {
                // Enviar randomPassword por correo
                var response = new ApiResponse<UsuarioDto>(usuarioDto);
                return Ok(response);
            }
            else   <-- 5
            {
                return NotFound("La creación del usuario falló.");
            }
        }
        catch (Exception ex)
        {
            return NotFound($"{ex}");
        }
    }
    else
    {
        // Ingresar instrucciones de actualización de usuario   <-- 6
        Usuario user = await _userManager.FindByIdAsync(usuarioDto.Id);
        user.FullName = usuarioDto.FullName;
        user.Email = usuarioDto.Email;
        user.DeptoId = usuarioDto.DeptoId;
        user.Rol = usuarioDto.Rol;
        var result = await _userManager.UpdateAsync(user);   <-- 7
        var response = new ApiResponse<UsuarioDto>(usuarioDto);
```

Ilustración 4.42 Método Post para crear y actualiza usuarios.

El proceso ***Mapear*** que se mencionará más seguido, se realiza instalando una librería llamada ***AutoMapper.Extensions.Microsoft.DependencyInjection***, el cual es un proceso que consiste en convertir un tipo de clase de dominio de la aplicación a un objeto de tipo ***DTO***, es decir, se crea una clase heredada de la clase ***Profile*** como se muestra en la ilustración 4.43 (ver indicador 1), cada clase del modelo de dominio tiene una clase de tipo DTO como se observa en la ilustración 4.43 en el indicador 2, para el caso de la clase Usuario se crea una propiedad ***CreateMap*** (ver indicador 3) el cual indica que se convierte del tipo ***UsuarioDto***, es decir, datos que vienen de lado del cliente (formularios web), al tipo ***Usuario***, que son, datos que se insertan en la base de datos (ver indicador 4).

```
public class AutoMapperProfile : Profile
{
    0 referencias
    public AutoMapperProfile()
    {
        CreateMap<AreaDto, Area>();
        CreateMap<Area, AreaDto>();

        CreateMap<TicketDto, Ticket>();
        CreateMap<Ticket, TicketDto>();

        CreateMap<PlanDto, Plan>();
        CreateMap<Plan, PlanDto>();

        CreateMap<ProblemaDto, Problema>();
        CreateMap<Problema, ProblemaDto>();

        CreateMap<ProveedorDto, Proveedor>();
        CreateMap<Proveedor, ProveedorDto>();

        CreateMap<RevisionDto, Revision>();
        CreateMap<Revision, RevisionDto>();

        CreateMap<ServicioDto, Servicio>();
        CreateMap<Servicio, ServicioDto>();

        CreateMap<UsuarioDto, Usuario>();
        CreateMap<Usuario, UsuarioDto>();
    }
}
```

Ilustración 4.43 Mapeador creado para transformar los tipos de datos.

Por último, en la ilustración 4.44 se muestra la clase ***Usuario*** (ver indicador 1) que contiene todas las propiedades de un usuario que son almacenadas en la base de datos (ver indicador 2), así como las clases con las que se relaciona un usuario (ver indicador 3). La clase ***Usuario*** hereda de la clase ***IdentityUser*** la cual contiene otras propiedades como correo, estado del usuario y contraseña, que se utilizan para iniciar sesión en la aplicación, así como los métodos que permiten crear, actualizar y consultar un usuario.

```
namespace Soportec.Models
{
    24 referencias
    public class Usuario : IdentityUser
    {
        2 referencias
        public string FullName { get; set; }
        2 referencias
        public string Rol { get; set; } // Administrador, Coordinador, Agente, Cliente
        4 referencias
        public int DeptoId { get; set; }

        1 referencia
        public Area Depto { get; set; }
        1 referencia
        public ICollection<Ticket> Agente2Ticket { get; set; }
        1 referencia
        public ICollection<Ticket> Usuario2Ticket { get; set; }
        1 referencia
        public ICollection<Revision> Usuario2Revision { get; set; }
        1 referencia
        public ICollection<Plan> Usuario2Plan { get; set; }
    }
}
```

Ilustración 4.44 Modelos de datos que representa al usuario almacenado en la base de datos.

Es importante mencionar que en futuros ***Sprints*** cuando se mencione la palabra ***Cliente***, se hace referencia a la aplicación que se ejecuta en ***Angular***, debido a que es quien envía la información a los métodos de los controladores en ***ASP.NET Core*** y quien recibe la información que los controladores envían para mostrar los resultados de una inserción, actualización o consulta.

4.5.2 Back-End: Implementación del controlador para el inicio de sesión de los usuarios de la aplicación

En la ilustración 4.45 se observa el método llamado ***Login*** que hace posible la identificación de un usuario que quiera ingresar a la aplicación a ejecutar alguna funcionalidad.

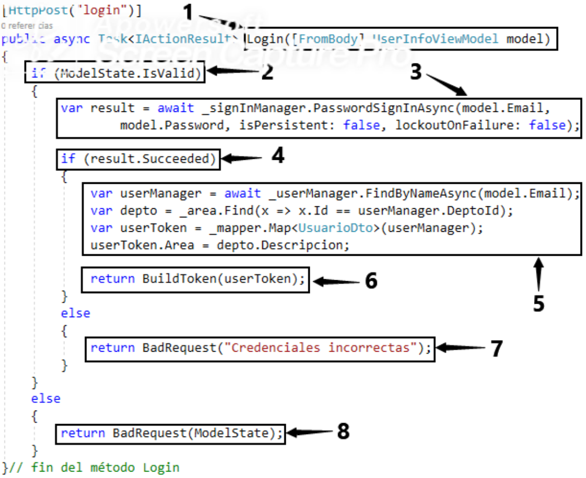


Ilustración 4.45 Método para login de usuario.

El método recibe un parámetro de tipo ***UserInfoViewModel*** el cual contiene dos propiedades que son utilizadas para iniciar sesión, estas propiedades son el ***correo*** del usuario con el que se registró y la ***contraseña*** (ver indicador 1), una vez que los datos son recibidos se hace una validación para comprobar si los datos son correctos (ver indicador 2), posteriormente se verifica si el correo y contraseña son correctos (ver indicador 3), si los datos con correctos (ver indicador 4) se hace una búsqueda por correo y se extrae de la base de datos la información del usuario y se hace una consulta para ver a que departamento corresponde el usuario y se mapean los datos al tipo de datos ***UsuarioDto*** (ver indicador 5) para enviar los datos obtenidos de la consulta al método ***BuildToken*** (ver indicador 6) que generara un token para que el usuario desde la aplicación ***Angular*** (aplicación cliente) pueda iniciar sesión, si el correo no cumpliera con el formato correcto o la propiedad contraseña de la clase ***UserInforViewModel*** fuera nula, se le enviaría un ***BadRequest*** a la aplicación cliente indicando que las credenciales son incorrectas, que intente nuevamente iniciar sesión (ver indicador 7 y 8).

En la ilustración 4.45 después de que se verifica que el usuario si existe y se obtiene los datos de dicho usuario el método ***BuildToken*** recibe un parámetro de tipo ***UsuarioDto*** como se muestra en la ilustración 4.46 (ver indicador 1), como los datos enviados son el correo, contraseña y el rol de usuario, estos datos son utilizados para cifrarlos en el token que se envía al cliente (ver indicador 2 y 3), al token generado se le proporciona una hora para que expire (ver indicador 4), es decir, a partir de que el usuario inicia sesión, tiene una hora para realizar actividades dentro de la aplicación, ya que después, el token expirará y tendrá que iniciar sesión nuevamente para que se le genere otro token de una hora, parece algo molesto, pero es una medida de seguridad muy utilizada en muchas aplicaciones ***ASP.NET CORE***, los tokens una vez que son generados se firman y se envian al cliente para que los almacene y los utilice de la mejor manera posible (ver indicador 5 y 6).

```
private IActionResult BuildToken(UsuarioDto userInfo)
{
    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.UniqueName, userInfo.Email),
        new Claim(ClaimTypes.Role, userInfo.Rol),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
    };

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Llave_super_secreta"]));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var expiration = DateTime.UtcNow.AddHours(1);

    JwtSecurityToken token = new JwtSecurityToken(
       issuer: "itsm.com",
       audience: "itsm.com",
       claims: claims,
       expires: expiration,
       signingCredentials: creds);

    return Ok(new
    {
        id = userInfo.Id,
        user = userInfo,
        token = new JwtSecurityTokenHandler().WriteToken(token),
        expiration
    });
}
```

Ilustración 4.46 Método para crear un token.

A continuación, se explicará la vista (***formulario***) en donde se ingresa toda la información que es enviada al controlador ***UsuarioController*** que contiene la lógica de negocios que hace posible la creación, actualización de los usuarios de la aplicación y el inicio de sesión para la gestión de actividades.

4.5.2 Front-End: Formularios para el registro de los usuarios de la aplicación

En la ilustración 4.47 se muestra la plantilla HTML utilizada para registrar un usuario (ver indicador 1).

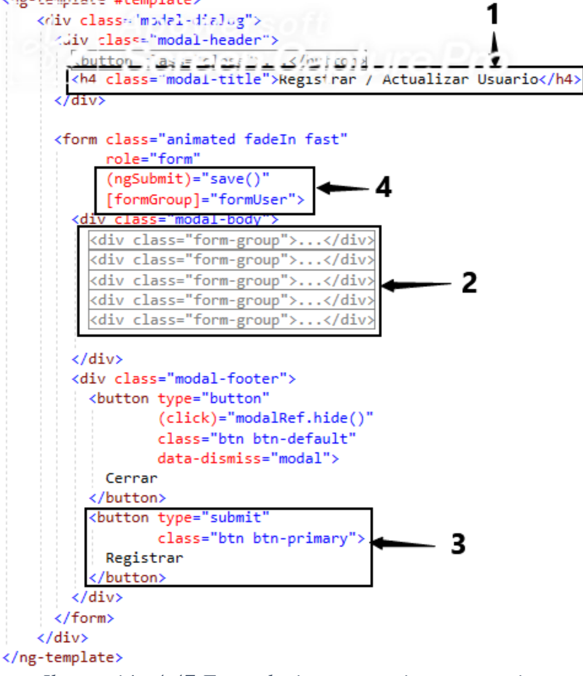


Ilustración 4.47 Formulario para registrar usuarios.

El formulario cuenta con propiedades que permiten obtener los datos ingresados por el usuario, como nombre apellidos, departamento al que pertenece y el rol en la aplicación (ver indicador 2), una vez que el usuario ingresa la información y se da clic en el botón registrar (ver indicador 3), se ejecutar la función ***save*** que hace posible la persistencia de la información introducida en el formulario (ver indicador 4).

```
save() {
  if (!this.formUser.valid) {
    return;
  }
  if (this.selectedUser) {
    // actualizar
    this.selectedUser = new Usuario(
      this.formUser.value.fullName,
      this.formUser.value.email,
      this.formUser.value.rol,
      Number.parseInt(this.formUser.value.deptoId),
      this.formUser.value.password,
      this.selectedUser.id
    );
    this._userService.createUser(this.selectedUser)
      .subscribe(user => {
        this.selectedUser = undefined;
        this.resetModalPopup();
      });
  } else {
    // crear
    this.selectedUser = new Usuario(
      this.formUser.value.fullName,
      this.formUser.value.email,
      this.formUser.value.rol,
      Number.parseInt(this.formUser.value.deptoId),
      this.formUser.value.password
    );
    this._userService.createUser(this.selectedUser)
      .subscribe(user => {
        this.selectedUser = undefined;
        this.resetModalPopup();
      });
  }
}
```

Ilustración 4.48 Función save que se comunica con el back-end para persistir información.

En la ilustración 4.48 se muestra el método que hace posible el envío de los datos del formulario al back-end, una vez que se ejecuta el método ***save***, se valida que los datos ingresados en el formulario sean correctos (ver indicador 1), posteriormente se valida si los datos se requieren actualizar, si es el caso, se ejecuta la instrucción que asigna los datos obtenidos del formulario y se asignan a un nuevo objeto de tipo ***Usuario*** (ver indicador 2) y se envían al back-end haciendo uso de la instrucción ***_userService.createUser(this.selectedUser)*** (ver indicador 3), si los datos no se requieren actualizar, entonces se ejecuta la instrucción que permite hacer una modificación de los datos existentes en la base de datos (ver indicador 4) y por último los datos son enviados al back-end para su modificación (ver indicador 5).

Otra funcionalidad que se implementó en la aplicación y que es indispensable, es la tabla donde se muestran todos los usuarios que han sido registrados, como se observa en la ilustración 4.49. Una tabla que muestra información, contiene un encabezado rotulando los

campos que son utilizados para mostrar al administrador los usuarios que están registrados (ver indicador 1), para poblar la tabla de información de los usuarios registrados cuando se carga la página que contiene la tabla se ejecuta un método llamado ***getUsers*** (ver ilustración 4.50), si al cargar los datos no existe ningún usuario se muestra un mensaje indicando que no se encontraron registros (ver indicador 2), por el contrario, si existen registros en la base de datos, los datos son embebidos uno a uno en la tabla (ver indicador 3).

```html
<table class="table">
  <tbody>
    <tr>
      <th class="text-center">Nombre</th>
      <th class="text-center">Correo</th>
      <th class="text-center">Área</th>
      <th class="text-center">Role</th>      ← 1
      <!--<th class="text-center">Estado</th>-->
      <!--<th class="text-center">Fecha Creado</th>-->
      <th></th>
    </tr>
    <tr *ngIf="users.length === 0">
      <th class="text-center" colspan="5">   ← 2
        No se encontraron registro...
      </th>
    </tr>
    <tr *ngFor="let user of users">
      <td>{{ user.fullName }}</td>
      <td>{{ user.email }}</td>
      <td>{{ user.depto['descripcion'] }}</td>
      <td class="text-center">
        <p class="badge bg-blue">{{ user.rol }}</p>
      </td>
      <td class="text-center">...</td>
    </tr>
  </tbody>                                   3
</table>
```

Ilustración 4.49 Código HTML que muestra los usuarios registrados.

En la ilustración 4.50 se puede observar el método que se mencionó anteriormente que ayuda a poblar la tabla de información de los usuarios registrados, una vez que se ejecuta el método, se hace una petición al back-end haciendo uso del método ***_userService.getPaged*** (ver indicador 1) y los datos enviados por el back-end se asignan a la variable ***user*** (ver indicador 2) para que la tabla con el código HTML pueda embeber dichos registros.

```typescript
getUsers() {
  this._userService.getPaged(this.page)
  .subscribe(users => {
    this.users = users.data;
    this.totalUsers = users.count;
    this.countPages();
  });
}
// 1      2
```

Ilustración 4.50 Método que llena la tabla de usuarios.

4.5.3 Front-End: Formulario utilizado para el inicio de sesión de los usuarios de la aplicación

Una vez que los usuarios son registrados en la aplicación, necesitaran de un mecanismo que los ayude a ingresar las credenciales con las que fueron registrados e iniciar sesión para realizar actividades como crear registros en los catálogos, crear listas de verificación o actualizar un plan de trabajo (ver ilustración 4.51). Para que tales actividades puedan ser realizadas, los usuarios al ingresar la ***URL*** donde se encuentra alojada la aplicación lo primero que observarán será un formulario que les pedirá que ingresen correo (ver indicador

1) y contraseña (ver indicador 2) y un checkbox por si desean que el navegador web que utilizan recuerde su correo (ver indicador 3), una vez que los datos son introducidos en el formulario el usuario al hacer clic en el botón iniciar sesión (ver indicador 4) se ejecutará el método login, el cual tiene programado una lógica que envía las credenciales al back-end para verificar las credenciales proporcionadas (ver indicador 5), si el usuario ingresó correctamente la información, tendrá acceso a la aplicación, según el rol con el que fue registrado.

```
<form ngNativeValidate #f="ngForm" (ngSubmit)="login(f)">         ← 5
  <div class="form-group has-feedback">
    <input [ngModel]="email"
           name="email"
           type="email"                                           ← 1
           class="form-control"
           placeholder="Correo Electrónico">
    <span class="glyphicon glyphicon-envelope form-control-feedback"></span>
  </div>
  <div class="form-group has-feedback">
    <input ngModel
           name="password"
           type="password"                                        ← 2
           class="form-control"
           placeholder="Contraseña">
    <span class="glyphicon glyphicon-lock form-control-feedback"></span>
  </div>

  <div class="row">
    <div class="col-xs-7">
      <div class="icheck-primary">
        <input [(ngModel)]="remember"
               name="remember"
               type="checkbox"                                    ← 3
               id="remember">
        <label for="remember"> Recuérdame</label>
      </div>
    </div>
    <div class="col-xs-5">                                        4
      <input type="submit"
             class="btn btn-primary bg-light-blue-active btn-block btn-flat"
             value="Iniciar Sesión">
    </div>
  </div>
</form>
```

Ilustración 4.51Código HTML para el formulario de Login.

El método login mostrado en la ilustración 4.52, corresponde a la lógica de negocios del lado del cliente para validar el formulario de login (ver indicador 1), donde se verifica que el correo tenga un formato adecuado (ver indicador 2), una vez que el formulario para la validación los datos ingresados en el formulario login se aginada a la variable ***data*** (ver indicador 3) y posteriormente la información que contiene la variables ***data*** es enviada al back-end para validarla contra la base de datos, y si la validación es correcta el usuario es redireccionado a la pantalla principal de la aplicación la cual consiste en un ***dashboard*** (ver indicador 4).

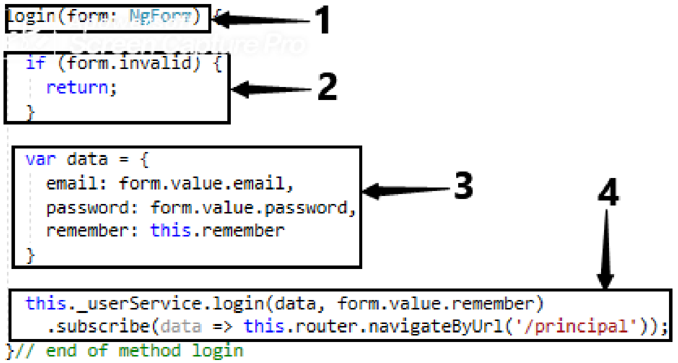


Ilustración 4.52 Lógica del lado del cliente del formulario de login.

4.6 Sprint 3: Implementación del módulo de catálogos

En la tabla 4.7 se presenta la planeación del Sprint 3, donde cada historia de usuario se desglosa en tareas que tiene un tiempo estimado para su realización, acordado por el equipo ***Scrum***, así como las fechas de inicio y finalización del Sprint.

Tabla 4.7 Planeación del Sprint 3.

Sprint 3				
Objetivo: El objetivo del Sprint es que el administrador pueda gestionar la información de los catálogos.				
Fecha de inicio		Fecha de finalización		N°. de semanas
01-07-2019		07-08-2019		6
Historias de usuario	**Tareas del Sprint 3**	**Fecha de inicio**	**Fecha de finalización**	**Horas de trabajo invertidas**
HU2, HU3	Configurar el proyecto para persistir la información en los catálogos.	01-07-2019	10-07-2019	30
	Crear la funcionalidad que permita registrar, actualizar, eliminar y obtener información de los catálogos.	11-07-2019	20-07-2019	72
	Crear la vista para que el administrador pueda ingresar información en los catálogos.	21-07-2019	06-08-2019	140
Total de horas de trabajo invertidas				242

Como en el sprint anterior las tareas realizadas en la historia de usuario serán presentadas en dos fases **Back-End** y **Front-End**.

Es importante mencionar que los catálogos desarrollados en este sprint son tres, los cuales se utilizan para almacenar información de los proveedores, los tipos de servicio que se realizarán y los problemas más comunes que se presentan en un equipo o infraestructura. La información de estos catálogos se pudo almacenar en una sola tabla y categorizarlos con un campo llamado tipo, pero debido a que si en un futuro se quieren implementar más características sobre cada uno de los proveedores, problemas o tipos de incidencias presentadas en una infraestructura se tendría que modificar la base de datos. En el presente sprint solo se hablará del desarrollo de un catálogo (problemas), debido a que el back-end y front-end es la misma estructura, la misma lógica, lo única diferencia entre estos es que solo cambia el concepto que se almacena en la base de datos.

4.6.1 Back-End: Implementación de las funcionalidades para la gestión de catálogos

En esta sección se hablará de la lógica de gestión del módulo de catálogos, debido a que la parte en que se prepara la aplicación para persistir la información en la base de datos se explicó en el Sprint 1. En el presente y los siguientes Sprints se explicará a partir de donde se comienza a gestionar la información para persistirla en la base de datos (controladores).

El patrón arquitectónico utilizado para el desarrollo de la presente aplicación es el MVC utilizando la tecnología ASP.NET Core 2 como se mencionó en el capítulo 2. Las aplicaciones que se rigen bajo este patrón comienzan su ciclo de vida en los controladores, puesto que cada vez que un usuario realiza una petición es el primer componente que se activa o ejecuta algún tipo de tarea, en este caso consultar, registrar, actualizar o eliminar algún registro del catálogo.

```
public class Area
{
    4 referencias
    public int Id { get; set; }
    2 referencias
    public string Descripcion { get; set; }
    0 referencias
    public bool Estado { get; set; }
    0 referencias
    public bool Servicio { get; set; }

    1 referencia
    public ICollection<Revision> Area2Revision { get; set; }
    1 referencia
    public ICollection<Plan> Depto2Plan { get; set; }
    1 referencia
    public ICollection<Revision> Depto2Revision { get; set; }
}
```


Ilustración 4.53 Clase de dominio que almacena Áreas.

En la ilustración 4.53 se muestra la clase que persiste la información en tiempo de ejecución, como se observa en la imagen, la clase tiene cuatro campos los cuales el primero almacena el identificador único del registro, la descripción la cual almacena el nombre del área, el estado almacena el estado activo o inactivo del registro y el campo servicio almacena si el área (nombre) que se almacena es un área que brinda o no un servicio a las demás áreas almacenadas, como se muestra en el indicador 1, el indicador 2 muestra las clases con las que se relaciona la clase Área, que son la clase Revisión y Plan.

La clase que almacena los problemas más comunes de una solicitud preventiva se persisten en tiempo de ejecución en la clase que se muestra en la ilustración 4.54.

```
public class Problema
{
    3 referencias
    public int Id { get; set; }
    1 referencia
    public string Descripcion { get; set; }
    0 referencias
    public bool Estado { get; set; }

    1 referencia
    public ICollection<Ticket> Tickets { get; set; }
}
```

Ilustración 4.54 Clase que almacena problemas más comunes.

Los campos mostrados en la clase Problema permiten almacenar el identificador único de cada registro, una descripción y el estado del registro, es decir, activo o inactivo (ver indicador 1), así como las clases con las que se relaciona dicha clase (ver indicador 2).

```
public class Proveedor
{
    3 referencias
    public int Id { get; set; }
    1 referencia
    public string Descripcion { get; set; }
    0 referencias
    public bool Estado { get; set; }

    1 referencia
    public ICollection<Ticket> Tickets { get; set; }
}
```

Ilustración 4.55 Clase que almacena información básica de proveedores.

Como requerimiento del cliente también se tomó en cuenta almacenar nombres de proveedores, debido a tal requerimiento se almacenan proveedores en la clase que se muestra en la ilustración 4.55, como se observa los datos requeridos son identificador único del registro, descripción del proveedor y el estado de dicho proveedor (ver indicador 1), así como las clases que se relacionan con dicha clase (ver indicador 2).

```
public class Servicio
{
    3 referencias
    public int Id { get; set; }
    1 referencia
    public string Descripcion { get; set; }
    1 referencia
    public bool Estado { get; set; }

    1 referencia
    public ICollection<Ticket> Tickets { get; set; }
}
```

Ilustración 4.56 Clase almacena información de los servicios realizados.

La clase servicios mostrada en la ilustración 4.56, se utiliza para almacenar los servicios más comunes que se realizan, es decir, pueden almacenar servicios como jardinería, obra civil, albañilería, herrería y demás conceptos que el administrador de la aplicación considere necesarios. Como se observa en las clases catálogo los campos son similares (ver indicador 1).

Las clases mostradas anteriormente son las entidades del dominio que permiten un panorama más amplio de lo información que se necesita almacenar. Los catálogos no son otra cosa más que información almacenada que se repite en uno u otra sección de la aplicación y que al desarrollar los catálogos lo que se consigue es centralizar dicha información para no repetirla.

A continuación, se explicará el controlador que gestiona cada una de las solicitudes que se le realizan a un catálogo, es decir, quien y que componente responde a las peticiones más comunes (creación, actualización, consulta y eliminación) que realiza el usuario desde las vistas de la aplicación.

Los controladores en el marco MVC son componentes que atienden las solicitudes realizadas por los usuarios (ver capítulo 2), en dichos controladores se desarrolla la lógica de una aplicación, en el caso de la aplicación que trata este tema de tesis, la lógica CRUD se desarrolla en un componente diferente como se explicó en el Sprint 1. En esta sección se detallará como es que en el controlador utiliza los patrones Repository para gestionar la información de los catálogos. Para la creación del módulo de catálogos se desarrollaron cuatro controladores (AreaController, ProblemaController, ProveedorController y ServicioController) para esta sección debido a que lo único que cambia entre los controladores son los nombre solo se explicará el AreaController, puesto que la lógica para todos es la misma.

```
[Route("api/[controller]")]        ← 1
[ApiController]
1 referencia                                   2
public class AreaController : ControllerBase
{                                                       3
    private readonly IAreaRepository _departamento;
    private readonly IMapper _mapper;
                                                          4
    0 referencias
    public AreaController(IMapper mapper,
                          IAreaRepository departamento)
    {
        _departamento = departamento;
        _mapper = mapper;
    }
```

Ilustración 4.57 Configuración del AreaController.

En la ilustración 4.57 se observa la estructura básica de configuración de un controlador, el cual cuenta con atributos de configuración que permiten renombrar la ruta de un controlador, en este caso la ruta a la que se tiene que hacer la petición es http://localhost:5000/area/nombreDelMetodo, como se puede observar en la ruta la nomenclatura es **dominio:puerto/nombre del controlador/nombre del método al cual se le hace la petición** (ver indicador 1).

El método al que se le hace la petición es el que contiene la lógica de negocio, en este caso la lógica se encuentra en un componente diferente.

El nombre del controlador puede seguir la nomenclatura con la que se nombre una clase, en este caso se llama AreaController y hereda características de una clase base llamada ControllerBase (ver indicador 2) que contiene muchas de las funciones que realiza un api web como lo es validaciones de modelos, filtros de configuración, configuraciones de peticiones y entre muchas otras características que son encontradas en la página oficial de Microsoft ASP.NET CORE.

El AreaController cuenta con dos propiedades, una que utiliza como tipo de datos una interface (IAreaRepository) que permite utilizar los métodos del patrón repositorio y la otra propiedad que me permite mapear (convertir un tipo de dato a otro) los datos que recibo de un usuario de la aplicación y se almacenan en la base de datos, así como convertir los datos que se extraen de la base de datos y se envían al usuario que los solicita. El constructor del controlador hace uso de una característica llamada inyección de dependencia, la cual su principio básico es que en el constructor del controlador declara las interfaces de las cuales se requiere un objeto y automáticamente la aplicación te brinda un objeto del tipo especificado (ver indicador 4) que posteriormente se asignan a las propiedades declaradas para referenciarlas y poder hacer uso de los métodos que brinda cada clase.

Una vez que la configuración del controlador está listo se procede a desarrollar los métodos que se necesitan para manipular información de la base de datos, en este caso los métodos utilizados son seis, es decir, un método para obtener todas las áreas almacenadas en la base de datos, un método para realizar una búsqueda por identificador único del registro, un método para almacenar un registro, un método para actualizar un registro, otro método para obtener registro paginados, este método es muy utilizado debido a que cuando la base de datos contiene pocos registro al realizar una consulta de todos los registros no tiene ningún problema, el problema se presenta cuando en la base de datos ya existen mil registros, al consultar todos el sistema se satura y comienza a tener problemas de rendimiento, es por eso que esta funcionalidad es muy útil, ya que no se consultan todos, sino que se van consultando de cinco en cinco o como el administrador del sistema lo requiera. El último método que se implementa es el de eliminar, debido a que en la aplicación se consideró un campo llamado estado en la clase Área no es necesario eliminar el registro, sino que se cambia de estado el registro, esto con la finalidad de que no existan inconsistencias en la base de datos si se eliminara un registro que ya está relacionado con otra tabla en la base de datos.

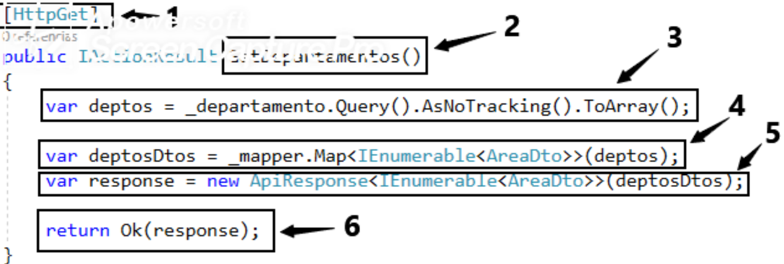


Ilustración 4.58 Método para obtener todos los departamentos.

El método (endpoind) para obtener todos los áreas registrados en la base de datos se muestra en la ilustración 4.58, al ser un método de una api a la que se puede consultar haciendo peticiones desde un navegador web, dicho método se firma con un verbo **Get** que indica que la información se puede consultar desde cualquier medio que permita peticiones

de servicio HttpGet (ver indicador 1), el nombre del método es importante ya que como toda clase en C# debe ser único (ver indicador 2), dentro del cuerpo del método se hace una consulta al repositorio que contiene la lógica de acceso a la base de datos en este caso la propiedad **_departamento** ejecutando el método **Query**, una vez que se obtiene la información de la base de datos se almacena en la variable **deptos** (ver indicador 3), posteriormente la información almacenada en la variable **deptos** se convierte al tipo **AreaDto** que no es otra cosa que una clase que permite comunicarme con el usuario que realiza las peticiones, esto con la finalidad de no mostrarle la clase de dominio llamada **Area** que debido a buenas prácticas no es recomendable mostrarla, es por eso que se hace uso de la clase **Dto** (ver indicador 4) , una vez que la información es mapeada se pasa como argumento al constructor de la clase **ApiResponse** (me permite dar un formato a la respuesta que envío al cliente) (ver indicador 5), por último se envía el resultado de las operaciones al cliente haciendo uso del método Ok que le indica al usuario final que la operación se realizó satisfactoriamente (ver indicador 6).

```
[HttpGet("{id}")]                    <- 1
public IActionResult GetId(int? id)  <- 2
{
    if (id == null)                  <- 3
        return NotFound();
                                        4
    Area depto = _departamento.Find(q => q.Id == id);

    if (depto == null)               <- 5
        return NotFound();
                                        6
    var deptoDto = _mapper.Map<AreaDto>(depto);
    var response = new ApiResponse<AreaDto>(deptoDto);

    return Ok(response);
}
```

Ilustración 4.59 Método para buscar usuario por Id.

El segundo método para consultar información de las áreas es el mostrados en la ilustración 4.59, el método contiene un atributo que se llama **id** (ver indicador 1) que permite pasarlo al método para realizar posteriormente una consulta (ver indicador 2), una vez que el identificador de tipo entero es recibido se valida si es un valor nulo, en caso de ser nulo envía al usuario un mensaje indicando que no fue encontrado el recurso que buscaba (ver indicador 3) si la validación del id es falsa se realiza la búsqueda del área comparando los id que correspondan con el que se pasó por parámetro (ver indicador 4), una vez que se busca, si el área se encuentra en la base de datos se realiza un mapeo de entidades y se envían al cliente (ver indicador 6), en caso contrario se envía un mensaje al usuario diciendo que el recurso buscado no fue encontrado (ver indicador 5).

En la ilustración 4.60 se muestra uno de los métodos más importantes del controlador ***AreaController***, el cual permite registrar un nuevo recurso, en este caso se indica con un atributo HttpPost (ver indicador 1) el cual el controlador interpreta que ese método realiza la lógica de crear. El método se nombra con un identificador único en la clase y con un parámetro que recibo de tipo **AreaDto** que como se mencionó anteriormente se hace uso de

buenas prácticas con las clase **Dto** (ver indicador 2), en el cuerpo del método (ver indicador 3) se pasa el parámetro **deptoDto** al constructor del mapeador de clases para convertir de tipo **AreaDto** a **Area** siendo esta última clase un modelo de dominio, una vez que se mapea la información se insertan los datos en la base de datos y se vuelve a mapear para enviarle al usuario final la respuesta, es decir, los datos que se insertaron en la base de datos se vuelven a enviar al usuario como confirmación de que la operación de guardar se realizó con éxito (ver indicador 4 y 5).

```
[HttpPost]
0 referencias
public async Task<IActionResult> Post(AreaDto deptoDto)
{
    var depto = _mapper.Map<Area>(deptoDto);

    await _departamento.AddAsync(depto);
    deptoDto = _mapper.Map<AreaDto>(deptoDto);

    var response = new ApiResponse<AreaDto>(deptoDto);

    return Ok(response);
}
```

Ilustración 4.60 Método para crear un nuevo registro.

La ilustración 4.61 muestra el segundo método más importante, el cual permite actualizar un recurso ya registrado en la base de datos. El método debe utilizar el atributo HttpPut (ver indicador 1) para que la aplicación interprete que necesita realizar una actualización de un recurso, el nombre del método como todos lo demás deben de ser únicos (ver indicador 2) y recibe dos valores un identificador y en el otro parámetro un valor de tipo **AreaDepto**, una vez que se reciben los valores se realiza una validación para asegurar que el identificador no es nulo, de ser nulo se envía un mensaje al usuario indicando que el valor es un recurso que no existe en la base de datos (ver indicador 3), posteriormente a la validación se hace un mapeo de datos y se procede a realizar la actualización haciendo uso del repositorio llamando al método **UpdateAsyn** al cual se le pasa el objeto de tipo **AreaDto** y el identificador, esto con la finalidad de que busque e inserte, una vez que se realiza la operación se vuelve a mapear la información de tipo **Area** a **AreaDto** (ver indicador 4), por último se envía la información mapeada al usuario final (ver indicador 5).

```
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> Put(int id, AreaDto deptoDto)
{
    if (id != deptoDto.Id)
        return NotFound();

    var depto = _mapper.Map<Area>(deptoDto);
    var result = await _departamento.UpdateAsync(depto, id);
    deptoDto = _mapper.Map<AreaDto>(result);

    var response = new ApiResponse<AreaDto>(deptoDto);

    return Ok(response);
}
```

Ilustración 4.61 Método para actualizar un área existente.

Como se mencionó anteriormente cuando existen muchos registros en la base de datos y se consultan todos en una sola operación el servidor de la aplicación suele saturarse por la cantidad de información que tiene que procesar, una de las soluciones más utilizadas para este problema es la paginación de registros, en el método llamado **GetAllPaged** (ver indicador 2) se reciben dos parámetros de tipo entero los cuales son la cantidad de registro que requiero de la base de datos y la cantidad de páginas en que debe de devolverme dicha información, en el cuerpo del método una vez que se recibe la cantidad de registros y páginas se validan para ver que no vengan cantidades menores a cero (ver indicador 3), una vez que se pasa la validación se hace la consulta con el método **_departamento.GetAllPageAsync** pasándole la cantidad y las páginas que necesito (ver indicador 4) una vez que el patrón repositorio realiza la operación se mapea la información, posteriormente se cuenta la cantidad de registro totales que hay en la base de datos en la tabla Áreas, esto con la finalidad de darle a conocer el total de registros que hay en la base de datos al usuario final (ver indicador 5), por último la información se envía al usuario que la solicitó, la información enviada es los registros consultados y la cantidad de los mismos (ver ilustración 4.62) (ver indicador 6).

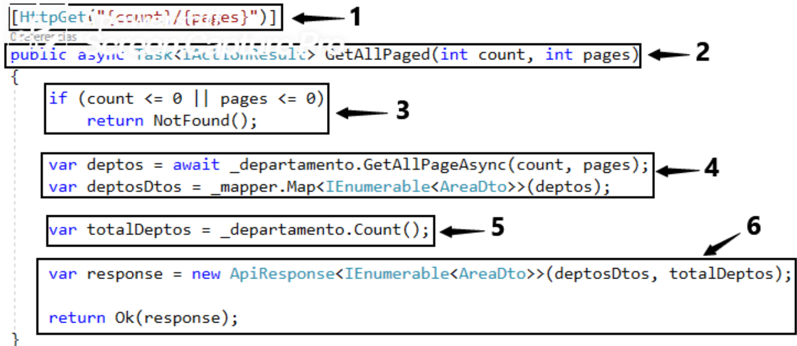


Ilustración 4.62 Método para consultar áreas paginadas.

El último método implementado en el controlador pero que no se utiliza por el momento, sino que se utilizará en futuras versiones para eliminar registros, como se observa en la ilustración 4.63, al inicio del método se decora con un atributo HttpDelete (ver indicador 1), esto se hace con la finalidad de que la aplicación interprete que si se ejecuta esta función es para eliminar un registro.

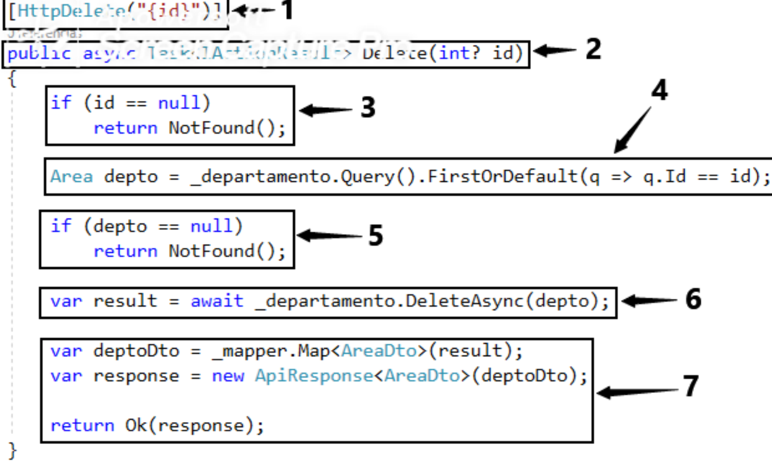


Ilustración 4.63 Método para eliminar un registro.

El método recibe un identificador de tipo entero (ver indicador 2), una vez que se valida que el entero recibido es diferente de nulo (ver indicador 3) se procede a hacer una búsqueda del registro que se desea eliminar (ver indicador 4), una vez que encuentra el registro se valida que el registro encontrado sea válido, sino se notifica al usuario (ver indicador 5), si el registro es encontrado en la base de datos se procede a eliminar el registro (ver indicador 6) y por último se mapea la información obtenido de la base de datos y se envía al usuario como confirmación de que la operación se realizó satisfactoriamente.

Como se mencionó al inicio, de las cinco clases que se diseñaron para la gestión de la información de los catálogos solo se habló de uno debido a que la lógica es la misma para los cinco controladores desarrollados.

A continuación, en el siguiente apartado se presenta la implementación del **front-end** de cada una de las funcionalidades explicadas en el **back-end**.

4.6.2 Front-End: Implementación del formulario para la creación de registros en un catálogo

La contra parte de la lógica de negocios (back-end) de una aplicación son las ventanas con las que interactúa un usuario en una aplicación (front-end), esta sección está dedicada a explicar una de las acciones con las que más interacciones tiene el usuario final y esa es la ventana donde el usuario ingresa información para alimentar la aplicación, así como las tablas para listar todos los registros que son guardados.

Para que una vista pueda recibir información, necesita de un modelo con la misma estructura que el que se encuentra en el lado del back-end, esto con la finalidad de que los datos enviados al back-end sean similares en cuanto al tipo de datos que manejan.

```
<ng template #template>
  <div class="modal-header">
    <h4 class="modal-title">Editar Departamento</h4>      1
  </div>
  <form class="animated fadeIn fast"      2
        role="form"
        (ngSubmit)="save()"
        [formGroup]="formGroup">      3
    <div class="modal-body">
      <div class="form-group">
        <label>Nombre</label>
        <input type="text"
               formControlName="descripcion"
               class="form-control"
               placeholder="Nombre del departamento">
      </div>      4
      <div class="form-group">
        <div class="icheck-primary">
          <input type="checkbox"
                 formControlName="estado"
                 id="remember">
          <label for="remember">
            Activo
          </label>
        </div>
      </div>
```

Ilustración 4.64 Formulario HTML para la creación de un área.

En la ilustración 4.64 se muestra el formulario que utiliza el usuario para registrar un área, el formulario consta de un encabezado que se utiliza para mostrarle un mensaje al usuario,

como aviso de para qué sirve dicho formulario, en este caso el mensaje dice **Editar departamento** (ver indicador 1), también el formulario cuenta con una etiqueta HTML que indica que semánticamente que lo que se está plasmando en el código es un formulario (ver indicador 2), el formulario cuenta con el atributo (ngSubmit) que indica que al hacer clic en el botón ***guardar*** se ejecutará el método **save** que internamente realiza una lógica que es la de recibir la información que tiene el formulario y enviarla al back-end para que se guarde el registro como se muestra en la ilustración 4.65.

```
save() {
  if (!this.formGroup.valid) {
    return;
  }

  if (this.selectedDepto) {
    // actualizar
    const data = {
      ...this.formGroup.value,
      id: this.selectedDepto.id
    }
    this._ubicacionService.updateUbicacion(data)
      .subscribe(depto => {
        this.selectedDepto = undefined;
        this.resetModalPopup();
      });
  } else {
    // crear
    this._ubicacionService.createUbicacion(this.formGroup.value)
      .subscribe(depto => {
        this.resetModalPopup();
      });
  }
}// end of save
```

Envía la información del formulario al backend para almacenarla en la base de datos

Ilustración 4.65 Lógica del formulario para crear registro.

En la ilustración 4.64 se muestra un control con la etiqueta ***Nombre*** en la cual se ingresa la descripción, que será la información que se envíe al back-end, el resto de campos de formulario son similares, la única diferencia es el nombre que recibe cada campo, en el caso del formulario para registrar las áreas, el modelo de dominio solo contiene tres campos que son introducidos por el usuario los cuales son, descripción, servicio y estado (ver ilustración 4.64 indicador 4).

```
    <div class="form-group">
      <div class="icheck-primary">
        <input type="checkbox"
               formControlName="servicio"
               id="servicio">
        <label for="servicio">
          Servicio
        </label>
      </div>
    </div>
  </div>
  <div class="modal-footer">
    <button type="button"
            (click)="closeModalPopup()"
            class="btn btn-default pull-left"
            data-dismiss="modal">Cerrar</button>
    <button type="submit"
            class="btn btn-primary">Guardar</button>
  </div>
</form>
</ng-template>
```

1 2 3

Ilustración 4.66 Botón para crear un registro con el formulario.

La ilustración 4.66 se muestra el campo servicio (ver indicador 1), dentro del formulario hay una etiqueta rotulada con la leyenda cerrar ese botón ejecuta una acción en particular que es la de cerrar formulario (ver indicador 2), por último, en el formulario hay un botón que muestra la leyenda Guardar, al hacer clic sobre dicho botón, se ejecuta el método **save** que permite guardar el registro que se introdujo en el formulario (ver indicador 3). El mismo formulario mostrado en la ilustración 4.66 se utiliza cuando se requiere actualizar algún registro, la única diferencia es la lógica que hay detrás del formulario al ejecutar una función que cargar los datos en el formulario.

Al hacer clic sobre el botón actualizar del registro que se requiera modificar, se ejecuta el método **getUbicacion** mostrado en la ilustración 4.67, que recibe una plantilla (formulario) y el identificador único del registro esto con la finalidad de que se pase al método **getUbicacion** del objeto **_ubicacionService** (ver indicador 1), una vez que se ejecuta dicho método el back-end retornará el usuario si se encuentra en la base de datos y se asignará a cada uno de los campos del formulario según correspondan, es decir, al campo llamado descripción se le asignará la propiedad descripción que se recibe de la base de datos y el campo servicio a la propiedad servicio del formulario html y así sucesivamente con cada uno de los campos del formulario (ver indicador 2), y por último se ejecutará una función llamada **openModal** que dicha función no hace otra cosa más que mostrar (emerge) el formulario en la pantalla del navegador, mostrando el formulario con cada uno de los campos obtenidos de la base de datos.

```
getUbicacion(template: TemplateRef<any>, id: number) {
  this._ubicacionService.getUbicacion(id)        ← 1
    .subscribe((depto: Ubicacion) => {
      const { descripcion, servicio, estado } = depto;
      this.selectedDepto = depto;
      this.formGroup.setValue({ descripcion, servicio, estado });
    });                                            ↖ 2

  this.openModal(template);   ← 3
}// fin de método cargarDepto
```

Ilustración 4.67 Lógica del front-end para actualizar registros.

4.7 Sprint 4: Implementación del módulo de listas de verificación

En la tabla 4.8 se presenta la planeación del Sprint 4, donde cada historia de usuario se desglosa en tareas que tiene un tiempo estimado para su realización, acordado por el equipo ***Scrum***, así como las fechas de inicio y finalización del Sprint.

Tabla 4.8 Planeación del Sprint 4.

Sprint 4		
Objetivo: El objetivo del Sprint es que los usuarios con el rol de administrador y coordinador puedan gestionar la información de las listas de verificación y agregar anomalías a las listas.		
Fecha de inicio	Fecha de finalización	N°. de semanas

08-08-2019		07-10-2019		10
Historias de usuario	**Tareas del Sprint 4**	**Fecha de inicio**	**Fecha de finalización**	**Horas de trabajo invertidas**
HU4, HU5, HU6	Configurar la aplicación para persistir la información haciendo uso de los controladores.	08-08-2019	15-08-2019	30
	Implementación de las funcionalidades para la gestión de las listas de verificación.	15-08-2019	29-08-2019	72
	Implementación de las funcionalidades para la gestión de anomalías encontradas en las infraestructuras y equipos.	29-08-2019	12-09-2019	60
	Implementación de las vistas para mostrar las listas de verificación creadas.	12-09-2019	19-09-2019	65
	Implementación del formulario para la creación de una lista de verificación.	19-09-2019	26-09-2019	81
	Implementación del formulario para el registro de una anomalía encontrada.	26-09-2019	08-10-2019	76
Total de horas de trabajo invertidas				384

Como en el sprint anterior las tareas realizadas en las historias de usuario serán presentadas en dos fases **Back-End** y **Front-End**.

El presente Sprint trata de la implementación del módulo de verificación de infraestructuras y equipos, proceso que se lleva al inicio del semestre (ver capítulo 1) que consiste en hacer anotaciones de las anomalías encontradas en las áreas con las que cuenta la institución u organización que haga uso de la aplicación que se desarrolla en este trabajo de tesis. El desarrollo del Sprint se desglosa en dos fases, la primera es la del back-end, en la cual se muestra la lógica para persistir la información haciendo uso de los controladores (ver capítulo 2) y la segunda fase es la de front-end, en donde se explica la forma en que se muestra la información al usuario final haciendo uso de formularios en el navegador web.

4.7.1 Configurar la aplicación para persistir la información haciendo uso de los controladores

Lo primero que se hace una vez que se crea y configura el proyecto es configurar los controladores que atenderán las peticiones de los usuarios que hacen uso de la aplicación, la tecnología elegida para desarrollar la presente aplicación dicta en su documentación que para persistir información, en el archivo de configuración de servicios se debe hacer uso de la inyección de dependencias como buena práctica, esto con el objetivo de que en cada controlador no se cree un objeto por cada Repositorio (ver capítulo 3) que se utilice, sino más bien el contenedor automático con el que cuenta el proyecto, cada que se solicite una interfaz en el controlador, sirva el objeto que le corresponde como se muestra en la ilustración 4.68.

```
// Repositories
services.AddTransient<IAreaRepository, AreaRepository>();
services.AddTransient<ITicketRepository, TicketRepository>();
services.AddTransient<IRevisionRepository, RevisionRepository>();
services.AddTransient<IProblemaRepository, ProblemaRepository>();
services.AddTransient<IPlanRepository, PlanRepository>();
services.AddTransient<IServicioRepository, ServicioRepository>();
services.AddTransient<IProveedorRepository, ProveedorRepository>();
services.AddTransient<IUsuarioRepository, UsuarioRepository>();
```

Ilustración 4.68 Configurando Repositorios.

En la ilustración 4.68 se configuran todos los repositorios que se utilizan en la aplicación, al ejecutar el proyecto en un servidor de producción cada vez que se hace petición a un controlador y este controlador haga uso de una interface de tipo ***IRevisionRepository***, la aplicación automáticamente le servirá un objeto de tipo ***RevisionRepository*** para realizar las operaciones que dicho repositorio tenga implementados, por ejemplo, obtener revisiones, buscar revisión por id, crear un revisión, actualizar un revisión.

```
[Route("api/[controller]")]
[ApiController]
1 referencia
public class RevisionController : ControllerBase
{
    private readonly IRevisionRepository _revision;
    private IMapper _mapper;

    0 referencias
    public RevisionController(IRevisionRepository revision, IMapper mapper)
    {
        _revision = revision;
        _mapper = mapper;
    }
```

Ilustración 4.69 Configuración del controlador Revisión.

Una vez que se configuran los repositorios, el controlador puede declarar una propiedad de tipo ***IRevisionRepository*** como se muestra en la ilustración 4.69 (ver indicador 1), para hacer referencia al objeto que le será proporcionado automáticamente por la aplicación, al detectar que en el constructor se le está pidiendo una implementación de la interface IRevisionRepository (ver indicador 2) para poder ejecutar los métodos que tiene implementador el IRevisionRepository, que no son otros métodos más que los CRUD

(***Create***, ***Read***, ***Update***, ***Delete***) implementados para la gestión de la información de las revisiones (listas de verificación).

La configuración básica del controlador y la implementación de los métodos de cada controlador que utiliza la aplicación es lo único que se explicará en cada Sprint, debido a que la implementación de los repositorios y su funcionamiento fue explicada en el Sprint uno (ver capítulo 3).

4.7.2 Back-End: Implementación de las funcionalidades para la gestión de las listas de verificación

Los controladores son el punto de entrada de las peticiones que realizan los usuarios, por lo tanto, es el primer componente que se utiliza cuando se requiere hacer una consulta de las listas de verificación almacenadas en la base de datos. Como se muestra en la ilustración 4.70, se hace una consulta a la base de datos haciendo uso del método **Query** (ver indicador 1) que permite llamar todos los registros de la base de datos, posteriormente se mapean a otro tipo de clase (**RevisionDto**) que permite serializar la información (ver indicado 2) y enviarla al usuario que la solicitó, por último la información serializada se enviar haciendo uso del método **Ok** (ver indicador 3).

```
[HttpGet]
0 referencias
public IActionResult GetVerificaciones()
{
    var revisiones = _revision.Query().AsNoTracking().ToArray();

    var revisionDtos = _mapper.Map<IEnumerable<RevisionDto>>(revisiones);
    var response = new ApiResponse<IEnumerable<RevisionDto>>(revisionDtos);

    return Ok(response);
}
```

Ilustración 4.70 Método para consultar todas las listas de verificación.

Los métodos de consulta en el controlador ***RevisionController*** son dos, los cuales el primero permite listar todos los registros obtenidos de la base de datos y el segundo, permite listar una cierta cantidad de registro que son pasados por parámetros al método (ver ilustración 4.71).

```
[HttpGet("{count}/{pages}")]
0 referencias
public IActionResult GetAllPage(int count, int pages)
{
    if (count <= 0 || pages <= 0)
        return NotFound();

    var revisiones = _revision.GetAllPageAsync(count, pages);
    var revisionDtos = _mapper.Map<IEnumerable<RevisionDto>>(revisiones);

    var totalRevisiones = _revision.Count();

    var response = new ApiResponse<IEnumerable<RevisionDto>>(revisionDtos, totalRevisiones);

    return Ok(response);
}
```

Ilustración 4.71 Método que obtiene información paginada.

El método mostrado en la ilustración 4.71 forma parte del controlador llamado ***RevisionController*** el cual permite obtener una cierta cantidad de registro con la finalidad de no saturar la base de datos consumir todos los registros en una sola consulta. Esta técnica llamada paginación recibe dos parámetros (ver indicador 1), siendo el primero la cantidad de registros que se requieren y el segundo la cantidad de páginas como si de un libro se tratara, es decir, como si se necesitara consultar un libro, pero en lugar de obtener todas las hojas del libro, solo se pidieran partes del libro para leer. Una vez que se reciben los parámetros en el método se hace una validación para verificar que los valores de los parámetros no sean menores que cero, posteriormente si la validación es falsa se procede a ejecutar el método ***_revision.GetAllPageAsync(count, pages)*** (ver indicador 2) la lógica del método se encuentra en el repositorio genérico tratado en el Sprint 1, el resultado que arroja el método es una lista de registros de la clase de tipo **Revision**, la siguiente instrucción que ejecuta el método es la de mapeo y por último cuenta la cantidad de registro que existen en la base de datos de tipo **Revision** y muestra los resultados al usuario (ver indicador 3) .

En la ilustración 4.72 se muestra la implementación que hace posible la obteniendo de un registro cuando se busca por el identificador único que lo caracteriza, haciendo uso del id que se pasa por parámetro al método y que se valida (ver indicador 1) para verificar que no es un valor nulo, se hace la búsqueda comparando el id que se pasa por parámetro contra todos los valores que existen en la base de datos, una vez que internamente la base de datos encuentra el registro que se busca, se devuelve y se almacena en una variable llamada **revision** (ver indicador 2), posiblemente no se encuentre ningún registro es por eso que se hace una segunda validación para verificar que el valor extraído de la base de datos no es nulo, si el valor llega a ser nulo, se envía un mensaje al usuario que el recurso buscado no fue encontrado (ver indicador 3), una vez que se encuentra un registro que cumple con la condición programada, los datos obtenidos se mapean al tipo de dato **RevisionDto**, es importante recordar que en toda la aplicación se convierte de un tipo de dato de dominio a un tipo de dato **Dto** (utilizado para mostrar datos al usuario final) con la finalidad de que los modelos de dominio no puedan ser alterados, por último los datos mapeados son enviados al usuario que realizo la operación de búsqueda para su maquetación (ver indicador 5).

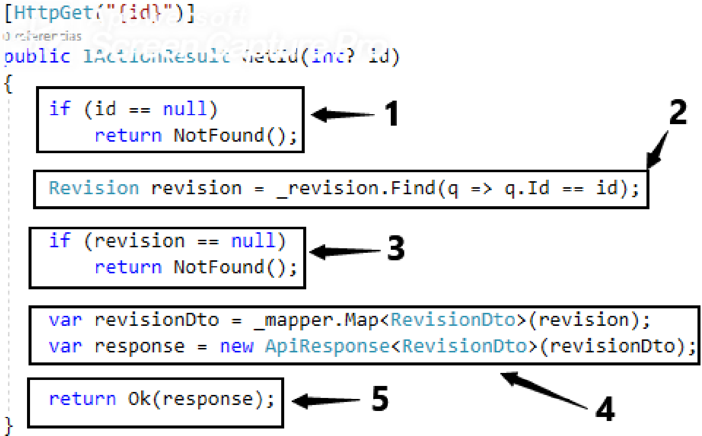


Ilustración 4.72 Obtener un registro con el identificador único de la clase que lo implementa.

En la ilustración 4.73 se muestra la funcionalidad que permite crear una revisión, dicha funcionalidad se encuentra declarada dentro del controlador ***RevisionController,*** el método se llama ***Post***, el cual permite insertar un registro en la base de datos. Lo primero que se

realiza es la aceptación de datos de tipo ***RevisionDto*** que son enviados con el formato de tipo **Json** (ver indicador 1), una vez que los datos son recibidos se mapean al tipo de dato **Revisión** y se almacenan los datos en la variables **revision** (ver indicador 2), posteriormente se realiza una lógica sencilla que permite asignar la fecha actual en la que se está insertando el registro y el campo periodo le asigna un formato AGO-DIC o ENE-JUN concatenando el año según sea el caso, si estamos en el mes mayor a seis, es decir, mayor a junio el periodo será AGO-DIC y sino el periodo será ENE-JUN (ver indicador 3).

```
[HttpPost]
0 referencias
public async Task<IActionResult> Post([FromBody] RevisionDto revisionDto)
{
    var revision = _mapper.Map<Revision>(revisionDto);

    revision.FechaRegistro = DateTime.UtcNow;
    string peridodo = DateTime.UtcNow.Month > 6 ? "AGO-DIC" : "ENE-JUN";
    revision.Periodo = peridodo + DateTime.UtcNow.Year;

    await _revision.AddAsync(revision);
    revisionDto = _mapper.Map<RevisionDto>(revisionDto);

    var response = new ApiResponse<RevisionDto>(revisionDto);

    return Ok(response);
}
```

Ilustración 4.73 Insertando un registro en el controlador RevisionController.

Cuando el periodo es asignado se procede a insertar los datos que en usuario envío (ver indicador 4), posteriormente se hace el mapeo de datos y por último la información guardada se envía al usuario para confirmar que los datos fueron guardados con éxito.

```
public class Revision
{
    3 referencias
    public int Id { get; set; }
    2 referencias
    public DateTime FechaRegistro { get; set; }
    2 referencias
    public string Periodo { get; set; }
    2 referencias
    public string UsuarioId { get; set; }
    1 referencia
    public bool Estado { get; set; }
    2 referencias
    public int DeptoId { get; set; }
    2 referencias
    public int AreaId { get; set; }

    2 referencias
    public Area Departamento { get; set; }
    1 referencia
    public ICollection<Ticket> Revision2Tickets { get; set; }
    1 referencia
    public Usuario Usuario { get; set; }
    2 referencias
    public Area Area { get; set; }
}
```

Ilustración 4.74 Clase que almacena datos de las revisiones.

Los datos almacenados en la base de datos de las listas de verificaciones de infraestructura y equipo son las mostradas en la ilustración 4.74. Para corroborar los datos se puede revisar la lista de verificaciones en los **Anexos** de este trabajo de tesis.

```
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> Put(int id, RevisionDto revisionDto)
{
    if (id != revisionDto.Id)
        return NotFound();

    var revision = _mapper.Map<Revision>(revisionDto);
    var result = await _revision.UpdateAsync(revision, id);
    revisionDto = _mapper.Map<RevisionDto>(result);

    var response = new ApiResponse<RevisionDto>(revisionDto);

    return Ok(response);
}
```

Ilustración 4.75 Método para actualizar listas de verificación.

El método mostrado en la ilustración 4.75 es muy utilizada, debido a que permite actualizar un registro, en esta caso la actualización de una lista de verificación, el método que se implementó recibe dos parámetros los cuales son el identificador único del registro y los datos que se desean actualizar (ver indicador 1), una vez que se reciben los parámetros se realiza una validación para ver si el identificador no es nulo, ya que se corrobora que no es valor nulo, se hace la inserción de los datos recibidos (ver indicador 3) y se envía el resultado al usuario que realizó la operación de actualización.

La funcionalidad de búsqueda se muestra en la ilustración 4.76, el método ***search*** recibe un parámetro que corresponde al periodo (ver indicador 1), posteriormente se hace una validación para verificar que el termino de búsqueda no sea nulo (ver indicador 2), una vez que se verifica que el valor recibido no es nulo, se hace la búsqueda comparando el parámetro de búsqueda con la propiedad de la base de datos de la tabla Revision (ver indicador 3), por último el resultado obtenido se envía al usuario para su correcta maquetación.

```
[HttpGet("search/{search}")]
0 referencias
public async Task<IActionResult> Search(string search)
{
    if (string.IsNullOrEmpty(search))
    {
        return NotFound();
    }

    var revision = await _revision.FindByAsync(x => x.Periodo == search);
    var revisionDto = _mapper.Map<IEnumerable<RevisionDto>>(revision);

    var response = new ApiResponse<IEnumerable<RevisionDto>>(revisionDto);

    return Ok(response);
}
```

Ilustración 4.76 Método para la búsqueda de una revisión por periodo.

La funcionalidad para la eliminación de un registro de tipo Revisión de la base de datos de la aplicación, se muestra en la ilustración 4.77. La operación de eliminar solo se utiliza cuando la tabla en donde se almacenan los datos de la revisiones de infraestructuras no tiene información almacenada, debido a que si en la tabla existieran registro y estos registro están relacionados con otra tabla podría existir un problema de inconsistencia al eliminar datos de una tabla y de otra no, por tal motivo la funcionalidad para eliminar registro no se implementa

como opción para el usuario final que hace uso de la aplicación, sin embargo se hace una explicación del funcionamiento del método para eliminar para futuras implementaciones del lado del cliente en la aplicación.

```
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> Delete(int? id)   ← 1
{
    if (id == null)                                 ← 2
        return NotFound();

    Revision revision = _revision.Query().FirstOrDefault(q => q.Id == id);   ← 3

    if (revision == null)                           ← 4
        return NotFound();

    var result = await _revision.DeleteAsync(revision);   ← 5

    var revisionDto = _mapper.Map<RevisionDto>(result);
    var response = new ApiResponse<RevisionDto>(revisionDto);

    return Ok(response);
}
```

Ilustración 4.77 Método para eliminar registros.

En la ilustración 4.77 se puede observar que el método recibe un parámetro de tipo entero (ver indicador 2), una vez que el parámetro es recibido se hace una evaluación para verificar que el valor recibido no sea nulo (ver indicador 2), posteriormente con la instrucción ***_revision.Query().FirstOrDefault(q => q.Id == id)*** se realiza una búsqueda del registro que se desea eliminar, si el registro no es encontrado se envía un mensaje al usuario que el elemento que desea eliminar es un recurso que no se encuentra en la base de datos (ver indicador 4), por último una vez que el recurso es encontrado se ejecuta la eliminación del recurso (ver indicador 5), es importante mencionar que si el registro que se desea eliminar tiene relación con otro registro de una tabla hija, no sé ejecutará la eliminación del registro.

4.7.3 Back-End: Implementación de las funcionalidades para la gestión de anomalías encontradas en las infraestructuras y equipos

Las funcionalidades para la gestión de anomalías encontradas en las infraestructuras y equipos son cuatro, las cuales son crear una anomalía, actualizar, buscar por id y listar todas las anomalías que se encuentran en la base de datos. En la ilustración 4.78 se muestra la clase que hace posible almacenar en memoria temporal una anomalía cuando se ejecutan los métodos mencionados anteriormente (ver indicador 1), el modelo de dominio cuenta con un identificador primario que se genera aleatoriamente (ver indicador 2), así como un conjunto de propiedades que permiten almacenar un folio, un número consecutivo, un periodo, la fecha en que se reporta la anomalía, el identificador del plan de trabajo al cual pertenece la anomalía (ver indicador 3), además del proveedor y el servicio que se reporta (ver indicador 4) y por último el identificador del agente que se asigna para que dé solución al problema reportado (ver indicador 5).

En la clase ***Ticket*** existen más propiedades además de la ya mencionadas, las cuales permiten relacionar los registros como si de una tabla de base de datos se tratará, en este apartado solo se mencionan las características más relevantes de la clase.

```
public class Ticket
{
    3 referencias
    public Guid Id { get; set; }
    3 referencias
    public string Folio { get; set; }
    5 referencias
    public int Consecutivo { get; set; }
    0 referencias
    public string Periodo { get; set; }
    1 referencia
    public DateTime FechaReporte { get; set; }
    2 referencias
    public int ProblemaId { get; set; }
    1 referencia
    public string Descripcion { get; set; }
    2 referencias
    public string UsuarioId { get; set; }

    // NOTA: ATENDIDO Y SOLUCIÓN PUEDEN IR JUNTOS
    0 referencias
    public bool Atendido { get; set; }
    2 referencias
    public int? PlanId { get; set; }
    0 referencias
    public bool Interno { get; set; }
    2 referencias
    public int? ProveedorId { get; set; }
    2 referencias
    public int? ServicioId { get; set; }
    1 referencia
    public string AgenteId { get; set; }
    0 referencias
    public DateTime FechaProgramada { get; set; }
```

Ilustración 4.78 Clase Ticket que permite almacena una anomalía.

El método que permite crear una anomalía se muestra en la ilustración 4.79, dicho método se encuentra declarado en el controlador ***TicketController*** al cual se accede desde una aplicación angular que haciendo uso de las peticiones ***HTTP***. El método ***Post*** (ver ilustración 4.79) recibe los datos enviados desde un formulario (ver indicador 1), posteriormente se hace una consulta a la base de datos para extraer el ultimo registro que se generó en el campo consecutivo de la tabla ***Ticket*** esto con la finalidad de obtener el ultimo folio e incrementarle una unidad y generar el nuevo folio, es decir, si no hay ninguna anomalía en la base de datos se genera un folio ***REQ1*** y el campo ***Consecutivo*** se le asigna el valor de uno (ver indicador 6) y si el campo consecutivo ya tiene un valor, se genera un folio con la palabra ***REQ*** más el nuevo valor que se le concatena (ver indicador 7), la propiedad periodo se genera obteniendo el mes actual y evaluando si es mayor que seis, es decir, si es mayor que junio si la validación es correcta se asigna la constante AGO-DIC más el año, de lo contrario se asigna a la propiedad periodo la constante ENE-JUN más el año (ver indicador 3), las propiedades ***planId***, ***proveedorId*** y ***servicioId*** se les asigna el valor nulo, debido a que cuando se crea una anomalía aún no se sabe el plan al que pertenece, ni el proveedor que resolverá la anomalía reportada, ni el servicio que se brindará (ver indicador 4), una vez que toda la lógica anteriormente explicada se resolvió se procede a almacenar en la base de datos la anomalía (ver indicador 8) y por último se notifica al usuario que la anomalía fue registrada (ver indicador 9).

```
[HttpPost]
0 referencias
public async Task<IActionResult> Post(TicketDto ordenDto)
{
    // Obtener el ultimo ID Generado
    var ticketConsecutivo = _orden.GetAll().OrderByDescending(t => t.Consecutivo)
        .FirstOrDefault();
    // FORMATO DE FOLIO ==>> REQ+NUM

    ordenDto.Id = Guid.NewGuid();
    string peridodo = DateTime.Now.Month > 6 ? "AGO-DIC" : "ENE-JUN";
    ordenDto.Periodo = peridodo + DateTime.Now.Year;

    ordenDto.PlanId = null;
    ordenDto.ProveedorId = null;
    ordenDto.ServicioId = null;

    var orden = _mapper.Map<Ticket>(ordenDto);
    orden.FechaReporte = DateTime.Now;

    if (ticketConsecutivo == null)
    {
        orden.Folio = "REQ1";
        orden.Consecutivo = 1;
    }
    else
    {
        orden.Folio = "REQ" + (ticketConsecutivo.Consecutivo + 1);
        orden.Consecutivo = ticketConsecutivo.Consecutivo + 1;
    }

    await _orden.AddAsync(orden);
    ordenDto = _mapper.Map<TicketDto>(ordenDto);

    var response = new ApiResponse<TicketDto>(ordenDto);

    return Ok(response);
}
```

Ilustración 4.79 Método que permite registrar una anomalía.

La actualización de una anomalía se lleva acabo cuando se requiere modificar un campo que fue introducido erróneamente, para tal caso se programó el método mostrado en la ilustración 4.80. El método ***Put*** recibe dos parámetros, uno de tipo ***Guid*** y otro de tipo ***TicketDto*** (ver indicador 1), una vez que los parámetros son recibidos se hace una validación en la que se verifica si el identificador primario es igual que los datos que son enviados por el formulario (ver indicador 2), cuando se comprueba que los datos son iguales se procede a mapear los datos y se genera la actualización de los registros (ver indicador 3) y posteriormente se notifica al usuario que la actualización se generó satisfactoriamente (ver indicador 4).

```
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> Put(Guid id, TicketDto ordenDto)
{
    if (id != ordenDto.Id)
        return NotFound();

    var orden = _mapper.Map<Ticket>(ordenDto);
    var result = await _orden.UpdateAsync(orden, id);
    ordenDto = _mapper.Map<TicketDto>(result);

    var response = new ApiResponse<TicketDto>(ordenDto);

    return Ok(response);
}
```

Ilustración 4.80 Método que permite actualizar una anomalía

Otra funcionalidad programada en el controlador ***TicketController*** es la mostrada en la ilustración 4.81, la cual permite recibir dos parámetros (ver indicador 1) a los cuales se les aplica una validad para saber si son negativos (ver indicador 2), una vez que se verifica que no son negativos se procede a llamar al método ***_orden.GetAllPageAsync(count, pages)*** (ver indicador 3) y se hace un conteo de todos los registros que están almacenados en la base de datos (ver indicador 4) y por último se envía una notificación al usuario (ver indicador 5) para informarle que la operación se realizó con éxito.

```
[HttpGet("{count}/{pages}")]
public IActionResult GetAllPaged(int count, int pages)
{
    if (count <= 0 || pages <= 0)
        return NotFound();

    var ordenes = _orden.GetAllPageAsync(count, pages);
    var ordenesDtos = _mapper.Map<IEnumerable<TicketDto>>(ordenes);

    var totalOrdenes = _orden.Count();

    var response = new ApiResponse<IEnumerable<TicketDto>>(ordenesDtos, totalOrdenes);

    return Ok(response);
}
```

Ilustración 4.81 Método que permite obtener un conjunto de anomalías.

El último método que se implementó en el controlador ***TicketController*** es el de buscar por id, el cual recibe el identificador de la anomalía (ver indicador 1) y se verifica si el valor es nulo, si el valor no es nulo (ver indicador 2), se ejecuta una operación de búsqueda (ver indicador 3) y una vez que se obtiene el registro buscado se valida si lo que la base de datos retorno no es nulo (ver indicador 4) y si el resultados contiene información de la anomalía buscada se retorna dicha información al usuario que realizó la operación de búsqueda (ver indicador 5)

```
[HttpGet("{id}")]
public async Task<IActionResult> GetId(Guid id)
{
    if (id == null)
        return NotFound();

    //Ticket orden = _orden.Find(q => q.Id == id);
    Ticket orden = await _orden.GetTicketId(id);

    if (orden == null)
        return NotFound();

    var ordenDto = _mapper.Map<TicketDto>(orden);
    var response = new ApiResponse<TicketDto>(ordenDto);

    return Ok(response);
}
```

Ilustración 4.82 Método que permite buscar una anomalía.

4.7.4 Front-End: Implementación de las vistas para mostrar las listas de verificación creadas

Una vez que las verificaciones son registradas en la base de datos, se necesita mostrar cierta información de cada una de las listas de verificación, una tabla en HTML es una forma que ayuda a navegar entre la información sin tener que consultar registro por registro. La ilustración 4.83 muestra como las tablas permiten visualizar ciertos registros de las verificaciones (ver indicador 1), si en el back-end no existieran registros se mostraría un mensaje notificándolo (ver indicador 2).

```
  <table class="table">
    <tbody>
      <tr>
        <th class="text-center">ÁREA</th>
        <th class="text-center">PERIODO</th>     <-- 1
        <th class="text-center">ESTADO</th>
        <th></th>
      </tr>
      <tr *ngIf="checklists.length === 0">
        <th class="text-center"
            colspan="4">                           <-- 2
          No se encontraron registro...
        </th>
      </tr>
      <tr *ngFor="let checklist of checklists">    <-- 3
        <td>{{ checklist.area['descripcion'] }}</td>
        <td class="text-center">{{ checklist.periodo }}</td>
        <!--<td>{{ checklist.departamento['descripcion'] }}</td>-->
        <td class="text-center">
          <p *ngIf="checklist.estado == true"
             class="badge bg-green">AUTORIZADA</p>
          <p *ngIf="checklist.estado == false"
             class="badge bg-blue">SIN AUTORIZAR</p>
        </td>
        <td class="text-center">...</td>
      </tr>
    </tbody>
  </table>
</div>
<div class="box-footer clearfix">                  <-- 4
  Total de verificaciones registradas ({{ totalChecklists }})
  <ul class="pagination pagination-sm no-margin pull-right">
    <li (click)="getPaged(-1)"
        class="pointer"><a>Anterior</a></li>
    <li class="pointer"><a>{{ page }} de {{ totalPages }}</a></li>
    <li (click)="getPaged(1)"
        class="pointer"><a>Siguiente</a></li>
  </ul>
</div>
```

Apowersoft Screen Capture Pro

Ilustración 4.83 Código HTML utilizado para mostrar en pantalla una lista de verificación.

Si los registros extraídos de la base de datos con el método de la ilustración 4.85 son más de diez, con el mecanismo llamado paginación se muestran de ocho en ocho (ver indicador 3 y 4) para no saturar la memoria temporal de la aplicación.

```html
<td class="text-center">
  <div class="btn-group">
    <button type="button"
            class="btn btn-default btn-flat">
      Acción
    </button>
    <button type="button"
            class="btn btn-default btn-flat dropdown-toggle"
            data-toggle="dropdown">
      <span class="caret"></span>
      <span class="sr-only">Toggle Dropdown</span>
    </button>
    <ul class="dropdown-menu"
        role="menu">
      <li>
        <a [routerLink]="['/anomalias', checklist.id]"
           class="pointer">
          <i class="fa fa-check-square-o"></i> Anomalía
        </a>
      </li>
      <li>
        <a (click)="getChecklist(template, checklist.id)"
           class="pointer">
          <i class="fa fa-pencil-square-o"></i> Editar
        </a>
      </li>
    </ul>
  </div>
</td>
```

Ilustración 4.84 Código HTML que muestra las opciones que se pueden ejecutar en una verificación.

En una lista de verificación se puede ejecutar dos tareas, una de ellas es agregar una anomalía a la lista de verificación (ver indicador 1) y la otra es actualizar una lista de verificación, ya sea porque los datos están erróneos o porque se requiera cambiar es estado.

```typescript
getChecklists() {
  this._checklistService.getPaged(this.page)
    .subscribe(checklists => {
      this.checklists = checklists.data;
      this.totalChecklists = checklists.count;
      this.countPages();
      this.cargando = false;
    });
}
```

Ilustración 4.85 Método utilizado para extraer las verificaciones del back-end.

El método ***getChecklist*** mostrado en la ilustración 4.85 permite obtener del back-end los registros paginados de las listas de verificación (ver indicador 1) y almacenarlos en una variable temporal (ver indicador 2) para mostrar los resultados en la tabla explicad en la ilustración 4.83.

El formulario que se utiliza para crear una lista de verificación, también se utiliza para actualizarlo, esto debido a que los campos que son creados al registrar una lista de verificación son los mismo que se pueden actualizar según lo analizado al diseñar el módulo,

sin embargo, para actualizar el registro seleccionado (ver ilustración 4.84 indicador 2), cuando se cargan los datos en el formulario se utiliza una lógica diferente del lado de front-end como se muestra en la ilustración 4.90. Cuando se manda a llamar al método que carga el registro que se desea actualizar, se le pasa como parámetros la plantilla del formulario con código ***HTML*** y el identificador del registro que se actualizará con la finalidad de ejecutar el método ***getRevision(id)*** que realiza una búsqueda en la base de datos y si encuentra el registro lo asigna a la variable ***selectedChecklist*** y posteriormente carga el formulario con los datos obtenidos del back-end con la instrucción ***formChecklist.setValue({ deptoId, areaId, estado })*** (ver indicador 3), por último se abre el formulario y se muestra en pantalla en el navegador web (ver indicador 4).

```
getChecklist(template: TemplateRef<any>, id: number) {   ← 1
  this._checklistService.getRevision(id)   ← 2
    .subscribe((checklists: Revision) => {
      const { deptoId, areaId, estado } = checklists;
      this.selectedChecklist = checklists;
      this.formChecklist.setValue({ deptoId, areaId, estado });
    });                                                   ↑ 3

  this.openModal(template);  ← 4
}
```

Ilustración 4.86 Lógica para carga un registro a actualizar.

4.7.5 Front-End: Implementación del formulario para la creación de una lista de verificación

El formulario mostrado en la ilustración 4.86 es utilizado para las operaciones de creación y actualización de listas de verificación (ver indicador 1), el formulario es una plantilla HTML que ayuda al usuario final a tener una interacción con ventanas en el navegador web para facilitar el uso de la aplicación. Al presionar el botón para crear una lista de verificación se muestran las propiedades que debe introducir el usuario (ver indicador 3) y una vez que el usuario ingresa correctamente los datos al formulario oprime el botón de guardar (ver indicador 4) se ejecuta la función **save** (ver indicador 2) llama al método save que envía los datos introducidos al back-end (ver sección 4.7.2) la cual realiza las operaciones CRUD.

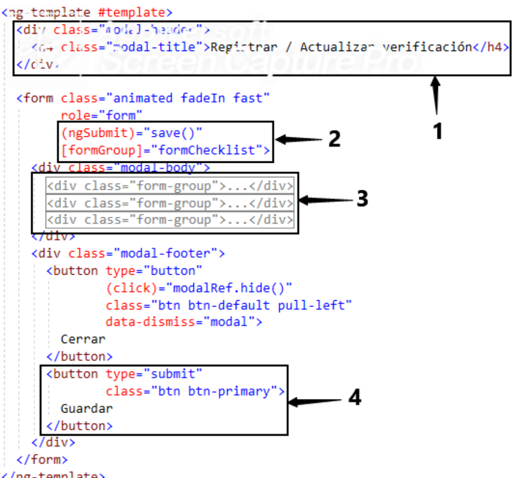


Ilustración 4.87 Formulario para la creación y actualización de listas de verificaciones.

En la ilustración 4.87 se muestra el método ***save*** que se ejecuta una vez que en el formulario se presiona el botón ***Guardar***, el método hace una validación del formulario, es decir, verifica que el formulario sea válido (que todos los campos estén llenados correctamente), ya que el formulario es válido se evalúa si la propiedad ***selectChecklist*** tiene un valor asignado, si la propiedad contiene un valor, es un indicador de que se desea actualizar un registro (ver indicador 2) y se ejecuta la instrucción de actualización (ver indicador 3), en caso de que ***selectChecklist*** no contenga ningún valor, se entiende que se desea crear un nuevo registro y se ejecuta la instrucción que asigna los datos contenidos en el formulario a un objeto de tipo ***Revision*** (ver indicador 4) para enviarlos al back-end para almacenar una nueva lista de verificación con los datos introducidos por el usuario en el formulario (ver indicador 5).

```
save() {
  if (!this.formChecklist.valid) {      ← 1
    return;
  }

  if (this.selectedChecklist) {         ← 2
    // actualizar
    const data =...
    this._checklistService.updateRevision(data)
      .subscribe(checklist => {
        this.selectedChecklist = undefined;   ← 3
        this.resetModalPopup();
      });

  } else {
    // crear
    this.selectedChecklist = new Revision(
      this.usuarioId,
      this.formChecklist.value.estado,        ← 4
      Number.parseInt(this.formChecklist.value.deptoId),
      Number.parseInt(this.formChecklist.value.areaId)
    );

    this._checklistService.createRevision(this.selectedChecklist)
      .subscribe(checklist => {
        this.selectedChecklist = undefined;
        this.resetModalPopup();
      });
  }
}// end of save                              5
```

Ilustración 4.88 Método para enviar los datos del formulario al back-end.

4.7.6 Front-End: Implementación del formulario para el registro de una anomalía

El formulario mostrador en la ilustración 4.89, se utiliza para operaciones de creación y actualización de las anomalías que se registran en una lista de verificación, el formulario cuenta con un método llamado **addAnomalia** (ver indicador 1) el cual es ejecutado cuando el usuario hace clic sobre el botón guardar del formulario. Los campos que se llenan con información es el problema (ver indicador 2) que se obtiene de los catálogos registros en las secciones anteriores, también el formulario pide una descripción del problema (ver indicador 3) y que se active o desactive un botón (ver indicador 3) que indique si se atendió en el momento la anomalía o se deja para programar en un plan de trabajo.

El formulario es el mismo que se utiliza para registrar una nueva anomalía o actualizar, eso es posible debido a que el método addAnomalia es llamado y utilizado para comprobar si existen datos en el formulario cuando se activa (ver indicador 5).

```
<div class="modal-dialog">
    <div class="modal-content">
        <div class="modal-header">
            <h3 class="modal-title">{{ anomalia !== null ? 'Editar anomalía' : 'Nueva anomalía '}}</h3>
        </div>
        <form (ngSubmit)="addAnomalia()"
              [formGroup]="anomaliaForm">
          <div class="modal-body">
            <div class="form-group"
                 [ngClass]="{'has-error': descripcionNoValido }">
                <label>Problema:</label>
                <select formControlName="idProblema" class="form-control">…
                </select>
                <span class="help-block" *ngIf="descripcionNoValido">El campo Problema es obligatorio</span>
            </div>

            <div class="form-group"
                 [ngClass]="{'has-error': problemaNoValido }">
                <label class="control-label">Descripción:</label>
                <textarea class="form-control" …
                </textarea>
                <span class="help-block" *ngIf="problemaNoValido">El campo Descripción es obligatorio</span>
            </div>

            <div class="form-group">
                <div class="icheck-primary">
                    <input type="checkbox" …
                           id="atendido">
                    <label for="atendido">Atendido</label>
                </div>
            </div>
        </div>
        <div class="modal-footer">…
        </div>
         </form>
    </div>
</div>
```

Ilustración 4.89 Formulario para registrar una anomalía.

La lógica del formulario mostrada en la ilustración 4.90, esta acoplado al formulario debido a que como se mencionó se utiliza para registrar y actualizar, dichas operaciones conllevan una lógica en la que se deben de validad y cargar información de la base de datos (ver indicador 1) que valida si al cargar el formulario la variables **animaliaForm** está vacía quiere decir que no hay datos para actualizar y se trata de una creación de registro y cuando se llenan los campos desde el formulario al presionar el botón guardar se almacenan los datos en la constante **data** (ver indicador 2) y se manda la información a la base de datos y se notifica haciendo uso de una alerta en el formulario que el registro fue creado con éxito (ver indicador 3).

```
    // });
}

addAnomalia() {

    if(this.anomaliaForm.invalid) {
        return Object.values(this.anomaliaForm.controls).forEach(control => {
            control.markAsTouched();
        });
    }

    if(this.revision) {
        this.editAnomalia();
    } else {

        const data = {
            id: 0,
            creadoPorId: this.usuario.id,
            folio: 'MAT-1',
            fechaRegistro: new Date,
            atendido: this.anomaliaForm.value.atendido,
            descripcion: this.anomaliaForm.value.descripcion,
            idProblema: Number.parseInt(this.anomaliaForm.value.idProblema),
            idDepartamento: this.usuario.idDepartamento,
            idRevision: this.idRevision
        }

        this.apiAnomalia.add(data).subscribe(response => {
            if(response.exito === 1) {
                this.bsModalRef.hide();
                this.toaster.success('Anomalía insertado con éxito', null);
            }
        });
    }
}
```

Ilustración 4.90 Lógica del formulario del registro de una anomalía.

Las funcionalidades explicadas en la sección son las de mayor relevancia en este módulo, en la siguiente sección se explicará el desarrollo del back-end y front-end del módulo de plan de trabajo. Es importante recordar que en el siguiente capítulo el cual trata de resultados, se mostrarán imágenes de los formularios y tablas descritas en este capítulo, así como la secuencia de pasos que el usuario final debe de seguir para utilizar las funcionalidades que en este capítulo se expusieron.

4.8 Sprint 5: Implementación del módulo de planes de trabajo

En la tabla 4.9 se presenta la planeación del Sprint 5, donde cada historia de usuario se desglosa en tareas que tiene un tiempo estimado para su realización, acordado por el equipo ***Scrum***, así como las fechas de inicio y finalización del Sprint.

Tabla 4.9 Planeación del Sprint 5.

Sprint 5				
Objetivo: El objetivo del Sprint es que los usuarios con el rol de administrador y coordinador puedan gestionar la información de los planes de trabajo.				
Fecha de inicio		Fecha de finalización		N°. de semanas
08-10-2019		20-12-2019		10
Historias de usuario	**Tareas del Sprint 5**	**Fecha de inicio**	**Fecha de finalización**	**Horas de trabajo invertidas**
HU7, HU8, HU9 y HU10	Configuración del controlador del módulo para acceder a la información de la base de datos.	08-10-2019	22-10-2019	62
	Implementación de las funcionalidades para la gestión de planes de trabajo.	23-10-2019	19-11-2019	96
	Implementación de los formularios para la gestión de los planes de trabajo.	20-11-2019	03-12-2019	74
	Implementación del formulario para la generación de formato pdf del plan de trabajo.	04-12-2019	12-12-2019	145
Total de horas de trabajo invertidas				377

Como en el sprint anterior las tareas realizadas en la historia de usuario serán presentadas en dos fases **Back-End** y **Front-End**.

Este quinto Sprint trata sobre la implementación del módulo de plan de trabajo, proceso que se lleva al inicio del semestre (ver capítulo 1) que consiste en registrar un plan y asignarle actividades (anomalías a reparar) que se tiene que realizar durante el semestre en curso. El desarrollo del Sprint se desglosa en dos fases como se ha ido haciendo en los Sprints anteriores, la primera fase es la del back-end, en la cual se muestra la lógica para persistir la información haciendo uso de los controladores (ver capítulo 2) y la segunda fase es la de front-end, en donde se explica la forma en que se muestra la información al usuario final haciendo uso de formularios en el navegador web.

4.8.1 Back-End: Configuración del controlador del módulo para acceder a la información de la base de datos

Para realizar las operaciones CRUD de las funcionalidades del módulo es necesario crear un vínculo entre la aplicación que muestra los datos al usuario final y la base de datos que los almacena, es por eso que haciendo uso de los controladores se mantiene una relación del backend con el frontend. Como se observa en la ilustración 4.91, el controlador que brindará el servicio CRUD para gestionar Planes de trabajo (ver indicador 1 y 2) se declara de tal forma que pueda ser usado y expuesto al frontend para su uso en cada petición realizada duran la creación, actualización y consulta de algún registro.

```
[Route("api/[controller]")]      ← 1
[ApiController]
0 referencias                    ← 2
public class PlanController : ControllerBase
```

Ilustración 4.91 Controlador plan de trabajo.

Cada uno de los controladores de la aplicación está vinculado a un modelo de dominio que como se muestra en la ilustración 4.92, donde se muestra la clase Plan que contiene seis campos los cuales representan la información que se re quiere gestionar en la base de datos.

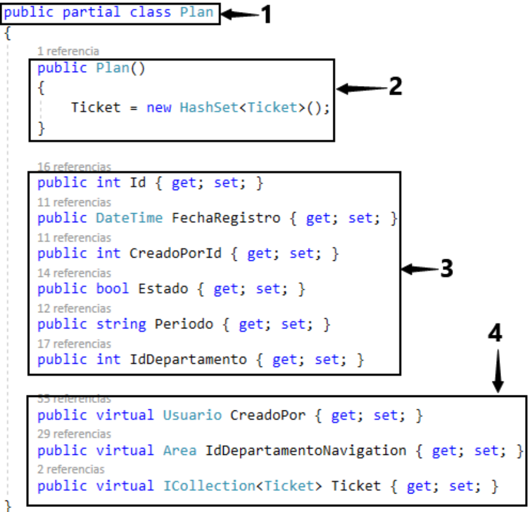


Ilustración 4.92 Modelo de dominio del plan de trabajos.

La clase consta de un nombre (ver indicador 1), de un constructor que inicializa a valores nulos las tablas con las clases con las que se relaciona (Ticket), también de campos que almacenan información relevante del dominio del negocio que se representa (ver indicador 4) y las relaciones con otras clases del dominio del negocio (ver indicador 4).

Una vez que el controlador esta creado e inicializado (configurado) para ejecutar operaciones contra la base de datos, se comienzan a crear los métodos que permiten obtener o insertar información.

4.8.2 Back-End: Implementación de las funcionalidades para la gestión de planes de trabajo

Los métodos que se implementaran en esta sección son los más comunes en una aplicación, dichos métodos permiten registrar, actualizar, consultar u obtener un registro por un identificador primario.

Para registrar un plan de trabajo, desde el backend haciendo uso de un formulario se llama al método que se muestra en la ilustración 4.93, el cual recibe como parámetro un modelo **DTO** que contiene la información que se envía desde el formulario para ser registrado (ver indicador 1).

```
[HttpPost]
0 referencias
public IActionResult Add(PlanRequest model)        <- 1
{
    Respuesta respuesta = new Respuesta();          <- 2
    respuesta.Exito = 0;
    try
    {
        using (SoportecContext db = new SoportecContext())...
    }
    catch (Exception ex)
    {
        respuesta.Mensaje = ex.Message;
    }                                               <- 3
    return Ok(respuesta);                           <- 4
}
```

Ilustración 4.93 Método para registrar un plan de trabajo.

Una vez que se recibe la información en el método, se crea un objeto del tipo Respuesta (ver indicador 2), el cual almacenará la respuesta que será enviada al usuario final, posteriormente se ejecuta la funcionalidad para crear el plan de trabajo (ver indicador 3) y se envía la respuesta el usuario que ejecuto el método (ver indicador 4).

Internamente lo que sucede en el método se observa en la ilustración 4.94, como se observa se hace una operación ternaria que evalúa el mes actual para saber qué periodo debe de registrar en el plan, si el mes actual es mayor que seis, quiere decir que el registro se está realizando entre agosto diciembre y si es menor que seis, el plan se está creando entre enero

y junio, posteriormente los datos recibidos del formulario se asignan al objeto **oPlan** (ver indicador 2), que finalmente los almacenará en la base de datos haciendo uso de la instrucción **db.SaveChange** (ver indicador 3).

```
string peridodo = DateTime.UtcNow.Month > 6 ? "AGO-DIC" : "ENE-JUN";
Plan oPlan = new Plan();
oPlan.Id = model.Id;
oPlan.FechaRegistro = DateTime.Now;
oPlan.CreadoPorId = model.CreadoPorId;
oPlan.Estado = model.Estado;
oPlan.Periodo = peridodo + DateTime.UtcNow.Year;
oPlan.IdDepartamento = model.IdDepartamento;
db.Plan.Add(oPlan);
db.SaveChanges();
```

Ilustración 4.94 Registro de un plan de trabajo.

Para ejecutar el método de registro el usuario debe de enviar la información a través de un método http con el verbo **Post**.

El segundo método que se programó para el controlador **PlanController** es el método para actualizar un plan de trabajo como se observa en la ilustración 4.94, el método recibe un modelo DTO (ver indicador 1) haciendo uso de un verbo HttpPut para actualizar, posteriormente se abre una conexión a la base de datos (ver indicador 2) y se comienza a asignar los datos recibidos por parámetro con los objetos **oPlan** los cuales irán directo a la base de datos (ver indicador 3), después de hacer la asignación se hace una guardado en la base de datos con la instrucción **db.SaveChange** (ver indicador 4) y por último se envía el resultado al usuario que solicitó la ejecución del método (ver indicador 5)

```
[HttpPut]
0 referencias
public IActionResult Edit(PlanRequest model)
{
    Respuesta respuesta = new Respuesta();
    respuesta.Exito = 0;
    try
    {
        using (SoportecContext db = new SoportecContext())
        {
            Plan oPlan = db.Plan.Find(model.Id);
            oPlan.Id = model.Id;
            oPlan.FechaRegistro = model.FechaRegistro;
            oPlan.CreadoPorId = model.CreadoPorId;
            oPlan.Estado = model.Estado;
            oPlan.Periodo = model.Periodo;
            oPlan.IdDepartamento = model.IdDepartamento;
            db.Entry(oPlan).State = Microsoft.EntityFrameworkCore.EntityState.Modified;
            db.SaveChanges();
            respuesta.Exito = 1;
        }
    }
    catch (Exception ex)
    {
        respuesta.Mensaje = ex.Message;
    }

    return Ok(respuesta);
}
```

Ilustración 4.95 Actualización de un plan de trabajo.

El tercer método que se implementó en el controlador es el método que ayuda a buscar un registro (plan de trabajo) haciendo uso de un identificador primario (ver indicador 1) como se observa en la ilustración 4.96.

```
[HttpGet("{Id}")]
0 referencias
public IActionResult Get(int Id)
{
    Respuesta oRespuesta = new Respuesta();
    oRespuesta.Exito = 0;
    try
    {
        using (SoportecContext db = new SoportecContext())
        {
            Plan oPlan = db.Plan.Find(Id);
            PlanRequest planRequest = new PlanRequest();
            planRequest.Id = oPlan.Id;
            planRequest.FechaRegistro = oPlan.FechaRegistro;
            planRequest.CreadoPorId = oPlan.CreadoPorId;
            planRequest.Estado = oPlan.Estado;
            planRequest.Periodo = oPlan.Periodo;
            planRequest.IdDepartamento = oPlan.IdDepartamento;
            oRespuesta.Exito = 1;
            oRespuesta.Data = planRequest;
        }
    }
    catch (Exception ex)
    {
        oRespuesta.Mensaje = ex.Message;
    }

    return Ok(oRespuesta);
}
```


Ilustración 4.96 Método para buscar un plan por id.

El método mostrado en la ilustración 4.96, recibe un identificador usando un verbo de tipo Get, en el cual por url se pasa el identificador del registro que se desea buscar (ver indicador 1), posteriormente haciendo uso de **LINQ**, se ejecuta una búsqueda por **ID**, si el registro existe, será asignador al objeto **DTO** y devuelto al usuario (ver indicador 3 y 4), si no se devolverá un objeto DTO vacío y un mensaje con un valor de éxito igual a cero, indicando que la operación no tuvo éxito.

El método mostrado en la ilustración 4.97 permite obtener los registros de la base de datos de una manera paginada, es decir, muchas veces todos los registros de una tabla de la base de datos son obtenidos en una sola consulta, esta forma de operar conforme pasa el tiempo y la tabla se va llenando de registros se vuelve ineficiente, pues consume demasiada memoria.

```
}

[HttpGet("{count}/{pages}")]
0 referencias
public IActionResult GetPaged(int count, int pages)
{
```


Ilustración 4.97 obtener planes de trabajo.

El la ilustración 4.97 haciendo uso de un verbo HttpGet se recibe por parámetro dos valores, la cantidad de registros que desea obtener y la cantidad de páginas en las que se recibe la información (ver indicador 1).

```
List<PlanRequest> lst2 = db.Plan
.OrderBy(x => x.Id)
.Skip((pages - 1) * count)
.Include(x => x.IdDepartamentoNavigation)
.Include(x => x.CreadoPor)
.Where(x => x.IdDepartamento == idDepto && x.Estado == true)
.Select(d => new PlanRequest
{
    Id = d.Id,
    FechaRegistro = d.FechaRegistro,
    CreadoPorId = d.CreadoPorId,
    Usuario = new UserRequest...,
    Estado = d.Estado,
    Periodo = d.Periodo,
    IdDepartamento = d.IdDepartamento,
    Departamento = new AreaRequest...
}).Take(count).ToList();

respuesta.Exito = 1;
respuesta.Total = db.Plan.Count();
respuesta.Data = lst2;
```

Ilustración 4.98 Operación interna de la paginación.

Dentro del método **GetPaged**, se crea una lista del tipo Plan (ver indicador 1) y se hace uso de una fórmula para que cada vez que se consulte el método se brinque ciertos registros, esto con la finalidad de que no se devuelva al usuario los mismo registros que se consultaron anteriormente (ver indicador 2), una vez que los datos son filtrados y obtenidos se mapean tradicionalmente sin el uso de **AutoMapper** (ver indicador 3) y posteriormente se envían al usuario que ejecuta el método **GetPaged** (ver indicador 4).

```
[HttpGet("search/{search}")]
0 referencias
public IActionResult Search(string search)
{
    Respuesta respuesta = new Respuesta();
    respuesta.Exito = 0;

    try
    {
        var idDepto = int.Parse(User.Identity.Name);

        using (SoporteccContext db = new SoporteccContext())
        {
            List<PlanRequest> lst = db.Plan.Where(x => x.Periodo.Contains(search))
                .Include(x => x.IdDepartamentoNavigation)
                .Where(x => x.IdDepartamento == idDepto)
                .Select(d => new PlanRequest
                {
                    Id = d.Id,
                    FechaRegistro = d.FechaRegistro,
                    CreadoPorId = d.CreadoPorId,
                    Estado = d.Estado,
                    Periodo = d.Periodo,
                    Usuario = new UserRequest...,
                    Departamento = new AreaRequest...
                }).ToList();

            respuesta.Exito = 1;
            respuesta.Data = lst;
        }
    }
    catch (Exception ex)
    {
        respuesta.Mensaje = ex.Message;
    }

    return Ok(respuesta);
}
```

Ilustración 4.99 Método para buscar plan por periodo.

El último método implementado en el controlador llamado **Plan**, es el que permite hacer una búsqueda por periodo entre los planes de trabajo registrados como se observa en la ilustración 4.99. El método recibe un término de búsqueda haciendo uso del verbo HttpGet (ver indicador 1), después se hace un filtrado haciendo coincidir el campo de la clase Plan llamado **Periodo** con el termino de búsqueda (ver indicador 2), si se encuentra un registro se mapea tradicionalmente (ver indicador 3) y se devuelve el resultado, si el registro no es encontrado, se devuelve un objeto sin registro y con el campo éxito con un valor de cero, indicando que la búsqueda no tuvo éxito.

Los métodos explicados anteriormente, son los que contienen la lógica de negocios que existe detrás del módulo de plan de trabajo, a partir de esta lógica implementada del lado del backend se realizan inserciones, actualizaciones, búsquedas por id y búsquedas por periodo de los planes de trabajo. Los formularios utilizados por el usuario final para interactuar con la lógica de negocios se explicarán en la siguiente sección.

4.8.3 Back-End: Implementación de los formularios para la gestión de los planes de trabajo

A continuación, se explicará cada uno de los formularios utilizados para registrar un plan de trabajo, realizar una búsqueda y listar los planes. La ilustración 4.100, muestra el código que hace posible el listar todos los planes de trabajo registrados en la base de datos, como toda tabla de información se muestran los campos que indican a que registro se refiere cada dato mostrado (ver indicador 1), si al llamar el método **GetPaged** (ver ilustración 4.99) del lado del servidor que permite obtener registro paginados no devolviera información, se mostraría un mensaje indicando que no hay registros en la base de datos (ver indicador 2), si la base de datos devolviera información, serían mostrados uno por uno en la tabla (ver indicador 3), la tabla cuenta con dos opciones posible sobre cada registro, cambiar de estado e imprimir el plan de trabajo.

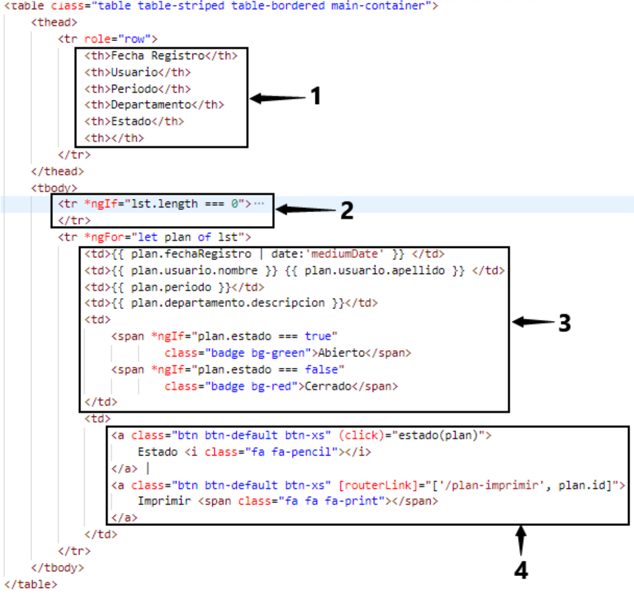


Ilustración 4.100 Código HTML para listar los planes de trabajo.

Como se observa en la ilustración 4.101, la opción estado ejecuta un método cuando se presiona el botón **Estado**, se pasa el plan obtenido del backend al método (ver indicador 1), después se hace una alternación del campo estado si es true (abierto) se pasa a estado false (cerrado) (ver indicador 2), esto con la finalidad de hacer el cambio de estado, posteriormente se envía el plan completo a la base de datos para actualizar los datos (ver indicador 3) y cuando el backend responde que la actualización se realizó con éxito, haciendo uso de una alerta se indican al usuario que ejecutó el método estado que el plan cambio de estado (ver indicador 4).

```
estado(plan: Plan) {
  if(plan.estado === true) {
    plan.estado = false;
  } else {
    plan.estado = true;
  }

  this.apiPlan.edit(plan).subscribe(response => {
    if(response.exito === 1) {
      this.toaster.info('Plan cambio de estado', null);
    }
  });
}
```

Ilustración 4.101 Método para cambiar de estado un plan de trabajo.

Como ya se mencionó anteriormente cada tabla o formulario contiene una lógica, y la lógica de la tabla que muestra los datos al usuario se observa en la ilustración 4.102.

```
getPlanes() {
  this.apiPlan.getPaged(this.page).subscribe(response => {
    this.lst = response.data;
    this.totalPlanes = response.total;
    this.countPages();
  });
}
```

Ilustración 4.102 Método para obtener los registros en la tabla.

El método llamado **getPlanes**, hace una llamada al backend para obtener diez registros (ver indicador 1), valor que es guardado en la variable **page**, posteriormente el backend response y se obtienen los datos y se almacenan en la variable **this.lst** y se muestran en la tabla cada uno de los registros obtenidos (ver indicador 2).

Las listas de planes de trabajo mostradas en una tabla son muy importantes, pero muchas veces lo usuarios requieren un plan de trabajo en específico, es por eso que se desarrolló la funcionalidad (ver ilustración 4.96) de obtener un registro haciendo uso de su id, en la ilustración 4.103 se observa el formulario que permite obtener el termino de búsqueda ingresado por el usuario y enviarlo al backend para que la base de datos busque y devuelva el registro que se desea.

El usuario al ingresar el periodo del plan, el formulario lo captura (ver indicador 1, ilustración 4.103) y lo envía el método **search** (ver ilustración 4.104).

```
</div>
<div class="col-sm-4">
    <div class="input-group">
        <input #input (keyup)="search(input.value)"
            type="text"
            name="message"
            placeholder="Buscar plan por periodo"
            class="form-control">
        <span class="input-group-btn">
            <button type="button" class="btn btn-default btn-flat">Buscar</button>
        </span>
    </div>
</div>
```

Ilustración 4.103 Formulario para realizar una búsqueda por periodo.

El método search recibe un valor de tipo string (ver indicador 1), que se válida para saber si no es un valor nulo o espacio en blanco (ver indicador 2) y posteriormente el termino de búsqueda es enviado al backend (ver indicador 3) el cual responde con el plan de trabajo que coincida en su campo **Estado** con el valor buscado (ver ilustración 4.99).

```
search(search: string) {
    if(search.length <= 0) {
        this.getPlanes();
        return;
    }

    this.apiPlan.search(search).subscribe(response => {
        this.lst = response.data;
    })
}
```

Ilustración 4.104 Método que realiza búsqueda.

De las funcionalidades importantes dentro del módulo para la gestión de planes de trabajo, hay una funcionalidad muy importante que permite alimentar la base de datos con nuevos registros, esta funcionalidad es la de agregar un nuevo plan, la ilustración 4.105 muestra el código HTML del botón que hace posible la inserción de información, el botón consta de un método llamado **openAdd**, que cuando es presionado por el usuario se manda información a la base de datos (ver indicador 1) para generar un nuevo plan (ver indicador 2).

```
</div>
<div class="col-lg-2">
    <a (click)="openAdd()" class="btn btn-success">
        <i class="fa fa-plus"></i> Nuevo Plan de Trabajo
    </a>
</div>
```

Ilustración 4.105 Botón para crear un nuevo plan de trabajo.

Cuando el usuario presiona el botón **Nuevo Plan de Trabajo** (ver ilustración 4.105) se ejecuta el método mostrado en la ilustración 4.106, que muestra un formulario (ver indicador 1) al usuario para que confirme que está de acuerdo en que se creará un nuevo plan de trabajo del departamento al cual pertenece el usuario, cuando el método se termina de ejecutar se hace una llamada el método **this.getPlanes** (ver ilustración 4.102) que actualiza los registros de la tabla que muestra todos los planes registrados (ver indicador 2).

```
openAdd() {
  this.bsModalRef = this.modalService.show(DialogPlanComponent);
  this.bsModalRef.onHidden.subscribe(result => {
    this.getPlanes();
  });
}
```

Ilustración 4.106 Método para agregar un nuevo plan

El método anterior llamado openAdd está ligado a un formulario HTML que lo ayuda a que el usuario haciendo uso de campos pueda ingresar la información para que sean enviados a la base de datos. La ilustración 4.107, muestra el código utilizado para que el usuario final pueda hacer clic en el botón que hace posible la generación de un nuevo plan de trabajo. El formulario mostrará la leyenda **Nuevo plan** o **Editar plan** dependiendo de si el objeto plan (ver ilustración 1) es nulo o no, también se muestra un mensaje (ver indicador 2) el cual da aviso al usuario que **Se genera un nuevo plan de trabajo**, el usuario tiene la opción en el formulario de cancelar la operación (ver indicador 3), también de aceptar el registro o edición del plan, todo dependerá de si el objeto plan (ver indicador 1) es nulo o no.

```
<div class="modal-dialog">
    <div class="modal-content">
        <div class="modal-header">
            <h3 class="modal-title">{{ plan !== null ? 'Editar plan' : 'Nuevo plan'}}</h3>
        </div>
            <div class="modal-body">
                <h4>Se generará un nuevo plan de trabajo</h4>
            </div>
            <div class="modal-footer">
                <button type="button" (click)="close()"
                        class="btn btn-default pull-left">Cerrar</button>
                <button type="button"
                    class="btn btn-primary"
                    (click)="plan !== null ? editPlan() : addPlan()">Registrar</button>
            </div>
    </div>
</div>
```

Ilustración 4.107 Código HTML del formulario para agregar planes de trabajo.

La ilustración 4.108 muestra el método que se ejecuta si el usuario decide crear un plan de trabajo, el objeto Plan (ver indicador 1) se llenará con el identificador del usuario que está creando el plan, el estado del plan, el periodo generado en el backend (ver ilustración 4.94), el departamento que crea el periodo y un identificador del plan (asignado del lado del backend).

```
addPlan() {
  const plan: Plan = { creadoPorId: this.usuario.id, estado: true, periodo: '',
                       idDepartamento: this.usuario.idDepartamento, id: 0 };
  this.apiPlan.add(plan).subscribe(response => {
    if(response.exito === 1) {
      this.bsModalRef.hide();
      this.toaster.success('Plan insertado con éxito', null);
    }
  });
}
```

Ilustración 4.108 Método que agrega un plan de trabajo.

El método editar mostrado en la ilustración 4.109, muestra cómo se obtiene los datos del formulario (ver indicador 1) y enviados backend (ver indicador 2), si el backend responde con una operación exitosa, se mostrará una alerta indicando que el plan fue editado con éxito.

```
editPlan() {
    const data = {
        id: this.plan.id,
        fechaRegistro: this.planForm.value.fechaRegistro,
        creadoPorId: this.planForm.value.creadiPorId,
        estado: this.planForm.value.estado,
        periodo: this.planForm.value.periodo,
        idDepartamento: Number.parseInt(this.planForm.value.idDepartamento)
    }

    this.apiPlan.edit(data).subscribe(response => {   ← 2
        if(response.exito === 1) {
            this.bsModalRef.hide();
            this.toaster.info('Plan editado con éxito', null);   ← 3
        }
    });
}
```

Ilustración 4.109 Método para editar un plan de trabajo.

Las funcionalidades explicadas en esta sección corresponden a las operaciones más comunes y utilizadas por los usuarios al gestionar los planes de trabajo. En la siguiente sección se explicará la generación del formato pdf del plan de trabajo, es importante mencionar que la generación del pdf se llena con los mismos métodos creados anteriormente en el backend de un plan de trabajo.

4.8.4 Implementación del formulario para la generación de formato pdf del plan de trabajo

El módulo plan de trabajo como se explicó en la sección anterior se encarga de gestionar las actividades que se realizaran durante el semestre en el que se crea el plan. Uno de los requerimientos de los usuarios que utilizaran la plataforma fue que se tenía que generar un pdf con la programación de las actividades obtenidas en las revisiones semestrales (lista de verificación). En la ilustración 4.110 se muestra la cabecera del formato generado de los planes de trabajo, el formato fue descargado de la página oficial del **TecNM**.

```
<table class="table table-bordered table-sm">
  <tr>
    <td rowspan="3">
      <img src="./assets/img/sgc_tecnm.jpg" class="img-sgc center" alt="Imagen de SGC-TECNM">
    </td>
    <td><b>Formato para la Orden de Trabajo de Mantenimiento</b></td>
    <td><b>Código: TecNM-AD-PO-001-03</b></td>
  </tr>
  <tr>
    <td rowspan="2"><b>Referencia a la Norma ISO 9001:2015 6.1, 7.1, 7.2, 7.4, 7.5.1, 8.1<br>Referencia a la Norma
      ISO 14001:2015 4.1, 6.1, 8.1, 8.2</b></td>
    <td><b>Revisión 0</b></td>
  </tr>
  <tr>
    <td><b>Página 1 de 2</b></td>
  </tr>
</table>
```

Ilustración 4.110 Cabecera del formato pdf de los planes de trabajo.

En la ilustración 4.111, se observa el título del formato (ver indicador 1), así como el semestre (ver indicador 2) y la fecha (ver indicador 3) en que se generó el formato.

```
<div class="cab-titulo">
  <div class="titulo text-center">
    <span ><b>PROGRAMA DE MANTENIMIENTO PREVENTIVO</b></span>   ← 1
    <br>
  </div>
  <div class="semestre">
    <span><b>Semestre: {{ plan.periodo }}</b></span>   ← 2
    <div class="text-right">
      <span><b>Año: {{ plan.fechaRegistro | date: 'yyyy' }} </b></span>
    </div>                                                  ↑ 3
  </div>
</div>
```

Ilustración 4.111 Título del formato pdf del plan de trabajo.

El formato es llenado con información obtenida del backend, específicamente del método **Get** (ver ilustración 4.96) al cual se le pasa por parámetro el id del plan de trabajo y devuelve la información del plan, el cuerpo del documento consta de un número auto incrementable (ver indicador 1) para mostrar un orden en el documento, también se muestra si es Interno o Externo la actividad que se tiene que realizar (ver indicador 2) y se muestra la fecha de los fallos a reparar (ver indicador 3).

```
<ng-container *ngFor="let anomalia of anomalias; let i = index">   ← 1
<tr>
  <td rowspan="2" class="text-center"><b>{{ i+1 }}</b></td>
  <td rowspan="2">{{ anomalia.descripcion }}</td>
  <td rowspan="2" class="text-center">
    <span *ngIf="anomalia.interno === true"><b>I</b></span>
    <span *ngIf="anomalia.interno === false"><b>E</b></span>
  </td>           2 →            3 ←
  <td>P</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 1 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 2 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 3 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 4 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 5 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 5 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 7 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 8 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 9 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 10 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 11 }}</td>
  <td>{{ anomalia.fechaProgramado | fechaProgramada: 12 }}</td>
</tr>
```

Ilustración 4.112 Cuerpo del formato pdf.

Una vez que los datos son mostrados al usuario en una página web, el usuario tiene la opción de hacer clic sobre un botón llamado **Descargar PDF** (ver ilustración1) como se observa en la ilustración 4.113, el cual ejecuta el método **downloadPDF** (ver indicador 2).

```
<div>
  <button class="col-lg-4 btn btn-danger pull-right"   ← 1
  2 → (click)="downloadPDF()">Descargar PDF</button>
</div>
```

Ilustración 4.113 Botón para descargar el pdf del plan de trabajo.

El método **downloadPDF** mostrado en la ilustración 4.114, obtiene todo el código HTML y crea un objeto del tipo **jsPDF** de forma horizontal con hoja del tipo A4 (ver indicador 1), con fondo blanco con una escala de 3 (ver indicador 2), posteriormente genera una imagen con el código HTML obtenido de la página web donde se encuentra el diseño (ver ilustración 4.112) y lo asignada al objeto pdf creado (ver indicador 4) para generar el pdf y hacer posible la descarga con el nombre de **_plan_de_trabajo.pdf**, concatenándole la fecha en que se generó el documento el inicio del nombre.

```
// Descarga del documento pdf
downloadPDF() {
  // Extraemos el
  const DATA = document.getElementById('htmlData');
  const doc = new jsPDF('l', 'pt', 'a4');
  const options = {
    background: 'white',
    scale: 3
  };
  html2canvas(DATA, options).then((canvas) => {

    const img = canvas.toDataURL('image/PNG');

    // Add image Canvas to PDF
    const bufferX = 15;
    const bufferY = 15;
    const imgProps = (doc as any).getImageProperties(img);
    const pdfWidth = doc.internal.pageSize.getWidth() - 2 * bufferX;
    const pdfHeight = (imgProps.height * pdfWidth) / imgProps.width;
    doc.addImage(img, 'JPEG', bufferX, bufferY, pdfWidth, pdfHeight, undefined, 'FAST');
    return doc;
  }).then((docResult) => {
    docResult.save(`${new Date().toISOString()}_plan_de_trabajo.pdf`);
  });
}
```

Ilustración 4.114 Lógica del lado del frontend para descargar el pdf.

Los Sprints presentados en esta sección son los que se realizaron durante el desarrollo de cada una de las funcionalidades que conforman la plataforma analizada y diseñada en el capítulo 3 de este trabajo de tesis. En el siguiente apartado se presentarán los resultados con pantallas de las funcionalidades explicadas en esta sección.

Capítulo 5 Resultados

En este capítulo se presentan los resultados obtenidos del trabajo de tesis haciendo uso de cuatro casos de estudios de las funcionalidades más utilizadas de la aplicación, la lógica de negocios que hay detrás de dichas funcionalidades, fueron explicadas en el capítulo 4.

Antes de explicar los casos de estudio es necesario ejecutar el proyecto en un ambiente de desarrollo e iniciar sesión para posteriormente explicar cada una de las funcionalidades que ofrece la plataforma ***Soportec***.

5.1 Inicializando la aplicación Soportec

Para ejecutar la aplicación ***Soportec*** y llevar a cabo los casos de estudios mencionados se debe de seleccionar el proyecto principal, tal como se observa en la ilustración 5.1 (ver indicador 1), posteriormente se hace clic sobre el botón ***IISExpress*** (ver indicador 2), esto con la finalidad de que se ejecute un servidor web con capacidades reducidas para hacer pruebas en un equipo local, este mini servidor lo proporciona Microsoft Visual Studio.

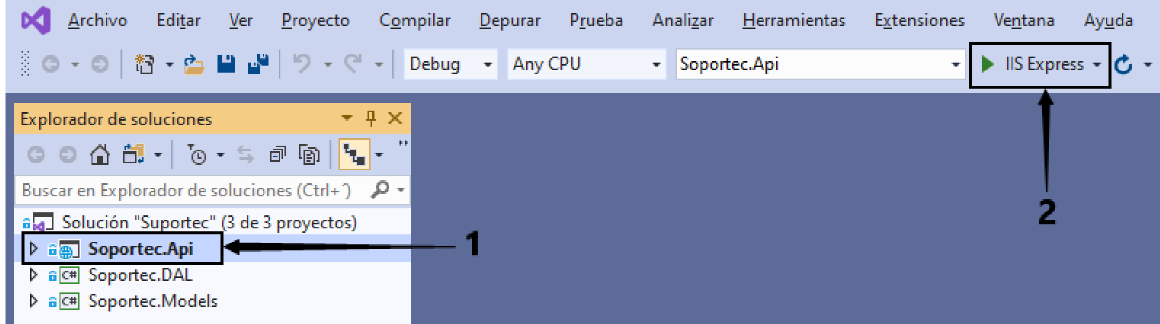


Ilustración 5.1 Aplicación Soportec en Visual Studio.

Después de que se selecciona el botón ***IIS Express***, Visual Studio levanta un servidor local, que a su vez el mismo servidor carga todas las librerías ***dll*** compiladas que son necesarias para el funcionamiento correcto del proyecto como se observa en la ilustración 5.2 (ver indicador 1).

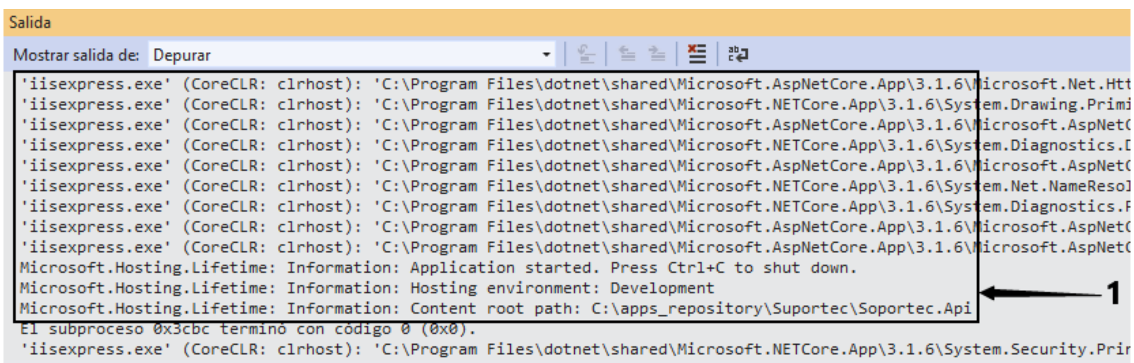


Ilustración 5.2 Salida de depuración y compilación del proyecto Soportec.

Cuando termina la carga del proyecto en el servidor, Visual Studio abre una ventana del navegador y automáticamente muestra la vista de inicio de sesión de la aplicación ***Soportec***.

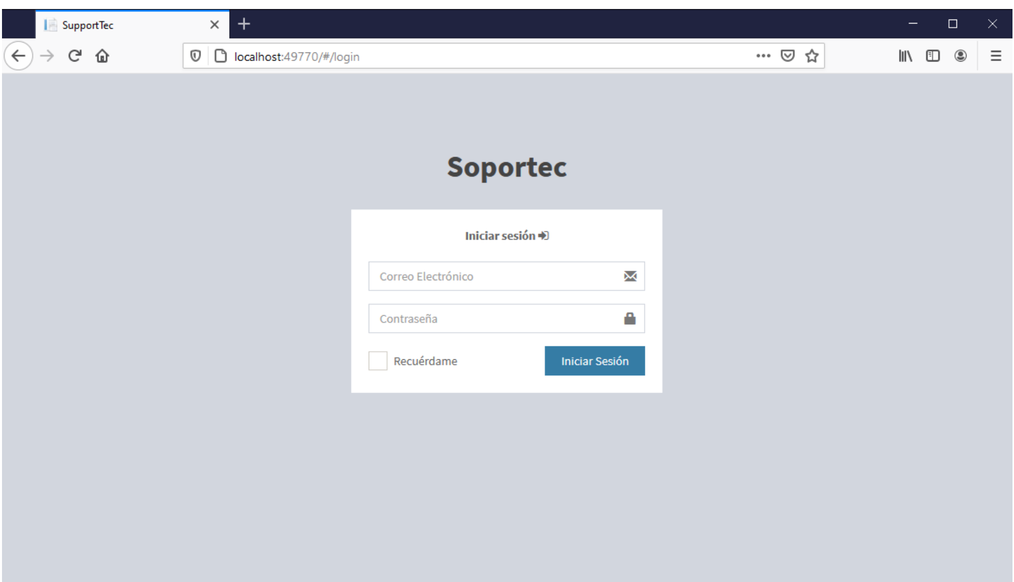


Ilustración 5.3 Inicio de sesión de la aplicación Soportec.

A partir de este punto es posible iniciar con la evaluación de los resultados obtenidos del desarrollo de la aplicación de la cual trata este trabajo de tesis. La estructura de cada uno de los casos de estudio, está compuesta por un nombre principal que engloba generalmente la funcionalidad que se mostrará, una descripción que aportará mayor detalle a la funcionalidad que se está mostrando, los objetivos que se pretenden alcanzar con el caso de estudio, en la metodología Scrum la cual se usó para el desarrollo de este proyecto se suele llamar criterios de validación, también se dará el procedimiento, el cual consiste en una serie de pasos a realizar para cumplir con el caso de estudio analizado, que ayuda a cumplir con los objetivos, cada caso de estudio se acompañará de sus respectivas ilustraciones y por último una conclusión para cada caso de estudio.

5.2 Caso de estudio 1: Gestión de usuarios (HU-1)

Descripción

En el presente caso de estudio, se realizará la creación de un usuario que accederá a la aplicación ***Soportec*** con el fin de gestionar las diversas funcionalidades con las que cuenta la plataforma ***Soportec***.

Objetivos

1. Realizar el registro de un usuario para que pueda tener acceso a la plataforma ***Soportec***.
2. Comprobar que en la base de datos se encuentren los datos del usuario registrado.
3. Verificar el correo que le llega al usuario registrado con la contraseña generada aleatoriamente para ingresar a la aplicación.

4. Realizar el inicio de sesión con el correo registrado.

Procedimiento

1. Abrir el navegador web Firefox Mozilla.
2. Ir a la dirección http://localhost: 49770
3. El administrador inicia sesión.
4. Crear un usuario en la plataforma Soportec (HU-1)
5. Iniciar sesión con el usuario registrado.
6. Verificar en la base de datos la información del usuario registrado.

Desarrollo

1. Abrir el navegador web Firefox Mozilla.
   En una computadora con el sistema operativo Windows 10 se abre el navegador web llamado Firefox, haciendo doble clic en el accedo rápido de la aplicación (ver ilustración 5.4)

Ilustración 5.4 Icono del navegador Firefox Mozilla.

2. Dentro del navegador ir a la dirección http://localhost: 49770.
   En la barra del navegador web Firefox se escribe la dirección mencionada en el punto anterior y se presiona la tecla ***ENTER*** (ver ilustración 5.5).

Ilustración 5.5 Barra de direcciones del navegador Firefox Mozilla.

3. El usuario con rol de administrador de la aplicación procede a iniciar el proceso de registro de usuario.
   El administrador se dirige a la opción ***Usuarios*** en la cual se encuentra una lista de todos los usuarios registrados en la aplicación como se observa en la ilustración 5.6.

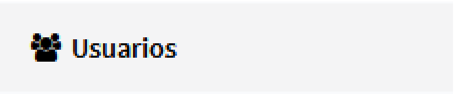


Ilustración 5.6 Opción de la aplicación para ir a la ventana de usuarios.

Una vez que el usuario se encuentra en la dirección http://localhost:49770/#/usuarios, se procede a hacer clic sobre el botón ***Usuario*** (ver ilustración 5.7 indicador 1), el cual a su vez abrirá un formulario que pedirá sean ingresada la información necesaria para registrar un nuevo usuario en la aplicación como se observa en la ilustración 5.8.

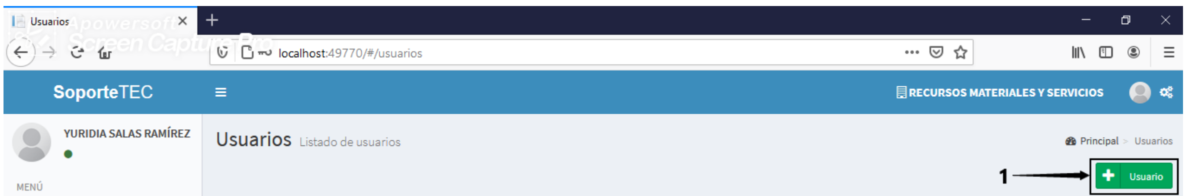


Ilustración 5.7 Botón para abrir ventana y registrar usuario.

4. Crear un usuario en la plataforma ***Soportec (HU-1)***
   Cuando se hace clic sobre el botón ***registrar usuario*** mostrado en la ilustración 5.7 (ver indicador 1), se abre una ventana como la que se muestra en la ilustración 5.8, a la cual se le ingresan los datos del usuario a registrar y se hace clic sobre el botón ***Registrar*** (ver indicador 1).

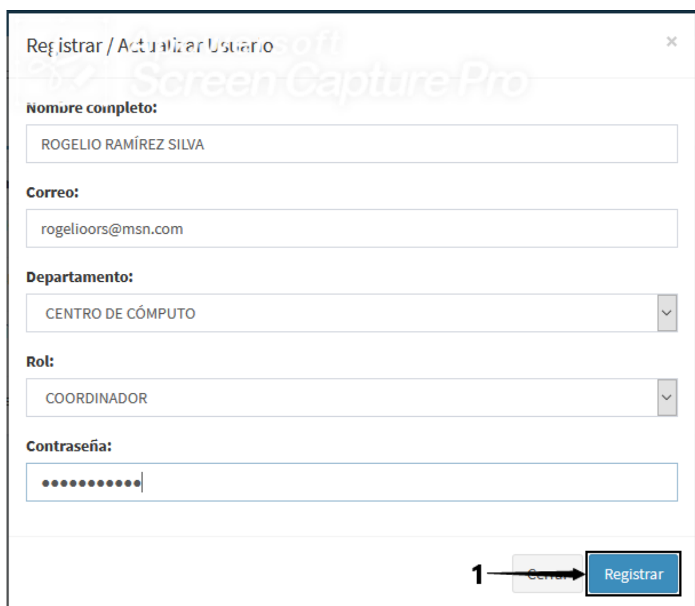


Ilustración 5.8 Ventana para registrar usuario.

Una vez que el usuario se registró en la aplicación la ventana en la que se ingresaron los datos para dar de alta un usuario se cierra y como se puede observar en la ilustración 5.9 (ver indicador 1) el usuario se muestra en la lista de los usuarios registrados.

Ilustración 5.9 Usuarios registrados en la aplicación.

5. Iniciar sesión con el usuario registrado.

Después de que el usuario fue registrado, a su cuenta de correo con el que fue registrado le llegará un mensaje con la contraseña, posteriormente el usuario ingresará las credenciales en la página de inicio de sesión como se muestra en la ilustración 5.10 (ver indicador 1).

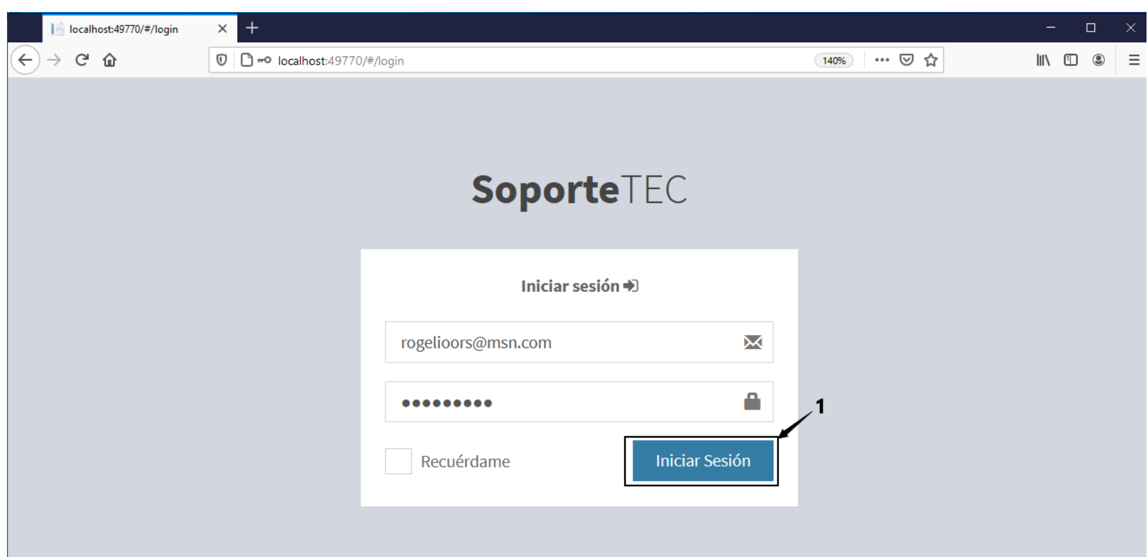


Ilustración 5.10 Usuario ingresando sus credenciales para iniciar sesión.

Una vez que el usuario hace clic sobre el botón ***iniciar sesión*** (ver indicador 1 de la ilustración 5.10), la aplicación valida las credenciales y si las credenciales son correctas, el usuario es redireccionado a la página principal como se observa en la ilustración 5.11.

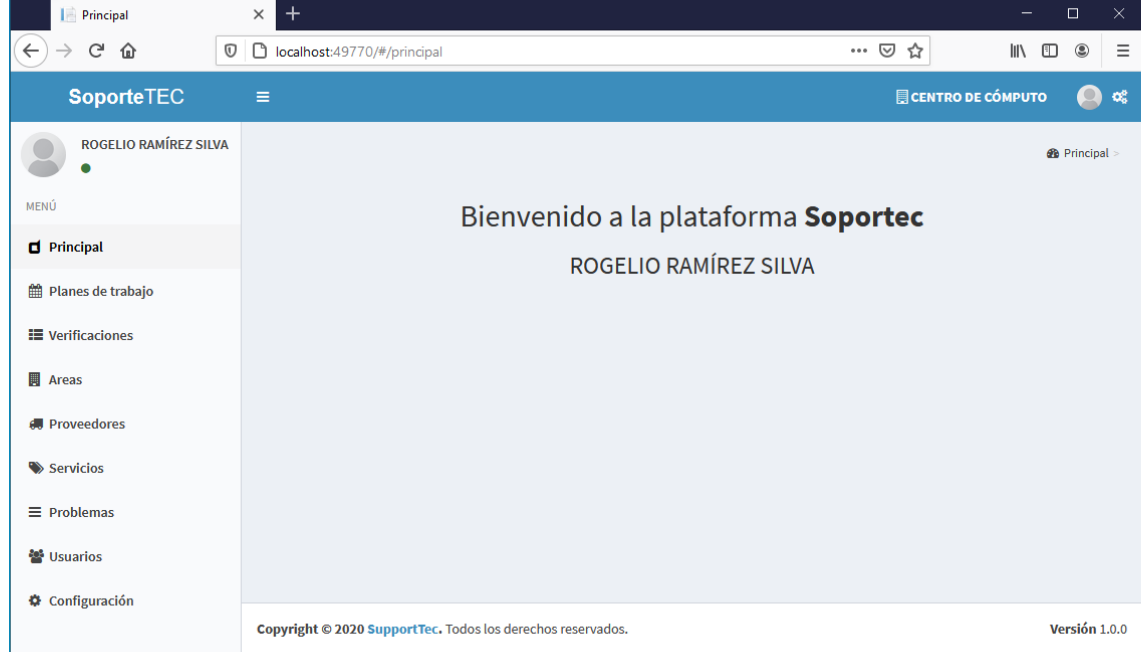


Ilustración 5.11 Inicio de sesión en la aplicación Soportec.

6. Verificar en la base de datos la información del usuario registrado.

Para verificar la información del usuario que fue registrado (***Rogelio Ramírez Silva***), en la aplicación ***Soportec***, se hace una consulta en la tabla ***Usuarios*** de la base de datos donde se persiste la información de la aplicación como se muestra en la ilustración 5.12 (ver indicador 1).

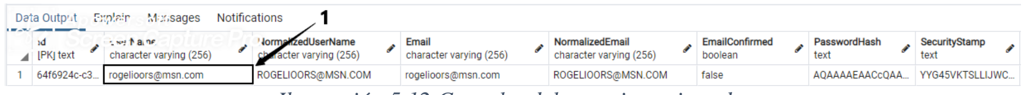


Ilustración 5.12 Consulta del usuario registrado.

En la ilustración 5.12 se muestra que el usuario registrado en la aplicación se almacenó correctamente en la base de datos.

Conclusión

Se puede observar como resultado el registro del usuario y el correcto funcionamiento de la aplicación ejecutando la tarea de la cual trata este caso de estudio, cumpliendo así con el objetivo y la ***HU-1*** que se desarrolló en el ***Sprint 2*** del capítulo 4.

5.3 Caso de estudio 2: Gestión de catálogo (HU-2)

Descripción

En el presente caso de estudio, se realizará la gestión de un registro en el catálogo de departamentos con el fin de probar la funcionalidad proporcionada por dicho módulo. Es importante mencionar que en la aplicación hay cuatro catálogos (departamentos, problemas, proveedores y servicios), debido a que los demás catálogos no ofrecen ninguna funcionalidad diferente al que se probará en este caso de estudio, se optó por presentar solo el caso de estudio de las funcionalidades de los departamentos.

Objetivos

1. Crear un registro con el nombre de un nuevo departamento en la aplicación.
2. Actualizar el registro creado.
3. Mostrar la lista de los departamentos registrados.
4. Comprobar el registro del departamento creado en la base de datos de la aplicación ***Soportec.***

Procedimiento

1. Abrir el navegador web Firefox Mozilla.
2. Dirigirse a http://localhost:49770/#/ubicaciones.
3. Iniciar sesión con el usuario Rogelio Ramírez.
4. Crear el departamento en el catálogo de áreas (***HU-2***).
5. Mostrar que se ha agregado a la base de datos el departamento creado, listando todas las áreas existentes.
6. Verificar en la base de datos el registro del departamento creado en ***Soportec.***

Desarrollo

1. Abrir el navegador web Firefox Mozilla.

En un equipo de cómputo con el sistema operativo Windows 10 se procede a abrir el navegador web Firefox, haciendo doble clic en el icono de la aplicación (ver ilustración 5.13).

Ilustración 5.13 Icono del navegador Firefox Mozilla.

2. Dirigirse a la dirección http://localhost:49770/#/ubicaciones.

   En la barra de navegación del navegador web Firefox se escribe la dirección http://localhost:49770/#/ubicaciones y se presiona la tecla [ENTER] (ver ilustración 5.14).

Ilustración 5.14 Barra de navegación de Firefox.

3. Iniciar sesión con el usuario ***Rogelio Ramírez***.

   Para crear un departamento en la plataforma ***Soportec*** es necesario iniciar sesión. Como se observa en la ilustración 5.15, en la cual se introducen las credenciales del usuario Rogelio Ramírez, el cual fue creado con el rol de coordinador del departamento de Centro de Cómputo. Una vez que las credenciales son introducidas y verificadas, se hace clic sobre el botón Iniciar sesión (ver indicador 1) para iniciar sesión.

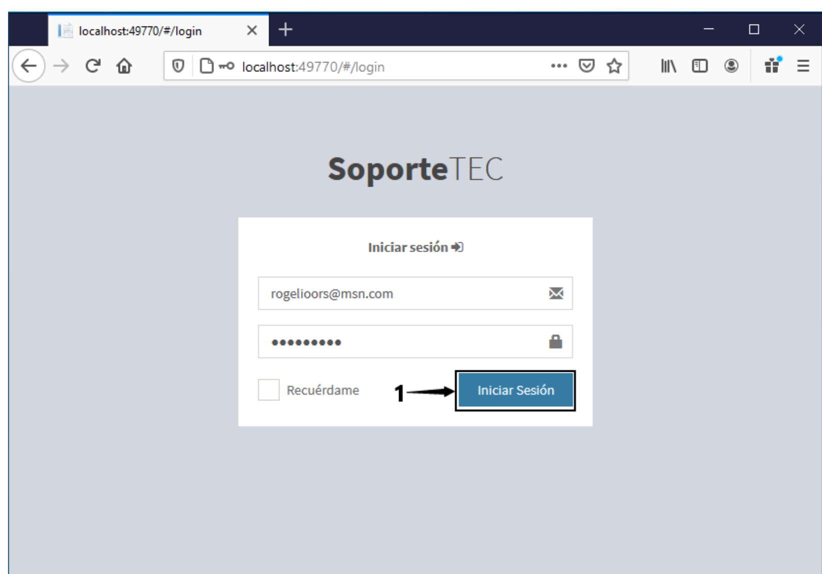


Ilustración 5.15 Inicio de sesión del usuario Rogelio Ramírez.

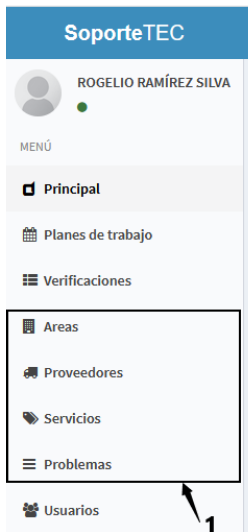


Ilustración 5.16 Menú de la plataforma Soportec.

Cuando el usuario ***Rogelio Ramírez*** ingresa a la plataforma ***Soportec***, del lado izquierdo se muestra un menú de navegación (ver ilustración 5.16), donde se tienen varias opciones, las cuales son opciones que el usuario que inicio sesión con el rol de coordinador podrá ejecutar, de las opciones que hay, existen cuatro que son catálogos (ver indicador 1), se debe hacer clic sobre la opción áreas para dar de alta un departamento.

4. Crear un departamento en el catálogo áreas de ***Soportec*** (***HU-2***)

   Cuando se hace clic la opción áreas del menú se redirecciona a una nueva ventana en la cual se hace clic sobre el botón Agregar departamento como se muestra en la ilustración 5.17 (ver indicador 1).

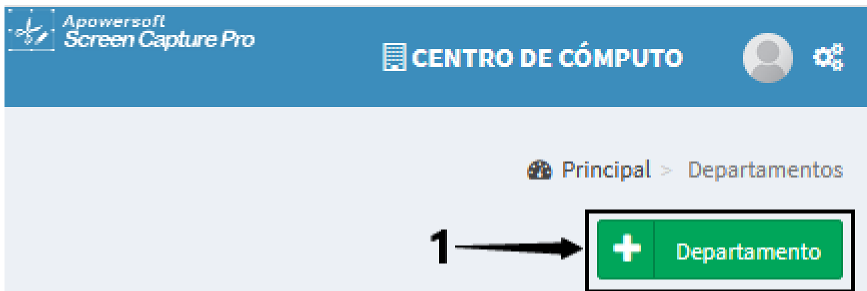


Ilustración 5.17 Botón para agregar un nuevo departamento.

Una vez que se hace clic sobre el botón ***Departamento*** (ver ilustración 5.17) (ver indicador 1), se abrirá una ventana que pedirá que se ingresen los datos necesario para dar de alta un departamento, en la ilustración 5.18 se puede observar que se piden un nombre de departamento (ver indicador 1), si está activo (ver indicador 2), si es un departamento que brinda servicio (como los departamentos de centro de cómputo y recursos materiales) (ver indicador 3) y por último, una vez que el formulario fue llenado con la información se hace clic en la opción ***Guardar*** del formulario (ver indicador 4).

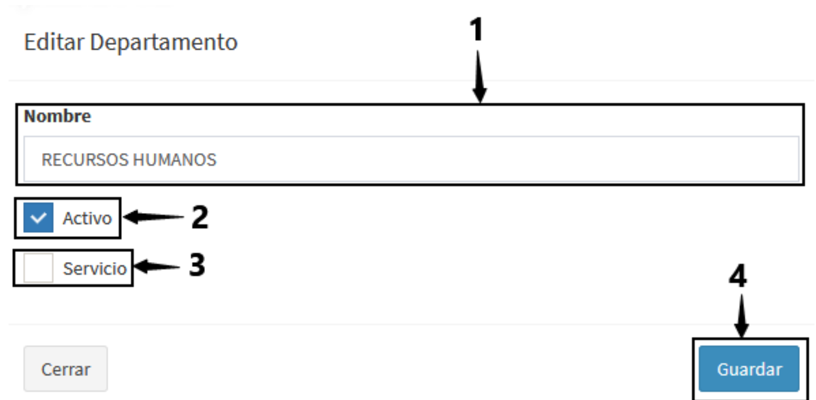


Ilustración 5.18 Formulario para registrar un nuevo departamento.

5. Listar todas las áreas existentes ***HU-3***.

Una vez que se hace clic en el botón guardar, la tabla de registro que contiene todos los departamentos que están registrados, se actualiza y se muestra el nuevo departamento como se observa en la ilustración 5.19 (ver indicador 1), dichos registros son mostrados a todos los usuarios con cualquier tipo de rol registrado en la plataforma ***Soportec***.

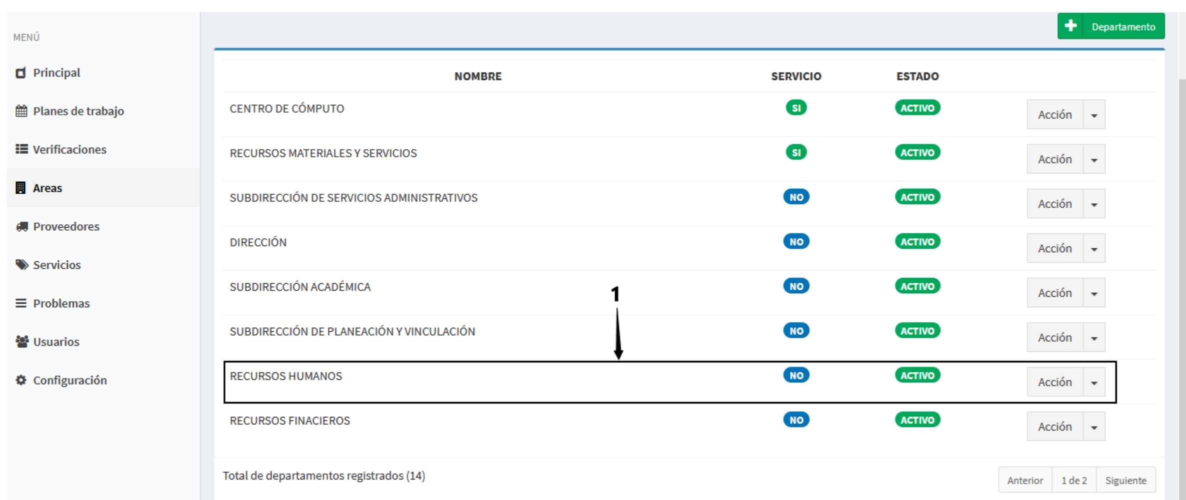


Ilustración 5.19 Listado de departamentos registrados.

6. Verificar en la base de datos el registro del departamento creado en ***Soportec***

Para verificar que los datos introducidos en el formulario **Nuevo Departamento** se almacenaron correctamente, se realiza una consulta a la tabla **área** de la base de datos de la plataforma ***Soportec***. Como se observa en la ilustración 5.20, los datos introducidos en el formulario al crear el departamento **RECURSOS HUMANOS** (ver indicador 1) se persistieron de manera correcta.

	Id [PK] integer	Descripcion character varying (200)	Estado boolean	Servicio boolean
1	14	SERVICIOS ESCOLARES	true	false
2	13	CENTRO DE INFORMACIÓN	true	false
3	12	CIENCIAS BÁSICAS	true	false
4	11	CIENCIAS ECONÓMICO-ADMI...	true	false
5	10	DESARROLLO ACADÉMICO	true	false
6	9	DIVISIÓN DE ESTUDIOS PROF...	true	false
7	8	RECURSOS FINACIEROS	true	false
8	7	RECURSOS HUMANOS	true	false
9	6	SUBDIRECCIÓN DE PLANEACI...	true	false

Ilustración 5.20 Tabla area con los registros que contiene.

Conclusión

Al verificar el correcto funcionamiento del módulo de catálogos de la aplicación al dar de alta un departamento. Se demuestra que se cumple con el objetivo establecido en el caso de estudio, además de cumplir con la ***HU-2*** y ***HU-3*** que se desarrolló en el Sprint 3 del capítulo 4.

5.4 Caso de estudio 3: Gestión de listas de verificación (HU-4, HU-5 y HU-6)

Descripción

En el presente caso de estudio se validará el funcionamiento para el registro y actualización de las listas de verificación de la aplicación ***Soportec***, con el objetivo de comprobar las siguientes historias de usuario ***HU-4, HU-5*** y ***HU-6***.

Objetivos

1. Agregar una nueva lista de verificación en la plataforma ***Soportec*** (***HU-4***).
2. Agregar anomalías a la lista de verificación (***HU-5***).
3. Ver el detalle de la anomalía registrada.
4. Actualizar el estado de la lista de verificación de ***NO AUTORIZADO*** a ***AUTORIZADO***.
5. Mostrar la información de la lista de verificación registrada (***HU-6***).
6. Verificar en la base de datos si se guardó correctamente la lista de verificación y la anomalía registrada.

Procedimiento

1. Abrir el navegador web Firefox Mozilla.
2. Dirigirse a http://localhost:49770/#/verificacion.
3. Iniciar sesión con el usuario Rogelio Ramírez.
4. Ir a la opción del menú ***Verificaciones***.
5. Agregar una nueva lista de verificación.

6. Ver la tabla de verificaciones en la aplicación para verificar que se agregó.
7. A la lista creada se le agrega una anomalía.
8. Ver los detalles de la anomalía creada.
9. Verificar en la base de datos de la aplicación que los registros creados se persistieron correctamente.

Desarrollo

1. Abrir el navegador web Firefox Mozilla.

En un equipo de cómputo con el sistema operativo Windows 10 se procede a abrir el navegador web Firefox, haciendo doble clic en el icono de la aplicación (ver ilustración 5.21).

Ilustración 5.21 Icono del navegador Firefox Mozilla.

2. Dirigirse a la dirección http://localhost:49770/#/verificaciones.

En la barra de navegación del navegador web Firefox se escribe la dirección http://localhost:49770/#/verificaciones y se presiona la tecla enter (ver ilustración 5.22).

Ilustración 5.22 Barra de navegación de Firefox.

3. Iniciar sesión con el usuario ***Rogelio Ramírez***.

Para crear una lista de verificación en la plataforma ***Soportec*** es necesario iniciar sesión. Como se observa en la ilustración 5.23, en la cual se introducen las credenciales del usuario Rogelio Ramírez, el cual fue creado con el rol de coordinador del departamento de Centro de Cómputo. Una vez que las credenciales son introducidas y verificadas, se hace clic sobre el botón Iniciar sesión (ver indicador 1) para iniciar sesión.

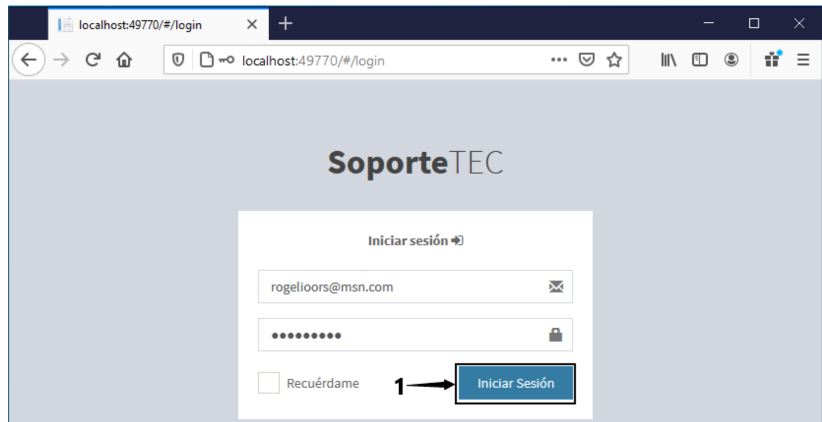


Ilustración 5.23 Inicio de sesión del usuario Rogelio Ramírez.

4. Ir a la opción del menú verificación

Ilustración 5.24 Menú de la plataforma Soportec.

Cuando el usuario ***Rogelio Ramírez*** ingresa a la plataforma ***Soportec***, del lado izquierdo se muestra un menú de navegación (ver ilustración 5.24), donde se tienen varias opciones, las cuales son opciones que el usuario que inicio sesión con el rol de coordinador podrá ejecutar, la opción a la que se debe de hacer clic para crear una lista de verificación se llama ***verificaciones*** (ver indicador 1).

5. Crear una lista de verificación en la plataforma ***Soportec*** *(**HU-4**)*

Cuando se hace clic en la opción verificaciones del menú, se redirecciona a una nueva ventana en la cual se hace clic sobre el botón agregar verificación como se muestra en la ilustración 5.25 (ver indicador 1).

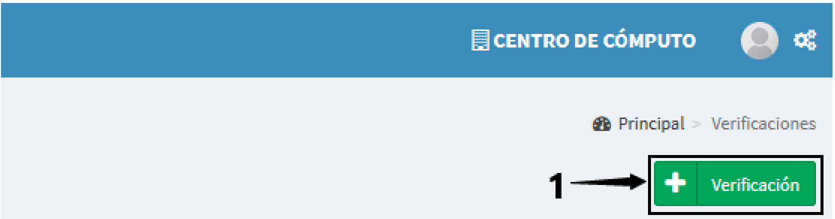


Ilustración 5.25 Botón para agregar una lista de verificación.

Una vez que se hace clic sobre el botón ***Verificación*** (ver ilustración 5.25) (ver indicador 1), se abrirá una ventana que pedirá que se ingresen los datos necesario para dar de alta una lista, en la ilustración 5.26 se puede observar que se pide el nombre del área a la cual se le hará una revisión (ver indicador 1), si se activa la lista se podrán agregar anomalías a la verificación creada (ver indicador 2) y por último, una vez que el formulario fue llenado con la información, se hace clic en la opción ***Guardar*** del formulario (ver indicador 3) para dar de alta la lista a la cual se le agregarán anomalías encontradas en los edificios e instalaciones de la institución.

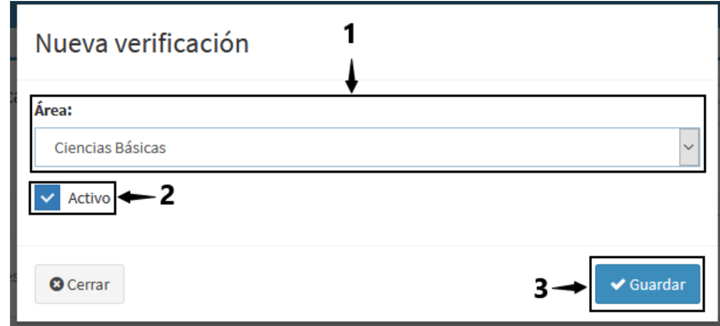


Ilustración 5.26 Formulario para registrar una nueva lista de verificación.

Después de que se hace clic sobre la opción **Guardar** el registro es creado y se muestra la tabla que contiene todas las verificaciones, tal como se observa en la ilustración 5.27.

6. Ver la tabla de verificaciones en la aplicación para corroborar que se agregó (**HU-5**)

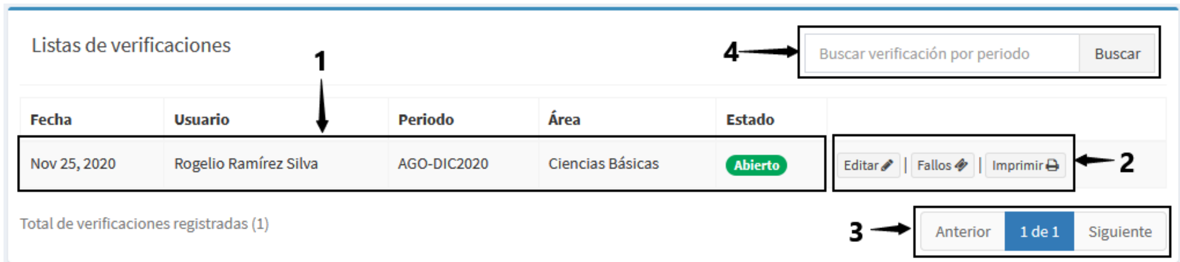


Ilustración 5.27 Listas de registros de las verificaciones.

En la ilustración 5.27, después de que se registró la verificación se muestran los datos del usuario, la fecha en que se creó, el periodo, el área que se revisará y el estado actual de la lista (ver indicador 1), la lista de verificación registrada tiene tres opciones posibles que se pueden ejecutar sobre ella las cuales son, **Editar**, **Fallos** e **Imprimir** (ver indicador 2), también se puede hacer uso de la paginación, es decir, navegar entre los registros haciendo uso de los botones **Anterior** y **Siguiente** (ver indicar 3) y buscar una lista por periodo (ver indicador 4), las funcionalidades mostradas anteriormente fueron desarrolladas en el Sprint 4 del capítulo 4 de este trabajo de tesis.

Si se hace clic sobre la opción **Editar** se muestra un formulario con los datos cargados de la lista de verificación, los mismo que cuando se creó dicha lista como se observa en la ilustración 5.28, la única diferencia es que el título del formulario indica que se trata de una edición y no de una nueva verificación (ver indicador 1).

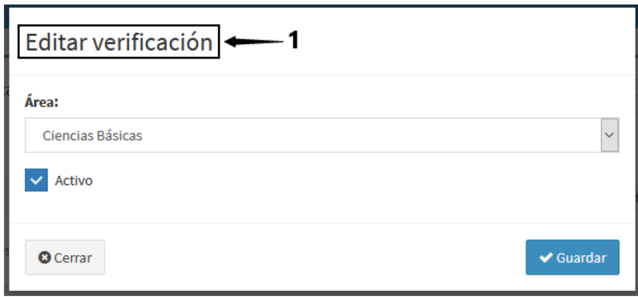
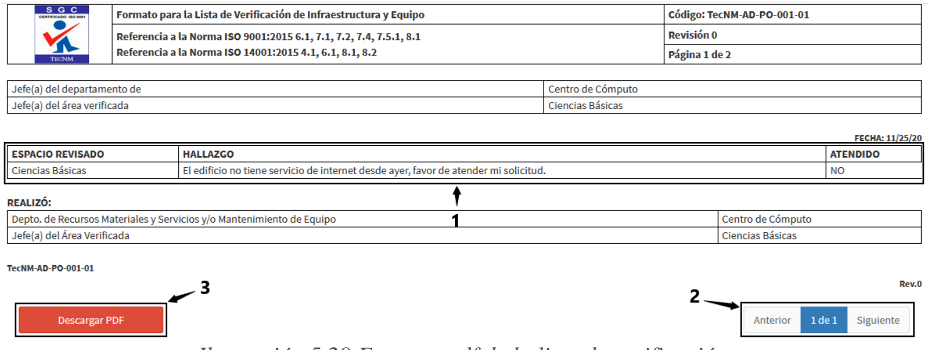


Ilustración 5.28 Formulario para editar una verificación.



S G C TecNM	Formato para la Lista de Verificación de Infraestructura y Equipo	Código: TecNM-AD-PO-001-01
	Referencia a la Norma ISO 9001:2015 6.1, 7.1, 7.2, 7.4, 7.5.1, 8.1 Referencia a la Norma ISO 14001:2015 4.1, 6.1, 8.1, 8.2	Revisión 0 Página 1 de 2

Jefe(a) del departamento de	Centro de Cómputo
Jefe(a) del área verificada	Ciencias Básicas

FECHA: 11/25/20

ESPACIO REVISADO	HALLAZGO	ATENDIDO
Ciencias Básicas	El edificio no tiene servicio de internet desde ayer, favor de atender mi solicitud.	NO

REALIZÓ:

Depto. de Recursos Materiales y Servicios y/o Mantenimiento de Equipo	Centro de Cómputo
Jefe(a) del Área Verificada	Ciencias Básicas

TecNM-AD-PO-001-01

Rev.0

Ilustración 5.29 Formato pdf de la lista de verificación.

Si se hace clic sobre la opción imprimir (ver ilustración 5.27) la plataforma redirecciona a una página con el diseño del formato de una lista de verificación tal como se observa en la ilustración 5.29, la cual muestra el hallazgo (anomalía o fallo) reportado, en este caso es uno, pero pueden ser varios y es posible navegar entre ellos con los botones **Anterior** y **Siguiente** (ver indicador 2) y descargar el formato pdf con el botón llamado **Descargar PDF** (ver indicador 3).

7. A la lista creada se le agrega una anomalía (**HU-6**)

La tabla de la ilustración 5.27, tiene una segunda opción después de la de edición, llamada **Fallo**, la cual permite redireccionar a la lista de fallos que se van registrando.

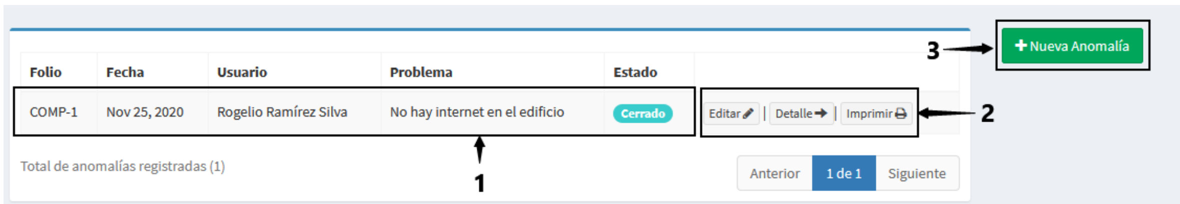


Ilustración 5.30 Lista de anomalías.

El registro de la ilustración 5.30, muestra la anomalía registrada por el usuario Rogelio Ramírez Silva con estado **Cerrado** (ver indicador 1), las anomalías tienen tres opciones para ejecutar, la primera es la edición de la anomalía reportada, la cual es el mismo formulario que se utiliza para registrar la anomalía (ver ilustración 5.30), la segunda es mostrar el detalle de la anomalía y la tercera la impresión de la orden de trabajo (ver indicador 2), la página de anomalías también cuenta con un botón para agregar una **Nueva Anomalía** (ver indicador 3).

La primera opción que se explicará es la de crear una nueva anomalía, la cual al presionar el botón **Nueva Anomalía** se abre un formulario como se observa en la ilustración 5.31, el cual contiene un título (ver indicado 1), el cual cambiara dependiendo de si se está agregando una nueva anomalía o se está editando la misma anomalía por cualquier error que pudo ser capturado, también se debe de seleccionar un problema, estos problemas son cargados desde el catálogo de problemas visto en secciones anteriores (ver ilustración 5.16), posteriormente se describe el problema que se está reportando, es decir, se describe más a detalle el problema (ver indicador 3), después se selecciona la opción **Atendido**, esto depende de si el problema

fue resuelto en el momento (ver indicador 4) y por último se hace clic en el botón **Guardar** para almacenar el registro en la base de datos.

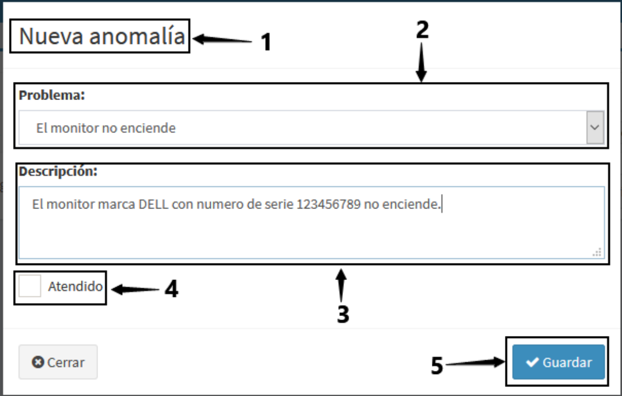


Ilustración 5.31 Formulario para registrar una anomalía.

Después de que se hace clic en el botón **Guardar**, el formulario se cierra y se actualiza la lista con el nuevo problema registrado, como puede observar en la ilustración 5.32 el nuevo fallo se registra con un estado **Nuevo** y tiene las mismas opciones para ejecutar, las cuales ya fueron explicadas anteriormente.

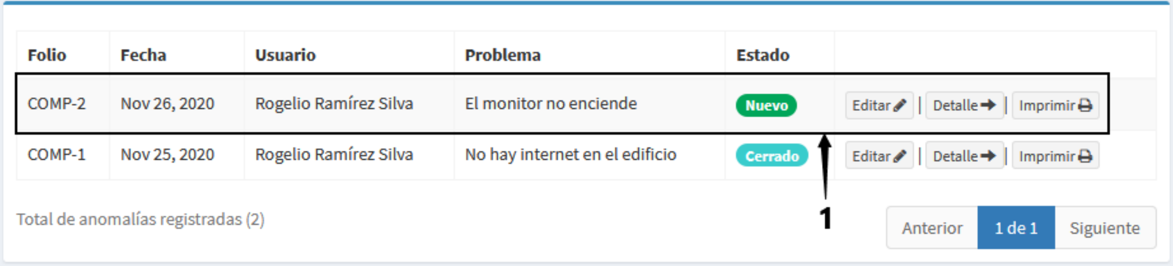


Ilustración 5.32 Anomalía insertada en la lista de verificación.

8. Ver los detalles de la anomalía creada

Después de que se guardó el fallo y se actualiza la lista de anomalías, en algún momento después de que la lista de verificación se apruebe y se cierre, se comenzará a revisar cada una de las anomalías registradas para programar las fechas (asignar a un plan de trabajo) en las que se tiene que atender y generar la orden de trabajo de la anomalía a atender. Se dirigirá a la pantalla de la ilustración 5.32 y presionará la opción **Detalle** y se mostrará la pantalla que se observa en la ilustración 5.33.

Ilustración 5.33 Anomalía detallada.

La cual muestra el folio asignado para su identificación, la fecha en que se registró, el estado actual en el que se encuentra, el usuario que registro la anomalía, el problema reportado y una descripción del mismo (ver indicador 1), después de que se observa de que se trata el fallo reportado el usuario debe hacer clic sobre la opción **Detalle** (ver indicador 2), para comenzar a introducir los datos que contendrá la orden de trabajo.

Ilustración 5.34 Programación de orden de trabajo.

La ilustración 5.34 muestra el formulario que se utiliza para programar una actividad en el plan de trabajo y que cuando se atienda el problema, estos datos contendrá la orden de trabajo generada, el mantenimiento puede ser **Interno** o **Externo** (ver indicador 1), lo puede brindar la misma institución o una empresa externa (ver indicador 2), estos datos son cargados en los catálogos de proveedores y servicios (ver ilustración 5.16), la **Fecha Programa** es la fecha en que se debe de atender el fallo reportado (ver indicador 3), el Agente es la persona que acudirá a resolver el problema (ver indicador 4), en este caso solo se ingresa a una persona pues aunque un equipo resuelva el problema solo habrá una responsable del trabajo, el estado del fallo puede ser **Nuevo** (estado inicial cuando se registra el fallo), **Proceso** (estado que se le asigna cuando se programa la actividad), **Cancelado** (estado que se le puede asignar si se decide que ya no se atenderá el fallo, esto con la finalidad de no

borrar el fallo y exista un historial de dicho fallo en la base de datos) y el último estado que puede adquirir una anomalía es **Cerrado** (estado que adquiere el fallo cuando se atiende y se libera la orden de trabajo), el periodo es en el que se atenderá la anomalía, este campo se alimenta de los planes de trabajo debido a que este campo es el que indica a que plan de trabajo pertenecerá la anomalía registrada (está es la etapa en la que se le asigna una falla a un plan de trabajo).

Cada uno de los campos del formulario de la ilustración 5.34 son obligatorios, si un campo no es llenado con información, el formulario no se podrá guardar. La ilustración 5.35 muestra los campos que deben ser llenado con información una vez que se va a generar la orden de trabajo, se debe de cambiar el estado de la anomalía a **Cerrado** (ver indicador 1), debe describirse la solución que se le dio al problema (ver indicador 2), si la actividad fue reprogramada (ver indicador 3), se debe de ingresar la nueva fecha e introducir una descripción del motivo por el cual no se atendió en la fecha establecida (ver indicador 4), el costo es otro campo que puede ser ingresado, pero también se toma en cuenta que puede quedar en ceros si no se desea ingresar un costo de operación de la actividad realizada (ver indicador 5) y por último para actualizar la información del formulario se debe presionar el botón **Actualizar** (ver indicador 6) para que los cambios se reflejen en la base de datos.

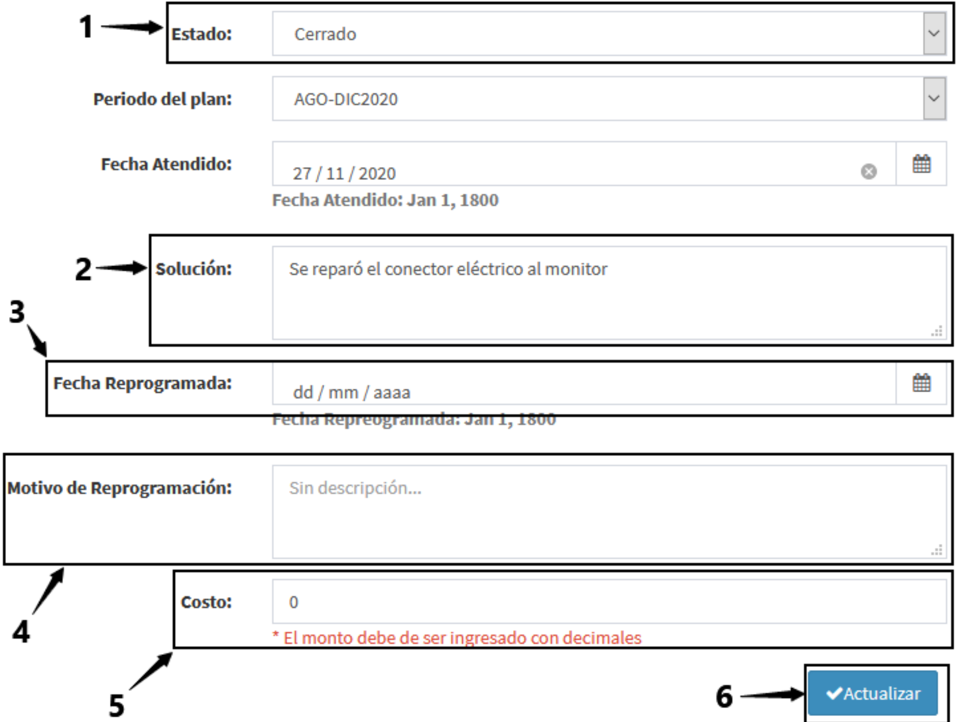
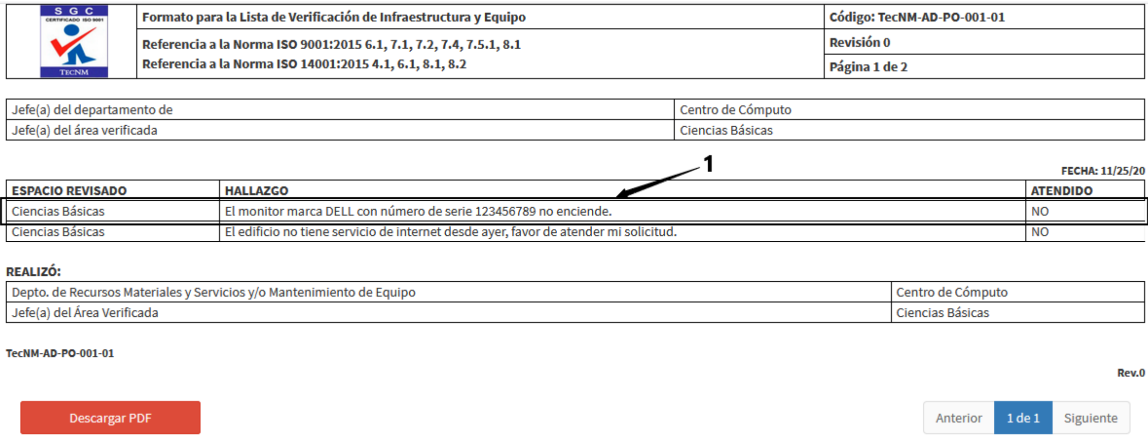


Ilustración 5.35 Detalle de la orden de trabajo.

Cuando se presiona el botón actualizar, se lanza una alerta indicando que el detalle de la anomalía fue actualizado, tal como se observa en la ilustración 5.36.

Ilustración 5.36 Alerta indicando que se actualizó el detalle de la anomalía.

Al regresar a las opciones de la lista de verificación (ver ilustración 5.27) y se hace clic sobre la opción **Imprimir** se muestra de nuevo la ventana donde se imprime el pdf y se observa que ya tiene dos hallazgos (anomalías) registrados y como se puede observar es el fallo registrado anteriormente, tal como se observa en la ilustración 5.33 (ver indicador 1).



	Formato para la Lista de Verificación de Infraestructura y Equipo	Código: TecNM-AD-PO-001-01
	Referencia a la Norma ISO 9001:2015 6.1, 7.1, 7.2, 7.4, 7.5.1, 8.1	Revisión 0
	Referencia a la Norma ISO 14001:2015 4.1, 6.1, 8.1, 8.2	Página 1 de 2

Jefe(a) del departamento de	Centro de Cómputo
Jefe(a) del área verificada	Ciencias Básicas

FECHA: 11/25/20

ESPACIO REVISADO	HALLAZGO	ATENDIDO
Ciencias Básicas	El monitor marca DELL con número de serie 123456789 no enciende.	NO
Ciencias Básicas	El edificio no tiene servicio de internet desde ayer, favor de atender mi solicitud.	NO

REALIZÓ:

Depto. de Recursos Materiales y Servicios y/o Mantenimiento de Equipo	Centro de Cómputo
Jefe(a) del Área Verificada	Ciencias Básicas

TecNM-AD-PO-001-01

Rev.0

Ilustración 5.37 Formato pdf de la lista de verificación con dos registros.

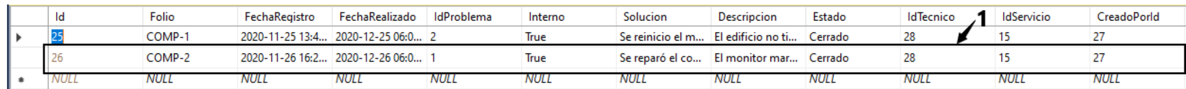
9. Verificar en la base de datos de la aplicación que los registros creados se persisten correctamente

La información capturada desde el formulario para la creación de una la lista de verificación fue almacenada en la base de datos tal como se observa en la ilustración 5.38 (ver indicador 1), la tabla en la que se almacena el registro fue en la llamada **Verificación**.

	Id	FechaRegistro	Estado	IdArea	IdDepartamento	Periodo	CreadoPorId
▶	9	2020-11-25	True	15	1	AGO-DIC2020	27
*	NULL	NULL	NULL	NULL 1	NULL	NULL	NULL

Ilustración 5.38 Persistencia de la información de la verificación.

Una vez que se registró la lista de verificación se creó la anomalía, como se puede observar en la ilustración 5.39 (ver indicador 1), la información persiste correctamente.



	Id	Folio	FechaRegistro	FechaRealizado	IdProblema	Interno	Solucion	Descripcion	Estado	IdTecnico	IdServicio	CreadoPorId
▶	25	COMP-1	2020-11-25 13:4...	2020-12-25 06:0...	2	True	Se reinicio el m...	El edificio no ti...	Cerrado	28	15	27
	26	COMP-2	2020-11-26 16:2...	2020-12-26 06:0...	1	True	Se reparó el co...	El monitor mar...	Cerrado	28	15	27
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 5.39 Persistencia de la información de una anomalía.

Conclusión

Como se observó, el correcto funcionamiento del módulo que permite gestionar las listas de verificación, cumple con el objetivo establecido de este caso de estudio, además de que se cumple con las **HU-4**, **HU-5**, **HU-6** y con los criterios de validación de cada una de las

historias de usuario, ya que se pueden registrar listas de verificación, se pueden editar, se le pueden agregar anomalías como mínimo en el módulo. De esta manera se cumple con los requerimientos dados por el Product Owner para el desarrollo del módulo que se desarrolló en el Sprint 4 del capítulo 4.

5.5 Caso de estudio 4: Gestión de planes de trabajo (HU-7, HU-8, HU-9 y HU-10)

Descripción

En el presente caso de estudio se validará el funcionamiento para el registro y cambio de estado de un plan de trabajo en la aplicación ***Soportec***, con el objetivo de comprobar las siguientes historias de usuario **HU-7**, **HU-8** y **HU-9**.

Objetivos

1. Agregar un nuevo plan de trabajo en la plataforma ***Soportec*** (***HU-7***).
2. Mostrar el plan de trabajo generado en la tabla de registros.
3. Cambiar el estado del plan de abierto a cerrado.
4. Ir a las anomalías registradas en las listas de verificación y asignarles el periodo del plan creado (***HU-8***).
5. Consultar los planes de trabajo registrados (**HU-9**).
6. Generar un pdf con los fallos asignados (**HU-10**).
7. Verificar en la base de datos que el plan fue almacena correctamente.

Procedimiento

1. Abrir el navegador web Firefox Mozilla.
2. Dirigirse a http://localhost: 4200/plan.
3. Iniciar sesión con el usuario Rogelio Ramírez.
4. Ir a la opción del menú ***Plan***.
5. Agregar un nuevo plan de trabajo.
6. Ver la tabla de planes de trabajo en la aplicación para verificar que se agregó.
7. Se realiza una búsqueda por periodo del plan creado.
8. Se cambia de estado de plan de trabajo.
9. Se le asignan tareas al plan de trabajo creado.
10. Consultar los planes de trabajo registrados.
11. Se crea un pdf con las tareas asignadas a un plan de trabajo.
12. Se corrobora que los datos fueron almacenados en la base de datos de la aplicación.

Desarrollo

1. Abrir el navegador web Firefox Mozilla.

   En un equipo de cómputo con el sistema operativo Windows 10 se procede a abrir el navegador web Firefox, haciendo doble clic en el icono de la aplicación (ver ilustración 5.40).

Ilustración 5.40 Icono del navegador Firefox Mozilla.

2. Dirigirse a la dirección http://localhost:4200/plan.

   En la barra de navegación del navegador web Firefox se escribe la dirección http://localhost:4200/plan y se presiona la tecla [ENTER] (ver ilustración 5.22).

Ilustración 5.41 Barra de navegación de Firefox.

3. Iniciar sesión con el usuario ***Rogelio Ramírez***.

   Para crear un plan de trabajo en la plataforma ***Soportec*** es necesario iniciar sesión. Como se observa en la ilustración 5.42, en la cual se introducen las credenciales del usuario Rogelio Ramírez, el cual fue creado con el rol de coordinador del departamento de Centro de Cómputo. Una vez que las credenciales son introducidas y verificadas, se hace clic sobre el botón Iniciar sesión (ver indicador 1) para iniciar sesión.

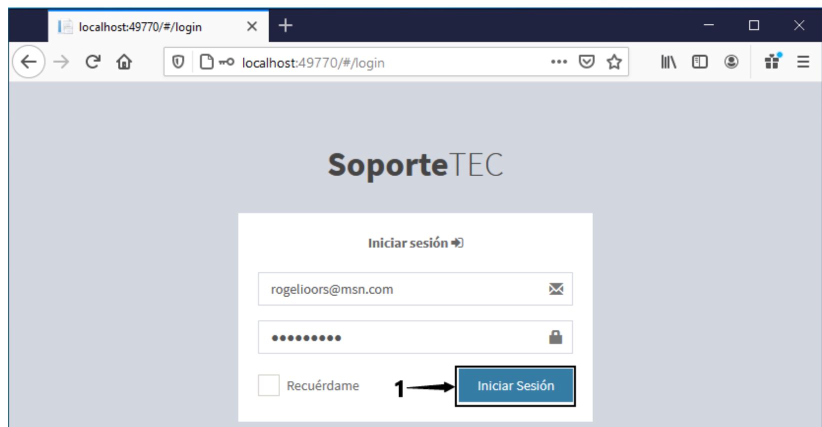


Ilustración 5.42 Inicio de sesión del usuario Rogelio Ramírez.

4. Ir a la opción del menú planes de trabajo.

Ilustración 5.43 Menú de la plataforma Soportec.

Cuando el usuario ***Rogelio Ramírez*** ingresa a la plataforma ***Soportec***, del lado izquierdo se muestra un menú de navegación como se muestra en la ilustración 5.43, el cual cuenta con varias opciones, las cuales son opciones que el usuario que inicio sesión con el rol de coordinador podrá ejecutar, la opción a la que se debe de hacer clic para crear un plan se llama ***Planes de trabajo***.

5. Crear un nuevo plan de trabajo ***Soportec*** *(**HU-7**)*

   Cuando se hace clic en la opción **plan de trabajo** (ver ilustración 5.34), el usuario es redireccionado a una nueva ventana que muestra la opción **Nuevo plan de trabajo**, como se observa en la ilustración 5.44 (ver indicador 1).

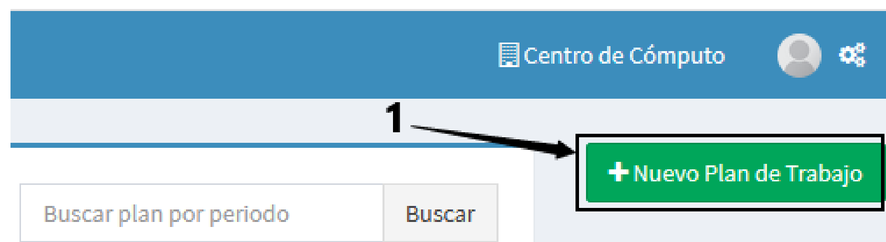
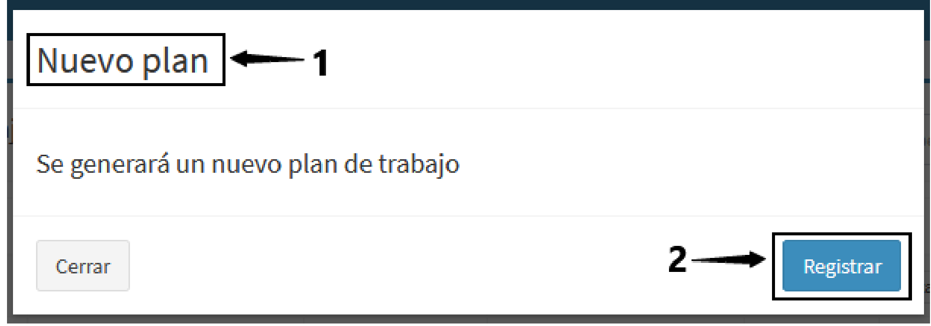


Ilustración 5.44 Opción para agregar un nuevo plan de trabajo.

Al hacer clic sobre la opción mostrada en la ilustración 5.44, se abrirá una ventana emergente preguntando si se desea crear el plan de trabajo, tal como se observa en la ilustración 5.45 (ver indicador 1).



Lo que el usuario tiene que hacer para crear el plan de trabajo, es dar clic sobre el botón **Registrar** (ver indicador 2).

6. Ver la tabla de planes de trabajo en la aplicación para verificar que se agregó (***HU-9***)

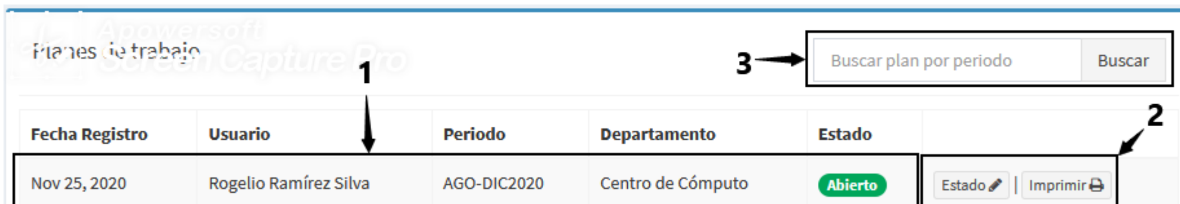


Ilustración 5.45 Lista de planes de trabajo.

Cuando se acepta crear el plan, el formulario se cierra y se muestra la tabla de la ilustración 5.45, con los datos de la persona que registro el plan de trabajo (ver indicador 1), dos opciones para ejecutar sobre el plan de trabajo (ver indicador 2) y un cuadro de búsqueda por periodo (ver indicador 3).

7. Se realiza una búsqueda por periodo del plan creado

   Cuando el usuario requiere buscar un plan de trabajo debe de ingresar el periodo del plan, como se observa en la ilustración 5.46 (ver indicador 1) e ingresar el periodo, si existe un plan de trabajo en el periodo indicado se mostrará en la tabla, pero si no existe ningún plan se mostrará un mensaje en la tabla indicando que **no se encontraron registros** (ver indicador 2).

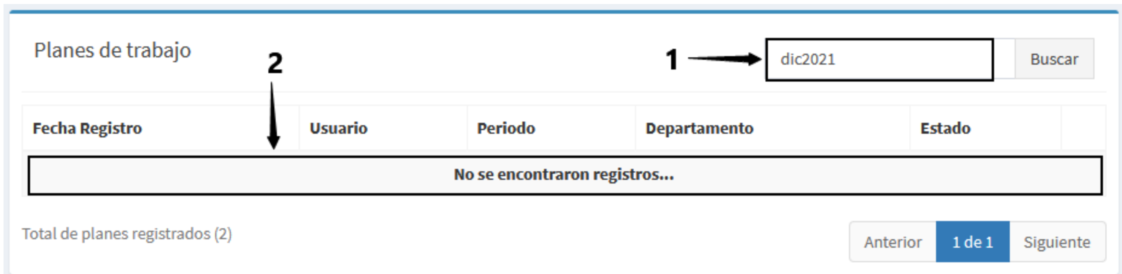


Ilustración 5.46 Buscando plan por periodo.

8. Se cambia de estado el plan de trabajo

Como se observa en la ilustración 5.47 (ver indicador 1) el plan está abierto, esto quiere decir que puede se le pueden asignar fallos a dicho plan, el segundo plan (ver indicador 2) está cerrado, eso quiere decir que a ese plan no se le podrán asignar fallos como actividad a realizar y por lo tanto no se mostrará en el pdf generado del plan.

Ilustración 5.47 Planes de trabajo cerrados.

9. Se le asignan tareas al plan de trabajo creado (***HU-8***)

   La asignación de tareas aun plan, está relacionado con el cambio de estado de un plan de trabajo, cuando un plan de trabajo se registra inicialmente se crea con el estado **Abierto** indicando que se pueden agregar anomalías al plan de trabajo en la opción ***Verificación > Fallos > Detalle*** (ver ilustración 5.34).

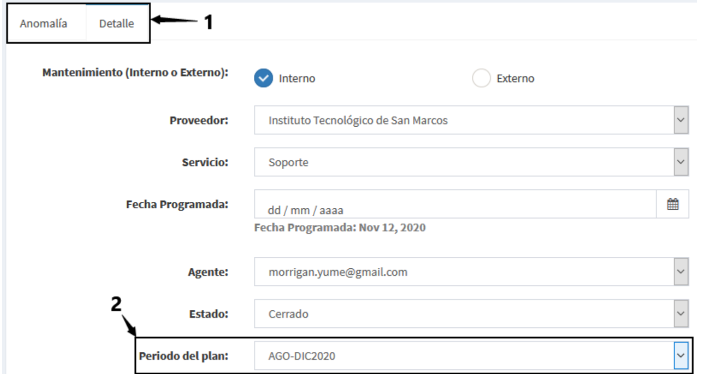


Ilustración 5.48 Asignando a una anomalía un plan de trabajo activo.

En la ilustración 5.48, se observa que en el detalle de la anomalía (ver indicador 1) al tener un plan de trabajo activo se muestran los periodos activos, para signarles un fallo a reparar. De lo contrario, si no hubiera planes de trabajos activos, en el mismo campo de la ilustración 5.48 (ver indicador 2) se mostraría un mensaje indicando que **debe ser abierto un plan de trabajo para asignarle la actividad**, como se muestra en la ilustración 5.49.

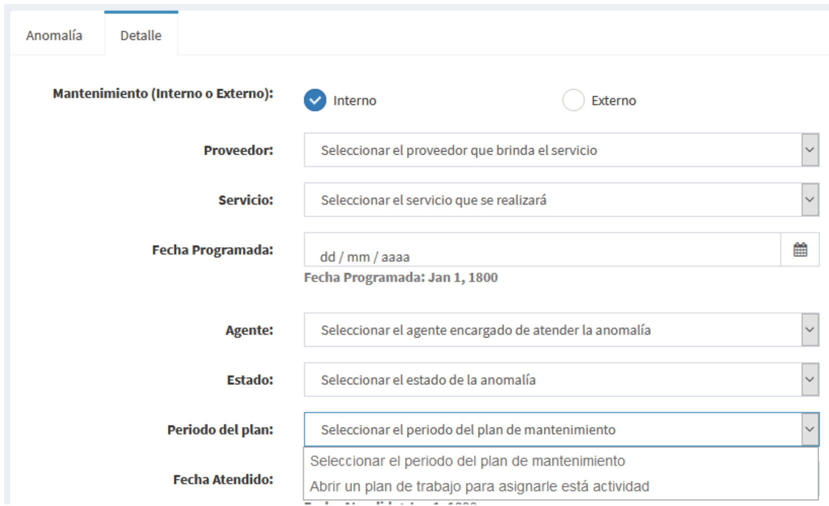
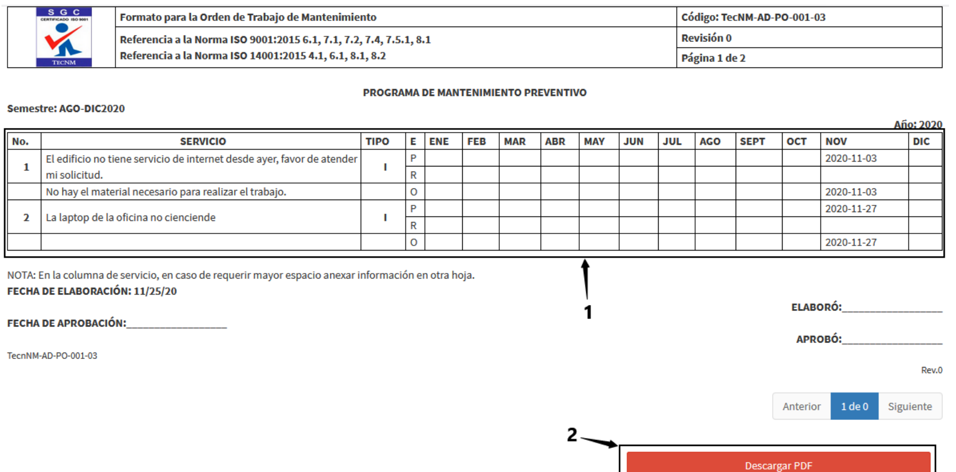


Ilustración 5.49 Fallo sin asignación de planes de trabajo.

10. Generar pdf con los fallos asignados (***HU-10***)

    El pdf generado del plan de trabajo se observa en la ilustración 5.50, con dos registros de prueba que permiten mostrar que las anomalías registradas (ver indicador 1), una vez que son asignadas al plan de trabajo, se muestran en el pdf generado. Al hacer clic sobre la opción **Imprimir** mostrado en la ilustración 5.45 (ver indicador 2) se abrirá una nueva ventana que mostrará el formato del plan de trabajo con las anomalías asignadas.

Al hacer clic sobre el botón **Descargar PDF** de la ilustración 5.50 (ver indicador 2) se descarga el pdf en el equipo que se encuentre el usuario, tal como se observa en la ilustración 5.51 (ver indicador 1).



Formato para la Orden de Trabajo de Mantenimiento	Código: TecNM-AD-PO-001-03
Referencia a la Norma ISO 9001:2015 6.1, 7.1, 7.2, 7.4, 7.5.1, 8.1	Revisión 0
Referencia a la Norma ISO 14001:2015 4.1, 6.1, 8.1, 8.2	Página 1 de 2

PROGRAMA DE MANTENIMIENTO PREVENTIVO

Semestre: AGO-DIC2020

Año: 2020

No.	SERVICIO	TIPO	E	ENE	FEB	MAR	ABR	MAY	JUN	JUL	AGO	SEPT	OCT	NOV	DIC
1	El edificio no tiene servicio de internet desde ayer, favor de atender mi solicitud.	I	P											2020-11-03	
			R												
	No hay el material necesario para realizar el trabajo.		O											2020-11-03	
2	La laptop de la oficina no cienciende	I	P											2020-11-27	
			R												
			O											2020-11-27	

NOTA: En la columna de servicio, en caso de requerir mayor espacio anexar información en otra hoja.
FECHA DE ELABORACIÓN: 11/25/20

ELABORÓ:________________

FECHA DE APROBACIÓN:________________

APROBÓ:________________

TecnNM-AD-PO-001-03

Rev.0

Ilustración 5.50 Pdf generado del plan de trabajo.

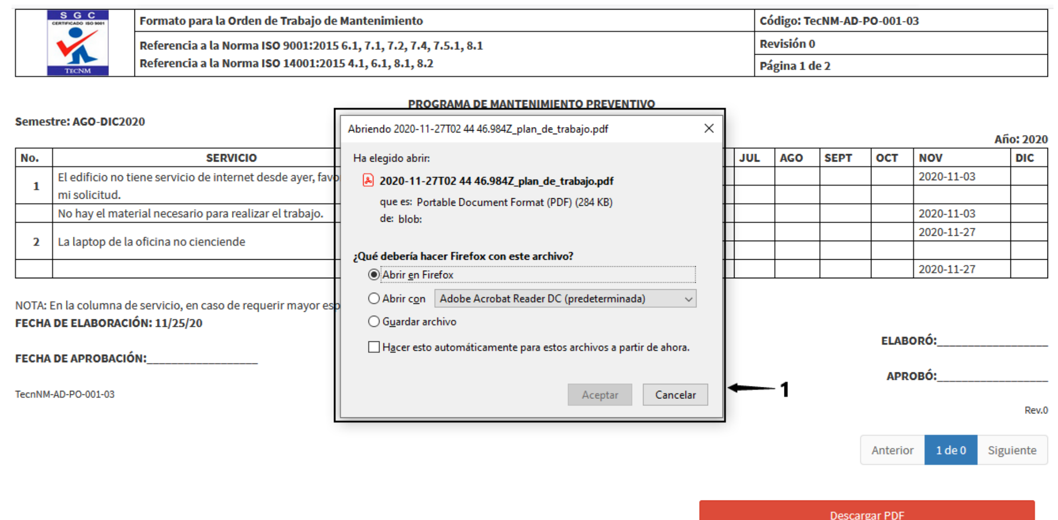


Ilustración 5.51 Descarga del formato pdf con información de los fallos.

11. Se corrobora que los datos fueron almacenados en la base de datos de la aplicación.

La información que se ingresó a través de los formularios programados en el módulo de plan de trabajo se almacenó correctamente, tal como se observa ilustración 5.52, en la tabla plan de trabajo de la base de datos.

	Id	FechaRegistro	CreadoPorId	Estado	Periodo	IdDepartamento
▶	4	2020-11-25	27	True	AGO-DIC2020	1
	5	2020-11-25	27	False	AGO-DIC2020	1
*	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 5.52 Información almacenada en la tabla que almacena los planes de trabajo.

Conclusión

Como se observó en el apartado de desarrollo de este caso de estudio, se llevaron a cabo cada una de las pasos del objetivo, cumpliendo con cada uno de ellos, además de que se cumplió con la **HU-7**, la cual dice que un usuario puede generar un plan de trabajo, también se cumplió con la **HU-8**, la cual dice que un usuario debe ser capaz de asignar tareas a un plan de trabajo, también consultar los planes generados como lo dice la **HU-9** y por último se cumplió con la **HU-10,** la cual dice que el usuario debe ser capaz de generar un pdf con las anomalías asignadas a un plan de trabajo. Los criterios de aceptación de cada una de las historias de usuario fueron cumplidos, de esta forma se cumple con el desarrollo de las **HU-7**, **HU-8**, **HU-9** y **HU-10** del Sprint 5 que se desarrolló en el capítulo 4.

Capítulo 6 Conclusiones

Las conclusiones a las que se puede llegar es que al utilizar **Soportec** el Instituto Tecnológico de San Marcos, resuelve la ausencia de un sistema que se encargue de dar seguimiento y control a las anomalías reportadas cada semestre, también se resuelve el problema de la descentralización de la información que ocasiona perdida de información por descuidos o accidentes.

Este sistema podría ser utilizado en otros tecnológicos que lleven a cabo el mismo procedimiento de mantenimiento de infraestructura y equipo apegado a los lineamientos que dicta el TecNM.

Los trabajos a futuro que se tienen contemplados realizar son la implementación de nuevos módulos, como las notificaciones vía correo electrónico de las anomalías que están por atenderse, dar el soporte para estar en posibilidad de concentrar toda la información de más de cinco tecnológicos en la misma base de datos, implementar el módulo de solicitudes de mantenimiento correctivo, debido a que por requerimientos del Instituto Tecnológico de San Marcos se le dio prioridad al desarrollo de los módulos que más utilizan actualmente, los cuales son el de listas de verificación y planes de trabajo.

Referencias

Boehm, B. W. (1976). Quantitative evaluation of software qualit. *TRW Systems and Energy Group*, 592-605.

Ceballos, J. (2013). *Enciclopedia de Microsoft Visual C#.* Madrid: RA-MA Editorial.

Craig, L. (2003). *UML y Patrones, una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Editorial Pearson Prentice Hall.

Fragas, Y. S. (2015). Sistema automatizado para la gestión. *Ciencias Técnicas Agropecuarias*, 85-90.

Fragas, Y. S. (2015). Sistema automatizado para la gestión del mantenimiento de equipos (módulos patrimonio y órdenes de trabajo). *Ciencias Técnicas Agropecuarias*, 79-89.

Freeman, A. (2017). *Pro ASP.NET Core MVC 2.* London, UK: Apress.

Freeman, A. (2018). *Pro Entity Framework Core 2 for ASP.NET Core MVC.* London, UK.: Apress.

Google. (14 de Septiembre de 2016). *Google.com*. Obtenido de https://angular.io/start

Holmes, S. (2015). *Mongoose for Application Development.* Birmingham, UK: packt publishing.

James Rumbaugh, I. J. (2000). *El Lenguaje Unificado de Modelado. Manual de Referencia.* Madrid: Pearson Educacion.S.A.

Ken Schwaber, J. S. (10 de 11 de 2017). *ScrumAlliance*. Obtenido de https://www.scrumalliance.org/learn-about-scrum/the-scrum-guide

La utilización de la Ingeniería de Software en hipermedia. (2015). *Ciencia UNEMI*, 102-117.

Maceda, H. C. (2016). *Arquitectura de software Conceptos y ciclo de desarrollo.* México, D.F.: Cengage Learning Editores, S.A. de C.V.

Mardan, A. (2014). *Pro Express.js.* Apress.

Mardan, A. (2015). *Practical Node.js.* UK: Apress.

Menzinsky, A. (2016). *Scrum Master.*

Mercerat, D. A. (2001). Construyendo aplicaciones Web con una metdología de diseño orientada a objetos. *Revista Colombiana de Computación*, 20.

Microsoft. (7 de Julio de 2013). *Microsoft .NET*. Obtenido de https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

mongoosejs. (22 de 8 de 2019). *mangosta*. Obtenido de https://mongoosejs.com/docs/built-with-mongoose.html

München, L. M. (2010). *UWE – UML-based Web Engineering*. Obtenido de http://uwe.pst.ifi.lmu.de/

Neustadt, J. A. (2005). *UML 2 and the Unified Process Practical Object-Oriented Analysis and Design.* United States: Pearson Education, Inc.

OMG. (5 de septiembre de 2013). *OMG Unified Modeling Language*. Obtenido de OMG Unified Modeling Language TM (OMG UML): http://www.omg.org

Overflow, S. (7 de octubre de 2015). *Free eBook*. Obtenido de https://www.chartjs.org/docs/latest/

Ríos, J. R. (2018). Comparación de metodologías en aplicaciones web. *3C Tecnología*, 1-19.

Rodríguez, M. J. (julio de 2015). Estudio comparativo entre las metodologías ágiles y las metodologías tradicionales para la gestión de proyectos software. Univerdiad de Oviedo.

Roger S. Pressman, P. (2010). *Ingeniería del software un enfoque práctico.* México: Mc Graw Hill.

Sierra, F. J. (2013). *Enciclopedia de Microsoft Visual C# Interfaces gráficas y aplicaciones para Internet con Windows Forms y ASP.NET.* Madrid: RA-MA Editorial.

Sommerville, I. (2011). *Ingeniería de Software.* México: Person Education.

Sutherland, J. (2016). *Scrum.* España: Oceano.

Sutherland, J. (2016). *Scrum El arte de hacer el doble trabajo en la mitad de tiempo.* Oceano.

Syed, B. A. (2014). *Beginning Node.js.* New York: Apress.

Twitter. (2 de Agosto de 2011). *Twitter*. Obtenido de https://twitter.com/

Valle, A. N. (2010). *Metodologías de diseño usadas en ingeniería Web, su vinculación con las NTICs.* Madrid.