



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE APIZACO

SISTEMA DE ADMINISTRACIÓN DE TICKETS PARA MESA DE SERVICIO MÓDULO 1 BACKEND: DISEÑO Y CREACIÓN DE BASE DE DATOS

TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

**PARA OBTENER EL TÍTULO PROFESIONAL DE: INGENIERO EN
TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES**

**PRESENTA:
OSCAR NETZAHUALCOYOTL CUATECONTZI**

**NOMBRE DEL ASESOR:
MTRO. EDUARDO SÁNCHEZ LUCERO**



APIZACO, TLAX. , OCTUBRE 2023



Apizaco, Tlaxcala, **20/octubre /2023**

OFICIO No: SC/OPV/202/23

ASUNTO: Liberación de proyecto
para titulación integral

MARTÍN ROJAS RAMÍREZ
JEFE DEL DEPTO. DE DIVISIÓN DE ESTUDIOS PROFESIONALES

POR ESTE MEDIO LE INFORMO QUE HA SIDO LIBERADO EL SIGUIENTE PROYECTO PARA LA TITULACIÓN INTEGRAL.

NOMBRE DEL EGRESADO:	OSCAR NETZAHUALCOYOTL CUATECONTZI
CARRERA:	INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES
NO. DE CONTROL:	18370988
NOMBRE DEL PROYECTO:	SISTEMA DE ADMINISTRACIÓN DE TICKETS PARA MESA DE SERVICIO MODULO 1 BACKEND: DISEÑO Y CREACIÓN DE BASE DE DATOS.
PRODUCTO U OPCIÓN:	TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

AGRADEZCO DE ANTEMANO SU VALIOSO APOYO EN ESTA IMPORTANTE ACTIVIDAD PARA LA FORMACIÓN PROFESIONAL DE NUESTROS EGRESADOS.

ATENTAMENTE

*Excelencia en Educación Tecnológica®
Pensar para Servir, Servir para Triunfar®*

ALAN AUGUSTO CALLEGOS CUELLAR
JEFE DEL DEPTO. DE SISTEMAS Y COMPUTACIÓN

c.c.p. División de Estudios Profesionales



Eduardo Sánchez Lucero
EDUARDO SÁNCHEZ LUCERO
ASESOR INTERNO





TECNOLÓGICO
NACIONAL DE MÉXICO®



Instituto Tecnológico de Apizaco
División de Estudios Profesionales

Apizaco, Tlaxcala, 27/noviembre/2024

NETZAHUALCOYOTL CUATECONTZI OSCAR.
EGRESADO DE LA CARRERA DE INGENIERÍA
EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES.
CON NÚMERO DE CONTROL 18370988.
PRESENTE.

Por este medio, me permito informar a usted que por aprobación de la comisión revisora asignada para valorar el trabajo que mediante la **TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL** cuyo tema es: **SISTEMA DE ADMINISTRACIÓN DE TICKETS PARA MESA DE SERVICIO MÓDULO 1 BACKEND: DISEÑO Y CREACIÓN DE BASE DE DATOS**

conforme a lo establecido en el procedimiento para la obtención del Título Profesional en el Instituto Tecnológico, la División de Estudios Profesionales a mi cargo le otorga la:

AUTORIZACIÓN DE IMPRESIÓN

Debiendo entregar 2 USB del mismo, apropiadamente empastados, a la brevedad, para presentar su Acto de Recepcion Profesional.

Sin otro particular por el momento me despido.

ATENTAMENTE

*Excelencia en Educación Tecnológica.
Pensar para Servir, Servir para Triunfar®*

CÉSAR REYNALDO RAMOS GÓMEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES



SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL
DE MÉXICO
INSTITUTO TECNOLÓGICO DE APIZACO
**DIVISIÓN DE
ESTUDIOS PROFESIONALES**

ccp. Archivo
CRRG/rls



Agradecimientos

Me gustaría extenuar mis agradecimientos a mi padre, Teodoro Netzahualcoyotl Cuahutle por ser la imagen inquebrantable que seguir, por siempre estar presente en todo el proceso y confiar en mí desde el primer momento que inició mi vida académica.

A mi madre, Dolores Cuatecontzi Cuahutle, quien con su apoyo también ha marcado la diferencia durante todo este periodo, con su amor, comprensión, paciencia y sacrificio, quiero que sepan que, aunque a veces no lo digo, los amo mas que a nadie y estaré eternamente agradecido con su apoyo y el ejemplo de tenacidad que me han brindado.

De la misma manera, agradezco al licenciado Francisco Flores Avendaño, por permitirme emplear los conocimientos adquiridos en el periodo de residencias, por su atención y su guía por este camino.

Por último, pero no menos importante, a mi asesor, Eduardo Sánchez Lucero, por su atención y tiempo para lograr una meta más.

Resumen

El presente trabajo expone la creación de un software de administración de tickets denominado mesas de servicios el cual es capaz de agilizar la solución de incidencias y/o problemas dentro de una empresa y dar una solución en el menor tiempo posible para, poder cubrir las necesidades del cliente/empresa, para ello este sistema estará enfocado a la generación de tickets para poder dar una solución pronta y eficaz a cualquier problema presentado dentro de su empresa o negocio.

Índice

Capítulo 1: generalidades del proyecto	7
1.1 Introducción.....	7
1.2 Datos de la empresa	8
1.2.1 Descripción de la Empresa	8
1.2.2 Misión	8
1.2.3 Visión	8
1.2.4 Áreas de trabajo del estudiante.....	8
1.3 Nombre del proyecto:	9
1.3.1 Problema a resolver	9
1.3.2 Objetivo general.....	9
1.3.3 Objetivos específicos	9
1.3.4 Justificación	10
Capítulo II: Marco teórico	10
2.4 Antecedentes	10
2.5 Estudio del mercado.....	11
2.5.1 OsTicket	12
2.5.2 Freshdesk	15
Características.....	15
2.5.3 UVDesk	18
2.5.4 Remedy Service Desk de BMC	20
2.5.5 OTRS	24
2.5.6 ngDesk.....	27
Capítulo III: Metodologías	30
3.1 Buenas practicas.....	30
3.2 ISO 20000	32
3.3 ITIL 4	34

Capítulo IV: Planeación.....	36
4.1 Explicación y entendimiento del modelo de negocio.....	36
4.2 Metodología Ágil.....	36
4.3 Desarrollo basado en componentes	36
4.4 Integración y entrega continuas.....	36
4.5 Versionado de código y control de versiones.....	37
Capítulo V: Requerimientos	37
5.1 Funcionales.....	37
5.1.1 Acceso de usuarios al sistema:	37
5.1.2 Gestión de SLA	37
5.1.3 Atención del Ticket	38
5.2 Requerimientos no funcionales	39
5.3 Historias de usuario	41
5.3.1 Diagramas BPM-Ticketing	45
Diagrama BPM-Ticketing para el levantamiento y creación de tickets.	45
Diagrama BPM-Ticketing para el seguimiento y atención de tickets.	46
Capítulo VI: Tecnologías utilizadas	47
6.1 Modelo cliente servidor.....	47
6.2 Modelo vista controlador (MVC)	47
6.3 Spring Boot	48
Librerías de Spring boot:.....	48
Capítulo VII: codificación del sistema.....	48
7.1 Diseño de la base de datos	48
7.2 Creación de paquetes para la fase	49
7.3 Conexión de Spring para la exportación de entidades a MySQL	49
7.4 Desarrollo entidades con atributos	50
7.5 Relaciones entre las entidades.....	52

7.6 Patrón DAO	53
7.7 Servicios.....	56
Anotaciones para servicios	56
7.8 Controlador@.....	58
@Anotaciones para el controlador.....	58
7.9 Principales servicios del CRUD	61
Listar todos los datos de la tabla:.....	61
Listar por Id:.....	62
Crear:	62
Actualizar:.....	63
Eliminar:	63
7.10 Ejecución del servidor	63
Postman	63
Capítulo VIII: Conclusiones.....	69
8.1 Competencias desarrolladas	70
Capítulo IX: Glosario de términos	72
Capítulo X: Referencias	74

Tabla de ilustraciones

Figura 1. Historia de usuario 3	41
Figura 2. Historia de usuario 6	41
Figura 3. Historia de usuario 8	42
Figura 4. Historia de usuario 10	42
Figura 5. Historia de usuario 13	43
Figura 6. Historia de usuario 14	43
Figura 7. Historia de usuario 15	44
Figura 8. Historia de usuario 20	44

Figura 9. Proceso general de levantamiento de tickets	45
Figura 10. Proceso de revisión de status	46
Figura 11. Diagrama Entidad Relación de la base de datos	48
Figura 12. Paquetes para el desarrollo	49
Figura 13. Ubicación del archivo application properties	49
Figura 14. Código para conexión de base de datos	50
Figura 15. Entidades.....	50
Figura 16. Atributos de la clase Cliente.java	50
Figura 17. Atributos de la clase Agente.java	51
Figura 18. Atributos de la clase Departamento.java	51
Figura 19. Atributos de la clase Rol.java	52
Figura 20. Ejemplo de relaciones entre entidades	52
Figura 21. Interface IClienteDao.java.....	54
Figura 22. Método save	54
Figura 23. Método saveAll	54
Figura 24. Método findAll	55
Figura 25. Método findAllById	55
Figura 26. Método deleteById	55
Figura 27. Método delete	56
Figura 28. Métodos de CrudRepository	56
Figura 29. Código de la implementación de la interface.....	58
Figura 30. Visualización abstracta del proceso de CrossOrigin.....	59
Figura 31. Conexión con el front-end	59
Figura 32. Especificación de ruta de mapeo	60
Figura 33. Asignación del PathVariable para la ruta específica del Id.....	61
Figura 34. Listado de datos.....	61
Figura 35. Listar un cliente por el Id	62
Figura 36. Crear Cliente.....	62
Figura 37. Actualizar con manejo de errores	63
Figura 38. Eliminar con manejo de errores	63
Figura 39. Servidor corriendo y primera petición.....	64
Figura 40. Resultado de la petición por parte del servidor	64
Figura 41. Datos que arroja la petición en Postman.....	64
Figura 42. Listado desde el navegador	65

Figura 43. Servidor Front-End y Back-End corriendo.....	65
Figura 44. Intento de registro sin datos.....	65
Figura 45. Inserción correcta en el sistema.....	66
Figura 46. Inserción en la base de datos desde el servidor.....	66
Figura 47. Eliminación de un cliente	66
Figura 48. Edición de datos de un cliente	67
Figura 49. Cliente Actualizado	67
Figura 50. Actualización en el listado	68
Figura 51. Peticiones realizadas al servidor.....	68

Capítulo 1: generalidades del proyecto

1.1 Introducción

Las mesas de ayuda o “Help Desk” por su nombre en inglés, están enfocada a todos los tipos de comercio, empresa u organización con el fin de brindar una forma más completa de poder atender las necesidades o problemas de una manera más eficaz y rápida posible presentando una solución mediante metodologías e investigaciones.

Las herramientas "Help Desk" se han convertido en una parte integral de las operaciones de soporte y gestión de servicios en muchas organizaciones. Estas herramientas surgieron en respuesta a la necesidad de gestionar eficientemente las solicitudes y problemas de los usuarios, y han evolucionado a lo largo del tiempo para adaptarse a las demandas cambiantes del entorno tecnológico.

Muchas herramientas de soporte técnico ofrecen portales de autoservicio para clientes, lo que les permite crear y rastrear sus propios tickets de soporte, acceder a artículos de la base de conocimientos y encontrar respuestas a problemas comunes. Esta funcionalidad de autoservicio ayuda a mejorar la satisfacción del cliente y reduce la carga de trabajo de los equipos de soporte.

1.2 Datos de la empresa



Nombre: Soft-TI Apizaco

Giro, ramo o sector: Consultoría

Dirección: BLVD 16 de septiembre 506 interior 4, Apizaco Tlaxcala.

RFC: FOAF700917QTA

Nombre del titular de la empresa: Lic. Francisco Flores Avendaño “director general”.

Teléfono: 241- 107- 3452

1.2.1 Descripción de la Empresa

Sof-Ti es una empresa dedicada a proveer consultoría TI (Tecnología de la Información), servicios tecnológicos, integrales y desarrollo de sistemas de información, somos un grupo de consultores especialistas que generamos valor a la empresa con la elaboración de soluciones de TI enfocadas a cumplir las expectativas de nuestros clientes.

1.2.2 Misión

Fortalecer y empoderar a la empresa a través del uso adecuado de las tecnologías de la información.

1.2.3 Visión

Ser una empresa líder en transformación digital e innovadora dedicada a fortalecer las estrategias tecnológicas de las empresas.

1.2.4 Áreas de trabajo del estudiante

- Analista diseñador de base de datos
- Analista de diferentes herramientas para implementar la base de datos
- Implementador de herramientas para la creación de la base de datos

- Desarrollador de aplicaciones web

1.3 Nombre del proyecto:

“Sistema de Administración de Tickets para Mesa de Servicio (Help Desk)”

Módulo 3: diseño y creación de aplicación móvil para el registro de tickets.

1.3.1 Problema a resolver

Actualmente en el mercado existen muchos sistemas de mesas de servicio el cual tienen algunas limitantes como lo son los altos costos de licenciamiento o la falta de componentes que hagan más fácil el proceso de atención y levantamiento de un ticket ya que una mesa de servicio es un sistema que es diseñado y utilizado específicamente en empresas u organizaciones, por esa razón no hay un sistema en el mercado que se integre o se adapte a cualquier sistema de negocio, empresa u organización, y lo más importante, un sistema que sea portable, móvil con funcionalidades nativas que le den un valor agregado frente a la competencia existente.

1.3.2 Objetivo general

Diseñar e implementar una herramienta para la gestión y administración de tickets bajo el marco de referencia ITIL y mejora continua, dando seguimiento al flujo de trabajo de cualquier tipo de organización que permita medir y configurar de manera adecuada los niveles de servicio establecidos con el propósito de poder aumentar los índices de calidad del servicio.

1.3.3 Objetivos específicos

- Diseñar y crear base de datos.
- Implementar base de datos.
- Crear servicios para su consumo en el front-end (Crear, editar, actualizar y eliminar).
- Crear mensajes de error dentro del servidor.
- Diseñar y crear una arquitectura eficaz y evolutiva.
- Crear configuraciones para creación, validación de tokens y encriptación de contraseñas.
- Inyectar y consumir las configuraciones para realizar autenticación.
- Registrar aplicación de fron-tend para generación y validación de tokens.
- Gestión y administración de base de datos de usuarios y perfiles.

1.3.4 Justificación

Basado en las buenas prácticas para la gestión de servicios de tecnologías de información orientadas a conseguir servicios más efectivos y de mejor calidad como los son (ISO 20000 e ITIL), se crea un espacio en IT el cual surge este proyecto el cual requiere una atención más estricta y específica hacia los aspectos que se incluirán en nuestro marco de trabajo.

Capítulo II: Marco teórico

2.4 Antecedentes

Los antecedentes de las herramientas "Help Desk" se remontan a las décadas de 1980 y 1990, cuando las empresas comenzaron a darse cuenta de la importancia de brindar un soporte técnico efectivo a sus usuarios. Inicialmente, el soporte se gestionaba a través de llamadas telefónicas, lo que resultaba en una falta de organización y dificultades para realizar un seguimiento adecuado de los problemas.

Con el crecimiento de las organizaciones y la creciente complejidad de los sistemas tecnológicos, surgió la necesidad de herramientas más estructuradas y automatizadas para gestionar el soporte técnico. Esto llevó al desarrollo de las primeras soluciones de "Help Desk" como sistemas basados en tickets, que permitían a los usuarios enviar solicitudes de soporte y recibir respuestas a través de una plataforma centralizada.

Con el tiempo, estas herramientas evolucionaron para incluir características como la asignación automática de tickets, la priorización de problemas, el seguimiento del estado de las solicitudes y la generación de informes. Además, se incorporaron funcionalidades de gestión de conocimientos, permitiendo a los usuarios acceder a bases de conocimientos y documentos técnicos para resolver problemas comunes por sí mismos.

Con el advenimiento de Internet, las herramientas de "Help Desk" también se trasladaron al entorno en línea, lo que facilitó aún más el acceso y la colaboración entre los usuarios y los equipos de soporte técnico. Se desarrollaron soluciones basadas en la web que permitían a los usuarios enviar solicitudes de soporte a través de formularios en línea y recibir actualizaciones en tiempo real sobre el estado de sus tickets.

2.5 Estudio del mercado

A continuación, el objetivo del estudio del mercado para el desarrollo del proyecto.

1. **Enfoque:** El estudio de mercado se centra en comprender el entorno y las características del mercado en el que se introducirá el producto o servicio. Analiza a los clientes potenciales, la competencia, las tendencias del mercado, los hábitos de consumo y otros factores relevantes.

2. **Objetivos:** El objetivo principal del estudio de mercado es identificar y evaluar la demanda del producto o servicio, conocer las necesidades y preferencias del público objetivo, y determinar la viabilidad comercial del proyecto. Proporciona información para tomar decisiones estratégicas de marketing, como segmentación de mercado, posicionamiento y estrategias de precios.
3. **Datos recopilados:** El estudio de mercado implica recopilar y analizar datos cualitativos y cuantitativos relacionados con el mercado objetivo. Esto puede incluir encuestas, entrevistas, análisis de datos demográficos, investigaciones de mercado existentes y análisis de la competencia.

2.5.1 OsTicket

OsTicket es un sistema de gestión de tickets de código abierto que se utiliza ampliamente para brindar soporte al cliente y administrar solicitudes de ayuda en muchas organizaciones.

OsTicket proporciona una interfaz fácil de usar para que los agentes de soporte puedan recibir, organizar y responder a los tickets de manera eficiente. Los usuarios pueden enviar sus consultas a través de diferentes canales, como correo electrónico, formularios web o incluso Twitter, y el sistema los convierte en tickets que se pueden asignar a los agentes correspondientes.

Características

Algunas de las características comunes de OsTicket incluyen:

- **Sistema de tickets:** permite la creación, seguimiento y gestión de tickets de soporte.
- **Automatización de procesos:** se pueden configurar respuestas automáticas, reglas de enrutamiento y notificaciones para mejorar la eficiencia.
- **Base de conocimientos:** una base de datos de artículos y preguntas frecuentes que los usuarios pueden consultar antes de enviar un ticket.
- **Integraciones:** OsTicket se puede integrar con otras herramientas y servicios, como sistemas de chat en vivo, correo electrónico, CRM, entre otros.
- **Informes y análisis:** proporciona datos y métricas para evaluar el rendimiento y la productividad del equipo de soporte.

Cabe destacar que OsTicket es una solución de código abierto, lo que significa que su código fuente está disponible para que los usuarios lo modifiquen y personalicen según sus necesidades. También existe una versión comercial llamada "osTicket Plus" que ofrece características adicionales y soporte técnico.

Limitaciones:

1. **Personalización limitada:** Aunque OsTicket permite cierta personalización a través de su código fuente abierto, realizar modificaciones más avanzadas puede requerir conocimientos técnicos especializados. Esto puede dificultar

la adaptación de la herramienta a necesidades específicas de la organización.

2. **Curva de aprendizaje inicial:** Para los nuevos usuarios y administradores, OsTicket puede tener una curva de aprendizaje inicial, especialmente si no están familiarizados con el sistema o no tienen experiencia previa en la gestión de un sistema de tickets. Requiere tiempo y esfuerzo para comprender todas las características y funcionalidades disponibles.
3. **Limitaciones de escalabilidad:** Aunque OsTicket es adecuado para pequeñas y medianas empresas, puede tener limitaciones en términos de escalabilidad para grandes organizaciones con altos volúmenes de tickets y múltiples equipos de soporte. En tales casos, es posible que se requieran soluciones más robustas y escalables.
4. **Soporte técnico:** Si bien OsTicket cuenta con una comunidad activa de usuarios y desarrolladores que brindan soporte a través de foros y recursos en línea, la disponibilidad de soporte técnico oficial puede ser limitada, especialmente en comparación con soluciones comerciales que ofrecen asistencia dedicada.
5. **Actualizaciones y mantenimiento:** Mantener OsTicket actualizado puede requerir tiempo y esfuerzo, especialmente si se han realizado modificaciones personalizadas en el código fuente. Además, como es una herramienta de código abierto, es responsabilidad del usuario asegurarse de aplicar las actualizaciones y parches de seguridad necesarios.

Versión móvil

OsTicket en sí mismo no ofrece una aplicación móvil oficial para dispositivos móviles.

Sin embargo, dado que OsTicket es una solución basada en la web, se puede acceder a ella a través de un navegador móvil en dispositivos móviles, lo que permite una experiencia similar a la de una aplicación.

2.5.2 Freshdesk

Freshdesk es una popular plataforma de servicio al cliente en la nube que ofrece una amplia gama de características para la gestión de tickets y la atención al cliente.

Características

Sistema de tickets: permite la creación, seguimiento y gestión de tickets de soporte de manera centralizada.

Automatización de procesos: se pueden establecer reglas y flujos de trabajo automatizados para agilizar las respuestas y asignaciones de tickets, así como para enviar notificaciones a los clientes y agentes.

Canales de soporte omnicanal: Freshdesk admite múltiples canales de comunicación, como correo electrónico, chat en vivo, teléfono, redes sociales y portal de autoservicio. Esto brinda a los clientes opciones para comunicarse según sus preferencias.

- **Base de conocimientos:** permite crear una base de datos de artículos, preguntas frecuentes y soluciones que los clientes pueden buscar antes de

presentar un ticket. Esto ayuda a reducir la carga de trabajo del equipo de soporte al proporcionar respuestas autoayuda.

- **Automatización de respuestas:** se pueden configurar respuestas automáticas y respuestas predefinidas para consultas comunes, lo que ahorra tiempo y mejora la eficiencia del equipo de soporte.
- **Informes y análisis:** Freshdesk ofrece informes y análisis detallados para evaluar el rendimiento del equipo de soporte, medir la satisfacción del cliente y obtener información valiosa sobre patrones de soporte.
- **Integraciones:** Freshdesk se integra con una amplia gama de herramientas y servicios populares, como CRM, chat en vivo, automatización de marketing y más, lo que permite una mayor funcionalidad y sincronización de datos.

Limitaciones

Costo: Freshdesk ofrece diferentes planes de precios, incluido un plan gratuito con funciones limitadas y planes de pago con características adicionales. Para algunas organizaciones, especialmente aquellas con presupuestos limitados, el costo de los planes de pago puede resultar un obstáculo.

1. **Complejidad inicial:** Al igual que con cualquier sistema de gestión de tickets, puede haber una curva de aprendizaje inicial para familiarizarse con la interfaz y las funcionalidades de Freshdesk. Esto puede requerir tiempo y capacitación adicional para los nuevos usuarios y administradores.

2. **Personalización limitada en planes más bajos:** Algunas características avanzadas y personalizaciones más específicas pueden estar disponibles solo en planes de pago más altos. Esto puede ser una limitación si una organización requiere una personalización más profunda para adaptarse a sus necesidades específicas.
3. **Dependencia de la conexión a Internet:** Freshdesk es una solución basada en la nube, lo que significa que requiere una conexión a Internet confiable para acceder y utilizar la plataforma. Si hay interrupciones en la conectividad o problemas con el proveedor de servicios en la nube, esto puede afectar la disponibilidad y el rendimiento de Freshdesk.
4. **Soporte técnico limitado en planes gratuitos:** En el plan gratuito de Freshdesk, el soporte técnico puede estar limitado o no estar disponible. Esto significa que puede haber una falta de asistencia directa de los representantes de Freshdesk si se encuentran problemas o dificultades técnicas.
5. **Restricciones de almacenamiento de datos:** Dependiendo del plan elegido, Freshdesk puede tener limitaciones en el almacenamiento de datos, como la cantidad de tickets almacenados o el espacio para adjuntos. Esto puede requerir una gestión cuidadosa de los datos almacenados y una posible necesidad de eliminar o archivar tickets antiguos.

Versión móvil

Freshdesk ofrece aplicaciones móviles oficiales que se pueden instalar en dispositivos móviles con los sistemas operativos iOS (iPhone/iPad) y Android. Estas

aplicaciones están diseñadas específicamente para brindar una experiencia optimizada en dispositivos móviles y facilitar la gestión de tickets y la interacción con los clientes mientras estás en movimiento.

2.5.3 UVDesk

UVDesk es una plataforma de código abierto para la gestión de tickets y el soporte al cliente. Proporciona una solución integral para administrar las solicitudes de soporte y brindar un servicio eficiente a los clientes.

Características

Algunas características clave de UVDesk incluyen:

- **Sistema de tickets:** permite la creación, seguimiento y gestión de tickets de soporte de manera centralizada.
- **Automatización de procesos:** se pueden establecer reglas y flujos de trabajo automatizados para asignar tickets, enviar notificaciones y mejorar la eficiencia en la gestión de solicitudes.
- **Colaboración en equipo:** UVDesk facilita la colaboración entre los miembros del equipo de soporte, lo que les permite trabajar juntos en la resolución de tickets y compartir información relevante.
- **Canales de comunicación múltiples:** admite diferentes canales de comunicación, como correo electrónico, formularios web, chat en vivo y redes

sociales, lo que permite a los clientes comunicarse a través de su canal preferido.

- **Base de conocimientos:** UVDesk ofrece una base de conocimientos donde se pueden almacenar artículos, preguntas frecuentes y soluciones para que los clientes puedan buscar respuestas antes de crear un ticket.
- **Personalización:** UVDesk es una plataforma de código abierto, lo que permite personalizar y adaptar la herramienta según las necesidades específicas de la organización.

Limitaciones

1. **Curva de aprendizaje inicial:** Para los nuevos usuarios y administradores, UVDesk puede tener una curva de aprendizaje inicial, especialmente si no están familiarizados con el sistema o no tienen experiencia previa en la gestión de un sistema de tickets. Puede requerir tiempo y esfuerzo para comprender todas las características y funcionalidades disponibles.
2. **Personalización y desarrollo técnico:** Si bien UVDesk es una plataforma de código abierto y ofrece la posibilidad de personalizar y adaptar la herramienta, realizar modificaciones más avanzadas o implementar nuevas características puede requerir conocimientos técnicos especializados en desarrollo web.
3. **Soporte técnico:** A diferencia de las soluciones comerciales que suelen ofrecer soporte técnico dedicado, el soporte para UVDesk puede ser limitado,

ya que depende en gran medida de la comunidad de usuarios y desarrolladores que brindan asistencia a través de foros y recursos en línea.

4. **Actualizaciones y mantenimiento:** Al ser una plataforma de código abierto, es responsabilidad del usuario realizar actualizaciones y aplicar parches de seguridad. Esto implica gestionar adecuadamente las actualizaciones y mantener la plataforma actualizada, lo que puede requerir tiempo y esfuerzos adicionales.
5. **Limitaciones de escalabilidad:** Si bien UVDesk puede ser adecuado para pequeñas y medianas empresas, puede tener limitaciones en términos de escalabilidad para organizaciones más grandes o con altos volúmenes de tickets. Esto puede afectar el rendimiento y la capacidad de gestión eficiente de la herramienta en escenarios de mayor carga de trabajo.

Versión móvil

Actualmente, UVDesk no ofrece una aplicación móvil oficial para dispositivos móviles.

Sin embargo, dado que UVDesk es una plataforma basada en la web, se puede acceder a ella a través de un navegador móvil en dispositivos móviles.

2.5.4 Remedy Service Desk de BMC

Remedy Service Desk es una solución popular de gestión de servicios y tickets de BMC, una empresa reconocida en el ámbito de la gestión de servicios de TI.

Remedy Service Desk ofrece una amplia gama de características para el manejo eficiente de tickets y la prestación de servicios de soporte.

Características

Algunas de las características clave de Remedy Service Desk incluyen:

- **Sistema de tickets:** permite la creación, seguimiento y gestión de tickets de soporte de manera centralizada. Los tickets se pueden asignar a grupos o agentes específicos para su resolución.
- **Automatización de procesos:** se pueden establecer reglas y flujos de trabajo automatizados para acelerar la gestión de tickets, la asignación y la respuesta a las solicitudes de soporte.
- **Catálogo de servicios:** permite definir y gestionar un catálogo de servicios, lo que facilita a los usuarios solicitar los servicios o productos necesarios y a los agentes proporcionar la asistencia correspondiente.
- **Base de conocimientos:** brinda una base de conocimientos centralizada donde se pueden almacenar artículos, preguntas frecuentes y soluciones para que los usuarios finales encuentren respuestas a sus preguntas antes de presentar un ticket.
- **Paneles de control y análisis:** ofrece paneles de control personalizables y herramientas de informes para obtener una visibilidad en tiempo real sobre los indicadores clave de rendimiento (KPI) y el desempeño del equipo de soporte.

- **Integraciones:** Remedy Service Desk se integra con otras herramientas y sistemas, como herramientas de monitoreo, gestión de activos, gestión de cambios y más, para proporcionar una experiencia más completa y una gestión más eficiente.

Limitaciones

1. **Complejidad y curva de aprendizaje:** Remedy Service Desk es una herramienta sofisticada con muchas características y opciones de configuración. Esto puede resultar en una curva de aprendizaje pronunciada para los nuevos usuarios y administradores. Requiere tiempo y esfuerzo para familiarizarse con la interfaz y las funcionalidades, y puede requerir capacitación adicional.
2. **Costo:** Remedy Service Desk es una solución de software empresarial y, como tal, puede tener un costo significativo, especialmente para organizaciones más pequeñas o con presupuestos limitados. Los costos pueden incluir licencias, mantenimiento, soporte y posibles integraciones adicionales.
3. **Requisitos de infraestructura:** Para implementar y ejecutar Remedy Service Desk de manera eficiente, puede ser necesario contar con una infraestructura de TI sólida y bien configurada. Esto puede implicar la necesidad de servidores dedicados, capacidad de almacenamiento, recursos de red y personal de TI experimentado.

4. **Personalización y configuración compleja:** Aunque la flexibilidad de Remedy Service Desk permite una amplia personalización y configuración, esto puede requerir habilidades técnicas avanzadas y conocimientos especializados en la herramienta. La implementación y personalización más complejas pueden requerir la asistencia de expertos o consultores de BMC.
5. **Dependencia del soporte de BMC:** Como Remedy Service Desk es una solución propietaria de BMC, la asistencia y el soporte técnico dependen de la disponibilidad y calidad del soporte ofrecido por BMC. Esto puede ser una preocupación si se experimentan problemas críticos o si se necesita un soporte rápido y confiable.
6. **Complejidad en actualizaciones y migraciones:** Las actualizaciones de software y las migraciones a nuevas versiones de Remedy Service Desk pueden ser complejas y requieren una planificación cuidadosa. Las organizaciones deben considerar el impacto de las actualizaciones en la personalización existente, los flujos de trabajo y las integraciones personalizadas.

Versión móvil

Remedy Service Desk de BMC ofrece una aplicación móvil oficial llamada "BMC Helix Remedy" que se puede instalar en dispositivos móviles con los sistemas operativos iOS (iPhone/iPad) y Android. Esta aplicación móvil permite acceder y gestionar los tickets y servicios desde dispositivos móviles.

2.5.5 OTRS

OTRS (Open-source Ticket Request System) es una plataforma de gestión de tickets de código abierto que se utiliza ampliamente para la administración de servicios de atención al cliente y soporte técnico.

OTRS ofrece una variedad de características y funcionalidades para la gestión eficiente de tickets y la automatización de procesos.

Características

Algunas de las características clave de OTRS incluyen:

- **Sistema de tickets:** permite la creación, seguimiento y gestión de tickets de soporte de manera centralizada. Los tickets pueden clasificarse, priorizarse y asignarse a agentes o grupos específicos.
- **Automatización de procesos:** OTRS ofrece potentes capacidades de automatización que permiten establecer reglas y flujos de trabajo para la gestión de tickets, la asignación automática, las respuestas automáticas y otras acciones relacionadas con el ciclo de vida del ticket.
- **Gestión de conocimientos:** cuenta con una base de conocimientos integrada donde se pueden almacenar artículos, soluciones y respuestas a preguntas frecuentes para facilitar la resolución rápida de problemas y permitir a los usuarios buscar información antes de crear un ticket.

- **Portal de autoservicio:** proporciona un portal de autoservicio para que los usuarios finales puedan enviar y rastrear sus propios tickets, así como acceder a la base de conocimientos y otros recursos útiles.
- **Integraciones:** OTRS se puede integrar con otras herramientas y sistemas, como sistemas de monitoreo, bases de datos externas, herramientas de chat en vivo y más, para una gestión más completa y eficiente de los servicios de soporte.
- **Informes y análisis:** OTRS ofrece funciones de informes y análisis para medir y evaluar el rendimiento del equipo de soporte, identificar tendencias, generar métricas y obtener una visión general de la carga de trabajo y el desempeño del servicio.

Limitaciones:

1. **Curva de aprendizaje:** OTRS puede tener una curva de aprendizaje pronunciada, especialmente para los nuevos usuarios y administradores sin experiencia previa en la gestión de un sistema de tickets. La configuración inicial y la comprensión de todas las características y funcionalidades pueden requerir tiempo y esfuerzo.
2. **Interfaz de usuario:** Algunos usuarios pueden considerar que la interfaz de usuario de OTRS es menos intuitiva y visualmente atractiva en comparación con otras herramientas de gestión de tickets. Esto puede afectar la usabilidad y la experiencia del usuario.
3. **Personalización y desarrollo técnico:** La personalización avanzada de OTRS o la implementación de nuevas características pueden requerir

habilidades técnicas y conocimientos especializados en desarrollo web. Realizar modificaciones más complejas o personalizaciones específicas puede requerir recursos adicionales.

4. **Documentación y soporte:** Aunque OTRS tiene documentación y una comunidad de usuarios activa, algunos usuarios pueden encontrar que la documentación oficial no es tan exhaustiva o detallada como desearían. Además, el soporte técnico dedicado puede requerir una suscripción adicional o una inversión adicional en servicios de soporte.
5. **Actualizaciones y mantenimiento:** Al ser una solución de código abierto, es responsabilidad del usuario realizar actualizaciones y aplicar parches de seguridad. Esto puede requerir tiempo y esfuerzo adicional para mantener el sistema actualizado y garantizar la seguridad.
6. **Escalabilidad:** Aunque OTRS es adecuado para organizaciones de diferentes tamaños, algunos usuarios pueden encontrar limitaciones en términos de escalabilidad en escenarios de alto volumen de tickets o en organizaciones más grandes. El rendimiento y la capacidad de gestión pueden verse afectados en tales casos.

Versión móvil

OTRS ofrece una versión móvil para dispositivos móviles llamada "OTRS Mobile App". Esta aplicación móvil está disponible tanto para iOS (iPhone/iPad) como para Android y permite a los usuarios acceder y gestionar los tickets de OTRS desde sus dispositivos móviles.

2.5.6 ngDesk

ngDesk es una plataforma de gestión de tickets y servicio al cliente que permite a las organizaciones administrar y responder a las solicitudes de soporte de manera eficiente.

Características

Algunas de las características clave de ngDesk incluyen:

- **Sistema de tickets:** ngDesk ofrece un sistema de tickets centralizado que permite a los usuarios crear, rastrear y gestionar tickets de soporte de manera organizada. Los tickets se pueden asignar a agentes específicos, priorizarse y seguir su progreso hasta su resolución.
- **Automatización de procesos:** la herramienta cuenta con capacidades de automatización que permiten la configuración de reglas y flujos de trabajo para agilizar la gestión de tickets. Esto incluye la asignación automática de tickets, respuestas automáticas y acciones predefinidas basadas en ciertos criterios.
- **Portal de autoservicio:** ngDesk ofrece un portal de autoservicio para que los usuarios finales puedan presentar sus solicitudes de soporte, buscar respuestas en la base de conocimientos y realizar un seguimiento del progreso de sus tickets.

- **Gestión de la base de conocimientos:** la herramienta proporciona una base de conocimientos donde se pueden almacenar artículos y soluciones para consultas frecuentes. Esto ayuda a los usuarios a encontrar respuestas rápidas a sus problemas antes de presentar un ticket.
- **Chat en vivo:** ngDesk cuenta con una función de chat en vivo que permite a los usuarios comunicarse directamente con los agentes de soporte en tiempo real para resolver problemas de manera instantánea.
- **Informes y análisis:** la herramienta ofrece herramientas de generación de informes y análisis para medir el rendimiento del equipo de soporte, evaluar la satisfacción del cliente y obtener información sobre las tendencias y los patrones de soporte.

Limitaciones

1. **Interfaz de usuario limitada:** Algunos usuarios pueden considerar que la interfaz de usuario de ngDesk es menos intuitiva o menos atractiva visualmente en comparación con otras herramientas de help desk. Esto puede afectar la experiencia del usuario y requerir un período de adaptación.
2. **Personalización limitada:** ngDesk puede tener limitaciones en cuanto a la personalización y adaptabilidad a las necesidades específicas de una organización. Algunas características y funcionalidades pueden no ser

totalmente personalizables o requerir conocimientos técnicos más avanzados.

3. **Escalabilidad limitada:** Para organizaciones con grandes volúmenes de tickets y equipos de soporte extensos, ngDesk puede presentar limitaciones en cuanto a su escalabilidad. Esto puede resultar en una experiencia más lenta o dificultades para gestionar grandes volúmenes de solicitudes de soporte.
4. **Integraciones limitadas:** A diferencia de otras herramientas de help desk más establecidas, ngDesk puede tener menos integraciones con otras aplicaciones o sistemas empresariales. Esto puede limitar su capacidad para conectarse y sincronizarse con otras herramientas o plataformas utilizadas en la organización.
5. **Soporte técnico limitado:** Al ser una herramienta menos conocida en comparación con algunas soluciones de help desk más populares, es posible que ngDesk no cuente con una amplia base de usuarios o una comunidad de soporte tan activa. Esto puede resultar en un soporte técnico limitado o tiempos de respuesta más largos para resolver problemas o consultas.
6. **Costos adicionales:** Aunque ngDesk ofrece una versión gratuita con funcionalidad básica, algunas características más avanzadas o personalizaciones pueden requerir una actualización a un plan de pago. Los costos adicionales pueden ser un factor a considerar, especialmente para organizaciones con presupuestos limitados

ngDesk no cuenta con una aplicación móvil dedicada para dispositivos móviles. Sin embargo, ngDesk es accesible a través de navegadores web en dispositivos móviles.

Capítulo III: Metodologías

3.1 Buenas practicas

La gestión de servicios de tecnologías de información (ITSM, por sus siglas en inglés) se refiere a las prácticas y enfoques utilizados para diseñar, entregar, administrar y mejorar los servicios de TI dentro de una organización. El objetivo principal de la gestión de servicios de TI es garantizar que los servicios de tecnologías de información sean efectivos, eficientes y de alta calidad, alineados con las necesidades y objetivos del negocio.

Las buenas prácticas para lograr servicios más efectivos y de mejor calidad aplicadas en la creación y desarrollo de este proyecto son las siguientes:

- **Enfoque centrado en el cliente:** El cliente debe ser el foco principal en la entrega de servicios de TI. Comprende las necesidades y expectativas de los clientes y diseña servicios que satisfagan sus requerimientos. Mantén una comunicación abierta con los clientes para obtener retroalimentación y mejorar continuamente los servicios.
- **Procesos claros:** Define procesos bien definidos y documentados para todas las actividades relacionadas con la gestión de servicios de TI. Esto incluye la gestión de incidentes, gestión de cambios, gestión de problemas,

gestión de configuraciones, entre otros. Los procesos deben ser claros, comprensibles y seguidos por todo el equipo.

- **Aplicación de las mejores prácticas de la industria:** Adopta marcos y estándares reconocidos en la gestión de servicios de TI, como ITIL (IT Infrastructure Library) u otros marcos similares. Estos marcos proporcionan pautas y buenas prácticas para gestionar eficazmente los servicios de TI.
- **Acuerdos de nivel de servicio (SLA):** Define y acuerda los niveles de servicio con los clientes internos o externos. Los SLA establecen las expectativas de los clientes en términos de calidad, tiempo de respuesta, disponibilidad, capacidad y otros aspectos clave del servicio.
- **Colaboración y el trabajo en equipo:** Promueve la colaboración entre los equipos de TI y otras áreas de la organización. La gestión de servicios de TI implica trabajar en estrecha colaboración con los usuarios finales, los proveedores y otras partes interesadas para garantizar la entrega exitosa de los servicios.
- **Mejoras continuas con una arquitectura evolutiva:** Establece un ciclo de mejora continua para identificar áreas de mejora en los servicios de TI. Realiza revisiones periódicas, recopila datos y métricas, e implementa acciones correctivas y preventivas para optimizar los servicios y garantizar la satisfacción del cliente.
- **Personal adecuado:** Contar con personal capacitado y con las habilidades adecuadas para gestionar los servicios de TI de manera efectiva.

Proporciona oportunidades de capacitación y desarrollo profesional para mantener actualizado al equipo.

3.2 ISO 20000

La norma ISO/IEC 20000, también conocida como ISO 20000, es un estándar internacional que establece los requisitos para implementar y gestionar un sistema de gestión de servicios de tecnología de la información (TI) de calidad. Fue publicada por la Organización Internacional de Normalización (ISO) y la Comisión Electrotécnica Internacional (IEC) con el objetivo de mejorar la calidad de los servicios de TI y garantizar la satisfacción del cliente.

La norma ISO 20000 se basa en el ciclo de mejora continua de la calidad conocido como PDCA (Planificar, Hacer, Verificar, Actuar) y se compone de dos partes principales:

ISO/IEC 20000-1: Requisitos del sistema de gestión de servicios de TI: Esta parte establece los requisitos que una organización debe cumplir para implementar un sistema de gestión de servicios de TI efectivo. Incluye aspectos como la planificación y diseño del servicio, la gestión de proveedores, la gestión del cambio, la gestión de incidentes y problemas, la gestión del nivel de servicio, entre otros.

ISO/IEC 20000-2: Código de práctica para la gestión de servicios de TI: Esta parte proporciona orientación y recomendaciones para la implementación de los requisitos establecidos en la parte 1. Ofrece ejemplos, casos de estudio y mejores prácticas para ayudar a las organizaciones a gestionar sus servicios de TI de manera efectiva.

La implementación de la norma ISO 20000 tiene varios beneficios, incluyendo:

Mejora de la calidad de los servicios de TI: Ayuda a las organizaciones a establecer y mantener altos estándares de calidad en la prestación de servicios de TI, lo que se traduce en una mayor satisfacción del cliente.

Mayor eficiencia operativa: La norma fomenta la estandarización de los procesos y la optimización de las operaciones de TI, lo que conduce a una mayor eficiencia y reducción de costos.

Gestión proactiva de los servicios: Proporciona un marco para la gestión proactiva de incidentes, problemas y cambios, lo que ayuda a prevenir interrupciones en los servicios y minimiza el impacto de los incidentes.

Cumplimiento de requisitos legales y regulatorios: Ayuda a las organizaciones a cumplir con los requisitos legales y regulatorios relacionados con la prestación de servicios de TI.

Mayor confianza del cliente: La certificación en ISO 20000 demuestra el compromiso de una organización con la calidad y la mejora continua de sus servicios de TI, lo que genera confianza en los clientes y socios comerciales.

3.3 ITIL 4

ITIL 4 es la versión más reciente del marco de trabajo ITIL para la gestión de servicios de tecnologías de información (ITSM). Fue lanzado en 2019 y representa una evolución significativa en comparación con las versiones anteriores de ITIL. Esta nueva versión se ha actualizado para abordar los desafíos y demandas actuales del entorno empresarial digital y ágil.

A continuación, se presentan algunos aspectos clave que se tomaron en cuenta para el desarrollo del presente proyecto de ITIL 4:

- **Enfoque en el valor:** ITIL 4 pone un mayor énfasis en la creación y entrega de valor para el cliente. Reconoce la importancia de comprender las necesidades y expectativas del cliente y la necesidad de adaptarse a los cambios rápidos en el mercado y la tecnología para proporcionar servicios valiosos y relevantes.
- **Sistema de valor del servicio:** ITIL 4 introduce el concepto del Sistema de Valor del Servicio (Service Value System, SVS), que es un enfoque holístico para la gestión de servicios de TI. El SVS proporciona una estructura para la coordinación y la integración de todos los componentes, actividades y procesos involucrados en la provisión de servicios de TI.
- **Cuatro dimensiones de la gestión del servicio:** ITIL 4 presenta cuatro dimensiones clave que deben tenerse en cuenta al diseñar e implementar servicios de TI: organización y personas, información y tecnología, socios y proveedores, y valor y resultados. Estas dimensiones ayudan a tener una visión integral de los diferentes aspectos que influyen en la gestión exitosa de los servicios.

- **El enfoque de flujo de valor:** ITIL 4 adopta un enfoque de flujo de valor, inspirado en enfoques ágiles y DevOps. Se centra en la entrega continua de valor a través de la colaboración y la automatización, eliminando silos y mejorando la eficiencia y la calidad en todo el ciclo de vida del servicio.
- **Prácticas de gestión de servicios:** ITIL 4 introduce nuevas prácticas y actualiza algunas de las prácticas existentes. Algunas de las nuevas prácticas incluyen la gestión del valor, la gestión de riesgos, la gestión de la configuración, la gestión del conocimiento y la gestión de la continuidad del servicio. Estas prácticas ofrecen orientación detallada sobre cómo realizar actividades y procesos específicos en la gestión de servicios de TI.
- **Integración con enfoques ágiles y DevOps:** ITIL 4 reconoce la importancia de los enfoques ágiles y DevOps en la entrega de servicios de TI rápidos y de alta calidad. Se alinea con estos enfoques y ofrece orientación sobre cómo integrar las prácticas y los principios ágiles y DevOps dentro del marco de trabajo de ITIL.
- **ITIL 4 como una base sólida para la gestión de servicios de TI:** En un entorno empresarial cada vez más digital y ágil. Su enfoque centrado en el valor, la adaptabilidad y la integración con enfoques ágiles y DevOps lo convierten en una herramienta relevante y actualizada para las organizaciones que buscan mejorar la calidad y eficiencia de sus servicios de TI.

Capítulo IV: Planeación

4.1 Explicación y entendimiento del modelo de negocio

Con base en la solución propuesta para el desarrollo de nuestro sistema web se planteó el modelo de negocio y creación del primer boceto de diagrama UML con la finalidad de poder entender el modelo de negocio y el enfoque que se le daría a este proyecto.

4.2 Metodología Ágil

Metodología ágil: El enfoque ágil es ampliamente utilizado en el desarrollo de software y se adapta bien a proyectos individuales. Puedes adoptar metodologías ágiles como Scrum o Kanban para dividir tu proyecto en tareas más pequeñas y manejables. Estas metodologías te permitirán iterar rápidamente, recibir retroalimentación y ajustar tu desarrollo según sea necesario.

4.3 Desarrollo basado en componentes

Desarrollo basado en componentes: Ionic con Angular se basa en el concepto de componentes reutilizables. Puedes aprovechar este enfoque al máximo desarrollando tu aplicación mediante la creación de componentes bien definidos y reutilizables. Esto te permitirá ahorrar tiempo y esfuerzo a medida que avanza el desarrollo de tu aplicación, ya que podrás utilizar y personalizar los componentes en diferentes partes de la app.

4.4 Integración y entrega continuas

Integración y entrega continuas (Continuous Integration and Continuous Delivery, CI/CD): Establecer un flujo de trabajo de CI/CD puede ser beneficioso, incluso en

un proyecto individual. Puedes utilizar herramientas como Jenkins o Travis CI para automatizar las pruebas, compilación y entrega de tu aplicación. Esto garantizará que los cambios se integren y entreguen de manera constante, lo que facilitará el seguimiento y la corrección de errores.

4.5 Versionado de código y control de versiones

Versionado de código y control de versiones: Es importante utilizar un sistema de control de versiones, como Git, para mantener un registro de los cambios realizados en tu código. Utiliza ramas (branch) para trabajar en nuevas funcionalidades o correcciones de errores sin afectar la versión principal. Esto facilitará la colaboración y permitirá un mejor manejo de las diferentes etapas de desarrollo.

Capítulo V: Requerimientos

5.1 Funcionales

5.1.1 Acceso de usuarios al sistema:

- El sistema debe reflejar el nombre y logo de la empresa.
- El sistema permitirá ingresar al sistema por medio de un “Login” (usuario y contraseña).
- En el proceso de registro de un usuario nuevo, el sistema debe solicitar los siguientes datos: Nombre de usuario, correo, contraseña.
- Al terminar el proceso de registro, el sistema enviará un correo de confirmación.

5.1.2 Gestión de SLA

- El sistema debe permitir la gestión de SLA proporcionados por la empresa
- El sistema permitirá agregar SLA
- El sistema permitirá la eliminación de SLA
- El sistema permitirá la edición de SLA

- El sistema mostrará el listado de SLA para seleccionar el deseado y atender algún ticket en específico

Proceso para levantamiento de Tickets en plataforma / app móvil

- El sistema permitirá crear el ticket por medio de un botón.
- Al crear un ticket necesito que me permita agregar el origen.
- Al crear un ticket necesito que me permita agregar el destinatario.
- Al crear un ticket necesito que me permita agregar el asunto.
- Al crear un ticket necesito que me permita agregar descripción de la incidencia.
- Al crear un ticket necesito que me permita agregar la evidencia de la incidencia.
- El sistema permitirá enviar el ticket por medio de un botón.
- El sistema debe permitir la adición de imágenes, archivos para complementar la descripción de la incidencia.

5.1.3 Atención del Ticket

- Al realizar la atención de un ticket, el sistema debe permitir el envío de mensajes entre agente/cliente.
- El sistema administrará las tareas como son los tickets y se canalizan hacia un área en específico si es necesario.
- El sistema debe permitir a los usuarios autorizados cambiar el estado de atención del Ticket (Abierto, pendiente, espera, cerrado).
- El sistema permitirá la revisión del estatus de sus Tickets creados.
- El sistema tendrá funciones administrativas.

- Administra las tareas como son los Tickets y se canalizan hacia un área en específico si es necesario.
- Catálogos unificados.
- El sistema mostrará una opción de que el cliente u/o usuario encuentre específicamente lo que necesita dentro de nuestro sistema.
- Gestión unificada de tareas.
- Se puede crear un ticket y ver el estatus de revisión de algún otro ticket
- Automatización inteligente.
- Si se requiere alguna atención de ticket básico como cambio de contraseña y búsqueda de usuario no es necesario levantar un ticket lo puede hacer solo desde la plataforma.
- Administración de datos.
- Niveles de prioridad y/o autorización.

5.2 Requerimientos no funcionales

Requerimientos no funcionales del producto		
No.	Categoría	Descripción
1	Seguridad	Al terminar el registro, se deberá enviar un correo de validación para el nuevo administrador.
2	Seguridad	Los permisos de acceso al sistema podrán ser cambiados solamente por el administrador de acceso a datos.
3	Seguridad	Todos los sistemas deben respaldarse cada 24 horas. Los respaldos deben ser almacenados en una localidad segura ubicada en un edificio distinto al que reside el sistema.
4	Seguridad	Todas las comunicaciones externas entre servidores de datos, aplicación y cliente del sistema deben estar encriptadas utilizando el algoritmo RSA.
5	Eficiencia	El sistema debe ser capaz de procesar N datos por segundo.
6	Eficiencia	Toda funcionalidad del sistema y transacción de negocio debe responder al usuario en menos de 5 segundos.

7	Eficiencia	Los datos modificados en la base de datos deben ser actualizados para todos los usuarios que acceden en menos de 2 segundos.
8	Usabilidad	Para hacer uso del sistema web el usuario debe contar con conexión a internet.
9	Usabilidad	El sistema debe contar con manuales de usuario estructurados adecuadamente.
10	Usabilidad	El sistema debe proporcionar mensajes de error que sean informativos y orientados a usuario final.
11	Disponibilidad	El sistema debe tener una disponibilidad del 99,99% de las veces en que un usuario intente accederlo.
12	Disponibilidad	El tiempo para iniciar o reiniciar el sistema no podrá ser mayor a 5 minutos.
13	Disponibilidad	La tasa de tiempos de falla del sistema no podrá ser mayor al 0,5% del tiempo de operación total.
14	Disponibilidad	El promedio de duración de fallas no podrá ser mayor a 15 minutos.
15	Disponibilidad	El promedio de duración de fallas no podrá ser mayor a 15 minutos.
16	Disponibilidad	La probabilidad de falla del Sistema no podrá ser mayor a 0,05.
17	Organizacionales	El procedimiento de desarrollo de software debe desarrollarse bajo el marco de referencia ITIL y mejora continua.
18	Organizacionales	El sistema debe ser desarrollado utilizando las herramientas Visual Studio y Android Studio bajo el lenguaje de programación java, Frame Angular más Ionic.

5.3 Historias de usuario

N° 3	Nombre de historia: registro de administrador		Sprint: 1	Tiempo estimado:
Módulo	Acceso de usuarios	Nivel de prioridad	Alto	
Dependencia		Programador asignado		
Descripción				
Usuario (Como)	Mesa de servicio			
Objetivo (Quiero)	Un apartado para a agregar y registrar datos de los administradores (Correo, nombre, apellido, teléfono, nombre de usuario, contraseña)			
Beneficio (Para poder)	Tener control sobre las personas asignadas y tener más de un administrador			
Proceso	Añadir botón que envíe formulario para llenar datos del administrador			
Ponderación	5			

Figura 1. Historia de usuario 3

N° 6	Nombre de historia: agregar agentes		Sprint: 1	Tiempo estimado:
Módulo	Acceso de usuarios	Nivel de prioridad	Alto	
Dependencia		Programador asignado		
Descripción				
Usuario (Como)	Administrador			
Objetivo (Quiero)	Agregar agentes y registrar sus datos (Nombre, apellido, teléfono, nombre de usuario, correo, contraseña, área de soporte)			
Beneficio (Para poder)	Tener control sobre los agentes, cuantos y en que área están registrados			
Proceso	Crear botón para añadir agente, al dar click en el botón muestra formulario para llenar con datos solicitados			
Ponderación	8			

Figura 2. Historia de usuario 6

N° 8	Nombre de historia: gestionar datos de agentes		Sprint: 1	Tiempo estimado:
Módulo	Acceso de usuarios	Nivel de prioridad	Alto	
Dependencia	Historia 6	Programador asignado	Luis Fernando Mendieta	
Descripción				
Usuario (Como)	Administrador			
Objetivo (Quiero)	Gestionar los datos de los agentes			
Beneficio (Para poder)	Tener gestión sobre los agentes, tener lista de agentes en el sistema			
Proceso	Apartado exclusivo para visualizar y editar datos de los agentes			
Ponderación	8			

Figura 3. Historia de usuario 8

N° 10	Nombre de historia: registro de usuarios		Sprint:	Tiempo estimado:
Módulo	Acceso de usuarios	Nivel de prioridad	Alto	
Dependencia		Programador asignado		
Descripción				
Usuario (Como)	Usuario			
Objetivo (Quiero)	Registrar datos (correo electrónico, nombre, apellidos, teléfono, dirección, Id de trabajador o cliente, departamento o rol)			
Beneficio (Para poder)	Ingresar al sistema			
Proceso	Desde la página mostrar opción de registro en caso de no contar con cuenta			
Ponderación	5			

Figura 4. Historia de usuario 10

N° 13	Nombre de historia: creación de ticket		Sprint:	Tiempo estimado:
Módulo	Levantamiento del Ticket	Nivel de prioridad	Alto	
Dependencia		Programador asignado	Oscar Netzahualcoyotl	
Descripción				
Usuario (Como)	Administrador, agente y usuario			
Objetivo (Quiero)	Que el sistema me permita crear tickets			
Beneficio (Para poder)	Indicar la incidencia			
Proceso	El sistema debe otorgar la opción de "levantar ticket" por medio de un botón que redirige al usuario a un formulario.			
Ponderación	3			

Figura 5. Historia de usuario 13

N° 14	Nombre de historia: correo de notificación		Sprint:	Tiempo estimado:
Módulo	Levantamiento del Ticket	Nivel de prioridad	Medio	
Dependencia	Historia 13	Programador asignado	Oscar Netzahualcoyotl	
Descripción				
Usuario (Como)	Administrador, agente y usuario			
Objetivo (Quiero)	Tener el ID del ticket que he creado			
Beneficio (Para poder)	Dar seguimiento del Ticket, buscar su status con el ID			
Proceso	Al crear un Ticket, automáticamente se genera y se asigna un ID			
Ponderación	34			

Figura 6. Historia de usuario 14

N° 15	Nombre de historia: Llenar ticket		Sprint:	Tiempo estimado:
Módulo	Levantamiento del Ticket	Nivel de prioridad	Alto	
Dependencia	Historia 14	Programador asignado		
Descripción				
Usuario (Como)	Administrador, agente y usuario			
Objetivo (Quiero)	Que el sistema permita añadir descripción de la incidencia con adición de imágenes, archivos para complementar la descripción de la incidencia.			
Beneficio (Para poder)	Describir y explicar la incidencia a detalle			
Proceso	El sistema debe otorgar las opciones de "destinatario (correo)", "asunto (descripción de incidencia)", y/o adjuntar archivos.			
Ponderación	3			

Figura 7. Historia de usuario 15

N° 20	Nombre de historia: Atención del Ticket		Sprint:	Tiempo estimado:
Módulo	Levantamiento del Ticket	Nivel de prioridad	Alto	
Dependencia		Programador asignado		
Descripción				
Usuario (Como)	Agente, mesa de servicio			
Objetivo (Quiero)	Tener una lista desplegable con los agentes registrados			
Beneficio (Para poder)	Seleccionar uno para poder asignarlo			
Proceso	Crear lista desplegable en el formulario para llenar descripción del Ticket			
Ponderación				

Figura 8. Historia de usuario 20

5.3.1 Diagramas BPM-Ticketing

Diagrama BPM-Ticketing para el levantamiento y creación de tickets.

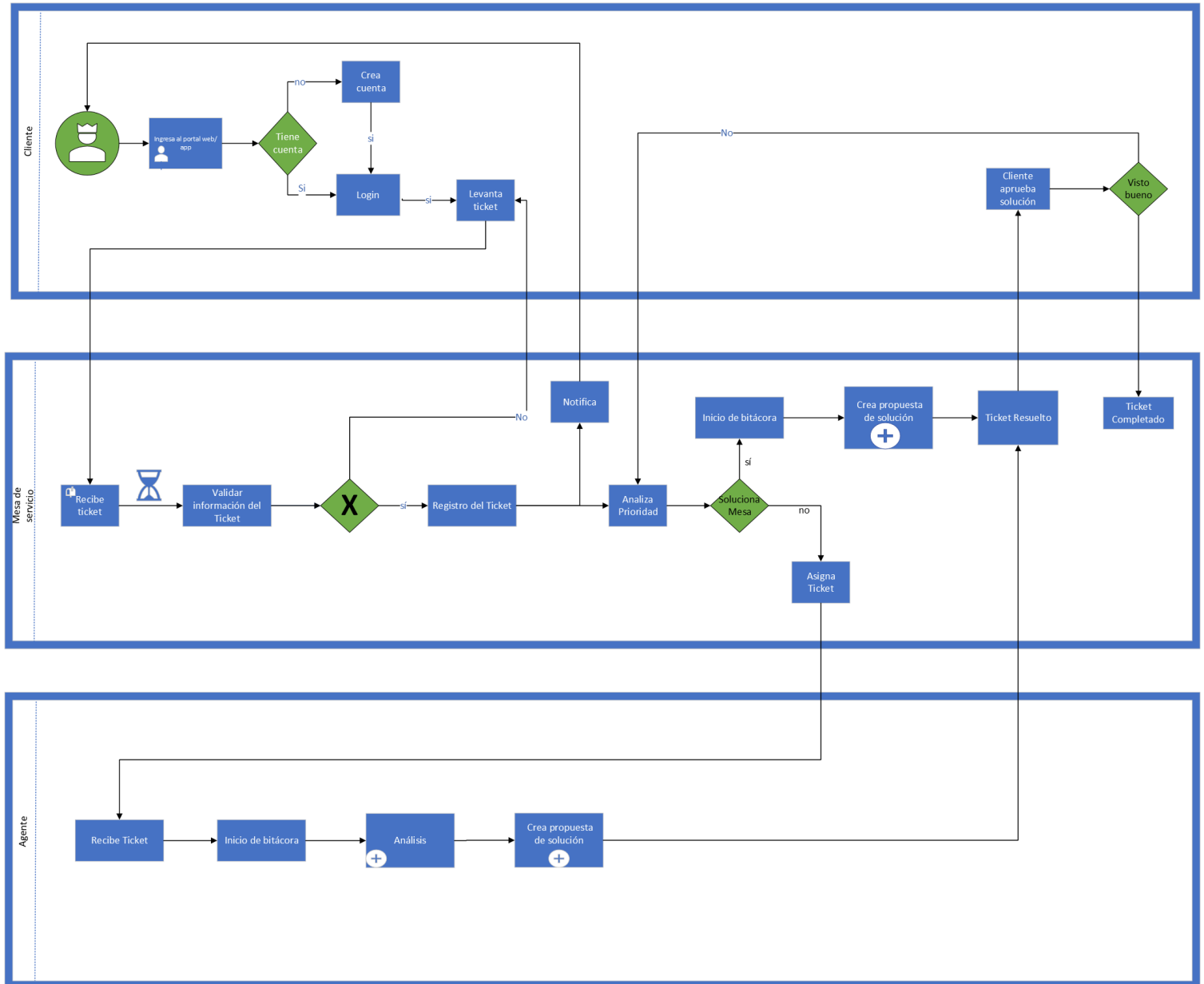


Figura 9. Proceso general de levantamiento de tickets

Diagrama BPM-Ticketing para el seguimiento y atención de tickets.

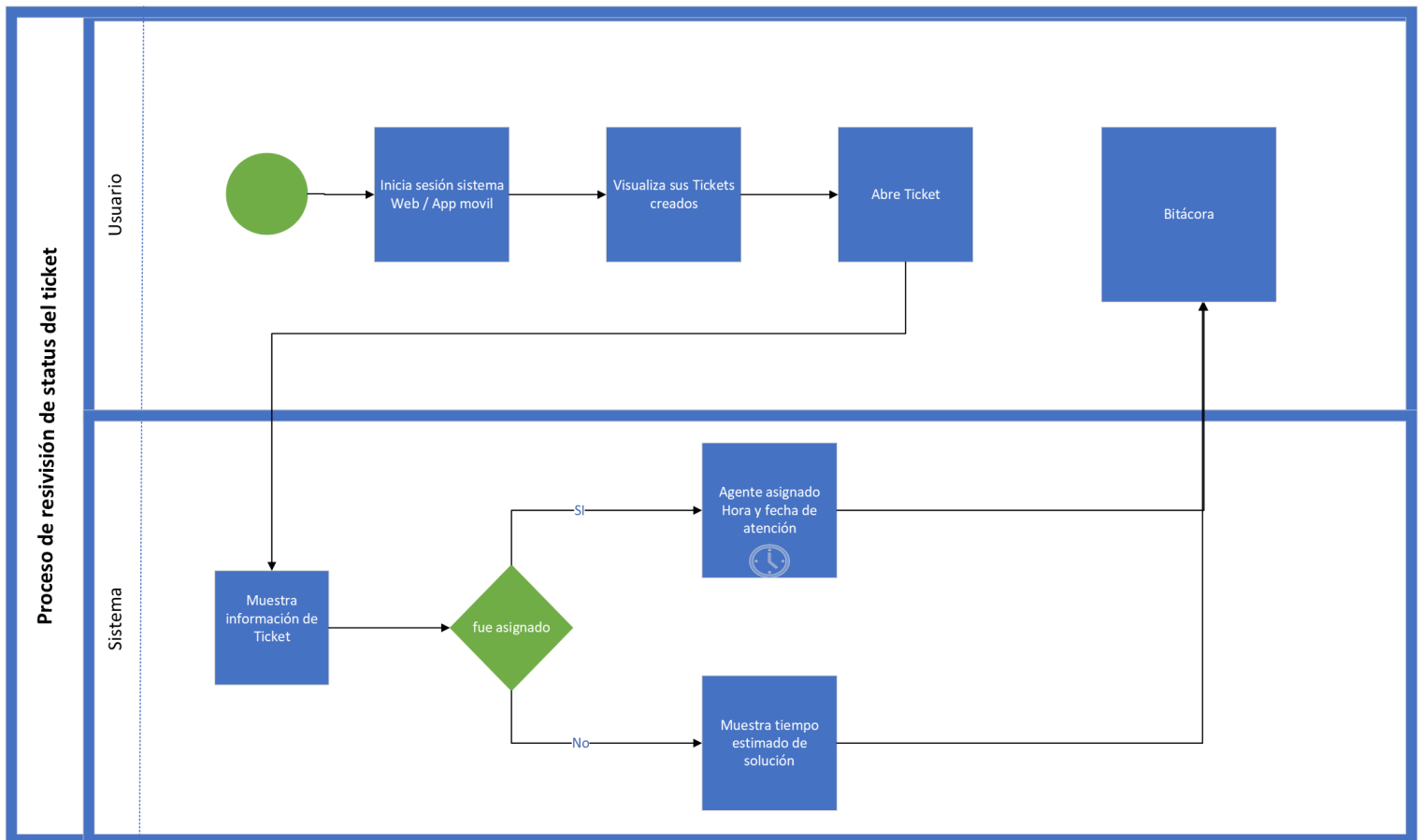


Figura 10. Proceso de revisión de status

Capítulo VI: Tecnologías utilizadas

6.1 Modelo cliente servidor

Un servidor es un sistema que contiene datos o proporciona recursos a los que deben acceder otros sistemas de la red. Un cliente es un sistema que solicita servicios o datos de un servidor.

Los tipos de servidor comunes son servidores de archivos que almacenan archivos, servidores de nombres que almacenan nombres y direcciones, servidores de aplicaciones que almacenan programas y aplicaciones y servidores de impresión que planifican y dirigen los trabajos de impresión al destino.

Un cliente puede solicitar código de programa actualizado o el uso de aplicaciones de un servidor de código. Para obtener un nombre o una dirección, un cliente se pone en contacto con un servidor de nombres. Un cliente también puede solicitar archivos y datos para la entrada de datos, las consultas o la actualización de registros de un servidor de archivos.

6.2 Modelo vista controlador (MVC)

En líneas generales, MVC es una propuesta de arquitectura del software utilizada para separar el código por sus distintas responsabilidades, manteniendo distintas capas que se encargan de hacer una tarea muy concreta, lo que ofrece beneficios diversos.

MVC se usa inicialmente en sistemas donde se requiere el uso de interfaces de usuario, aunque en la práctica el mismo patrón de arquitectura se puede utilizar para distintos tipos de aplicaciones. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, Model, Views & Controllers, si lo prefieres en inglés.

6.3 Spring Boot

Spring Boot facilita la creación de aplicaciones basadas en Spring independientes y de grado de producción que puede "simplemente ejecutar".

Librerías de Spring boot:

- JPA
- Spring Web
- MySQL Driver

Capítulo VII: codificación del sistema

7.1 Diseño de la base de datos

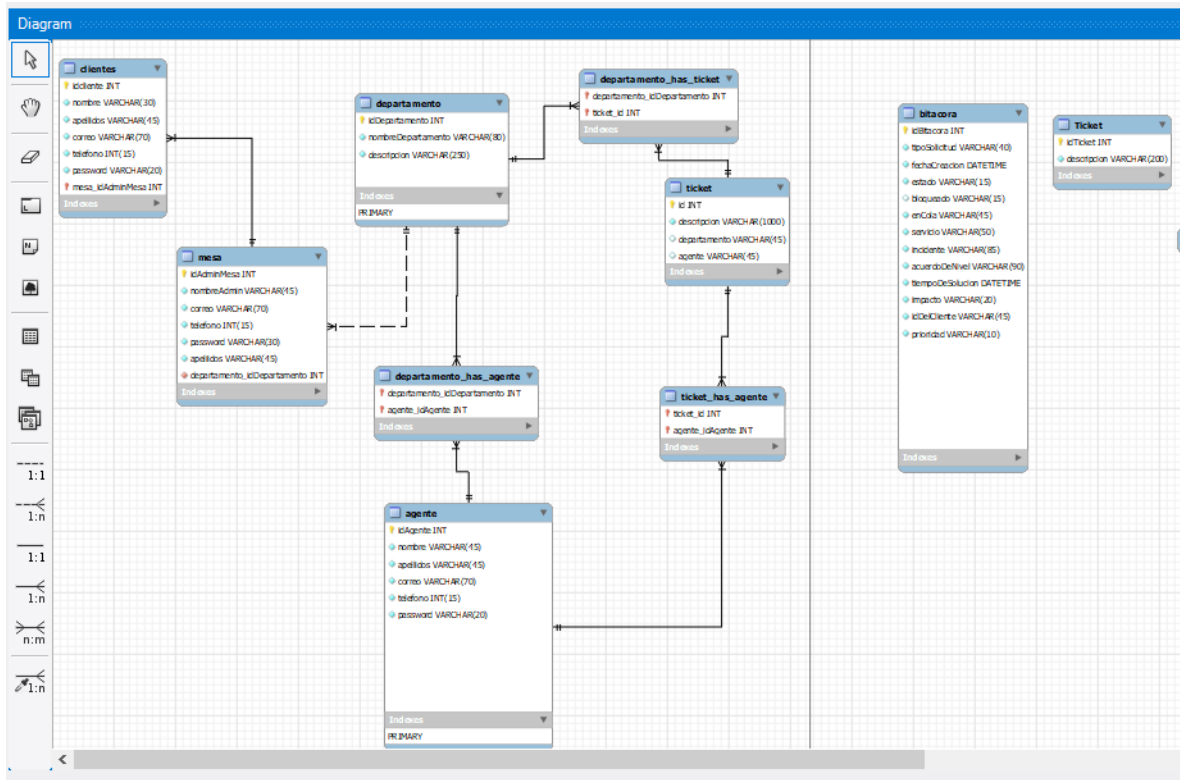


Figura 11. Diagrama Entidad Relación de la base de datos

7.2 Creación de paquetes para la fase

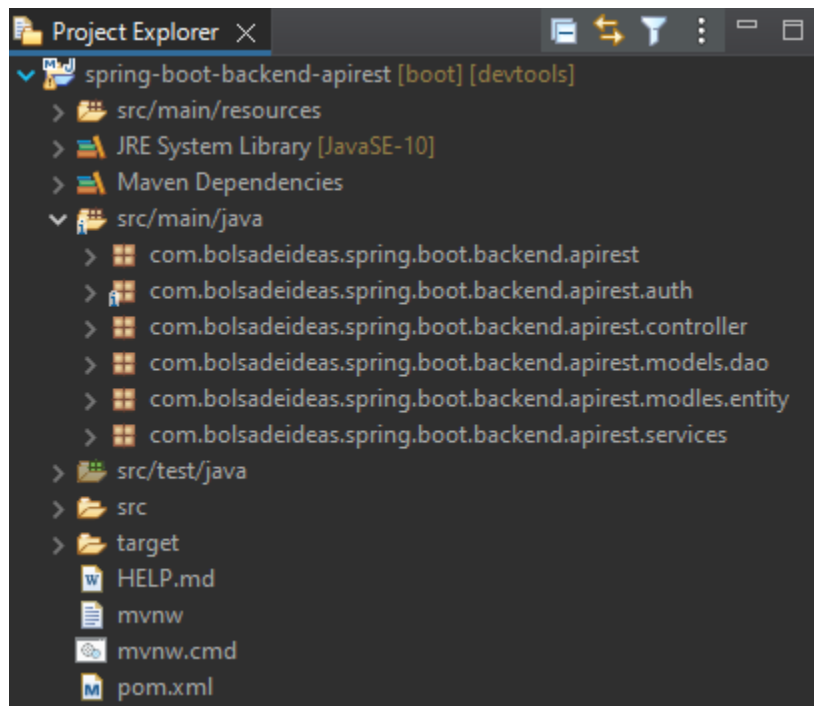


Figura 12. Paquetes para el desarrollo

7.3 Conexión de Spring para la exportación de entidades a MySQL

Dentro del archivo application.properties, escribiremos la conexión con MySQL

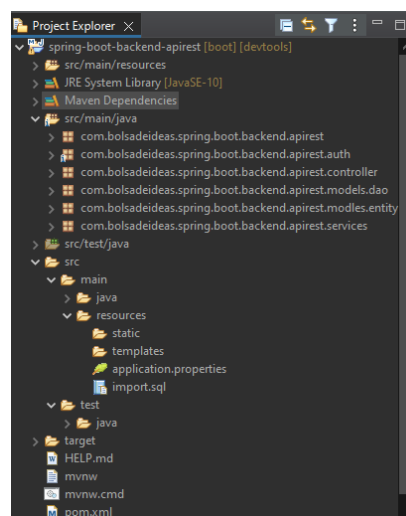


Figura 13. Ubicación del archivo application properties

```

1
2 spring.datasource.url=jdbc:mysql://localhost/db_springboot_backend
3 spring.datasource.username=root
4 spring.datasource.password=Fito.2510
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6 spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
7 spring.jpa.hibernate.ddl-auto=update
8 logging.level.org.hibernate.SQL=debug
9

```

Figura 14. Código para conexión de base de datos

7.4 Desarrollo entidades con atributos

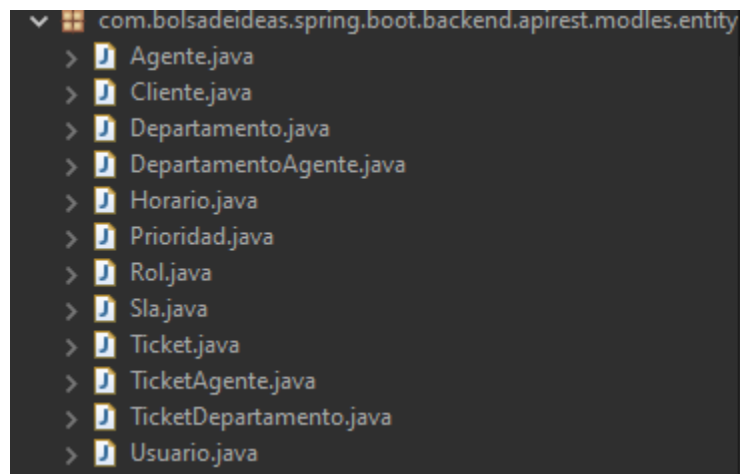


Figura 15. Entidades

```

30 import java.util.Date;
16
17 @Entity
18 @Table(name="clientes")
19 public class Cliente implements Serializable{
20
21
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     private Long id;
25
26     @Column(nullable=false)
27     private String nombre;
28     private String apellido;
29     @Column(nullable=false, unique = true)
30     private String username;
31     @Column(nullable = false, unique = true)
32     private String email;
33     private String password;
34
35     @Column(name="create_at")
36     @Temporal(TemporalType.DATE)
37     private Date createAt;
38
39
40     //ya se está asignando el evento
41     @PrePersist
42     public void prePersist() {
43         createAt = new Date();
44     }

```

Figura 16. Atributos de la clase Cliente.java

```

@Entity
@Table(name="agentes")
public class Agente implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nombre;
    private String apellido;

    @Column(nullable = false, unique = true, length = 20)
    private String username;

    @Column(length = 60)
    private String password;

    private String telefono;
    private boolean enabled = true;

    @Column(nullable = false, unique = true)
    private String email;

```

Figura 17. Atributos de la clase Agente.java

```

@Entity
@Table(name="departamentos")
public class Departamento implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String nombre;
    private String descripcion;
    private boolean enabled = true;

    @Column(name="create_at")
    @Temporal(TemporalType.DATE)
    private Date createAt;

    @PrePersist
    public void prePersist() {
        createAt = new Date();
    }

    //RELACION DEPARTAMENTO-AGENTE
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "departamento")
    private Set<DepartamentoAgente> agenteDepartamento = new HashSet<>();

```

Figura 18. Atributos de la clase Departamento.java

```

@Entity
@Table(name="roles")
public class Rol implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, length = 40)
    private String nombre;

    // @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "rol")
    // private Set<AgenteRol> agenteRoles = new HashSet<>();

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}

```

Figura 19. Atributos de la clase Rol.java

Decoradores @ / Anotaciones @

- @Entity: especifica que la clase será tomada como entidad para la base de datos.
- @Table: con este decorador podemos especificar el nombre de la tabla.
- @Column: sirve para declarar si ese atributo puede ser nulo, único, etc.
- @Id: especifica que ese atributo debe resaltar con id
- @GeneratedValue(strategy=GenerationType.IDENTITY): hace que el id, sea auto-incrementable
- @Prepersist: nos ayudará a registrar en automático la fecha del registro ejecutado
- @ManyToMany: decorador para especificar la relación muchos a muchos
- @JoinTable: nos permitirá crear la tabla de la relación, especificando el nombre de los atributos que estarán dentro de la relación

7.5 Relaciones entre las entidades

```

@ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
@JoinTable(name = "agentes_rols", joinColumns=@JoinColumn(name = "agente_id"), inverseJoinColumns = @JoinColumn(name = "role_id") )
private List<Rol> roles;

```

Figura 20. Ejemplo de relaciones entre entidades

Cuando tenemos relaciones entre objetos, los ORM al momento de traer a memoria un objeto también van a traer todos los objetos que se relacionan con el, y esto puede ser una ventaja o una desventaja dependiendo el contexto y del momento en que se haga la obtención de estos objetos. En este posteo vamos a ver los dos tipos de obtención que nos provee JPA y lo que sucede cuando usamos cada uno.

Los dos tipos de carga que tenemos con JPA, son los llamados Eager y Lazy. Estos nombres nos dan un indicio sobre que comportamiento tiene cada uno.

- Eager: La carga de los objetos de la relación se produce en el mismo momento
- Lazy: La carga de los objetos de la relación se producen a demanda, es decir, cuando un cliente los solicita.

7.6 Patrón DAO

En una aplicación J2EE necesitamos acceder a datos, ya sea por persistencia (hibernate, jdo, iBatis...), jdbc, ficheros de texto, LDAP, etc. Esto puede suponer un problema, pues la forma de acceder a los datos depende del fabricante y del tipo de almacenamiento que estamos accediendo.

Los componentes de nuestra aplicación deben ser transparentes en la medida de lo posible al actual sistema de persistencia o fuente de datos para permitir migraciones entre distintos fabricantes, distintos tipos de almacenamiento y diferentes fuentes de datos. Supongamos que en un momento dado queremos cambiar el motor de persistencia. Siguiendo este módulo será mucho más fácil.

El patrón DAO viene a resolver este problema usando un Objeto de Acceso a Datos para abstraer y encapsular el acceso a los datos. Un DAO maneja la conexión con la fuente de datos para obtener y guardar los datos.

Un DAO siempre realiza operaciones atómicas contra la base de datos, nunca son necesarias las transacciones. Claros ejemplos de esto serían búsquedas por una

clave, creación, actualización y borrado de un registro, obtener todos los registros y cualquier otra operación que vayamos a realizar muy a menudo.

Normalmente se crea un DAO por cada Objeto que usemos en nuestra aplicación. Objeto puede ser muchas cosas, según estemos usando puro JDBC o un framework de persistencia.

```
package com.bolsadeideas.spring.boot.backend.apirest.models.dao;

import org.springframework.data.repository.CrudRepository;

import com.bolsadeideas.spring.boot.backend.apirest.modles.entity.Cliente;

public interface IClienteDao extends CrudRepository<Cliente, Long> {

}
```

Figura 21. Interface IClienteDao.java

Se hereda *CrudRepository* con los atributos Cliente (entidad) y Long (tipo de dato del ID)

Dentro de esta interface encontraremos los siguientes métodos:

- Save:

```
<S extends T> S save(S entity);

/**
 * Saves all given entities.
 *
 * @param entities must not be {@literal null} nor must it contain {@literal null}.
 * @return the saved entities; will never be {@literal null}. The returned {@literal Iterable}
 *         as the {@literal Iterable} passed as an argument.
 * @throws IllegalArgumentException in case the given {@link Iterable} or one of its
 *         elements is {@literal null}.
 * @throws OptimisticLockingFailureException when at least one entity uses optimistic locking
 *         attribute with a different value from that found in the persistence store.
 *         entity is assumed to be present but does not exist in the database.
 */
```

Figura 22. Método save

- SaveAll:

```
<S extends T> Iterable<S> saveAll(Iterable<S> entities);

/**
 * Retrieves an entity by its id.
 *
 * @param id must not be {@literal null}.
 * @return the entity with the given id or {@literal Optional#empty()} if no entity is found.
 * @throws IllegalArgumentException if {@literal id} is {@literal null}.
 */
```

Figura 23. Método saveAll

- FindAll:

```

Iterable<T> findAll();

/**
 * Returns all instances of the type {@code T} with the given IDs.
 * <p>
 * If some or all ids are not found, no entities are returned for these IDs.
 * <p>
 * Note that the order of elements in the result is not guaranteed.
 *
 * @param ids must not be {@literal null} nor contain any {@literal null} values.
 * @return guaranteed to be not {@literal null}. The size can be equal or less than
 *       {@literal ids}.
 * @throws IllegalArgumentException in case the given {@link Iterable ids} or one of
 */

```

Figura 24. Método findAll

- FindAllById:

```

Iterable<T> findAllById(Iterable<ID> ids);

/**
 * Returns the number of entities available.
 *
 * @return the number of entities.
 */

```

Figura 25. Método findAllById

- DeleteById

```

void deleteById(ID id);

/**
 * Deletes a given entity.
 *
 * @param entity must not be {@literal null}.
 * @throws IllegalArgumentException in case the given entity is {@literal null}.
 * @throws OptimisticLockingFailureException when the entity uses optimistic locking and
 *       a different value from that found in the persistence store. Also thrown if
 *       present but does not exist in the database.
 */

```

Figura 26. Método deleteById

- Delete

```

void delete(T entity);

/**
 * Deletes all instances of the type {@code T} with the given IDs.
 * <p>
 * Entities that aren't found in the persistence store are silently ignored.
 *
 * @param ids must not be {@literal null}. Must not contain {@literal null} elements.
 * @throws IllegalArgumentException in case the given {@literal ids} or one of its elem
 * @since 2.5
 */

```

Figura 27. Método delete

7.7 Servicios

En el paquete de services, se crean interfaces donde se invocan a los métodos de CrudRepository y posteriormente poder inyectarlos a la clase donde se especifica el servicio.

Se escriben los métodos que se usarán para inyectar en el servicio:

```

import java.util.List;

public interface IClienteServicie {
    public List<Cliente> findAll();

    public Cliente findById(Long id);

    public Cliente save(Cliente cliente);

    public void delete(Long id);
}

```

Figura 28. Métodos de CrudRepository

Anotaciones para servicios

@Service: Spring @Service es una de las anotaciones más habituales de Spring Framework. Se usa para construir una clase de Servicio que habitualmente se conecta a varios repositorios y agrupa su funcionalidad.

@Autowired: Spring @Autowired es una de las anotaciones más habituales cuando trabajamos con Spring Framework ya que se trata de la anotación que permite inyectar unas dependencias con otras dentro de Spring.

@Override: El concepto de Java Override o overriding es uno de los conceptos que cuesta más entender cuando una comienza a trabajar con programación orientada a objeto. ¿Para qué sirve el overriding o polimorfismo dinámico? En Java en muchas ocasiones nos encontramos con situaciones en las cuales tenemos una jerarquía de clases en la que existe diferencia en cuanto a la implementación de alguno de sus métodos comunes.

@Transactional: El uso de Spring @Transactional readonly es una de las características que en muchas ocasiones nos olvidamos al manejar Spring Framework. Spring nos provee de las capacidades de inyección de gestión transaccional distribuida y transparente para nuestros repositorios y servicios.

```
@Service
public class ClienteServiceImpl implements IClienteService{

    //CONTRATO DEL CRUD

    //autowired es la notacion para inyectar a "clienteDao"
    @Autowired
    private IClienteDao clienteDao;

    @Override
    @Transactional(readOnly = true)
    public List<Cliente> findAll() {
        // TODO Auto-generated method stub
        return (List<Cliente>) clienteDao.findAll();
    }

    @Override
    @Transactional(readOnly =true)
    public Cliente findById(Long id) {
        // TODO Auto-generated method stub
        return clienteDao.findById(id).orElse(null) ;
    }

    @Override
    @Transactional
    public Cliente save(Cliente cliente) {
        // TODO Auto-generated method stub
        return clienteDao.save(cliente);
    }
}
```

```
@Override
@Transactional
public void delete(Long id) {
    clienteDao.deleteById(id);
}
```

Figura 29. Código de la implementación de la interface

7.8 Controlador@

@Anotaciones para el controlador

- **@CrossOrigins:** En Spring boot tenemos cors, que significa intercambio de recursos entre orígenes, básicamente significa que el host que representa la interfaz de usuario y el host que sirve los datos son diferentes. Significa que ambos se ejecutan en un host diferente; en este tipo de casos podemos encontrar este problema al ejecutar la aplicación.

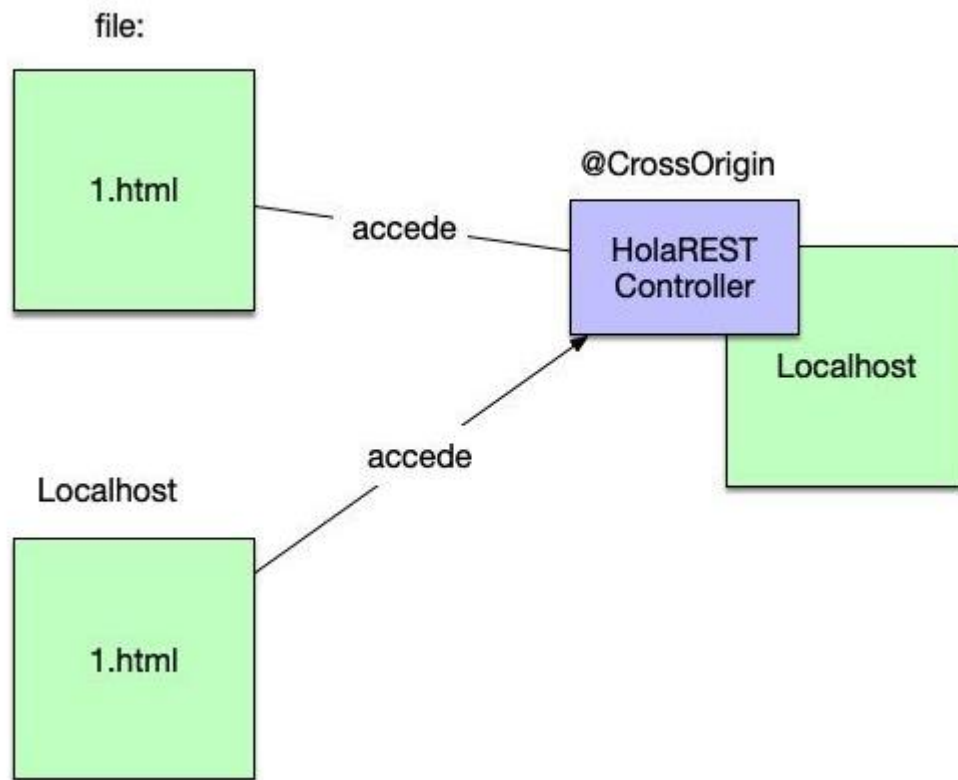


Figura 30. Visualización abstracta del proceso de CrossOrigin

```
//decorador que permite conexion entre Angular y spring
@CrossOrigin(origins= {"http://localhost:4200"})
```

Figura 31. Conexión con el front-end

- **@RestController:** La anotación Spring RestController es una anotación conveniente que a su vez está anotada con @Controller y @ResponseBody. Esta anotación se aplica a una clase para marcarla como controlador de solicitudes. La anotación Spring RestController se utiliza para crear servicios web RESTful utilizando Spring MVC.
- **@RequestMapping:** Es una de las anotaciones Spring MVC más utilizadas. org.springframework.web.bind.annotation.RequestMapping La anotación de mapeo se utiliza para mapear solicitudes web en clases de controlador y/o métodos de controlador específicos.

```
28 @RequestMapping("/api")
```

Figura 32. Especificación de ruta de mapeo

Actualmente, Spring admite cinco tipos de anotaciones incorporadas para manejar diferentes tipos de métodos de solicitud HTTP entrantes que son GET, POST, PUT, DELETE y PATCH. Estas anotaciones son:

- **@GetMapping:** Es una anotación de Spring Framework que aparece con Spring 4.3 y nos permite simplificar el manejo de los diferentes métodos de Spring MVC.
- **@PostMapping:** Anotación para asignar solicitudes HTTP POST a métodos de controlador específicos.

Específicamente, `@PostMapping` es una anotación compuesta que actúa como un atajo para `@RequestMapping(method = RequestMethod.POST)`.

Todas las anotaciones de mapeo de solicitudes admiten los siguientes atributos, pero deben usarse cuando sea apropiado para el tipo de solicitud:

Ruta o valor: especifica el URI de mapeo. Un método de controlador que no está asignado explícitamente a ninguna ruta se asigna efectivamente a una ruta vacía.

- **@PutMapping:** Una petición PUT debe adjuntar a la url la clave primaria del recurso que deseamos actualizar. En este caso vamos a suponer que es el nombre . Aparte de esto debe enviar en formato JSON o XML el nuevo recurso que vamos a almacenar. De tal forma que a través de la clave primaria podamos elegir el recurso antiguo y actualizarlo por el nuevo.
- **@DeleteMapping:** Anotación para asignar solicitudes HTTP DELETE a métodos de controlador específicos.

Específicamente, `@DeleteMapping` es una anotación compuesta que actúa como un atajo para `@RequestMapping(method = RequestMethod.DELETE)`.

- **@PathVariable:** es la anotación que nos sirve dentro de Spring framework para configurar variables dentro de los propios segmentos de la URL algo que a nivel de Arquitecturas REST es imprescindible. Vamos a ver un par de

ejemplos de cómo construir unas urls REST que se apoyen en este tipo de anotación.

```
@GetMapping("/clientes/{id}")  
public ResponseEntity<?> show(@PathVariable Long id) {
```

Figura 33. Asignación del PathVariable para la ruta específica del Id

7.9 Principales servicios del CRUD

Listar todos los datos de la tabla:

```
//decorador que permite conexion entre Angular y spring  
@CrossOrigin(origins= {"http://localhost:4200"})  
@RestController  
@RequestMapping("/api")  
public class ClienteRestController {  
  
    @Autowired  
    private IClienteService clienteService;  
  
    @GetMapping("/clientes")  
    public List<Cliente> index(){  
        return clienteService.findAll();  
    }  
}
```

Figura 34. Listado de datos

Listar por Id:

```
@GetMapping("/clientes/{id}")
public ResponseEntity<?> show(@PathVariable Long id) {

    Cliente cliente = null;

    Map<String, Object> response = new HashMap<>();

    try {
        cliente = clienteService.findById(id);
    } catch (DataAccessException e) {
        response.put("mensaje", "Error al consultar la base de datos");
        response.put("Error", e.getMessage().concat(": ").concat(e.getMostSpecificCause().getMessage()));
        return new ResponseEntity<Map<String, Object>>(response, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    if (cliente == null) {
        response.put("mensaje", "El cliente ID: " .concat(id.toString()).concat(" no existe"));
        return new ResponseEntity<Map<String, Object>>(response, HttpStatus.NOT_FOUND);
    }

    return new ResponseEntity<Cliente>(cliente, HttpStatus.OK);
}
```

Figura 35. Listar un cliente por el Id

En la ilustración 26 podemos observar la implementación de mensajes de errores para ser mostrados dentro del servidor y consumidos por el front-end.

Crear:

```
@PostMapping("/clientes")
public ResponseEntity<?> create(@RequestBody Cliente cliente) {

    Cliente clienteNew = null;
    Map<String, Object> response = new HashMap<>();

    try {
        clienteNew = clienteService.save(cliente);
    } catch (DataAccessException e) {
        response.put("mensaje", "Error al realizar la inserción en la base de datos");
        response.put("error", e.getMessage().concat(": ").concat(e.getMostSpecificCause().getMessage()));
        return new ResponseEntity<Map<String, Object>>(response, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    response.put("mensaje", "Cliente creado con éxito!");
    response.put("Cliente", clienteNew);
    return new ResponseEntity<Map<String, Object>>(response, HttpStatus.CREATED);
}
```

Figura 36. Crear Cliente

Actualizar:

```
84 @PutMapping("/clientes/{id}")
85 public ResponseEntity<Map<String, Object>> update(@RequestBody Cliente cliente, @PathVariable Long id) {
86
87     Cliente clienteActual = clienteService.findById(id);
88     Cliente clienteUpdated = null;
89
90     Map<String, Object> response = new HashMap<>();
91
92     if (clienteActual == null) {
93         response.put("mensaje", "Error: el cliente no pudo editar, el cliente ID: " + id.toString().concat(" no existe"));
94         return new ResponseEntity<Map<String, Object>>(response, HttpStatus.NOT_FOUND);
95     }
96
97     try {
98         clienteActual.setApellido(cliente.getApellido());
99         clienteActual.setNombre(cliente.getNombre());
100         clienteActual.setEmail(cliente.getEmail());
101         clienteActual.setPassword(cliente.getPassword());
102         clienteActual.setUsername(cliente.getUsername());
103
104         clienteUpdated = clienteService.save(clienteActual);
105
106     } catch (DataAccessException e) {
107         response.put("mensaje", "Error al actualizar en la base de datos");
108         response.put("error", e.getMessage().concat(": ").concat(e.getMostSpecificCause().getMessage()));
109         return new ResponseEntity<Map<String, Object>>(response, HttpStatus.INTERNAL_SERVER_ERROR);
110     }
111
112     response.put("Mensaje: ", "el cliente ha sido actualizado con éxito!");
113 }
```

Figura 37. Actualizar con manejo de errores

Eliminar:

```
@DeleteMapping("/clientes/{id}")
public ResponseEntity<?> delete (@PathVariable Long id) {

    Map<String, Object> response = new HashMap<>();

    try {
        clienteService.delete(id);
    } catch (DataAccessException e) {
        response.put("mensaje", "Error al eliminar en la base de datos");
        response.put("error", e.getMessage().concat(": ").concat(e.getMostSpecificCause().getMessage()));
        return new ResponseEntity<Map<String, Object>>(response, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    response.put("mensaje: ", "el cliente ha sido eliminado con éxito");
    return new ResponseEntity<Map<String, Object>>(response, HttpStatus.OK);
}
```

Figura 38. Eliminar con manejo de errores

7.10 Ejecución del servidor

Postman

Postman es una plataforma API para crear y utilizar API. Postman simplifica cada paso del ciclo de vida de la API y agiliza la colaboración para que puedas crear mejores API y más rápido.

A continuación, utilizaremos Postman para realizar consultas al servidor:

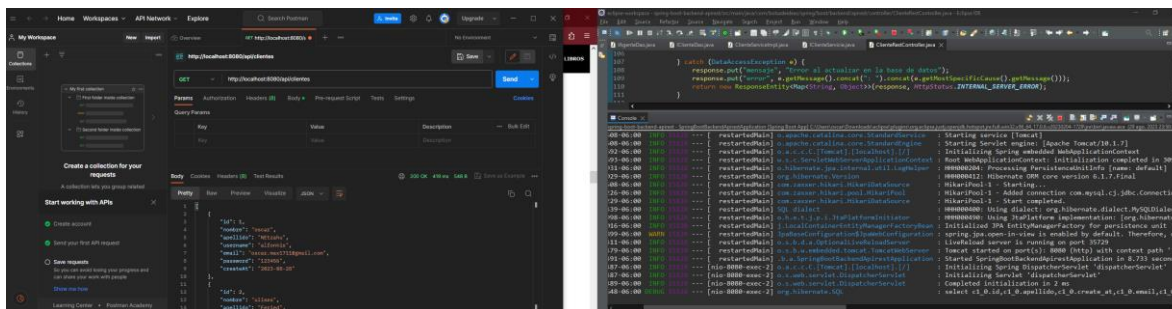


Figura 39. Servidor corriendo y primera petición

Cada que realizamos una petición al servidor nos aparecerá un mensaje de acuerdo a la petición realizada:

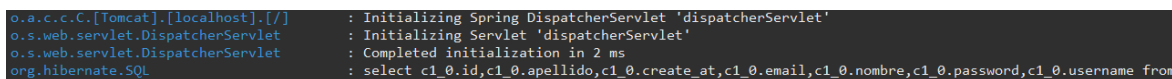


Figura 40. Resultado de la petición por parte del servidor

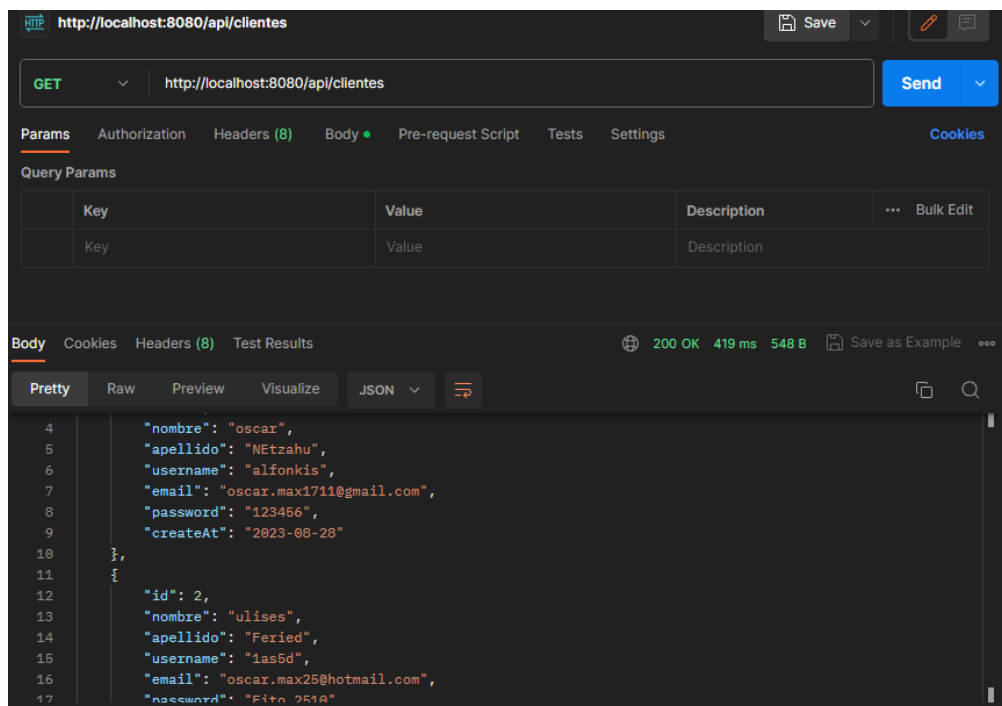


Figura 41. Datos que arroja la petición en Postman

También podemos verlo en el navegador:

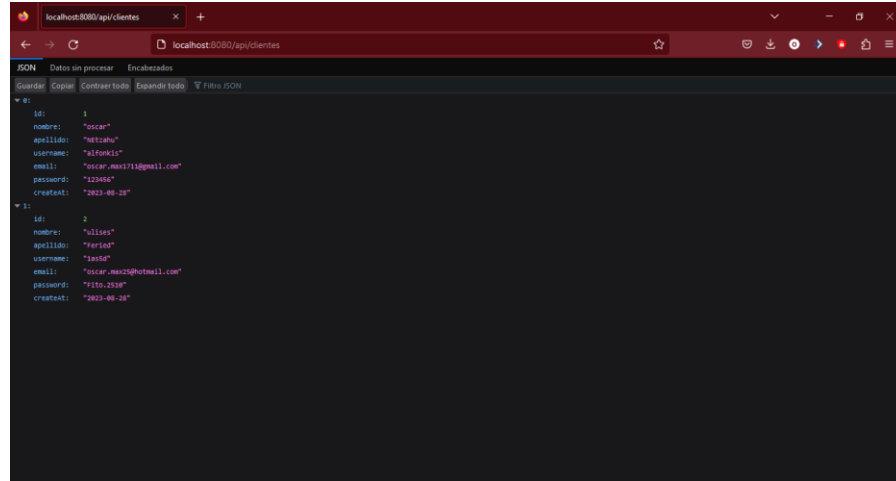


Figura 42. Listado desde el navegador

Como podemos ver, el listado de datos nos lo arroja en forma JSON.

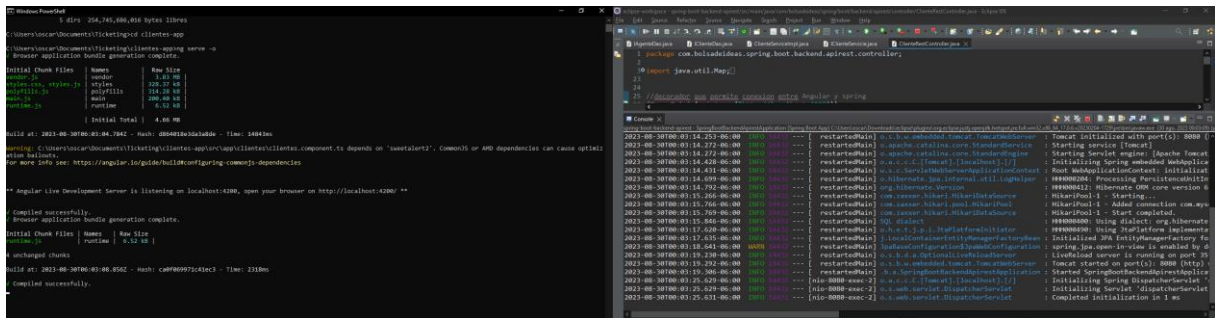


Figura 43. Servidor Front-End y Back-End corriendo

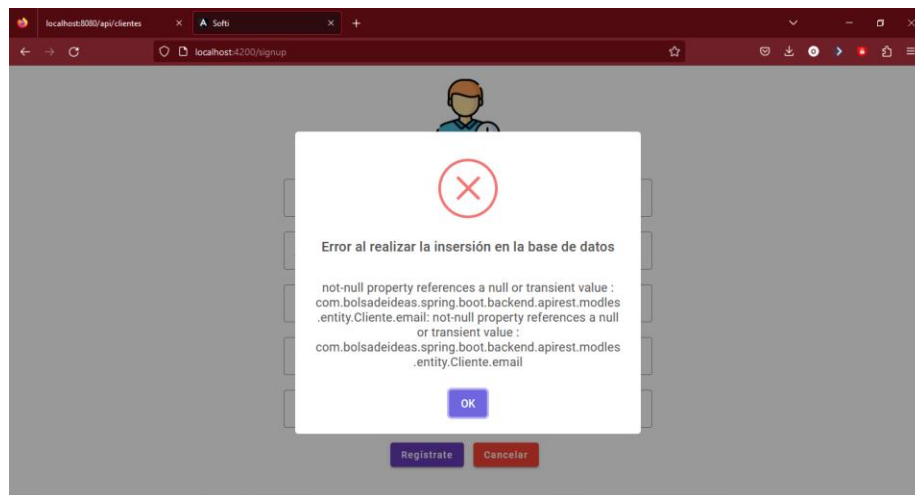


Figura 44. Intento de registro sin datos

Como podemos ver en la ilustración 33, nos arroja el mensaje de error que especificamos al codificar el método para crear el cliente.

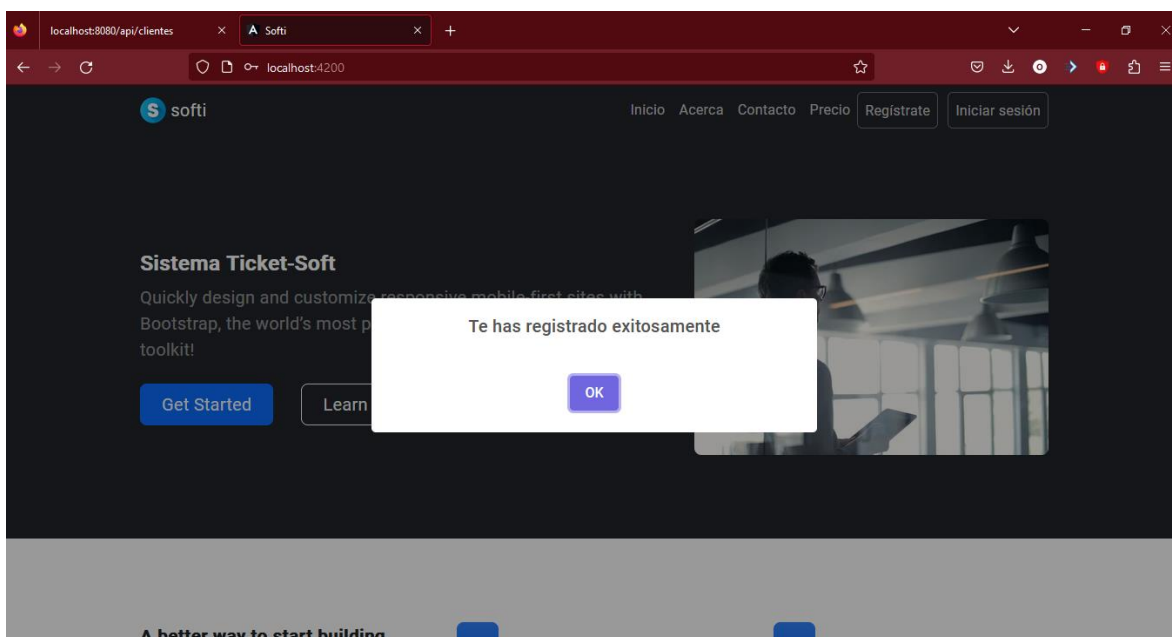


Figura 45. Inserción correcta en el sistema

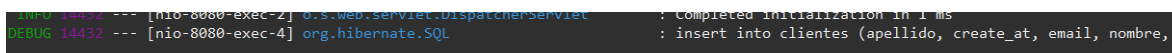


Figura 46. Inserción en la base de datos desde el servidor

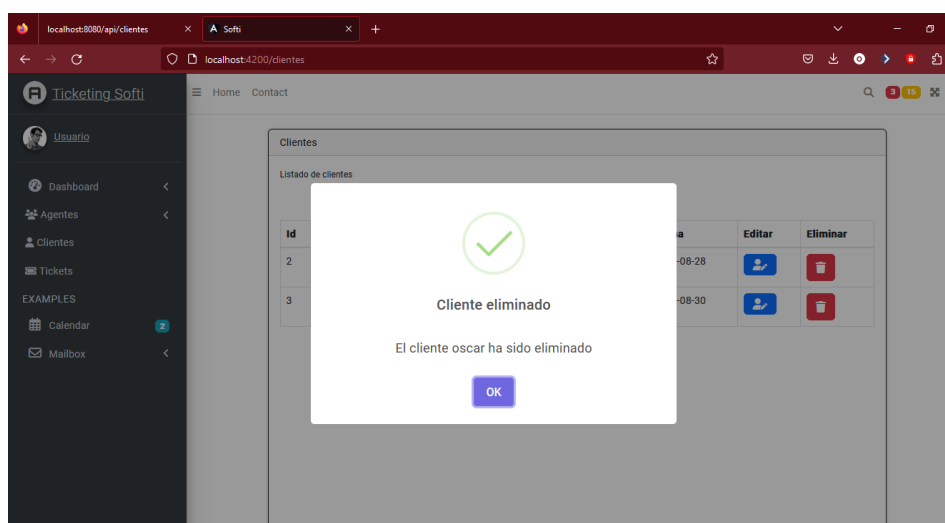


Figura 47. Eliminación de un cliente

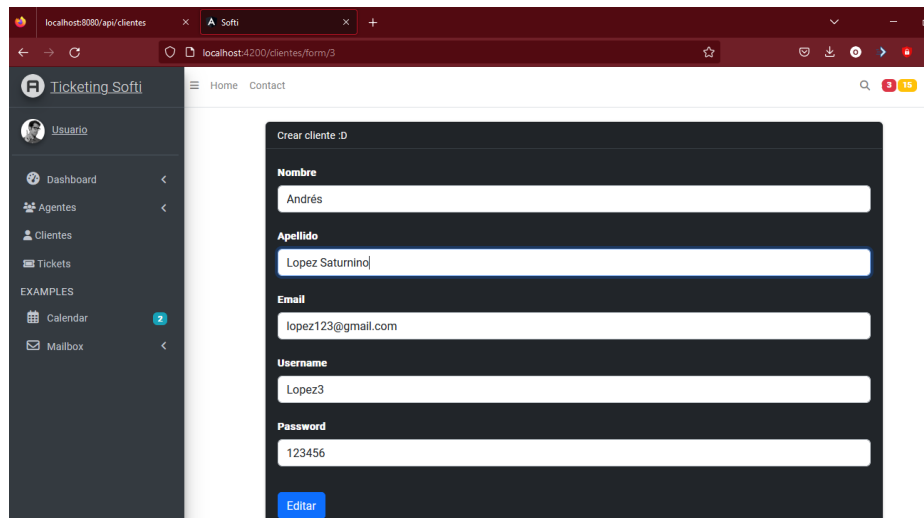


Figura 48. Edición de datos de un cliente

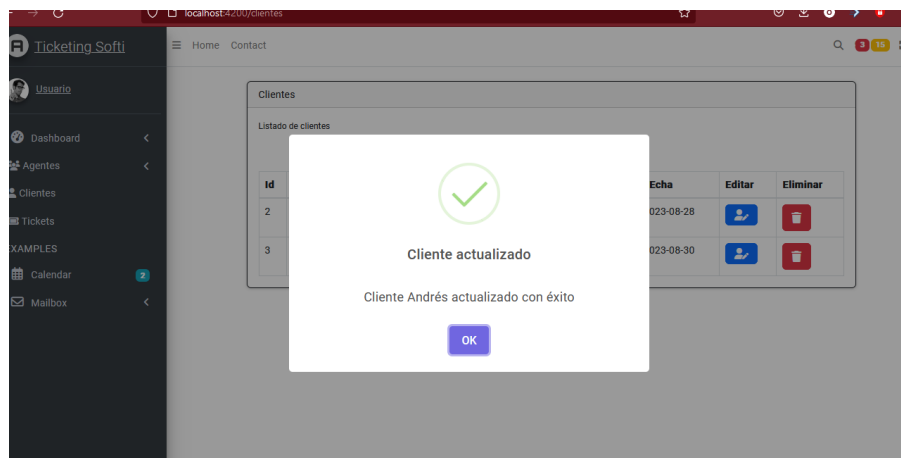


Figura 49. Cliente Actualizado

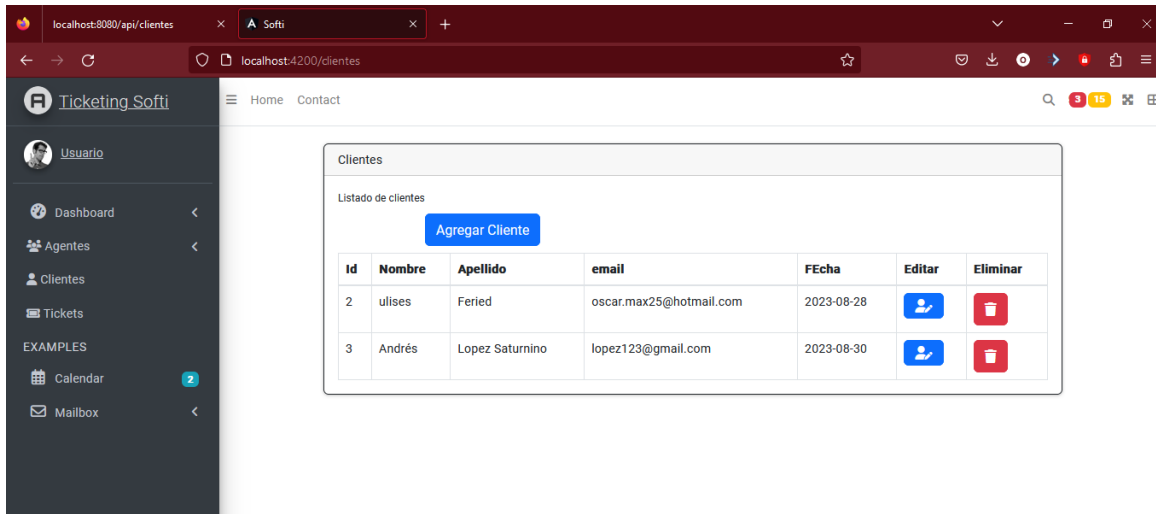


Figura 50. Actualización en el listado

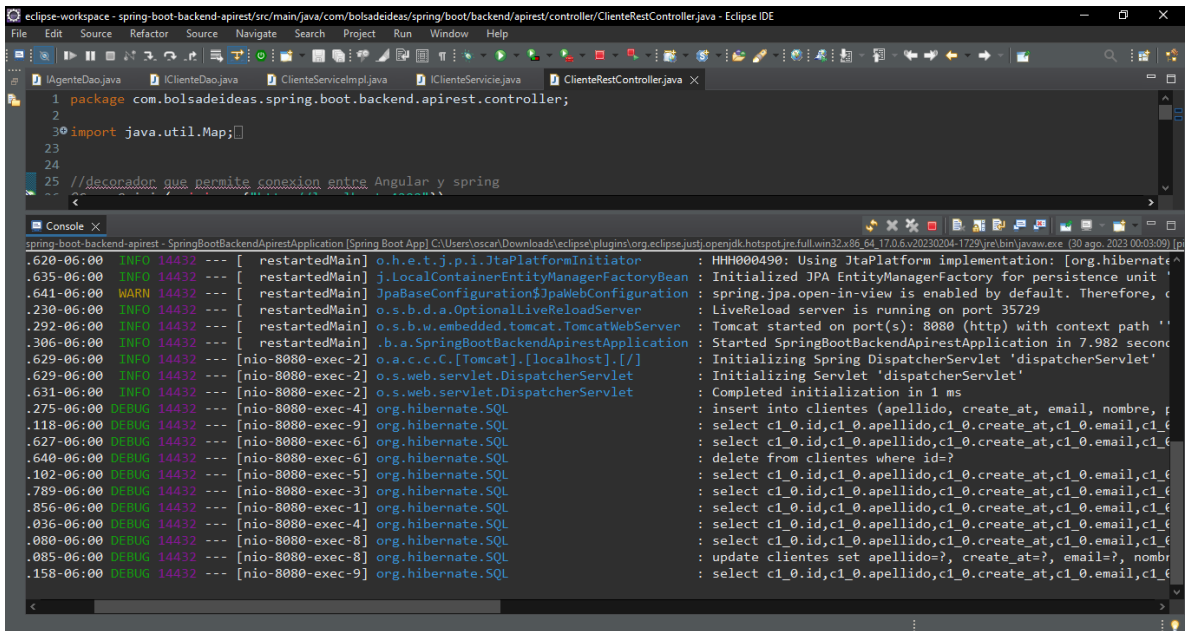


Figura 51. Peticiones realizadas al servidor

Capítulo VIII: Conclusiones

Con respecto al Software desarrollado, podemos ver la importancia de todas anotaciones que Spring boot nos proporciona para facilitar el desarrollo de cualquier sistema que se base en su arquitectura y su optimización de procesos, asimismo podemos notar mayor rapidez al realizar consultas hacia el servidor, y su adaptación a MVC (modelo vista-controlador), mejora notablemente la recepción de datos u órdenes para luego remitirlas a la vista.

Hardware, el rendimiento se puede ver afectado en computadoras de menor capacidad al ejecutar ambos servidores (front-end y back-end), por lo que se recomienda ejecutarlos por separado y especificar la ruta de donde será consumido el servidor de Spring.

Spring boot facilitó mucho el desarrollo de la primera fase, uno de los puntos relevantes dentro del desarrollo es la versión que será utilizada, ya que en versiones posteriores a la 3.1, la documentación para algunas funcionalidades en específico no son del todo claras, sin embargo, en versiones como la 2.7, podemos encontrar mayor información para la resolución de errores dentro del desarrollo, ambas versiones son muy estables, lo que hizo más cómodo el desarrollo del sistema.

Dentro del tiempo empleado en el sistema, la mayor parte fue aplicado en investigación de compatibilidad de anotaciones con la nueva versión, se presentó este problema gracias a que gran parte de la información encontrada, es para

versiones anteriores, por lo tanto, pudimos percibir un desfase dentro de los tiempos de desarrollo.

Recomiendo ampliamente el uso de Spring Boot para el desarrollo de sistemas robustos con el de “Help Desk”, la sintaxis es bastante entendible y al estar basado en Java, mejora la comodidad para su desarrollo, es importante recalcar que se debe hacer una investigación previa sobre la información que está disponible para la versión que será utilizada y no presentarse con problemas a futuro.

8.1 Competencias desarrolladas

1. Desarrollo de aplicaciones web: al trabajar con spring y angular, se puede ver mayor facilidad de desarrollo con la arquitectura cliente – servidor, ya que ambos mejoran la comunicación del backend, con el frontend, creando una aplicación completa con menores tiempos de respuesta.
2. Creación de servicios desde el backend para el consumo en el frontend.
3. Metodologías ágiles: Aprendizaje al trabajar con metodologías ágiles, a colaborar en equipos ágiles, participar en sprints, realizar seguimiento de tareas, participar en reuniones diarias y adaptarse a los cambios rápidos en los requisitos del proyecto. También adquisición de habilidades de comunicación efectiva y trabajo en equipo.
4. Gestión de servicios de TI (ITSM): Al familiarizarse con ITIL 4 y las prácticas de gestión de servicios de TI. Entender cómo se administran los servicios de TI en una organización. Esto implica comprender los conceptos de estrategia

de servicios, diseño, transición, operación y mejora continua. También se puede aprender sobre la gestión de incidentes, problemas, cambios y configuraciones, así como la importancia de los SLA y la entrega de valor al cliente.

5. Cumplimiento de estándares ISO: Al tener conocimiento sobre ISO 20000. Comprender los requisitos y las mejores prácticas para la gestión de servicios de TI y trabajar en consonancia con los estándares internacionales. Esto implica el conocimiento sobre la estructura y los procesos establecidos por la norma, así como la implementación de controles para garantizar la calidad y la mejora continua.
6. Versionado de código y control de versiones: Es importante utilizar un sistema de control de versiones, como Git, para mantener un registro de los cambios realizados en tu código. Utiliza ramas (branches) para trabajar en nuevas funcionalidades o correcciones de errores sin afectar la versión principal. Esto facilitará la colaboración y permitirá un mejor manejo de las diferentes etapas de desarrollo.

Capítulo IX: Glosario de términos

Aplicación móvil: aquella herramienta que los usuarios pueden utilizar accediendo a un dispositivo inteligente móvil a través de componentes nativos e internet.

Caso de uso: Un caso de uso es la descripción de una acción o actividad. Un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso.

Caso del negocio: Un caso de negocio es una herramienta de soporte y planeación que proyecta resultados financieros probables, así como otras consecuencias de negocio derivadas de una acción.

GitHub: (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git.

Historias de usuario: Una historia de usuario es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario.

ITIL: Las siglas ITIL significan Information Technology Infrastructure Library, que traduciríamos literalmente como Biblioteca de Infraestructura de Tecnologías de Información. ITIL es una guía de buenas prácticas para la gestión de servicios de tecnologías de la información (TI)

Login: Combinación de números y letras que sirve para identificar a un usuario dentro del sistema.

Mesa de servicio: Un Help Desk es un equipo centralizado dentro de una empresa que atiende a empleados o clientes en masa, utilizando un producto de software para organizar las conversaciones.

Metodología ágil: Aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

Modelado flujo del negocio: Diagrama de flujo de negocios es sobre los eventos de transacción, incluidas las normas y reglamentos, mientras el diagrama de flujo del sistema es satisfacer al primero para informar y sistematizar todo, o parte de al menos el diagrama de flujo.

Normas ISO: ISO (International Standardization Organization) es la organización que se encarga de la creación de normas de fabricación, comercio y comunicación que tienen un alcance internacional.

Open Source: Es un modelo de desarrollo de software basado en la colaboración abierta.

Requerimiento: un requerimiento es una exigencia que tiene un software para poder funcionar de manera correcta.

SCRUM: Scrum es un marco que permite el trabajo colaborativo entre equipos. Al igual que un equipo de rugby (de donde proviene su nombre) cuando entrena para un gran partido, scrum anima a los equipos a aprender a través de las experiencias, a autoorganizarse mientras aborda un problema y a reflexionar sobre sus victorias y derrotas para mejorar continuamente.

Servidor: Un servidor es un conjunto de computadoras capaces de atender las peticiones de un cliente y devolverle una respuesta en concordancia.

SLA: "Service Level Agreement", en español (Acuerdo de nivel de servicio o Garantía de nivel de servicio).

Spring: Se trata de un miniproyecto de no más de un mes (ciclos de ejecución muy cortos -entre una y cuatro semanas), cuyo objetivo es conseguir un incremento de valor en el producto que estamos construyendo

Ticket de servicio: Un ticket de soporte técnico es una forma de hacernos llegar por escrito una consulta o de reportar un problema relacionado con cualquiera de nuestros servicios.

Usuario: un usuario es una persona que utiliza una computadora o un servicio de red.

Capítulo X: Referencias

- Instituto Nacional de Estadística y Geografía(INEGI). (s/f). *Demografía de los Negocios (DN) 2019 - 2021*. Org.mx. Recuperado el 21 de junio de 2023, de <https://www.inegi.org.mx/programas/dn/2021/>
- Atlassian. (s/f). *Jira Service Management*. Atlassian. Recuperado el 21 de junio de 2023, de <https://www.atlassian.com/software/jira/service>
- En, R., & Hurtado -Código, D. M. (s/f). *PROPUESTA DE UN SISTEMA DE INFORMACIÓN "MESA DE AYUDA" PARA*. Edu.co. Recuperado el 21 de junio de 2023, de <https://alejandria.poligran.edu.co/bitstream/handle>

- HIXSA. (2022, octubre 7). *Help Desk y Service Desk Antecedentes históricos (Primera parte)*. HIXSA Blog | Sobre el mercado de ITSM, RPA y Gestión documental. <https://blog.hixsa.com/help-desk-y-service-desk-antecedentes-historicos-primera-parte/>
- Hurtado, A. M. Z. (1606413300000). *Evolución del Intelligent Service Desk de TI*. LinkedIn.com. <https://es.linkedin.com/pulse/evoluci%C3%B3n-del-intelligent-service-desk-de-ti-zegarra-hurtado>
- Instituto Nacional de Estadística y Geografía(INEGI). (s/f). *Demografía de los Negocios (DN) 2019 - 2021*. Org.mx. Recuperado el 21 de junio de 2023, de <https://www.inegi.org.mx/programas/dn/2021/>
- Navas Gómez, T. I., & Flórez Moreno, D. S. (2014). *Metodología para la implementación de una mesa de servicio usando buenas prácticas*. Universidad Autónoma de Bucaramanga UNAB.
- ((OTRS)) *Community Edition*. (2021, septiembre 2). OTRS; OTRS Group. <https://otrs.com/es/soluciones-de-software-otrs/otrs/otrs-community-edition/>
- Rodríguez Gallardo, J. A., López de la Madrid, M. C., & Espinoza de los Monteros Cárdenas, A. (2018). Study of the implementation of Help Desk software in an institution of higher education. *PAAKAT: Revista de Tecnología y Sociedad*, 8(14), 1–20. <https://doi.org/10.32870/pk.a8n14.298>
- *Una Revisión de la Mesa de Ayuda TI de Sociedad Portuaria Puerto Bahía: Un Paso para la Automatización*. (s/f). Edu.co. Recuperado el 21 de junio de 2023, de <https://revistascientificas.cuc.edu.co/CESTA/article/view/4270/427>
- *Zendesk: Software de servicio de atención al cliente y CRM de ventas*. (2020, octubre 15). Zendesk. <https://www.zendesk.com.mx/>
- Lloves, F. (2022, 14 febrero). *FetchType en relaciones con JPA*. <https://sospnt.com/blog/243-fetchtype-con-jpa>
- *Spring boot*. (s. f.). Spring Boot. <https://spring.io/projects/spring-boot>
- Caules, C. Á. (2023). Java override y encapsulación. *Arquitectura Java*. <https://www.arquitecturajava.com/java-override-y-encapsulacion/>
- Mangrar. (2011). Spring Framework: el patrón DAO. *Genbeta*. <https://www.genbeta.com/desarrollo/spring-framework-el-patrn-dao>
- *IBM documentation*. (s. f.). <https://www.ibm.com/docs/es/aix/7.1?topic=systems-client-server>

- Pankaj. (2022). Spring RestController. *DigitalOcean*.
<https://www.digitalocean.com/community/tutorials/spring-restcontroller>
- *PostMapping* (Spring Framework 6.0.11 API). (s. f.).
<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/PostMapping.html>
-