



Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Apizaco

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE APIZACO

**“DISEÑO Y DESARROLLO DEL FRONTEND DE UNA PLATAFORMA WEB
PARA SEGUIMIENTO A PROCESOS DE GESTIÓN DE CALIDAD EN
PRODUCTOS ALIMENTICIOS, BAJO UNA ARQUITECTURA DE SOFTWARE
BASADO EN LA NUBE.”**

TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

PARA OBTENER EL TÍTULO DE PROFESIONAL DE:

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES

PRESENTA:

EDUARDO VASQUEZ CONDE

NOMBRE DEL ASESOR:

JOSÉ JUAN HERNANDEZ MORA

APIZACO, TLAX. MAYO DE 2025



Tzompantepec, Tlaxcala, **06/marzo/2025**

OFICIO No: SC/OPV/0072/25

ASUNTO: Liberación de proyecto
para titulación integral

JORGE ISRAEL SÁNCHEZ SÁNCHEZ
JEFA DEL DEPTO. DE DIVISIÓN DE ESTUDIOS PROFESIONALES
P R E S E N T E.

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

NOMBRE DEL EGRESADO:	VASQUEZ CONDE EDUARDO
CARRERA:	INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES
NO. DE CONTROL:	19371076
NOMBRE DEL PROYECTO:	Diseño y desarrollo del frontend para una plataforma web de seguimiento a procesos de gestión de calidad de productos alimenticios, con una arquitectura basada en la nube.
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

MIQUELINA SÁNCHEZ PULIDO
JEFA DEL DEPTO. DE SISTEMAS Y COMPUTACIÓN

c.c.p. División de Estudios Profesionales
c.c.p. Depto. de Servicios Escolares
c.c.p. Archivo
MSP/mrol*

JOSÉ JUAN HERNÁNDEZ MORA
ASESOR INTERNO





Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Apizaco
División de Estudios Profesionales

Tzompantepec, Tlaxcala, **07/Abril/2025**

VASQUEZ CONDE EDUARDO.
EGRESADO DE LA CARRERA DE INGENIERÍA
EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES.
CON NÚMERO DE CONTROL 19371076.
P R E S E N T E.

Por este medio, me permito informar a usted que por aprobación de la comisión revisora asignada para valorar el trabajo que mediante la opción **TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL** y cuyo tema es: **DISEÑO Y DESARROLLO DEL FRONTEND PARA UNA PLATAFORMA WEB DE SEGUIMIENTO A PROCESOS DE GESTIÓN DE CALIDAD DE PRODUCTOS ALIMENTICIOS, CON UNA ARQUITECTURA BASADA EN LA NUBE**, 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 Recepción Profesional.

Sin otro particular por el momento me despido.

ATENTAMENTE

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



SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE APIZACO

ISRAEL JORGE SÁNCHEZ SÁNCHEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES

ccp. Archivo
ijss/Gsc*



2025
Año de
La Mujer
Indígena

Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec,
Municipio de Tzompantepec, Tlaxcala, Mex. C.P. 90491 Tel. (241) 4172010
Ext. 304, e-mail: dep_titulacion@apizaco.tecnm.mx | apizaco.tecnm.mx



Agradecimientos

Quiero expresar mi más profundo agradecimiento a Dios por darme la fuerza para llegar hasta este punto. A mis padres, Ubaldo Vasquez Vasquez y Dolores Conde Cruz, por ser mi ejemplo de perseverancia y dedicación; y a mi hermana, Laura Vasquez Conde, por su constante apoyo.

Agradezco al Instituto Tecnológico de Apizaco por brindarme la oportunidad de crecer profesionalmente, y a todos los docentes que contribuyeron a mi formación, en especial a mi asesor José Juan Hernández Mora, por su guía y apoyo durante el desarrollo de este proyecto.

Extiendo también mi gratitud al Dr. Juan Ramos Ramos por darme la oportunidad de realizar mis prácticas profesionales en la fábrica de software del Instituto, permitiéndome aplicar y ampliar mis conocimientos.

Finalmente, agradezco a mis amigos y compañeros, quienes con su apoyo incondicional han sido parte importante en la consecución de este logro.

¡Muchas gracias!

Resumen

Este proyecto de investigación se desarrolló en la fábrica de software del Instituto Tecnológico de Apizaco como parte de las residencias profesionales internas, bajo la supervisión del Dr. Juan Ramos Ramos. El objetivo principal fue analizar y mejorar la gestión de usuarios y permisos en sistemas de información utilizando Django y Python, para optimizar el acceso y manejo de reportes.

La investigación se centró en identificar las limitaciones en la estructura de permisos del sistema de información, que dificultaba la asignación de roles y el control eficiente de acceso. Como solución, se diseñó e implementó un módulo de gestión de usuarios basado en grupos jerárquicos con diferentes niveles de acceso: Químico Analista, Analista y Coordinador. Cada grupo fue definido con permisos específicos para la generación, modificación y consulta de reportes, facilitando una administración clara y segura de la información.

Adicionalmente, se integró una funcionalidad para alimentar el sistema con catálogos, mejorando la interfaz de usuario y permitiendo una organización más eficiente de la información disponible. Los resultados muestran que el nuevo sistema de gestión de usuarios y permisos mejora la accesibilidad, la coordinación y la eficiencia en el manejo de reportes y estadísticas, optimizando así los procesos.

ÍNDICE

Agradecimientos	4
Resumen.....	5
I. GENERALIDADES DEL PROYECTO	8
Introducción	8
Datos del instituto	9
Problemática	10
Objetivos	11
Objetivo general.....	11
Objetivos específicos	11
Justificación	12
II. MARCO TEÓRICO.....	13
Metodología de desarrollo ágil Sprint.....	13
¿Qué es Design Sprint?	13
Fases del Design Sprint	13
Beneficios de la Metodología en el Proyecto.....	14
Resultados Obtenidos	14
Herramientas y tecnologías.....	14
Django.....	14
Python.....	17
SQLite	21
Visual Studio Code	22
AdminLTE	23
SweetAlert	24
Git y GitHub	25
Lucidchart.....	25
Django-Jazzmin	26
Django-Widget-Tweaks.....	26
JavaScript.....	26
III. Desarrollo.....	29
Análisis de requerimientos.....	29
Diseño de sistema – Requerimientos funcionales	29
Diseño del sistema – Requerimientos no funcionales	30

.....	31
Maquetación de página web	33
Desarrollo del sistema	36
IV. RESULTADOS.....	78
V. CONCLUSIONES	84
VI. COMPETENCIAS DESARROLLADAS	85
VII. REFERENCIAS	87

I. GENERALIDADES DEL PROYECTO

Introducción

En la actualidad, la gestión de usuarios y permisos es un aspecto crítico en los sistemas de información, ya que permite organizar el acceso a la información de manera controlada, asegurando la integridad y seguridad de los datos. Este proyecto se enfoca en el desarrollo e implementación de un sistema de gestión de usuarios basado en roles, utilizando Django y Python como tecnologías principales, aprovechando su robustez, escalabilidad y capacidad para manejar requerimientos complejos.

La investigación se lleva a cabo en el contexto de la fábrica de software del Instituto Tecnológico de Apizaco, como parte de las residencias profesionales internas. El objetivo principal es optimizar el acceso y control de reportes, facilitando la administración y consulta de la información en un entorno de desarrollo seguro y eficiente. Este sistema aborda los desafíos asociados con la asignación de roles, garantizando que cada usuario tenga acceso únicamente a las funcionalidades y datos necesarios según su rol en la organización.

La estructura del proyecto fue diseñada para incorporar tres roles clave: Químico Analista, Analista y Coordinador, cada uno con diferentes niveles de acceso y permisos que abarcan la generación, consulta y modificación de reportes.

Con este enfoque, se espera proporcionar una solución tecnológica que no solo mejore la eficiencia operativa, sino que también fomente la colaboración entre los usuarios, optimice la organización de datos y facilite el cumplimiento de los estándares de seguridad en el manejo de información crítica. Este proyecto busca ser un ejemplo de cómo la tecnología puede transformar procesos internos y ofrecer valor tangible en entornos organizativos complejos.

Datos del instituto

Instituto Tecnológico de Apizaco

Es una institución de educación superior que ofrece diversas carreras a nivel licenciatura y programas de posgrado. Se destaca en áreas como ciencias sociales y administrativas.

MISIÓN:

Formar profesionistas que sean líderes visionarios, emprendedores, con capacidad de generar soluciones disruptivas, innovando en la ciencia y la tecnología, comprometidos con el desarrollo económico, responsabilidad social teniendo una visión sostenible y sustentable.

VISIÓN:

Ser una Institución de educación superior tecnológica de vanguardia, formadora de ciudadanos del mundo, con conocimientos intercultural, respetando la diversidad social.

VALORES

- **HONESTIDAD:**

Se demuestra día a día a través de la transparencia con la ejecución de las actividades laborales cotidianas.

- **RESPONSABILIDAD:**

Es la actitud que caracteriza al personal en el cumplimiento de su tarea con eficacia y eficiencia en beneficio de la comunidad.

- **LEALTAD:**

Es la actitud que caracteriza al personal del Instituto Tecnológico de Apizaco, aplica sus conocimientos y habilidades en el logro de las metas institucionales.

Problemática

La administración y control de acceso en sistemas de información son esenciales para evitar problemas de seguridad y facilitar un flujo de trabajo eficiente, especialmente en empresas que gestionan grandes volúmenes de datos y requieren reportes constantes para la toma de decisiones, como en los sectores de alimentos y bebidas. Empresas que dependen de sistemas robustos que no solo permitan la generación de reportes y gráficos con base en variables de producción, sino que también ofrezcan una administración segura y organizada a través de distintos roles de usuarios.

La falta de un sistema estructurado de gestión de permisos y acceso dificultaría en estos casos la asignación de roles específicos y el control de acceso a datos críticos. Esto genera ineficiencias en la administración de la información y puede exponer al sistema a riesgos de acceso no autorizado, afectando tanto la seguridad de los datos como la productividad de los usuarios.

Para resolver esta problemática, es necesario diseñar e implementar un módulo de gestión de usuarios que clasifique a los usuarios en grupos con diferentes niveles de permisos. De esta forma, cada usuario accede únicamente a las funciones y datos correspondientes a su rol. Este enfoque facilita la generación y consulta de reportes, proporcionando una estructura más segura y eficiente para el manejo de datos y el acceso a estadísticas clave.

Objetivos

Objetivo general

Desarrollar un submódulo de gestión de usuarios y permisos que optimice el control de acceso dentro de un sistema de generación de reportes basado en registros de análisis de laboratorio, asegurando una administración segura y organizada de la información según los roles específicos de cada usuario.

Objetivos específicos

- Implementar un sistema de gestión de usuarios basado en roles jerárquicos que permita clasificar a los usuarios en grupos con permisos diferenciados (Químico Analista, Analista y Coordinador), asegurando que cada usuario acceda únicamente a las funcionalidades correspondientes a su rol.
- Desarrollar un sistema de autenticación mediante login que garantice el acceso controlado y seguro al sistema, validando las credenciales de los usuarios y restringiendo el acceso a usuarios no autorizados.
- Diseñar e integrar un módulo para la asignación y actualización de permisos que facilite a los administradores definir y modificar los niveles de acceso a las funciones del sistema de manera dinámica y eficiente.
- Crear una funcionalidad para la consulta y actualización de catálogos que permita a los usuarios acceder a información actualizada en tiempo real, optimizando la generación de reportes y la organización de datos.
- Optimizar la experiencia del usuario mediante el uso de herramientas de diseño moderno, proporcionando una interfaz intuitiva, visualmente atractiva y funcional.
- Asegurar un entorno de desarrollo controlado mediante el uso de entornos virtuales y control de versiones con Git y GitHub, permitiendo un manejo eficiente del código y las dependencias del proyecto.

Justificación

La gestión de usuarios y permisos es una necesidad primordial en sistemas de información que manejan datos sensibles y requieren accesos diferenciados según los roles. Este proyecto responde a esa necesidad, proponiendo una solución tecnológica que mejora significativamente la organización, seguridad y eficiencia en el manejo de información y generación de reportes.

El desarrollo de un submódulo de gestión de usuarios basado en roles jerárquicos, con funcionalidades específicas como la generación de reportes y el acceso a catálogos dinámicos, asegura que los datos estén disponibles únicamente para los usuarios autorizados. Este enfoque no solo optimiza los procesos internos, sino que también minimiza riesgos asociados a accesos no controlados, incrementando la confiabilidad del sistema.

Además, el uso de tecnologías como Django Framework para el backend y JavaScript en el frontend permitió desarrollar un sistema robusto, dinámico y orientado a las necesidades del usuario. JavaScript fue esencial para implementar funcionalidades interactivas en la interfaz, optimizando la experiencia del usuario mediante formularios dinámicos y alertas personalizadas. La integración de librerías como SweetAlert mejoró la comunicación visual en el sistema, mientras que la plantilla AdminLTE facilitó la creación de un diseño coherente y profesional. Por otro lado, el uso de entornos virtuales y el control de versiones con Git y GitHub garantizó un desarrollo estructurado, colaborativo y fácilmente replicable, asegurando la escalabilidad y mantenibilidad del sistema.

Este proyecto no solo atiende la problemática de seguridad y control en sistemas de información, sino que también contribuye al desarrollo profesional, permitiendo aplicar y reforzar conocimientos adquiridos durante la carrera. Asimismo, destaca por su impacto práctico al proporcionar una herramienta eficiente y alineada con las necesidades reales del entorno laboral, cumpliendo con los objetivos de las residencias profesionales y fortaleciendo el vínculo entre teoría y práctica.

II. MARCO TEÓRICO

Metodología de desarrollo ágil Sprint

El desarrollo de software ha evolucionado hacia enfoques más dinámicos y flexibles, como las metodologías ágiles, que permiten a los equipos adaptarse rápidamente a los cambios y enfocarse en la entrega continua de valor. En este proyecto, la metodología ágil seleccionada fue Design Sprint, la cual se destaca por su capacidad de proporcionar soluciones rápidas y probadas en un corto periodo de tiempo.

¿Qué es Design Sprint?

Design Sprint es una metodología desarrollada por Google Ventures que se centra en resolver problemas complejos mediante el diseño, el prototipado y la validación con usuarios finales en tan solo cinco días. Se utiliza ampliamente para proyectos de software debido a su enfoque estructurado y enfocado en el usuario.

Fases del Design Sprint

El ciclo de Design Sprint se divide en cinco fases principales:

1. Comprender: Durante esta etapa, se analizó el problema a resolver en el sistema de reportes, identificando las necesidades clave de los usuarios y los objetivos del proyecto.
2. Idear: Se llevaron a cabo sesiones de lluvia de ideas y bocetado de posibles soluciones, buscando innovaciones que se alinearan con los requisitos funcionales y las expectativas del cliente.
3. Decidir: A partir de las ideas generadas, se seleccionaron las soluciones más viables mediante votaciones y análisis de impacto, asegurando que las decisiones estuvieran basadas en criterios técnicos y funcionales.
4. Prototipar: Se creó un prototipo funcional del sistema, destacando las funciones principales, como la generación de reportes automatizados.

5. Validar: Se realizaron pruebas con usuarios finales para evaluar la efectividad y usabilidad del prototipo, obteniendo retroalimentación que permitió refinar las soluciones implementadas.

Beneficios de la Metodología en el Proyecto

- Flexibilidad: La metodología permitió adaptarse a los cambios en los requerimientos y prioridades del proyecto.
- Colaboración: Facilitó la participación de los interesados, asegurando que las soluciones desarrolladas estuvieran alineadas con sus expectativas.
- Rapidez en la entrega: Al enfocarse en ciclos cortos de desarrollo, se lograron avances significativos en poco tiempo, reduciendo los riesgos asociados a proyectos más largos y tradicionales.

Resultados Obtenidos

Gracias al uso de Design Sprint, se consiguió un diseño funcional y validado del sistema de reportes en un tiempo eficiente, sentando las bases para el desarrollo e implementación del producto final. Este enfoque permitió minimizar los errores y garantizar que las funcionalidades diseñadas estuvieran centradas en las necesidades del usuario final.

Herramientas y tecnologías

En este proyecto, se emplearon diversas herramientas y tecnologías para el desarrollo e implementación de funcionalidades relacionadas con la gestión de usuarios y catálogos. Estas incluyen:

Django

Framework de desarrollo web en Python, utilizado para la creación del backend del sistema. Su sistema de autenticación y permisos integrado permitió una implementación eficiente de roles y control de acceso para la gestión de usuarios.

Arquitectura Modelo-Vista-Controlador (MVC):

Django sigue el patrón arquitectónico MVC, donde:

Modelo (Model): Gestiona la lógica de acceso y manipulación de datos, conectándose directamente a la base de datos mediante su ORM (Object-Relational Mapping).

Vista (View): Procesa las solicitudes del usuario y devuelve respuestas en forma de plantillas renderizadas.

Controlador (Controller): En Django, este rol lo cumplen las vistas y URLs, que actúan como intermediarios entre el modelo y las plantillas.

Sistema de Autenticación y Permisos:

Django proporciona un sistema de autenticación robusto, incluyendo la gestión de usuarios, grupos y permisos, lo cual permite implementar un control de acceso basado en roles con facilidad. Estas funcionalidades resultaron fundamentales para desarrollar el sistema de gestión de usuarios y permisos en este proyecto.

Django escalable

En comparación con el tiempo de desarrollo, el hardware es barato, por lo que Django lo es. Diseñado para aprovechar la mayor cantidad de hardware que pueda lanzarle.

Django utiliza una arquitectura de "nada compartido", lo que significa que puedes añadir hardware a cualquier nivel: servidores de bases de datos, servidores de almacenamiento en caché o servidores web/de aplicaciones.

El marco separa limpiamente los componentes, como su capa de base de datos y capa de aplicación. Y se envía con un marco de caché simple pero potente.¹

Seguridad en Django

¹ FAQ: General. (s/f). Django Project. Recuperado el 9 de mayo de 2024, de <https://docs.djangoproject.com/en/5.1/faq/general/>

Protección contra la falsificación de solicitudes entre sitios (CSRF)

Los ataques CSRF permiten a un usuario malintencionado ejecutar acciones utilizando las credenciales de otro usuario sin el conocimiento o consentimiento de ese usuario.

Django tiene protección incorporada contra la mayoría de los tipos de ataques CSRF, lo que le proporciona lo han habilitado y utilizado en su caso. Sin embargo, al igual que con cualquier técnica de mitigación, tiene limitaciones. Por ejemplo, es posible deshabilitar el módulo CSRF globalmente o para vistas particulares. Solo debes hacer esto si sabes lo que estás haciendo. Existen otras limitaciones si tu sitio tiene subdominios que están fuera de tu control.

La protección CSRF funciona mediante la comprobación de un secreto en cada Solicitud POST. Esto garantiza que un usuario malintencionado no pueda "reproducir" un formulario POST en su sitio web y hacer que otro usuario registrado envíe ese formulario sin saberlo. El usuario malintencionado tendría que conocer el secreto, que es específico del usuario (usando una galleta).

Cuando se implementa con HTTPS, comprobará que el encabezado de referencia HTTP esté establecido en un URL en el mismo origen (incluyendo subdominio y puerto). Debido a que HTTPS proporciona seguridad adicional, es imperativo garantizar que las conexiones usen HTTPS donde esté disponible mediante el reenvío de solicitudes de conexión inseguras y el uso de HSTS para navegadores compatibles. `CsrfViewMiddleware`

Protección contra inyecciones SQL

La inyección SQL es un tipo de ataque en el que un usuario malintencionado es capaz de ejecutar código SQL arbitrario en una base de datos. Esto puede dar lugar a registros o fuga de datos.

Los conjuntos de consultas de Django están protegidos contra la inyección SQL, ya que sus consultas son Construido mediante la parametrización de consultas. Se define el código SQL de una consulta independientemente de los parámetros de la

consulta. Dado que los parámetros pueden ser proporcionados por el usuario y, por lo tanto, no son seguros, el controlador de base de datos subyacente los escapa.

Django también les da a los desarrolladores el poder de escribir consultas sin procesar o ejecutar sql personalizado. Estas capacidades deben usarse con moderación y siempre debe tener cuidado de Escape correctamente cualquier parámetro que el usuario pueda controlar.²

Seguridad de la sesión

De manera similar a las limitaciones de CSRF que requieren que un sitio implementarse de manera que los usuarios que no son de confianza no tengan acceso a ningún subdominio, también tiene limitaciones.

Python

Definición y Origen:

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, creado en 1991 por Guido van Rossum. Diseñado con un enfoque en la simplicidad y la legibilidad, Python ha evolucionado para convertirse en uno de los lenguajes más populares y versátiles del mundo. Su sintaxis limpia y su estructura lógica facilitan el aprendizaje y la implementación, lo que lo hace ideal tanto para principiantes como para desarrolladores avanzados.

Características Clave:

Legibilidad y Claridad:

Python se basa en una sintaxis minimalista que prioriza la legibilidad del código. Esto facilita la colaboración en proyectos grandes, ya que los desarrolladores pueden comprender rápidamente el trabajo de otros.

² Security in Django. (s/f). Django Project. Recuperado el 9 de mayo de 2024, de <https://docs.djangoproject.com/en/5.1/topics/security/>

Versatilidad:

- Python se utiliza en una amplia variedad de áreas, incluyendo:
- Desarrollo web (frameworks como Django y Flask).
- Ciencia de datos y análisis (bibliotecas como Pandas, NumPy y Matplotlib).
- Automatización de tareas (librerías como Selenium y PyAutoGUI).
- Inteligencia artificial y aprendizaje automático (bibliotecas como TensorFlow y scikit-learn).

Ecosistema de Librerías:

- Python cuenta con una extensa colección de bibliotecas y frameworks que extienden su funcionalidad. Estas herramientas permiten a los desarrolladores centrarse en la resolución de problemas sin necesidad de reinventar soluciones.
- Portabilidad:

Python es multiplataforma, lo que significa que el código escrito en un sistema operativo puede ejecutarse en otro con mínimas modificaciones.

Comunidad Activa:

Una comunidad global activa y en crecimiento garantiza soporte continuo, documentación amplia y contribuciones regulares al desarrollo del lenguaje.

Ventajas de Python en el Desarrollo Backend

En el contexto de sistemas basados en la web, Python se ha consolidado como una herramienta poderosa gracias a:

Frameworks Especializados:

Django, uno de los frameworks más robustos de Python, facilita el desarrollo de aplicaciones web seguras y escalables al proporcionar herramientas integradas para la gestión de bases de datos, autenticación y administración de usuarios.

Integración con Bases de Datos:

El ORM (Object-Relational Mapping) de Django, escrito en Python, permite interactuar con bases de datos relacionales de manera eficiente y segura.

Manejo de API REST:

Python simplifica la creación de API RESTful mediante extensiones como Django Rest Framework (DRF), que agilizan la conexión entre el backend y el frontend.

Python en el Proyecto

En este proyecto, Python fue la base del backend y jugó un papel clave en la implementación de funcionalidades relacionadas con la gestión de usuarios, roles y catálogos:

Gestión de Usuarios y Roles:

Se utilizaron las capacidades nativas de Django en combinación con Python para implementar un sistema jerárquico de roles (Químico Analista, Analista y Coordinador).

La flexibilidad de Python permitió personalizar vistas y permisos para cada rol, asegurando que los usuarios interactuaran únicamente con los recursos necesarios.

Administración de Catálogos:

Python facilitó la creación de catálogos dinámicos, permitiendo la consulta y actualización de datos según los permisos asignados al usuario.

La integración con plantillas y vistas personalizadas en Django permitió una experiencia de usuario fluida.

Validaciones y Seguridad:

Se implementaron validaciones en los modelos de Django escritos en Python para asegurar la integridad de los datos. Por ejemplo:

Validaciones personalizadas para garantizar que los campos obligatorios fueran consistentes con las reglas del negocio.

Python también fue clave en la implementación de medidas de seguridad como la gestión de sesiones y la protección contra accesos no autorizados.

Impacto de Python en el Proyecto

El uso de Python aportó beneficios significativos al desarrollo del sistema:

Eficiencia en el Desarrollo:

La claridad y simplicidad del lenguaje redujeron los tiempos de desarrollo, permitiendo centrarse en la solución de problemas en lugar de en la sintaxis.

Escalabilidad:

Gracias a la estructura modular de Python, el sistema puede adaptarse fácilmente para incluir nuevos roles, catálogos o funcionalidades en el futuro.

Compatibilidad:

Python permitió integrar de manera fluida otras tecnologías, como AdminLTE y SweetAlert, optimizando la interacción entre el backend y el frontend.³

³ Welcome to. (s/f). Python.org. Recuperado el 10 de mayo de 2024, de <https://www.python.org/>

SQLite

SQLite es una base de datos relacional ligera y embebida, diseñada para ser sencilla y eficiente. Al no requerir un servidor independiente para su funcionamiento, SQLite se distingue como una opción ideal para entornos de desarrollo y proyectos con recursos limitados. Desde su creación en el año 2000 por D. Richard Hipp, ha ganado popularidad por su facilidad de uso y portabilidad.

Características Principales

- **Ligereza y Portabilidad:**

Almacena toda la información en un único archivo, lo que simplifica su gestión y transporte entre sistemas.

- **Simplicidad en la Configuración:**

No requiere instalación compleja ni configuración adicional, siendo accesible desde el inicio del desarrollo.

- **Compatibilidad con Estándares SQL:**

Admite comandos estándar SQL, lo que facilita su adopción por desarrolladores familiarizados con bases de datos relacionales.

- **Integración Multiplataforma:**

Funciona en múltiples lenguajes de programación, como Python, y frameworks como Django.

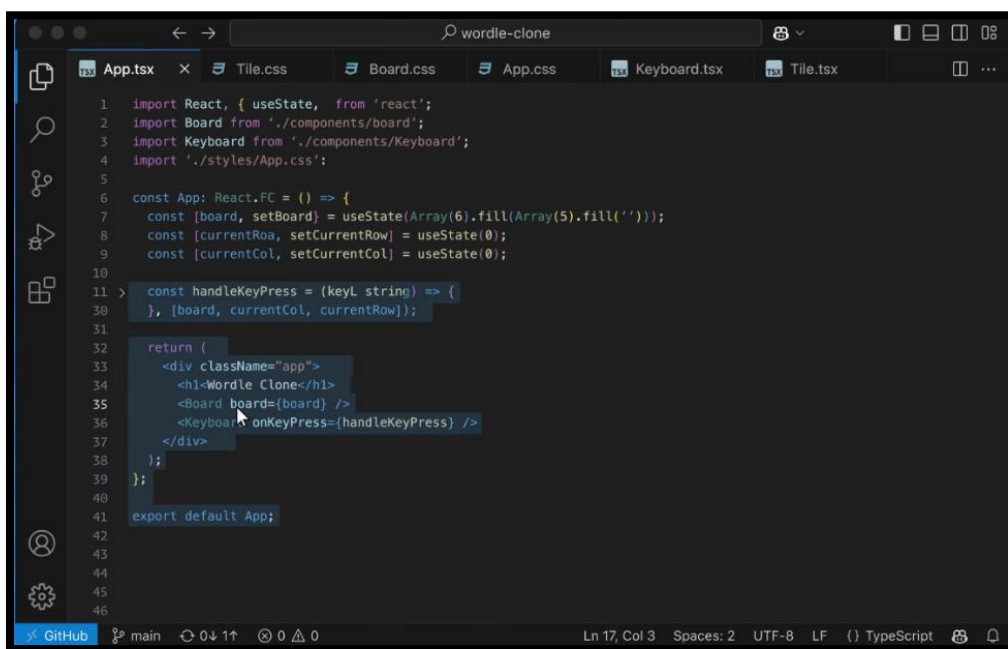
Ventajas en el Desarrollo

Rápida Implementación: SQLite es la base de datos predeterminada en Django, lo que permite un inicio inmediato del desarrollo sin necesidad de configuraciones adicionales.

Pruebas y Prototipado: Su simplicidad la hace ideal para etapas iniciales, permitiendo pruebas ágiles y ajustes rápidos antes de implementar en entornos más complejos.

Visual Studio Code

Es un entorno de desarrollo integrado (IDE) ampliamente utilizado debido a su robustez, flexibilidad y capacidad de integración con herramientas y extensiones clave. Su soporte para entornos virtuales permite gestionar dependencias de manera organizada y aislada, lo que asegura la estabilidad y portabilidad en diferentes entornos de desarrollo. Además, su terminal integrada facilita la ejecución de comandos esenciales, como la instalación de paquetes y la administración de frameworks como Django. Extensiones específicas para lenguajes como Python y tecnologías como Django mejoran significativamente la productividad al ofrecer autocompletado, resaltado de sintaxis y sugerencias inteligentes. También, su integración nativa con sistemas de control de versiones como Git y plataformas como GitHub permite un manejo eficiente y organizado del código fuente, lo que lo convierte en una herramienta esencial para el desarrollo moderno.



```
1 import React, { useState, from 'react';
2 import Board from './components/board';
3 import Keyboard from './components/Keyboard';
4 import './styles/App.css';
5
6 const App: React.FC = () => {
7   const [board, setBoard] = useState(Array(6).fill(Array(5).fill('')));
8   const [currentRow, setCurrentRow] = useState(0);
9   const [currentCol, setCurrentCol] = useState(0);
10
11   const handleKeyPress = (key: string) => {
12     // ...
13   };
14
15   return (
16     <div className="app">
17       <h1>Wordle Clone</h1>
18       <Board board={board} />
19       <Keyboard onKeyPress={handleKeyPress} />
20     </div>
21   );
22 };
23
24 export default App;
```

Ilustración 1. Visual Studio Code

AdminLTE

Plantilla comercial implementada para construir la interfaz gráfica del sistema. Proporcionó componentes predefinidos como modales y formularios, necesarios para la interacción con los catálogos y la administración de usuarios.

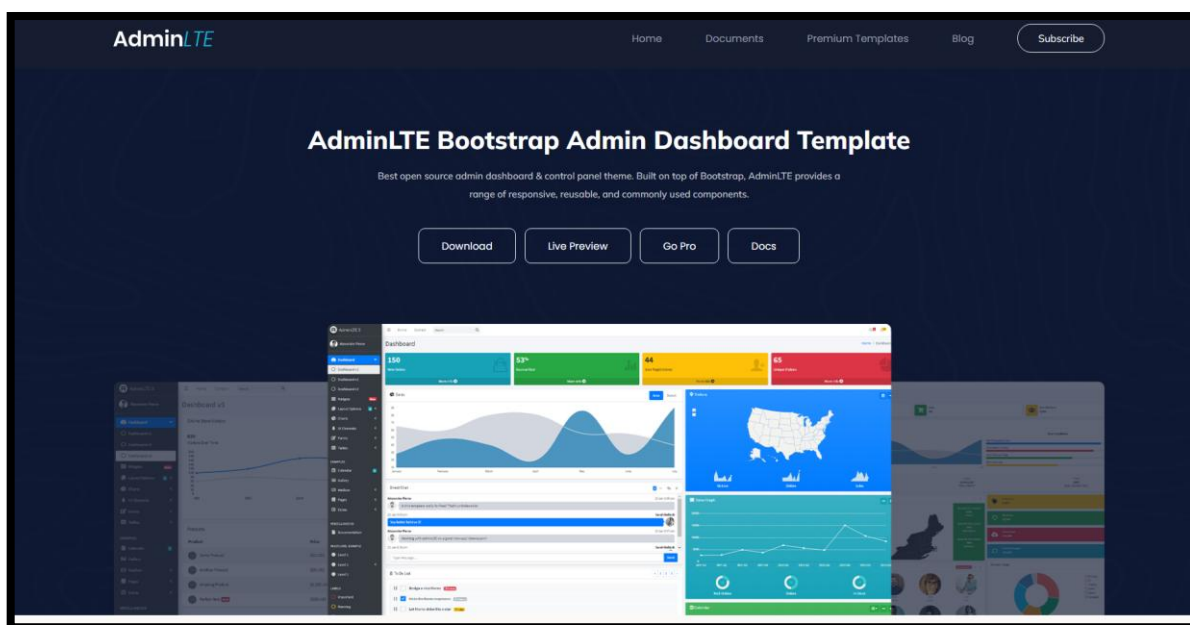


Ilustración 2. AdminLTE

AdminLTE es una plantilla de administración basada en Bootstrap, ampliamente utilizada para el desarrollo de interfaces gráficas en aplicaciones web. Diseñada para ser moderna, responsiva y altamente personalizable, AdminLTE ofrece una solución integral para construir sistemas de administración visualmente atractivos y funcionales. Su integración sencilla con frameworks como Django la convierte en una herramienta ideal para proyectos orientados al manejo de datos y usuarios.

Características Principales

1. Componentes Predefinidos:

- AdminLTE incluye una amplia gama de elementos listos para usar, como:
 - Modales: Ventanas emergentes para mostrar información o formularios interactivos.
 - Formularios dinámicos: Diseñados para capturar y validar datos de manera eficiente.
 - Tablas: Personalizables para mostrar grandes volúmenes de información.

2. Compatibilidad con Bootstrap:

- Al estar basado en Bootstrap, AdminLTE es completamente responsivo, garantizando que las interfaces se adapten correctamente a diferentes dispositivos.

3. Interactividad Mejorada:

- Integración con JavaScript para ofrecer una experiencia fluida al usuario, incluyendo navegación intuitiva y elementos interactivos.

4. Estilo Moderno y Personalizable:

- Proporciona un diseño atractivo y profesional, con opciones de personalización que permiten adaptar los colores, iconos y elementos visuales a las necesidades del proyecto.

SweetAlert

Biblioteca utilizada para mostrar alertas y confirmaciones de acciones dentro del sistema, contribuyendo a una experiencia de usuario más fluida y moderna.

Git y GitHub

Herramientas utilizadas para el control de versiones y la colaboración en el proyecto, lo que permitió un seguimiento detallado de los cambios realizados en el código y una organización estructurada de las funcionalidades desarrolladas.

Lucidchart

Herramienta empleada para diseñar diagramas y representar gráficamente la arquitectura del sistema, los flujos de trabajo de gestión de usuarios y las relaciones de catálogos. Su uso permitió una documentación clara y profesional.

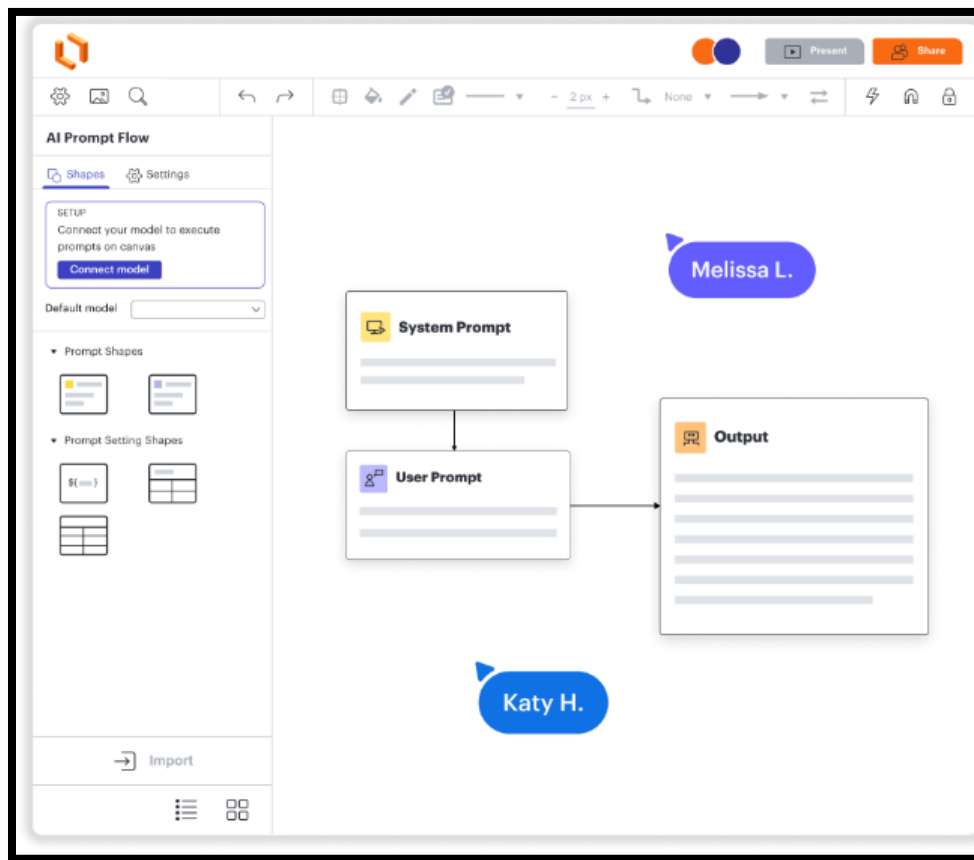


Ilustración 3. Diagramas con lucidchart

Django-Jazzmin

Esta librería se utilizó para personalizar y optimizar el módulo administrativo de Django, permitiendo un diseño más atractivo y funcional. Esto aceleró significativamente el proceso de desarrollo al proporcionar una interfaz moderna y lista para usar.

Django-Widget-Tweaks

Facilitó la personalización de formularios en las vistas, lo que mejoró la experiencia del usuario al permitir una integración más flexible con el diseño del sistema.

JavaScript

JavaScript es un lenguaje de programación interpretado, orientado a objetos y diseñado para hacer que las páginas web sean interactivas. Desde su creación en 1995 por Brendan Eich, se ha convertido en una pieza fundamental del desarrollo web, formando parte del estándar de tecnologías esenciales junto con HTML y CSS.

Características Clave de JavaScript

1. Interactividad:

- Permite manipular el DOM (Document Object Model) para actualizar dinámicamente el contenido de las páginas web sin necesidad de recargarlas.

2. Versatilidad:

- Se puede ejecutar tanto en el lado del cliente como en el servidor (por ejemplo, con Node.js), lo que amplía sus aplicaciones.

3. Compatibilidad:

- Funciona en todos los navegadores modernos y es ampliamente soportado por frameworks y bibliotecas.

4. Extensibilidad:

- Ofrece integración con diversas librerías y frameworks como React, Vue.js, jQuery y herramientas como SweetAlert.

Ventajas de JavaScript en Aplicaciones Web

1. Actualización Dinámica de Contenido:

- La capacidad de modificar elementos del DOM en tiempo real mejora la experiencia del usuario al permitir interfaces más interactivas.

2. Validaciones del Lado del Cliente:

- Reduce la carga en el servidor al realizar validaciones rápidas directamente en el navegador.

3. Personalización de Interacciones:

- Gracias a librerías como SweetAlert, se pueden implementar alertas y confirmaciones visualmente atractivas y funcionales.

4. Compatibilidad con Herramientas Visuales:

- JavaScript es compatible con plantillas como AdminLTE, mejorando la integración de elementos como modales y menús.

Uso de JavaScript en el Proyecto

En este proyecto, JavaScript fue clave para implementar funcionalidades que mejoraron la interacción entre el usuario y el sistema. Entre los principales usos destacan:

1. Formularios Dinámicos:

- Validaciones inmediatas al capturar datos, asegurando que los valores ingresados cumplan con los requisitos establecidos.

- Modificación y actualización de campos en tiempo real, según las selecciones del usuario.

2. Modales Interactivos:

- En combinación con AdminLTE, se utilizaron modales para la edición y consulta de datos en los catálogos, mejorando la fluidez en las operaciones sin recargar la página.

3. Confirmaciones Visuales con SweetAlert:

- Se integró esta librería para proporcionar confirmaciones estilizadas y notificaciones de éxito o error, lo que mejoró la experiencia del usuario.

4. Personalización de la Experiencia de Usuario:

- Se implementaron scripts para controlar la visibilidad de elementos y funciones, adaptándolos al rol del usuario autenticado.

Impacto en la Experiencia del Usuario

1. Interacción Fluida:

- Las funcionalidades dinámicas redujeron la cantidad de recargas y simplificaron tareas como la gestión de usuarios y la edición de catálogos.

2. Mejoras Visuales:

- Las alertas y notificaciones enriquecieron la interfaz, haciéndola más atractiva y profesional.

3. Reducción de Errores:

- Las validaciones en tiempo real minimizaron errores antes de que los datos fueran enviados al servidor.

III. Desarrollo

Análisis de requerimientos

El análisis de requerimientos se llevó a cabo para identificar las necesidades específicas del sistema, enfocándose en la gestión de usuarios y catálogos. Este proceso permitió determinar las funcionalidades necesarias y las restricciones del sistema, asegurando que la solución cumpliera con las expectativas y necesidades de los usuarios finales. Entre los principales requerimientos identificados se encuentran:

La necesidad de un sistema de gestión de usuarios basado en roles, que permita clasificar a los usuarios y asignar permisos específicos. La implementación de un sistema de login que garantice un acceso seguro y controlado.

La creación de vistas dinámicas y personalizadas para los usuarios, dependiendo de sus roles asignados. La gestión eficiente de catálogos, permitiendo consultas y actualizaciones de datos en tiempo real.

Diseño de sistema – Requerimientos funcionales

Los requerimientos funcionales definen las características y funciones específicas que el sistema debe cumplir. En este proyecto, los requerimientos funcionales incluyen:

1. Gestión de Usuarios y Roles:

- Crear, modificar y eliminar usuarios.
- Asignar roles específicos (Administrador, Analista, Usuario) con permisos personalizados.
- Controlar el acceso a funcionalidades del sistema en función del rol asignado.

2. Autenticación y Autorización:

- Implementar un sistema de login que valide las credenciales de los usuarios.

- Redirigir a vistas específicas según el rol del usuario después del inicio de sesión.
- Asegurar que usuarios no autenticados no puedan acceder a funcionalidades restringidas.

3. **Gestión de Catálogos:**

- Permitir la consulta de catálogos dinámicos adaptados a los permisos del usuario.
- Ofrecer opciones de edición, adición y eliminación de registros en los catálogos mediante modales.

4. **Interfaz de Usuario:**

- Diseñar una interfaz intuitiva utilizando AdminLTE, con componentes como formularios, alertas y modales.
- Implementar SweetAlert para mensajes de confirmación y alertas visuales.

Diseño del sistema – Requerimientos no funcionales

Los requerimientos no funcionales describen las cualidades del sistema relacionadas con su rendimiento, escalabilidad y seguridad. En este proyecto, se identificaron los siguientes requerimientos no funcionales:

- **Escalabilidad:**
Garantizar que el sistema permita la incorporación de nuevos roles y funcionalidades sin requerir modificaciones significativas en la arquitectura.
- **Seguridad:**
Proteger el acceso al sistema mediante autenticación segura y validación de permisos.
Garantizar la integridad de los datos en las operaciones de consulta y actualización de catálogos.
- **Rendimiento:**
Asegurar tiempos de respuesta rápidos para la consulta y actualización de catálogos, incluso con múltiples usuarios simultáneos.

- **Usabilidad:**
Diseñar una interfaz clara y amigable para los usuarios, minimizando la curva de aprendizaje.
Facilitar la navegación mediante una organización lógica de las funcionalidades.
- **Mantenibilidad:**
Utilizar estándares y buenas prácticas de codificación para facilitar el mantenimiento y la evolución del sistema.
Implementar un control de versiones mediante Git y GitHub para gestionar cambios de manera eficiente.

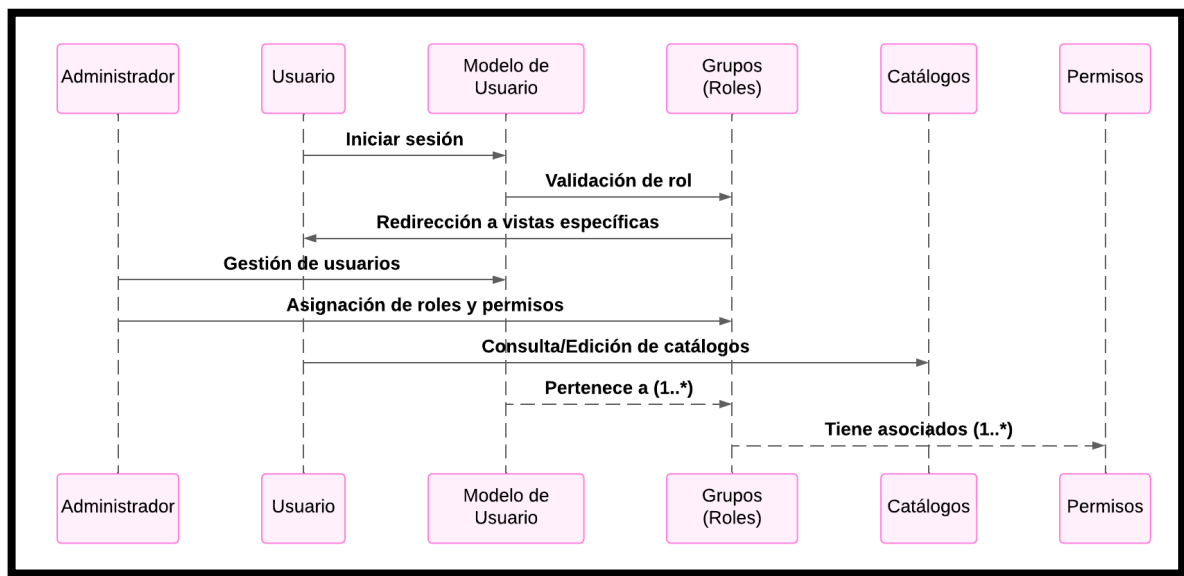


Ilustración 4 Diagrama de relacion de usuarios y permisos

Como se aprecia en la ilustración 4, se esquematiza la relación entre los usuarios y sus respectivos permisos.

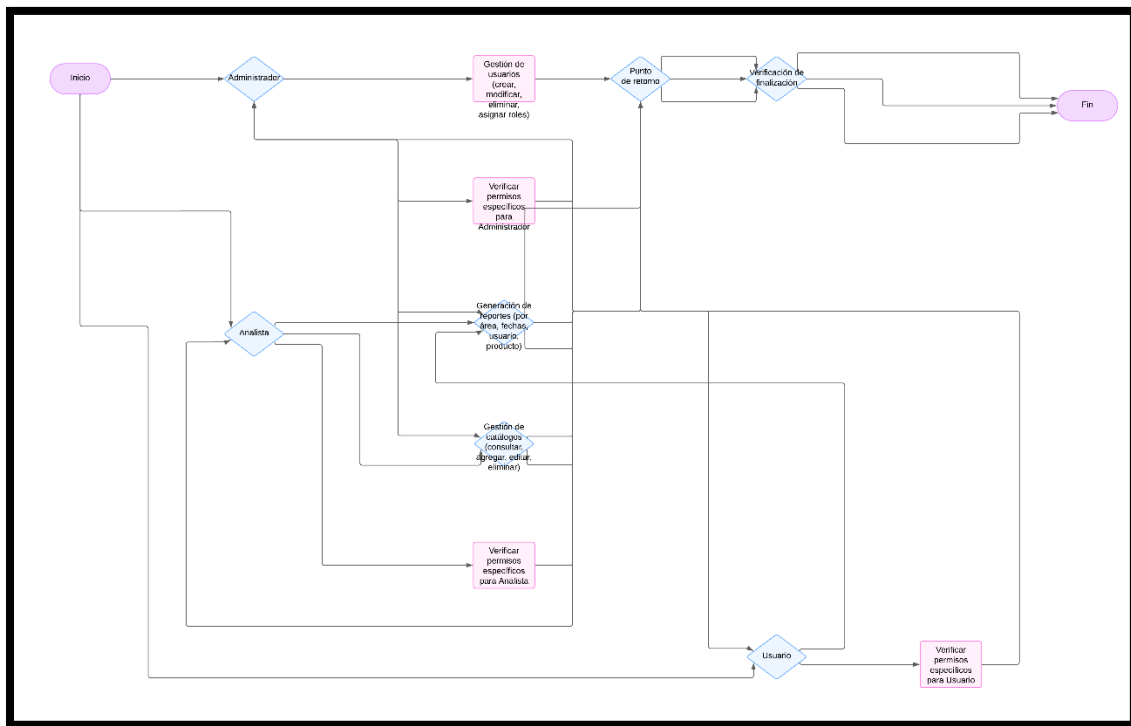


Ilustración 5. Diagrama de casillas

Diagrama de Casillas

Descripción: El diagrama de casillas refleja cómo los distintos tipos de usuarios interactúan con las operaciones principales del sistema. En este esquema, se detalla la relación entre los roles establecidos (Administrador, Analista y Usuario) y las funcionalidades específicas a las que tienen acceso. El administrador tiene un control total sobre la gestión de usuarios, catálogos y generación de reportes, mientras que el analista y el usuario tienen permisos más limitados, enfocados en consultar y generar información.

Maquetación de página web

A continuación, en las ilustraciones 6 a 10, se presentan las principales vistas de la plataforma web desarrollada, las cuales muestran desde la página de inicio de sesión hasta la creación de grupos de usuarios y el registro de información empresarial. Estas pantallas reflejan la estructura visual y funcional del sistema construido, priorizando una experiencia de usuario clara, moderna e intuitiva.

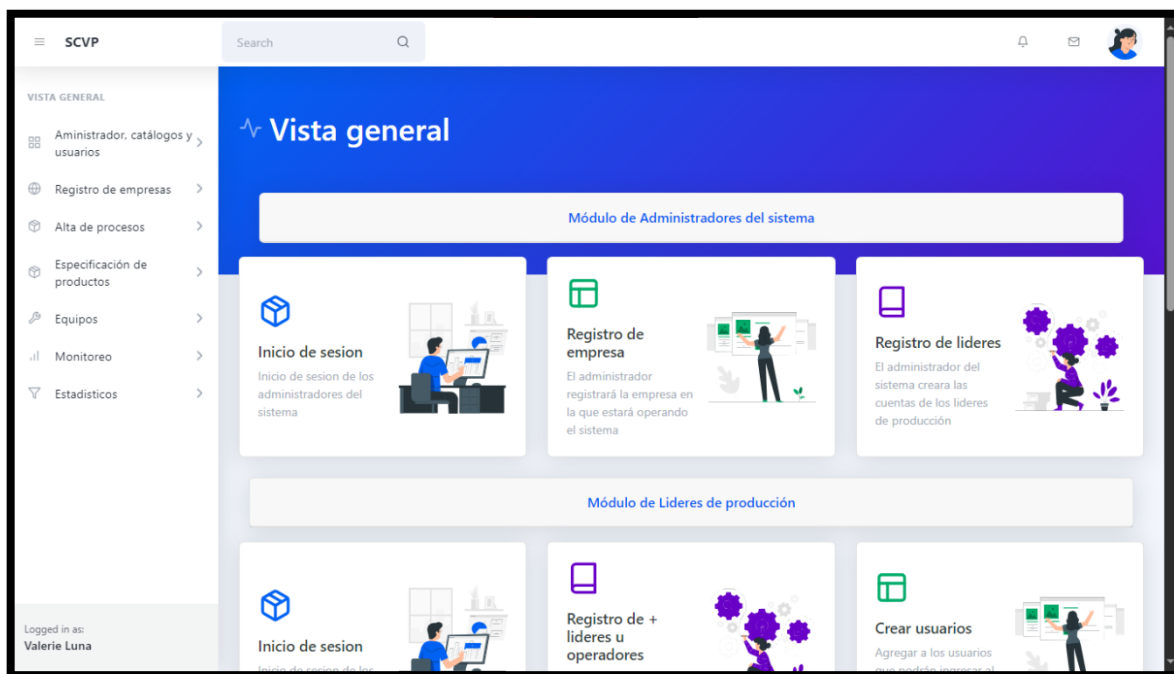


Ilustración 6. Página principal del sistema

Como se aprecia en la ilustración 6, se muestra la página principal del sistema, diseñada para brindar acceso directo a las principales funcionalidades mediante una interfaz ordenada y responsiva, facilitando la navegación de los usuarios.

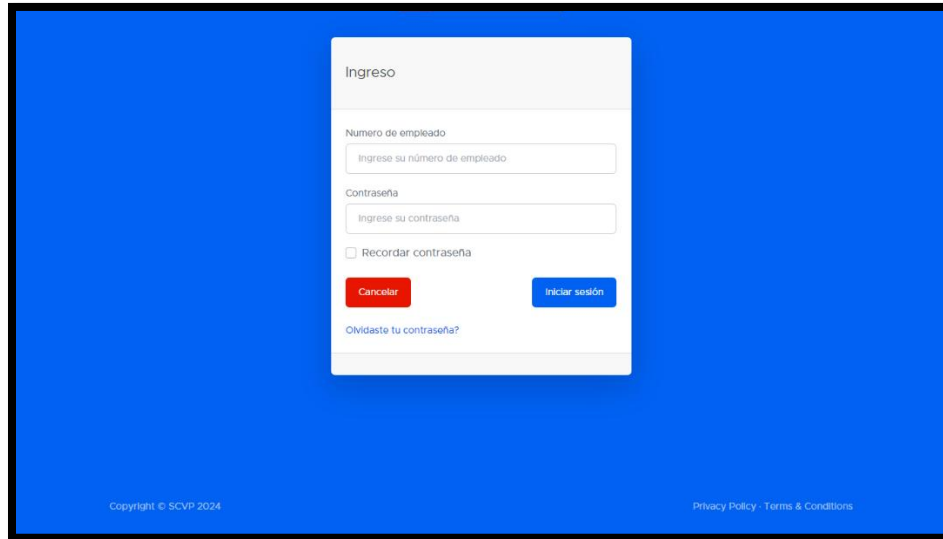


Ilustración 7. Vista de ingreso al sistema con credenciales

En la ilustración 7 se observa la vista de ingreso al sistema, donde los usuarios deben autenticar su acceso mediante sus credenciales (numero de empleado y contraseña), asegurando así un control adecuado y personalizado para cada tipo de usuario.

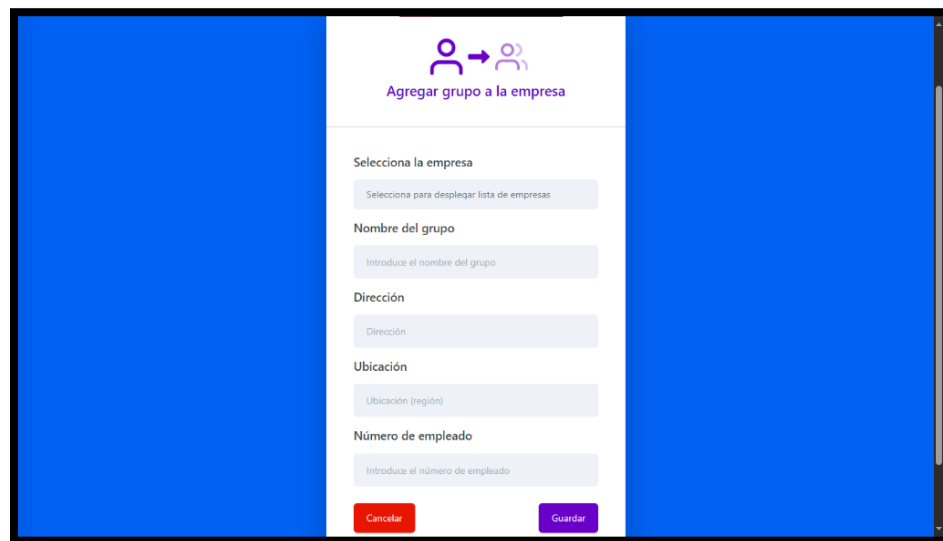


Ilustración 8. Vista de creación de grupos

La ilustración 8 muestra la sección destinada a la creación de grupos de usuarios. Desde esta vista, los administradores pueden definir nuevos grupos, asignar permisos específicos y estructurar la jerarquía de acceso dentro del sistema.

Ilustración 9. Registro de información de empresa

Como se visualiza en la ilustración 9, se dispone de una vista para el registro de información de empresas, la cual permite almacenar y gestionar los datos relevantes asociados a las entidades participantes, optimizando su consulta y organización.

Ilustración 10. Vista de registro de usuario

Finalmente, la ilustración 10 presenta la vista de registro de nuevos usuarios. Esta sección permite la creación de cuentas de usuario asignándoles número de empleado, garantizando así una administración eficiente y segura de la plataforma.

Desarrollo del sistema

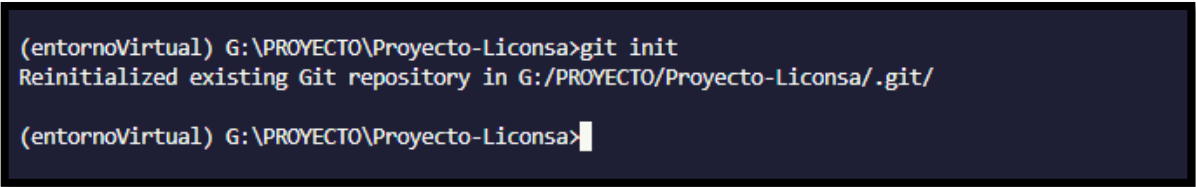
Uso de Git y GitHub

Como parte de las herramientas utilizadas para el desarrollo de este proyecto, se implementó Git como sistema de control de versiones, permitiendo un seguimiento detallado de los cambios realizados en el código a lo largo de todas las etapas. Asimismo, se utilizó GitHub como plataforma para alojar el repositorio remoto, facilitando la colaboración, la organización y la gestión del código fuente.

Inicialización del Repositorio Local

En primera instancia, se creó un repositorio local en el directorio del proyecto mediante el comando:

git init

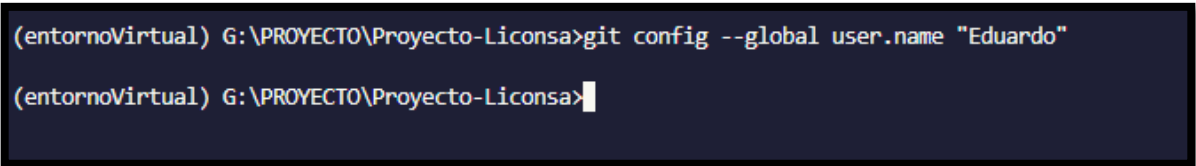


```
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>git init
Reinitialized existing Git repository in G:/PROYECTO/Proyecto-Liconsas/.git/

(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>
```

Ilustración 11. Git init

Esto permitió configurar un entorno de control de versiones directamente en el proyecto. Posteriormente, se realizaron configuraciones básicas, como el registro del nombre y correo electrónico del desarrollador, asegurando la trazabilidad de los commits realizados.



```
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>git config --global user.name "Eduardo"

(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>
```

Ilustración 12. Comando asignación nombre

```
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>git config --global user.email "calle13.201@gmail.com"
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>
```

Ilustración 13. Comando asignación correo

Gestión de Archivos y Cambios

Una vez inicializado el repositorio, los archivos del proyecto fueron añadidos al área de preparación utilizando:

`git add .`

```
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>git add .
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>
```

Ilustración 14. Comando git add.

Esto garantizó que todos los archivos relevantes fueran incluidos en los commits. Los cambios se registraron en el historial del repositorio mediante el comando:

`git commit -m "Mensaje descriptivo sobre los cambios realizados"`

Cada commit se utilizó para documentar de manera clara y precisa las modificaciones realizadas durante el desarrollo, lo que facilitó la revisión y auditoría del código.

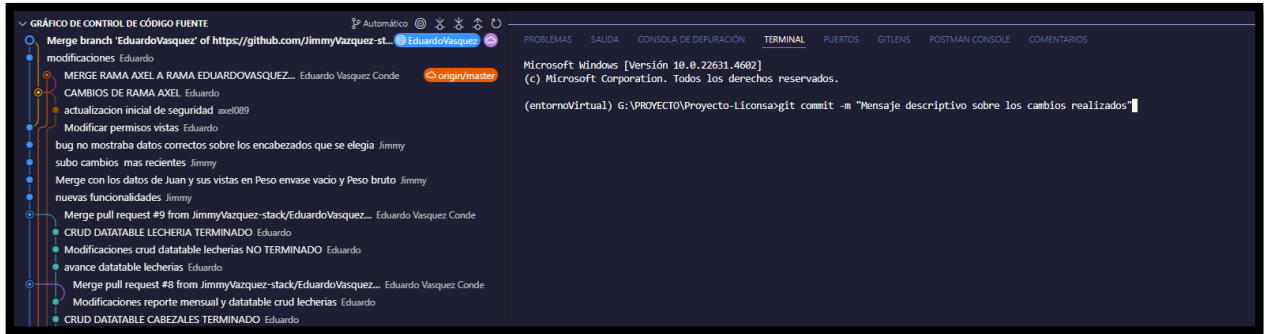


Ilustración 15. Historial de commits

Conexión con GitHub

Se creó un repositorio remoto en GitHub para alojar y respaldar el código del proyecto. Posteriormente, el repositorio local se vinculó con el remoto utilizando:

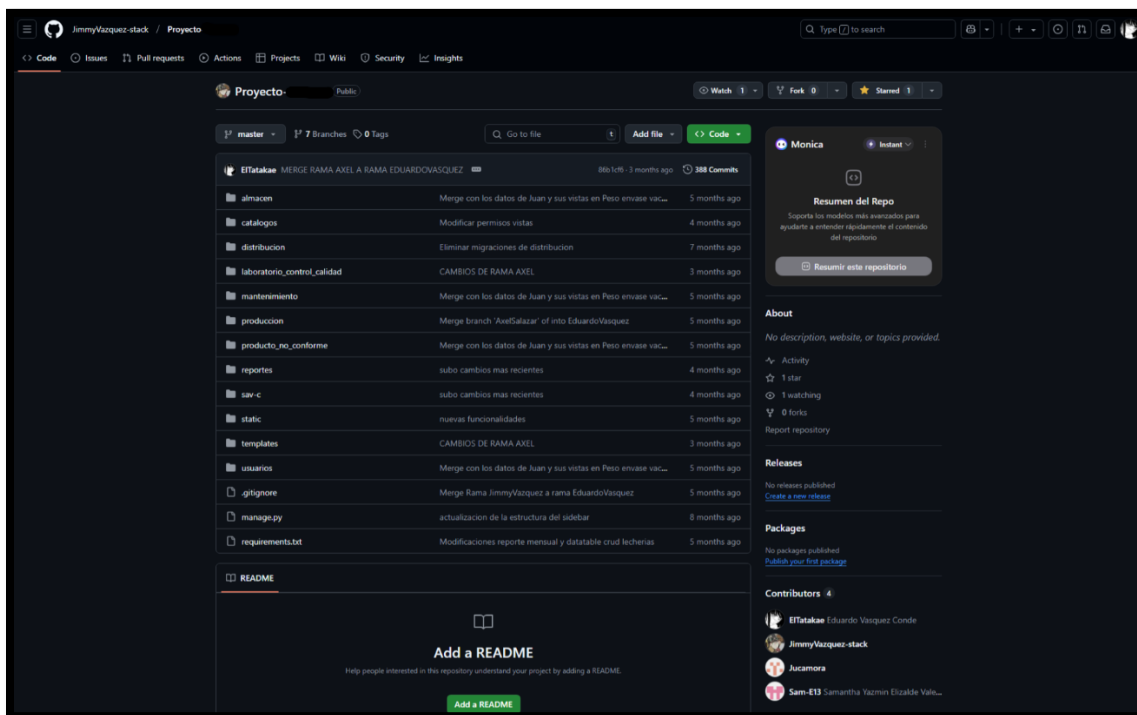


Ilustración 16. Repositorio en github

git remote add origin <https://github.com/JimmyVazquez-stack/Proyecto-Liconsagit>

```
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsa>git remote add origin https://github.com/JimmyVazquez-stack/Proyecto-Liconsa.git
```

Esto permitió sincronizar el código local con GitHub y mantener una copia segura y accesible del proyecto.

Subida y Sincronización de Cambios

Durante el desarrollo, se subieron los cambios al repositorio remoto para garantizar que la última versión del proyecto estuviera disponible. Esto se logró mediante:

```
git push -u origin master
```

Además, para garantizar la actualización constante del repositorio local, se utilizó el comando:

```
git pull origin master
```

De esta manera, el repositorio local y remoto permanecieron sincronizados en todo momento.

Ramas y Colaboración

Para facilitar la organización del código y aislar diferentes funcionalidades en desarrollo, se utilizaron ramas específicas. Estas ramas se crearon con:

```
git checkout -b nombre_rama
```

```
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsa>git checkout -b EduardoVasquez
```

Ilustración 17. Comando git crear nueva rama

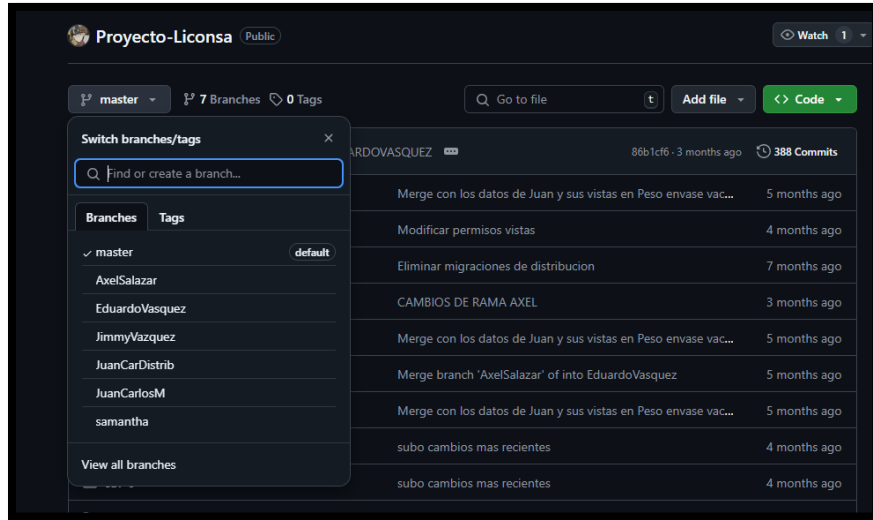


Ilustración 18. Ramas creadas para el repositorio

El uso de ramas permitió trabajar en nuevas características o correcciones de errores sin afectar directamente el desarrollo principal del proyecto.

Instalación de Python

Para el desarrollo de este sistema, se utilizó Python 3.11, el cual se descargó desde su sitio oficial como se aprecia en la ilustración 19 y 20 (<https://www.python.org>). La instalación incluyó la configuración de variables de entorno para garantizar el acceso al intérprete desde la línea de comandos.⁴

Version	Operating System	Description	MD5 Sum	File Size	GPG	Signature
Gzipped source tarball	Source release		c5f771ea256dc5bdb0897eeb4d35bb0	25.1 MB	SIG	signature
XZ compressed source tarball	Source release		fe92acfa0db969f5044958edba451d463	18.9 MB	SIG	signature
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	98fa94815780c9330fc2154559365834	40.6 MB	SIG	CRT SIG
Windows installer (64-bit)	Windows	Recommended	4fe11b2b0bb0c744cf4aff5377cd7f	24.0 MB	SIG	CRT SIG
Windows installer (32-bit)	Windows		e369a267acaa62487223bd835279bb9	22.9 MB	SIG	CRT SIG
Windows installer (ARM64)	Windows	Experimental	18e5bd9a4854109adf3b77c7c9dc1ded	23.2 MB	SIG	CRT SIG
Windows embeddable package (64-bit)	Windows		7df0f4244e5a66760b7caaed58e86c93	10.1 MB	SIG	CRT SIG
Windows embeddable package (32-bit)	Windows		0888959642cc8af087d88da3866490a5	9.1 MB	SIG	CRT SIG
Windows embeddable package (ARM64)	Windows		e3dbbd5d63c6cb203adc60c8ca5f5f7	9.3 MB	SIG	CRT SIG

Ilustración 19. Selección de archivo de la versión python 3.11

⁴ Python Release Python 3.11.0. (s/f). Python.org. Recuperado el 10 de abril de 2024, de <https://www.python.org/downloads/release/python-3110/>

Proceso de instalación

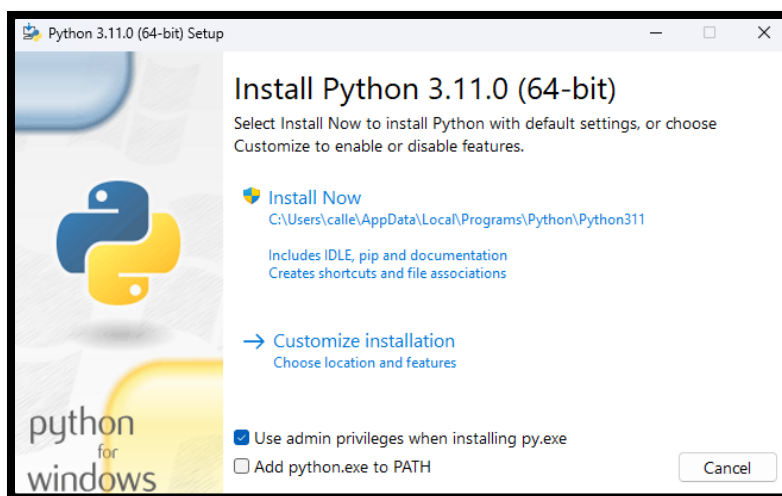


Ilustración 20. Ventana de instalación Python

Configuración de SQLite

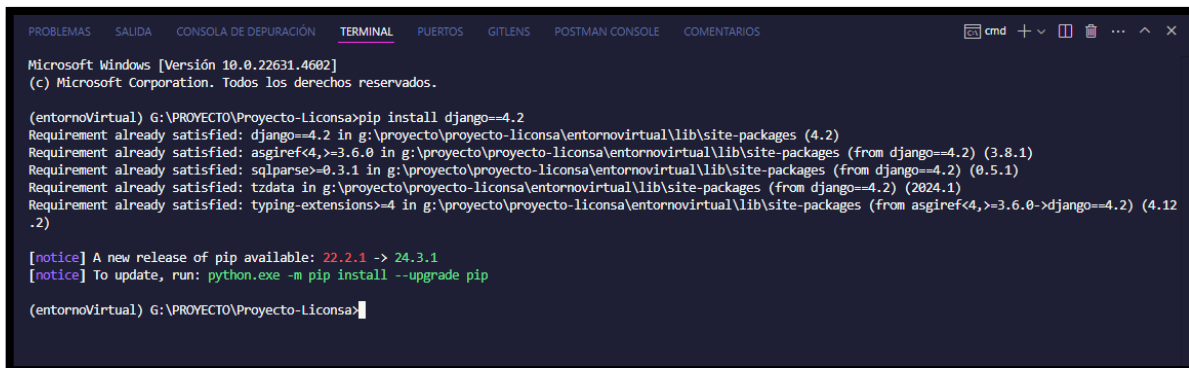
SQLite fue utilizada como la base de datos principal durante la etapa de desarrollo por su ligereza y simplicidad. Django la configuró automáticamente en el archivo settings.py, asegurando su integración sin pasos adicionales como se visualiza en la ilustración 21.



Ilustración 21. Configuración para uso de SQLite

Instalación de Django

Django 4.2 fue instalado en un entorno virtual creado con el módulo venv de Python. Esto aseguró que las dependencias del proyecto estuvieran aisladas del sistema operativo.



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  GITLENS  POSTMAN CONSOLE  COMENTARIOS
Microsoft Windows [Versión 10.0.22631.4602]
(c) Microsoft Corporation. Todos los derechos reservados.

(entornoVirtual) G:\PROYECTO\Proyecto-Licons>pip install django==4.2
Requirement already satisfied: django==4.2 in g:\proyecto\proyecto-licons\entornovirtual\lib\site-packages (4.2)
Requirement already satisfied: asgiref<4,>=3.6.0 in g:\proyecto\proyecto-licons\entornovirtual\lib\site-packages (from django==4.2) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in g:\proyecto\proyecto-licons\entornovirtual\lib\site-packages (from django==4.2) (0.5.1)
Requirement already satisfied: tzdata in g:\proyecto\proyecto-licons\entornovirtual\lib\site-packages (from django==4.2) (2024.1)
Requirement already satisfied: typing-extensions>=4 in g:\proyecto\proyecto-licons\entornovirtual\lib\site-packages (from asgiref<4,>=3.6.0->django==4.2) (4.12.2)

[notice] A new release of pip available: 22.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(entornoVirtual) G:\PROYECTO\Proyecto-Licons>
```

Ilustración 22. Instalación en terminal de la versión de Django 4.2

Creación del proyecto con Django

```
(entornoVirtual) G:\PROYECTO\Proyecto-Licons>django-admin startproject sav-c
```

Este comando generó la estructura básica del proyecto, incluyendo archivos clave como settings.py, urls.py y manage.py, los cuales permiten la configuración global, la gestión de rutas y la administración del proyecto, respectivamente.

Creación de Aplicaciones

Para organizar las funcionalidades, se creó una aplicación independiente enfocada en la gestión de catálogos mediante:

```
(entornoVirtual) G:\PROYECTO\Proyecto-Licons>python manage.py startapp catalogos
```

Esta aplicación se registró en el archivo settings.py bajo la sección INSTALLED_APPS, asegurando que Django reconociera sus componentes.



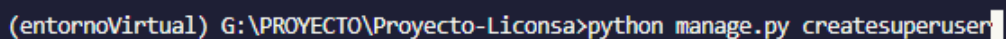
```
settings.py

INSTALLED_APPS = [
    'jazzmin',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'widget_tweaks',
    'rest_framework',

    #Mis aplicaciones
    'laboratorio_control_calidad',
    'almacen',
    'produccion',
    'mantenimiento',
    'producto_no_conforme',
    'catalogos',
    'reportes',
    'usuarios',
]
```

Ilustración 23. Registro de apps en settings.py

Adicionalmente, se creó un superusuario (admin) con el comando:



```
(entornoVirtual) G:\PROYECTO\Proyecto-Liconsas>python manage.py createsuperuser
```

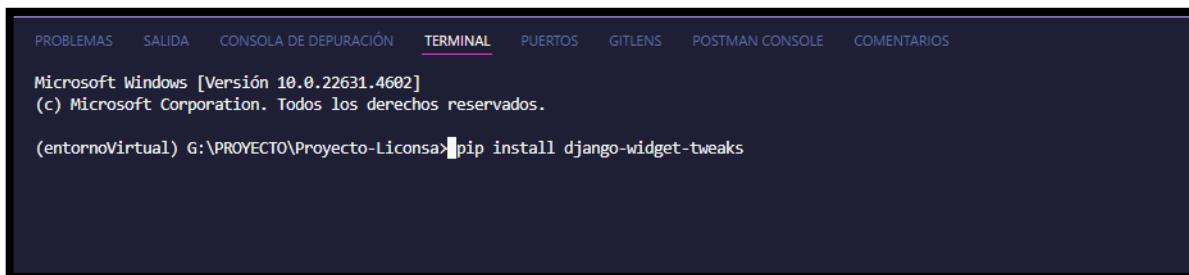
Esto permitió acceder al panel de administración en la dirección:

<http://127.0.0.1:8000/admin/>

Instalación y configuración del entorno de desarrollo

Se utilizó Visual Studio Code como IDE principal, configurando un entorno virtual con venv para gestionar las dependencias del proyecto. Django fue instalado mediante pip, junto con bibliotecas adicionales como django-widget-tweaks, que permitió personalizar formularios y optimizar su presentación en las plantillas.

Además, se utilizó Django Jazzmin, una biblioteca que facilita la personalización del módulo administrativo de Django, proporcionando una interfaz moderna, funcional y fácil de usar. Esto permitió que el módulo administrativo fuera más accesible y estéticamente agradable, acelerando la gestión de usuarios, catálogos y permisos durante el desarrollo.

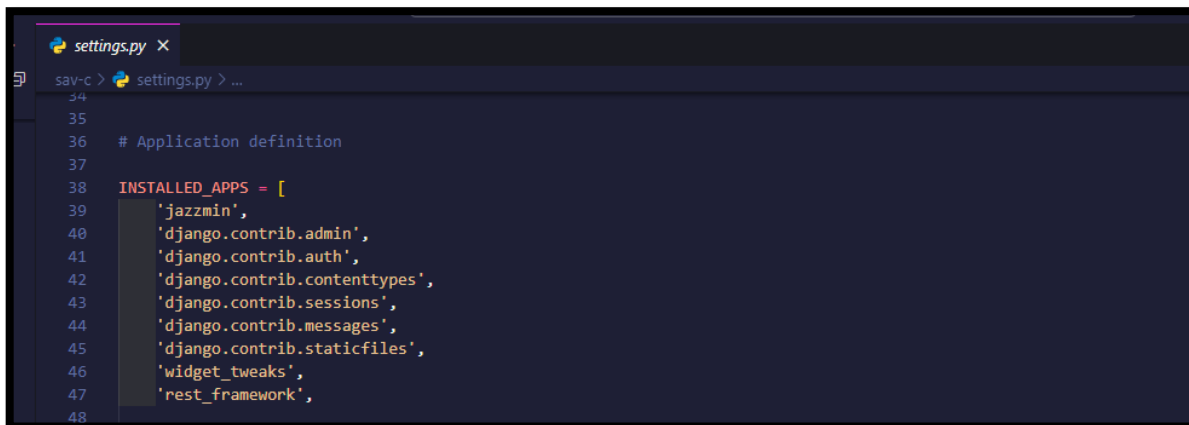


```
Microsoft Windows [Versión 10.0.22631.4602]
(c) Microsoft Corporation. Todos los derechos reservados.

(entornoVirtual) G:\PROYECTO\Proyecto-Liconsal>pip install django-widget-tweaks
```

Ilustración 24. Instalación widget-tweaks

Para habilitar widget_tweaks en el proyecto, fue agregarlo a INSTALLED_APPS en el archivo settings.py del proyecto:



```
settings.py
sav-c > settings.py > ...
34
35
36 # Application definition
37
38 INSTALLED_APPS = [
39     'jazzmin',
40     'django.contrib.admin',
41     'django.contrib.auth',
42     'django.contrib.contenttypes',
43     'django.contrib.sessions',
44     'django.contrib.messages',
45     'django.contrib.staticfiles',
46     'widget_tweaks',
47     'rest_framework',
48
```

Ilustración 25. Widget- tweaks añadido a settings.py

```
(entornoVirtual) G:\PROYECTO\Proyecto-Licons>pip install -U django-jazzmin
Requirement already satisfied: django-jazzmin in g:\projecto\proyecto-licons\entornovirtual\lib\site-packages (3.0.0)
Collecting django-jazzmin
  Downloading django_jazzmin-3.0.1-py3-none-any.whl (2.1 MB)
    2.1/2.1 MB 13.5 MB/s eta 0:00:00
Requirement already satisfied: django>=4.2 in g:\projecto\proyecto-licons\entornovirtual\lib\site-packages (from django-jazzmin) (4.2)
Requirement already satisfied: sqlparse>=0.3.1 in g:\projecto\proyecto-licons\entornovirtual\lib\site-packages (from django>=4.2->django-jazzmin) (0.5.1)
Requirement already satisfied: asgiref<4,>=3.6.0 in g:\projecto\proyecto-licons\entornovirtual\lib\site-packages (from django>=4.2->django-jazzmin) (3.8.1)
Requirement already satisfied: tzdata in g:\projecto\proyecto-licons\entornovirtual\lib\site-packages (from django>=4.2->django-jazzmin) (2024.1)
Requirement already satisfied: typing-extensions>=4 in g:\projecto\proyecto-licons\entornovirtual\lib\site-packages (from asgiref<4,>=3.6.0->django>=4.2->django-jazzmin) (4.12.2)
Installing collected packages: django-jazzmin
  Attempting uninstall: django-jazzmin
    Found existing installation: django-jazzmin 3.0.0
    Uninstalling django-jazzmin-3.0.0:
      Successfully uninstalled django-jazzmin-3.0.0
Successfully installed django-jazzmin-3.0.1
```

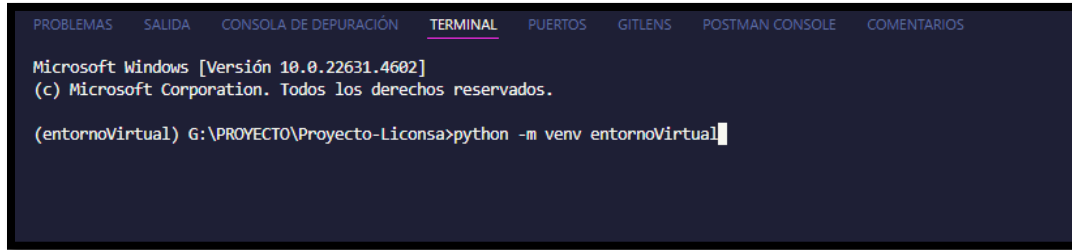
Ilustración 26. Instalación Django-jazzmin

Como se aprecia en la ilustración 26, se muestra el proceso de instalación de la librería Django-Jazzmin mediante la terminal de Visual Studio Code. Esta herramienta permite personalizar la interfaz administrativa de Django, ofreciendo un diseño más moderno y funcional para mejorar la experiencia de administración del sistema.

```
settings.py X
sav-c > settings.py > ...
33 AUTH_USER_MODEL = 'usuarios.Usuario'
34
35 Jimmy, hace 7 meses • Primer commit
36 # Application definition
37
38 INSTALLED_APPS = [
39     'jazzmin',
40     'django.contrib.admin',
41     'django.contrib.auth',
42     'django.contrib.contenttypes',
43     'django.contrib.sessions',
44     'django.contrib.messages',
45     'django.contrib.staticfiles',
46     'widget_tweaks',
47     'rest_framework',
48 ]
```

Ilustración 27. Jazzmin añadido a installed_apps

En la ilustración 27 se observa la configuración del archivo settings.py, donde se registra Django-Jazzmin dentro de la lista INSTALLED_APPS. Esta inclusión es necesaria para que Django reconozca la librería y pueda aplicar la nueva plantilla visual en el módulo de administración.



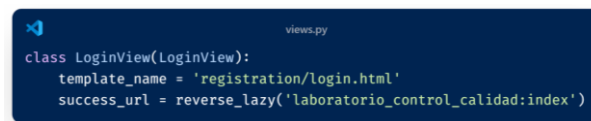
```
Microsoft Windows [Versión 10.0.22631.4602]
(c) Microsoft Corporation. Todos los derechos reservados.

(entornoVirtual) G:\PROYECTO\Proyecto-Liconsap>python -m venv entornoVirtual
```

Ilustración 28. Creación entorno virtual

Como se muestra en la ilustración 28, se realizó la creación de un entorno virtual utilizando el módulo venv de Python. Esta práctica permite aislar las dependencias del proyecto, asegurando que los paquetes instalados no afecten otras configuraciones del sistema y facilitando un entorno de desarrollo más controlado y replicable.

Login



```
views.py

class LoginView(LoginView):
    template_name = 'registration/login.html'
    success_url = reverse_lazy('laboratorio_control_calidad:index')
```

Ilustración 29. Vista LoginView

Herencia de LoginView: Esta clase utiliza la vista genérica LoginView que Django proporciona por defecto en su módulo de autenticación (django.contrib.auth).

template_name: Indica el archivo HTML que se usará como interfaz para el inicio de sesión, en este caso, registration/login.html. Este archivo debe estar ubicado en una carpeta llamada registration, que es la ubicación predeterminada que Django espera para las plantillas relacionadas con la autenticación.

success_url: Define la URL a la que se redirige al usuario después de un inicio de sesión exitoso. En este caso, apunta a laboratorio_control_calidad:index, que probablemente sea la vista principal de tu aplicación.

Ruta de login y logout

```
from django.urls import path
from . import views
from django.contrib.auth import views as auth_views

app_name = 'usuarios'

urlpatterns = [
    path('accounts/login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('perfil/', views.PerfilView.as_view(), name='perfil'),
    path('crear_usuario/', views.CrearUsuarioView.as_view(), name='crear_usuario'),
    path('listar_usuarios/', views.ListarUsuariosView.as_view(), name='listar_usuarios'),
    path('data/', views.UsuariosDataView.as_view(), name='usuarios_data'),
    path('eliminar_usuario/<int:user_id>/', views.EliminarUsuarioView.as_view(), name='eliminar_usuario'),
    path('editar_usuario/<int:id>/', views.EditarUsuarioView.as_view(), name='editar_usuario'),
]
```

Ilustración 30. Rutas en app usuarios

accounts/login/: La URL en la que se encuentra la página de inicio de sesión. Cuando un usuario accede a esta dirección, se carga la plantilla especificada en `template_name`.

`LoginView.as_view()`: Llama a la vista genérica `LoginView`, que se encarga de gestionar el proceso de autenticación.

`name='login'`: Este nombre permite referirse a esta ruta de manera fácil en las plantillas o en otras partes del código usando `{% url 'usuarios:login' %}`.

Funcionamiento del login:

- Formulario de inicio de sesión:

Cuando un usuario accede a `/accounts/login/`, Django muestra un formulario que solicita `username` y `password`.

Este formulario se encuentra en `registration/login.html`.

- Validación de credenciales:

Django verifica las credenciales proporcionadas contra los usuarios registrados en la base de datos.

Si son correctas, inicia una sesión para el usuario y redirige a `success_url`.

- Redirección tras el login:

El usuario es enviado a la página especificada en `success_url`.

Ruta del logout

`logout/`: La URL para cerrar la sesión. Cuando un usuario accede a esta dirección, Django termina la sesión actual.

`LogoutView.as_view()`: Utiliza la vista genérica `LogoutView` para gestionar el proceso de cierre de sesión.

`name='logout'`: Permite referirse a esta ruta fácilmente usando `{% url 'usuarios:logout' %}`.

Funcionamiento del logout:

- Cierre de sesión:

Cuando el usuario accede a `/logout/`, la vista `LogoutView` destruye la sesión actual del usuario.

Esto implica eliminar las cookies asociadas y cualquier otra información de sesión almacenada.

- Redirección tras el logout:

Por defecto, `LogoutView` redirige a la página principal o una página específica configurada en `LOGOUT_REDIRECT_URL` en el archivo de configuración (`settings.py`).

Plantilla de Login (registration/login.html)

```
login.html

{% extends 'base_iniciar_sesion.html' %}
{% load widget_tweaks %}
{% block title %}
<title>Ingreso de usuarios</title>
{% endblock %}

{% block login %}
<div class="login-box d-flex p-4 d-flex justify-content-center">
  <div class="card card-outline card-primary">
    <div class="card-header text-center">
      <a href="{% url 'usuarios:login' %}" class="h1"><b>SAV</b>-C</a>
    </div>
    <div class="card-body">
      <p class="login-box-msg d-flex justify-content-center">Inicio de sesion</p>

      <form method="post">
        {# csrf_token #}
        <div class="input-group mb-3">
          {{ form.username|add_class:"form-control"|attr:"placeholder:Nombre de usuario" }}
          <div class="input-group-append">
            <div class="input-group-text">
              <span class="fas fa-user"></span>
            </div>
          </div>
        </div>
        <div class="input-group mb-3">
          {{ form.password|add_class:"form-control"|attr:"placeholder:Contraseña" }}
          <div class="input-group-append">
            <div class="input-group-text">
              <i class="fas fa-lock"></i>
            </div>
          </div>
        </div>
        {# if form.errors #}
        <div class="alert alert-danger" role="alert">
          Usuario o contraseña incorrectos
        </div>
        {# endif #}

        <div class="col-12">
          <button type="submit" class="btn btn-primary btn-block"> Iniciar sesion <i class="fas fa-sign-in-
alt"></i></button>
        </div>
      </form>
    </div>
  </div>
</div>
{% endblock %}
```

Ilustración 31. Código HTML de Login

Extensión de la Plantilla Base

La plantilla hereda una estructura base para mantener uniformidad en el diseño del proyecto.

Cargador de Herramientas

Se utiliza `widget_tweaks` para personalizar atributos de los campos del formulario de Django.

Título de la Página

Define un título específico para identificar claramente la página en el navegador.

Contenedor del Login

Estructura central donde se aloja el formulario de inicio de sesión, con diseño responsivo y estilo moderno.

Encabezado del Login

Incluye un logo y el título de la aplicación para facilitar la identificación visual del sistema.

Formulario de Inicio de Sesión

Integra los campos para ingresar el nombre de usuario y contraseña, con personalización de diseño y validaciones:

Los campos tienen atributos visuales adicionales como íconos y placeholders.

Se muestra un mensaje de error en caso de datos incorrectos.

Botón de Envío

Botón que permite al usuario enviar sus credenciales para autenticarse en el sistema.

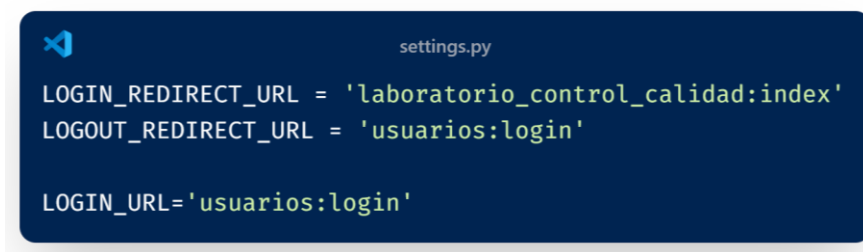
Seguridad

Utiliza el token CSRF para proteger el formulario contra ataques de tipo Cross-Site Request Forgery.

Diseño y Estilo

Implementa una estructura visual basada en tarjetas (cards) y clases de Bootstrap, asegurando una interfaz intuitiva y profesional.

Configuración adicional en settings.py



```
settings.py

LOGIN_REDIRECT_URL = 'laboratorio_control_calidad:index'
LOGOUT_REDIRECT_URL = 'usuarios:login'

LOGIN_URL='usuarios:login'
```

Ilustración 32. Configuración settings.py

LOGIN_REDIRECT_URL

Propósito:

Esta variable especifica la URL a la que se redirige al usuario después de iniciar sesión correctamente.

Valor:

'laboratorio_control_calidad:index' representa un nombre de ruta definido en urls.py. Django utiliza este nombre para redirigir al usuario al lugar indicado después del login. En este caso, parece que redirige a la página principal del módulo "Laboratorio Control de Calidad".

LOGOUT_REDIRECT_URL

Propósito:

Esta variable define la URL a la que se redirige al usuario después de cerrar sesión exitosamente.

Valor:

'usuarios:login' apunta a la ruta de inicio de sesión definida en la aplicación usuarios. Esto garantiza que, una vez que un usuario cierre sesión, sea redirigido de inmediato al formulario de login.

LOGIN_URL

Propósito:

Esta variable indica la URL que Django utiliza para redirigir automáticamente a los usuarios no autenticados cuando intentan acceder a una vista protegida (que requiere autenticación).

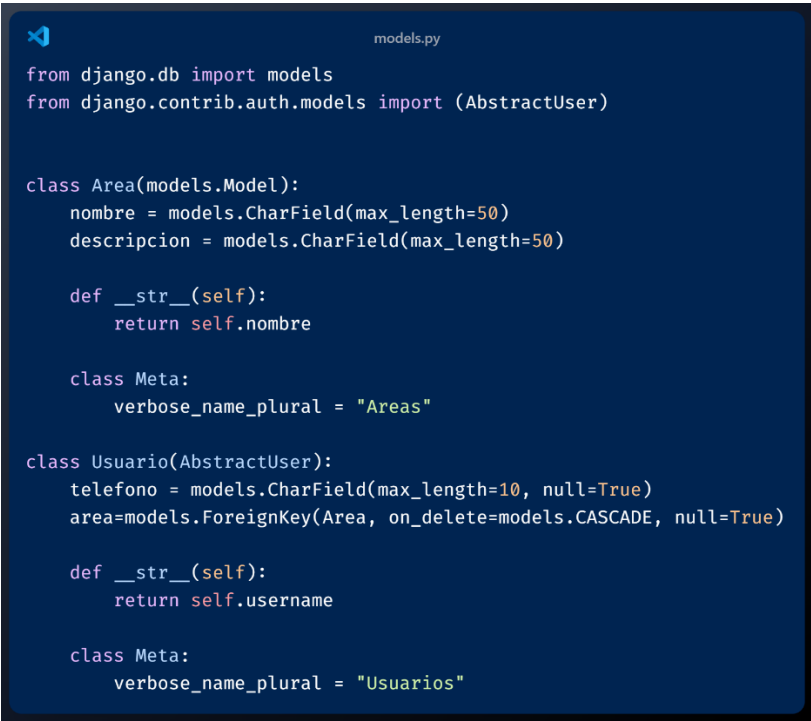
Valor:

'usuarios:login' apunta nuevamente a la ruta de inicio de sesión dentro de la app usuarios. Si un usuario intenta acceder a una vista restringida sin estar autenticado, será redirigido al formulario de login.

Modelado de datos

Esta sección abarca el diseño de los modelos en Django para la gestión de usuarios, roles y catálogos:

Descripción: Los modelos se implementaron en el archivo models.py, estructurando las relaciones entre usuarios, roles y catálogos mediante llaves foráneas. Se aplicaron validaciones personalizadas para garantizar la integridad de los datos.



```
models.py

from django.db import models
from django.contrib.auth.models import (AbstractUser)

class Area(models.Model):
    nombre = models.CharField(max_length=50)
    descripcion = models.CharField(max_length=50)

    def __str__(self):
        return self.nombre

    class Meta:
        verbose_name_plural = "Areas"

class Usuario(AbstractUser):
    telefono = models.CharField(max_length=10, null=True)
    area=models.ForeignKey(Area, on_delete=models.CASCADE, null=True)

    def __str__(self):
        return self.username

    class Meta:
        verbose_name_plural = "Usuarios"
```

Ilustración 33. Personalización de AbstractUser

La personalización de **AbstractUser** en Django permite modificar el modelo de usuario predeterminado para agregar campos adicionales o ajustar la funcionalidad a las necesidades del proyecto. En este caso, se extiende el modelo base para incluir información como el teléfono del usuario y su relación con un área específica dentro del sistema.

2. Importaciones

Se utilizan las siguientes librerías y clases:

- **models**: Permite definir y manejar modelos en Django.
- **AbstractUser**: Modelo base predeterminado de Django que incluye campos como username, password, email, etc.

Estas importaciones son esenciales para extender y personalizar el modelo de usuario.

3. Modelo Area

El modelo **Area** es una tabla que almacena información sobre las áreas o departamentos de una organización. Se relaciona directamente con los usuarios mediante una clave foránea.

- **Campos principales:**
 - nombre: Un campo de texto (máximo 50 caracteres) que almacena el nombre del área.
 - descripcion: Un campo de texto (máximo 50 caracteres) que describe brevemente el área.
- **Método __str__**: Devuelve el nombre del área como representación en texto. Esto facilita la lectura al trabajar con este modelo en el administrador de Django o en otros contextos.

- **Meta opciones:**

- Define que el plural de Area será "Áreas", ajustando su representación en la interfaz de administración.

Propósito: Este modelo ayuda a categorizar usuarios en diferentes áreas para una mejor organización y control.

4. Modelo Usuario

El modelo **Usuario** extiende la funcionalidad del modelo predeterminado de Django (AbstractUser), añadiendo campos personalizados y relaciones adicionales.

- **Herencia:**

- Se utiliza la herencia de **AbstractUser** para no duplicar la lógica existente, como autenticación y manejo de contraseñas.

- **Campos adicionales:**

1. **telefono:**

- Un campo de texto que almacena el número de teléfono del usuario (máximo 10 caracteres).
- Es opcional, ya que se define con null=True.

2. **area:**

- Relación **ForeignKey** con el modelo Area, lo que indica a qué área pertenece el usuario.
- Si se elimina un área, todos los usuarios asociados se eliminan automáticamente debido a on_delete=models.CASCADE.

- **Método __str__:**

- Devuelve el nombre de usuario (username) como representación en texto, lo que es útil en la administración o consultas rápidas.
- **Meta opciones:**
 - Cambia el nombre plural del modelo a "Usuarios" para una mejor legibilidad.

Propósito: Este modelo personalizado es la base del sistema de autenticación y control de acceso, adaptado específicamente a las necesidades del proyecto.

Models.py de app catálogos

Importaciones

A screenshot of a code editor with a dark blue background. The title bar shows a VS Code icon and the filename 'models.py'. The code is written in a light blue font and shows three import statements: 'from django.db import models', 'from django.core.exceptions import ValidationError', and 'from datetime import datetime, time'.

```
models.py
from django.db import models
from django.core.exceptions import ValidationError
from datetime import datetime, time
```

Ilustración 34. Importaciones models.py app catálogos

Descripción del código

El archivo models.py define la estructura de la base de datos para la app catalogos, incluyendo las relaciones entre las diferentes entidades. Este modelo abarca funcionalidades como la gestión de rutas, poblaciones, lecherías, y otros elementos clave para la organización y manejo de datos dentro del sistema. A continuación, se desglosa cada modelo:

```

models.py

class Ruta(models.Model):
    nombre = models.CharField(max_length=50)
    numero = models.IntegerField(unique=True)

    def __str__(self):
        return f'{self.numero} - {self.nombre}'

    class Meta:
        verbose_name_plural = "Rutas"

class Poblacion(models.Model):
    nombre = models.CharField(max_length=50)
    municipio = models.CharField(max_length=50)
    estado = models.CharField(max_length=50)

    def __str__(self):
        return self.nombre

    class Meta:
        verbose_name_plural = "Poblaciones"

class Lecheria(models.Model):
    numero = models.CharField(max_length=10, unique=True)
    nombre = models.CharField(max_length=50)
    responsable = models.CharField(max_length=50)
    telefono = models.CharField(max_length=50)
    direccion = models.CharField(max_length=50)
    ruta = models.ForeignKey(Ruta, on_delete=models.CASCADE)
    poblacion = models.ForeignKey(Poblacion, on_delete=models.CASCADE)

    def __str__(self):
        return '%s - %s - %s - %s' % (self.numero, self.nombre, self.ruta, self.poblacion)

    def clean(self):
        super().clean()
        if len(self.numero) != 10:
            raise ValidationError('El número debe tener exactamente 10 caracteres.')

    class Meta:
        verbose_name_plural = "Lecherias"

class Rotos(models.Model):
    fecha_venta = models.DateField()
    fecha_evaluacion = models.DateField()
    lecheria = models.ForeignKey(Lecheria, on_delete=models.CASCADE)
    rotos_reportados = models.IntegerField()

    def __str__(self):
        return '%s : %s' % (self.lecheria, self.rotos_reportados)

    class Meta:
        verbose_name_plural = "Rotos"

```

Ilustración 35. Modelos app catálogos primer parte del código después de importaciones

1. Modelo Ruta

- Propósito: Representa las rutas relacionadas con el sistema.
- Campos:
 - nombre: Define el nombre de la ruta con un límite de 50 caracteres.
 - numero: Un identificador único de tipo entero.
- Comportamiento:
 - Método `__str__`: Devuelve una representación en cadena que combina el número y el nombre.

- Meta: Define el nombre plural como "Rutas".

2. Modelo Poblacion

- Propósito: Almacena información de poblaciones vinculadas a las rutas.
- Campos:
 - nombre: Nombre de la población.
 - municipio: Municipio al que pertenece.
 - estado: Estado de la población.
- Comportamiento:
 - Método `__str__`: Devuelve el nombre de la población.
 - Meta: Define el nombre plural como "Poblaciones".

3. Modelo Lecheria

- Propósito: Representa una lechería, incluyendo su ubicación y contacto.
- Campos:
 - numero: Identificador único de 10 caracteres.
 - nombre: Nombre de la lechería.
 - responsable: Nombre del encargado.
 - telefono: Número de contacto.
 - direccion: Dirección de la lechería.
 - ruta: Relación con el modelo Ruta (clave foránea).
 - poblacion: Relación con el modelo Poblacion (clave foránea).
- Validación:

- Método `clean`: Valida que el campo `numero` tenga exactamente 10 caracteres.
- Comportamiento:
 - Método `__str__`: Devuelve una representación combinada de número, nombre, ruta y población.
 - Meta: Define el nombre plural como "Lecherías".

4. Modelo Rotos

- Propósito: Registra datos sobre lecherías y productos defectuosos o rotos.
- Campos:
 - `fecha_venta`: Fecha de la venta.
 - `fecha_evaluacion`: Fecha en la que se evaluó el producto.
 - `lecheria`: Relación con el modelo `Lecheria`.
 - `rotos_reportados`: Número de productos rotos reportados.
- Comportamiento:
 - Método `__str__`: Muestra la lechería y el número de productos rotos.
 - Meta: Define el nombre plural como "Rotos".

```

models.py

#Modelos generales y reportes
class Planta(models.Model):
    nombre = models.CharField(max_length=100)
    ubicacion = models.CharField(max_length=100)
    correo = models.EmailField(max_length=100)
    contacto = models.CharField(max_length=100)
    telefono = models.CharField(max_length=100)

    def __str__(self):
        return self.nombre

    class Meta:
        verbose_name_plural = "Plantas"

class Proveedor(models.Model):
    nombre = models.CharField(max_length=100)
    telefono = models.CharField(max_length=100)
    correo = models.EmailField(max_length=100)
    planta = models.ForeignKey(Planta, on_delete=models.CASCADE)

    def __str__(self):
        return self.nombre

    class Meta:
        verbose_name_plural = "Proveedores"

class Maquina(models.Model):
    numero = models.IntegerField()
    planta = models.ForeignKey(Planta, on_delete=models.CASCADE)

    def __str__(self):
        return f'{self.numero} - {self.planta}'

    class Meta:
        verbose_name_plural = "Máquinas"
        unique_together = ('numero', 'planta') # Asegura que el número sea único por planta

class Cabezal(models.Model):
    nombre = models.CharField(
        max_length=1,
        help_text="Ingrese una letra de la A a la F. A y B para la máquina 1, C y D para la máquina 2, E y F para la máquina 3."
    )
    maquina = models.ForeignKey(Maquina, on_delete=models.CASCADE)

    class Meta:
        verbose_name_plural = "Cabezales"
        unique_together = ('nombre', 'maquina') # Asegurar unicidad por máquina

    def clean(self):
        super().clean()

        # Convertir a mayúsculas para la validación de unicidad
        self.nombre = self.nombre.upper()

        # Definir las letras permitidas para cada máquina
        letras_permitidas = {
            1: ['A', 'B'],
            2: ['C', 'D'],
            3: ['E', 'F'],
        }
        # Permitir letras adicionales si es necesario

        # Validar letras adicionales
        letras_adicionales = 'GHIJKLMNOPQRSTUVWXYZ'

        # Validar el nombre del cabezal y la máquina
        if self.maquina.numero in letras_permitidas:
            if self.nombre not in letras_permitidas[self.maquina.numero] + list(letras_adicionales):
                raise ValidationError(f'El cabezal {self.nombre} no es válido para la máquina {self.maquina.numero}.')
        elif self.nombre not in letras_adicionales:
            raise ValidationError('El nombre del cabezal no está permitido.')

        # Comprobar unicidad del nombre del cabezal para la misma máquina
        if Cabezal.objects.filter(maquina=self.maquina, nombre__iexact=self.nombre).exclude(pk=self.pk).exists():
            raise ValidationError(f'El cabezal {self.nombre} ya existe para la máquina {self.maquina.numero}.')

    def save(self, *args, **kwargs):
        self.clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return f'{self.nombre} - {self.maquina}'

    class Meta:
        verbose_name_plural = "Cabezales"

```

Ilustración 36. Modelos app catálogos segunda parte del código

5. Modelo Planta

- Propósito: Define las plantas de producción dentro del sistema.
- Campos:
 - nombre: Nombre de la planta.
 - ubicacion: Dirección o ubicación de la planta.
 - correo: Correo electrónico de contacto.
 - contacto: Persona de contacto.
 - telefono: Teléfono de contacto.
- Comportamiento:
 - Método `__str__`: Devuelve el nombre de la planta.
 - Meta: Define el nombre plural como "Plantas".

6. Modelo Proveedor

- Propósito: Representa los proveedores asociados con una planta.
- Campos:
 - nombre: Nombre del proveedor.
 - telefono: Teléfono de contacto.
 - correo: Correo electrónico de contacto.
 - planta: Relación con el modelo Planta (clave foránea).
- Comportamiento:
 - Método `__str__`: Devuelve el nombre del proveedor.
 - Meta: Define el nombre plural como "Proveedores".

7. Modelo Maquina

- Propósito: Registra información de máquinas dentro de una planta.
- Campos:
 - numero: Número de identificación de la máquina.
 - planta: Relación con el modelo Planta.
- Comportamiento:
 - Método `__str__`: Devuelve el número de la máquina junto con la planta a la que pertenece.
 - Meta:
 - Define el nombre plural como "Máquinas".
 - Garantiza unicidad entre número y planta (`unique_together`).

8. Modelo Cabezal

- Propósito: Representa los cabezales asociados con una máquina.
- Campos:
 - nombre: Nombre o identificador del cabezal.
 - maquina: Relación con el modelo Maquina.
- Validación:
 - Método `clean`: Valida que el nombre del cabezal sea válido según la máquina asociada y que sea único dentro de la misma máquina.
- Comportamiento:
 - Método `__str__`: Devuelve el nombre del cabezal junto con la máquina asociada.
 - Meta:
 - Define el nombre plural como "Cabezales".
 - Garantiza unicidad entre nombre y máquina (`unique_together`).

```

models.py

class TipoProducto(models.Model):
    nombre = models.CharField(max_length=100)
    descripcion = models.CharField(max_length=100)

    def __str__(self):
        return self.nombre

    class Meta:
        verbose_name_plural = "Tipos de productos"

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    tipo_producto = models.ForeignKey(TipoProducto, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.nombre} - {self.tipo_producto}"

    class Meta:
        verbose_name_plural = "Productos"

class Turno(models.Model):
    TURNOS_CHOICES = [
        ('Matutino', 'Matutino'),
        ('Vespertino', 'Vespertino'),
    ]
    nombre = models.CharField(max_length=100, unique=True, choices=TURNOS_CHOICES)
    descripcion = models.TextField()
    hora_inicio = models.TimeField()
    hora_fin = models.TimeField()
    estatus = models.BooleanField(default=True)

    def __str__(self):
        if self.nombre == 'Matutino':
            return "X"
        elif self.nombre == 'Vespertino':
            return "Y"
        else:
            return self.get_nombre_display()

    def clean(self):
        # Define la hora limite para el turno matutino
        hora_limite = time(14, 0) # 2:00 PM

        # Validación para turnos matutinos
        if self.nombre == 'Matutino':
            if isinstance(self.hora_inicio, str):
                self.hora_inicio = datetime.strptime(self.hora_inicio, '%H:%M').time()
            if isinstance(self.hora_fin, str):
                self.hora_fin = datetime.strptime(self.hora_fin, '%H:%M').time()

            if self.hora_inicio >= hora_limite or self.hora_fin > hora_limite:
                raise ValidationError('El turno matutino no puede tener horas de inicio o fin después de las 2:00 PM.')

        # Validación para turnos vespertinos
        if self.nombre == 'Vespertino':
            if isinstance(self.hora_inicio, str):
                self.hora_inicio = datetime.strptime(self.hora_inicio, '%H:%M').time()
            if isinstance(self.hora_fin, str):
                self.hora_fin = datetime.strptime(self.hora_fin, '%H:%M').time()

            if self.hora_inicio < hora_limite or self.hora_fin <= hora_limite:
                raise ValidationError('El turno vespertino debe comenzar después de las 2:00 PM y terminar antes de las 11:59 PM.')
```

```

    def save(self, *args, **kwargs):
        self.clean() # Llama a la validación personalizada
        super().save(*args, **kwargs) # Llama al método save original

    class Meta:
        verbose_name_plural = "Turnos"

class Silo(models.Model):
    numero = models.IntegerField()
    capacidad = models.DecimalField(max_digits=10, decimal_places=2)
    producto = models.ForeignKey(Producto, on_delete=models.CASCADE)
    planta = models.ForeignKey(Planta, on_delete=models.CASCADE)

    def __str__(self):
        return str(self.numero)

    class Meta:
        verbose_name_plural = "Silos"
        unique_together = ('numero', 'planta') # Asegura que el número de silo sea único solo dentro de una planta

```

Ilustración 37. Modelos app catálogos tercera parte del código

9. Modelo TipoProducto

- Propósito: Clasifica los productos según su tipo.
- Campos:
 - nombre: Nombre del tipo de producto.
 - descripcion: Descripción breve del tipo de producto.
- Comportamiento:
 - Método `__str__`: Devuelve el nombre del tipo de producto.
 - Meta: Define el nombre plural como "Tipos de productos".

10. Modelo Producto

- Propósito: Representa productos específicos y su clasificación.
- Campos:
 - nombre: Nombre del producto.
 - tipo_producto: Relación con el modelo TipoProducto.
- Comportamiento:
 - Método `__str__`: Devuelve el nombre del producto junto con su tipo.
 - Meta: Define el nombre plural como "Productos".

11. Modelo Turno

- Propósito: Define turnos laborales con restricciones de tiempo.
- Campos:
 - nombre: Nombre del turno (Matutino o Vespertino).
 - descripcion: Descripción del turno.

- hora_inicio y hora_fin: Horarios del turno.
- estatus: Indica si el turno está activo.
- Validación:
 - Método clean: Valida las horas de inicio y fin según el turno (Matutino o Vespertino).
- Comportamiento:
 - Método __str__: Devuelve una representación personalizada según el turno.
 - Meta: Define el nombre plural como "Turnos".

12. Modelo Silo

- Propósito: Representa silos de almacenamiento en una planta.
- Campos:
 - numero: Número único del silo.
 - capacidad: Capacidad de almacenamiento en unidades decimales.
 - producto: Relación con el modelo Producto.
 - planta: Relación con el modelo Planta.
- Comportamiento:
 - Método __str__: Devuelve el número del silo.
 - Meta:
 - Define el nombre plural como "Silos".
 - Garantiza unicidad entre número y planta (unique_together).

Configuración de autenticación y permisos

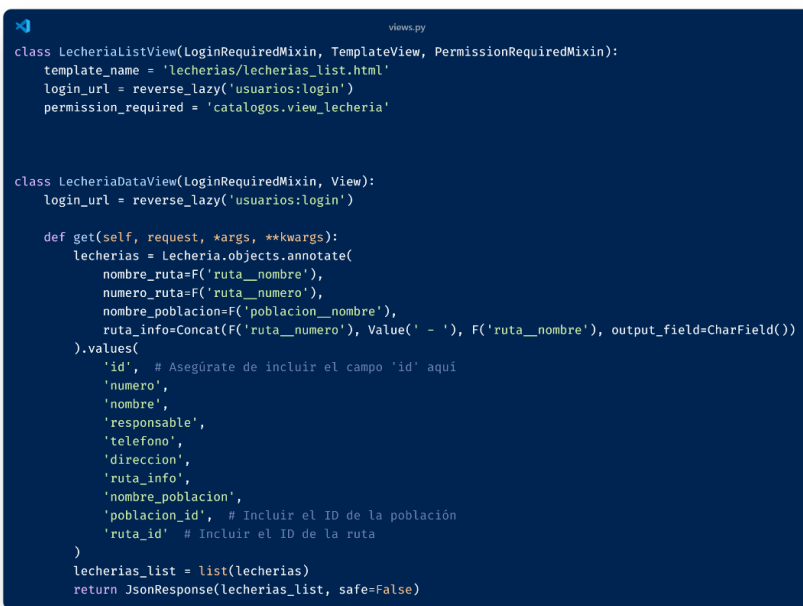
En esta sección se documenta la configuración del sistema de autenticación de Django y la implementación de permisos basados en roles:

Descripción: Se personalizó la clase `UserAdmin` para gestionar usuarios y roles jerárquicos. Los permisos se configuraron en el `admin.py`, asignando vistas específicas a cada rol.

Desarrollo del backend

Incluye las vistas, URLs y lógica de negocio del sistema:

Descripción: Se implementaron vistas personalizadas utilizando clases basadas en Django (`View`, `DetailView`, `UpdateView`). Las rutas se definieron en el archivo `urls.py`, y se añadieron métodos para manejar solicitudes relacionadas con la gestión de usuarios y catálogos como se aprecia en la ilustración 38.



```
views.py

class LecheriaListView(LoginRequiredMixin, TemplateView, PermissionRequiredMixin):
    template_name = 'lecherias/lecherias_list.html'
    login_url = reverse_lazy('usuarios:login')
    permission_required = 'catalogos.view_lecheria'

class LecheriaDataView(LoginRequiredMixin, View):
    login_url = reverse_lazy('usuarios:login')

    def get(self, request, *args, **kwargs):
        lecherias = Lecheria.objects.annotate(
            nombre_ruta=F('ruta__nombre'),
            numero_ruta=F('ruta__numero'),
            nombre_poblacion=F('poblacion__nombre'),
            ruta_info=Concat(F('ruta__numero'), Value(' - '), F('ruta__nombre')), output_field=CharField()
        ).values(
            'id', # Asegúrate de incluir el campo 'id' aquí
            'numero',
            'nombre',
            'responsable',
            'telefono',
            'direccion',
            'ruta_info',
            'nombre_poblacion',
            'poblacion_id', # Incluir el ID de la población
            'ruta_id' # Incluir el ID de la ruta
        )
        lecherias_list = list(lecherias)
        return JsonResponse(lecherias_list, safe=False)
```

Ilustración 38. Vistas List y Data de lecherias

```

class LecheriaCreateView(LoginRequiredMixin, View, PermissionRequiredMixin):
    login_url = reverse_lazy('usuarios:login')
    permission_required = 'catalogos.add_lecheria'

    def post(self, request, *args, **kwargs):
        data = request.POST
        numero = data.get('numero')
        nombre = data.get('nombre')
        responsable = data.get('responsable')
        telefono = data.get('telefono')
        direccion = data.get('direccion')
        ruta_id = data.get('ruta')
        poblacion_id = data.get('poblacion')

        # Verificar que todos los campos estén presentes
        if not all([numero, nombre, responsable, telefono, direccion, ruta_id, poblacion_id]):
            return JsonResponse({'error': 'Todos los campos son obligatorios'}, status=400)

        try:
            ruta = get_object_or_404(Ruta, id=ruta_id)
            poblacion = get_object_or_404(Poblacion, id=poblacion_id)

            # Crear la lecheria
            lecheria = Lecheria.objects.create(
                numero=numero,
                nombre=nombre,
                responsable=responsable,
                telefono=telefono,
                direccion=direccion,
                ruta=ruta,
                poblacion=poblacion
            )
            return JsonResponse({
                'id': lecheria.id,
                'numero': lecheria.numero,
                'nombre': lecheria.nombre,
                'responsable': lecheria.responsable,
                'telefono': lecheria.telefono,
                'direccion': lecheria.direccion,
                'ruta': lecheria.ruta.numero, # Cambiar si es necesario
                'poblacion': lecheria.poblacion.nombre # Cambiar si es necesario
            })

        except IntegrityError as e:
            if 'UNIQUE constraint failed' in str(e):
                return JsonResponse({'error': 'Ya existe una lecheria con este número.'}, status=400)
            return JsonResponse({'error': 'Error de integridad al crear la lecheria.'}, status=400)

        except ValidationError as e:
            return JsonResponse({'error': f'Error de validación: {e.messages}'}, status=400)

        except Exception as e:
            return JsonResponse({'error': f'Error desconocido: {str(e)}'}, status=500)

class LecheriaUpdateView(LoginRequiredMixin, View, PermissionRequiredMixin):
    login_url = reverse_lazy('usuarios:login')
    permission_required = 'catalogos.change_lecheria'

    def post(self, request, pk, *args, **kwargs):
        lecheria = get_object_or_404(Lecheria, pk=pk)
        data = request.POST

        numero = data.get('numero')
        nombre = data.get('nombre')
        responsable = data.get('responsable')
        telefono = data.get('telefono')
        direccion = data.get('direccion')
        ruta_id = data.get('ruta')
        poblacion_id = data.get('poblacion')

        if not (numero and nombre and responsable and telefono and direccion and ruta_id and poblacion_id):
            return JsonResponse({'error': 'Todos los campos son obligatorios'}, status=400)

        try:
            ruta = get_object_or_404(Ruta, id=ruta_id)
            poblacion = get_object_or_404(Poblacion, id=poblacion_id)
            lecheria.numero = numero
            lecheria.nombre = nombre
            lecheria.responsable = responsable
            lecheria.telefono = telefono
            lecheria.direccion = direccion
            lecheria.ruta = ruta
            lecheria.poblacion = poblacion
            lecheria.save()
            return JsonResponse({'status': 'success', 'message': 'Lecheria actualizada con éxito'})

        except IntegrityError:
            return JsonResponse({'error': 'El número de lecheria ya existe. Por favor, elija uno diferente.'}, status=400)

        except Exception as e:
            return JsonResponse({'status': 'error', 'message': str(e)}, status=400)

class LecheriaDeleteView(LoginRequiredMixin, View, PermissionRequiredMixin):
    login_url = reverse_lazy('usuarios:login')
    permission_required = 'catalogos.delete_lecheria'

    def delete(self, request, id, *args, **kwargs):
        try:
            lecheria = Lecheria.objects.get(pk=id)
            lecheria.delete()
            return JsonResponse({'status': 'success'})

        except Lecheria.DoesNotExist:
            return JsonResponse({'error': 'Lecheria no encontrada'}, status=404)

        except Exception as e:
            return JsonResponse({'error': str(e)}, status=400)

```

Ilustración 39 Vistas CRUD Lecherías

Como se aprecia en la ilustración 39, se muestran las vistas encargadas de gestionar las operaciones CRUD (crear, leer, actualizar y eliminar) para el módulo de lecherías. Estas vistas, implementadas en Django, permiten manipular los datos del catálogo de forma dinámica y controlada, de acuerdo con los permisos del usuario autenticado.

```
from django.urls import path
from . import views

app_name = 'catalogos'

urlpatterns = [
    path('lecherias/list/', views.LecheriaListView.as_view(), name='lecherias_list'),
    path('lecherias/list/data/', views.LecheriaDataView.as_view(), name='lecherias_data'),
    path('lecherias/create/', views.LecheriaCreateView.as_view(), name='crear_lecheria'),
    path('lecherias/update/<int:pk>/', views.LecheriaUpdateView.as_view(), name='actualizar_lecheria'),
    path('lecherias/delete/<int:id>/', views.LecheriaDeleteView.as_view(), name='eliminar_lecheria'),
    #rutas a poblaciones
    path('poblaciones/list/', views.PoblacionListView.as_view(), name='poblaciones_list'),
    path('poblaciones/list/data/', views.DataPoblacionView.as_view(), name='poblaciones_data'),
    path('poblaciones/create/', views.PoblacionCreateView.as_view(), name='crear_poblacion'),
    path('poblaciones/update/<int:pk>/', views.PoblacionUpdateView.as_view(), name='actualizar_poblacion'),
    path('poblaciones/delete/<int:pk>/', views.PoblacionDeleteView.as_view(), name='eliminar_poblacion'),
    #rutas a áreas
    path('areas/list/', views.AreaListView.as_view(), name='areas_list'),
    path('areas/list/data/', views.DataAreaView.as_view(), name='areas_data'),
    path('areas/create/', views.AreaCreateView.as_view(), name='crear_area'),
    path('areas/update/<int:pk>/', views.AreaUpdateView.as_view(), name='actualizar_area'),
    path('areas/delete/<int:pk>/', views.AreaDeleteView.as_view(), name='eliminar_area'),
    #rutas a maquinas
    path('maquinas/list/', views.MaquinaListView.as_view(), name='maquinas_list'),
    path('maquinas/list/data/', views.MaquinaDataView.as_view(), name='maquinas_data'),
    path('maquinas/create/', views.MaquinaCreateView.as_view(), name='crear_maquina'),
    path('maquinas/update/<int:pk>/', views.MaquinaUpdateView.as_view(), name='actualizar_maquina'),
    path('maquinas/delete/<int:id>/', views.MaquinaDeleteView.as_view(), name='eliminar_maquina'),
    #rutas a cabezales
    path('cabezales/list/', views.CabezalListView.as_view(), name='cabezales_list'),
    path('cabezales/list/data/', views.CabezalDataView.as_view(), name='cabezales_data'),
    path('cabezales/create/', views.CabezalCreateView.as_view(), name='crear_cabezal'),
    path('cabezales/update/<int:id>/', views.CabezalUpdateView.as_view(), name='actualizar_cabezal'),
    path('cabezales/delete/<int:id>/', views.CabezalDeleteView.as_view(), name='cabezal_delete'),
    #rutas a plantas
    path('plantas/list/', views.PlantaListView.as_view(), name='plantas_list'),
    path('plantas/list/data/', views.PlantaDataView.as_view(), name='plantas_data'),
    path('plantas/create/', views.PlantaCreateView.as_view(), name='crear_planta'),
    path('plantas/update/<int:pk>/', views.PlantaUpdateView.as_view(), name='actualizar_planta'),
    path('plantas/delete/<int:pk>/', views.PlantaDeleteView.as_view(), name='eliminar_planta'),
    #rutas a proveedores
    path('proveedores/list/', views.ProveedorListView.as_view(), name='proveedores_list'),
    path('proveedores/list/data/', views.ProveedorDataView.as_view(), name='proveedores_data'),
    path('proveedores/create/', views.ProveedorCreateView.as_view(), name='proveedores_create'),
    path('proveedores/update/<int:pk>/', views.ProveedorUpdateView.as_view(), name='proveedores_update'),
    path('proveedores/delete/<int:pk>/', views.ProveedorDeleteView.as_view(), name='proveedores_delete'),
    #rutas a turnos
    path('turnos/list/', views.TurnoListView.as_view(), name='turnos_list'),
    path('turnos/list/data/', views.TurnoDataView.as_view(), name='turnos_data'),
    path('turnos/create/', views.TurnoCreateView.as_view(), name='turnos_create'),
    path('turnos/update/<int:pk>/', views.TurnoUpdateView.as_view(), name='turnos_update'),
    path('turnos/delete/<int:pk>/', views.TurnoDeleteView.as_view(), name='turnos_delete'),
    #rutas a silos
    path('silos/list/', views.SiloListView.as_view(), name='silos_list'),
    path('silos/list/data/', views.DataSilosView.as_view(), name='silos_data'),
    path('silos/create/', views.SiloCreateView.as_view(), name='silo_create'),
    path('silos/update/<int:silo_id>/', views.SiloUpdateView.as_view(), name='silo_update'),
    path('silos/delete/<int:silo_id>/', views.SiloDeleteView.as_view(), name='silo_delete'),
    #rutas a gestion de productos
    path('productos/list/', views.ProductoListView.as_view(), name='productos_list'),
    path('productos/list/data/', views.ProductoDataView.as_view(), name='productos_data'),
    path('productos/create/', views.ProductoCreateView.as_view(), name='producto_create'),
    path('productos/update/<int:pk>/', views.ProductoUpdateView.as_view(), name='producto_update'),
    path('productos/delete/<int:producto_id>/', views.ProductoDeleteView.as_view(), name='producto_delete'),
    #rutas a gestion de tipos de productos
    path('tipo_producto/list/', views.TipoProductoListView.as_view(), name='tipo_producto_list'),
    path('tipo_producto/list/data/', views.TipoProductoDataView.as_view(), name='tipo_producto_data'),
    path('tipo_producto/create/', views.TipoProductoCreateView.as_view(), name='tipo_producto_create'),
    path('tipo_producto/update/<int:tipo_producto_id>/', views.TipoProductoUpdateView.as_view(),
    name='tipo_producto_update'),
    path('tipo_producto/delete/<int:tipo_producto_id>/', views.TipoProductoDeleteView.as_view(),
    name='tipo_producto_delete'),
    #rutas
    path('rutas/list/', views.RutaListView.as_view(), name='rutas_list'),
    path('rutas/list/data/', views.RutaDataView.as_view(), name='rutas_data'),
    path('rutas/create/', views.RutaCreateView.as_view(), name='rutas_create'),
    path('rutas/update/<int:pk>/', views.RutaUpdateView.as_view(), name='rutas_update'),
    path('rutas/delete/<int:id>/', views.RutaDeleteView.as_view(), name='rutas_delete'),
]
```

Ilustración 40. *Urls.py de app catalogos*

En la ilustración 40 se visualiza la configuración del archivo urls.py correspondiente a la app de catálogos. Aquí se definen las rutas que vinculan las solicitudes del usuario con las vistas correspondientes, facilitando así la navegación dentro del sistema y el acceso a las funcionalidades de gestión de lecherías.

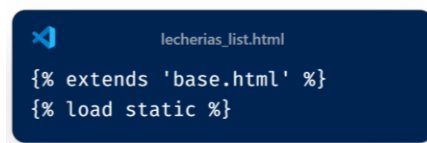
Desarrollo del frontend

Se detalla el diseño y la interacción del sistema mediante HTML, CSS y JavaScript:

Descripción: El diseño de las plantillas se realizó con el motor de plantillas de Django. Se incluyó JavaScript para manejar interacciones dinámicas, como validaciones en formularios y uso de modales para editar datos sin recargar la página.

Explicación del Código HTML para la Gestión de Lecherías

El código HTML presentado implementa una interfaz dinámica y funcional para la gestión de lecherías, utilizando la plantilla base: base.html extendida mediante Django Template Language. A continuación, se detallan las secciones clave de este archivo y su propósito dentro del sistema:

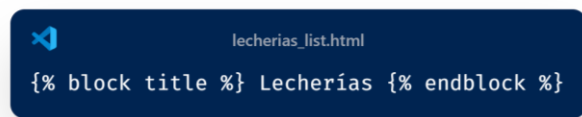


```
lecherias_list.html
{% extends 'base.html' %}
{% load static %}
```

Ilustración 41. Extensión de plantilla base

Extensión de plantilla: El archivo extiende una plantilla base común (base.html) para garantizar consistencia en el diseño y estructura del sistema.

Carga de archivos estáticos: Utiliza {% load static %} para habilitar la referencia a recursos estáticos como scripts o estilos.



```
lecherias_list.html
{% block title %} Lecherías {% endblock %}
```

Ilustración 42. Nombre de página

Define el título específico de esta página como "Lecherías", que se muestra en la pestaña del navegador.

A screenshot of a code editor with a dark blue background. The code is written in a light blue font and represents a Bootstrap 4 page header structure. It includes a container-fluid with a row containing a single column. Inside the column is a card with a dark gray header containing a bullhorn icon and the word 'Información'. The card body contains a callout box with the title 'Registro de Lecherías' and a paragraph of text. The code is enclosed in a block comment for a 'principal' block.

```
lecherias_list.html

{% block principal %}
{% block informacion %}
<div class="container-fluid p-4">
  <div class="row">
    <div class="col-xl-12 mt-1">
      <div class="card card-default">
        <div class="card-header bg-gray-dark">
          <h3 class="card-title">
            <i class="fas fa-bullhorn"></i>
            Información
          </h3>
        </div>
        <div class="card-body">
          <div class="callout callout-info">
            <h5>Registro de Lecherías</h5>
            <p>En esta sección se presenta el listado de lecherias.</p>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
{% endblock %}
```

Ilustración 43. Bloque principal encabezado pagina

Como se muestra en la ilustración 43, se presenta el bloque principal que contiene el encabezado de la página dentro del sistema. Este encabezado proporciona el título contextual de cada vista y facilita la orientación del usuario al navegar entre las diferentes secciones de la plataforma.

```

[! block datatable %]
<div class="container-fluid p-4">
  <div class="row">
    <div class="col-12">
      <div class="card">
        <div class="card-header d-flex justify-content-center align-items-center bg-gray-dark">
          <h3 class="card-title">Listado de Lecherías</h3>
        </div>
        <div class="card-body bg-light">
          <div class="col-xl-12 mt-3 p-3">
            <div class="table-responsive">
              <div class="d-flex justify-content-between mb-3">
                <button id="btnAddLecheria" class="btn btn-primary">
                  Añadir Lecheria
                  <i class="fas fa-plus"></i>
                </button>
                <div id="example_filter"></div>
              </div>
              <table id="tabla_lecherias" class="table table-bordered bg-light table-hover"
style="width:100%;">
                <thead class="thead-light" style="font-size: 14px;">
                  <tr>
                    <th>Número de lecheria</th>
                    <th>Nombre de lecheria</th>
                    <th>Responsable</th>
                    <th>Teléfono</th>
                    <th>Dirección</th>
                    <th>Número y nombre de ruta</th>
                    <th>Localidad</th>
                    <th>Opciones</th>
                  </tr>
                </thead>
                <tfoot style="font-size: 14px;">
                  <tr>
                    <th>Número de lecheria</th>
                    <th>Nombre de lecheria</th>
                    <th>Responsable</th>
                    <th>Teléfono</th>
                    <th>Dirección</th>
                    <th>Número y nombre de ruta</th>
                    <th>Localidad</th>
                    <th>Opciones</th>
                  </tr>
                </tfoot>
              </table>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

Ilustración 44. Bloque código para datatable

En la ilustración 44 se aprecia el bloque de código responsable de inicializar una tabla interactiva utilizando el plugin DataTables. Esta configuración permite mostrar registros de manera dinámica, habilitando funciones como la paginación, búsqueda y ordenamiento de los datos en el catálogo de lecherías.

```

lecherias.list.html
<!-- Modal para añadir/editar lechería -->
<div class="modal fade" id="lecheriaModal" tabindex="-1" role="dialog" aria-labelledby="lecheriaModalLabel" aria-
hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="lecheriaModalLabel">Añadir Lechería</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        <form id="lecheriaForm">
          <div class="form-group">
            <label for="numero">Número</label>
            <input type="text" class="form-control" id="numero" name="numero" maxlength="10" required>
          </div>
          <div class="form-group">
            <label for="nombre">Nombre de lechería</label>
            <input type="text" class="form-control" id="nombre" name="nombre" required>
          </div>
          <div class="form-group">
            <label for="responsable">Responsable</label>
            <input type="text" class="form-control" id="responsable" name="responsable" required>
          </div>
          <div class="form-group">
            <label for="telefono">Teléfono</label>
            <input type="text" class="form-control" id="telefono" name="telefono" required>
          </div>
          <div class="form-group">
            <label for="direccion">Dirección</label>
            <input type="text" class="form-control" id="direccion" name="direccion" required>
          </div>
          <div class="form-group">
            <label for="poblacion">Población</label>
            <div class="input-group">
              <select id="poblacion" name="poblacion" class="form-control" required>
                <!-- Opciones de población cargadas aquí -->
              </select>
              <div class="input-group-append">
                <button type="button" class="btn btn-secondary" id="btnAddPoblacion">Añadir
Población</button>
              </div>
            </div>
          </div>
          <div class="form-group">
            <label for="ruta">Ruta</label>
            <div class="input-group">
              <select id="ruta" name="ruta" class="form-control" required>
                <!-- Opciones de ruta cargadas aquí -->
              </select>
              <div class="input-group-append">
                <button type="button" class="btn btn-secondary" id="btnAddRuta">Añadir Ruta</button>
              </div>
            </div>
          </div>
        </form>
      </div>
      <input type="hidden" id="lecheriaId" name="lecheriaId">
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancelar</button>
        <button type="button" class="btn btn-primary" id="saveLecheria">Guardar</button>
      </div>
    </div>
  </div>
</div>

```

Ilustración 45. Código de añadir y editar registro con modal

Modales para Formularios

Los modales permiten realizar acciones específicas como añadir o editar registros directamente desde la interfaz sin recargar la página como se muestra en la ilustración 45.

Inclusión de JavaScript

A screenshot of a code editor showing HTML code for a file named 'lecherias_list.html'. The code includes a block for JavaScript, a comment indicating it's a script to initialize DataTables, and an <script> tag that dynamically loads 'js/datatables/datatable_lecherias.js' using the static attribute.

```
lecherias_list.html

{% block javascript %}
{{ block.super }}

<!-- Script para inicializar DataTables -->
<script src="{% static 'js/datatables/datatable_lecherias.js' %}"></script>
{% endblock %}
```

Ilustración 46. Código para cargar bloque de javascript

Propósito: Cargar un archivo JavaScript específico (datatable_lecherias.js) para inicializar y gestionar la tabla de lecherías con DataTables.

Obtener el token CSRF

A screenshot of a code editor showing JavaScript code for a file named 'datatable_lecherias.js'. It defines a 'getCookie' function that iterates through cookies to find a specific one by name and returns its value. At the bottom, it calls this function to retrieve the 'csrftoken' cookie.

```
datatable_lecherias.js

function getCookie(name) {
    var cookieValue = null;
    if (document.cookie && document.cookie !== '') {
        var cookies = document.cookie.split(';');
        for (var i = 0; i < cookies.length; i++) {
            var cookie = cookies[i].trim();
            if (cookie.substring(0, name.length + 1) === (name + '=')) {
                cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                break;
            }
        }
    }
    return cookieValue;
}

var csrftoken = getCookie('csrftoken');
```

Ilustración 47. CSRFToken

Este fragmento define una función `getCookie` que busca y extrae el valor de una cookie específica, en este caso, el token CSRF (`csrftoken`). Este token es necesario para proteger las solicitudes POST y DELETE contra ataques CSRF, garantizando que las acciones solo sean realizadas por usuarios autorizados.

```

// Configurar jQuery para incluir el token CSRF en las solicitudes AJAX
$.ajaxSetup({
  beforeSend: function(xhr, settings) {
    if (!/^(GET|HEAD|OPTIONS|TRACE)$/i.test(settings.type) && !this.crossDomain) {
      xhr.setRequestHeader("X-CSRFToken", csrftoken);
    }
  }
});

```

Ilustración 48 Configuración JQuery

Este código configura jQuery para que incluya automáticamente el token CSRF en el encabezado de las solicitudes AJAX. Esto asegura que todas las peticiones que modifiquen datos (como POST, PUT o DELETE) sean autorizadas por el servidor.

```

var table = $('#tabla_lecherias').DataTable({
  ajax: {
    url: '/catalogos/lecherias/list/data/',
    dataSrc: ''
  },
  columns: [
    { data: 'numero' },
    { data: 'nombre' },
    { data: 'responsable' },
    { data: 'telefono' },
    { data: 'direccion' },
    { data: 'ruta_info' },
    { data: 'nombre_poblacion' },
    {
      data: null,
      defaultContent: `
<div class="d-flex justify-content-between">
  <button class="btn btn-edit btn-warning"><i class="fa fa-pencil"></i></button>
  <button class="btn btn-delete btn-danger"><i class="fa fa-trash"></i></button>
</div>
`,
      orderable: false
    }
  ],
  pageLength: 5,
  lengthMenu: [[5, 10, 25, 50, -1], [5, 10, 25, 50, "Todo"]],
  language: {
    lengthMenu: "Mostrar _MENU_ entradas",
    zeroRecords: "No se encontraron resultados",
    info: "Mostrando entradas _START_ a _END_ de _TOTAL_",
    infoEmpty: "Mostrando entradas 0 a 0 de 0",
    infoFiltered: "(filtrado de _MAX_ entradas totales)",
    search: "Buscar:",
    paginate: {
      first: "Primero",
      last: "Último",
      next: "Siguiente",
      previous: "Anterior"
    }
  },
});

```

Ilustración 49 Asignación de valores dinámicos de la tabla lecherias

Este fragmento inicializa la tabla interactiva utilizando el plugin DataTables.

La tabla se llena dinámicamente con datos obtenidos de la URL `/catalogos/lecherias/list/data/` mediante una solicitud AJAX.

La configuración `columns` especifica los campos que se mostrarán en la tabla, y el último campo incluye botones para editar y eliminar registros.

La tabla también tiene configuraciones para paginación, búsqueda y mensajes personalizados en español.

Funciones de carga de datos

```
datatable_lecherias.js

// Función para cargar opciones de población
function loadPoblaciones(selectedPoblacion) {
  $.ajax({
    url: '/catalogos/poblaciones/list/data',
    method: 'GET',
    success: function(response) {
      var $poblacionSelect = $('#poblacion');
      $poblacionSelect.empty();

      if (response.length === 0) {
        $poblacionSelect.append('<option value="">No hay ninguna población disponible</option>');
      } else {
        $poblacionSelect.append('<option value="">Seleccione una población</option>');
        response.forEach(function(poblacion) {
          var selected = poblacion.id === selectedPoblacion ? 'selected' : '';
          var displayText = `${poblacion.nombre} (${poblacion.municipio}, ${poblacion.estado})`;
          $poblacionSelect.append('<option value="${poblacion.id}" ${selected}>${displayText}</option>');
        });
      }
    },
    error: function(xhr) {
      Swal.fire({
        icon: 'error',
        title: 'Error',
        text: 'No se pudieron cargar las poblaciones.',
      });
    }
  });
}
```

Ilustración 50 Carga de datos dinámicos de otra tabla dentro de tabla lecherias

Esta función envía una solicitud GET al servidor para obtener una lista de poblaciones.

Las opciones de población se cargan dinámicamente en un `<select>` con el ID `#poblacion`.

Si no hay poblaciones disponibles, muestra un mensaje indicando esto.

En caso de error, muestra una alerta utilizando SweetAlert2.

```

// Función para cargar opciones de rutas
function loadRutas(selectedRuta) {
  $.ajax({
    url: '/catalogos/rutas/list/data',
    method: 'GET',
    success: function(response) {
      var $rutaSelect = $('#ruta');
      $rutaSelect.empty();

      if (response.length === 0) {
        $rutaSelect.append('<option value="">No hay ninguna ruta disponible</option>');
      } else {
        $rutaSelect.append('<option value="">Seleccione una ruta</option>');
        response.forEach(function(ruta) {
          var selected = ruta.id === selectedRuta ? 'selected' : '';
          $rutaSelect.append('<option value="' + ruta.id + '" ' + selected + '>' + ruta.numero + ' - ' + ruta.nombre + '</option>');
        });
      }
    },
    error: function(xhr) {
      Swal.fire({
        icon: 'error',
        title: 'Error',
        text: 'No se pudieron cargar las rutas.',
      });
    }
  });
}

```

Ilustración 51 Carga dinámica de rutas en tabla lecherías

Similar a la función anterior, esta carga dinámicamente las opciones de rutas en un `<select>` con el ID `#ruta`. Incluye validación para manejar errores y mensajes amigables.

Eventos de los Modales

Abrir modal para añadir lechería

```
datatable_lecherias.js

// Abrir modal para añadir lechería
$('#btnAddLecheria').click(function() {
    // Cerrar los modales de población y ruta si están abiertos
    if ($('#poblacionModal').hasClass('show')) {
        $('#poblacionModal').modal('hide');
    }
    if ($('#rutaModal').hasClass('show')) {
        $('#rutaModal').modal('hide');
    }

    $('#lecheriaModalLabel').text('Añadir Lechería');
    $('#lecheriaForm')[0].reset();
    $('#lecheriaId').val('');
    $('#btnAddPoblacion').show();
    $('#btnAddRuta').show();
    $('#lecheriaModal').modal('show');
    loadPoblaciones(); // Cargar opciones de población
    loadRutas(); // Cargar opciones de ruta
});
```

Ilustración 52 Modal de adición

Cuando el usuario hace clic en el botón para añadir una lechería:

- Cierra otros modales abiertos, como población o ruta.
- Limpia los campos del formulario #lecheriaForm.
- Carga las opciones de población y ruta en los <select> correspondientes.
- Muestra el modal #lecheriaModal.

Misma acción para modales de ruta y población

```

// Abrir modal para añadir ruta
> $('#btnAddRuta').click(function() { ...
});

// Abrir modal para añadir población
> $('#btnAddPoblacion').click(function() { ...
});

// Abrir modal para añadir lechería
> $('#btnAddLecheria').click(function() { ...
});

```

Ilustración 53 Botones modales de adición

```

// Guardar lechería (añadir o editar) con validación usando SweetAlert2
$('#savelecheria').click(function() {
    var numero = $('#numero').val().trim();
    var nombre = $('#nombre').val().trim();
    var responsable = $('#responsable').val().trim();
    var telefono = $('#telefono').val().trim();
    var direccion = $('#direccion').val().trim();
    var ruta = $('#ruta').val().trim();
    var poblacion = $('#poblacion').val().trim();

    if (!numero || !nombre || !responsable || !telefono || !direccion || !ruta || !poblacion) {
        Swal.fire({
            icon: 'error',
            title: 'Error',
            text: 'Todos los campos son obligatorios',
        });
        return;
    }

    var numberPattern = /^[0-9]{10}$/;
    if (!numberPattern.test(numero)) {
        Swal.fire({
            icon: 'error',
            title: 'Error',
            text: 'El número debe tener exactamente 10 dígitos',
        });
        return;
    }

    var lecheriaId = $('#lecheriaId').val();
    var url = lecheriaId ? '/catalogos/lecherias/update/' + lecheriaId : '/catalogos/lecherias/create/';
    var method = 'POST';

    $.ajax({
        url: url,
        method: method,
        data: $('#lecheriaForm').serialize(),
        success: function(response) {
            $('#lecheriaModal').modal('hide');
            table.ajax.reload();
            Swal.fire({
                icon: 'success',
                title: 'Guardado',
                text: 'Lechería guardada con éxito',
            });
        },
        error: function(xhr) {
            var errorMessage = 'Error al guardar la lechería';
            if (xhr.status === 400 || xhr.responseJSON || xhr.responseJSON.error) {
                errorMessage = xhr.responseJSON.error;
            }
            Swal.fire({
                icon: 'error',
                title: 'Error',
                text: errorMessage,
            });
        }
    });
});

```

Ilustración 54 Función guardado y edición de lechería con sweetalert2

Este código guarda una lechería (nueva o editada):

- Valida que todos los campos del formulario estén completos.
- Envía una solicitud AJAX con los datos del formulario al servidor.
- Recarga la tabla al guardar exitosamente o muestra un error en caso de fallo.

Validaciones y seguridad

Se añadieron validaciones personalizadas en los modelos para campos obligatorios y restricciones específicas. También se configuraron protecciones contra CSRF y ataques SQL mediante las funcionalidades nativas de Django.

IV. RESULTADOS

El proyecto desarrollado permitió mejorar significativamente la gestión de usuarios y catálogos dentro del sistema, ofreciendo un enfoque centrado en la organización y control de acceso. La implementación de roles jerárquicos y catálogos dinámicos optimizó la administración de permisos y la consulta de datos relevantes, asegurando una experiencia más eficiente para los usuarios.

Resultados Obtenidos:

- Se logró desarrollar un sistema funcional que permite a los administradores asignar permisos y roles específicos, asegurando un control de acceso preciso y acorde a las necesidades del proyecto.
- La integración de Django como framework principal proporcionó un entorno seguro, escalable y eficiente, permitiendo la implementación de funcionalidades clave en tiempo récord.
- Los catálogos dinámicos mejoraron la consulta y edición de datos, brindando una experiencia más fluida a los usuarios finales.
- La personalización de vistas en función del rol del usuario incrementó la seguridad y optimizó el flujo de trabajo dentro del sistema.

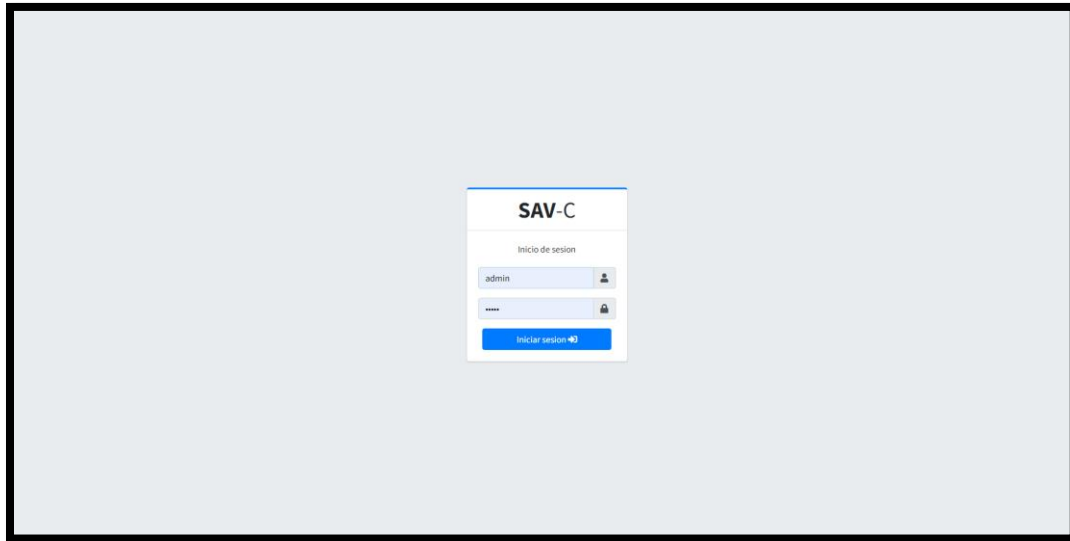


Ilustración 55 Vista final de ingreso al sistema

Como se aprecia en la ilustración 55, se muestra la versión final de la pantalla de login, donde el diseño intuitivo y el uso de plantillas modernas garantizan una experiencia de acceso clara y segura para el usuario.



Ilustración 56 Vista final de página de inicio del sistema

En la ilustración 56 se observa la página principal del sistema ya en funcionamiento, organizada con accesos directos a las funciones clave según el rol del usuario, permitiendo una navegación fluida y eficiente.

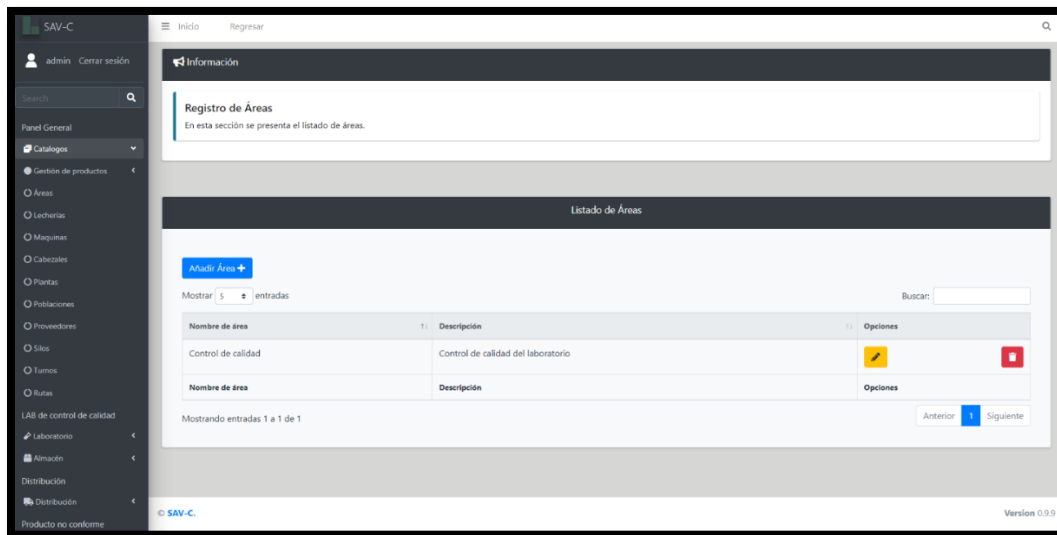


Ilustración 57 Vista del listado de catálogos

La ilustración 57 muestra el listado general de catálogos disponibles en el sistema. Esta vista facilita la gestión y consulta de datos dinámicos, permitiendo su edición o eliminación según los permisos asignados.

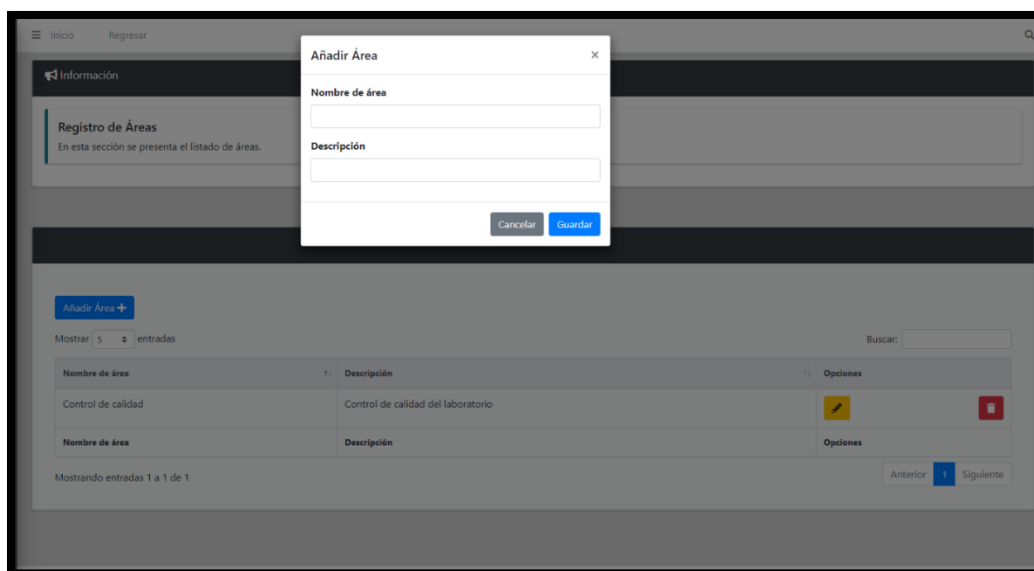


Ilustración 58 Vista modal de adición

Como se muestra en la ilustración 58, el sistema permite añadir registros a los catálogos mediante un modal interactivo, lo que mejora la experiencia de usuario al evitar recargas innecesarias y mantener el flujo de trabajo.

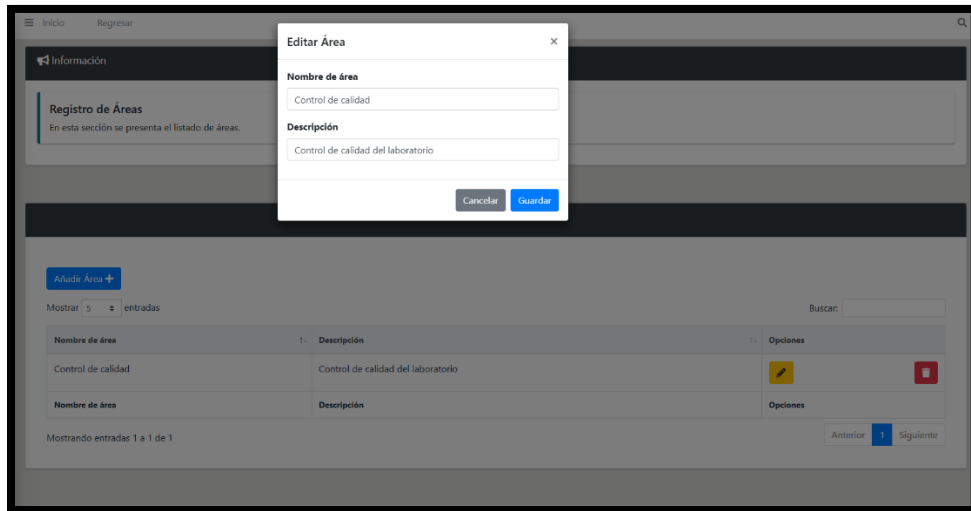


Ilustración 59 Vista de modal de edición

En la ilustración 59 se aprecia el uso del modal para editar registros existentes. Esta funcionalidad permite modificar los datos de manera precisa, con validaciones activas que aseguran la integridad de la información.

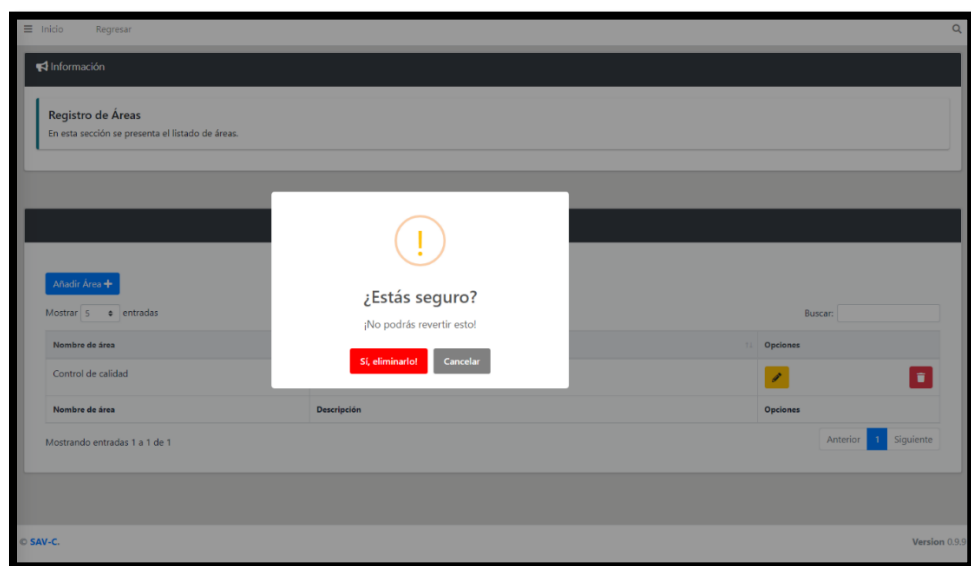


Ilustración 60 Vista de modal de eliminación de registro

La ilustración 60 presenta el modal de confirmación para eliminar registros. Utilizando SweetAlert, el sistema muestra una alerta visual que solicita confirmación del usuario antes de proceder, fortaleciendo la seguridad del sistema.

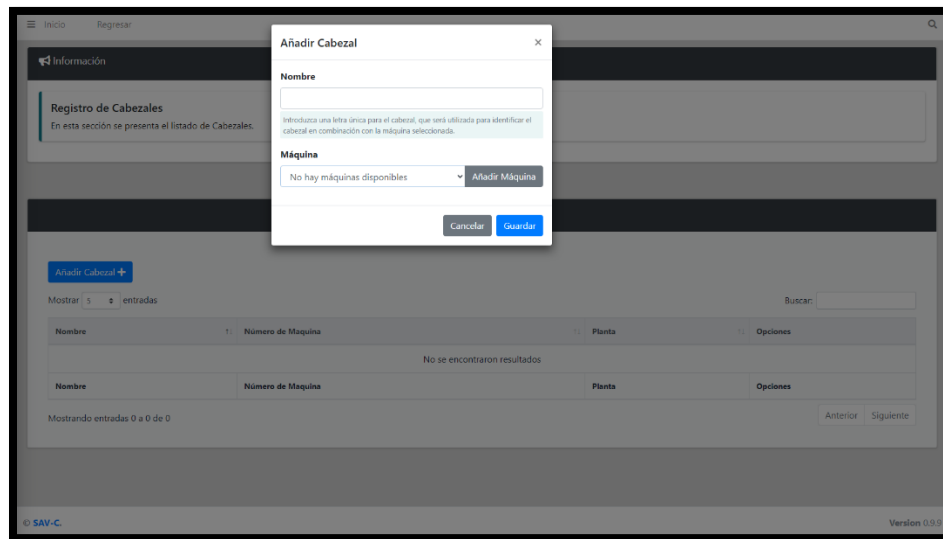


Ilustración 61 Vista de botón de adición dentro de otro modal

En la ilustración 61 se observa cómo se puede acceder a la creación de registros relacionados (por ejemplo, población o ruta) desde otro modal, permitiendo agregar elementos sin abandonar el flujo de trabajo principal.

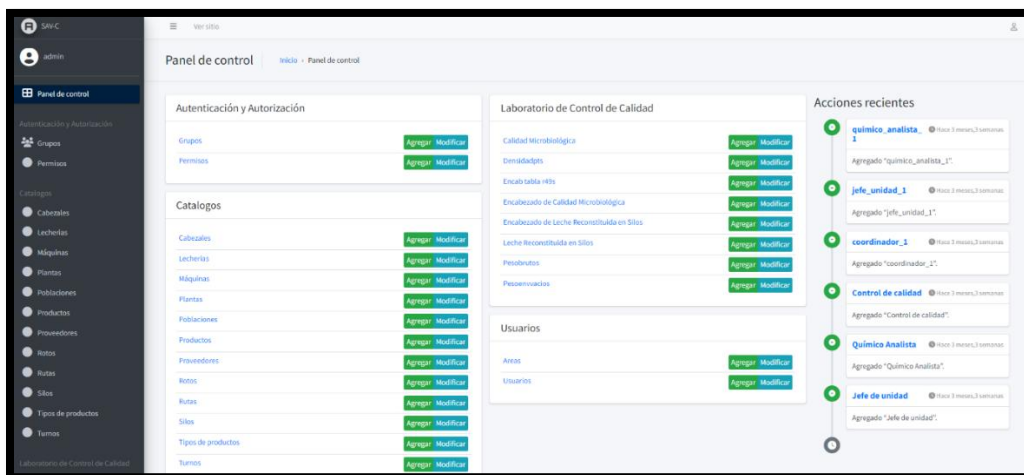


Ilustración 62 Vista de administración de usuarios

Como se muestra en la ilustración 62, el sistema ofrece un módulo de administración de usuarios donde es posible gestionar cuentas, roles y asignaciones de manera ordenada y visualmente clara, con filtros por área y permisos.

☐	Nombre de usuario	Nombre	Apellido	Dirección de correo electrónico	Telefono	Es staff	Area	Grupos
☐	admin							
☐	coordinador_1	Eduardo	Vasquez Conde	calle13.201@gmail.com	2412405674		Control de calidad	Coordinador
☐	jefe_unidad_1	Eduardo	Vasquez Conde	calle13.201@gmail.com	2412405674		Control de calidad	Jefe de unidad
☐	quimico_analista_1	Eduardo	Vasquez Conde	calle13.201@gmail.com	2412405674		Control de calidad	Químico Analista

Ilustración 63 Listado de usuarios y datos correspondientes del usuario asignados

Finalmente, en la ilustración 63 se presenta el listado completo de usuarios registrados, junto con sus datos asignados, como el área y grupo al que pertenecen, permitiendo una supervisión y control eficiente del acceso al sistema.

Checklist de funcionalidades validadas

La siguiente tabla resume las principales funciones implementadas y probadas durante el desarrollo del sistema, validando su correcto funcionamiento en entorno controlado.

Función probada	Resultado	Observaciones
Se probó el login	✓	Acceso seguro validado
Se validó el sistema de logout	✓	Cierre de sesión exitoso
Se creó y asignó grupos de usuarios	✓	Permisos asignados correctamente
Se registraron nuevos usuarios	✓	Roles asignados conforme a jerarquía definida
Se editaron y eliminaron usuarios	✓	Cambios reflejados en el sistema
Se visualizaron los catálogos dinámicos	✓	Acceso según permisos del rol
Se añadieron registros a los catálogos mediante modales	✓	Interfaz sin recarga, con validaciones
Se editaron registros de	✓	Ediciones reflejadas

los catálogos		inmediatamente
Se eliminaron registros de los catálogos	✓	Confirmación visual con SweetAlert
Se controló el acceso por roles	✓	Cada usuario solo accedió a funciones permitidas
Se validó el funcionamiento de DataTables	✓	Búsqueda, ordenamiento y paginación activas
Se integraron alertas visuales con SweetAlert	✓	Confirmaciones y mensajes efectivos
Se garantizó consistencia visual con AdminLTE	✓	Interfaz responsiva y profesional
Se verificó la conexión con la base de datos SQLite	✓	Consultas y actualizaciones exitosas
Se usaron tokens CSRF en formularios	✓	Protección activa contra ataques CSRF
Se aplicaron validaciones de datos en formularios	✓	Campos obligatorios bien gestionados

V. CONCLUSIONES

Conclusiones del Proyecto:

- El uso de metodologías ágiles, como Design Sprint, fue clave para estructurar las fases del proyecto y garantizar un enfoque iterativo, reduciendo riesgos y errores durante el desarrollo.
- La elección de tecnologías como Django, Python y SQLite demostró ser adecuada para cubrir los requerimientos del sistema, ofreciendo un balance entre simplicidad y funcionalidad.
- El proyecto resalta la importancia de la planificación y el análisis previo en la creación de soluciones tecnológicas, destacando cómo un enfoque ordenado puede impactar positivamente en los resultados.

Recomendaciones:

- En futuros proyectos, considerar la migración a bases de datos más robustas, como PostgreSQL, para manejar mayores volúmenes de datos y usuarios concurrentes.
- Explorar la posibilidad de incorporar herramientas de análisis de datos para generar reportes avanzados que complementen las funcionalidades existentes.
- Continuar optimizando las interfaces gráficas para garantizar una experiencia de usuario más intuitiva y atractiva.

Experiencia Personal y Profesional Adquirida: Durante el desarrollo de este proyecto, adquirí habilidades clave en el uso de herramientas tecnológicas como Django, HTML, CSS, JavaScript y Git, lo que fortaleció mis competencias en el desarrollo de sistemas de información. Además, trabajar con metodologías ágiles me permitió desarrollar un enfoque más estructurado y eficiente para abordar problemas complejos.

Asimismo, la experiencia reforzó mi capacidad para trabajar de manera colaborativa, resolver problemas técnicos y gestionar recursos tecnológicos, habilidades esenciales para mi crecimiento profesional. Este proyecto no solo representó un desafío técnico, sino también una oportunidad para aplicar los conocimientos adquiridos en un contexto real, contribuyendo al éxito de la solución implementada.

VI. COMPETENCIAS DESARROLLADAS

1. Administrar proyectos de desarrollo de software, como se abordó en la materia Gestión Ágil de Proyectos de Software, aplicando metodologías ágiles como Design Sprint para garantizar el cumplimiento de objetivos organizacionales, académicos y funcionales.

2. Diseñar e implementar sistemas de gestión de usuarios y permisos utilizando Django y Python, tal como se desarrolló en Arquitectura de Software y Taller de Ingeniería de Software, asegurando una administración segura, escalable y adaptable a distintos perfiles de usuario.
3. Aplicar tecnologías fundamentales como HTML, CSS y JavaScript para el desarrollo de interfaces web interactivas, optimizando la experiencia del usuario. Estas habilidades se reforzaron en Programación Web e Interacción Humano Computadora.
4. Analizar y resolver problemas de diseño e integración relacionados con la gestión de usuarios y catálogos, aplicando criterios metodológicos y técnicos vistos en Ingeniería de Software y Desarrollo de Aplicaciones para Dispositivos Móviles, permitiendo la integración eficiente de funcionalidades clave.
5. Desarrollar interfaces funcionales y responsivas personalizadas con HTML, CSS, JavaScript y plantillas de Django, asegurando claridad y accesibilidad para cada rol de usuario, como se practicó en Herramientas Emergentes de Desarrollo de Software y Interacción Humano Computadora.
6. Implementar sistemas de control de versiones con Git y GitHub, organizando y documentando el desarrollo de forma colaborativa, flujo trabajado desde Fundamentos de Investigación y utilizado ampliamente en prácticas de Taller de Investigación I y II.
7. Aplicar principios de seguridad y responsabilidad en el manejo de datos, garantizando la integridad de la información y el cumplimiento de aspectos éticos, legales y sociales, como se estudió en Administración y Seguridad de Redes y Auditoría en Tecnologías de la Información.

VII. REFERENCIAS

1. Django documentation. (s/f). Django Project. Recuperado el 8 de mayo de 2024, de <https://docs.djangoproject.com/en/5.1/>
2. Manual. (s/f). Datatables.net. Recuperado el 12 de abril de 2024, de <https://datatables.net/manual/>
3. Lucid Software Inc. (s.f.). Lucidchart: Diagrams Done Right. Recuperado de <https://www.lucidchart.com/>
4. Font Awesome. (s/f). Fontawesome.com. Recuperado el 12 de abril de 2024, de <https://fontawesome.com>
5. Trigás Gallego, M. (2012). Metodología Scrum. Universitat Oberta de Catalunya.
6. Python Release Python 3.11.0. (s/f). Python.org. Recuperado el 2 de abril de 2024, de <https://www.python.org/downloads/release/python-3110/>
7. SQLite Documentation. (s/f). Sqlite.org. Recuperado el 20 de mayo de 2024, de <https://www.sqlite.org/docs.html>
8. Jazzmin. (s/f). Readthedocs.io. Recuperado el 8 de mayo de 2024, de <https://django-jazzmin.readthedocs.io/>
9. Documentation. (s/f). Git-scm.com. Recuperado el 9 de mayo de 2024, de <https://git-scm.com/doc>
10. GitHub. (s.f.). Documentación de GitHub. Recuperado el 7 de mayo de 2024, de <https://docs.github.com/es>
11. Free Bootstrap Admin Template. (2021, abril 8). AdminLTE.IO. <https://adminlte.io/>
12. Welcome to. (s/f). Python.org. Recuperado el 10 de mayo de 2024, de <https://www.python.org/>
13. venv — Creation of virtual environments. (s/f). Python documentation. Recuperado el 10 de abril de 2024, de <https://docs.python.org/3/library/venv.html>