

MEMORIA DE PROYECTO CFGS – DESARROLLO DE APLICACIONES WEB(DAW)

CURSO 2023 – 2024

GSAapp

Gestión Seguimiento Administracion app



Alumno: Guillermo Iglesias Velasco

Tutor individual: Eva Maria Rivas Ojanguren

Tutor colectivo: Eva Maria Rivas Ojanguren

Fecha de Entrega Memoria: 08 - Diciembre – 2023

ÍNDICE

1. INTRODUCCIÓN.....	3
1.1. Presentación y Objetivos.....	3
1.2. Contexto.....	3
1.3. Planteamiento del Problema.....	3
2. ESPECIFICACIÓN DE REQUISITOS.....	7
2.1. Introducción.....	7
2.2. Descripción general.....	7
2.3. Requisitos Específicos.....	8
2.3.1. Requerimientos Funcionales.....	8
2.3.2. Requerimientos de Interfaces Externas.....	9
2.3.2.1. Interfaces de los Usuarios.....	9
2.3.2.2. Interfaces Software.....	10
2.3.2.3. Interfaces de Comunicaciones.....	11
2.3.3. Obligaciones del Diseño.....	12
2.3.3.1. Estándares Cumplidos.....	12
2.3.4. Atributos.....	12
2.3.4.1. Seguridad.....	12
2.3.4.2. Facilidades de Mantenimiento.....	13
2.3.4.3. Portabilidad.....	14
3. ANÁLISIS.....	15
3.1. Introducción.....	15
3.2. Diagrama de Clases.....	15
3.3. Diagrama de Casos de Uso.....	15
4. DISEÑO.....	16
4.1. Introducción.....	16
4.2. Capa de Presentación.....	17
4.3. Capa de Negocio o Lógica de la Aplicación.....	24
4.4. Capa de Persistencia o Datos.....	25
5. IMPLEMENTACIÓN.....	29
5.1. Tecnologías utilizadas en el desarrollo del proyecto.....	29
5.2. Descripción del Proyecto.....	29
5.2.1. Capa de Presentación.....	29
5.2.2. Capa de Negocio o Lógica de la Aplicación.....	46
6. CONCLUSIÓN.....	83
6.1. Valoración Personal del Trabajo Realizado (análisis DAFO + análisis CAME).....	83
6.2. Bibliografía y Webgrafía.....	85

1. INTRODUCCIÓN

1.1. Presentación y Objetivos.

Mi proyecto de fin de ciclo se centra en el desarrollo de una aplicación web denominada GSAapp, cuyo objetivo principal es la gestión y administración de los permisos asignados a los usuarios integrantes de la plantilla de una entidad empresarial. La concepción de este sistema se fundamenta en la necesidad de optimizar los procesos administrativos en los departamentos de recursos humanos, buscando facilitar la coordinación y control eficientes de las autorizaciones conferidas a los empleados. Además, la aplicación ofrece a los empleados de la empresa la facilidad de solicitar permisos de manera más cómoda e inmediata, agilizando el proceso de solicitud y respuesta. Esta característica contribuye a mejorar la experiencia del usuario y a impulsar la eficiencia en la gestión de permisos, al permitir una interacción ágil y accesible para todos los miembros de la plantilla.

El término "GSAapp" se deriva de "Gestión Seguimiento Administración app", reflejando así su función primordial de proporcionar una herramienta integral para la gestión, seguimiento y administración de los permisos dentro del contexto laboral.

1.2. Contexto.

Independientemente de la motivación específica detrás de mi proyecto final de ciclo, la concepción de la aplicación surgió a raíz de la observación de la creciente expansión de empresas, lo cual implica un aumento proporcional en el número de empleados. Este crecimiento plantea desafíos en la comunicación efectiva entre el departamento de recursos humanos y los demás departamentos, debido a la complejidad que conlleva la gestión de una plantilla más extensa.

La premisa central de la aplicación es abordar este desafío, facilitando la comunicación en entornos laborales de diversos tamaños, tanto para empresas grandes como para aquellas de menor escala. El objetivo es simplificar y dinamizar la interacción entre el departamento de recursos humanos y otros departamentos, eliminando las barreras tradicionales y promoviendo un flujo eficiente de información. Al prescindir del uso de papel y lápiz, la aplicación busca modernizar y agilizar los procesos de comunicación, contribuyendo así a la mejora de la gestión de recursos humanos en un espectro empresarial amplio.

1.3. Planteamiento del Problema.

El marco que impulsa la creación de la aplicación **GSAapp** se sitúa en el dinámico ámbito de la **gestión de recursos humanos**, donde la necesidad de **gestionar, administrar y dar seguimiento al estado de los empleados** se vuelve primordial. La aplicación se erige como una herramienta integral destinada a proporcionar una comunicación eficiente y sincronizada entre los **empleados y el departamento de recursos humanos**.

En la actualidad empresarial, la **gestión de la plantilla y la administración de permisos** representan desafíos cruciales. **GSAapp busca simplificar estos procesos** al permitir que los empleados soliciten sus permisos de manera fluida y eficaz a través de la aplicación. Esta funcionalidad no solo agiliza el proceso de solicitud, sino que también facilita una interacción más directa y sincrónica entre los empleados y el departamento de recursos humanos.

La aplicación, al ser una **plataforma centralizada**, posibilita a los empleados **gestionar sus solicitudes de permisos de manera autónoma**. Simultáneamente, proporciona a los responsables de recursos humanos las herramientas necesarias para **revisar, gestionar y tomar decisiones informadas sobre las solicitudes recibidas**. Esta comunicación bidireccional optimiza **la toma de decisiones**, contribuyendo a una gestión más ágil y transparente de las actividades laborales y de los permisos concedidos.

GSAapp, al integrarse en este contexto laboral, se convierte en un facilitador esencial para la comunicación y la gestión de permisos entre empleados y recursos humanos. Su propósito central es ofrecer una solución que, a través de la tecnología, mejore la eficiencia operativa y ****fortalezca la conexión colaborativa entre los distintos actores dentro de la empresa****.

1.4. Análisis de costes.

En esta sección, se presenta una tabla que detalla los elementos y los costos empleados en la ejecución del proyecto.

RECURSOS	COSTE
ERAZER GAMING NOTEBOOK	800 €
Eclipse IDE for Java Developers	0 €
Java con Spring Boot	0 €
HTML5	0 €
JavaScript	0 €
JQuery	0 €
CSS (Cascading Style Sheets)	0 €
DBeaver Community	0 €
MySQL	0 €
XAMP	0 €
000Webhost	0 €

- **ERAZER GAMING NOTEBOOK:** Portátil de la marca ERAZER diseñado para actividades de gaming y desarrollo.
- **Eclipse IDE for Java Developers:** Entorno de desarrollo integrado para programación en Java. En este caso, la versión utilizada para desarrolladores Java es de código abierto.
- **Java con Spring Boot:** Es el entorno de desarrollo utilizado para la implementación del proyecto. Java es un lenguaje de programación de código abierto, y Spring Boot es un marco que facilita el desarrollo de aplicaciones Java.

- **HTML5:** Última versión del lenguaje de marcado estándar para la creación y presentación de contenido en la web. Es de código abierto.
- **JavaScript:** Lenguaje de programación utilizado para agregar interactividad a las páginas web. Es de código abierto.
- **JQuery:** Biblioteca de JavaScript que simplifica la manipulación del DOM y la interacción con el usuario. Es de código abierto.
- **CSS (Cascading Style Sheets):** Se utiliza para dar estilo y mejorar la presentación de las páginas web. Es de código abierto.
- **DBeaver Community:** Herramienta de administración de bases de datos. La versión "Community" es gratuita y ofrece funcionalidades esenciales para el desarrollo y la gestión de bases de datos.
- **MySQL:** Sistema de gestión de bases de datos relacional utilizado en el proyecto. Es de código abierto y se integra con DBeaver para la administración de datos.
- **XAMP:** Proporciona un entorno de servidor local para el desarrollo web. Es de código abierto.
- **000Webhost:** Servicio de alojamiento web gratuito. Permite alojar sitios web sin costes asociados, siendo una opción económica para el despliegue y prueba de proyectos web.

1.5. Plan de Financiación.

A través de Kickstarter, una plataforma de financiación colectiva, también reconocida como crowdfunding, se brinda a emprendedores, artistas, creadores y diversos proyectos la oportunidad de obtener fondos con el propósito de materializar sus iniciativas. Esta plataforma no solo opera como un medio de recaudación de recursos, sino también como un megáfono estratégico para promocionar la aplicación ante audiencias más amplias, incluyendo grandes empresas.

Dada su naturaleza como sistema de donaciones, aquellos que contribuyen financieramente al proyecto en Kickstarter reciben recompensas específicas. En este contexto, se ofrece a los donantes una versión gratuita de la aplicación como una forma de reconocimiento y agradecimiento por su respaldo financiero. Este enfoque contribuye a fortalecer la relación entre los creadores del proyecto y sus partidarios, al tiempo que estimula la difusión y adopción de la aplicación en la audiencia objetivo.

1.6. Plan de recursos humanos.

OCUPACIÓN	PERSONAL	DNI
Técnico Desarrollo Aplicaciones Web	Guillermo Iglesias Velasco	71775551 - A

1.7. Plan de prevención de riesgos.

Aunque el ámbito de este sector no implica una carga física intensiva, no se puede ignorar la presencia de riesgos inherentes, entre los que se incluyen:

❖ **Fatiga visual o muscular:**

- La exposición prolongada a pantallas o la realización de tareas repetitivas pueden contribuir a la fatiga visual y muscular. Esto puede afectar la productividad y el bienestar de los trabajadores.

❖ **Golpes o caídas:**

- Aunque el entorno de trabajo no sea eminentemente físico, aún existe el riesgo de golpes o caídas, especialmente al considerar posibles distracciones o condiciones del entorno que puedan causar accidentes.

❖ **Contacto eléctrico:**

- La presencia de dispositivos electrónicos y equipos en el entorno laboral introduce el riesgo de contacto eléctrico. Este riesgo puede manifestarse debido a cables defectuosos, equipos mal mantenidos o instalaciones eléctricas inadecuadas.

❖ **Carga mental:**

- Las demandas cognitivas pueden generar una carga mental significativa. Tareas complejas, plazos ajustados o ambientes laborales estresantes pueden contribuir a la carga mental, afectando la salud mental y el rendimiento laboral.

❖ **Distintos factores en la organización:**

- La complejidad organizativa, cambios en la estructura de la empresa o insuficiente comunicación interna pueden generar incertidumbre y estrés entre los trabajadores.

Soluciones:

❖ **Fatiga visual o muscular:**

- Implementar pausas regulares para descansar la vista y realizar ejercicios de estiramiento muscular.
- Asegurar la ergonomía de estaciones de trabajo para reducir la fatiga física.

❖ **Golpes o caídas:**

- Mantener áreas de trabajo limpias y organizadas.
- Fomentar la conciencia sobre la importancia de la seguridad en el lugar de trabajo.

❖ **Contacto eléctrico:**

- Realizar inspecciones periódicas de equipos eléctricos y cables.
- Proporcionar capacitación sobre seguridad eléctrica y promover el uso de dispositivos de protección adecuados.

❖ **Carga mental:**

- Distribuir tareas de manera equitativa y establecer plazos realistas.
- Ofrecer programas de apoyo y recursos para la gestión del estrés.

❖ **Distintos factores en la organización:**

- Mejorar la comunicación interna para mantener a los empleados informados sobre cambios organizativos.
- Facilitar canales para que los empleados expresen inquietudes y sugerencias.

Al abordar estos riesgos de manera proactiva y promover prácticas seguras, se contribuye a mantener un entorno laboral saludable y seguro para todos los involucrados en el sector.

2. ESPECIFICACIÓN DE REQUISITOS.

2.1. Introducción

La sección de Especificación de Requisitos constituye un componente esencial en el desarrollo de cualquier proyecto, proporcionando una estructura detallada y comprensible de los elementos fundamentales que guiarán el diseño y la implementación de la aplicación GSAapp. En esta fase crucial, definimos de manera exhaustiva los requisitos específicos que la aplicación debe cumplir para satisfacer las necesidades y expectativas planteadas.

A lo largo de esta sección, se desglosan los distintos aspectos, desde los requisitos funcionales hasta las interfaces de usuario y las restricciones técnicas, con el objetivo de brindar una visión integral de los elementos esenciales para el éxito del proyecto. El análisis detallado aquí presentado servirá como cimiento para las etapas subsiguientes del desarrollo, asegurando una implementación coherente y alineada con los objetivos trazados inicialmente.

2.2. Descripción general.

GSAapp, abreviatura de "Gestión, Seguimiento y Administración de Permisos," representa una aplicación web innovadora concebida para abordar los desafíos inherentes a la gestión de permisos en entornos laborales modernos. Diseñada con un enfoque integral, GSAapp busca revolucionar la manera en que las empresas coordinan y supervisan las

autorizaciones concedidas a sus empleados, estableciendo una plataforma eficiente y colaborativa.

Características Clave:

❖ **Gestión Centralizada:**

- GSAapp proporciona una interfaz centralizada que simplifica la gestión de permisos, permitiendo a los responsables de recursos humanos supervisar y autorizar solicitudes de manera eficiente.

❖ **Automatización de Procesos:**

- La aplicación agiliza los procedimientos administrativos al incorporar funciones de automatización, reduciendo tiempos y recursos necesarios para la gestión de permisos.

❖ **Comunicación Bidireccional:**

- Facilita una comunicación efectiva entre empleados y departamentos de recursos humanos, mejorando la coordinación y el control sobre las autorizaciones otorgadas.

❖ **Acceso Remoto:**

- GSAapp ofrece la flexibilidad del acceso remoto, permitiendo a los responsables de recursos humanos gestionar permisos desde cualquier ubicación, lo que contribuye a una mayor agilidad operativa.

❖ **Seguridad y Cumplimiento:**

- La aplicación ha sido diseñada con un enfoque robusto en seguridad de datos y cumplimiento de normativas laborales, garantizando un manejo seguro de la información sensible.

2.3. Requisitos Específicos.

2.3.1. Requerimientos Funcionales.

Los requerimientos funcionales de GSAapp se centran en las acciones y capacidades específicas que la aplicación debe brindar a los usuarios. Aquí se presentan algunos ejemplos de posibles requerimientos funcionales para GSAapp:

❖ **Registro de Usuarios:**

- Los empleados y responsables de recursos humanos deben poder registrarse en la aplicación con información verificada y roles específicos.

❖ **Solicitud de Permisos:**

- Los empleados deben poder solicitar permisos específicos a través de la aplicación, proporcionando detalles como la fecha, duración y motivo.

❖ **Aprobación y Rechazo de Permisos:**

- Los responsables de recursos humanos deben tener la capacidad de revisar y aprobar o rechazar las solicitudes de permisos recibidas.

❖ **Calendario de Permisos:**

- La aplicación debe contar con un calendario que muestre de manera visual los permisos aprobados y planificados, facilitando la coordinación y planificación.

❖ **Acceso Remoto y Móvil:**

- Los usuarios deben poder acceder a la aplicación de forma remota y a través de dispositivos móviles para gestionar permisos desde cualquier ubicación.

❖ **Gestión de Usuarios y Roles:**

- Los administradores de la aplicación deben tener la capacidad de gestionar usuarios, asignar roles y ajustar permisos según las necesidades organizativas.

❖ **Seguridad de Datos:**

- La aplicación debe implementar medidas de seguridad robustas para proteger la información sensible relacionada con los permisos de los empleados.

2.3.2. Requerimientos de Interfaces Externas.

2.3.2.1. Interfaces de los Usuarios.

La aplicación presenta dos interfaces distintas, una destinada a los **usuarios** y otra diseñada para los **administradores**. En la interfaz de usuario, se implementa una barra de navegación con un menú desplegable que permite **cerrar la sesión**. A continuación, se exhibe un contenedor con formato de tarjeta que muestra la **foto de perfil** del usuario junto con su **información**. Adyacente a este contenedor, se dispone otro que presenta el **historial de los permisos solicitados o en proceso de solicitud** por parte del usuario.

Dentro de esta interfaz, se incorpora un botón que redirige a una página con un **calendario**, permitiendo al usuario seleccionar la **fecha y el tipo de permiso** deseado. Una vez completada esta acción, se redirige al usuario a la página principal, y el botón correspondiente queda **deshabilitado hasta que el permiso sea gestionado por el administrador**.

La interfaz destinada a los administradores guarda similitudes con la de los usuarios. Además de contener la **información del usuario**, incluye un enlace que dirige a la **página de registro de nuevos usuarios**. Asimismo, se integra una **tabla que presenta a los usuarios con nivel de acceso "usuario" en la aplicación**, ofreciendo al administrador la capacidad de **gestionar sus permisos**. Se incluye un botón denominado **"Ver Asunto**"**, que lleva a otra página donde se despliega información detallada del respectivo usuario. Si el usuario tiene **permisos pendientes**, el administrador puede **aceptar o denegar dichas solicitudes**. En caso de aceptación, los **permisos se registran en el historial del usuario con el estado actualizado**, mientras que, en caso de denegación, se eliminan.

2.3.2.2. Interfaces Software.

La correcta operación de la aplicación GSAapp requiere la presencia de ciertos elementos de software que garantizan su funcionalidad óptima. A continuación, se detallan los requisitos de software esenciales:

❖ **Navegador Web Compatible:**

- GSAapp es compatible con los navegadores web modernos, como Google Chrome, Mozilla Firefox, Microsoft Edge y Safari. Se recomienda utilizar la última versión disponible para una experiencia óptima.

❖ **Entorno de Ejecución Java:**

- Para el correcto funcionamiento del back-end desarrollado en Java con Spring Boot, se requiere un entorno de ejecución Java instalado en el servidor donde se aloja la aplicación.

❖ **Servidor Web:**

- La aplicación se despliega en un servidor web compatible con Java, como Apache Tomcat o cualquier servidor compatible con contenedores de servlets.

❖ **Sistema de Gestión de Base de Datos:**

- GSAapp utiliza MySQL como sistema de gestión de base de datos. Se debe contar con una instancia de MySQL configurada y accesible para el almacenamiento y recuperación de datos.

❖ **Intérprete de Thymeleaf:**

- La interfaz de usuario se genera mediante Thymeleaf. El sistema requiere un intérprete de Thymeleaf compatible para procesar y renderizar las plantillas de manera adecuada.

❖ **Habilitación de JavaScript y jQuery:**

- Para la interfaz de usuario dinámica, se debe asegurar la habilitación de JavaScript y jQuery en el navegador, ya que estas tecnologías se utilizan para mejorar la interactividad.

❖ **Conexión a Internet:**

- La aplicación requiere una conexión a Internet para acceder a recursos externos y garantizar la funcionalidad de las características que dependen de servicios en línea.

2.3.2.3. Interfaces de Comunicaciones.

Las interfaces de comunicaciones desempeñan un papel crucial en la interacción eficiente entre los diversos componentes del sistema GSAapp.

❖ **Comunicación Cliente-Servidor:**

- GSAapp utiliza una arquitectura cliente-servidor, donde los clientes (usuarios y administradores) se comunican con el servidor para realizar operaciones como solicitar y gestionar permisos. La comunicación debe ser segura y eficiente.

❖ **Protocolo HTTPS:**

- Todas las comunicaciones entre el cliente y el servidor deben realizarse a través del protocolo HTTPS para garantizar la seguridad de los datos transmitidos.

❖ **Formato de Datos Estandarizado:**

- Se establece un formato de datos estandarizado (por ejemplo, JSON) para la comunicación entre el cliente y el servidor, facilitando la consistencia y la interpretación precisa de la información.

❖ **Comunicación Asíncrona:**

- En determinados casos, como notificaciones y actualizaciones en tiempo real, se implementa la comunicación asíncrona para garantizar una respuesta rápida y la sincronización en tiempo real de la interfaz de usuario.
- ❖ API de Base de Datos:
 - Se establece una interfaz de programación de aplicaciones (API) para la comunicación entre la aplicación y la base de datos MySQL. La API debe gestionar las consultas y actualizaciones de manera eficiente.
- ❖ Comunicación Segura con Servicios Externos y Spring Security:
 - Se implementa un enfoque robusto que incorpora protocolos estándar de seguridad, como OAuth, para gestionar de manera efectiva los procesos de autenticación y autorización. De esta manera, se garantiza la integridad y la confidencialidad de la comunicación con servicios externos, fortaleciendo la seguridad global de la aplicación y preservando la privacidad de los datos transmitidos. La integración de Spring Security potencia esta capa de seguridad, proporcionando una gestión eficiente y coherente de la autenticación y autorización en la interacción con servicios externos.
- ❖ Integración con Sistemas Existentes:
 - GSAapp puede integrarse con sistemas existentes en el entorno laboral. Se establecen interfaces de comunicación estándar para facilitar la interoperabilidad y la transferencia de datos.

2.3.3. Obligaciones del Diseño.

2.3.3.1. Estándares Cumplidos.

GSAapp adhiere a los estándares del Consorcio World Wide Web (W3C) para HTML y CSS.

2.3.4. Atributos.

2.3.4.1. Seguridad.

Con el objetivo de **salvaguardar la integridad y confidencialidad** de la aplicación, se ha integrado **Spring Security** como un componente esencial en el marco de seguridad. Spring Security proporciona una infraestructura robusta para gestionar aspectos críticos de seguridad, como la **autenticación y autorización**. Su implementación contribuye significativamente a la protección de la aplicación contra amenazas potenciales, garantizando un entorno seguro para la gestión de permisos y la interacción de los usuarios.

Además, Spring Security facilita la configuración de **políticas de seguridad personalizadas**, asegurando el cumplimiento de los estándares y requisitos específicos de protección de datos en el contexto de la aplicación GSAapp.

2.3.4.2. Facilidades de Mantenimiento.

Entre las principales facilidades de mantenimiento se encuentran:

❖ **Modularidad del Código Fuente:**

- La estructura modular del código fuente permite realizar actualizaciones y modificaciones específicas en componentes individuales sin afectar otras partes del sistema, facilitando la implementación de mejoras y correcciones.

❖ **Documentación Exhaustiva:**

- Se proporciona documentación detallada que abarca aspectos técnicos y funcionales del sistema. Esto incluye manuales de usuario, guías de desarrollo y comentarios en el código fuente, lo que simplifica la comprensión y el mantenimiento futuro.

❖ **Seguimiento de Problemas (Issue Tracking):**

- La implementación de un sistema de seguimiento de problemas permite identificar, registrar y abordar eficientemente cualquier inconveniente o mejora necesaria. Facilita la colaboración entre desarrolladores y garantiza una respuesta rápida a posibles problemas.

❖ **Automatización de Pruebas:**

- La presencia de un conjunto robusto de pruebas automatizadas contribuye a detectar y corregir posibles errores de manera proactiva. Esto agiliza el proceso de mantenimiento al proporcionar una validación continua de la integridad del sistema.

❖ **Control de Versiones (Version Control):**

- El uso de sistemas de control de versiones, como Git, permite realizar un seguimiento preciso de las modificaciones realizadas en el código fuente. Facilita la reversión a versiones anteriores en caso de problemas y facilita la colaboración entre equipos de desarrollo.

❖ **Integración Continua:**

- La implementación de procesos de integración continua asegura que las nuevas modificaciones sean probadas y validadas automáticamente,

reduciendo el riesgo de errores y facilitando el despliegue de actualizaciones de manera consistente.

❖ **Escalabilidad:**

- La arquitectura escalable de GSAapp permite la incorporación de nuevas funcionalidades y la adaptación del sistema a cambios en los requisitos, asegurando su relevancia a lo largo del tiempo.

2.3.4.3. Portabilidad.

La portabilidad de GSAapp, entendida como la capacidad del sistema para ejecutarse eficientemente en diferentes entornos, se respalda mediante una serie de requisitos diseñados para garantizar su flexibilidad y adaptabilidad. Entre los aspectos clave relacionados con la portabilidad del sistema, se encuentran:

❖ **Compatibilidad de Navegadores:**

- GSAapp se ha diseñado para ser compatible con una amplia gama de navegadores web modernos. Esto asegura una experiencia consistente para los usuarios, independientemente del navegador que utilicen.

❖ **Requisitos Mínimos de Hardware:**

- Se establecen requisitos mínimos de hardware para garantizar que GSAapp pueda ejecutarse de manera eficiente en una variedad de dispositivos, desde estaciones de trabajo tradicionales hasta dispositivos móviles.

❖ **Escalabilidad Eficiente:**

- La arquitectura de GSAapp se ha diseñado para ser escalable, permitiendo adaptarse a cambios en la carga de trabajo y requerimientos de rendimiento sin comprometer su funcionamiento en diferentes configuraciones.

❖ **Estándares de Desarrollo Web:**

- La adhesión a estándares de desarrollo web asegura que GSAapp pueda aprovechar tecnologías y prácticas ampliamente aceptadas, mejorando su capacidad para funcionar en diferentes contextos y entornos.

3. ANÁLISIS.

3.1. Introducción.

En esta sección se exhibe el diagrama de clases, que representa la estructura estática del sistema, así como los casos de uso, que describen las interacciones entre los usuarios y el sistema. Este conjunto de representaciones gráficas ofrece una visión detallada y estructurada de la arquitectura y las funcionalidades previstas en el diseño del proyecto.

3.2. Diagrama de Clases.

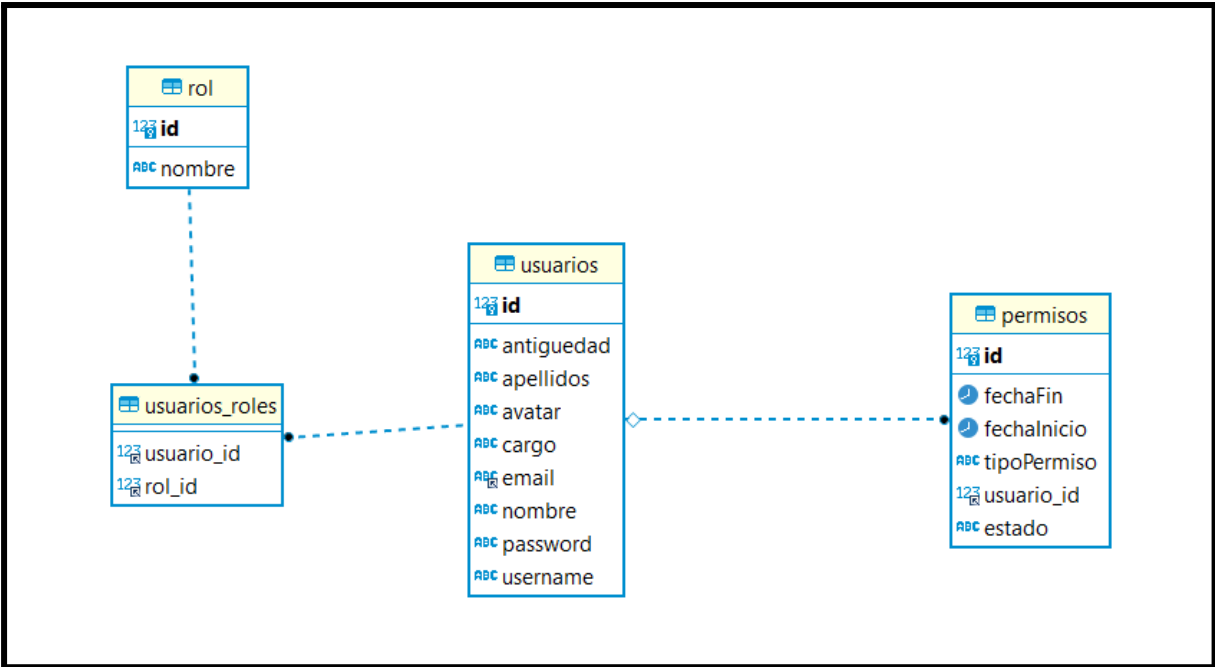


DIAGRAMA DE CLASES

3.3. Diagrama de Casos de Uso.

CASO DE USO	
Actores Principales	<ul style="list-style-type: none">• Usuario• Administrador
Precondiciones	El usuario y el administrador inician sesión en la aplicación.
Flujo Principal	
Usuario Solicita Permiso	<ul style="list-style-type: none">• El usuario inicia sesión en GSAapp.

	<ul style="list-style-type: none"> ● Accede a la interfaz de usuario. ● Visualiza su información personal y el historial de permisos. ● Selecciona la opción para solicitar un nuevo permiso. ● Es redirigido a una página con un calendario. ● Selecciona la fecha deseada y el tipo de permiso. ● Confirma la solicitud. ● El sistema registra la solicitud y deshabilita temporalmente la opción de solicitud hasta que el administrador gestione el permiso.
Administrador Gestiona Permiso	<ul style="list-style-type: none"> ● El administrador inicia sesión en GSAapp. ● Accede a la interfaz de administrador. ● Visualiza la lista de usuarios con permisos pendientes. ● Selecciona un usuario y accede a la página "Ver Asunto." ● Observa la información detallada del usuario y los permisos pendientes. ● Decide aceptar o denegar los permisos pendientes. ● Si acepta, los permisos se registran en el historial del usuario con el estado actualizado. ● Si se deniega, los permisos se eliminan. ● El sistema notifica al usuario sobre la decisión del administrador.
Flujo Alternativo	<p>Usuario Cancela Solicitud:</p> <ul style="list-style-type: none"> ● El usuario, antes de que el administrador gestione el permiso, tiene la opción de cancelar la solicitud desde su historial. ● La solicitud se elimina del sistema y se notifica al usuario sobre la cancelación.
Postcondiciones	<ul style="list-style-type: none"> ● El estado de los permisos queda registrado en el historial del usuario. ● El administrador comunica al usuario la decisión sobre la solicitud. ● La interfaz del usuario se actualiza reflejando el estado de las solicitudes y permisos gestionados.

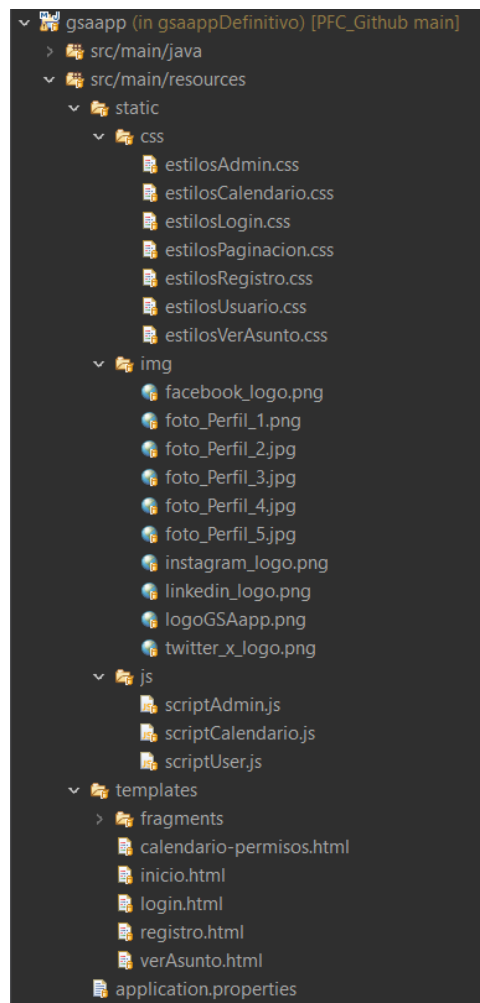
CASO DE USO

4. DISEÑO

4.1. Introducción.

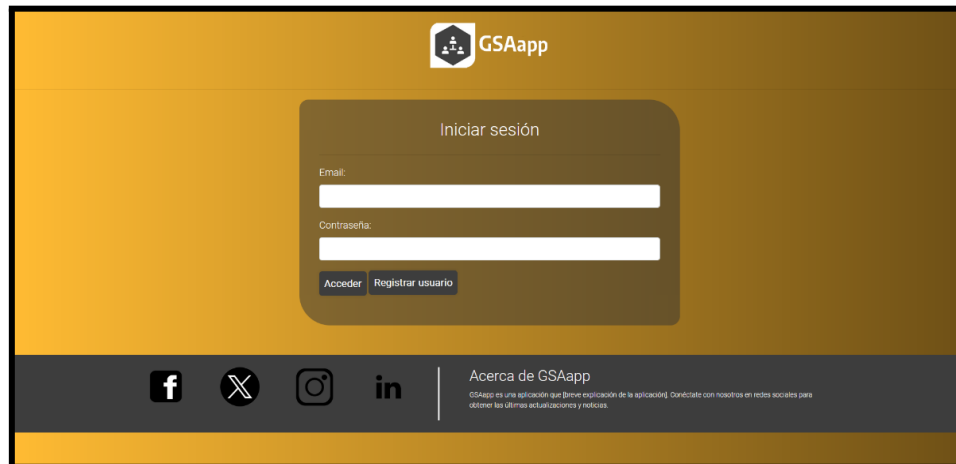
En esta sección, se presenta el diseño detallado de la estructura de la aplicación.

4.2. Capa de Presentación.



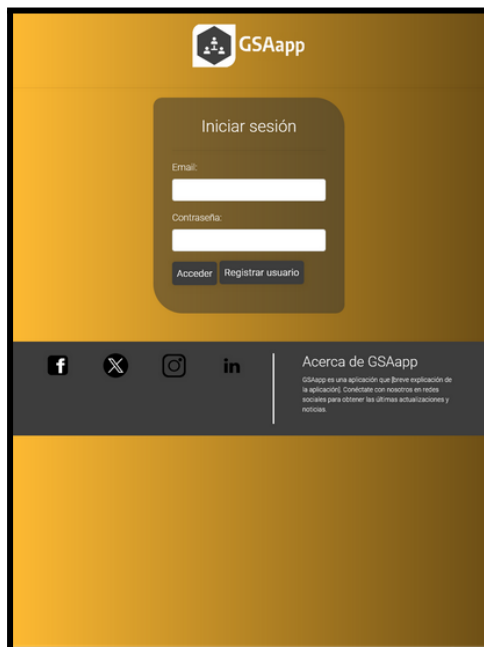
ESTRUCTURA FRONT-END

Vista inicio sesión:



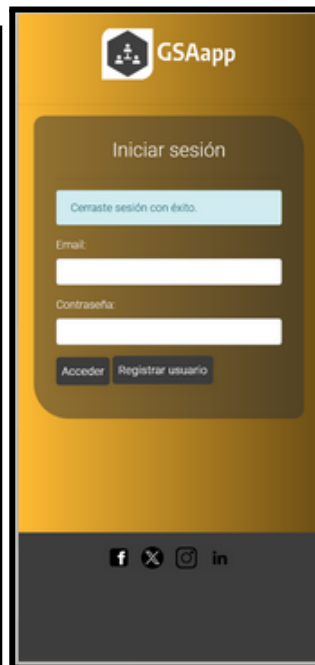
The mockup shows a desktop browser view of the GSAapp login page. The header features the GSAapp logo. The main content area is a dark brown rectangle with a lighter brown border. Inside, the text "Iniciar sesión" is centered. Below it are two input fields labeled "Email:" and "Contraseña:". At the bottom of this section are two buttons: "Acceder" and "Registrar usuario". The footer is a dark gray bar containing social media icons for Facebook, Twitter, Instagram, and LinkedIn, followed by the text "Acerca de GSAapp" and a small paragraph of text.

VISTA NAVEGADOR



This mockup shows the GSAapp login page at a resolution of 810 x 1080. The layout is similar to the desktop version but scaled to fit the resolution. The "Iniciar sesión" section is centered, and the footer is visible at the bottom.

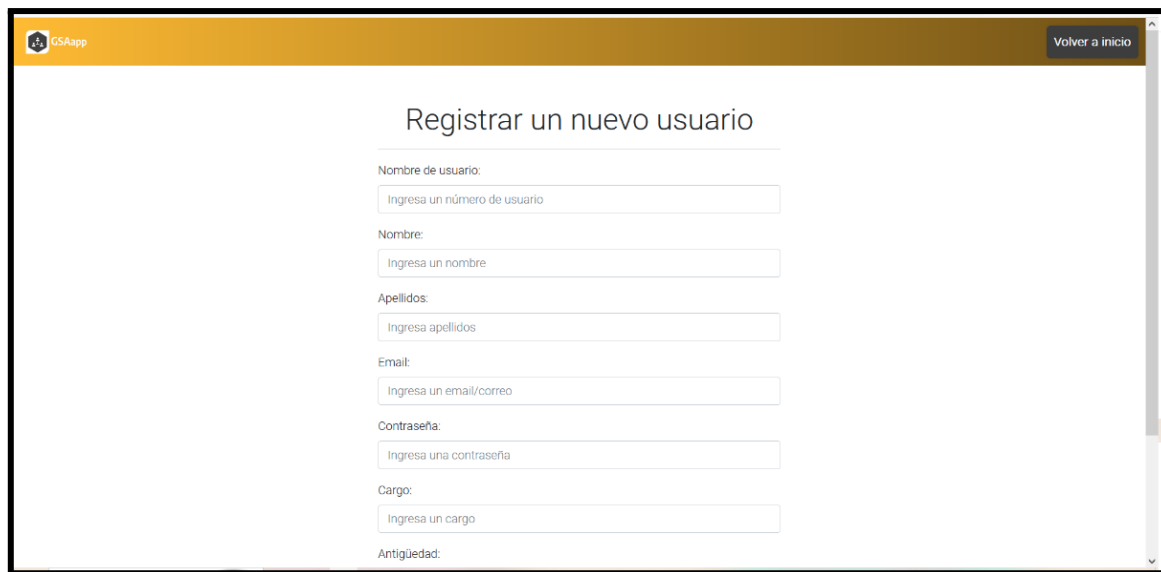
VISTA 810 x 1080



This mockup shows the GSAapp login page at a resolution of 428 x 926. The layout is scaled to fit the resolution. The "Iniciar sesión" section is centered, and the footer is visible at the bottom. A light blue message box at the top of the login section reads "Cerraste sesión con éxito."

VISTA 428 x 926

Vista de registro:



CSAapp [Volver a inicio](#)

Registrar un nuevo usuario

Nombre de usuario:

Nombre:

Apellidos:

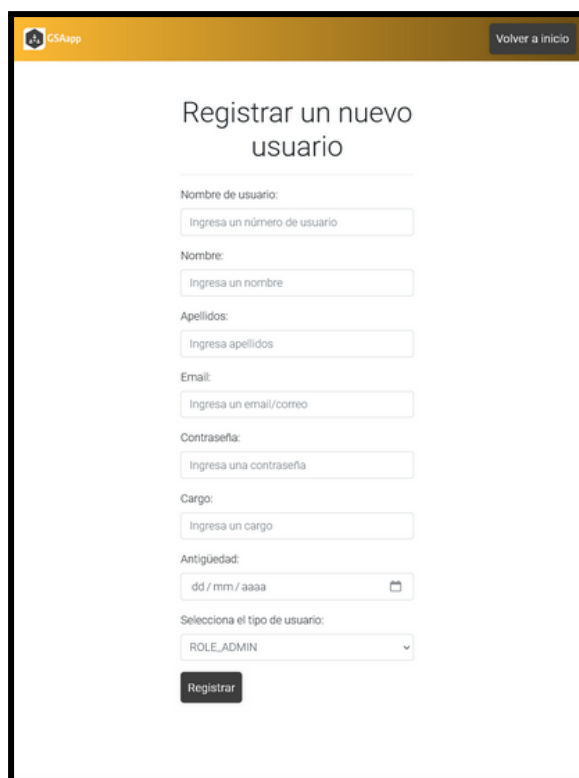
Email:

Contraseña:

Cargo:

Antigüedad:

VISTA NAVEGADOR



CSAapp [Volver a inicio](#)

Registrar un nuevo usuario

Nombre de usuario:

Nombre:

Apellidos:

Email:

Contraseña:

Cargo:

Antigüedad:

Selecciona el tipo de usuario:

[Registrar](#)

VISTA 810 x 1080



CSAapp [Volver a inicio](#)

Registrar un nuevo usuario

Nombre de usuario:

Nombre:

Apellidos:

Email:

Contraseña:

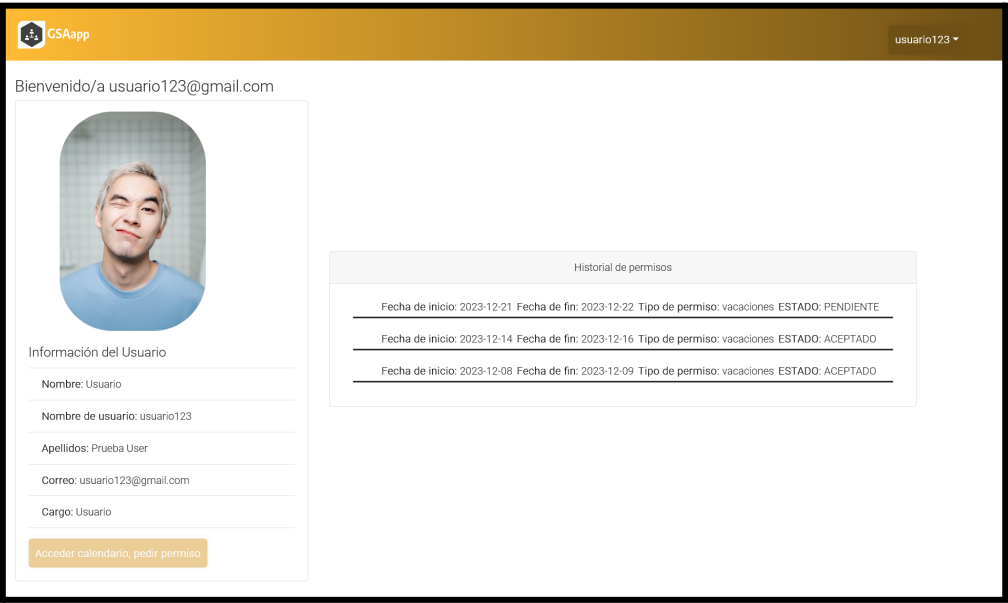
Cargo:

Antigüedad:

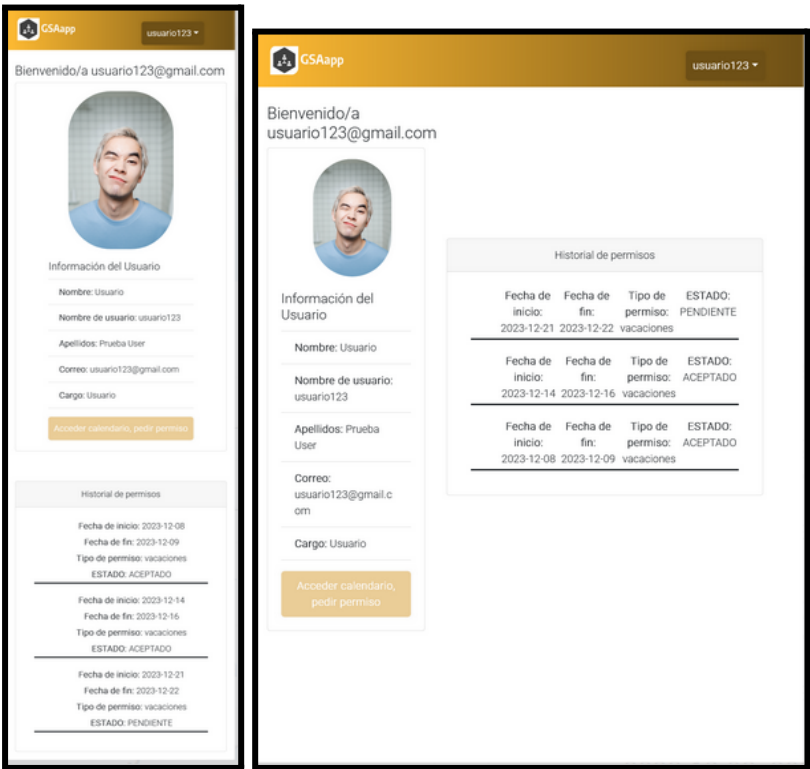
Selecciona el tipo de usuario:

VISTA 428 x 926

Vista usuario:



VISTA NAVEGADOR



VISTA 428 x 926

VISTA 810 x 1080

Vista calendario:

[Volver a inicio](#)

Año anterior

<<

>>

Próximo año

HOY

Diciembre 2023

Lun	Mar	Mié	Jue	Vie	Sáb	Dom
					9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Selecciona el tipo de permiso:

Vacaciones

Solicitar permiso

VISTA NAVEGADOR

[Volver a inicio](#)

Año anterior

<<

>>

Próximo año

HOY

Diciembre 2023

Lun	Mar	Mié	Jue	Vie	Sáb	Dom
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Selecciona el tipo de permiso:

Vacaciones

Solicitar permiso

VISTA 428 x 926

[Volver a inicio](#)

Año anterior

<<

>>

Próximo año

HOY

Diciembre 2023

Lun	Mar	Mié	Jue	Vie	Sáb	Dom
					9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7


Selecciona el tipo de permiso:

Vacaciones


Solicitar permiso

VISTA 810 x 1080

Vista administrador:

GSAApp

admin123



Información del Usuario

Nombre: Admin




Nombre de usuario: admin123

Apellidos: Prueba Admin

Correo: admin123@gmail.com


Cargo: Administrador

Pulsa [aquí](#) para registrar un nuevo usuario


Foto	Nombre de usuario	Email	Cargo	Acciones
	usuario123	usuario123@gmail.com	Usuario	Ver asunto
	juanUS456	juan123@correo.com	Arquitecto de software	Ver asunto
	joseUS888	jose888@gsaapp.es	Programador Back End	Ver asunto

Primera < 1 2 3 >

VISTA NAVEGADOR

GSAApp

admin123



Información del Usuario

Nombre: Admin




Nombre de usuario: admin123

Apellidos: Prueba Admin

Correo: admin123@gmail.com


Cargo: Administrador

Pulsa [aquí](#) para registrar un nuevo usuario


Foto	Nombre de usuario	Email	Acciones
	usuario123	usuario123@gmail.com	Ver asunto
	juanUS456	juan123@correo.com	Ver asunto
	joseUS888	jose888@gsaapp.es	Ver asunto

Primera < 1 2 3 >

VISTA 810 x 1080

GSAApp

admin123



Información del Usuario

Nombre: Admin




Nombre de usuario: admin123

Apellidos: Prueba Admin

Correo: admin123@gmail.com

Cargo: Administrador

Pulsa [aquí](#) para registrar un nuevo usuario

Foto	Nombre de usuario	Acciones
	usuario123	Ver asunto
	juanUS456	Ver asunto
	joseUS888	Ver asunto

Primera < 1 2 3 >

VISTA 428 x 926

Vista de verAsunto:

GSAapp

Detalles del Usuario: usuario123

Nombre: Usuario

Apellidos: Prueba User

Email: usuario123@gmail.com

Cargo: Usuario

Antigüedad: 2023-09-06

Detalles de los Permisos

Los permisos aceptados no pueden ser denegados

ESTADO: ACEPTADO

Fecha de inicio: 2023-12-08

Fecha de fin: 2023-12-09

Tipo de permiso: vacaciones

Aceptar

Denegar

ESTADO: ACEPTADO

Fecha de inicio: 2023-12-14

Fecha de fin: 2023-12-16

Tipo de permiso: vacaciones

Aceptar

Denegar

Volver a inicio

VISTA NAVEGADOR

GSAapp

Detalles del Usuario: usuario123

Nombre: Usuario

Apellidos: Prueba User

Email: usuario123@gmail.com

Cargo: Usuario

Antigüedad: 2023-09-06

Detalles de los Permisos

Los permisos aceptados no pueden ser denegados

ESTADO: ACEPTADO

Fecha de inicio: 2023-12-14

Fecha de fin: 2023-12-16

Tipo de permiso: vacaciones

Aceptar

Denegar

ESTADO: ACEPTADO

Fecha de inicio: 2023-12-08

Fecha de fin: 2023-12-09

Tipo de permiso: vacaciones

Aceptar

Denegar

Volver a inicio

VISTA 810 x 1080

GSAapp

Detalles del Usuario: usuario123

Nombre: Usuario

Apellidos: Prueba User

Email: usuario123@gmail.com

Cargo: Usuario

Antigüedad: 2023-09-06

Detalles de los Permisos

Los permisos aceptados no pueden ser denegados

ESTADO: ACEPTADO

Fecha de inicio: 2023-12-14

Fecha de fin: 2023-12-16

Tipo de permiso: vacaciones

Aceptar

Denegar

ESTADO: ACEPTADO

Fecha de inicio: 2023-12-08

Fecha de fin: 2023-12-09

Tipo de permiso: vacaciones

Aceptar

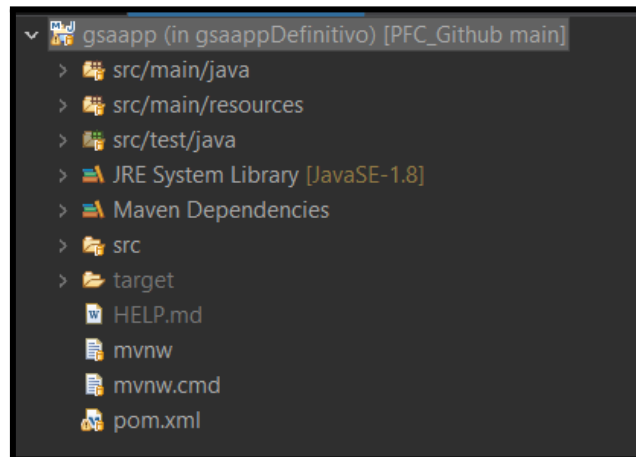
Denegar

Volver a inicio

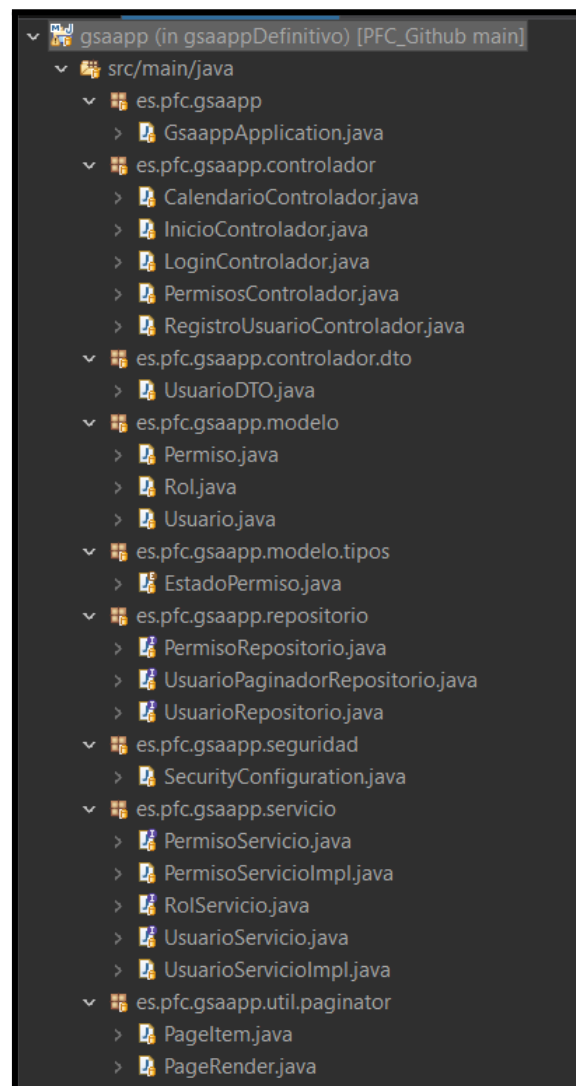
VISTA 428 x 926

4.3. Capa de Negocio o Lógica de la Aplicación.

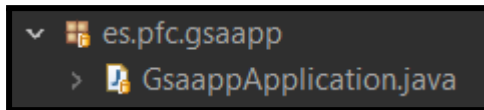
Estructura de carpetas con la lógica de la app.



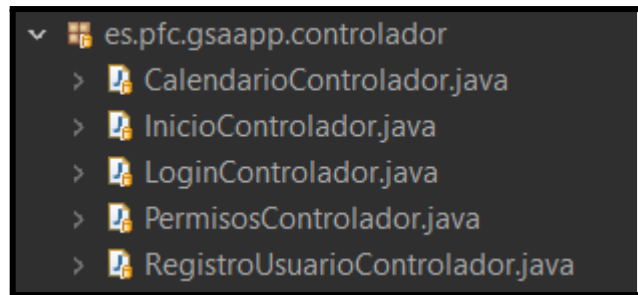
ESTRUCTURA DE CARPETAS LÓGICA



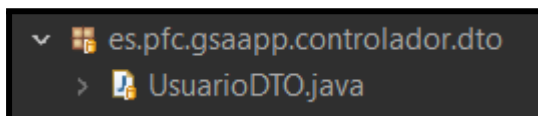
ESTRUCTURA BACK-END



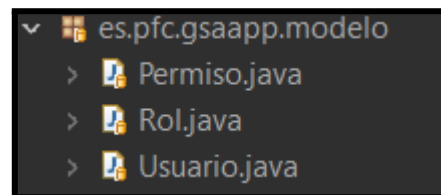
CLASE INICIADORA DE APP



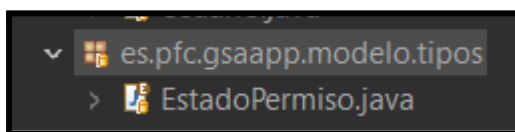
ESTRUCTURA DE CLASES CONTROLADOR



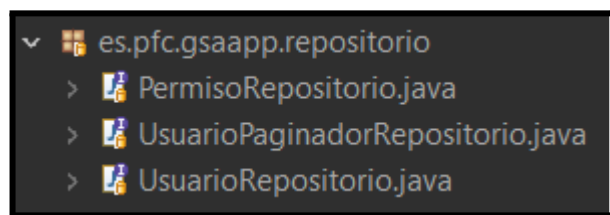
ESTRUCTURA CLASES DTO



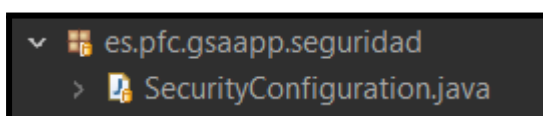
ESTRUCTURA CLASES MODELO



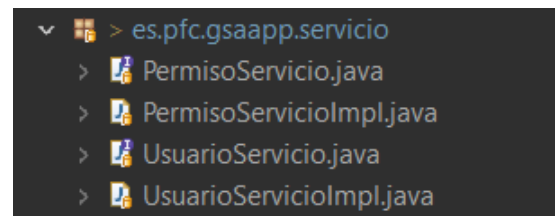
ESTRUCTURA INTERFACESTIPO



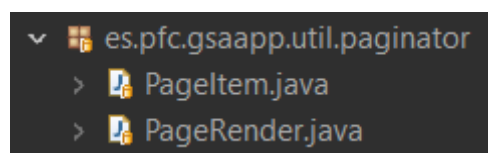
ESTRUCTURA INTERFACES REPOSITORIO



ESTRUCTURA CLASES SEGURIDAD



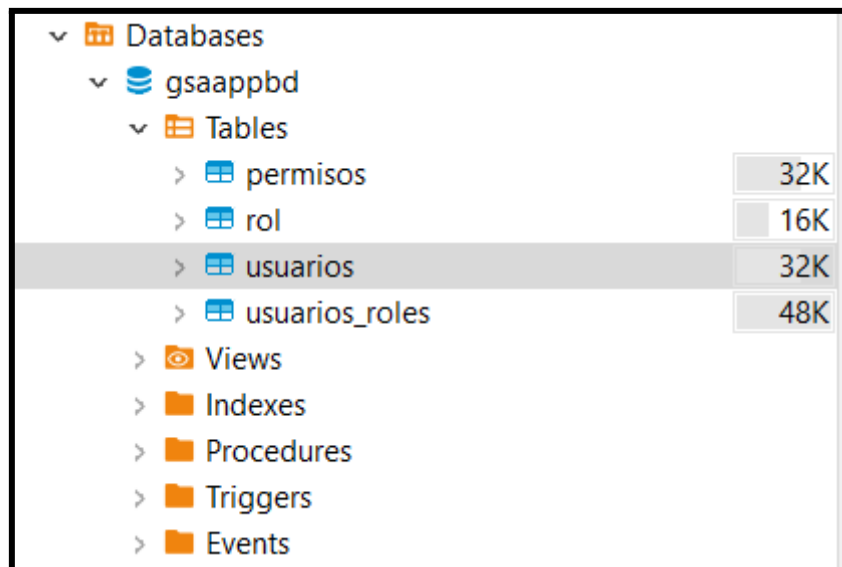
ESTRUCTURA CLASES / INTERFACES SERVICIO



ESTRUCTURA CLASES PAGINADOR

4.4. Capa de Persistencia o Datos.

Estructura de la capa de persistencia o datos:



ESTRUCTURA DE CARPETAS BASE DE DATOS

usuarios								
	id	antiguedad	apellidos	avatar	cargo	email	nombre	password
1	1	2023-09-06	Prueba User	foto_Perfil_2.jpg	Usuario	usuario123@gmail.com	Usuario	\$2a\$10\$ULw9gZbvP
2	2	2023-07-12	Prueba Admin	foto_Perfil_3.jpg	Administrador	admin123@gmail.com	Admin	\$2a\$10\$0AmQxYfxbwr
3	3	2023-09-13	Rodriguez Rodriguez	foto_Perfil_3.jpg	Arquitecto de software	juan123@correo.com	Juan	\$2a\$10\$3FFfEXz5Kme3
4	4	2010-07-06	Gomez Alvarez	foto_Perfil_4.jpg	Administrador	mariaga465@gsaapp.com	Maria	\$2a\$10\$weEgKG.w5xG
5	5	2016-10-20	Hernandez Ferreiro	foto_Perfil_5.jpg	Programador Back End	jose888@gsaapp.es	Jose	\$2a\$10\$poPe7rHE0M1
6	6	2015-03-05	Gonzalez Garcia	foto_Perfil_1.png	Programador Front End	lucia999@gsaapp.com	Lucia	\$2a\$10\$4t7YQTeCfoB
7	7	2020-09-07	Gutierrez Iglesias	foto_Perfil_3.jpg	Programador Back End	fernando4RTE@gsaapp.com	Fernando	\$2a\$10\$9LschMFAXXX
8	8	2023-02-14	Fernandez Fernandez	foto_Perfil_4.jpg	Programador Front End	alejandro564@gsaapp.com	Alejandro	\$2a\$10\$6417KSboxAD
9	9	2013-01-15	Suarez Montes	foto_Perfil_3.jpg	Programador Full Stack	rodrigo333@gsaapp.com	Rodrigo	\$2a\$10\$y2FFFR67W.0C

ESTRUCTURA DE LA TABLA DE USUARIOS

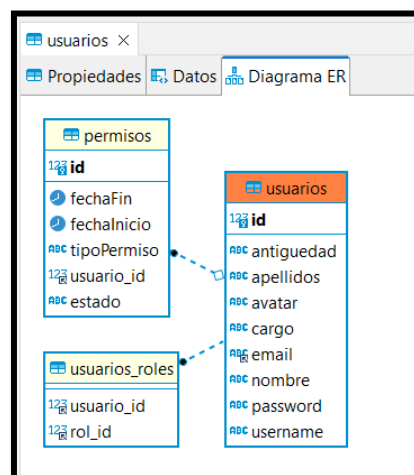


DIAGRAMA ER DE USUARIOS

rol ×

Propiedades Datos Diagrama ER

rol Enter a SQL expression to filter results

	123 id	ABC nombre
1	1	ROLE_USER
2	2	ROLE_ADMIN
3	3	ROLE_USER
4	4	ROLE_ADMIN
5	5	ROLE_USER
6	6	ROLE_USER
7	7	ROLE_USER
8	8	ROLE_USER
9	9	ROLE_USER

ESTRUCTURA DE LA TABLA DE ROL

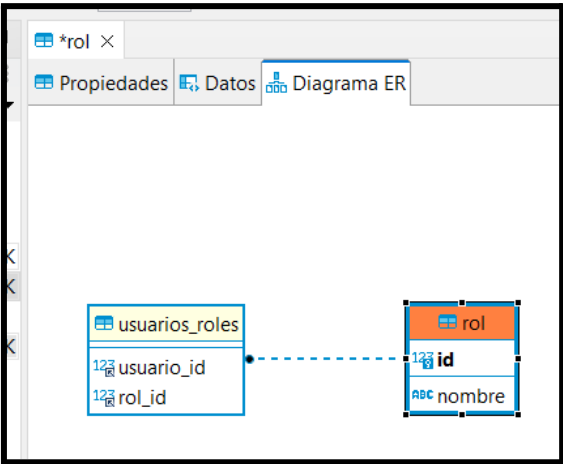


DIAGRAMA ER DE ROL

usuarios_roles ×

Propiedades Datos Diagrama ER

usuarios_roles Enter a SQL expression to filter results

	123 usuario_id	123 rol_id
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9

ESTRUCTURA DE LA TABLA DE USUARIOS_ROL

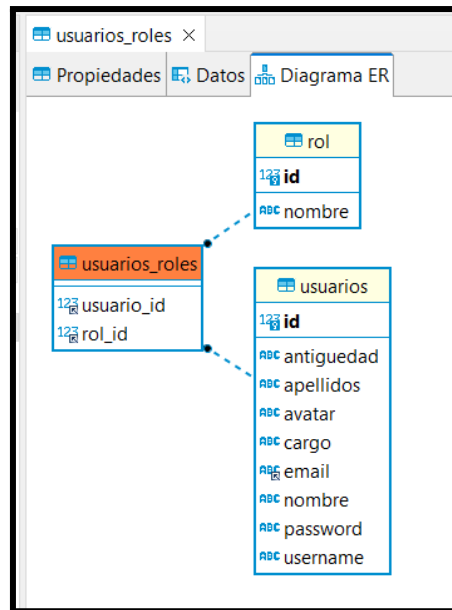


DIAGRAMA ER DE USUARIOS_ROL

permisos							
Propiedades Datos Diagrama ER							
permisos Enter a SQL expression to filter results (use Ctrl+Space)							
	123 id	🕒 fechaFin	🕒 fechaInicio	ABC tipoPermiso	123 usuario_id	ABC estado	
1	2	2023-12-16	2023-12-14	vacaciones	1	ACEPTADO	
2	3	2023-12-09	2023-12-08	vacaciones	1	ACEPTADO	
3	4	2023-12-09	2023-12-08	otro	3	PENDIENTE	

ESTRUCTURA DE LA TABLA DE PERMISOS

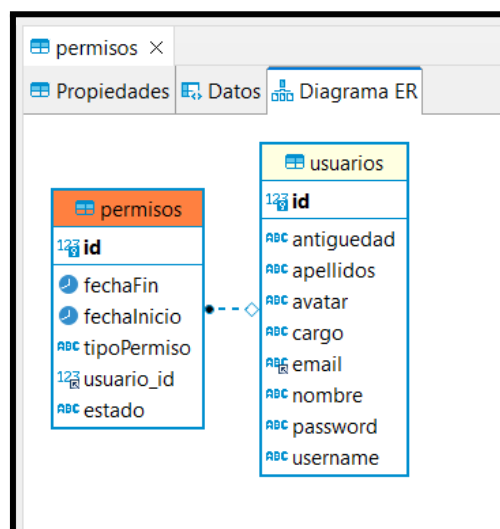


DIAGRAMA ER DE PERMISOS

5. IMPLEMENTACIÓN

5.1. Tecnologías utilizadas en el desarrollo del proyecto

Para la implementación del proyecto, se optó por el uso de Java en el lado del servidor, específicamente utilizando el framework Spring Boot para el desarrollo del back-end. Java es un lenguaje de programación versátil y robusto que proporciona una sólida base para la lógica empresarial de la aplicación.

En el lado del cliente, se utilizó Thymeleaf para la generación de plantillas HTML. Thymeleaf es un motor de plantillas que se integra fácilmente con aplicaciones basadas en Spring y facilita la creación de vistas dinámicas.

En el ámbito del desarrollo web, se emplearon varios lenguajes y tecnologías. HTML (HyperText Markup Language) se utilizó para la estructura básica de las páginas web, mientras que CSS (Cascading Style Sheets) se encargó de definir el diseño y estilo visual. JavaScript y jQuery se implementaron para mejorar la interactividad y dinamismo de la interfaz de usuario, permitiendo una experiencia más fluida.

La gestión de datos se llevó a cabo mediante MySQL, un sistema de gestión de bases de datos relacional. MySQL proporciona un entorno confiable y eficiente para almacenar y recuperar datos de manera estructurada.

Esta selección de tecnologías se realizó de manera estratégica, aprovechando las fortalezas individuales de cada lenguaje y herramienta para construir una aplicación integral y eficiente. La combinación de Java, Thymeleaf, JavaScript, jQuery, CSS, HTML y MySQL garantiza un desarrollo sólido y coherente en todas las capas de la aplicación.

5.2. Descripción del Proyecto

En esta sección, se detalla el funcionamiento de la lógica del proyecto.

5.2.1. Capa de Presentación.

Aquí se presenta el código correspondiente a la parte cliente de la aplicación.

→ **login html:**

Página web que forma parte de un sistema de inicio de sesión y registro para la aplicación GSAapp

```

<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">

<head>
  <!-- Configuración del encabezado -->
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title th:text="'GSAapp - ' + ${titulo}">GSAapp - Iniciar sesión</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJISAWiGgFAW/dAiS6JX
m"
  crossorigin="anonymous">
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KChRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5Kk
N"
  crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
  crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYI"
  crossorigin="anonymous"></script>
  <link rel="stylesheet" href="/css/estilosLogin.css">
</head>

<body>
  <!-- Contenido principal -->
  <nav class="navbar navbar-light justify-content-between top-bar">
    
  </nav>

  <hr>

  <div class="formulario container-fluid justify-content-center">
    <h3 class="text-center mb-4">Iniciar sesión</h3>
    <hr>

    <!-- Mensajes de error/success -->
    <div th:if="{param.error}" class="alert alert-danger">
      <span>Email o contraseña incorrectos.</span>
    </div>
    <div th:if="{param.logout}" class="alert alert-info">

```

```

        <span>Cerraste sesión con éxito.</span>
    </div>

    <!-- Formulario de inicio de sesión -->
    <form th:action="@{/login}" method="post">
        <div class="form-group">
            <label for="username">Email:</label>
            <input class="form-control" type="text" id="username" name="username" size="30">
        </div>

        <div class="form-group">
            <label for="password">Contraseña:</label>
            <input class="form-control" type="password" id="password" name="password" size="30">
        </div>

        <div class="form-group">
            <input type="submit" class="btn btn-primary" value="Acceder">
            <a th:href="@{/registro}" class="btn btn-secondary">Registrar usuario</a>
        </div>

        <!-- Token CSRF para seguridad -->
        <input type="hidden" th:name="{$_csrf.parameterName}" th:value="{$_csrf.token}" />
    </form>
</div>

<!-- Pie de página -->
<footer class="text-white mt-5">
    <div class="container footerInterior">
        <!-- Iconos de redes sociales -->
        <div class="footer-icons">
            <a href="https://www.facebook.com" class="text-white mr-3">
                
            </a>
            <!-- Otros enlaces a redes sociales -->
        </div>
        <!-- Información sobre la aplicación -->
        <div class="infoApp">
            <h4>Acerca de GSAApp</h4>
            <p>GSAApp es una aplicación que [breve explicación de la aplicación]. Conéctate con
nosotros en redes
                sociales para obtener las últimas actualizaciones y noticias.</p>
        </div>
    </div>
</footer>
</body>

</html>

```

Es una página web que utiliza Thymeleaf para la generación dinámica de contenido y Bootstrap para el diseño y los estilos. La página incluye un formulario de inicio de sesión, mensajes de error o éxito, un encabezado con el logo de GSAapp, y un pie de página con enlaces a redes sociales e información sobre la aplicación. Además, se implementa seguridad mediante un token CSRF en el formulario de inicio de sesión para prevenir ataques CSRF.

→ registro.html:

Representa una página web de registro de usuarios.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
  <!-- Configuración del encabezado -->
  <meta charset="utf-8">
  <title>Registro de usuarios</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css">
  <script src="https://cdn.jsdelivr.net/npm/jquery@3.7.1/dist/jquery.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"></script>
  <!-- CSS personalizado -->
  <link rel="stylesheet" href="/css/estilosRegistro.css">
</head>

<body>
  <!-- Contenido principal -->
  <nav class="navbar navbar-light justify-content-between top-bar">
    
    <a th:href="@{/inicio}"><button class="botonVolverInicio">Volver a inicio</button></a>
  </nav>

  <br>
  <br>

  <div class="container">
    <div class="row">
      <div class="col-md-6 mx-auto">
        <!-- Mensaje de éxito -->
        <div th:if="${param.exito}">
```



```

        <div class="alert alert-info">Se registró con éxito</div>
    </div>

    <h1 class="text-center">Registrar un nuevo usuario</h1>

    <hr>

    <!-- Formulario de registro de usuarios -->
    <form th:action="@{/registro}" method="post" th:object="${usuario}">
        <!-- Campos del formulario -->
        <!-- ... -->
        <div class="form-group">
            <button type="submit">Registrar</button>
        </div>
    </form>
</div>
</div>
</div>

<!-- Scripts de jQuery, Popper.js y Bootstrap -->
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCKRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5Kk
N"
        crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
        crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYI"
        crossorigin="anonymous"></script>
</body>

</html>

```

Es una página de registro de usuarios que utiliza Thymeleaf para la generación dinámica de contenido y Bootstrap para el diseño y los estilos. La página incluye un formulario de registro con campos como nombre de usuario, nombre, apellidos, email, contraseña, cargo, antigüedad y rol. También tiene un mensaje de éxito después del registro y un enlace para volver a la página de inicio.

→ **inicio.html:**

Página web que puede mostrar diferentes vistas según el rol del usuario (USER o ADMIN).

```

<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">

<head>
  <!-- Configuración del encabezado -->
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title th:text="'GSAapp - ' + ${titulo}"></title>
  <!-- CSS de Bootstrap -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJISAwIGgFAW/dAiS6JX
m"
    crossorigin="anonymous">
  <!-- Scripts de Bootstrap y jQuery -->
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KChRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5Kk
N"
    crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
    crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYI"
    crossorigin="anonymous"></script>
  <!-- CSS personalizado y scripts -->
  <link rel="stylesheet" href="/css/estilosUsuario.css" sec:authorize="hasRole('USER')">
  <script src="js/scriptUser.js" sec:authorize="hasRole('USER')"></script>
  <link rel="stylesheet" href="/css/estilosAdmin.css" sec:authorize="hasRole('ADMIN')">
  <link rel="stylesheet" href="/css/estilosPaginacion.css">
</head>

<body>
  <!-- Barra de navegación -->
  <nav class="navbar navbar-light justify-content-between top-bar">
    
    <!-- Menú desplegable con el nombre de usuario y opción de cerrar sesión -->
    <div class="dropdown">
      <button type="button" class="dropdown-toggle" data-toggle="dropdown"
th:text="${usuario.username}">
      </button>
      <div class="dropdown-menu">

```

```

        <a class="dropdown-item" th:href="@{/logout}" th:text="'Cerrar sesión'"
sec:authorize="isAuthenticated()"></a>
    </div>
</div>
</nav>

<!-- Contenido principal -->
<div class="container-fluid" sec:authorize="hasRole('USER')">
    <!-- Vista para usuarios -->
    <div class="contenedorPrincipal">
        <!-- Información del usuario y acciones -->
        <div class="espacioUsuario py-4">
            <!-- ... -->
        </div>

        <!-- Historial de permisos -->
        <div class="card text-center">
            <div class="card-header">Historial de permisos</div>
            <div class="card-body">
                <!-- Detalles de permisos del usuario -->
                <div class="container-fluid">
                    <ul class="detallesPermisos" th:each="permiso : ${permisos}">
                        <!-- Detalles de cada permiso -->
                        <li>
                            <strong>Fecha de inicio: </strong>
                            <span th:text="${permiso.fechaInicio}"></span>
                        </li>
                        <!-- Otros detalles... -->
                    </ul>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Vista para administradores -->
<div class="espacioAdmin container-fluid" sec:authorize="hasRole('ADMIN')">
    <!-- ... -->
</div>
</body>

</html>

```

Este código HTML utiliza Thymeleaf y Spring Security para gestionar las vistas según el rol del usuario. Además, incorpora Bootstrap para estilos y scripts, y jQuery para funcionalidades adicionales. La página tiene una barra de navegación con un menú desplegable que muestra el nombre de usuario y la opción de cerrar sesión. El contenido

principal varía dependiendo del rol del usuario entre una vista para usuarios y otra para administradores.

→ **calendario-permisos.html:**

```
<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">

<head>
  <!-- Configuración del encabezado -->
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- CSS personalizado -->
  <link rel="stylesheet" href="/css/estilosCalendario.css">

  <!-- Referencias a bibliotecas externas -->
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/fullcalendar/3.10.0/fullcalendar.min.css"/>
  <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.24.0/moment.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/fullcalendar/3.10.0/fullcalendar.min.js"></script>

  <!-- Biblioteca Modernizr -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.min.js"></script>

  <!-- JS personalizado -->
  <script src="js/scriptCalendario.js"></script>

  <!-- Configuración de caracteres y título de la página -->
  <meta charset="UTF-8">
  <title>GSAApp - Calendario</title>
</head>

<body>
  <!-- Barra de navegación -->
  <nav class="navbar navbar-light justify-content-between top-bar">
    
    <a th:href="@{/inicio}"><button class="botonVolverInicio">Volver a inicio</button></a>
  </nav>

  <!-- Mensaje para usuarios con el rol ADMIN -->
  <h1 sec:authorize="hasRole('ADMIN')">Solo tienen acceso al calendario los usuarios de nivel
USUARIO.</h1>

  <!-- Contenido principal para usuarios con el rol USER -->
```

```

<div sec:authorize="hasRole('USER')">
  <!-- Calendario -->
  <div id="calendario" class="container-fluid"></div>

  <!-- Formulario para seleccionar permisos -->
  <form th:action="@{/guardar-permiso}" th:object="{permiso}" method="post">
    <!-- Campos del formulario -->

    <!-- Campos ocultos para fechaInicio, fechaFin y ID del usuario -->
    <input type="hidden" th:field="**{fechaInicio}" id="fechaInicio" th:format="yyyy-MM-dd" />
    <input type="hidden" th:field="**{fechaFin}" id="fechaFin" th:format="yyyy-MM-dd" />
    <input type="hidden" th:field="**{usuario.id}" id="userId" />

    <!-- Mensajes de error -->
    <div th:if="{#fields.hasErrors('fechaInicio')}}" th:errors="**{fechaInicio}">Fecha de inicio
requerida</div>
    <div th:if="{#fields.hasErrors('fechaFin')}}" th:errors="**{fechaFin}">Fecha de fin
requerida</div>
    <div th:if="{#fields.hasErrors('tipoPermiso')}}" th:errors="**{tipoPermiso}">Tipo de permiso
requerido</div>

    <!-- Selección del tipo de permiso -->
    <label for="tipoPermiso">Selecciona el tipo de permiso:</label>
    <select id="tipoPermiso" name="tipoPermiso" th:field="**{tipoPermiso}">
      <option value="vacaciones">Vacaciones</option>
      <option value="enfermedad">Enfermedad</option>
      <option value="otro">Otro Permiso</option>
    </select>

    <!-- Botón para enviar el formulario -->
    <button type="submit">Solicitar permiso</button>
  </form>
</div>
</body>

</html>

```

Este código HTML define una página web que muestra un calendario y un formulario para que los usuarios soliciten permisos. La página también incluye un mensaje que indica que solo las cuentas de nivel USUARIO tienen acceso al calendario. Además, se incluyen referencias a bibliotecas externas como FullCalendar y jQuery para mejorar la funcionalidad de la página.

→ **verAsunto.html:**

```

<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">

<head>
  <!-- Configuración del encabezado -->
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <!-- Título de la página -->
  <title th:text="GSAapp - Detalles de ' + ${usuario.username}"></title>

  <!-- CSS -->
  <link rel="stylesheet" href="./css/estilosVerAsunto.css">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJISAwIGgFAW/dAiS6JX
m" crossorigin="anonymous">

  <!-- Estilos en línea -->
  <style>
    /* Estilos generales */
    body {
      font-family: 'Roboto', sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f8f9fa; /* Color de fondo */
    }

    .container-fluid {
      margin: 0 auto;
      padding: 20px;
      background-color: #ffffff; /* Color de fondo del contenido */
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    .infoPermiso {
      display: grid;
      grid-template-columns: auto auto auto auto;
      grid-template-rows: auto;
    }

    div {
      margin: 3rem;
    }

    /* Barra superior */
    .top-bar {

```

```

display: flex;
justify-content: space-between;
align-items: center;
background: linear-gradient(to right, #FFBB33, #6e5016);
color: white;
padding: 10px;
}

.top-bar img {
width: 6rem;
}

.dropdown-toggle {
background-color: #6e5016;
border: none;
color: #fff;
border-radius: 5px;
padding: 10px;
font-size: 16px;
font-weight: bold;
box-shadow: 0px 0px 5px rgba(0, 0, 0, 0.1);
transition: background-color 0.2s ease-in-out;
margin-right: 3rem;
}

.dropdown-toggle:hover {
background-color: #e4c07c;
}

h2 {
color: #333;
margin-top: 20px; /* Espaciado superior para el título principal */
}

/* Detalles del Usuario */
p {
margin: 10px 0;
color: #555;
}

/* Detalles de los Permisos */
ul {
list-style: none;
padding: 0;
}

li {

```

```

padding: 15px 0;
border-bottom: 1px solid #ddd;
}

.botonos {
padding: 15px 0;
border-bottom: none;
}

/* Botón de Volver a Inicio */
button {
background-color: #3d3d3d;
color: #fff;
padding: 10px 20px;
border: none;
border-radius: 5px;
cursor: pointer;
margin-top: 20px; /* Espaciado superior para el botón */
}

button:hover {
background-color: #696969;
}

@media only screen and (max-width: 815px) {
.infoPermiso {
display: grid;
grid-template-columns: auto auto;
grid-template-rows: auto auto;
}
}

@media only screen and (max-width: 475px) {
.infoPermiso {
display: grid;
grid-template-columns: auto auto;
grid-template-rows: auto auto;
}

.botonos {
display: grid;
grid-template-columns: auto;
grid-template-rows: auto auto;
}
}
</style>

```



```

</head>

<body>
  <!-- Contenido según el rol del usuario -->
  <div sec:authorize="hasRole('USER')">
    <h1>Acceso solo a ADMINISTRADORES</h1>
    <a th:href="@{/inicio}"><button class="botonVolverInicio">Volver a inicio</button></a>
  </div>

  <!-- Contenido para usuarios con el rol ADMIN -->
  <div class="container-fluid" sec:authorize="hasRole('ADMIN')">
    <!-- Barra de navegación -->
    <nav class="navbar navbar-light justify-content-between top-bar">
      <h5>GSAApp</h5>
    </nav>

    <!-- Detalles del Usuario y Permisos -->
    <div class="infoUsuarioPermiso">
      <!-- Detalles del Usuario -->
      <h2 th:text="Detalles del Usuario: ' + ${usuario.username}"></h2>
      <hr>
      <div class="infoUsuario container-fluid">
        <p th:text="Nombre: ' + ${usuario.nombre}"></p>
        <p th:text="Apellidos: ' + ${usuario.apellidos}"></p>
        <p th:text="Email: ' + ${usuario.email}"></p>
        <p th:text="Cargo: ' + ${usuario.cargo}"></p>
        <p th:text="Antigüedad: ' + ${usuario.antigüedad}"></p>
      </div>

      <!-- Detalles de los Permisos -->
      <h2>Detalles de los Permisos</h2>
      <h5>Los permisos aceptados no pueden ser denegados</h5>
      <hr>
      <div class="container-fluid">
        <ul class="infoPermiso" th:each="permiso : ${permisos}">
          <li><p th:text="ESTADO: ' + ${permiso.estado}"></p></li>
          <li><p>Fecha de inicio: <span th:text="${permiso.fechaInicio}"></span></p></li>
          <li><p>Fecha de fin: <span th:text="${permiso.fechaFin}"></span></p></li>
          <li><p>Tipo de permiso: <span th:text="${permiso.tipoPermiso}"></span></p></li>
          <li class="botones">
            <!-- Formulario para Aceptar -->
            <form th:action="@{/inicio/aceptarPermisos}" method="post" style="display: inline;">
              <input type="hidden" name="usuariold" th:value="${usuario.id}" />
              <input type="hidden" name="accion" value="ACEPTAR" />
              <button class="btn btn-success" type="submit"
onclick="mostrarAlertAceptar()">Aceptar</button>
            </form>
          </li>
        </ul>
      </div>
    </div>
  </div>

```

```

        <!-- Formulario para Denegar -->
        <form th:action="@{/inicio/denegarPermisos}" method="post" style="display: inline;">
            <input type="hidden" name="usuariold" th:value="{usuario.id}" />
            <input type="hidden" name="accion" value="DENEGAR" />
            <button class="btn btn-danger" type="submit"
onclick="mostrarAlertDenegar()">Denegar</button>
        </form>
    </li>
</ul>
</div>
</div>
<a th:href="@{/inicio}"><button class="">Volver a inicio</button></a>
</div>

<!-- Scripts de Bootstrap y jQuery -->
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5Kk
N" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYI"
crossorigin="anonymous"></script>
</body>

</html>

```

Este código HTML define una página web que muestra detalles de un usuario y sus permisos, permitiendo a los administradores aceptar o denegar los permisos. La página también tiene estilos personalizados y utiliza Bootstrap para mejorar el diseño y la interactividad. Además, se incluyen scripts de jQuery y Bootstrap para funcionalidades adicionales.

→ **pagination.html:**

```

<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">
<head>
    <!-- Configuración del encabezado -->
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pagination Fragment</title>
</head>
<body>

```

```

<!-- Definición del fragmento de paginación -->
<div th:fragment="pagination">
    <ul class="pagination">
        <!-- Primer ítem de la paginación -->
        <li class="page-item" th:class="${page.first? 'page-item disabled': 'page-item'}">
            <span class="page-link" th:if="${page.first}">Primera</span>
            <a class="page-link" th:if="${not page.first}" th:href="@{${page.url}(page = 0)}">Primera</a>
        </li>

        <!-- Ítem anterior -->
        <li class="page-item" th:class="${not page.hasPrevious? 'page-item disabled': 'page-item'}">
            <span class="page-link" th:if="${not page.hasPrevious}">&laquo;</span>
            <a class="page-link" th:if="${page.hasPrevious}" th:href="@{${page.url}(page =
${page.paginaActual - 2})}">&laquo;</a>
        </li>

        <!-- Páginas individuales -->
        <li class="page-item" th:each="item : ${page.paginas}" th:class="${item.actual? 'page-item
active': 'page-item'}">
            <span class="page-link" th:if="${item.actual}" th:text="${item.numero}"></span>
            <a class="page-link" th:if="${not item.actual}" th:href="@{${page.url}(page=${item.numero -
1})}" th:text="${item.numero}">anterior</a>
        </li>

        <!-- Ítem siguiente -->
        <li class="page-item" th:class="${not page.hasNext? 'page-item disabled': 'page-item'}">
            <span class="page-link" th:if="${not page.hasNext}">&raquo;</span>
            <a class="page-link" th:if="${page.hasNext}" th:href="@{${page.url}(page =
${page.paginaActual})}">&raquo;</a>
        </li>

        <!-- Último ítem de la paginación -->
        <li class="page-item" th:class="${page.last? 'page-item disabled': 'page-item'}">
            <span class="page-link" th:if="${page.last}">Último</span>
            <a class="page-link" th:if="${not page.last and totalPaginas != null}"
th:href="@{${page.url}(page = ${totalPaginas - 1})}">Último</a>
        </li>
    </ul>
</div>

</body>
</html>

```

Este código utiliza Thymeleaf para aplicar lógica condicional y enlaces dinámicos en la generación de los elementos de paginación. El fragmento proporciona enlaces para la primera página, página anterior, páginas individuales, página siguiente y última página.

Además, utiliza clases de Bootstrap para estilizar la paginación. Este fragmento se puede incorporar en otras plantillas Thymeleaf donde se necesite un control de paginación.

→ **scriptCalendario.js:**

```
// Espera a que el documento HTML se haya cargado completamente
$(document).ready(function () {
    // Variables para almacenar las fechas seleccionadas en el calendario
    var fechaInicioSeleccionada;
    var fechaFinSeleccionada;

    // Inicializa el calendario utilizando el plugin FullCalendar
    $('#calendario').fullCalendar({
        // Configuración de la barra de encabezado del calendario
        header: {
            left: 'prevYear,prev,next,nextYear today', // Botones de navegación
            center: 'title', // Título del calendario en el centro
            right: '' // Sección de encabezado derecha vacía
        },
        // Permite la selección de fechas en el calendario
        selectable: true,
        // Función que se ejecuta al seleccionar un rango de fechas
        select: function (start, end, jsEvent, view) {
            // Si es un rango de un solo día, ajusta la fecha de fin para que sea igual a la fecha de inicio
            fechaFinSeleccionada = start.isSame(end, 'day') ? moment(start).endOf('day') : end;

            // Actualiza los campos de inicio y fin en el formulario
            $('#fechaInicio').val(start.format());
            $('#fechaFin').val(fechaFinSeleccionada.format());

            // Muestra el formulario
            $('#form').show();
        },
        // Función que se ejecuta al hacer clic en un día del calendario
        dayClick: function (date, jsEvent, view) {
            // Establece tanto la fecha de inicio como la de fin al hacer clic en un solo día
            fechaInicioSeleccionada = date.format();
            fechaFinSeleccionada = date.format();

            // Actualiza los campos de inicio y fin en el formulario
            $('#fechaInicio').val(fechaInicioSeleccionada);
            $('#fechaFin').val(fechaFinSeleccionada);

            // Muestra el formulario
            $('#form').show();
        }
    });
});
```

```

},
// Función que se ejecuta al renderizar un día del calendario
dayRender: function (date, cell) {
  // Deshabilita los días pasados
  var today = moment();
  if (date.isBefore(today, 'day')) {
    cell.css('background-color', '#f2f2f2'); // Cambia el color de fondo de los días pasados
    cell.addClass('fc-disabled-day'); // Agrega una clase para deshabilitar el clic
  }
},
// Función que especifica el rango de fechas válido
validRange: function (nowDate) {
  // Deshabilita las fechas pasadas
  return {
    start: nowDate
  };
},
// Función que se ejecuta al renderizar la vista del calendario
viewRender: function (view, element) {
  // Agrega clases de Bootstrap a los botones de navegación y estilos adicionales
  $(' .fc-prevYear-button, .fc-prev-button, .fc-next-button, .fc-nextYear-button, .fc-today-button')
    .addClass('btn btn-primary')
    .css({
      'margin': '1rem', // Ajusta los valores de padding según tus necesidades
      'font-size': '1rem' // Ajusta el tamaño de fuente según tus necesidades
    });

  // Obtiene la fecha actual
  var today = moment();
  var currentYear = today.year();

  // Deshabilita el botón de año anterior si estamos en el año actual
  if (view.intervalEnd.year() === currentYear) {
    $(' .fc-prevYear-button').prop('disabled', true).addClass('fc-state-disabled');
  } else {
    $(' .fc-prevYear-button').prop('disabled', false).removeClass('fc-state-disabled');
  }
},
// Configuración de idioma y botones de texto
locale: 'es',
buttonText: {
  today: 'HOY',
  month: 'mes',
  prev: '<<',
  next: '>>',
  prevYear: 'Año anterior',
  nextYear: 'Próximo año'
}

```

```

    },
    // Configuración adicional para el calendario
    firstDay: 1,
    dayNamesShort: ['Dom', 'Lun', 'Mar', 'Mié', 'Jue', 'Vie', 'Sáb'],
    dayNames: ['Domingo', 'Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado'],
    monthNames: ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre',
    'Octubre', 'Noviembre', 'Diciembre'],
    monthNamesShort: ['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic']
  });
});

```

Este script utiliza FullCalendar para crear un calendario interactivo en la página HTML. Algunos aspectos notables incluyen:

- Configuración de la apariencia del calendario, como los botones de navegación y la disposición del encabezado.
- Habilitación de la selección de fechas y manejo de eventos asociados a la selección.
- Configuración de estilos y deshabilitación de días pasados.
- Especificación del rango de fechas válido y configuración de idioma y botones de texto.

5.2.2. Capa de Negocio o Lógica de la Aplicación.

Aquí se presenta el código correspondiente a la parte servidor de la aplicación.

→ **GsaappApplication.java:**

```

package es.pfc.gsaapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class GsaappApplication {

    public static void main(String[] args) {
        SpringApplication.run(GsaappApplication.class, args);
    }
}

```

@SpringBootApplication: Esta anotación es una combinación de varias anotaciones, incluyendo @Configuration, @EnableAutoConfiguration, y @ComponentScan. Indica que esta clase es una clase de configuración de Spring, habilita la configuración automática de Spring Boot y escanea componentes dentro del paquete de la aplicación.

public static void main(String[] args): Este método es el punto de entrada principal para la aplicación. La línea SpringApplication.run(GsaappApplication.class, args); inicia la aplicación Spring Boot.

GsaappApplication: Esta es la clase principal de la aplicación, y es el punto de entrada cuando se ejecuta la aplicación Spring Boot.

→ **Usuario.java:**

```
package es.pfc.gsaapp.modelo;

import java.util.Collection;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

import es.pfc.gsaapp.modelo.tipos.EstadoPermiso;

@Entity
@Table(name = "usuarios", uniqueConstraints = @UniqueConstraint(columnNames = {"email",
"username"}))
public class Usuario {

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

    @Column(name = "username")
```

```

private String username;

@Column(name = "nombre")
private String nombre;

@Column(name = "apellidos")
private String apellidos;

@Column(name = "email")
private String email;

@Column(name = "password")
private String password;

@Column(name = "cargo")
private String cargo;

@Column(name = "antiguedad")
private String antiguedad;

@Column(name = "avatar")
private String avatar;

@ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
@JoinTable(
    name = "usuarios_rols",
    joinColumns = @JoinColumn(name = "usuario_id", referencedColumnName = "id"),
    inverseJoinColumns = @JoinColumn(name = "rol_id", referencedColumnName = "id")
)
@Column(name = "rol")
private Collection<Rol> roles;

@OneToMany(mappedBy = "usuario", cascade = CascadeType.ALL)
private Set<Permiso> permisos;

// Constructor, getters y setters

public boolean hasPermisoPendiente() {
    return this.getPermisos().stream().anyMatch(permiso -> permiso.getEstado() ==
EstadoPermiso.PENDIENTE);
}

// Otros métodos y constructores

public Usuario(Long id, String username, String nombre, String apellidos, String email, String
password, String cargo,
    String antiguedad, String avatar, Collection<Rol> roles, Set<Permiso> permisos) {

```



```

super();
this.id = id;
this.username = username;
this.nombre = nombre;
this.apellidos = apellidos;
this.email = email;
this.password = password;
this.cargo = cargo;
this.antiguedad = antiguedad;
this.avatar = avatar;
this.roles = roles;
this.permisos = permisos;
}

public Usuario(String nombre, String username, String apellidos, String email, String password,
String cargo,
    String antiguedad, String avatar, Collection<Rol> roles, Set<Permiso> permisos) {
    super();
    this.username = username;
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.email = email;
    this.password = password;
    this.cargo = cargo;
    this.antiguedad = antiguedad;
    this.avatar = avatar;
    this.roles = roles;
    this.permisos = permisos;
}

public Usuario() {}
}

```

@Entity: Esta anotación indica que la clase Usuario es una entidad JPA, lo que significa que está mapeada a una tabla en la base de datos.

@Table: Especifica el nombre de la tabla en la base de datos y las restricciones únicas en las columnas "email" y "username".

@Id: Indica que el campo id es la clave primaria de la entidad.

@GeneratedValue: Especifica la estrategia de generación de valores para la clave primaria (en este caso, se utiliza GenerationType.IDENTITY).

@Column: Se utiliza para mapear campos a columnas en la base de datos.

@ManyToMany y @JoinTable: Establecen la relación muchos a muchos entre usuarios y roles.

@OneToMany: Establece la relación uno a muchos entre usuarios y permisos. Se proporcionan constructores, getters y setters para acceder y modificar los atributos de la entidad.

hasPermisoPendiente(): Un método que verifica si el usuario tiene algún permiso pendiente.

→ **Rol.java:**

```
package es.pfc.gsaapp.modelo;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "rol")
public class Rol {

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

    // Constructores, getters y setters

    public Long getId() {
        return id;
    }

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

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

```

    }

    public Rol(Long id, String nombre) {
        super();
        this.id = id;
        this.nombre = nombre;
    }

    public Rol() {

    }

    public Rol(String nombre) {
        super();
        this.nombre = nombre;
    }
}

```

@Entity: Esta anotación indica que la clase Rol es una entidad JPA, lo que significa que está mapeada a una tabla en la base de datos.

@Table: Especifica el nombre de la tabla en la base de datos.

@Id: Indica que el campo id es la clave primaria de la entidad.

@GeneratedValue: Especifica la estrategia de generación de valores para la clave primaria (en este caso, se utiliza GenerationType.IDENTITY).

Se proporcionan constructores, getters y setters para acceder y modificar los atributos de la entidad.

→ **Permiso.java:**

```

package es.pfc.gsaapp.modelo;

import java.time.LocalDate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

```

```

import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

import org.springframework.format.annotation.DateTimeFormat;

import es.pfc.gsaapp.modelo.tipos.EstadoPermiso;

@Entity
@Table(name = "permisos")
public class Permiso {

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

    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @NotNull(message = "La fecha de inicio no debe ser nula")
    @Column(name = "fechaInicio")
    private LocalDate fechaInicio;

    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @NotNull(message = "La fecha de fin no debe ser nula")
    @Column(name = "fechaFin")
    private LocalDate fechaFin;

    @NotNull(message = "El tipo de permiso no debe ser nulo")
    @Column(name = "tipoPermiso")
    private String tipoPermiso;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "usuario_id")
    private Usuario usuario;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false) // Asegura que el estado no sea nulo en la base de datos
    private EstadoPermiso estado = EstadoPermiso.PENDIENTE;

    // Constructores, getters y setters

    public Permiso() {
        // Constructor por defecto
    }

    public Permiso(Long id, LocalDate fechaInicio, LocalDate fechaFin, String tipoPermiso) {
        this.id = id;
    }

```

```

        this.fechaInicio = fechaInicio;
        this.fechaFin = fechaFin;
        this.tipoPermiso = tipoPermiso;
        this.estado = EstadoPermiso.PENDIENTE; // Estado por defecto al crear un nuevo permiso
    }

    public Permiso(LocalDate fechaInicio, LocalDate fechaFin, String tipoPermiso) {
        this.fechaInicio = fechaInicio;
        this.fechaFin = fechaFin;
        this.tipoPermiso = tipoPermiso;
        this.estado = EstadoPermiso.PENDIENTE; // Estado por defecto al crear un nuevo permiso
    }

    // Getters y setters

    public Long getId() {
        return id;
    }

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

    public LocalDate getFechaInicio() {
        return fechaInicio;
    }

    public void setFechaInicio(LocalDate fechaInicio) {
        this.fechaInicio = fechaInicio;
    }

    public LocalDate getFechaFin() {
        return fechaFin;
    }

    public void setFechaFin(LocalDate fechaFin) {
        this.fechaFin = fechaFin;
    }

    public String getTipoPermiso() {
        return tipoPermiso;
    }

    public void setTipoPermiso(String tipoPermiso) {
        this.tipoPermiso = tipoPermiso;
    }

```

```

public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
}

public EstadoPermiso getEstado() {
    return estado;
}

public void setEstado(EstadoPermiso estado) {
    this.estado = estado;
}
}

```

@Entity: Esta anotación indica que la clase Permiso es una entidad JPA, lo que significa que está mapeada a una tabla en la base de datos.

@Table: Especifica el nombre de la tabla en la base de datos.

@Id: Indica que el campo id es la clave primaria de la entidad.

@GeneratedValue: Especifica la estrategia de generación de valores para la clave primaria (en este caso, se utiliza GenerationType.IDENTITY).

@DateTimeFormat: Anotación utilizada para especificar el formato de las fechas al ser representadas como cadenas.

@NotNull: Anotación de validación que asegura que los campos anotados no sean nulos.

@ManyToOne: Relación muchos a uno con la entidad Usuario.

@JoinColumn: Especifica la columna utilizada para la unión con la entidad relacionada (Usuario).

@Enumerated: Indica que el campo estado es un tipo enumerado (EstadoPermiso en este caso).

→ **UsuarioDTO.java:**

```

package es.pfc.gsaapp.controlador.dto;

public class UsuarioDTO {

```

```

private Long id;
private String username;
private String nombre;
private String apellidos;
private String email;
private String password;
private String cargo;
private String antiguedad;
private String rol;

// Constructores, getters y setters

public Long getId() {
    return id;
}

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

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

public String getEmail() {
    return email;
}

```

```

    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getCargo() {
        return cargo;
    }

    public void setCargo(String cargo) {
        this.cargo = cargo;
    }

    public String getAntiguedad() {
        return antiguedad;
    }

    public void setAntiguedad(String antiguedad) {
        this.antiguedad = antiguedad;
    }

    public String getRol() {
        return rol;
    }

    public void setRol(String rol) {
        this.rol = rol;
    }

    public UsuarioDTO(Long id, String username, String nombre, String apellidos, String email, String
password,
        String cargo, String antiguedad, String rol) {
        super();
        this.id = id;
        this.username = username;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.email = email;
    }

```



```

        this.password = password;
        this.cargo = cargo;
        this.antiguedad = antiguedad;
        this.rol = rol;
    }

    public UsuarioDTO(String username, String nombre, String apellidos, String email, String password,
String cargo,
        String antiguedad, String rol) {
        super();
        this.username = username;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.email = email;
        this.password = password;
        this.cargo = cargo;
        this.antiguedad = antiguedad;
        this.rol = rol;
    }

    public UsuarioDTO() {
    }
}

```

UsuarioDTO es un objeto de transferencia de datos utilizado para transferir información relacionada con el usuario entre las capas de la aplicación (por ejemplo, desde el controlador hasta los servicios).

Los campos en la clase corresponden a los atributos de la entidad Usuario.

Se proporcionan constructores para crear instancias de la clase UsuarioDTO con diferentes conjuntos de atributos.

Se proporcionan getters y setters para acceder y modificar los atributos de la clase.

Este DTO es útil para evitar exponer todos los detalles de la entidad Usuario en la capa de presentación y permite una mayor flexibilidad al definir la información que se transfiere entre las capas de la aplicación.

→ **EstadoPermiso.java:**

```

package es.pfc.gsaapp.modelo.tipos;

public enum EstadoPermiso {
    PENDIENTE, // Estado cuando el permiso está pendiente de aprobación
}

```

```
ACEPTADO, // Estado cuando el permiso ha sido aceptado
DENEGADO // Estado cuando el permiso ha sido denegado
}
```

EstadoPermiso es un enum que define tres constantes: PENDIENTE, ACEPTADO y DENEGADO, representando los diferentes estados en los que puede encontrarse un permiso.

Un **enum** es una estructura de datos en Java que representa un conjunto fijo de constantes. Cada constante del enum representa un estado específico que puede tener un permiso (pendiente, aceptado o denegado).

Los enums son útiles para representar conjuntos fijos de valores que no cambian durante la ejecución del programa.

Este enum será utilizado para definir el estado de los permisos en la entidad Permiso que compartiste anteriormente.

→ **SecurityConfiguration.java:**

Este código configura la seguridad en una aplicación Spring Boot utilizando Spring Security.

```
package es.pfc.gsaapp.seguridad;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

import es.pfc.gsaapp.servicio.UsuarioServicio;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    private UsuarioServicio usuarioServicio;
```

```

@Bean
public BCryptPasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
public DaoAuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
    auth.setUserDetailsService(usuarioServicio);
    auth.setPasswordEncoder(passwordEncoder());
    return auth;
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(authenticationProvider());
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers(
        "/registro**",
        "/js/**",
        "/css/**",
        "/img/**").permitAll()
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .loginPage("/login")
        .permitAll()
        .and()
        .logout()
        .invalidateHttpSession(true)
        .clearAuthentication(true)
        .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
        .logoutSuccessUrl("/login?logout")
        .permitAll();
}
}

```

@Configuration y @EnableWebSecurity: Estas anotaciones indican que esta clase es una configuración de Spring y habilita la seguridad web.

SecurityConfiguration extiende WebSecurityConfigurerAdapter: Esta clase proporciona configuraciones por defecto para la seguridad web.

@Autowired private UsuarioServicio usuarioServicio: Inyecta el servicio personalizado UsuarioServicio, que implementa UserDetailsService, necesario para cargar detalles del usuario durante la autenticación.

@Bean public BCryptPasswordEncoder passwordEncoder(): Define un bean para el codificador de contraseñas BCrypt. Se utiliza para codificar y verificar contraseñas de usuarios.

@Bean public DaoAuthenticationProvider authenticationProvider(): Define un bean para el proveedor de autenticación DAO. Configura el servicio de detalles del usuario y el codificador de contraseñas.

configure(AuthenticationManagerBuilder auth): Configura el administrador de autenticación para utilizar el proveedor de autenticación personalizado.

configure(HttpSecurity http): Configura la seguridad HTTP. Permite el acceso a ciertos recursos sin autenticación, configura la página de inicio de sesión, y maneja la lógica de cierre de sesión.

- **.authorizeRequests()** establece reglas de autorización.
- **.antMatchers("/registro**", "/js/**", "/css/**", "/img/**").permitAll()** permite el acceso sin autenticación a ciertos recursos.
- **.anyRequest().authenticated()** requiere autenticación para todas las demás solicitudes.
- **.formLogin()** configura el formulario de inicio de sesión.
- **.loginPage("/login")** establece la página de inicio de sesión personalizada.
- **.permitAll()** permite el acceso sin autenticación a la página de inicio de sesión.
- **.logout()** configura la funcionalidad de cierre de sesión.
- **.invalidateHttpSession(true)** invalida la sesión HTTP.
- **.clearAuthentication(true)** limpia la autenticación.
- **.logoutRequestMatcher(new AntPathRequestMatcher("/logout"))** establece la URL de cierre de sesión.

- `.logoutSuccessUrl("/login?logout")` redirige después de cerrar sesión.

→ **UsuarioServicio.java:**

```
package es.pfc.gsaapp.servicio;

import java.util.Optional;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.security.core.userdetails.UserDetailsService;

import es.pfc.gsaapp.controlador.dto.UsuarioDTO;
import es.pfc.gsaapp.modelo.Usuario;

public interface UsuarioServicio extends UserDetailsService {

    //    public List<Usuario> listarUsuarios();

    public Page<Usuario> listarUsuariosPorRolUser(Pageable pageable);

    public Usuario findByEmail(String email);

    public Usuario findByUsername(String username);

    public Optional<Usuario> findById(Long idUsuario);

    public Usuario guardar(UsuarioDTO registroDTO);

    public Usuario obtenerDetallesUsuarioPorId(Long id);

    public void denegarPermisos(Long usuarioid);

    public void aceptarPermisos(Long usuarioid);
}
```

extends UserDetailsService: La interfaz `UsuarioServicio` extiende `UserDetailsService`, lo que significa que este servicio proporciona funcionalidades necesarias para cargar detalles del usuario durante la autenticación de Spring Security.

Métodos del Servicio:

- **listarUsuariosPorRolUser(Pageable pageable):** Retorna una página de usuarios filtrados por roles.
- **findByEmail(String email):** Busca un usuario por su dirección de correo electrónico.
- **findByUsername(String username):** Busca un usuario por su nombre de usuario.
- **findById(Long idUsuario):** Busca un usuario por su ID.
- **guardar(UsuarioDTO registroDTO):** Guarda un nuevo usuario utilizando los datos proporcionados en un objeto UsuarioDTO.
- **obtenerDetallesUsuarioPorId(Long id):** Obtiene los detalles de un usuario por su ID.
- **denegarPermisos(Long usuarioid):** Deniega los permisos de un usuario.
- **aceptarPermisos(Long usuarioid):** Acepta los permisos de un usuario.

Estos métodos representan diversas operaciones relacionadas con la gestión de usuarios y permisos en la aplicación. La interfaz está diseñada para ser implementada por una clase de servicio con la lógica de negocio correspondiente.

→ **UsuarioServicioImpl.java:**

```
package es.pfc.gsaapp.servicio;

import java.util.Arrays;
import java.util.Collection;
import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
```

```

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import es.pfc.gsaapp.controlador.dto.UsuarioDTO;
import es.pfc.gsaapp.modelo.Permissions;
import es.pfc.gsaapp.modelo.Rol;
import es.pfc.gsaapp.modelo.Usuario;
import es.pfc.gsaapp.modelo.tipos.EstadoPermissions;
import es.pfc.gsaapp.repositorio.PermissionsRepositorio;
import es.pfc.gsaapp.repositorio.UsuarioPaginadorRepositorio;
import es.pfc.gsaapp.repositorio.UsuarioRepositorio;

@Service
public class UsuarioServicioImpl implements UsuarioServicio {

    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @Autowired
    private UsuarioRepositorio usuarioRepositorio;

    @Autowired
    private PermissionsRepositorio permissionsRepositorio;

    @Autowired
    private UsuarioPaginadorRepositorio usuarioPaginadorRepositorio;

    @Override
    public Usuario guardar(UsuarioDTO registroDTO) {
        Usuario usuario = new Usuario(registroDTO.getUsername(), registroDTO.getNombre(),
            registroDTO.getApellidos(), registroDTO.getEmail(),
            passwordEncoder.encode(registroDTO.getPassword()),
registroDTO.getCargo(),
            registroDTO.getAntiguedad(), null, Arrays.asList(new
Rol(registroDTO.getRol()), null);

        return usuarioRepositorio.save(usuario);
    }

    @Override
    public Page<Usuario> listarUsuariosPorRolUser(Pageable pageable) {
        return usuarioPaginadorRepositorio.listarUsuariosPorRolUser(pageable);
    }

    @Override
    public Usuario findByEmail(String email) {
        return usuarioRepositorio.findByEmail(email);
    }

```

```

    }

    @Override
    public Usuario findByUsername(String username) {
        return usuarioRepositorio.findByUsername(username);
    }

    @Override
    public Optional<Usuario> findById(Long idUsuario) {
        return usuarioRepositorio.findById(idUsuario);
    }

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        Usuario usuario = usuarioRepositorio.findByEmail(email);

        if(usuario == null) {
            throw new UsernameNotFoundException("Correo o password inválidos");
        }

        return new User(usuario.getEmail(), usuario.getPassword(),
mapearAutoridadesRoles(usuario.getRoles()));
    }

    private Collection<? extends GrantedAuthority> mapearAutoridadesRoles(Collection<Rol>
roles){
        return roles.stream().map(role -> new
SimpleGrantedAuthority(role.getNombre())).collect(Collectors.toList());
    }

    @Override
    public Usuario obtenerDetallesUsuarioPorId(Long id) {
        return usuarioRepositorio.findById(id)
        .orElseThrow(() -> new UsernameNotFoundException("Usuario no encontrado con ID: " +
id));
    }

    @Override
    public void aceptarPermisos(Long usuarioid) {
        Usuario usuario = usuarioRepositorio.findById(usuarioid)
        .orElseThrow(() -> new NoSuchElementException("Usuario no encontrado con id: " +
usuarioid));

        Set<Permiso> permisos = usuario.getPermisos();
        for (Permiso permiso : permisos) {
            if (permiso.getEstado() == EstadoPermiso.PENDIENTE) {
                permiso.setEstado(EstadoPermiso.ACEPTADO);
            }
        }
    }

```



```

    }
}

usuarioRepositorio.save(usuario);
}

@Override
public void denegarPermisos(Long usuarioid) {
    Usuario usuario = usuarioRepositorio.findById(usuarioid)
        .orElseThrow(() -> new NoSuchElementException("Usuario no encontrado con id: " +
usuarioid));

    Set<Permiso> permisos = usuario.getPermisos();
    Iterator<Permiso> iterator = permisos.iterator();
    while (iterator.hasNext()) {
        Permiso permiso = iterator.next();
        if (permiso.getEstado() == EstadoPermiso.PENDIENTE) {
            iterator.remove();
            permisoRepositorio.delete(permiso);
        }
    }

    usuarioRepositorio.save(usuario);
}
}

```

Inyección de Dependencias: Se utilizan las anotaciones `@Autowired` para inyectar dependencias, como el `BCryptPasswordEncoder`, repositorios y otros.

Método guardar: Crea y guarda un nuevo usuario en la base de datos. Codifica la contraseña utilizando `BCryptPasswordEncoder`.

Método listarUsuariosPorRolUser: Retorna una página de usuarios filtrados por roles mediante el uso de un repositorio paginador.

Métodos findByEmail y findByUsername: Buscan un usuario por su dirección de correo electrónico o nombre de usuario, respectivamente.

Método loadUserByUsername: Implementa la carga de detalles del usuario para la autenticación de Spring Security. Utiliza `User` de Spring Security.

Método mapearAutoridadesRoles: Convierte los roles del usuario en una colección de autoridades de Spring Security.

Método obtenerDetallesUsuarioPorId: Obtiene los detalles de un usuario por su ID.

Métodos aceptarPermisos y denegarPermisos: Aceptan o deniegan los permisos de un usuario, actualizando el estado de los permisos en la base de datos.

→ **PermisoServicio.java:**

```
package es.pfc.gsaapp.servicio;

import java.util.Collection;

import es.pfc.gsaapp.modelo.Permiso;

public interface PermisoServicio {

    void guardarPermiso(Permiso permiso);

    Collection<String> mapearUsuariosPermisos(Collection<Permiso> permisos);
}
```

guardarPermiso: Este método define la firma para guardar un objeto Permiso. Sin embargo, la implementación real de este método está en las clases que implementan esta interfaz.

mapearUsuariosPermisos: Este método toma una colección de objetos Permiso y devuelve una colección de nombres de usuarios asociados con esos permisos. La implementación de este método también se encuentra en las clases que implementan esta interfaz.

→ **PermisoServicioImpl.java:**

```
package es.pfc.gsaapp.servicio;

import java.util.Collection;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import es.pfc.gsaapp.modelo.Permiso;
import es.pfc.gsaapp.repositorio.PermisoRepositorio;

@Service
public class PermisoServicioImpl implements PermisoServicio {
```

```

@Autowired
private PermisoRepositorio permisoRepositorio;

public void guardarPermiso(Permiso permiso) {
    permisoRepositorio.save(permiso);
}

@Override
public Collection<String> mapearUsuariosPermisos(Collection<Permiso> permisos) {
    return permisos.stream().map(Permiso::getTipoPermiso).collect(Collectors.toList());
}
}

```

guardarPermiso: Este método utiliza el PermisoRepositorio para guardar un objeto Permiso en la base de datos.

mapearUsuariosPermisos: Este método toma una colección de objetos Permiso y utiliza Java Stream API para mapear los tipos de permisos (getTipoPermiso()) a una colección de cadenas. Devuelve una colección de strings que representan los tipos de permisos.

→ **Pageltem.java:**

```

package es.pfc.gsaapp.util.paginator;

public class Pageltem {

    private int numero;
    private boolean actual;

    public Pageltem(Integer numero, Boolean actual) {
        super();
        this.numero = numero;
        this.actual = actual;
    }

    public int getNumero() {
        return numero;
    }

    public boolean getActual() {
        return actual;
    }
}

```

Atributos:

- numero: Representa el número de la página.
- actual: Indica si el objeto PageItem representa la página actual.

Constructor:

- Se proporciona un constructor que toma un número de página (Integer numero) y un indicador de si es la página actual (Boolean actual).

Métodos:

- getNumero(): Devuelve el número de la página.
- getActual(): Devuelve un booleano indicando si es la página actual.

→ PageRender.java:

```
package es.pfc.gsaapp.util.paginator;

import java.util.ArrayList;
import java.util.List;

import org.springframework.data.domain.Page;

public class PageRender<T> {

    private String url;
    private Page<T> page;
    private int totalPaginas;
    private int numElementosPorPagina;
    private int paginaActual;
    private List<PageItem> paginas;

    public PageRender(String url, Page<T> page) {
        super();
        this.url = url;
        this.page = page;
        this.paginas = new ArrayList<PageItem>();

        numElementosPorPagina = page.getSize();
        totalPaginas = page.getTotalPages();
        paginaActual = page.getNumber() + 1;
    }
}
```

```

Integer desde, hasta;

if (totalPaginas <= numElementosPorPagina) {
    desde = 1;
    hasta = totalPaginas;
} else {
    if (paginaActual <= numElementosPorPagina / 2) {
        desde = 1;
        hasta = numElementosPorPagina;
    } else if (paginaActual >= totalPaginas - numElementosPorPagina / 2) {
        desde = totalPaginas - numElementosPorPagina + 1;
        hasta = numElementosPorPagina;
    } else {
        desde = paginaActual - numElementosPorPagina / 2;
        hasta = numElementosPorPagina;
    }
}

for (int i = 0; i < hasta; i++) {
    paginas.add(new PageItem(desde + i, paginaActual == desde + i));
}

public String getUrl() {
    return url;
}

public int getTotalPaginas() {
    return totalPaginas;
}

public int getPaginaActual() {
    return paginaActual;
}

public List<PageItem> getPaginas() {
    return paginas;
}

public boolean isFirst() {
    return page.isFirst();
}

public boolean isLast() {
    return page.isLast();
}

```

```
public boolean hasNext() {  
    return page.hasNext();  
}  
  
public boolean hasPrevious() {  
    return page.hasPrevious();  
}  
}
```

Atributos:

- url: La URL base para la paginación.
- page: La instancia de Page que se está paginando.
- totalPaginas: El número total de páginas disponibles.
- numElementosPorPagina: La cantidad de elementos por página.
- paginaActual: La página actual.
- paginas: Una lista de objetos PageItem que representan las páginas en el paginador.

Constructor:

- El constructor toma la URL base y la instancia de Page y realiza el procesamiento para determinar las páginas que se deben mostrar y crea objetos PageItem para cada página.

Métodos:

- Varias funciones de acceso (getters) para obtener información sobre la paginación.
- Métodos como isFirst(), isLast(), hasNext(), hasPrevious() para verificar la posición actual en la paginación.

→ **UsuarioRepositorio.java:**

```
package es.pfc.gsaapp.repositorio;
```

```

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import es.pfc.gsaapp.modelo.Usuario;

@Repository
public interface UsuarioRepositorio extends JpaRepository<Usuario, Long> {

    public Usuario findByEmail(String email);

    @Query("SELECT u FROM Usuario u WHERE u.username = :username")
    public Usuario findByUsername(String username);
}

```

Anotación @Repository:

- La anotación @Repository se utiliza para indicar que esta interfaz es un bean de repositorio Spring. Proporciona la funcionalidad de un repositorio de datos para la entidad Usuario.

Interfaz UsuarioRepositorio:

- Extiende JpaRepository<Usuario, Long>, lo que indica que es un repositorio de Spring Data JPA para la entidad Usuario con identificadores de tipo Long.
- Proporciona operaciones CRUD estándar para la entidad Usuario, como save, findById, findAll, delete, etc.

Método findByEmail:

- Define un método de consulta derivado por nombre convencional. Spring Data JPA generará automáticamente la consulta basada en el nombre del método. En este caso, busca un usuario por su dirección de correo electrónico.

Método findByUsername con @Query:

- Utiliza la anotación @Query de Spring Data JPA para definir una consulta personalizada. La consulta busca un usuario por su nombre de usuario (username).

→ PermisoRepositorio.java:

```

package es.pfc.gsaapp.repositorio;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import es.pfc.gsaapp.modelo.Permissions;

@Repository
public interface PermissionsRepository extends JpaRepository<Permissions, Long> {

    @Query("SELECT p FROM Permissions p WHERE p.tipoPermiso = :tipoPermiso")
    Permissions findByTipoPermiso(String tipoPermiso);
}

```

Interfaz PermissionsRepository:

- Extiende JpaRepository<Permissions, Long>, lo que indica que es un repositorio de Spring Data JPA para la entidad Permissions y utiliza identificadores de tipo Long.

Método findByTipoPermiso:

- Este método utiliza la anotación @Query de Spring Data JPA para definir una consulta personalizada.
- La consulta busca un Permissions basado en el tipo de permiso proporcionado (tipoPermiso).
- La consulta utiliza el nombre de los atributos de la entidad (tipoPermiso en este caso) en lugar de nombres de columna de base de datos.

Este repositorio proporciona métodos estándar CRUD (Create, Read, Update, Delete) para la entidad Permissions, además de un método personalizado (findByTipoPermiso) que utiliza una consulta personalizada para buscar permisos por tipo. Este método será implementado automáticamente por Spring Data JPA.

→ UsuarioPaginadorRepositorio.java:


```

package es.pfc.gsaapp.repositorio;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;

import es.pfc.gsaapp.modelo.Usuario;

@Repository
public interface UsuarioPaginadorRepositorio extends PagingAndSortingRepository<Usuario, Long> {

    @Query("SELECT u FROM Usuario u JOIN u.roles r WHERE r.nombre = 'ROLE_USER'")
    Page<Usuario> listarUsuariosPorRolUser(Pageable pageable);

}

```

Interfaz UsuarioPaginadorRepositorio:

- Extiende `PagingAndSortingRepository<Usuario, Long>`, indicando que es un repositorio de Spring Data JPA que admite operaciones de paginación y ordenación para la entidad `Usuario` con identificadores de tipo `Long`.

Método `listarUsuariosPorRolUser`:

- Este método utiliza la anotación `@Query` de Spring Data JPA para definir una consulta personalizada.
- La consulta busca usuarios (`Usuario`) que tienen roles con el nombre `"ROLE_USER"`. Utiliza una operación de `JOIN` en la consulta para obtener solo los usuarios con ese rol.
- Se utiliza paginación (`Pageable`) para obtener los resultados de la consulta de manera paginada.

→ `LoginControlador.java`:

```

package es.pfc.gsaapp.controlador;

import java.security.Principal;

```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller
public class LoginControlador {

    @GetMapping("/login")
    public String verPaginaInicioSesion(@RequestParam(value="error", required=false) String
error,

        @RequestParam(value="logout", required = false) String logout,
        Model modelo, Principal principal, RedirectAttributes flash,
        HttpServletRequest request, HttpServletResponse response) {

        // Verificar si ya hay una sesión activa
        if(principal != null) {
            flash.addFlashAttribute("info", "Ya ha iniciado sesión anteriormente");
            return "redirect:/";
        }

        // Limpiar la sesión al cargar la página de inicio de sesión
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.invalidate();
        }

        if(error != null) {
            modelo.addAttribute("error", "Error en el login: Nombre de usuario o contraseña
incorrecta, por favor vuelva a intentarlo!");
        }

        if(logout != null) {
            modelo.addAttribute("success", "Ha cerrado sesión con éxito!");
        }

        modelo.addAttribute("titulo", "Inicio sesión");

        return "login";
    }
}

```

Anotación @Controller:

- La anotación @Controller indica que esta clase sirve como un controlador de Spring MVC.

Método verPaginaInicioSesion con @GetMapping("/login"):

- Este método maneja las solicitudes GET para la URL /login.
- Utiliza parámetros de solicitud (error y logout) para manejar mensajes de error y éxito durante el proceso de inicio de sesión.
- Se verifica si ya hay un usuario autenticado (principal != null). En ese caso, se redirige a la página principal con un mensaje informativo.
- Se limpia la sesión al cargar la página de inicio de sesión para garantizar un inicio de sesión limpio.
- Los mensajes de error y éxito se agregan al modelo (modelo.addAttribute), lo que permite mostrar mensajes en la vista.
- Finalmente, devuelve el nombre de la vista "login", que probablemente corresponda a un archivo HTML o Thymeleaf.

→ InicioControlador.java:

```
package es.pfc.gsaapp.controlador;

import java.util.Optional;
import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
```

```

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import es.pfc.gsaapp.modelo.Permissions;
import es.pfc.gsaapp.modelo.Usuario;
import es.pfc.gsaapp.servicio.UsuarioServicio;
import es.pfc.gsaapp.util.paginator.PageRender;

@Controller
public class InicioControlador {

    @Autowired
    private UsuarioServicio usuarioServicio; // Inyección de dependencia del servicio de usuario

    @GetMapping(value = {"/", "/inicio"})
    public String verPaginaDelInicio(Model modelo, Authentication autenticacion,
        @RequestParam(name = "page", defaultValue = "0") int page,
        @RequestParam(name = "size", defaultValue = "3") int size) {
        String email = autenticacion.getName();
        Usuario usuario = usuarioServicio.findByEmail(email);

        if (usuario == null) {
            throw new UsernameNotFoundException("Usuario no encontrado");
        }

        modelo.addAttribute("titulo", "Inicio");
        modelo.addAttribute("usuario", usuario);

        Set<Permissions> permisos = usuario.getPermisos();
        modelo.addAttribute("permisos", permisos);

        // Paginador
        Pageable pageable = PageRequest.of(page, size);
        Page<Usuario> usuariosPage = usuarioServicio.listarUsuariosPorRolUser(pageable);

        PageRender<Usuario> pageRender = new PageRender<>("/inicio", usuariosPage);

        modelo.addAttribute("usuariosUser", usuariosPage);
        modelo.addAttribute("page", pageRender);

        return "inicio";
    }

    @GetMapping("/inicio/verAsunto/{idUsuario}")
    public String verDetalleUsuario(@PathVariable Long idUsuario, Model modelo) {
        Usuario usuario = usuarioServicio.obtenerDetallesUsuarioPorId(idUsuario);
        Set<Permissions> permisos = usuario.getPermisos();
    }

```

```

        modelo.addAttribute("usuario", usuario);
        modelo.addAttribute("permisos", permisos);
        return "verAsunto"; // Nombre de la nueva vista
    }

    @PostMapping("/inicio/denegarPermisos")
    public String denegarPermisos(@RequestParam Long usuarioid, Model modelo) {
        Optional<Usuario> optionalUsuario = usuarioServicio.findById(usuarioid);

        if (optionalUsuario.isPresent()) {
            usuarioServicio.denegarPermisos(usuarioid);
        }

        return "redirect:/inicio";
    }

    @PostMapping("/inicio/aceptarPermisos")
    public String aceptarPermisos(@RequestParam Long usuarioid, @RequestParam String accion,
        Model modelo) {
        Optional<Usuario> optionalUsuario = usuarioServicio.findById(usuarioid);

        if (optionalUsuario.isPresent()) {
            // Realizar la acción correspondiente (aceptar o denegar)
            if ("ACEPTAR".equals(accion)) {
                usuarioServicio.aceptarPermisos(usuarioid);
            }
        }

        return "redirect:/inicio";
    }
}

```

Inyección de Dependencia:

- Se utiliza la anotación `@Autowired` para inyectar el servicio `UsuarioServicio`.

Método `verPaginaDelInicio` con `@GetMapping(value = {"/", "/inicio"})`:

- Este método maneja las solicitudes GET para las URL `/` y `/inicio`.
- Utiliza el objeto `Authentication` para obtener el nombre del usuario autenticado y buscar la información del usuario correspondiente.
- Agrega atributos al modelo para mostrar información del usuario y sus permisos.

- Implementa un paginador para mostrar una lista paginada de usuarios.

Método verDetalleUsuario con @GetMapping("/inicio/verAsunto/{idUser}"):

- Este método maneja solicitudes GET para la URL /inicio/verAsunto/{idUser} y muestra detalles sobre un usuario específico.
- Utiliza la anotación @PathVariable para obtener el idUsuario de la URL.
- Agrega atributos al modelo para mostrar detalles del usuario y sus permisos.

Método denegarPermisos con @PostMapping("/inicio/denegarPermisos"):

- Este método maneja solicitudes POST para denegar permisos a un usuario específico.
- Utiliza la anotación @RequestParam para obtener el usuarioid de la solicitud.
- Llama al servicio para denegar los permisos y redirige a la página de inicio.

Método aceptarPermisos con @PostMapping("/inicio/aceptarPermisos"):

- Este método maneja solicitudes POST para aceptar permisos a un usuario específico.
- Utiliza la anotación @RequestParam para obtener el usuarioid y accion de la solicitud.
- Realiza la acción correspondiente (aceptar permisos) y redirige a la página de inicio.

→ CalendarioControlador.java:

```
package es.pfc.gsaapp.controlador;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class CalendarioControlador {
```

```

@GetMapping("/calendario")
public String verPaginaCalendario(Model modelo) {

    modelo.addAttribute("titulo", "Calendario");

    return "calendario-permisos";
}
}

```

Anotación @Controller:

- Indica que esta clase es un controlador de Spring.

Método verPaginaCalendario con @GetMapping("/calendario"):

- Este método maneja solicitudes GET para la URL /calendario.
- Utiliza la anotación @GetMapping para especificar la URL a la que responde.
- Agrega un atributo "titulo" al modelo con el valor "Calendario".
- Retorna la cadena "calendario-permisos", que es el nombre de la vista que se mostrará.

→ RegistroUsuarioControlador.java:

```

package es.pfc.gsaapp.controlador;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import es.pfc.gsaapp.controlador.dto.UsuarioDTO;
import es.pfc.gsaapp.servicio.UsuarioServicio;

@Controller
@RequestMapping("/registro")
public class RegistroUsuarioControlador {

    private UsuarioServicio usuarioServicio;
}

```

```

public RegistroUsuarioControlador(UsuarioServicio usuarioServicio) {
    super();
    this.usuarioServicio = usuarioServicio;
}

@ModelAttribute("usuario")
public UsuarioDTO retornarNuevoUsuarioRegistroDTO() {
    return new UsuarioDTO();
}

@GetMapping
public String mostrarFormularioDeRegistro() {
    return "registro";
}

@PostMapping
public String registrarUsuario(@ModelAttribute("usuario") UsuarioDTO registroDTO) {
    usuarioServicio.guardar(registroDTO);
    return "redirect:/registro?exito";
}
}

```

Anotaciones @Controller y @RequestMapping("/registro"):

- Indican que esta clase es un controlador y que responde a las solicitudes bajo la URL /registro.

Inyección de dependencias:

- El constructor toma una instancia de UsuarioServicio como un parámetro para la inyección de dependencias.

Método retornarNuevoUsuarioRegistroDTO con @ModelAttribute("usuario"):

- Este método se ejecuta antes de cada solicitud y agrega un atributo "usuario" al modelo con un nuevo objeto UsuarioDTO.

Método mostrarFormularioDeRegistro con @GetMapping:

- Maneja las solicitudes GET para la URL /registro.
- Retorna la cadena "registro", que es el nombre de la vista que se mostrará.

Método registrarUsuario con @PostMapping:

- Maneja las solicitudes POST para la URL /registro.
- Llama al método guardar del servicio de usuario para registrar al usuario.
- Después del registro exitoso, redirige a la URL /registro?exit.

→ PermisoControlador.java:

```
package es.pfc.gsaapp.controlador;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import es.pfc.gsaapp.modelo.Permiso;
import es.pfc.gsaapp.modelo.Usuario;
import es.pfc.gsaapp.servicio.PermisoServicioImpl;
import es.pfc.gsaapp.servicio.UsuarioServicio;

@Controller
public class PermisosControlador {

    @Autowired
    private UsuarioServicio usuarioServicio;

    private final PermisoServicioImpl permisoServicioImpl;

    // Inyecta el servicio en el constructor
    public PermisosControlador(PermisoServicioImpl permisoServicioImpl) {
        this.permisoServicioImpl = permisoServicioImpl;
    }

    @GetMapping("/calendario-permisos")
    public String mostrarCalendario(Model model) {
```

```

        model.addAttribute("permiso", new Permiso());
        return "calendario-permisos";
    }

    @PostMapping("/guardar-permiso")
    public String guardarPermiso(@ModelAttribute @Valid Permiso permiso, BindingResult result,
String email) {
        if (result.hasErrors()) {
            // Manejar errores de validación
            return "calendario-permisos";
        }

        // Obtener el ID del usuario actual (puedes usar el UserDetails de Spring Security)
        UserDetails userDetails = (UserDetails)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        Usuario usuario = usuarioServicio.findByEmail(userDetails.getUsername());

        // Asignar el usuario al permiso
        permiso.setUsuario(usuario);

        // Lógica para guardar el permiso en la base de datos
        permisoServicioImpl.guardarPermiso(permiso);

        // Redirige a la página de éxito o a donde sea necesario
        return "redirect:/inicio"; // Ajusta según tu aplicación
    }
}

```

Anotaciones y Autowiring:

- @Controller indica que esta clase es un controlador.
- @Autowired se utiliza para inyectar dependencias, en este caso, el servicio UsuarioServicio y PermisoServicioImpl.

Método mostrarCalendario con @GetMapping:

- Maneja solicitudes GET para la URL /calendario-permisos.
- Añade un nuevo objeto Permiso al modelo.

Método guardarPermiso con @PostMapping:

- Maneja solicitudes POST para la URL /guardar-permiso.

- Utiliza la anotación `@Valid` para activar la validación de bean.
- Verifica si hay errores de validación. Si los hay, regresa a la vista calendario-permisos.
- Obtiene el usuario actual desde el contexto de seguridad de Spring.
- Asigna el usuario al permiso y luego utiliza `permisoServicioImpl` para guardar el permiso en la base de datos.
- Redirige a `/inicio` después de guardar el permiso con éxito.

6. CONCLUSIÓN

6.1. Valoración Personal del Trabajo Realizado (análisis DAFO + análisis CAME).

<p>DEBILIDADES:</p> <p>Complejidad en la Gestión de Permisos: La gestión de permisos puede volverse compleja a medida que la aplicación crece, lo que podría llevar a posibles errores o problemas de seguridad.</p> <p>Posibles Desafíos de Rendimiento: Si la aplicación maneja grandes conjuntos de datos, podría enfrentar desafíos de rendimiento que podrían afectar la experiencia del usuario.</p> <p>Dependencia de Tecnologías Específicas: Si la aplicación está altamente acoplada a tecnologías específicas, puede haber desafíos en la adaptabilidad a nuevas tecnologías o actualizaciones.</p>	<p>FORTALEZAS:</p> <p>Sistema de Seguridad Robusto: La aplicación cuenta con un sistema de seguridad sólido que sigue las mejores prácticas y está en constante evolución para enfrentar posibles amenazas.</p> <p>Gestión de Usuarios y Roles Efectiva: La aplicación tiene un sistema eficaz de gestión de usuarios y roles, lo que facilita la administración y asignación de permisos.</p> <p>Base de Usuarios Estable: Contar con una base de usuarios establecida proporciona una base sólida para obtener retroalimentación y realizar mejoras.</p>
<p>AMENAZAS:</p> <p>Ciberseguridad y Amenazas Online: Dada la importancia de la seguridad, la aplicación podría enfrentar amenazas cibernéticas, como intentos de piratería o robo de datos.</p> <p>Competencia en Evolución: La competencia en el mercado de aplicaciones web puede ser intensa. Identificar y mantenerse al tanto de las acciones de la competencia es crucial.</p> <p>Cambios en la Legislación: Cambios en las leyes de privacidad y protección de datos pueden requerir ajustes significativos en la aplicación para cumplir con los nuevos requisitos.</p>	<p>OPORTUNIDADES:</p> <p>Optimización del Rendimiento: Mejorar el rendimiento de la aplicación para garantizar una experiencia fluida, incluso al manejar grandes cantidades de datos.</p> <p>Expansión de Funcionalidades: Identificar nuevas características y funcionalidades que podrían agregar valor a la aplicación y atraer a nuevos usuarios.</p> <p>Colaboración con Usuarios: Fomentar la colaboración con los usuarios para comprender mejor sus necesidades y expectativas, lo que puede impulsar el desarrollo de características demandadas.</p>

Confrontar (Desafíos y Problemas)
<ul style="list-style-type: none"> Desafíos en la Implementación de la Seguridad: La seguridad es crítica en las aplicaciones web, y es fundamental asegurarse de que se implementen las mejores prácticas de seguridad para proteger la información sensible de los usuarios y prevenir posibles amenazas. Manejo de Permisos y Roles: La gestión de permisos y roles puede volverse compleja a medida que la aplicación crece. Asegurarse de que el sistema actual sea escalable y fácil de mantener es esencial.
Adaptar (Oportunidades para Mejorar)
<ul style="list-style-type: none"> Optimización del Rendimiento: Evaluar y mejorar el rendimiento de la aplicación, especialmente al manejar grandes conjuntos de datos o un número creciente de usuarios. Interfaz de Usuario (UI) y Experiencia de Usuario (UX): Evaluar y actualizar la interfaz de usuario para mejorar la experiencia del usuario, asegurándose de que la aplicación sea intuitiva y fácil de usar. Implementación de Nuevas Funcionalidades: Identificar oportunidades para agregar nuevas características o funcionalidades que puedan mejorar el valor de la aplicación para los usuarios.
Mantener (Fortalezas a Conservar)
<ul style="list-style-type: none"> Estructura de Seguridad: Mantener y fortalecer las medidas de seguridad existentes, garantizando que la aplicación siga las mejores prácticas de seguridad y esté actualizada con los últimos estándares. Sistema de Gestión de Usuarios y Roles: Mantener y mejorar el sistema de gestión de usuarios y roles para garantizar que sea eficiente y cumpla con las necesidades cambiantes de la aplicación.
Explotar (Aprovechar los Éxitos)
<ul style="list-style-type: none"> Base de Usuarios Existente: Aprovechar la base de usuarios actual para obtener retroalimentación y realizar mejoras iterativas en la aplicación. Funcionalidades Diferenciadoras: Identificar y resaltar las características únicas que hacen que la aplicación se destaque en el mercado, utilizándolas como puntos de venta. Colaboración con Usuarios: Fomentar la colaboración con los usuarios para obtener información valiosa sobre sus necesidades y expectativas, lo que puede guiar el desarrollo futuro.

ANÁLISIS CAME

6.2. Bibliografía y Webgrafía.

Documentación CSS: <https://developer.mozilla.org/en-US/docs/Web/CSS>

Documentación Bootstrap 5:

<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

Validador HTML: <https://validator.w3.org/nu/>

Validador CSS: <https://jigsaw.w3.org/css-validator/>

Documentación Eclipse: <https://www.eclipse.org/documentation/>

Documentación Maven:

<https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/>

Tutorial Spring Boot: <https://danlaho.wordpress.com/?s=spring+boot>

Documentación Spring Boot: <https://www.baeldung.com/spring-boot>

Documentación Anotaciones Spring:

<https://www.digitalocean.com/community/tutorials/spring-annotations>

Repositorio de dependencias de Maven: <https://central.sonatype.com/>

Documentación DBeaver: <https://dbeaver.com/docs/wiki/>

Documentación MySQL: <https://dev.mysql.com/doc/>

Documentación Spring Boot: <https://www.arquitecturajava.com/categoria/spring-boot/>

Documentación Thymeleaf: <https://www.thymeleaf.org/documentation.html>

Documentación Apache Tomcat: <https://tomcat.apache.org/tomcat-8.0-doc/>

Documentación GitHub: <https://docs.github.com/es>

Riesgos laborales: <https://www.unir.net/ingenieria/revista/riesgos-laborales-informatica/>

Kickstarter:

<https://www.kickstarter.com/design-tech?ref=section-homepage-nav-click-design-tech>

Arquitectura de capas(1): <https://www.arimetrics.com/glosario-digital/capa-de-datos>

Arquitectura de capas(2): <https://blog.hubspot.es/website/que-es-arquitectura-en-capas>