Trabajo Fin de Máster

Aplicación Web de un bufete de abogados usando Spring Boot y Vaadin

Máster Universitario en Desarrollo Ágil de Software para la Web

Curso 2022/2023

Presentado por:
D. Jawher Landoulssi

Dirigido por:
D. Salvador Otón Tortosa

Alcalá de Henares, a 22 de septiembre de 2023

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster en Desarrollo Ágil de Software para la Web

Trabajo Fin de Máster

Aplicación Web de un bufete de abogados Usando Spring Boot y Vaadin

Autor: **Jawher Landoulssi**
Director: **Dr. Salvador Otón Tortosa**

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

CALIFICACIÓN: FECHA:

Universidad de Alcalá

Agradecimientos

El éxito alcanzado en mi recorrido académico ha sido posible gracias a la ayuda de varias personas, a las que deseo expresar mi más sincero agradecimiento.

En primer lugar, quisiera expresar mi sincero agradecimiento al supervisor de esta tesina, el Dr. Salvador Otón Tortosa por su paciencia, disponibilidad y, sobre todo, sus acertados consejos, que han contribuido a fundamentar mi reflexión.

También quiero dar las gracias a todo el personal docente de la Universidad de Alcalá y a los profesores profesionales responsables de mi formación, por su apoyo teórico y práctico.

A mis padres, Sami y Monia a quienes debo este logro, sin los cuales nada de esto hubiera sido posible. Les agradezco de todo corazón su dedicación, paciencia, amor y valentía. Si hoy escribo estas líneas es gracias a vosotros.
Gracias por tener fe en mí, y gracias por no dudar nunca de mis capacidades. Habéis dedicado vuestra vida a mi éxito, así que hoy os lo dedico a vosotros. Gracias a todos.

A mis hermanos, Hamza y Hazem, que han sido verdaderos pilares a lo largo del camino. Vuestras palabras de ánimo me han ayudado a superar todos los retos. Gracias a vosotros, he podido mantener la cabeza alta y no rendirme nunca. Gracias a todos.

Por último, quiero dar las gracias a mis amigos y a mi novia Sofia, que siempre han estado a mi lado. Su apoyo incondicional y sus ánimos han sido de gran ayuda.

A todas estas personas, mi agradecimiento, respeto y gratitud.

Resumen

El objetivo principal del proyecto fin de máster es desarrollar una aplicación web robusta que responda a las necesidades específicas de un bufete de abogados. Utilizando frameworks robustos, Spring Boot y Vaadin. Estos entornos de trabajo aprovechan las capacidades de Java como lenguaje de programación del lado del servidor para crear una lógica de negocio de alta calidad y una interfaz de usuario intuitiva y fácil de usar.

La presente aplicación está diseñada como una solución integral que ofrece una serie de funcionalidades esenciales tanto para el negocio jurídico como para sus clientes. Entre las características más destacadas se encuentran la capacidad de los usuarios para programar citas con el despacho y entablar una comunicación eficaz con los abogados a través de una sala de chat específica. Esta aplicación mejora enormemente el nivel de comunicación y colaboración entre el bufete de abogados y sus clientes.

Además, la aplicación proporciona a los usuarios la capacidad de supervisar y mantenerse al corriente del progreso de sus asuntos jurídicos, lo que les permite estar bien informados sobre el estado actual de sus expedientes. Además, la plataforma facilita la transmisión de los documentos pertinentes a los profesionales jurídicos, optimizando así el procedimiento y mejorando la facilidad de comunicación.

Además, se ha creado un área de trabajo que brinda a los abogados una visión completa del funcionamiento del bufete y les permite administrar de manera ágil los clientes, los expedientes y las citas. En este lugar, encontrarán una tabla de Excel detallada y gráficos interactivos que muestran el flujo de trabajo de manera visual y clara. Para optimizar la gestión de su trabajo en el bufete, esta herramienta les permitirá realizar un seguimiento eficiente de casos, identificar áreas de mejora y tomar decisiones inteligentes.

A lo largo de la duración de este proyecto, se tenido la oportunidad de utilizar los conocimientos y metodologías que se han aprendido y adquirido durante el transcurso del máster. Esta comprensión ha desempeñado un papel crucial a la hora de abordar eficazmente los obstáculos y las tareas asociadas a la creación de la aplicación.

Palabras Clave:

- Vaadin
- Spring Boot
- Java
- Base de Datos
- Interfaz de Usuario
- Back-end
- Abogado
- Usuario

Abstract

The main objective of the final master project is to develop a robust web application that meets the specific needs of a law firm. Using robust frameworks, Spring Boot and Vaadin. These frameworks leverage the capabilities of Java as a server-side programming language to create a high quality business logic and an intuitive and easy to use user interface.

This application is designed as an end-to-end solution that offers a number of essential functionalities for both the legal business and its clients. Among the most prominent features are the ability for users to schedule appointments with the firm and engage in effective communication with lawyers through a dedicated chat room. This application greatly enhances the level of communication and collaboration between the law firm and its clients.

In addition, the application provides users with the ability to monitor and keep abreast of the progress of their legal matters, allowing them to be well informed about the current status of their files. In addition, the platform facilitates the transmission of relevant documents to legal professionals, thus optimizing the procedure and improving the ease of communication.

In addition, a work area has been created that gives lawyers a complete overview of the law firm's operations and allows them to manage clients, files and appointments in a streamlined manner. Here, they will find a detailed Excel table and interactive graphics that show the workflow in a visual and clear way. To optimize the management of their work at the firm, this tool will allow them to efficiently track cases, identify areas for improvement and make intelligent decisions.

Throughout the duration of this project, there has been the opportunity to use the knowledge and methodologies that have been learned and acquired during the course of the master's degree. This understanding has played a crucial role in effectively addressing the obstacles and tasks associated with the creation of the application.

Key Words:

- Vaadin
- Spring Boot
- Java
- Database
- User Interface
- Back-end
- Lawyer
- User

Índice Resumido

Índice Detallado

Índice Figuras

Estado del Arte

Especificación Requisitos

Arquitectura de Software

Vaadin Framework

Configuracion Base de Datos

Configuracion General

Implementación Lógica de negocio

Diseño interfaz de Usuario

Autenticación y Autorización

Insalacion de la applicacion

Manual de despliegue

Manual De Usuario

Acesso Admin

Acesso Abogado

Acceso Abogado no-cliente

Acceso Cliente

I. Introducción

En esta memoria, se explicará de manera detallada el desarrollo de una aplicación web diseñada para un bufete de abogados. Esta plataforma se concibe como un espacio donde se facilita la interacción entre abogados y clientes, proporcionando a ambas partes una solución eficiente y ágil para mantener una comunicación fluida y efectiva.

Para entender mejor los retos de la digitalización de los servicios jurídicos, Maître Maghrebi, un abogado entrevistado para mi trabajo de fin de máster explica que "mantener a los clientes informados del progreso de su caso es vital. Sin embargo, a veces es difícil mantener una comunicación constante con todos tus clientes".
Por tanto, parece esencial buscar formas de optimizar esta relación.

Gracias a las tecnologías utilizadas como mySql, Spring boot y Vaadin, se ha creado una aplicación completa que satisface las necesidades de los usuarios.

II. Objetivos Del Proyecto

Los servicios jurídicos en línea existen desde hace algunos años. Esta revolución en la forma de actuar de los abogados ha mejorado y facilitado su trabajo.

Para justificar la elección del sector para este proyecto, es importante analizar los distintos problemas y necesidades a los que puede enfrentarse un abogado en el ejercicio de su profesión.

La creación de una plataforma digital dedicada al cliente y gestionada por el abogado parece ser una solución. Esta aplicación podría facilitar el contacto entre los clientes y abogados, haciendo que el abogado sea más eficaz y también más accesible para sus clientes. El objetivo sería facilitar el seguimiento de la evolución del expediente del cliente (nombre del gestor, jurisdicción, documentos, etc.) y, al mismo tiempo, poder comunicarse con el abogado.

Además, desde una perspectiva técnica, el proyecto consiste en la construcción del programa completo, incluyendo tanto el back-end como el front-end, utilizando el lenguaje Java del lado del servidor.

Esto es posible gracias al framework Vaadin, que destaca en el diseño de interfaces de usuario con sus robustos componentes, lo que permite a los desarrolladores agilizar el proceso de desarrollo y centrarse en ofrecer una experiencia de usuario fluida, reduciendo al mismo tiempo la complejidad de trabajar directamente con JavaScript y HTML.

III. Estado del arte

El sector informático, experimenta una rápida evolución, con la aparición cada día de tecnologías nuevas, donde los desarrolladores tienen acceso a multitud de tecnologías para crear diversas aplicaciones web.

La administración de bases de datos es crucial, y entre las opciones más comunes se encuentran PostgreSQL, MongoDB y MySQL, que permiten a los desarrolladores almacenar y procesar datos de forma eficiente.

Cuando se trata de crear una lógica de negocio sólida, los desarrolladores pueden elegir lenguajes de programación de alto rendimiento como C#, Java, Python y otros. Estos lenguajes son fundamentales en el campo del desarrollo de software, ya que proporcionan una amplia gama de funciones.

Para la interfaz de usuario se dispone de una diversa gama de lenguajes, cada uno capaz de ofrecer experiencias interesantes y completas. JavaScript, HTML y TypeScript se encuentran entre los principales candidatos, proporcionando a los desarrolladores las herramientas que necesitan para construir interfaces de usuario atractivas y repletas de funciones.

Es imprescindible mencionar los lenguajes de programación sin hablar de los entornos de trabajos (frameworks) o librerías asociadas a ellos. Hasta el día de hoy existieron varios entornos de trabajaos tanto del lado del servidor como del cliente.

En cuanto el lado del servidor (back-end) tenemos por ejemplo Django, Spring y .NET que facilitan el desarrollo de la capa de negocio del software gracias a sus clases predefinidas y a los sistemas de administración de bases de datos, como JPA en el caso de Spring.

Acerca de la interfaz de usuario, existen también entornos de trabajo importantes en el mercado como React, Angular y muchos más.

Figura 1 : Tecnologías Web

Fuente: Third Rock Techkn

Para poder desarrollar una aplicación eficiente y potente, se ha decidido utilizar MySql para la base de datos y como lenguajes de programación se ha elegido trabajar todo el proyecto con Java (back-end y front-end). Esto se ha logrado gracias a vaadin flow que nos ha permitido crear la interfaz de usuario utilizando solo con un lenguaje de server-side sin la necesidad de usar HTML/CSS/JS.

IV. Especificación de requisitos

El principal objetivo de esta aplicación es ofrecer accesibilidad sin restricciones a todos los usuarios. Al acceder a la página de inicio de la aplicación, se ofrecerá a las personas la oportunidad de solicitar una cita con la oficina jurídica. Una vez realizada la consulta, en caso de que un usuario, en lo sucesivo denominado Usuario X, decida contratar los servicios del bufete para su caso jurídico, se le proporcionará un nombre de usuario y una contraseña exclusivos. En esta situación, el Usuario X pasa de ser un "USUARIO" a ser un "CLIENTE".

La transición de roles va acompañada de funcionalidades y privilegios específicos para cada tipo de usuario, que se detallarán más adelante. Los usuarios tendrán acceso a un conjunto personalizado de funcionalidades y activos, en función de su designación como "USUARIO" o "CLIENTE".

Además, el sistema asignará la función de administrador únicamente al líder o fundador del bufete. El administrador designado será responsable de supervisar y gestionar los aspectos operativos de la aplicación, con el objetivo principal de mantener una funcionalidad sin fisuras y prestar una asistencia crucial tanto a los usuarios como a los clientes. Las personas que permanezcan empleadas como profesionales del derecho dentro del bufete de abogados serán designadas como "abogados", a cada uno de los cuales se le asignarán funciones y puntos de vista específicos dentro de la aplicación para satisfacer eficazmente las demandas de los clientes.

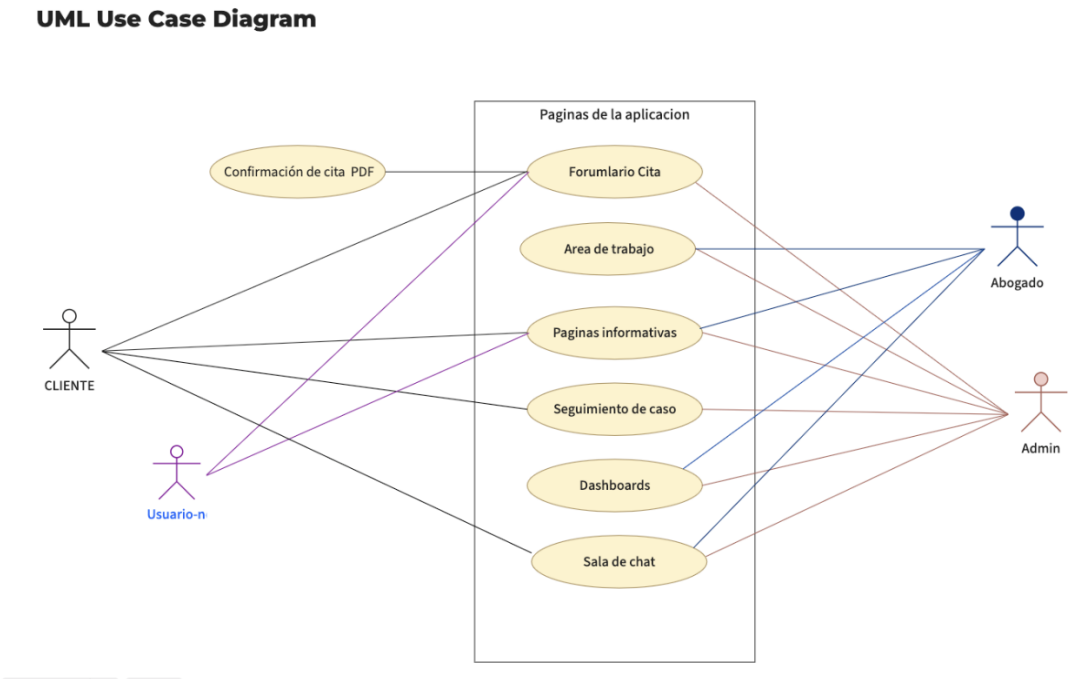


Figura 2 : Diagrama UML Caso de Uso de la aplicación

V. Arquitectura de software

Nuestro sistema está compuesto de varias partes, desde el almacenamiento de los datos hasta la interfaz de usuario, en este proceso se lleva a cabo interacciones entre las diferentes capas del software como indica la imagen a continuación.

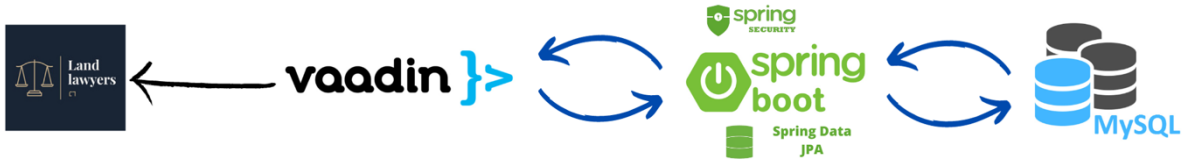


Figura 3 : Arquitectura de software

1. Base de Datos

1.1 MySql

En cuanto el almacenamiento de datos se ha utilizado Mysql, que es un sistema de gestión de bases de datos relacionales de código abierto(open-source) que proporciona un fuerte rendimiento y una gran capacidad de gestión de todos los tipos de datos.

La elección de MySQL como sistema de gestión de bases de datos asegura un almacenamiento eficiente y un óptimo rendimiento, ya que es conocido por su capacidad para gestionar una amplia variedad de datos y su naturaleza de código abierto

Al estar basado en código abierto permite a pequeñas empresas y desarrolladores disponer de una solución fiable y estandarizada para sus aplicaciones. Por ejemplo, si se cuenta con un listado de clientes, una tienda online con un catálogo de productos o incluso una gran selección de contenidos multimedia disponible, MySQL ayuda a gestionarlo todo debida y ordenadamente.

Figura 4 : Logotipo mySql

2. Lógica de negocio

2.1 Spring Boot

Como hemos mencionado, se ha elegido Spring Boot para desarrollar la lógica de negocio.

Spring Boot es una extensión del entorno de trabajo Spring que es de código abierto diseñado para facilitar el desarrollo de aplicaciones. Además de automatizar la configuración, también acelera el proceso de creación e implementación de aplicaciones.

Además, tiene como objetivo reducir la longitud del código y simplificar el desarrollo de aplicaciones web en comparación. Esta herramienta reduce el tiempo que lleva desarrollar aplicaciones aprovechando las anotaciones y la configuración estándar. Esto nos ayuda a crear aplicaciones independientes con menos o casi ninguna sobrecarga de configuración.

Figura 5 : Logotipo Spring Boot

2.2 Spring Data JPA

Spring Data JPA, que forma parte de la familia Spring Data más amplia, simplifica la construcción de repositorios basados en JPA. Se aborda el soporte mejorado para capas de acceso a datos basadas en JPA. Simplifica el desarrollo de aplicaciones impulsadas por Spring que aprovechan los métodos de acceso a datos.

Además, promete reducir al mínimo el trabajo necesario para mejorar significativamente la implementación de la capa de acceso a datos. Como desarrolladores, debemos crear las interfaces de su repositorio, que incluyen métodos de localización personalizados, y Spring las implementará automáticamente.

Figura 6 :Logotipo Spring Data

2.3 Spring Security

Spring Security es un framework de autenticación y control de acceso que es a la vez robusto y muy flexible. Se usa ampliamente para proteger aplicaciones basadas en Spring.

Spring Security se centra en proporcionar a las aplicaciones Java autenticación y permiso. El valor real de Spring Security, como todos los proyectos de Spring, se encuentra en la facilidad con la que se puede modificar para satisfacer requisitos personalizados.

Figura 7 : Logotipo Spring Security

3. Vaadin Framework

3.1 Vaadin componentes

Los componentes de Vaadin, además de ser modernos, ofrecen varias funcionalidades predefinidas que nos brindan una interfaz de usuario atractiva e inteligente. Existen varios componentes de diversos tipos, la mayoría de los cuales son de código abierto, aunque algunos requieren una licencia de pago.

Sin embargo, Vaadin ofrece una suscripción especial para estudiantes. Una vez que tu solicitud ha sido verificada y aprobada a través de GitHub, obtendrás todas las licencias de forma gratuita.

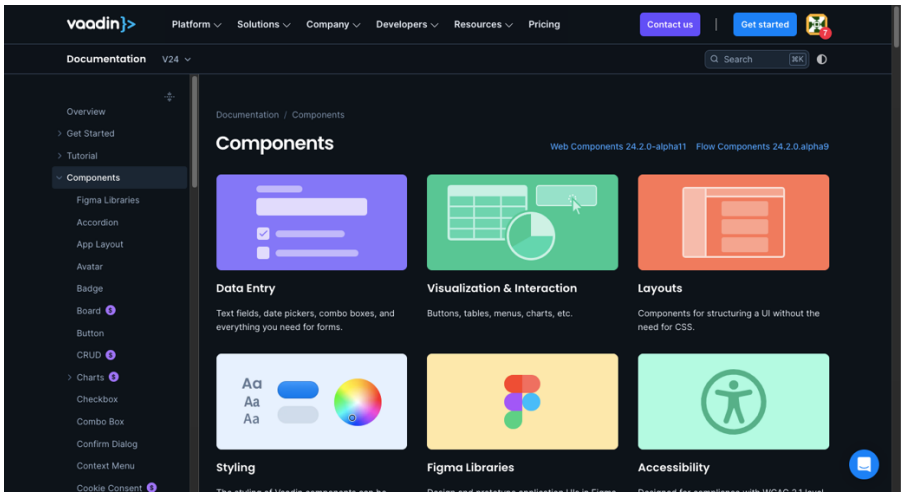


Figura 8 : Componenetes Vaadin

Como se muestra en la imagen anterior, Vaadin ofrece componentes para una amplia variedad de usos y funcionalidades, abarcando desde la entrada de datos hasta el manejo de estilos, así como las áreas de visualización, interacción y diseño.

3.1.1 Entrada de datos

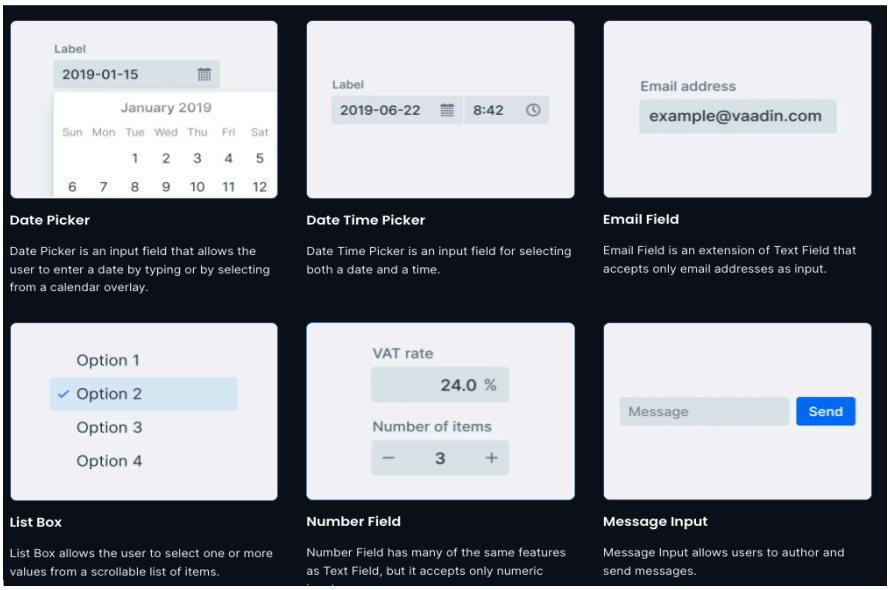


Figura 9 : Componentes tipo entrada de datos

Los componentes de la entrada de datos presentan un diseño moderno y cuentan de lógicas predefinidas. Por ejemplo, el campo **Email** acepta únicamente correos electrónicos, mientras que el campo **Number**, es similar a **TextField** pero solo permite la entrada de números.

3.1.2 Visualización e Interacción

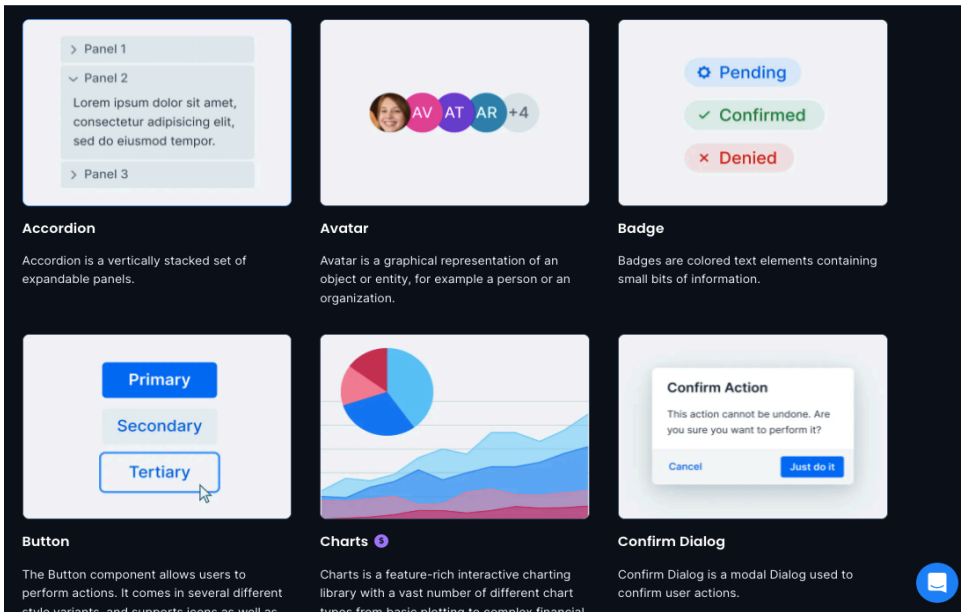


*Figura 10 : Componentes tipo **Visualización** y Interacción*

Para una experiencia de usuario atractiva, Vaadin proporciona una gama de componentes de visualización representan los datos y la información de manera efectiva en la interfaz de usuario como por ejemplo “Badge” or “Avatar” que permiten mostrar objetos o entidades en una forma organizada.

3.1.3 Layout

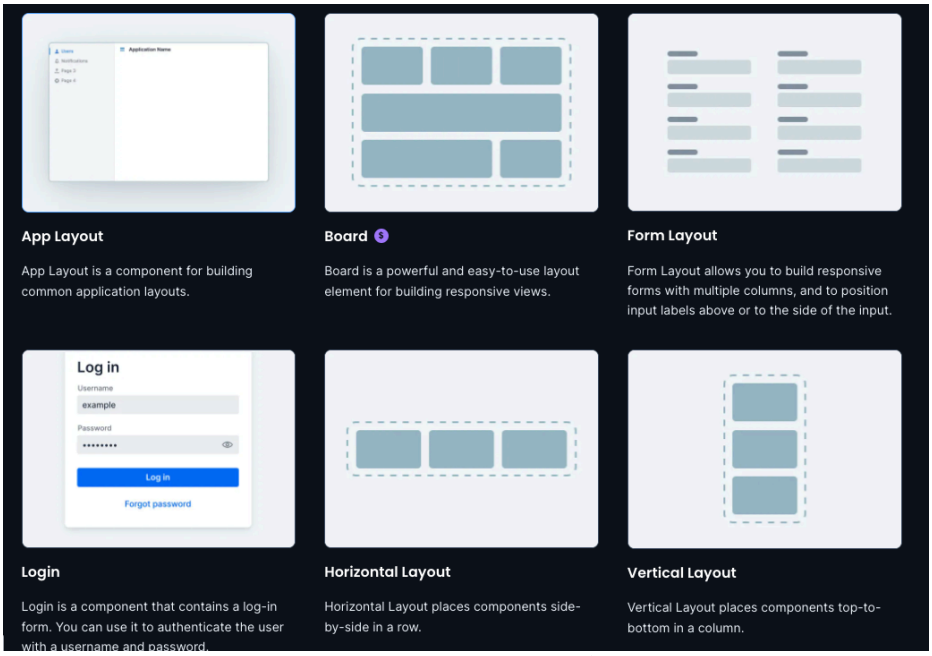


Figura 11 : Componentes tipo layout

En Vaadin, una vista (view) puede presentarse horizontal, vertical o directamente en una estructura de formulario. Esto es posible gracias a los componentes de layout (diseño) que ofrecen una disposición sencilla y atractiva

3.2 Vaadin Arquitectura

Vaadin es un entorno de trabajo open-source basado componentes para aplicaciones java, gracias a estos componentes no necesitamos utilizar HTML/JavaScript para crear la capa de presentación, Vaadin proporciona componentes de UI y patrones de UX que ayudan a crear experiencias de usuario consistentemente buenas en cada aplicación.

Vaadin Flow permite que una aplicación Java del lado del servidor cree una interfaz de usuario utilizando componentes Java. Estos componentes están vinculados a otros basados-web en el navegador. Además, maneja la retransmisión de las interacciones del usuario a la aplicación del lado del servidor, que puede manejarlas mediante detectores de eventos (event listeners)

Las vistas de aplicaciones y sus componentes se utilizan comúnmente para mostrar y aceptar la entrada de datos de la aplicación. Esta información frecuentemente se guarda en un servicio de back-end, como una base de datos. Los frameworks de aplicaciones como Spring se utilizan con frecuencia para construir la lógica de la aplicación, el diseño siguiente nos explica un poco más la arquitectura de Vaadin.

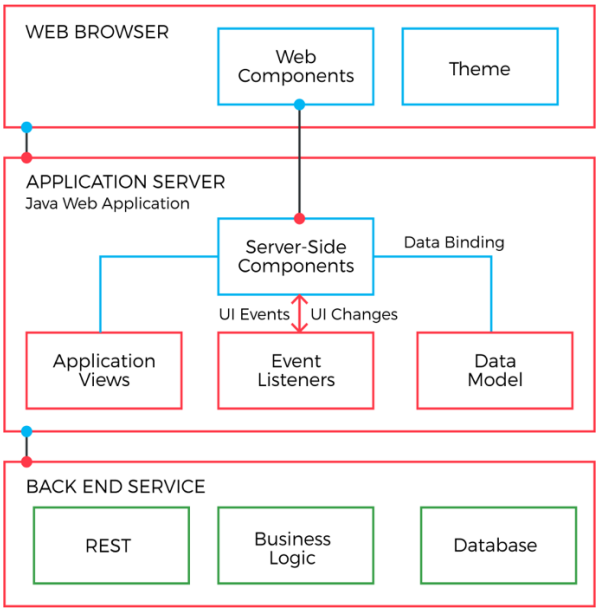


Figura 12 : Arquitectura de Vaadin

Fuente: Vaadin Application web site

Vaadin Flow posee una arquitectura que facilita la comunicación entre cliente- servidor y nos permite poner el foco en la interfaz de usuario para que sea una prioridad, También tiene un conjunto de componentes de interfaz de usuario de que se centran tanto en la experiencia del usuario final como en la del desarrollador para ofrecer una UX atractiva.
Además, en cuanto la navegación, vaadin tiene una API que permite al usuario navegar en las distintas páginas de la aplicación según su estructura.

VI. Concepción de la aplicación Land Lawyers

1. Creación Base de Datos

En nuestra aplicación se ha creado una base de dato con 7 tablas, que nos permiten gestionar y almacenar los datos de manera robusta y eficiente. A continuación, se mostrará un diagrama EER que nos ofrece una vista general de la arquitectura/diseño de nuestra base de datos.

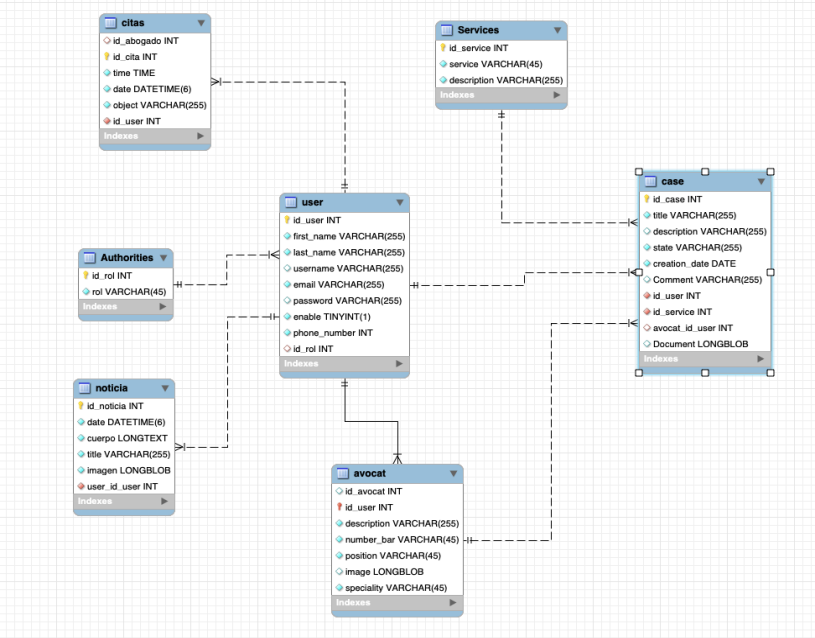


Figura 13 : Diagrama EER base datos

Según el diseño anterior, se puede notar que la mayoría de las relaciones entre las distintas tablas son relaciones Uno a muchos (One-To-Many) con la Tabla User, esto porque cada usuario puede tener multiples records asociados en otras tablas,por ejemplo citas o caso… lo que permite una representación completa y estructurada de las interacciones de los datos del usuario.
Existe la misma relación entre caso, abogado y cita servicio también.

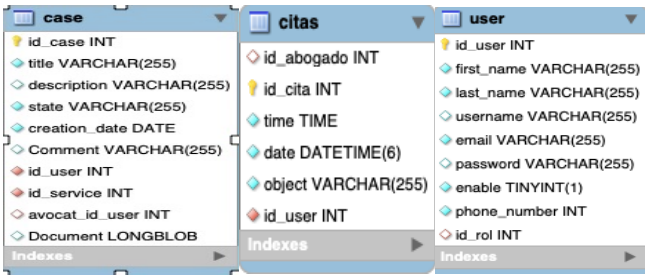


Figura 14 : Tablas caso,cita,usuario

Tenemos una relación uno a mucho **Identificativa** entre User y abogado es decir existe una fuerte dependencia entre ambas, tablas,

lo que significa que sin la existencia de la tabla padre (user) la tabla hija (avocat) no tiene sentido ya que un abogado está compuesto por información de ambas tablas.

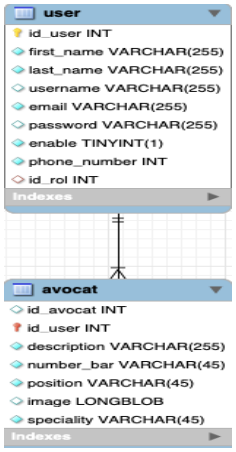


Figura 15 : Herencia tabla usuario y abogado

En cuanto a los roles de los usuarios, se ha creado una relación muchos a uno (Many-to-one) entre las tablas “authorities” y “user”, lo que permite que varios usuarios tengan solo un rol.

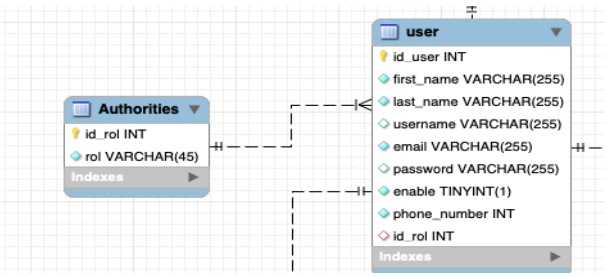


Figura 16 : Relación Tabla authorities y usuario

2. Configuración general

2.1 Configuración Vaadin

El primer paso para la creación de una aplicación web con Vaadin, es acceder a la interfaz del inicializador de Vaadin, donde se puede configurar la aplicación, crear vistas, seleccionar un tema y definir las tecnologías a utilizar, También hay opciones de despliegue, como Docker, Kubernetes or a Random port.

. Light Mode. .Dark Mode. . Technologys

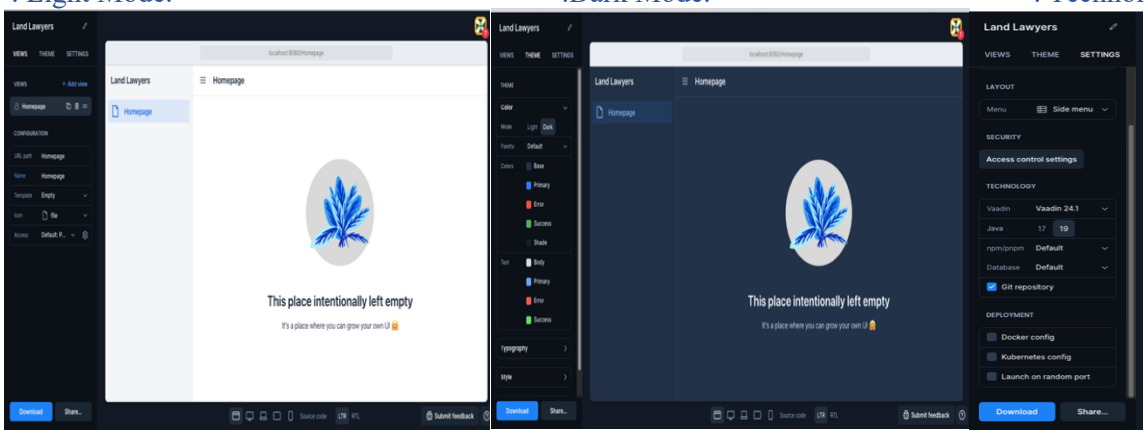


Figura 17 : Configuración Modo y Tecnologias

Una vez se ha terminado la etapa de configuración, al puslar “download”, se descargará un fichero ZIP que contiene las capetas del proyecto con Spring Boot y Maven integrados.

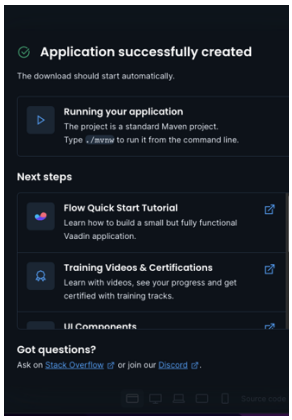


Figura 18 : Aplicación creada con éxito

Tras crear la aplicación, abrimos el archivo en el entorno de desarrollo Java correspondiente, donde accedemos a la carpeta del proyecto con todos los archivos necesarios, incluyendo la gestión de dependencias y los estilos, como se muestra en la imagen a continuación.

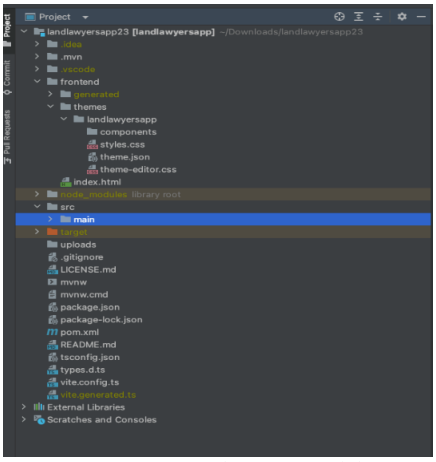


Figura 19 : Estructura de carpetas del proyecto

2.2 Configuración Spring boot y base de datos

Al abrir el archivo Pom.xml, encontramos las dependencias, plugins necesarias para generar nuestra app como nos indica la imagen siguiente.

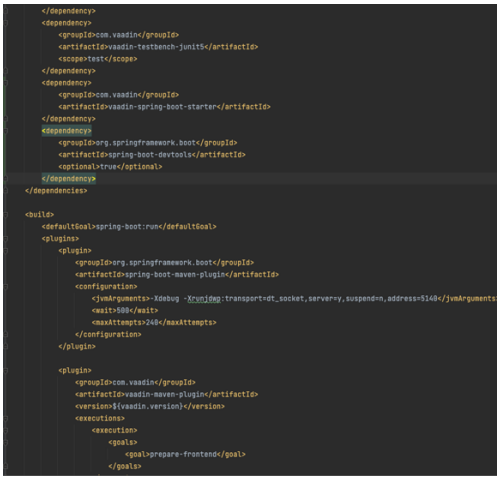


Figura 20 : Archivo pom xml

Una vez confirmamos que nuestra aplicación cuenta con los elementos necesarios para su generación, procedemos a la configuración propiedades de la aplicación, que incluye aspectos relacionados con la base de datos, JPA y la puerta de acceso del servidor.

```
server.port=${PORT:8080}
logging.level.org.atmosphere = warn
spring.mustache.check-template-location = false

# Launch the default browser when starting the application in development mode
vaadin.launch-browser=true
# To improve the performance during development.
# For more information https://vaadin.com/docs/flow/spring/tutorial-spring-configuration.html#special-configuration-parameters
vaadin.whitelisted-packages = com.vaadin,org.vaadin,dev.hilla,com.example.application
spring.jpa.defer-datasource-initialization = true
spring.datasource.url=jdbc:mysql://localhost:3306/firstDb
spring.datasource.username=root
spring.datasource.password=admin
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# CONFIGURACION MULTIPART (SUBIDA DE ARCHIVOS)
# Habilitamos subida de archivos
spring.servlet.multipart.enabled=true
# Maximo tamaño de archivos que se pueden subir
spring.servlet.multipart.max-file-size=20MB
spring.servlet.multipart.max-request-size=20MB

spring.web.resources.static-locations=classpath:/static/
path.to.directory.uploads=/Users/macbookair/Downloads/landlawversapp23/src/main/resources/uploads
```

Figura 21 : Archivo de configuración application properties

Como muestra la imagen anterior, en primer lugar, tenemos la puerta del servidor que es por defecto 8080 de localhost, luego encontramos las configuraciones de la conexión con la base de datos, donde mencionamos la ruta de nuestra base de datos, nombre de usuario y contraseña para accederla y otras configuraciones básicas de JPA. Finalmente realizamos la configuración de subida de archivos.

3. Implementación lógica de negocio

Nuestra aplicación cuenta con diversas funcionalidades distintas, por lo que es necesario implementar une lógica de negocia sólida y funcional, creando clases de entidades, repositorios y de servicios para lógralo.

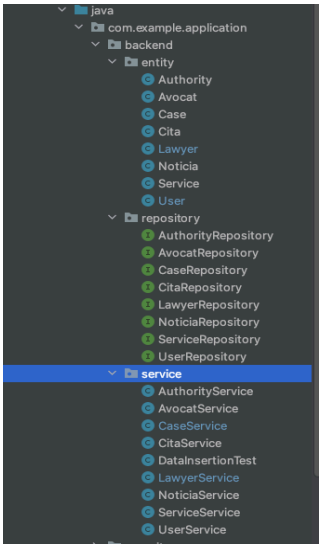


Figura 22 : Carpeta de servicios

3.1 Clases de entidad

Las clases de entidad son clases que representan los objetos y manejan los datos de dichos objetos.
Se toma como ejemplo, las clases de User y Abogado.

User Entity.

La clase de usuario es una clase que maneja la información de usuario, como nombre, apellido, correo, etc...
La clase User, es una superclase que hereda sus atributos, funciones a la clase abogado que es una subclase o clase hija de la clase usuario.

Se ha utilizado el tipo JOINED de la herencia, usando la anotación **@Inheritance (strategy=InheritanceType.JOINED)** esta estrategia consiste en generar 2 tablas por cada clase, permitiendo el acceso a los datos de la superclase 'User' a través del objeto 'abogado', y gracias a esta estrategia, se obtienen los objetos de esta manera.

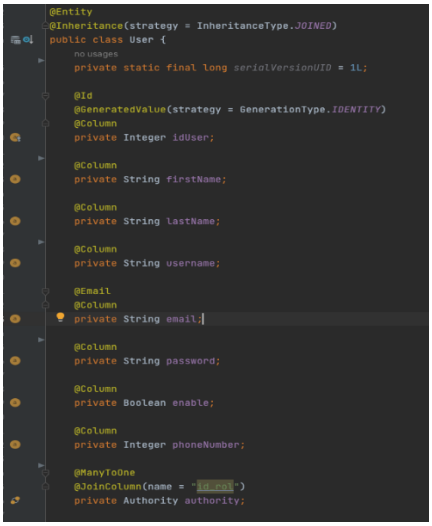


Figura 23 : Clase padre user

Abogado entity

Esta clase es la clase hija de 'User', que contiene información específica de los abogados, como su position, la especialidad o su imagen y otras importantes.

Implementando la herencia, usar un id como una clave primera no tiene sentido, porque las 2 tablas/clases van a utilizar el mismo id que es de la clase padre "User" y se realiza esto añadiendo la anotación **@PrimaryKeyJoinColumn(name = "id_user").**

Figura 24 : Clase hija Abogado

3.2 Clases de Repositorios

Las clases de repositorios son enlaces con la base de datos que nos permiten conectar a ella, donde también podemos añadir peticiones/consultas directas.

User Repository

```java
@Repository
public interface UserRepository extends JpaRepository<User, Long>, JpaSpecificationExecutor<User> {
    3 usages  MacbookAIR
    User findByUsername(String username);


    1 usage  MacbookAIR
    User findByEmail(String email);

    1 usage  MacbookAIR
    @Query("SELECT u FROM User u WHERE u.authority.idRol = :idRol")
    List<User> findUserByIdRol(@Param("idRol") Integer idRol);

    MacbookAIR
    @Query ("SELECT u FROM User u " +
            "where lower(u.firstName) like lower(concat('%', :searchTerm,'%')) " +
            " or lower(u.lastName) like lower(concat('%', :searchTerm,'%'))"
    )
    Collection<User> search(@Param("searchTerm") String filter);
}
```

Figura 25 : User repository

Abogado Repository

```java
@Repository
public interface AvocatRepository  extends JpaRepository<Abogado,Integer>{

    new *
    @Query("SELECT a FROM Abogado a " +
            "where lower(a.firstName) like lower(concat('%', :searchTerm,'%')) " +
            " or lower(a.position) like lower(concat('%', :searchTerm,'%'))"
    )
    Collection<Abogado> search(@Param("searchTerm") String filterLawyer);
}
```

Figura 26 : Abogado Repository

Se ha utilizado la anotación **@Query** para hacer peticiones explicitando las sentencias JPQL(Java Persistence Query Language) para buscar un abogado según su nombre o su puesto/posición en el bufete.

3.3 Clases de Servicio

Las clases de servicios son las encargadas de la lógica de negocio, donde se han implementado las operaciones CRUD y otras más.

User Service

```java
@Service
public class UserService implements CrudListener<User> {
11 usages
private final UserRepository userRepository;
2 usages
private  final AuthorityRepository authorityRepository;
    2 usages
    private final PasswordEncoder passwordEncoder;

    MacbookAIR *
    public UserService(UserRepository userRepository, AuthorityRepository authorityRepository,PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.authorityRepository = authorityRepository;
        this.passwordEncoder = passwordEncoder;
    }
    MacbookAIR
    @Override
    public Collection<User> findAll() {
        return userRepository.findAll();
    }
    MacbookAIR
    @Transactional
    public User add(User user) {
        if(user.getPassword() != null) {

            String encodedPassword = passwordEncoder.encode(user.getPassword());
            user.setPassword(encodedPassword);
        }
            return userRepository.save(user);

    }
    MacbookAIR
    public User update(User user) {
        return userRepository.save(user);
    }

    MacbookAIR
    public void delete(User user) {
userRepository.delete(user);
    }
```

Figura 27 : Servicio User

Abogado Service

```java
@Service
public class AvocatService implements CrudListener<Abogado> {

    7 usages
    private final AvocatRepository avocatRepository;

    MacbookAIR
    public AvocatService(AvocatRepository avocatRepository) {
        this.avocatRepository = avocatRepository;
    }

    MacbookAIR *
    @Override
    public Collection<Abogado> findAll() {
        return avocatRepository.findAll();
    }

    new *
    @Override
    public Abogado add(Abogado abogado) {
        return avocatRepository.save(abogado);
    }

    new *
    @Override
    public Abogado update(Abogado abogado) {
        return avocatRepository.save(abogado);
    }

    new *
    @Override
    public void delete(Abogado abogado) {
        avocatRepository.delete(abogado);
    }
    1 usage   new *
    public Collection<Abogado> findAllLawyers(String filter) {
        if(filter == null || filter.isEmpty()){
            return avocatRepository.findAll();
```

Figura 28 : Servicio Abogado

Como Vaadin ofrece una interfaz predefinida llamada CrudListener que encapsula las operaciones CRUD para delegarlas en nuestro back-end, aún no se han creado clases de implementación para las clases de servicios

Nota: para utilizar esta interfaz es necesario añadir la dependencia Crud-UI que está en el directory Add-on

4. Diseño Interfaz de Usuario

Después de haber desarrollar la lógica de negocio de nuestro sistema, comenzamos a crear las páginas con las que los usuarios van a interactuar

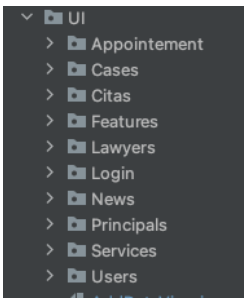


Figura 29 : Carpeta Interfaces de usuario

4.1 Menú de navegación

El primer paso es crear el menú de navegación y el encabezado, que permiten a los usuarios navegar en la aplicación de manera sencilla y rápida.

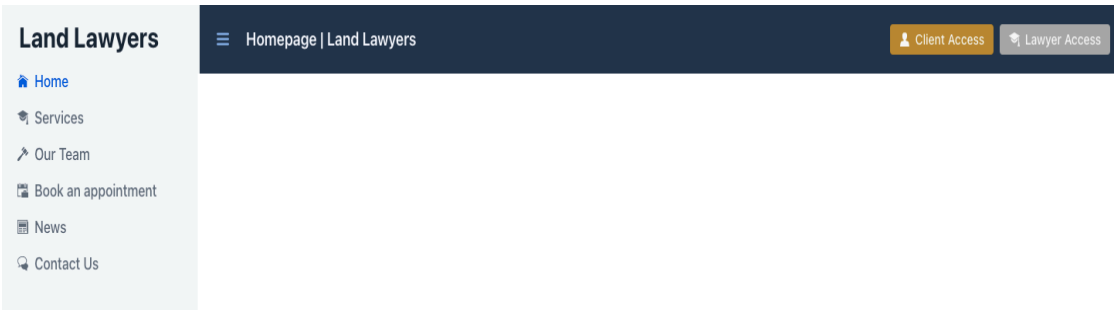


Figura 30 : Menu de Navegación

Creacion de drawer (cajon)

```
private Tabs createMenu() {
    final Tabs tabs = new Tabs();
    tabs.setOrientation(Tabs.Orientation.VERTICAL);
    tabs.addThemeVariants(TabsVariant.LUMO_MINIMAL);
    tabs.setId("tabs");

        Tab homeTab = createTab("Home", HomePage.class, new Icon(VaadinIcon.HOME));
        Tab ServicesTab = createTab("Services", ServiceListView.class, new
Icon(VaadinIcon.ACADEMY_CAP));
        Tab EquipoTab = createTab("Our Team", TeamView.class, new Icon(VaadinIcon.GAVEL));
        Tab CitaTab = createTab("Book an appointment", CitaForm.class, new
Icon(VaadinIcon.CALENDAR_BRIEFCASE));
        Tab Noticia = createTab("News", NewsView.class, new Icon(VaadinIcon.NEWSPAPER));
        Tab ContactTab = createTab("Contact Us", ContactView.class, new Icon(VaadinIcon.CHAT));
        tabs.add(homeTab, ServicesTab, EquipoTab, CitaTab, Noticia, ContactTab);
        return tabs;
}
```

Creamos los tabs que representan las vistas, luego pasamos el componente 'Tabs' como parámetro al método 'createDrawerContent' el cual se encarga de añadir estos tabs a la interfaz.

```
private Component createDrawerContent(Tabs menu) {
    VerticalLayout layout = new VerticalLayout();
    // styling for the drawer
    layout.setSizeFull();
    layout.setPadding(false);
    layout.setSpacing(false);
    layout.setMargin(false);
    layout.getThemeList().remove("spacing-s");
    layout.setAlignItems(FlexComponent.Alignment.STRETCH);
    layout.getStyle().set("margin-bottom","150px");

    // logo layout
    HorizontalLayout logoLayout = new HorizontalLayout();
    logoLayout.setId("logo");
    H2 titulo = new H2("Land Lawyers");
    titulo.getStyle().setColor("dark Blue");
    titulo.getStyle().set("margin-top","15px");
    titulo.getStyle().set("margin-left","22px");
    titulo.getStyle().set("font-weight","bold");
    layout.add(titulo,menu);

    return layout;
}
```

Basado en los componentes Vertical Layout y Horizantal layout se ha creado el drawer que es la parte a la derecha que contiene las rutas de navegación, añadido estilos con el método getStyle().

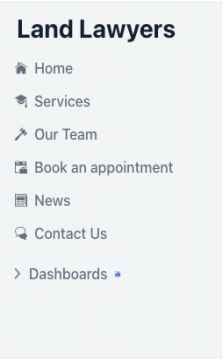


Figura 31 : Navegación lateral

Creación del encabezado

```
    private Component createHeaderContent() {
    HorizontalLayout layout = new HorizontalLayout();
    // styling the header
    layout.setId("header");
    layout.getThemeList().set("dark", true);
    layout.setWidthFull();
    layout.setHeightFull();
    layout.setHeight("70px");
    layout.setSpacing(false);
    layout.setMargin(false);
    layout.setPadding(false);
    layout.getThemeList().remove("spacing-s");
    layout.setAlignItems(FlexComponent.Alignment.CENTER);
    // drawer toggle button on the left
    layout.add(new DrawerToggle());
    // logo layout
    viewTitle = new H4();
    layout.add(viewTitle);
    var header = new HorizontalLayout( createAvatar() );
    header.setDefaultVerticalComponentAlignment(FlexComponent.Alignment.CENTER);
    //header.expand(createAvatar());
     header.getStyle().set("margin-left","auto");
    // Add the avatar
    layout.add(header);
    return layout;
}
```

se ha creado un layout horizantal para el encabezado, se ha elegido un tema oscuro.

```
Button clientButton = new Button("Client Access", new Icon(VaadinIcon.USER), e ->
getUI().ifPresent(ui -> ui.navigate("login")));
clientButton.getStyle().set("background-color","darkgoldenrod");
clientButton.getStyle().set("margin","8px");
clientButton.getStyle().set("color","whitesmoke");
clientButton.addThemeVariants(ButtonVariant.LUMO_SMALL);

Button lawyerButton = new Button("Lawyer Access", new Icon(VaadinIcon.ACADEMY_CAP), e ->
getUI().ifPresent(ui -> ui.navigate("login-lawyer")));
lawyerButton.getStyle().set("background-color","darkgrey");
lawyerButton.getStyle().set("margin-right","5px");
lawyerButton.getStyle().set("color","whitesmoke");
lawyerButton.addThemeVariants(ButtonVariant.LUMO_SMALL);
```

dentro del encabezado se han añadido dos distintos botones de acceso uno para los clientes, otro para los abogados

```
private String getCurrentPageTitle() {
    return getContent().getClass().getAnnotation(PageTitle.class).value();
}
public MainView() {
    // drawer for the menu
    setPrimarySection(Section.DRAWER);
    addToNavbar(true, createHeaderContent());
    // the menu in the drawer
    menu = createMenu();
    addToDrawer(createDrawerContent(menu));
}
```

También se ha creado un método que cambia el título según la página o vista en la que se encuentre el usuario
Por último, se han llamado las funciones en el constructor para que se muestren en la vista.

Figura 32 : Encabezado

4.2 Página de Inicio

La página de inicio es la página principal, donde se encontrará información general y diferentes rutas de distintas partes en la plataforma, para facilitar la navegación para los usuarios.

```
public HomePage() {


    //  the background image component
    Image backgroundImage = new Image( src: "images/background.jpg",  alt: "Background Image");
    backgroundImage.setHeight("500px");
    // backgroundImage.setWidth("1425px");
    //backgroundImage.getStyle().set("margin-top", "800px");
    getElement().getStyle().set("width", "100%");
    getElement().getStyle().set("margin-bottom", "30px");
    backgroundImage.setWidth("100%");
    backgroundImage.getElement().getStyle().set("border", "none");
    backgroundImage.addClassName(LumoUtility.Margin.NONE);
    //the text overlay
    Div textOverlay = createTextOverlay();
    // Adding components to the layout
    add(backgroundImage, textOverlay);
    setMargin(false);
    setPadding(false);
```

```
// Creation of 3 boxes under the image
1 usage   MacbookAIR *
private HorizontalLayout createImageSetsLayout() {
    HorizontalLayout imageSetsLayout = new HorizontalLayout();
    imageSetsLayout.setSpacing(false);
    String[] titles = {"Our Office", "Expertise Areas", "Contact Us"};
    for (int i = 0; i < 3; i++) {
        Div imageLayerContainer = createImageLayerContainer(titles[i]);
        imageSetsLayout.add(imageLayerContainer);
    }
    return imageSetsLayout;
}

1 usage   MacbookAIR
private Div createImageLayerContainer(String title) {
    Div imageLayerContainer = new Div();
    imageLayerContainer.addClassName("image-layer-container");
    Icon homeIcon = new Icon(VaadinIcon.GAVEL);
    homeIcon.setSize("60px");
    homeIcon.setColor("tan");
    homeIcon.getStyle().set("margin-bottom", "40px");
```

En cuanto la página de inicio se ha creado una imagen de fondo con un botón que dirige los clientes a la página de citas, donde se detallara en los apartados posteriores, se ha incluido tres cajas informativas que ofrecen al usuario una experiencia atractiva y comprensible.

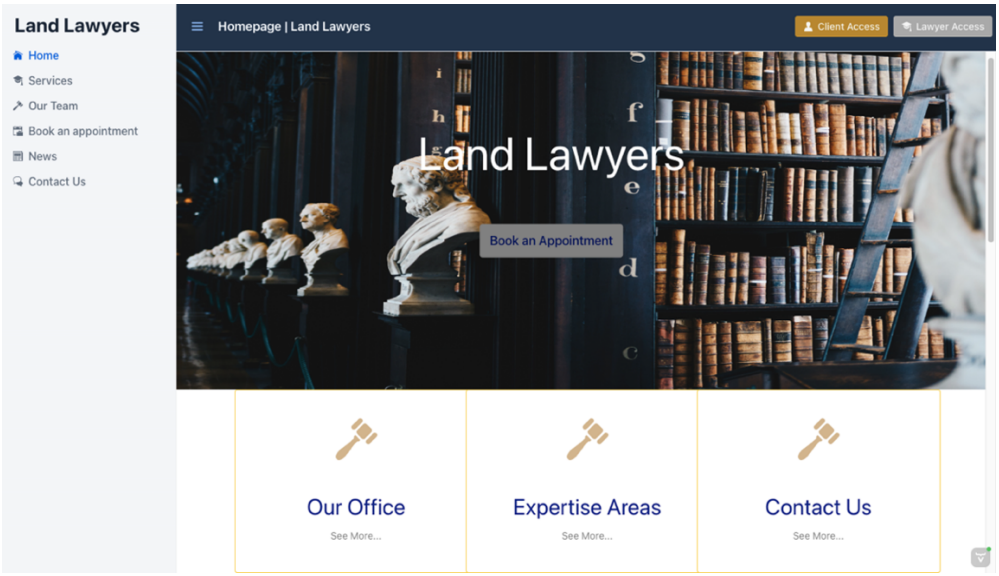


Figura 33 : Página de Inicio

4.3 Página de servicios

Para informar los clientes de nuestros servicios se ha diseño una página que contiene unos cuadros que indican los servicios que ofrece el bufete de abogados.

```
private void constructUI() {
    addClassNames("services");

    VerticalLayout verticalLayout = new VerticalLayout();
    verticalLayout.setSpacing(true);

    List<Service> services = serviceService.findAll();
     int cardsPerRow = 3;
     int totalRows = 10;

    for (int row = 0; row < totalRows; row++) {
```

```
        HorizontalLayout horizontalLayout = new HorizontalLayout();
        horizontalLayout.setSpacing(true);

        for (int col = 0; col < cardsPerRow; col++) {
            int index = row * cardsPerRow + col;
            if (index >= services.size()) {
                break;
            }

            Service service = services.get(index);
            Div div = new Div();
```

Se ha creado un título y un bucle que permiten extraer los servicios de la base de datos, incluyendo una descripción e iconos, para mostrarlos en líneas de tres columnas, como se muestra en la siguiente imagen.

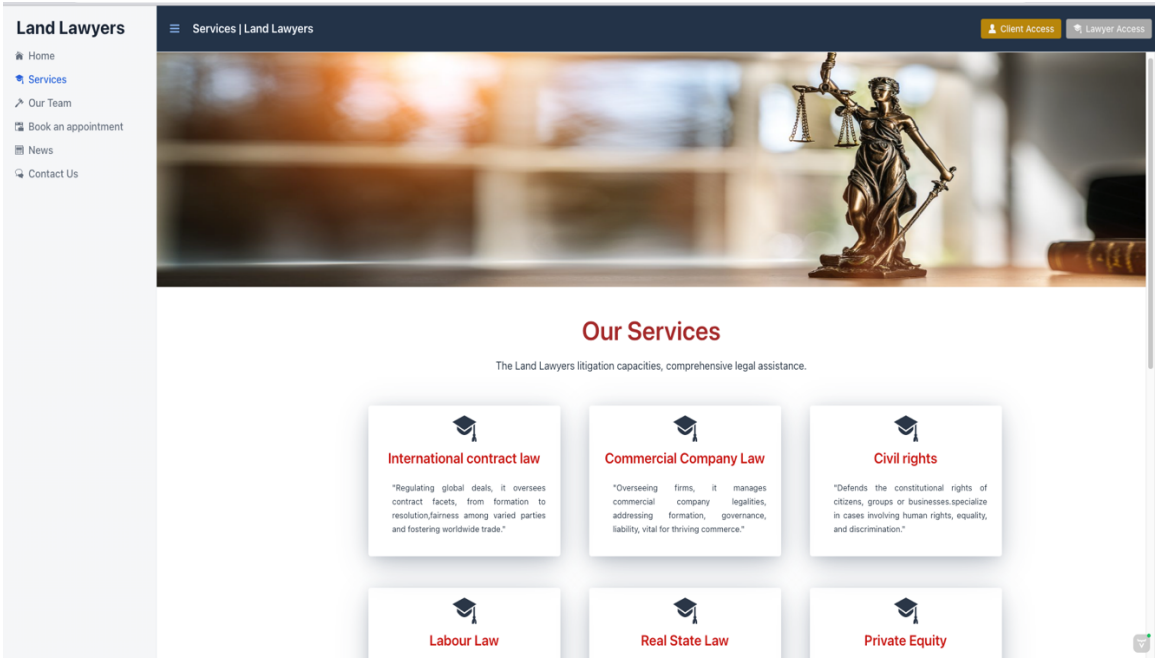


Figura 33: Vista de servicios jurídicos

4.4 Plantilla (team)

Esta página web es una interfaz informativa diseñada para mostrar a los usuarios quiénes forman parte de nuestro despacho, ofreciendo total transparencia

```
for (Abogado abogado : abogados) {
    if (Objects.equals(abogado.getNumberBar(),"Y88987K")) {
        mainAbogado = abogado;
    } else if (abogado.getImage() != null && abogado.getImage().length > 0) {
        otherAbogados.add(abogado);
    } else {
        Text error = new Text("no hay abogado 1");
        add(error);
    }
}
```

La visualización de abogados se ha divido, una línea para el abogado fundador y las otras líneas para el resto del equipo utilizando el número de registro del abogado para filtrar.

Cada abogado tiene un título, una descripción, su posición y una imagen.

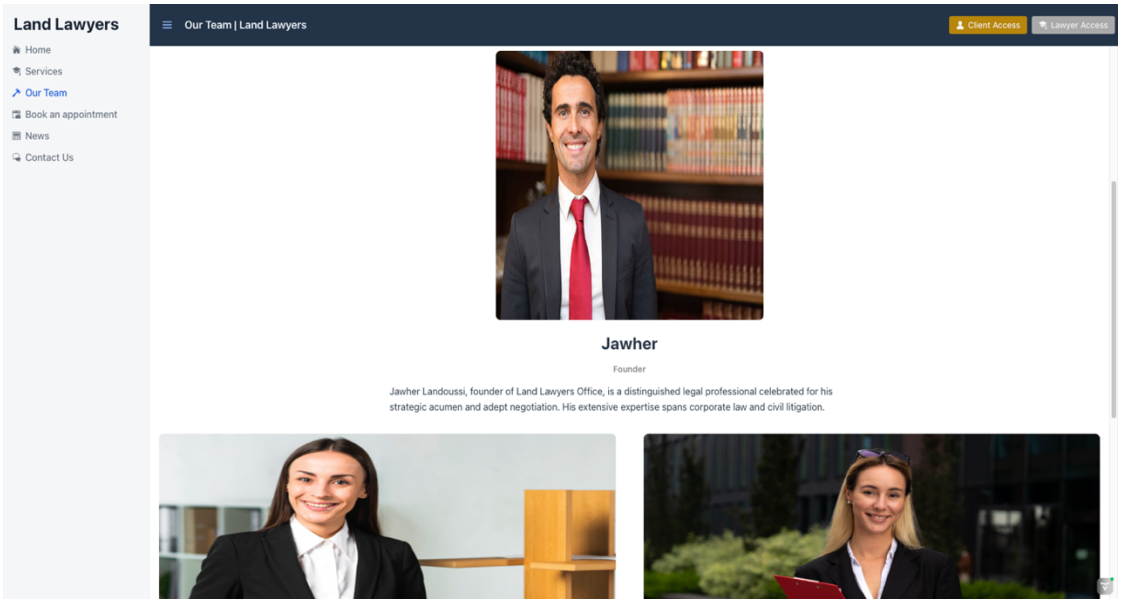


Figura 34 : Vista de Abogados

4.5 Formulario de cita

Una de las funcionalidades que ofrece la plataforma, es reservar una cita con un abogado del despacho, rellanado un formulario con los datos del usuario y de la cita como se muestra en las siguientes imágenes.

```
FormLayout formLayout = new FormLayout();
firstName = new TextField("Your name here");
lastName = new TextField("last name");
email = new EmailField("email");
phoneNumber = new IntegerField("phone number");
TextArea object = new TextArea("object");
DatePicker date = new DatePicker("Appointment date");
```

para crear el formulario de la cita, se han utilizado los componentes de vaadin, como **FormLayout**, **TextField**, **EmailField** y **DatePicker**.

```
LocalDate now = LocalDate.now(ZoneId.systemDefault());

date.setMin(now);
date.setMax(now.plusDays(120));
date.setHelperText("Must be within 60 days from today");
date.setHelperText("Mondays - Fridays only");
Binder<Cita> binder = new Binder<>(Cita.class);
binder.forField(date).withValidator(localDate -> {
    int dayOfWeek = localDate.getDayOfWeek().getValue();
    boolean validWeekDay = dayOfWeek >= 1 && dayOfWeek <= 5;
    return validWeekDay;
}, "Select a weekday").bind(Cita::getDate,
        Cita::setDate);
Button saveButton = new Button("Send");
saveButton.addThemeVariants(ButtonVariant.LUMO_PRIMARY,
        ButtonVariant.LUMO_CONTRAST);

TimePicker time = new TimePicker();
time.setLabel("Appointment time");
time.setHelperText("Open 8:00-12:00, 13:00-16:00");
time.setStep(Duration.ofMinutes(30));
time.setMin(LocalTime.of(8, 0));
time.setMax(LocalTime.of(16, 0));

binder.forField(time).withValidator(startTime -> {
    return !(LocalTime.of(8, 0).isAfter(startTime)
```

```
            || (LocalTime.of(12, 0).isBefore(startTime)
            && LocalTime.of(13, 0).isAfter(startTime))
            || LocalTime.of(16, 0).isBefore(startTime));
}, "The selected time is not available, please select a working time").bind(Cita::getTime,
        Cita::setTime);
```

El código anterior ejemplifica la creación de una lógica que restringe al usuario a seleccionar citas únicamente en días laborables y dentro del horario de trabajo.

```
saveButton.addClickListener(event -> {
             saveButton.setEnabled(true);

             LocalDate dateCita = date.getValue();
             LocalTime timeCita = time.getValue();
             Cita existingAppointment = citaService.findCitaByDateAndTime(dateCita,
timeCita);
             if(existingAppointment != null ) {
                 Notification notification2 = new Notification(
                         "No available appointment in this time",
                         5000,
                         Notification.Position.BOTTOM_CENTER
                 );
                 notification2.addThemeVariants(NotificationVariant.LUMO_ERROR);
                 notification2.open();
             } else {
                 String userEmail = email.getValue();
                 User existingUser = userService.findUserByEmail(userEmail);
```

Se ha desarrollado una lógica que verifica en la base de datos si ya existe una cita programada para el mismo día y hora seleccionados por el usuario. En caso de que no exista ninguna cita previa, la lógica procede a guardar la cita en la base de datos. Sin embargo, si ya hay una cita registrada en ese mismo día y horario, se muestra un mensaje de error indicando que el tiempo o el día elegidos no están disponibles

```
if (existingUser == null) {
       User newUser = new User();
       newUser.setFirstName(firstName.getValue());
       newUser.setLastName(lastName.getValue());
       newUser.setEmail(userEmail);
       newUser.setPhoneNumber(phoneNumber.getValue());
       Authority authority = new Authority();
       authority.setIdRol(4);
       newUser.setAuthority(authority);
       existingUser = userService.add(newUser);
   }
   Cita newCita = new Cita();
   newCita.setDate(date.getValue());
   newCita.setTime(time.getValue());
   newCita.setObject(object.getValue());
   newCita.setUser(existingUser);
   citaService.add(newCita);
   Notification notification = Notification
           .show("Appointment Booked successfully !");
```

Después de verificar la disponibilidad del día y la hora seleccionados y confirmar que no existe una cita previa en ese momento, el sistema procede a realizar las siguientes acciones:
Si el usuario que está programando la cita es un nuevo cliente y no está registrado en el sistema, se recopilan sus datos y se crea un nuevo perfil con un rol de 'Usuario no cliente' (User non-client).
Si el usuario ya es un cliente registrado (Usuario autenticado), se utilizan automáticamente sus datos de perfil existentes, y solo se registran los detalles específicos de la cita según se detalla en el siguiente código.

```
Optional<User> userOptional = authenticatedUser.get();

if (userOptional.isPresent()) {
    User connectedUser = userOptional.get();
    System.out.println("*****" + connectedUser);
    firstName = new TextField("Your name here", connectedUser.getFirstName(),
connectedUser.getFirstName());
    lastName = new TextField("last name", connectedUser.getLastName(),
connectedUser.getLastName());
    email = new EmailField("email", connectedUser.getEmail(), listener);
    phoneNumber = new IntegerField("phone number", connectedUser.getPhoneNumber(), listener2);
} else {
    firstName = new TextField("Your name here");
    lastName = new TextField("last name");
    email = new EmailField("email");
    phoneNumber = new IntegerField("phone number");
}
```

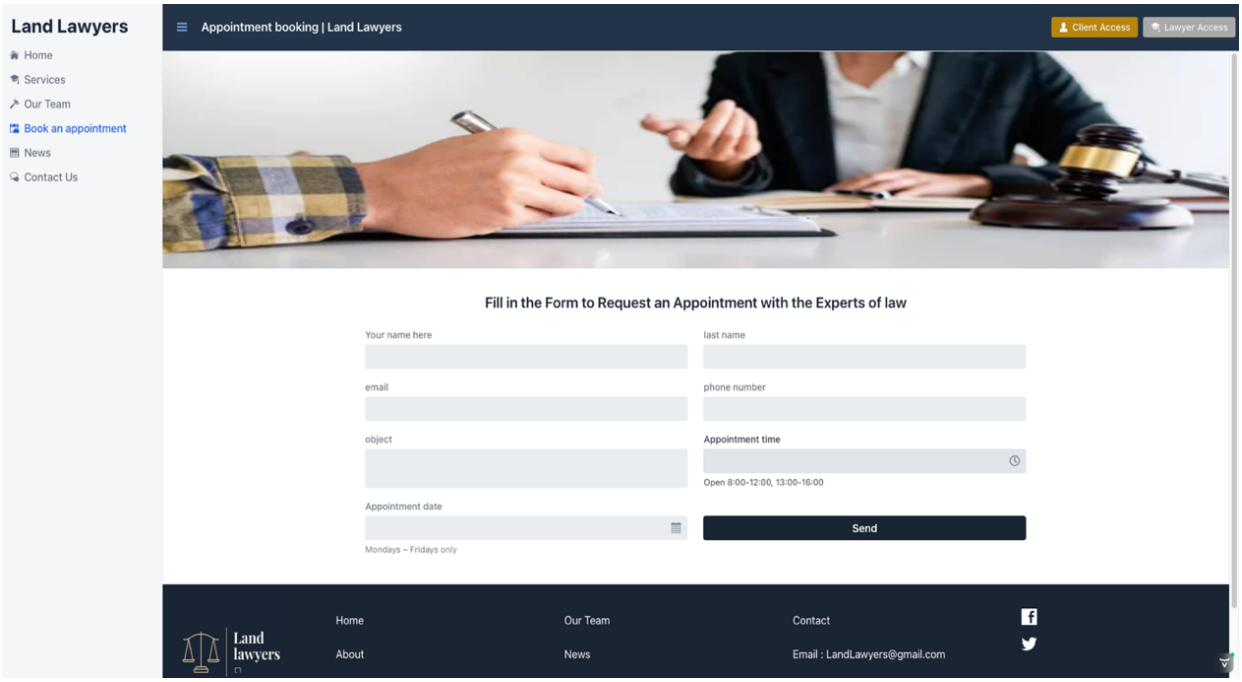


Figura 35 : Vista Formulario de cita

4.6 Página de Noticias

Esta página de noticias proporciona una amplia gama de actualidades de todos los tipos, manteniendo el usuario informado sobre una variedad de temas de interés

```
  for (Noticia noticia : news) {
        VerticalLayout verticalLayout = new VerticalLayout();
        verticalLayout.addClassNames(LumoUtility.Margin.Horizontal.AUTO,
LumoUtility.MaxWidth.SCREEN_LARGE, LumoUtility.Margin.Bottom.NONE);
        H1 title = new H1(noticia.getTitle());
        title.getStyle().set("font-size", "36px");

        Paragraph body = new Paragraph(noticia.getCuerpo());
        body.getStyle().set("width", "90%");

        Div dateDiv = new Div(new Text(String.valueOf(noticia.getDate())));
        dateDiv.getStyle().set("display", "inline-block");
        dateDiv.getStyle().set("margin-bottom", "10px");
        dateDiv.getStyle().set("margin-top", "20px");
        dateDiv.getStyle().set("width", "80%");

        dateDiv.getStyle().set("padding-bottom", "50px");
        dateDiv.getStyle().set("border-bottom", "2px solid grey");
```

```
        verticalLayout.add(title, body, dateDiv);

        HorizontalLayout horizontalLayout = new HorizontalLayout();
        Image image = new Image(getImageResource(noticia), "Image not found");
        image.setHeight("350px");
        image.setWidth("600px");

        image.getStyle().set("border-radius","5px");
        horizontalLayout.add(verticalLayout, image);
        horizontalLayout.getStyle().setMargin("20px");
        horizontalLayout.getStyle().set("margin-bottom","-8px");
        add(horizontalLayout);
    }
}
```

En cuanto la página de noticias se ha sacado las noticias de la base de datos, luego se ha mostrado la fecha de creación de la noticia, titulo y párrafo en una columna y la imagen en la misma línea, pero en otra columna.

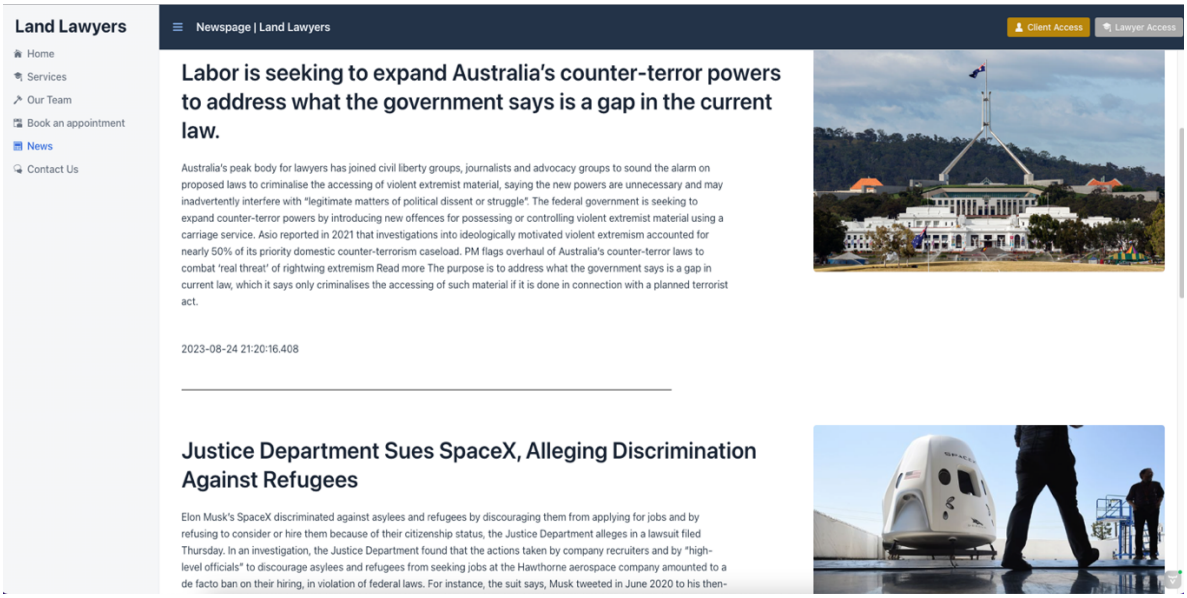


Figura 36 : Vista de Noticias

4.7 Página de Contacto

Se ha implementado una página que está compuesta de varios elementos, que proporcionan al usuario información que le permite a comunicar con el bufete.

```
Map map = new Map();
add(map);
MarkerFeature customizedMarker = new MarkerFeature(new Coordinate(  10.64431827626331,
35.656913226747434   ));
customizedMarker.setText("Land Lawyer Office");
customizedMarker.setTextStyle(textStyle);
customizedMarker.setDraggable(true);
map.getFeatureLayer().addFeature(customizedMarker);
// Listen to marker drop event
map.addFeatureDropListener(event -> {
    MarkerFeature droppedMarker = (MarkerFeature) event.getFeature();
    Coordinate startCoordinates = event.getStartCoordinate();
    Coordinate endCoordinates = event.getCoordinate();
    Notification.show("Marker \"" + droppedMarker.getId() + "\" dragged from " +
startCoordinates + " to " + endCoordinates);
});
```

Se ha utilizado el componente 'Map' de Vaadin, que nos ofrece un mapa completo y avanzado que enriquece nuestra interfaz.

```
H3 title = new H3("Address & Infos");
Accordion accordion = new Accordion();
accordion.getElement().setProperty("opened", true);
Span name = new Span("Land Lawyers");
Span email = new Span("Land.Lawyers@company.com");
Span phone = new Span("(+34) 655-912-871");
```

En la página de contacto se ha utilizado el componente H3 y Accordion para mostrar las informaciones del bufete de abogados como número, correo…

```
  Paragraph description = new Paragraph("if you are facing serious charges.it is
imperative that you seek assistance from a knowledgeable criminal defense lawyer as
as possible. An experienced attorney can guide you through the legal process.
Contact us for a better assistance.");
            description.addClassNames(LumoUtility.Margin.Vertical.MEDIUM,
LumoUtility.FontSize.MEDIUM,
LumoUtility.FlexDirection.COLUMN,
LumoUtility.TextAlignment.JUSTIFY,
LumoUtility.Margin.Horizontal.LARGE,
LumoUtility.Margin.Left.XLARGE
            );
    H3 title = new H3("Let's get started");
    title.getElement().getStyle().set("margin-left","80px");
    title.getElement().getStyle().set("margin-top","-6px");
```

```
H3 title = new H3("Reach Us");
title.getElement().getStyle().set("margin-left","32px");
Button bookButton2 = new Button();
bookButton2.setText("Book now");
```

Se ha añadido un párrafo informativo y un botón para reservar una cita.

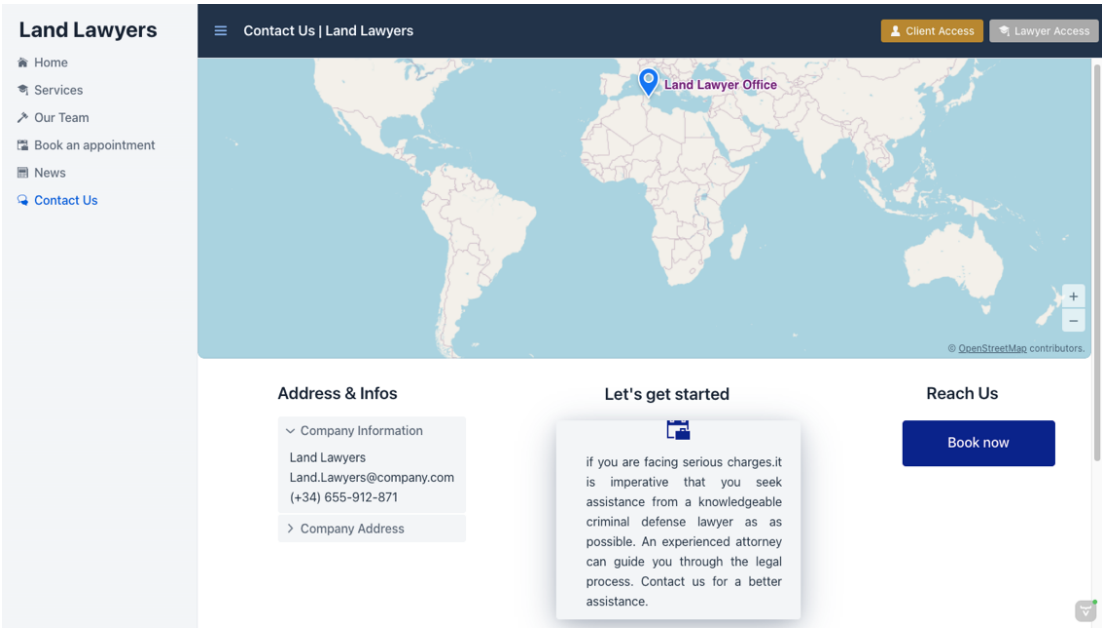


Figura 37 : Vista mapa y contacto

4.8 Dashboards

Para gestionar los datos dentro la aplicación se ha creado dashboards para todos los objetos de la aplicación, implementando operaciones CRUD (Read, creacte, update, delete).

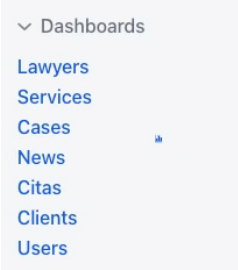


Figura 38 : Tablas de comando

Utilizando un Grid y un formulario, se han creado 2 clases distintas una para el formulario y el otro para la tabla, luego se importa la clase de formulario y la tabla, se toma como ejemplo las clases Abogado y User

```
private Grid<Abogado> grid = new Grid<>(Abogado.class);
```

```
public AbogadoGrid(@Autowired AvocatService avocatService) {
    this.avocatService = avocatService;
    this.form = new TeamForm();
    // Setup  grid with columns for title, date, and body
    grid.setColumns("firstName", "lastName", "email",  "phoneNumber", "speciality",
"numberBar","password", "position","description");
    grid.setItems(avocatService.findAll());

    grid.asSingleSelect().addValueChangeListener(evt -> editAbogado(evt.getValue()));

    form1 = new TeamForm();
    form1.addSaveListener(TeamForm.SaveEvent.class, this::saveAbogado);
    form1.addDeleteListener(TeamForm.DeleteEvent.class,this::deleteAbogado);
    form1.addCloseListener(TeamForm.CloseEvent.class,e -> closeEditor());
```

```
private void deleteAbogado(TeamForm.DeleteEvent event) {
    status = new Span();
    status.setVisible(false);
    ConfirmDialog dialog = new ConfirmDialog();
    dialog.setHeader("Delete This lawyer ?");
    dialog.setText("Are you sure you want to delete it?");
    dialog.setCancelable(true);
    dialog.setConfirmText("Delete");
    dialog.setConfirmButtonTheme("error primary");
    dialog.addConfirmListener(e -> {
        // Delete the news item after user confirms
        avocatService.delete(event.getAbogado());
        closeEditor();
        updateList();
        Notification notification = Notification
                .show("Lawyer deleted");
```

```
private void saveAbogado(TeamForm.SaveEvent evt) {
    avocatService.add(evt.getAbogado());
    closeEditor();
    updateList();

    Notification notification = Notification
            .show("Lawyer Added");
    notification.addThemeVariants(NotificationVariant.LUMO_PRIMARY);
    notification.setPosition(Notification.Position.BOTTOM_START);

}
```

Como indica el código anterior, se han configurado las tablas, añadiendo los datos del abogado y las funciones del CRUD como borrar y añadir…
Después de implementar estos elementos, obtendremos un diseño similar al siguiente:

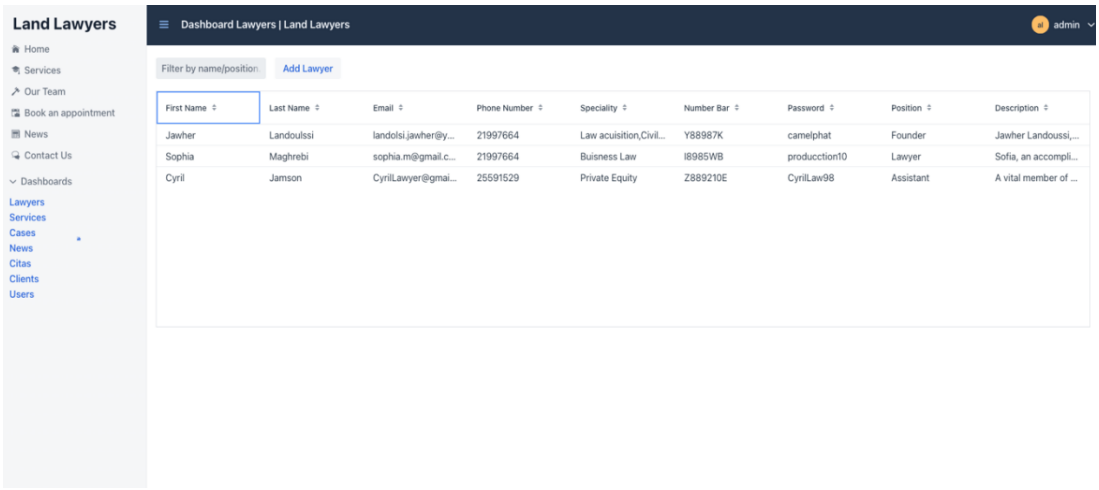


Figura 39 : Dashboard abogados

Incluyendo el formulario creado con el siguiente código:

```
private byte[] imageData; // Define imageData as an instance variable
TextField firstName = new TextField("First Name", "Write your first name here ");
TextField lastName = new TextField("Last Name");
EmailField email = new EmailField("Email" , " write your email here @");
PasswordField password = new PasswordField("Password");
TextField phoneNumber = new TextField("Phone Number");
TextField speciality = new TextField("Speciality");
TextField numberBar = new TextField("Bar Number");
TextField position = new TextField(" position");
TextArea description = new TextArea(" description","write a litlle paragraph to describe the
skills of the new lawyer");
Button save = new Button("Save");
Button delete = new Button("Delete");
Button close = new Button("Close");
```

Al pulsar a una línea de la tabla se abrirá un formulario. Si se trata de una operación de modificación, el formulario se abrirá con los datos existentes prellenados. En caso de agregar un nuevo abogado, el formulario se abrirá en blanco y estará listo para que se ingresen los datos.

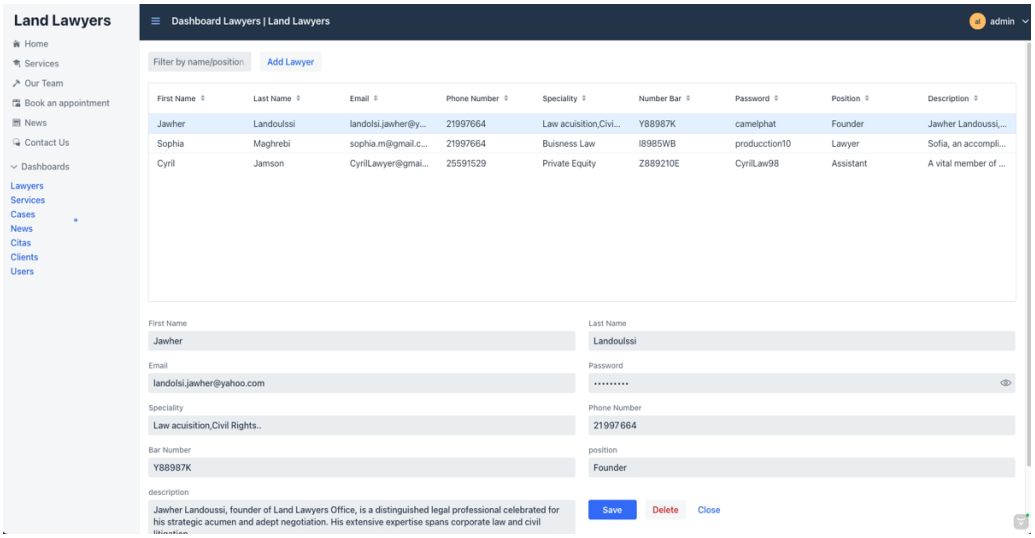


Figura 40 : Anadir, Borrar y Modificar abogados

En cuanto los casos y como todos los otros objetos se ha implementado un dashboard para gestionarlos.

```
grid.setColumns("title", "description", "state","creation_date");
grid.setItems(caseService.findAll());
grid.addColumn(service -> service.getService().getService()).setHeader("Service");
grid.addColumn(user -> user.getUser().getFirstName()).setHeader("User");
grid.addColumn(avocat -> avocat.getAbogado().getFirstName()).setHeader("Lawyer");
grid.getColumns().forEach(col -> col.setAutoWidth(true));
grid.addComponentColumn(this::createDocumentLinkRenderer)
        .setHeader("Document");
grid.asSingleSelect().addValueChangeListener(event ->
        editCase(event.getValue()));
```

Se han añadido los datos pertinentes del expediente, con el servicio relacionado, abogado y cliente.

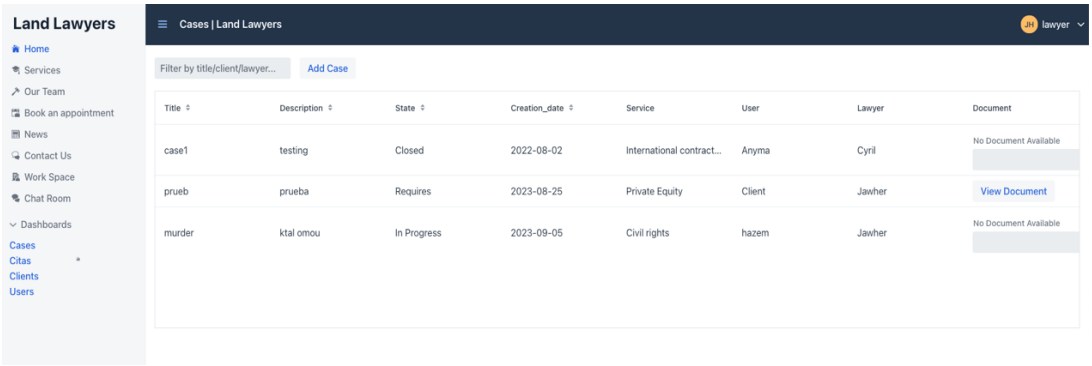


Figura 41 : Dashboard Casos

También se ha agregado una columna donde el abogado tiene la posibilidad de ver los documentos añadidos por el cliente, para conseguir esto se ha utilizado **PDF-Viewer** de vaadin directory añadiendo la dependencia siguiente.

```
<dependency>
    <groupId>org.vaadin.addons.componentfactory</groupId>
    <artifactId>vcf-pdf-viewer</artifactId>
    <version>2.7.2</version>
</dependency>
```

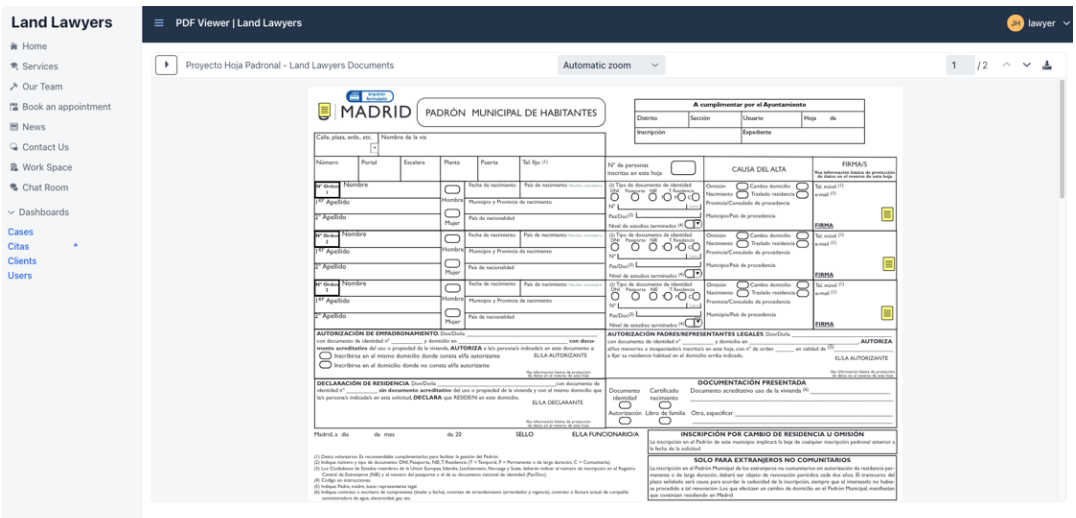


Figura 42 : Vista documento PDF

Se ha creado un panel de control (dashboard) para cada uno de los objetos: usuarios y citas, con el fin de facilitar la gestión de la aplicación desde su interior.

4.9 Página de seguimiento de caso

Para mejorar la experiencia de usuario, se ha creado un área para que los clientes puedan seguir sus expedientes.

```
VerticalLayout layout = new VerticalLayout();
Grid<Case> grid = new Grid<>(Case.class, false);
grid.addColumn(createEmployeeRenderer()).setHeader("Name")
        .setAutoWidth(true).setFlexGrow(0);
grid.addColumn(Case::getTitle).setHeader("Case")
        .setAutoWidth(true);
grid.addColumn(Case::getCreation_date).setHeader("Creation Date")
        .setAutoWidth(true);
grid.addColumn(createStateComponentRenderer()).setHeader("state")
        .setAutoWidth(true);
grid.addColumn(createLawyerRenderer()).setHeader("Lawyer")
        .setAutoWidth(true).setFlexGrow(0);
grid.addColumn(Case::getComment).setAutoWidth(true)
        .setHeader("Comment").getStyle().set("text-align", "end");
```

Se ha creado una tabla que contiene los datos más importantes del expediente. De esta manera cuando el abogado solicita algo, o hace cambio, se mostrara dicho cambio en la interfaz de usuario(cliente). manteniéndolo informado de manera oportuna.

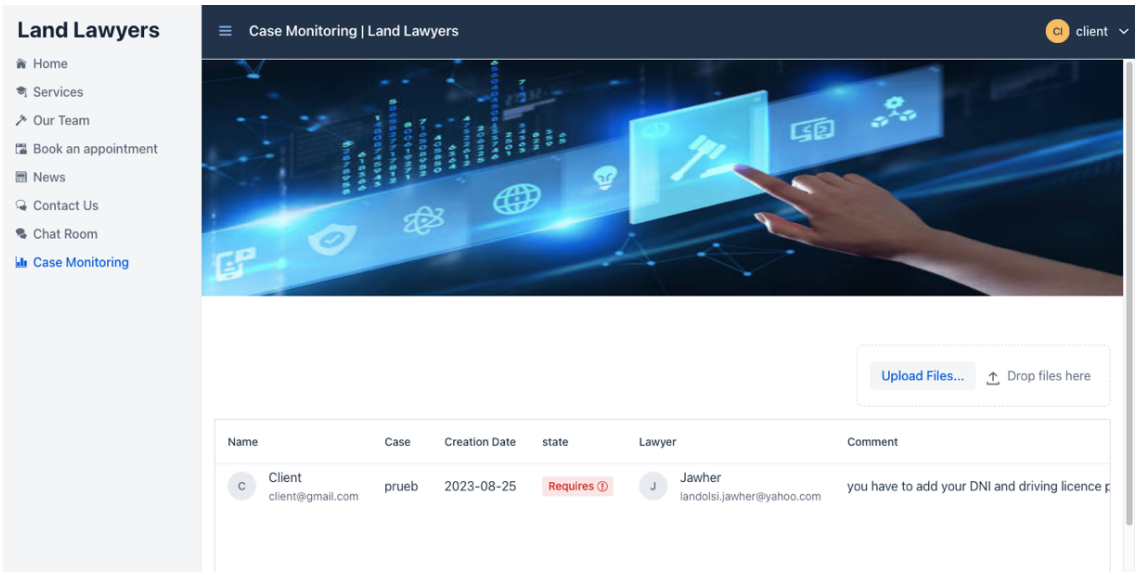


Figura 43 : Vista Seguimiento de caso

4.10 Página de Chat

Como uno de los objetivos de nuestro proyecto es facilitar la comunicación entre el abogado y su cliente, se ha creado una interfaz de chat, donde se pueden comunicar dentro la aplicación.

```
public CollaborationAvatarGroup createAvatarGroup(UserInfo userInfo) {
    CollaborationAvatarGroup avatars = new CollaborationAvatarGroup(
            userInfo, "chat");
    return avatars;
}
```

```
var messageInput = new CollaborationMessageInput(messageList);
messageInput.getStyle().set("margin-top", "100px");
messageInput.setWidthFull();
```

Se ha utilizado el add-on de vaadin **Collaborative engine** y sus componentes como **CollaborationMessageInput** y **CollaborationAvatarGroup.**

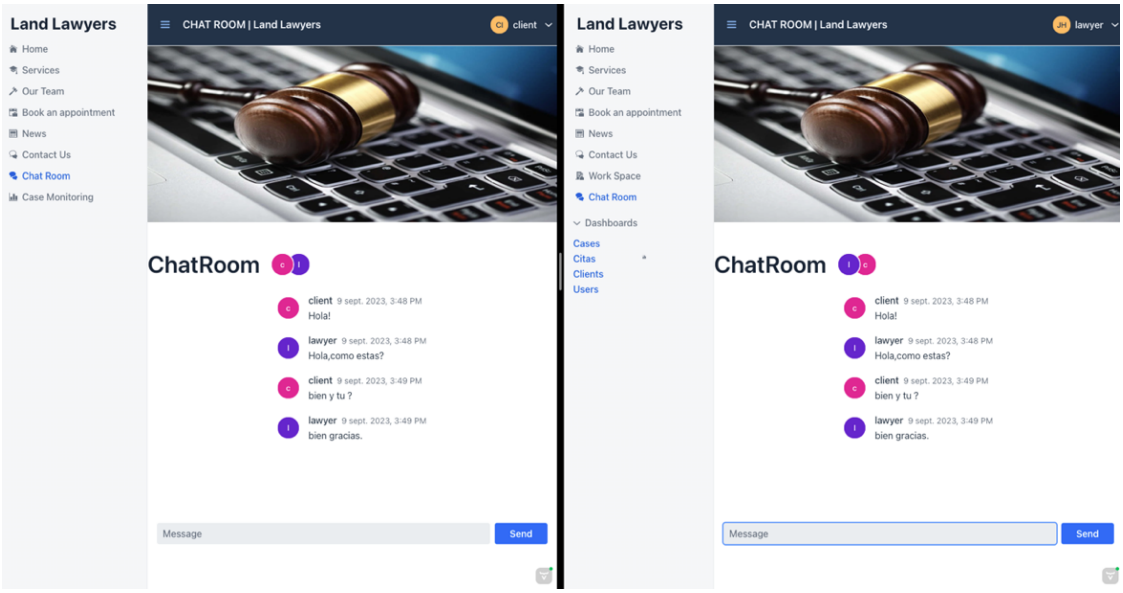


Figura 44 : Vista sala de chat

4.11 Área de trabajo

Con el objetivo de proporcionar todas las herramientas necesarias tanto a los usuarios generales como a los abogados, hemos implementado un espacio de trabajo que incluye una tabla de Excel y otros elementos estáticos. En esta sección, se proporcionará un detalle completo de estos elementos estáticos utilizados para gestionar el flujo de trabajo en el bufete.

Primero se ha utilizado el componente **Spreadsheet-basic** Para integrar una tabla Excel en nuestra aplicación, hemos añadido la correspondiente dependencia.

```
<dependency>
    <groupId>com.vaadin</groupId>
    <artifactId>vaadin-charts-flow</artifactId>
</dependency>
```

```
Spreadsheet spreadsheet = new Spreadsheet();
spreadsheet.setHeight("400px");
add(spreadsheet);
```

Luego se ha añadido el Componente **CHART** de tipo **SPLINE,** que nos indica el Rendimiento financiero y fechas claves en 2 Ejes.

```
Chart chart = new Chart (ChartType.SPLINE);
Configuration configuration = chart.getConfiguration();
configuration.getTitle().setText("Land Lawyer Financial Performance and Key
Dates");
configuration.getxAxis().setType(AxisType.DATETIME);
```

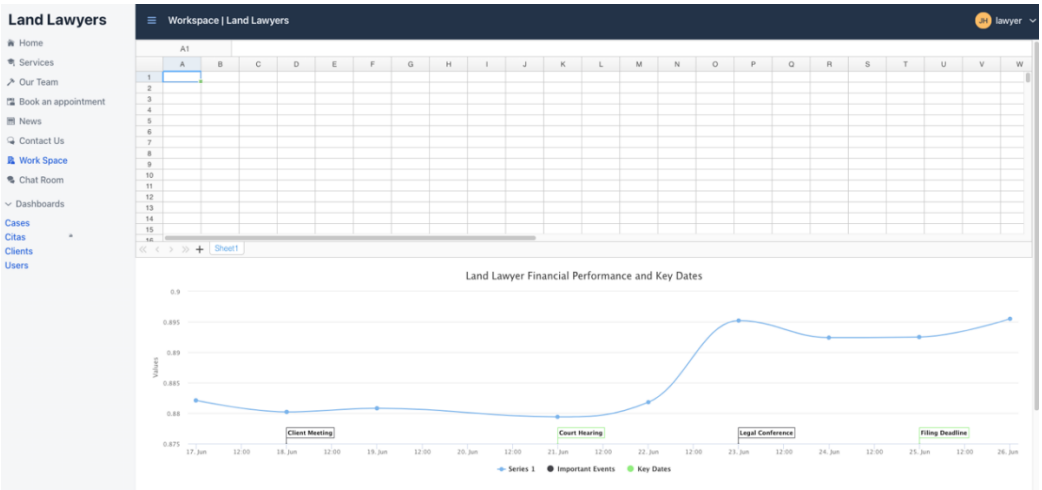


Figura 45 Vista Espacio de trabajo

Finalmente, hemos implementado el mismo componente CHART, pero esta vez en formato de gráfico tipo PIE. Primero, recuperamos una lista de usuarios con el rol 'client' y luego otra lista de usuarios con el rol 'user'. A partir de estas dos listas, calculamos el porcentaje de cada tipo de usuario y representamos esta información de forma gráfica en el gráfico de tipo PIE.

```
List<User> usersWithRoleUser = userService.findUserByIdRol(3);
List<User> usersWithRoleClient = userService.findUserByIdRol(4);

int totalUsers = usersWithRoleUser.size();
int totalClients = usersWithRoleClient.size();
System.out.print("****"+totalClients);
int total = totalUsers + totalClients;

double percentageUsers = ((double) totalUsers / total) * 100;
double percentageClients = ((double) totalClients / total) * 100;

DataSeriesItem userSlice = new DataSeriesItem("Non-Client Users ", percentageUsers);
DataSeriesItem clientSlice = new DataSeriesItem("Client Users", percentageClients);

series.add(userSlice);
series.add(clientSlice);
conf3.addSeries(series);

    VerticalLayout layout = new VerticalLayout();
    layout.add(chart, chart3);
    add(layout);
}
```

Con este componente gráfico de tipo PIE, se representa visualmente el porcentaje de usuarios que son clientes con expediente en el despacho y aquellos que son usuarios no clientes y han realizado solo una cita sin continuar con el despacho. Esta representación en un círculo de porcentaje permite una rápida evaluación de la situación y ayuda a determinar si el despacho está siguiendo el rumbo deseado. Es una herramienta estática útil para obtener una instantánea de la composición de la base de usuarios y su nivel de compromiso con el despacho.

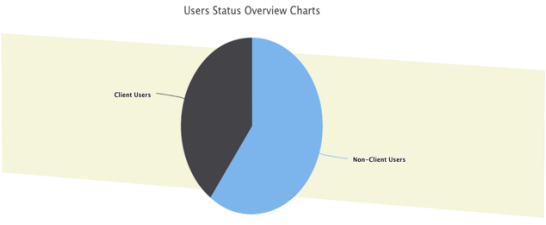


Figura 46 : Análisis Visual: Composición de Clientes y Usuarios en el Despacho Legal

5. Autenticación y Autorización

uno de los pillares de una aplicación web es la seguridad, es necesario tener una forma de proteger y controlar el acceso de un sitio web. Por eso existe Spring Security, que nos ofrece una personalización y configuración especifica y sencilla.

Autenticación: es la verificación del usuario antes de entrar.
Autorización: tipos de permisos que tienen los usuarios

5.1 Configuración Spring Security

La primera etapa es agregar las dependencias necesarias de Spring Security en el fichero de dependencias pom.xml como muestra la imagen a continuación.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Tras haber añadir las dependencias, se ha creado un paquete llamado security que contiene distinas clases de configuración.

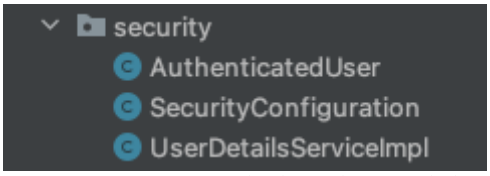


Figura 47 : Carpeta de clases de seguridad

5.1.1 Clase Authenticated User

La clase authenticated User es la clase encargada de la recuperación de credenciales del usuario loggeado.

Esta clase actúa como enlace entre los procesos de autenticación de Spring Security y los datos de usuario de la aplicación.

Es fundamental para gestionar de forma segura las sesiones de usuario, recuperar la información del usuario y gestionar el cierre de sesión del usuario.

```
@Component
public class AuthenticatedUser {

    private final UserRepository userRepository;
    private final AuthenticationContext authenticationContext;

    public AuthenticatedUser(AuthenticationContext authenticationContext, UserRepository
userRepository) {
        this.userRepository = userRepository;
        this.authenticationContext = authenticationContext;
    }
    @Transactional
    public Optional<User> get() {
        return authenticationContext.getAuthenticatedUser(UserDetails.class)
                .map(userDetails ->
userRepository.findByUsername(userDetails.getUsername()));
    }
    public void logout() {
        authenticationContext.logout();
```

```
    }
}
```

5.1.2 Clase Security Configuración

La clase SecurityConfiguration es el punto central de la configuración de la seguridad.

Esta clase orquesta elementos de seguridad esenciales como la gestión de sesiones con una política de una sesión por usuario y una página de expiración de sesión que puede personalizarse.

También especifica las reglas de permisos, que permiten el acceso no autenticado a determinados recursos de imagen. También configura la autenticación con un servicio personalizado de detalles de usuario (UserDetailsServiceImpl) y utiliza BCrypt para la codificación segura de contraseñas.

```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration extends VaadinWebSecurity {
    private final UserDetailsServiceImpl userDetailsServiceImpl;

    public SecurityConfiguration(UserDetailsServiceImpl userDetailsServiceImpl) {
        this.userDetailsServiceImpl = userDetailsServiceImpl;
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .sessionManagement(sessionManagement ->
                        sessionManagement
                                .sessionCreationPolicy(SessionCreationPolicy.ALWAYS) // Create
a session if necessary
                                .maximumSessions(1) // only one session by user
                                .expiredUrl("/session-expired") //  displayed it page when the
session expire
                );
            http.authorizeHttpRequests(auth ->
                    auth.requestMatchers(

AntPathRequestMatcher.antMatcher(HttpMethod.GET,"/images/**")).permitAll());  // <3>
            super.configure(http);
            setLoginView(http, LoginView.class); // <4>
        }

    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsServiceImpl)
                .passwordEncoder(passwordEncoder());
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

5.1.3 Clase UserDetailsServiceImpl

Esta clase Se encarga de recuperar la información del usuario de la base de datos. Cuando se invoca el método loadUserByUsername, se busca en la base de datos un usuario con el nombre de usuario proporcionado.
Si se encuentra un usuario, se crea un objeto UserDetails utilizando el nombre de usuario, la contraseña y las autoridades (roles) del usuario.

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
```

```
    private final UserRepository userRepository;

    public UserDetailsServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("no user present with this username" +
username);
        } else {
            return new
org.springframework.security.core.userdetails.User(user.getUsername(),user.getPassword(),
                    getAuthorities(user));
        }
    }
    private static List<GrantedAuthority> getAuthorities(User user) {
        System.out.println("  msajel " + user.getAuthority().getRol());
        return Collections.singletonList(new SimpleGrantedAuthority("ROLE_" +
user.getAuthority().getRol()));
    }
}
```

5.2 Creación Vista Inicio de sesión

Tras haber implementar las clases de autentificación, y las configuraciones necesarias para el inicio de sesión, se ha creado una vista de inicio de sesión que facilita el acceso a la plataforma.

Primero se han creado dos botones de accesos diferentes, uno para los clientes y el otro para los abogados.

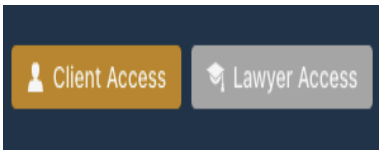


Figura 48 : Botones de acceso

Login de Abogado

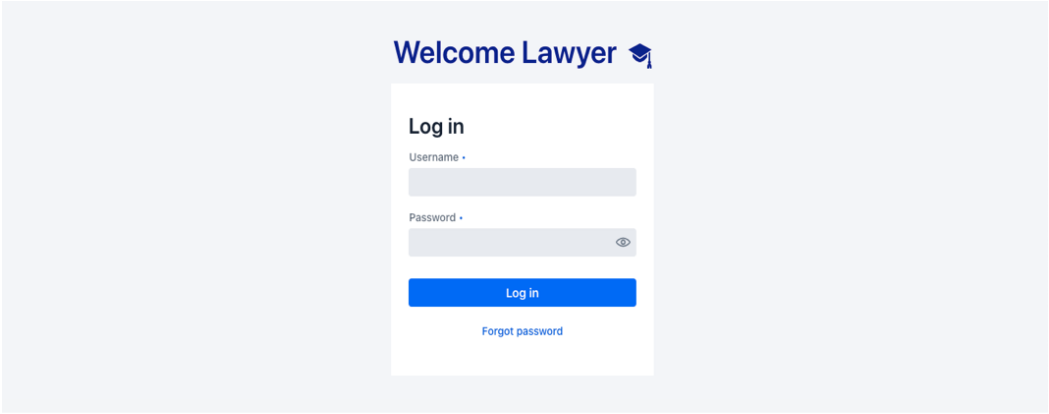


Figura 49 : Pagina Inicio de sesión Abogados

Login de Cliente

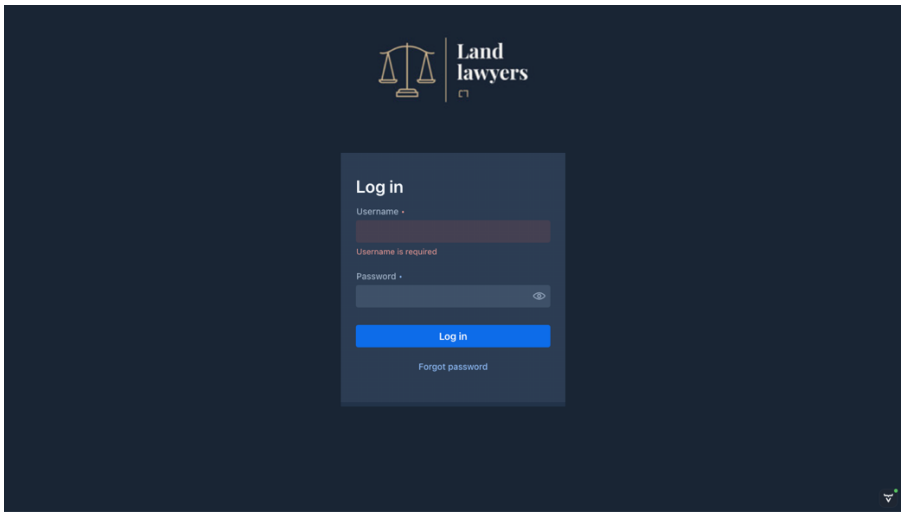


Figura 50 : Pagina Inicio de sesión clientes

5.3 Configuración Autorización

Para filtrar las vistas y controlar el acceso en función de los roles de usuario, vamos a aprovechar las anotaciones de Spring Security en conjunto con Vaadin. Estas anotaciones incluyen:

- **@AnonymousAllowed:** permite a cualquier persona navegar a la vista sin ningún tipo de autenticación o autorización.

Los usuarios no autenticados y aquellos que no tienen roles predeterminados pueden acceder a estas vistas.

```
@AnonymousAllowed
public class ServiceListView extends VerticalLayout {
```

```
@AnonymousAllowed
public class ContactView extends Div {
```

```
@AnonymousAllowed
public class TeamView extends VerticalLayout {
```

```
@AnonymousAllowed
public class HomePage extends VerticalLayout {
```

```
 @AnonymousAllowed
public class NewsView extends VerticalLayout {
```

```
@AnonymousAllowed
public class CitaForm extends Composite<Div> {
```

- **@Permit All**: Permite a cualquier usuario autenticado navegar a la vista.

La página de chat está diseñada exclusivamente para usuarios autenticados.

```
@PermitAll
public class ChatView extends VerticalLayout {
```

- **@RolesAllowed:** Concede acceso a los usuarios que tienen los roles especificados en el valor de la anotación.

Por ejemplo, tenemos la página de seguimiento de casos que está dedicada solamente a los usuarios clientes.

```
@RolesAllowed("CLIENT")
public class CaseTrackerView extends VerticalLayout {
```

Existen página dedicadas a los abogados como por ejemplo la vista del área de trabajo que solo los abogados tienen el acceso a ellos.

```
@RolesAllowed("LAWYER")
public class WorkSpace extends Div {
```

En la mayoría de los casos, los paneles de control (dashboards) están reservados exclusivamente para los administradores, quienes tienen privilegios especiales para acceder y gestionar las informaciones críticas y configuraciones avanzadas del sistema.

```
@RolesAllowed("ADMIN")
public class AbogadoGrid extends VerticalLayout {
```

Hay algunos que son compartidos entre abogados y administrador de la página como la tabla de clientes, citas, casos…

```
@RolesAllowed({"ADMIN","LAWYER"})
public class CitaGrid extends VerticalLayout {
```

```
@RolesAllowed({"ADMIN",                                                    "LAWYER"})
public class CaseGrid extends VerticalLayout {
```

```
@RolesAllowed({"ADMIN","LAWYER"})
public class DashClients extends VerticalLayout {
```

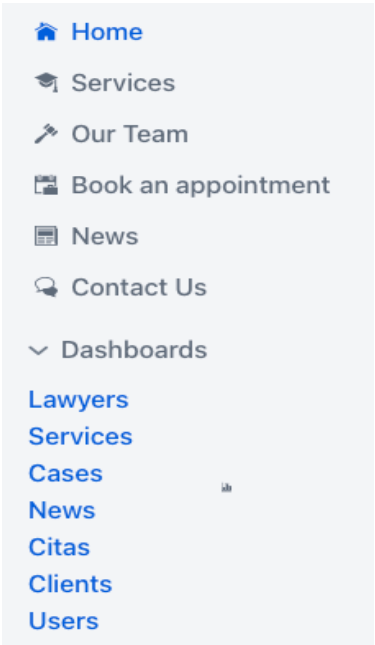
5.4 Configuración menú de navegación

Se ha configurado el menú de navegación adecuadamente según el rol del usuario conectado, es decir cada usuario tiene rutas específicas de navegación.

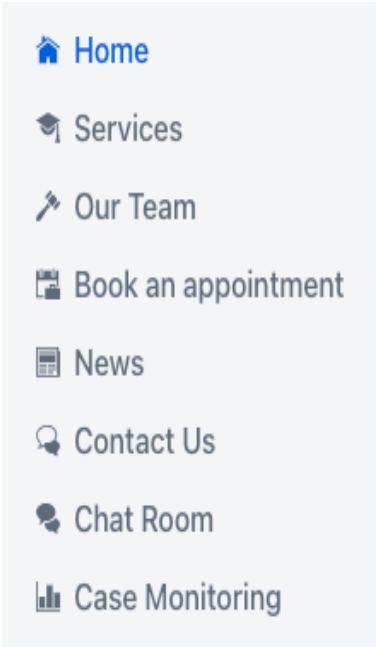
```
  if (authenticatedUserOptional.isPresent()) {
      User authenticatedUser = authenticatedUserOptional.get();
      Authority authority = authenticatedUser.getAuthority();
      if(authority != null &&  "LAWYER".equals(authority.getRol()) ) {
          Tab WorkingTab = createTab("Work Space", WorkSpace.class, new
Icon(VaadinIcon.WORKPLACE));
          tabs.add(WorkingTab);
      }
      if(authority != null && "CLIENT".equals(authority.getRol()) ||
"LAWYER".equals(authority.getRol()) ) {
          Tab ChatRoom = createTab("Chat Room", ChatView.class, new
Icon(VaadinIcon.COMMENTS));
          tabs.add(ChatRoom);
}
      if(authority != null && "CLIENT".equals(authority.getRol()) ) {
          Tab CaseTracker = createTab("Case Monitoring", CaseTrackerView.class, new
Icon(VaadinIcon.BAR_CHART_H));
          tabs.add(CaseTracker);
      }

      if (authority != null && "ADMIN".equals(authority.getRol())) {
```

```
            //  component only for the admin user
            Details analyticsDetails = createDetails("Dashboards",
                    createStyledAnchor("http://localhost:8080/Dashboard_abogados", " Lawyers"),
                    createStyledAnchor("http://localhost:8080/dashServices", " Services"),
                    createStyledAnchor("http://localhost:8080/Case-grid", " Cases"),
                    createStyledAnchor("http://localhost:8080/Dashboard_News", " News"),
                    createStyledAnchor("http://localhost:8080/Dashboard_appointments", "
Citas"),
                    createStyledAnchor("http://localhost:8080/dashClients", " Clients"),
                    createStyledAnchor("http://localhost:8080/dashboard_users", " Users"));

            Tab analyticsTab = createTab("", AbogadoGrid.class, new
Icon(VaadinIcon.BAR_CHART_H));
            analyticsTab.addComponentAsFirst(analyticsDetails);
            tabs.add(analyticsTab);

        } else if ( "LAWYER".equals(authority.getRol())) {
            Details analyticsDetails = createDetails("Dashboards",
                    createStyledAnchor("http://localhost:8080/Case-grid", " Cases"),
                    createStyledAnchor("http://localhost:8080/Dashboard_appointments", "
Citas"),
                    createStyledAnchor("http://localhost:8080/dashClients", " Clients"),
                    createStyledAnchor("http://localhost:8080/dashboard_users", " Users"));

            Tab analyticsTab = createTab("", AbogadoGrid.class, new
Icon(VaadinIcon.BAR_CHART_H));
            analyticsTab.addComponentAsFirst(analyticsDetails);
            tabs.add(analyticsTab);
        }
```

Admin



Abogado



Client

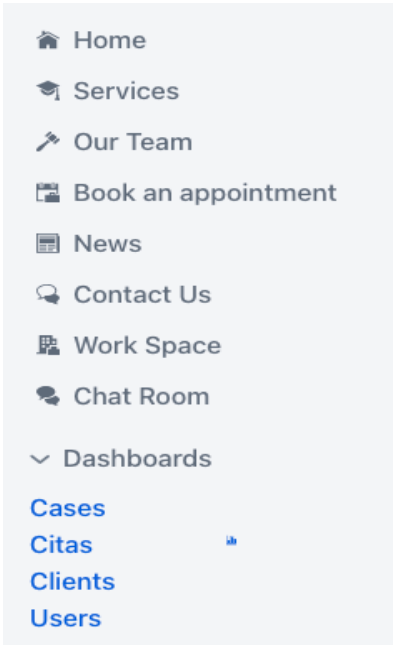


Figura 51 : Menú Lateral Admin. *Figura 52 : Menú Lateral Abogado.* *Figura 53 : Menú Lateral Cliente*

6. Instalación de la aplicación

6.1 PWA INSTALLATION

Las Aplicaciones Web Progresivas (PWA) son aplicaciones web modernas que reproducen la experiencia de usuario asociada a menudo con los programas nativos. Las tecnologías de aplicaciones web progresivas (PWA) desempeñan un papel crucial en el aumento del rendimiento de las aplicaciones a través de la mejora de la velocidad, la fiabilidad y el compromiso del usuario.

Tienen la capacidad de instalarse fácilmente en una amplia gama de dispositivos móviles y ordenadores de escritorio, y son compatibles con los navegadores web que las soportan. Además, las Aplicaciones Web Progresivas (PWA) tienen el potencial de alcanzar la inclusión en destacados mercados de aplicaciones como Microsoft Store y Google Play Store, ampliando así su alcance y disponibilidad a una mayor base de usuarios.

Los elementos fundamentales que impulsan las tecnologías de las Aplicaciones Web Progresivas (PWA) incluyen el ServiceWorker, que es un archivo JavaScript worker responsable de gestionar el tráfico de red e implementar un control de caché personalizado. Además, el Web Application Manifest, un archivo JSON, designa la aplicación web como instalable, de forma similar a las aplicaciones convencionales.

Para poder instalar la aplicación en varios dispositivos. Es necesario agregar la anotación @PWA in el punto de entrada de la aplicación Application.java

```
@SpringBootApplication
@PWA (name = "Land Lawyers", shortName = "LL" , offlineResources = {"images/icon.png" , "./images/offline.png"}
@Theme(value = "landlawyersapp")
public class Application implements AppShellConfigurator {


    start.vaadin.com
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);

    }

}
```

Figura 54 : Configuracion PWA

- La anotación @PWA, cree el ServiceWorker y archivo manifest
- Name: Es el nombre completo de la aplicación en el archivo manifest
- shortName: La longitud del texto debe ser concisa para que quepa en el espacio designado cuando se coloque como icono, y se recomienda que no supere un máximo de 12 caracteres.

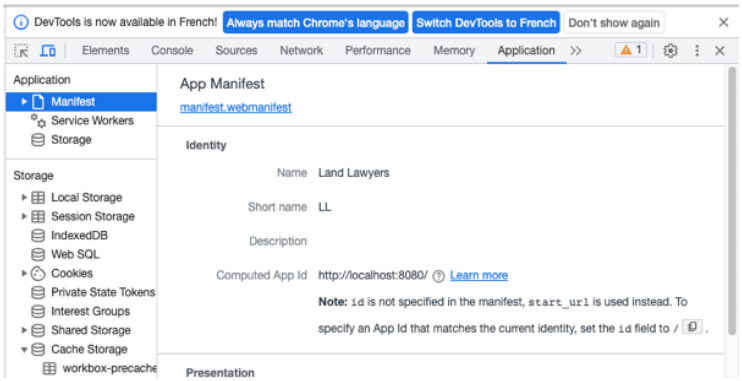


Figura 55 : Archivo Manifest

Tras haber configurado la anotación y añadir los recursos necesarios, se muestra un icono que nos permite descargar la aplicación en nuestro ordenador.

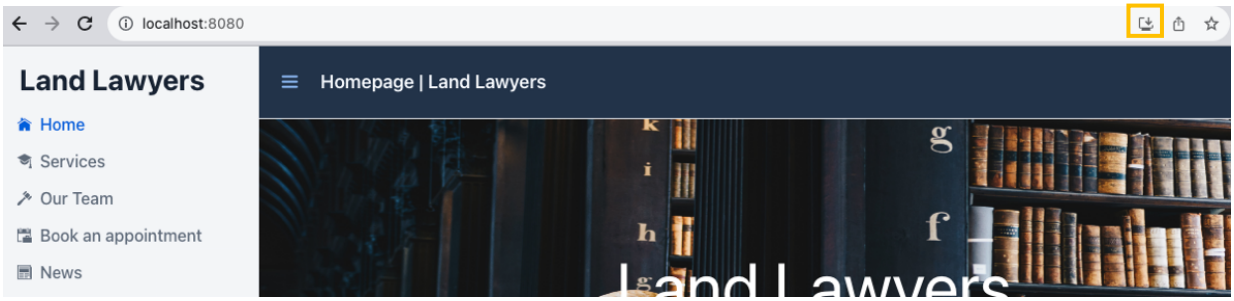


Figura 56 : icono de descarga

Al pulsar el icono, se descargará la aplicación en nuestro local, como muestra la imagen siguiente.

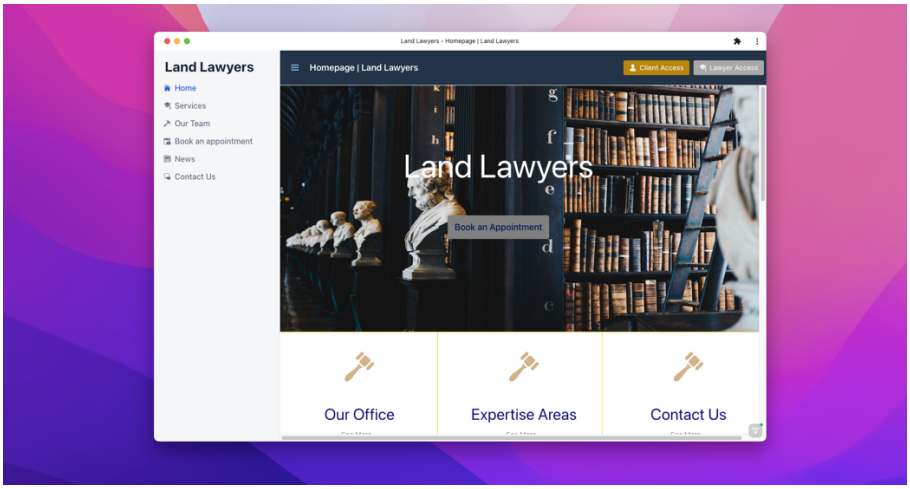


Figura 57 : Aplicación instalada

De esta manera, la aplicación ha sido descargada correctamente y se encuentra junto con todo el demás software instalados en nuestro dispositivo

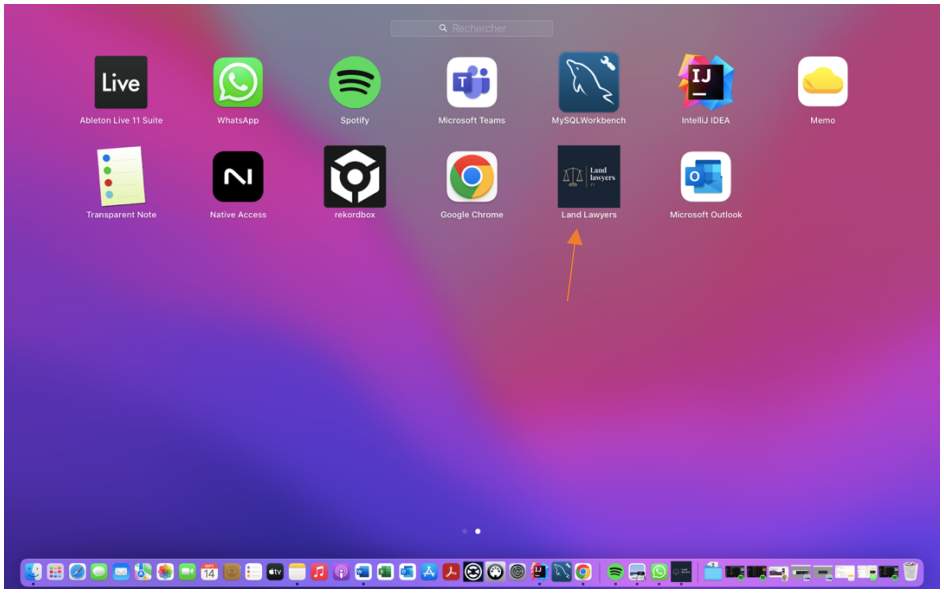


Figura 58 : Descarga Exitosa

VII. Manuel de despliegue

En este apartado se explica en detalles como poner en marcha el software completo desde la base de datos hasta la parte front-end.

Nota: Hay que asegurar que se han instalado GIT, JDK, MySql correctamente para hacer esta parte con éxito.

1. Clonar el proyecto desde github

El primer paso consistirá en clonar el proyecto desde el repositorio de GitHub. De esta manera, obtendremos el código fuente en nuestro entorno local.
Para descargar el proyecto existen 3 opciones

- Descárgalo en un fichero zip
- entrar a este enlace donde se encuentra el repositorio: https://github.com/landolsi98/Land_LawyersApp
- Abrir directamente con Visual Studio

Luego se abre la rama «final-version», una vez allí, se ejecuta el comando a continuación: git clone https://github.com/landolsi98/Land_LawyersApp.git

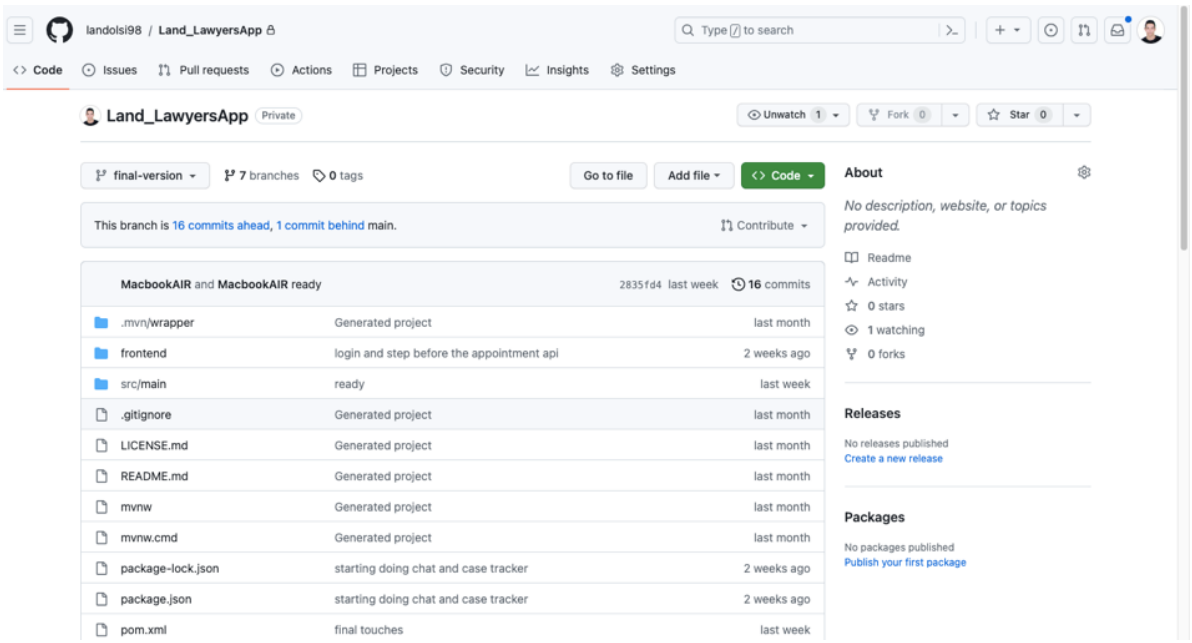


Figura 59 : Repositorio github

2. Implementación base de datos.

El primer paso consiste en implementar la base de datos localmente. Para lograrlo, se encuentra una carpeta llamada 'Database' dentro del repositorio del proyecto, la cual contiene un archivo llamado 'LandLawyerScriptDB'. Posteriormente, abrimos dicho archivo y lo generamos.

Una vez que tengamos la base de datos en nuestro entorno local, procedemos a abrir el archivo de propiedades de la aplicación y añadimos las siguientes líneas

```
spring.datasource.url=jdbc:mysql://localhost:3306/landb
spring.datasource.username=root
spring.datasource.password=admin
```

VIII. Manual de Usuario

1. Acceso Administrador

El administrador de la página tiene acceso a todas las vistas en la plataforma, ya que ocupa un puesto de alto nivel y debe gestionar toda la página

El administrador entra con el botón **acceso abogado** que está situado en el encabezado 'header'.

Se usa las credenciales: **username:admin / password : a**

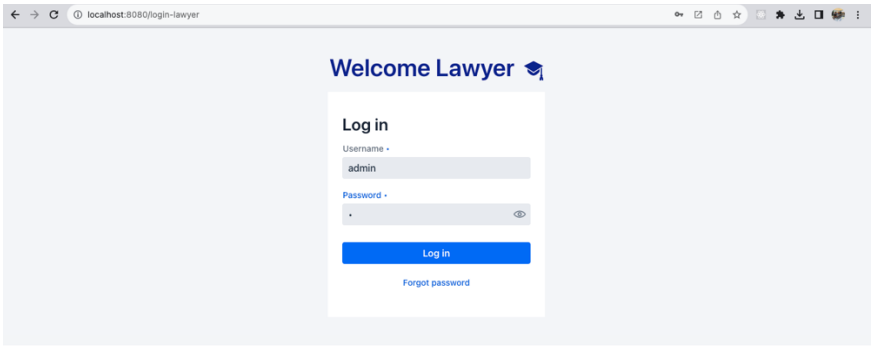


Figura 60 : Inicio de session Admin

Una vez el usuario ha tenido acceso con existo, se le abrirá la página inicial con su nombre de usuario y su avatar con la primera letra de su nombre de usuario y se mostraran todas las rutas de navegación.

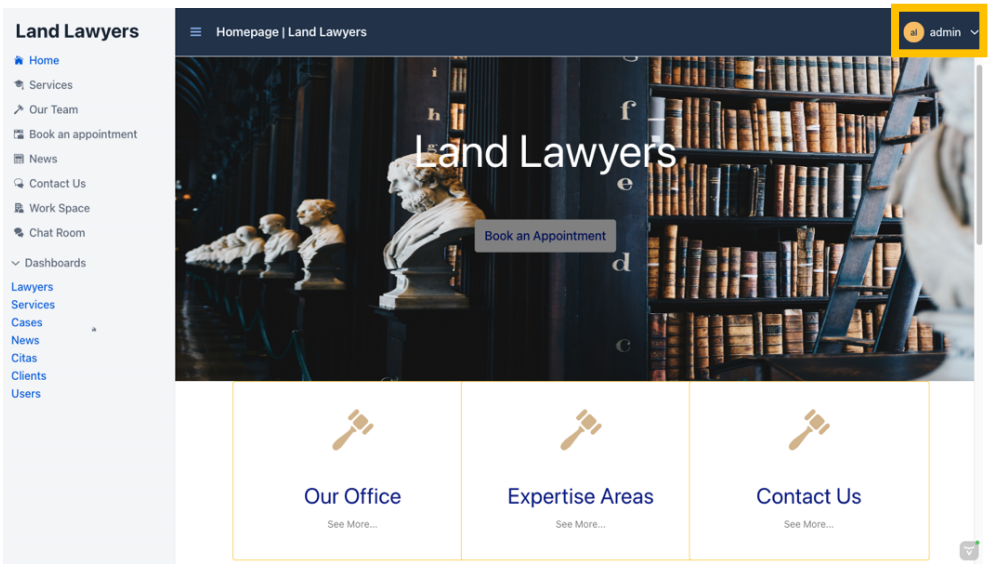


Figura 61 : Página de inicio / Acceso Admin

Dado que el administrador es el fundador del bufete, tiene la capacidad de gestionar a los abogados que trabajan allí. Esta gestión se llevará a cabo en la siguiente pantalla del panel de control.

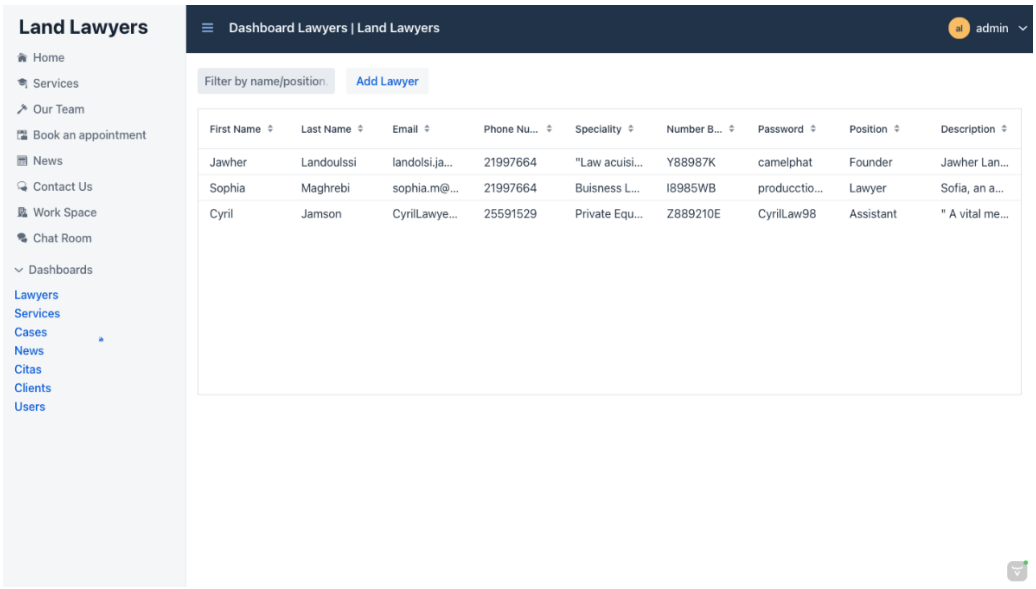


Figura 62 : Dashboard Abogados / Acceso Admin

Se puede buscar un abogado por su nombre o su puesto en el bufete

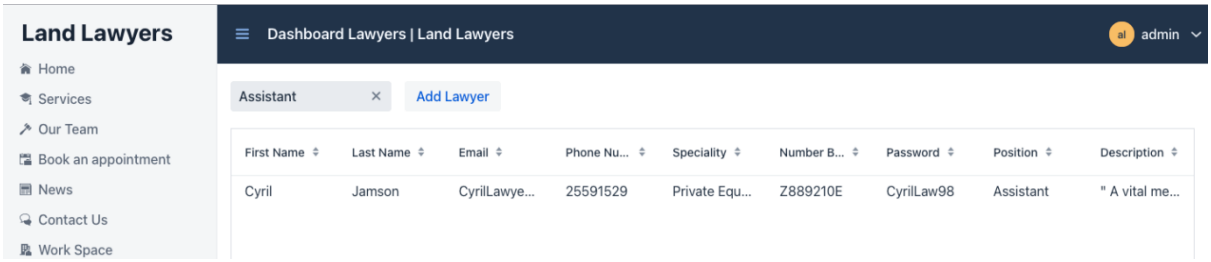


Figura 63 : Buscar Abogado

También el administrador puede añadir un nuevo abogado. Para modificar o borrar un abogado, hay que pulsar a un abogado directamente, se abrirá un formulario justo debajo de la tabla como muestra la imagen a continuación.

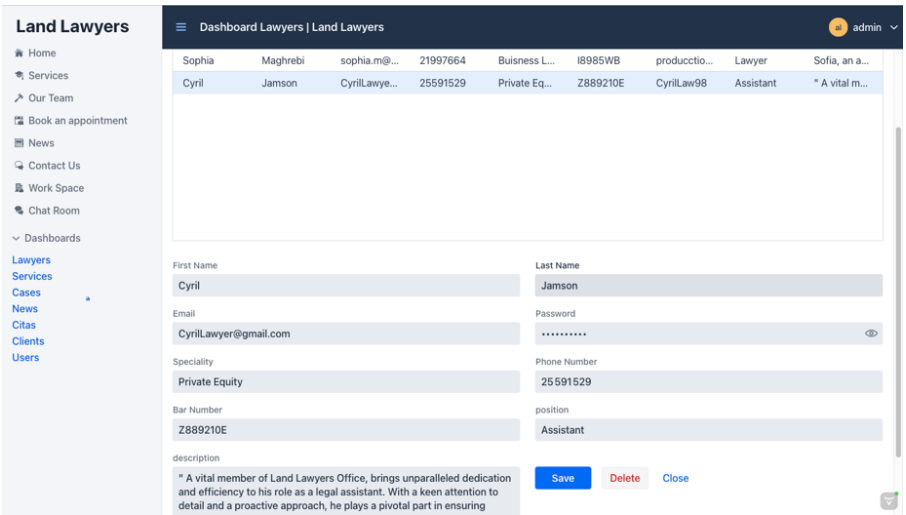


Figura 64 : Crud Abogados / Acceso Admin

Antes de borrar se tiene que confirmar desde el dialogo (pop-up).

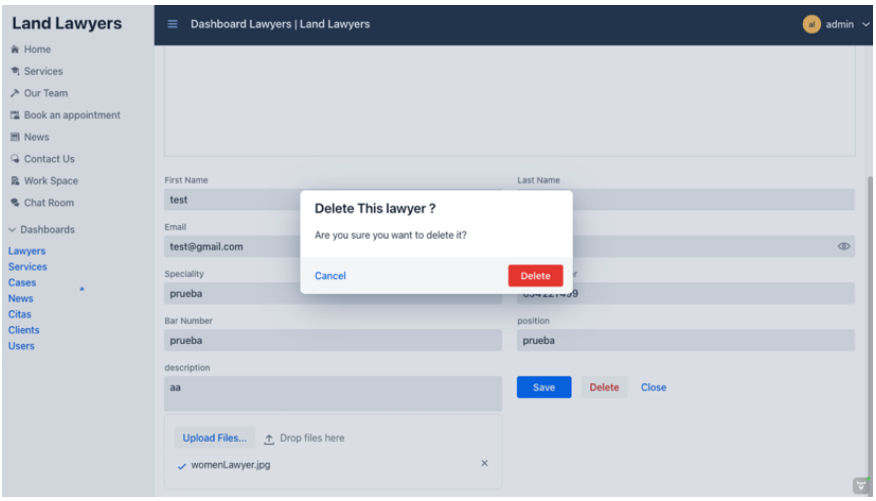
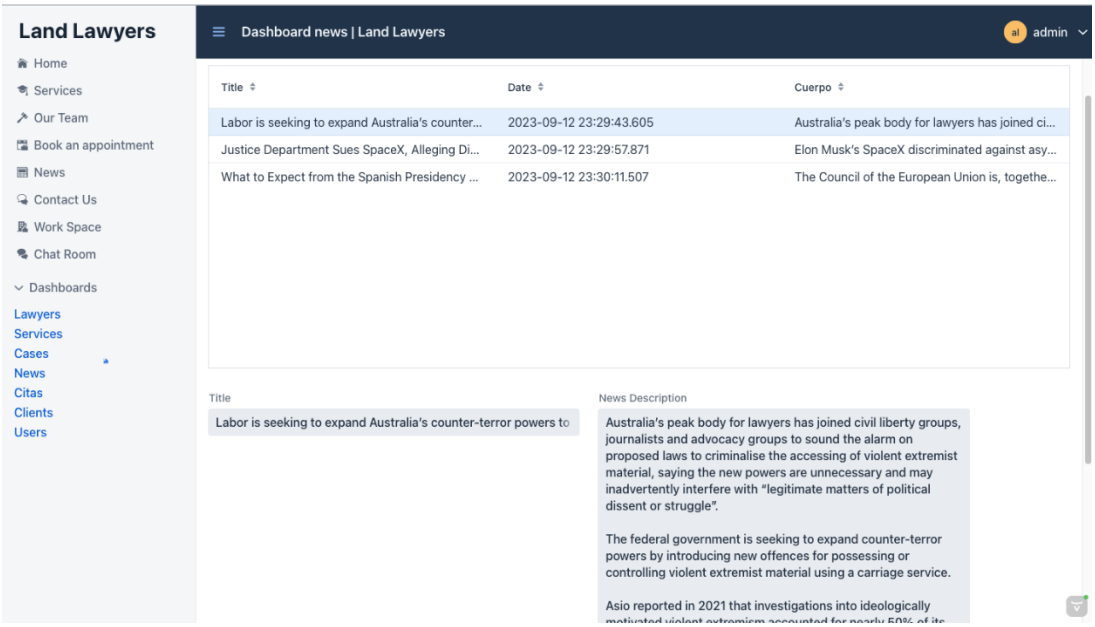


Figura 65 : Dialogo confirmación borrar

Solo los administradores tienen la posibilidad de gestionar las noticias y los servicios, las noticias se gestionan de la misma manera, como muestra la siguiente imagen:



En cuanto la fecha la cita, indica la fecha y el tiempo lo cual se ha agregado la noticia en la plataforma.
No se ha creado un campo a la fecha, ya que se rellena automáticamente según la fecha y la hora actual.

2. Acceso Abogado

Los abogados tienen la misma página de inicio de sesión como el administrador de la página, pulsando al botón Acceso abogado.

Las credenciales de prueba de un abogado serían las siguientes: username: lawyer / contraseña: abogado como se muestra en la imagen siguiente.
Con el icono dentro del campo de contraseña, puedes ocultar o mostrar la contraseña.

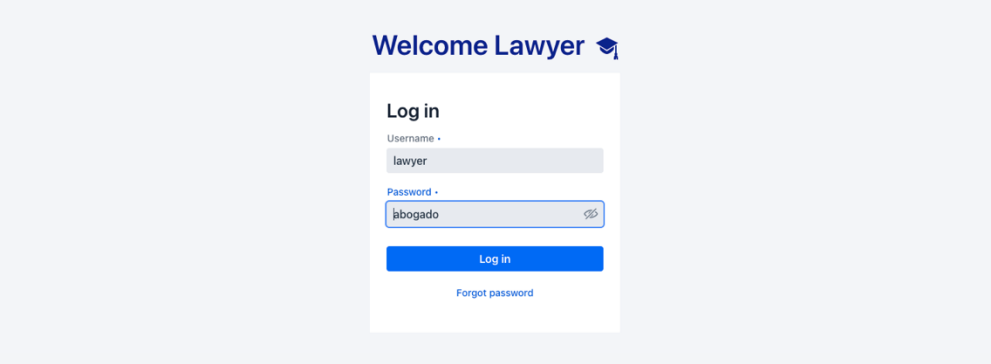


Figura 66 : Inicio de session / Acceso Abogado

Tras agregar las credenciales con éxito, se abre la página de inicio con el nombre de usuario y su avatar.
Como indica la imagen a continuación, los abogados tienen la posibilidad solo de gestionar los casos, citas, clientes y usuarios.

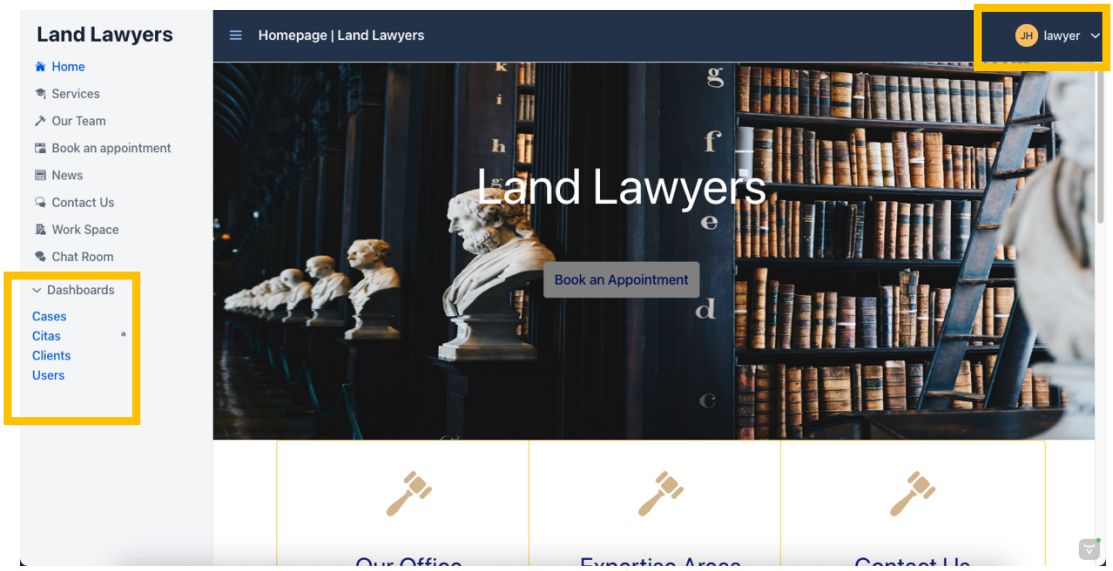


Figura 67 : Pagina de Inicio / Acceso Abogado

Los abogados tendrán acceso a un espacio de trabajo exclusivo que les brindará una visión completa del funcionamiento del bufete. En este espacio, encontrarán una tabla de Excel detallada y gráficos interactivos que representan de manera visual y clara el flujo de trabajo. Esta herramienta les permitirá realizar un seguimiento eficiente de los casos, identificar áreas de mejora y tomar decisiones informadas para optimizar la gestión de su trabajo en el bufete.

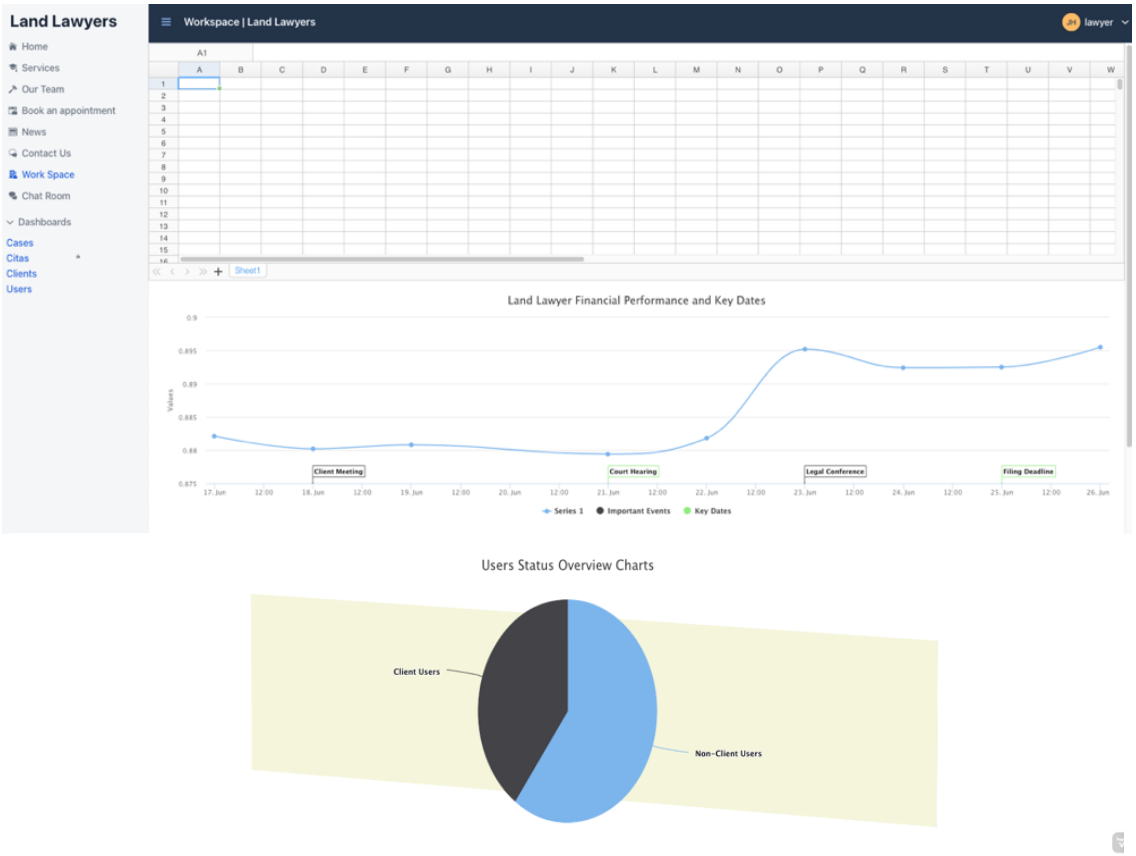


Figura 68 : Vista Espacio de Trabajo / Acceso Abogado

Va a entrar al dashboard de cita para ver las citas disponibles:

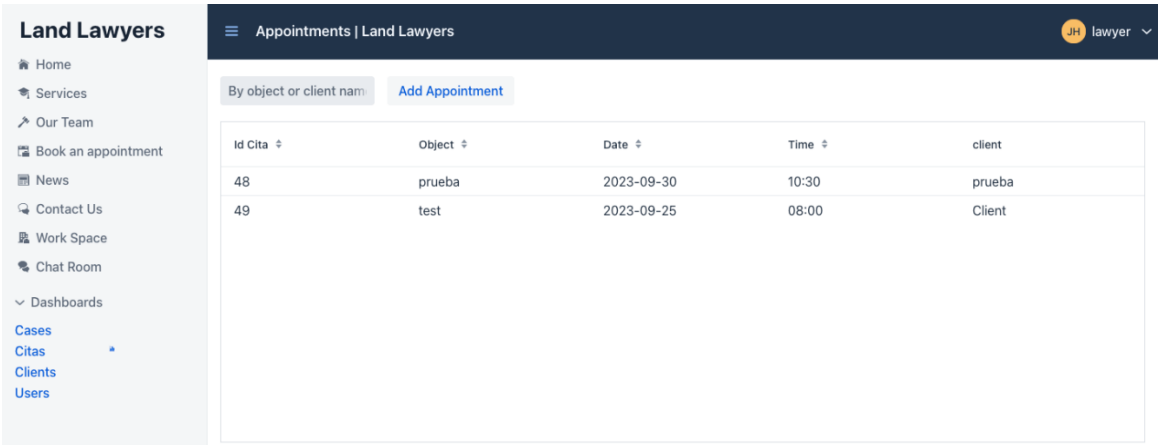


Figura 69 : Dashboard Citas / Acceso Abogado

Se puede filtrar las citas por objeto o por nombre del cliente de dicha cita.

También se puede filtrar por antiguad de tiempo o fecha utilizando las flechas.

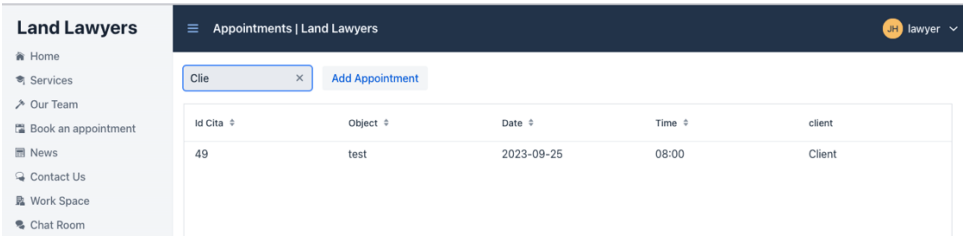


Figura 70 : Filtrar Cita / Acceso Abogado

Después de programar la cita con el cliente y cuando se va a abrir un expediente en el despacho, la primera etapa consiste en que el abogado registra al usuario para que se convierta en cliente

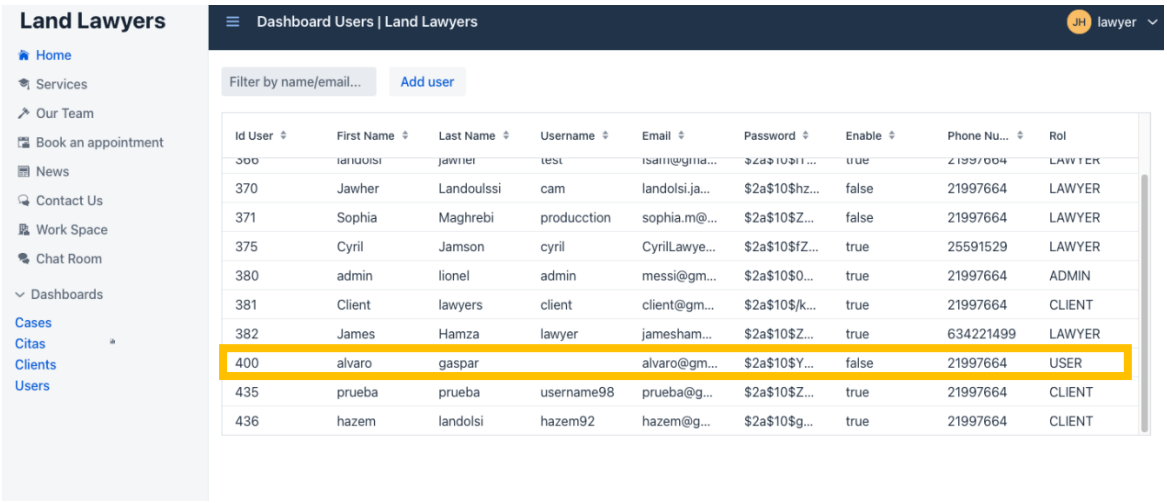


Figura 71 Usuario no-cliente

Los datos de un usuario se almacenan automáticamente cuando reserva una cita en la plataforma. Solo hay que hacer algunas modificaciones en este escenario, como darle un nombre de usuario y contraseña y cambiar su rol de "usuario" a "cliente".

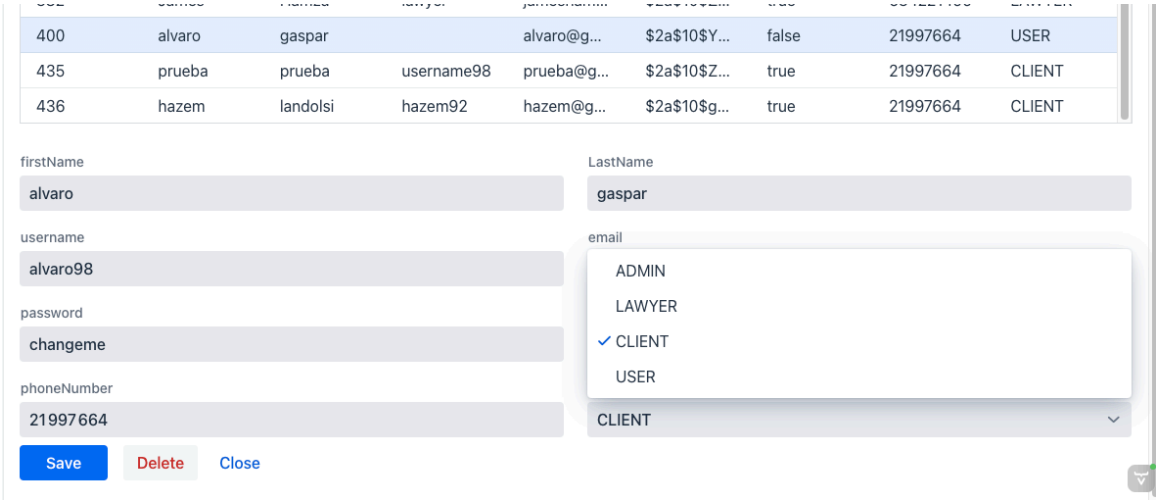


Figura 72 : Actualizar Usuario a Cliente

① Una de las tareas del abogado, es cambiar el estado del expediente, dejar el cliente siempre actualizado, esto se puede lograrlo desde el dashboard de los casos, cambiando el estado y si por ejemplo falta una hoja o ha cambiado algo, el abogado tiene la posibilidad de escribir un comentario al cliente mediante el formulario siguiente.

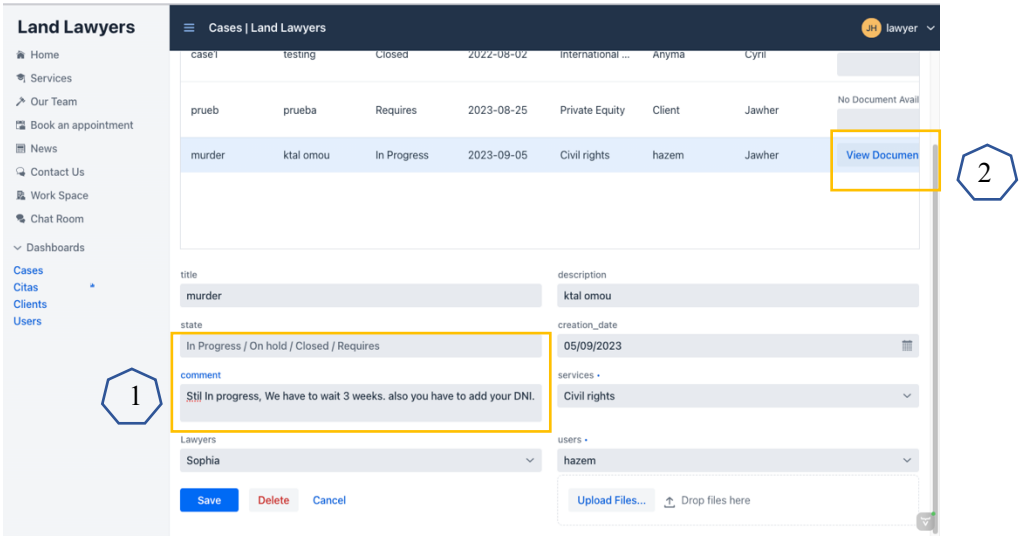


Figura 73 : Gestion de los casos / Acceso Abogado

② En la columna document, si existe un documento agregado por el abogado o el cliente, se encuentra un botón “View Document” que ofrece el acceso al documento pdf, si no se muestra el texto “No Document available”

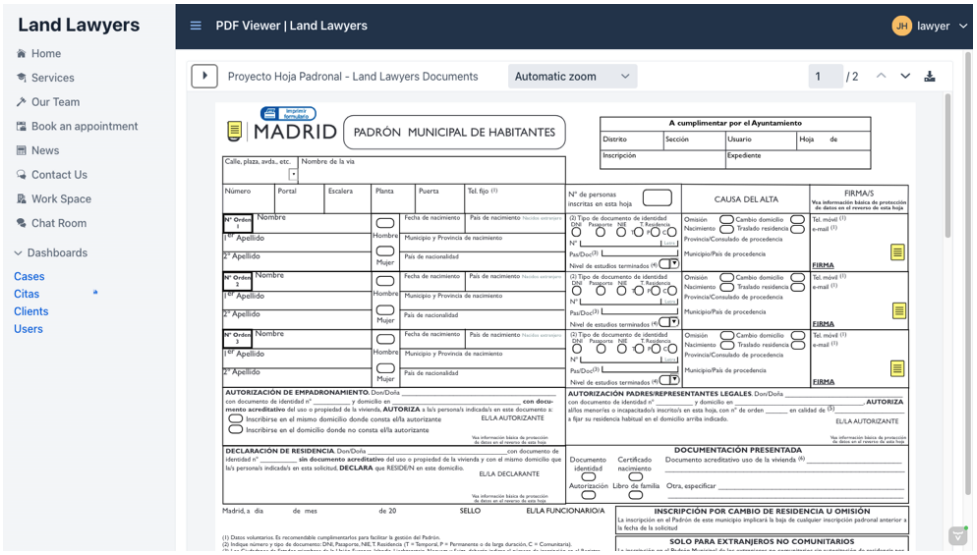


Figura 74 : Vista Documento PDF / Acceso Abogado

3. Acceso usuarios no-clientes

Los usuarios no-clientes, son los usuarios que aún no tienen un expediente en el despacho, que solo tienen el acceso a las vistas generales.

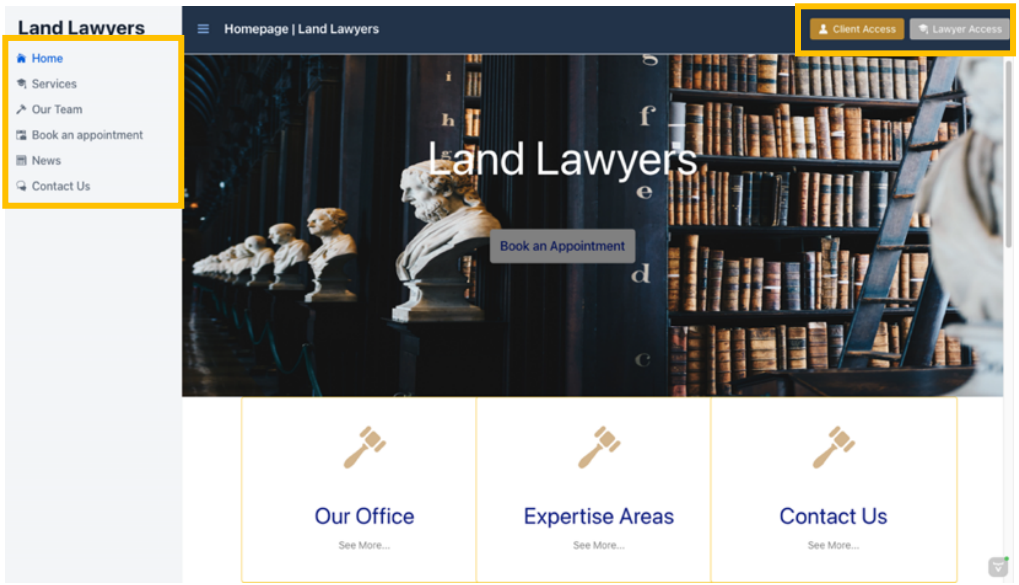


Figura 75 : Página de Inicio / Acceso Usuario no-cliente

Además de ofrecer al usuario una serie de botones en la página de inicio que les permiten reservar una cita, también ofrecemos una página de contacto que contiene toda la información relevante, como la ubicación del despacho y otros detalles, como se muestra en la siguiente imagen.

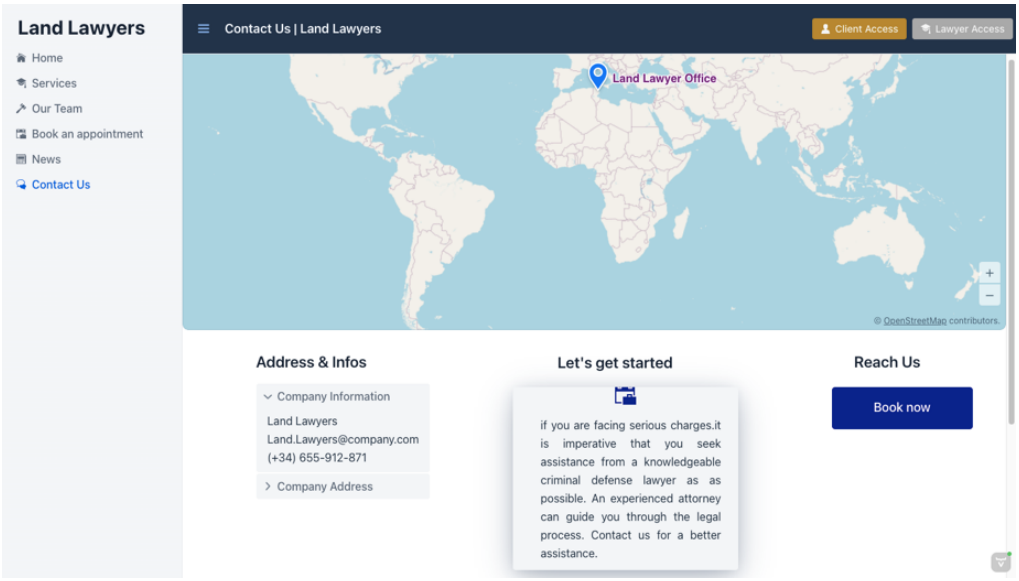


Figura 76 : Vista de mapa y contactos / Acceso Usuario no-cliente

El usuario se dirige a la página de reserva de citas después de navegar y obtener toda la información necesaria sobre el bufete. Complete sus datos personales en esta página, incluido el motivo, la fecha y la hora deseados para la cita, tal como se muestra en la imagen.

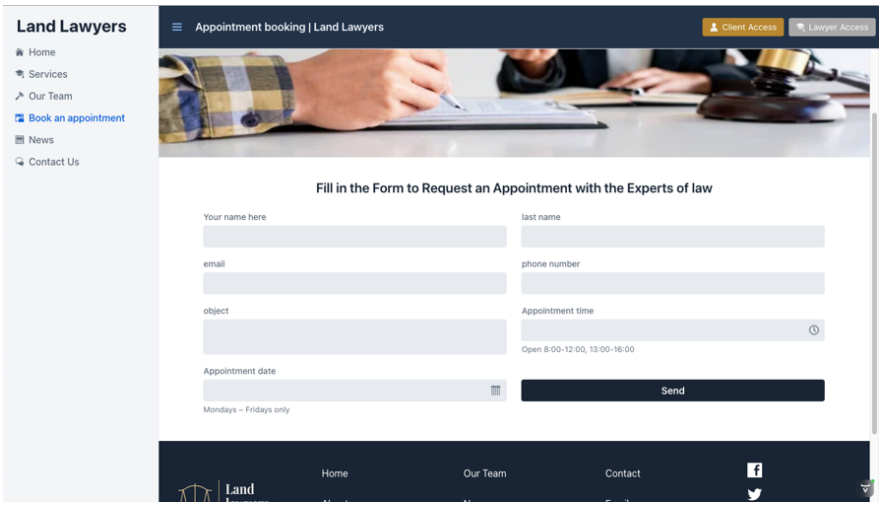


Figura 77 : Formulario Cita / Acceso Usuario no-cliente

Al rellenar los datos, es necesario elegir una hora de la jornada laboral que es de 8:00-12 y 13:00-16, y elegir un día laboral es decir de lunes a viernes. En el caso contrario se mostrarán los errores siguientes y se desactiva el botón "enviar", como se muestra la imagen a continuación.

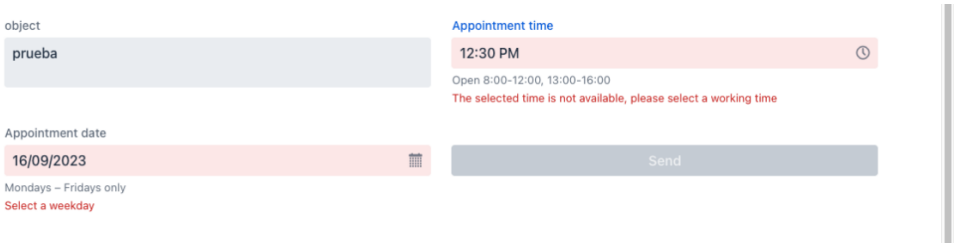


Figura 78 : Horarios y Días Laborables Requeridos

Además, si el usuario elige una fecha y una hora ya reservadas se muestra una notificación indicando la indisponibilidad del tiempo.

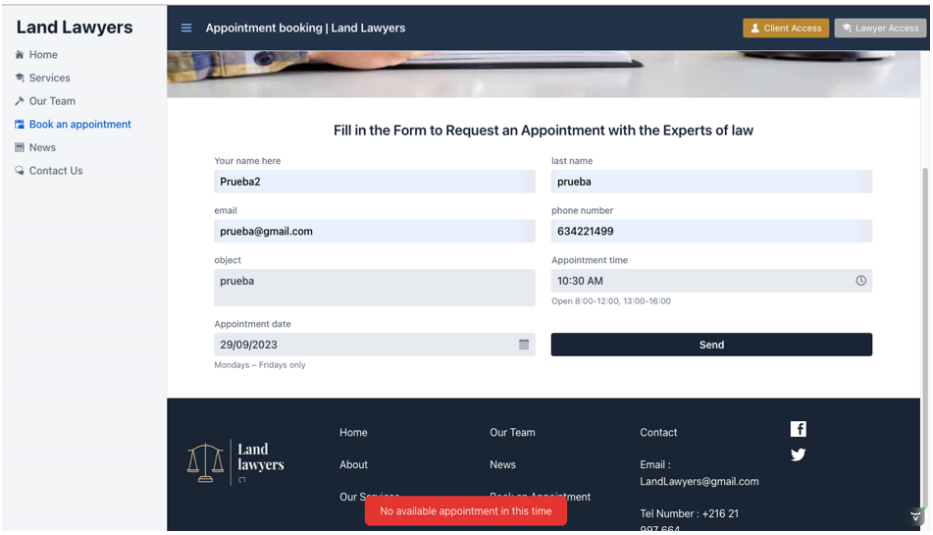


Figura 79 : Notificación Cita no disponible

Si se han elegido el tiempo o la fecha disponibles, se mostrará una notificación de existo y se mostrará al usuario una hoja de confirmación con los datos de la cita y un botón que permite a descargar esta confirmación.

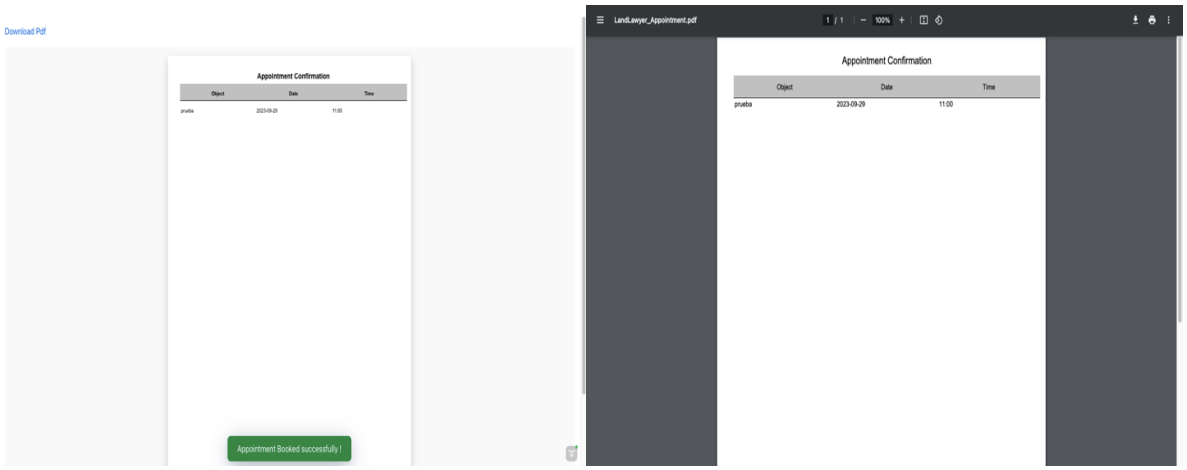


Figura 80 : Descarga de Confirmación de Cita

4. Acceso cliente

Como se ha explicado en la parte de **Funcionamiento de la aplicación,** si el usuario ha sido cliente, se tendrá un nombre de usuario y una contraseña y entre utilizando el botón **acceso cliente.**

Se utilizarán como credenciales**: username: client / password: c**

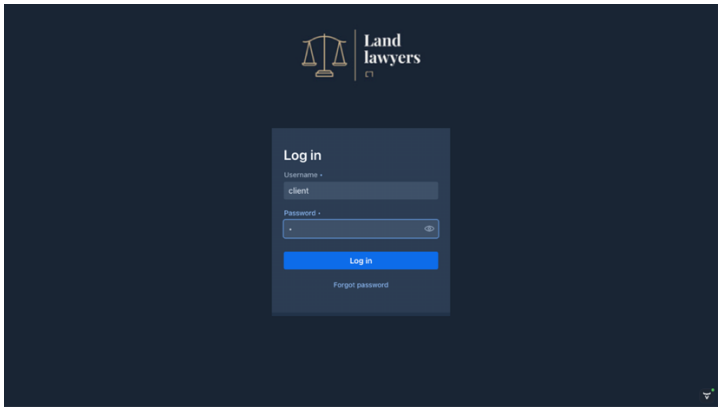


Figura 81 : Inicio de sesión / Acceso Cliente

Si se ingresan datos incorrectos, se desplegará un mensaje de error, tal como se ilustra en la siguiente imagen:

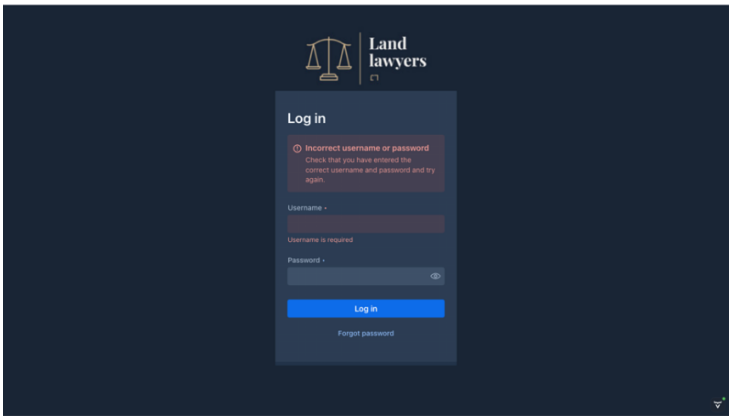


Figura 82 : Error en el Ingreso de Datos

El cliente tiene la posibilidad de chatear con los abogados y de acceder a la página de seguimiento de su caso.

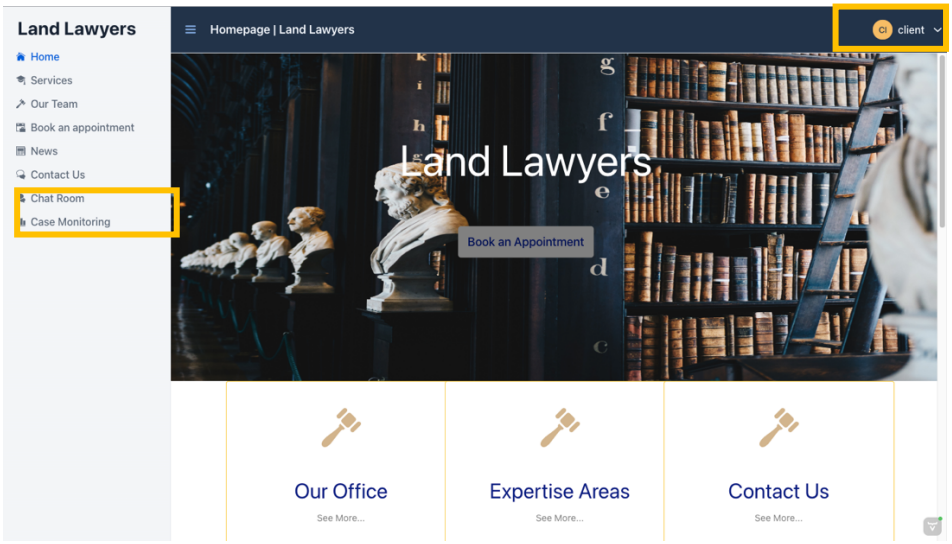


Figura 83 : Página de Inicio / Acceso Cliente

Gracias a la sala de chat, el cliente puede comunicar con su abogado de manera rápida y eficiente, y esto facilita la comunicación entre las dos.

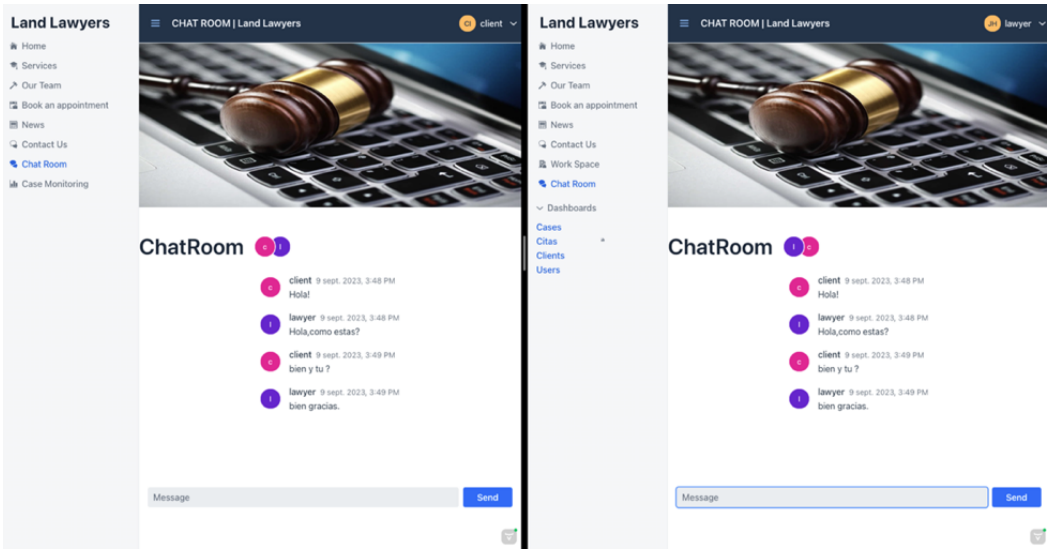


Figura 84 : Sala de chat / Acceso Cliente

Para mejorar la experiencia del usuario y crear un entorno trasparente, se ha creade la interfaz de seguimiento de expediente, donde el usuario puede seguir el estado de su expediente, si en pendiente, cerrado…

Además, en caso de que falte documentación, el abogado puede dejar un comentario para el cliente, lo que facilita la posibilidad de que este agregue el documento directamente en la plataforma.

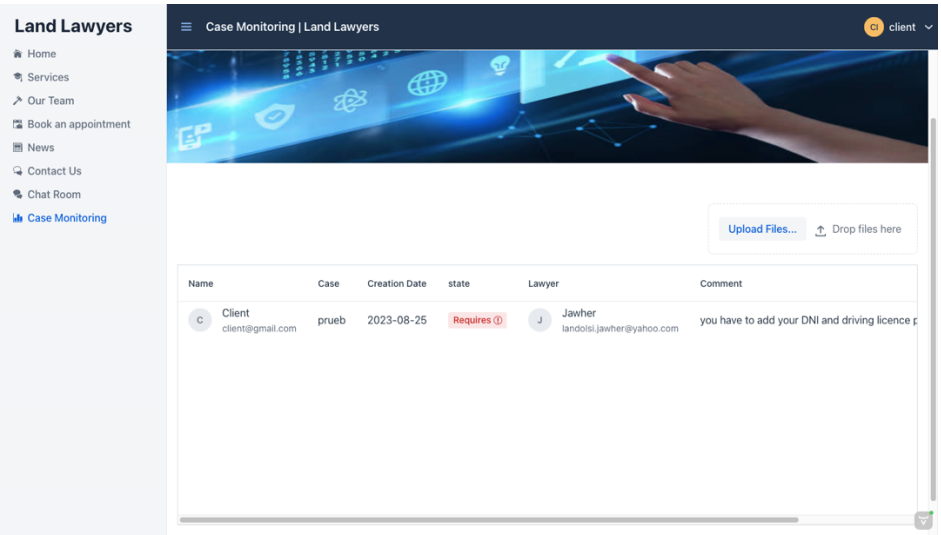


Figura 85 : Seguimiento de caso / Acceso Cliente

En el ejemplo anterior, se marca el estado como "requerido" y se adjunta un comentario que informa al cliente que debe agregar los documentos faltantes. De esta manera, el cliente siempre estará informado.

Si el cliente necesita programar una cita con un abogado, solo debe ingresar el motivo de la cita y seleccionar la fecha y la hora, ya que los demás campos ya están configurados.

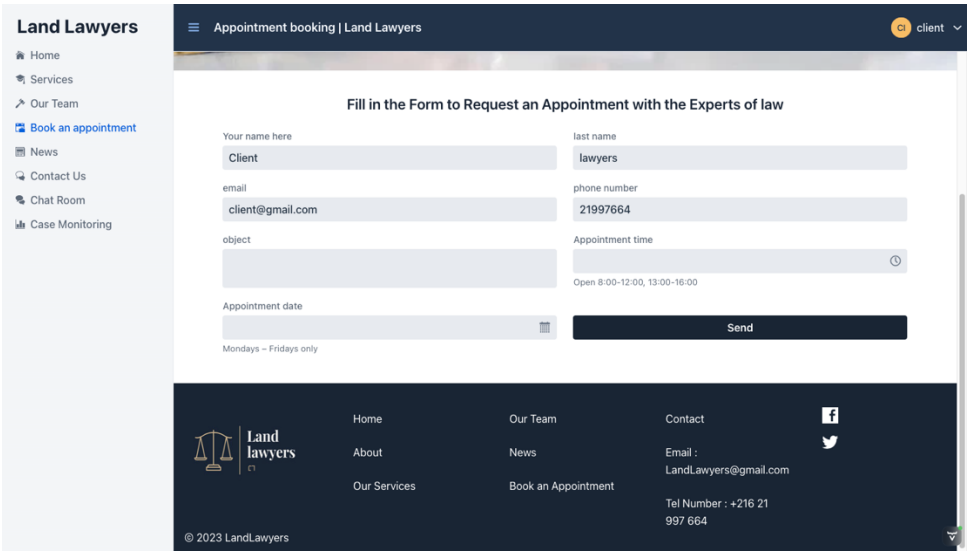


Figura 86 : Formulario Cita / Acceso Cliente

IX. Conclusiones y trabajos futuros.

La realización de este proyecto demuestra que la tecnología es cada vez más potente, e incluso necesaria, en diversos sectores de actividad.

Hoy en día el desarrollar de una aplicación integral utilizando un lenguaje del lado del servidor es posible, lo que abre todo un nuevo mundo de posibilidades en la informática.

Gracias a las innovaciones tecnológicas, ahora se pueden encontrar soluciones que facilitan el trabajo diario de los profesionales, en este caso de los abogados, y que se adaptan a los nuevos usos y prácticas.

A nivel más personal, desarrollar una aplicación utilizando el lenguaje Java fue muy enriquecedor y me aportó muchos conocimientos.

En cuanto a futuros desarrollos, se plantea integrar la API de Google Calendar para automatizar el proceso de programación de citas en los calendarios personales tanto del cliente como del abogado.

Además, en la versión 2.0 implementara la funcionalidad de consultas online a través de videoconferencia dentro la aplicación

Asimismo, se explorará la posibilidad de integrar un sólido sistema de pago en línea, garantizando que todas las transacciones financieras puedan realizarse de forma sencilla y segura dentro de los confines de la aplicación.

X. Bibliografía

Official page Vaadin : https://vaadin.com/

[1] Vaadin Documentation: https://vaadin.com/docs/latest/application

[2] Vaadin Directory Add-on : https://vaadin.com/directory/component/crud-ui-add-on

[3] Vaadin PWA : https://vaadin.com/docs/latest/tutorial/installing-and-offline-pwa

[4] Vaadin documentation securing Spring boot applications : https://vaadin.com/docs/latest/security/enabling-security

[5] Spring Data JPA : https://spring.io/projects/spring-data-jpa

[6] Spring security Overview : https://spring.io/projects/spring-security - overview

[7] Authentication and Authorization: https://www.programaenlinea.net/spring-security-autenticacion-y-autorizacion-basica/

[8] Spring Boot Framework Explication: https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot/

[9] Mysql Advantages and inconvenient: https://openwebinars.net/blog/que-es-mysql/