

Trabajo de Fin de Grado

ÍNDICE GENERAL

1. Introducción y Objetivos	3
1.1 Objetivos Funcionales	4
1.2 Objetivos Técnicos	4
2. Especificación de Requisitos	5
2.1 Requisitos Teóricos	5
2.2 Requisitos Técnicos	7
3. Planificación Temporal y Evaluación de Costes	7
3.1 Planificación Temporal	7
3.2 Evaluación de Costes	8
4. Tecnologías Utilizadas	9
5. Desarrollo e Implementación	10
5.1 Estructura del Proyecto	11
5.1.1 Base de Datos	11
5.1.2 Backend	12
5.1.3 Frontend	20
5.2 Funcionamiento de la Aplicación	21
5.2.1 Administrador	22
5.2.2 Abogado	24
5.2.3 Cliente	26
5.3 Despliegue	28
6. Conclusiones y Líneas Futuras	29
6.1 Conclusiones	29
6.2 Líneas Futuras	29
7. Bibliografía	30

1. Introducción y Objetivos.

Este proyecto de fin de ciclo, desarrollado como parte del módulo profesional del Grado Superior en Desarrollo de Aplicaciones Web (DAW), consiste en una aplicación web de gestión documental para bufetes de abogados y asesorías jurídicas. Su objetivo principal es poner en práctica los conocimientos adquiridos durante los dos años de formación, aplicando tecnologías actuales del sector como Spring Boot para el backend y React para el frontend.

Propósito del sistema:

La aplicación está diseñada para automatizar procesos manuales en el ámbito legal, permitiendo:

- La subida y descarga segura de documentos entre clientes y abogados.
- La gestión de usuarios con roles diferenciados: administradores, abogados y clientes.
- El envío automático de enlaces de registro por correo electrónico (usando JavaMail).

Tecnologías clave:

- Backend: Spring Boot con Spring Security para autenticación mediante JWT.
- Frontend: React con Fetch API para la comunicación con el backend.
- Base de datos: PostgreSQL para almacenar usuarios, documentos y relaciones.

Funcionalidades principales

Para administradores:

- Aprobar solicitudes de nuevos abogados.
- Visualizar listados de abogados y sus clientes asociados.

Para abogados:

- Gestionar sus clientes y documentos.
- Recibir archivos subidos por los clientes.

Para clientes:

- Subir documentos para su abogado asignado.
- Ver el estado de sus archivos.

Este sistema simplifica el día a día de los profesionales del derecho, eliminando el uso de correos electrónicos o memorias USB para compartir documentos sensibles. Además, al estar desarrollado con tecnologías estándar de la industria, sirve como demostración de las competencias profesionales adquiridas durante el ciclo formativo.

1.1 Objetivos Funcionales.

- Desarrollar un Sistema funcional entre backend y frontend.
- Permitir la autenticación y autorización de los usuarios en base de roles.
- Implementar la subida, visualización y descarga de archivos entre usuarios y roles.
- Uso de una interfaz simple.

1.2 Objetivos Técnicos.

- Utilizar Springboot como tecnología backend.
- Utilizar React como frontend.
- Uso de JWT (JSON Web Tokens) para la autenticación.
- Uso de Security para la seguridad.
- Uso de Cors para la conexión con el frontend.
- Uso de PasswordEncoder para la codificación de las contraseñas.

- Uso de Javamail y SimpleMailMessage para enviar los correos
- Separación de accesos según rol: administrador, abogado y cliente.
- Permitir que el administrador vea los abogados y los clientes por abogados.
- Permitir que el abogado vea sus clientes y los documentos subidos por cada cliente.
- Permitir que el cliente suba documentos a su abogado y este pueda descargarlos.
- Implementar que los enlaces de registro sean mandados por correo.

Este trabajo tiene como finalidad describir el desarrollo técnico de el proyecto, explicar las decisiones tomadas y las posibles mejoras futuras que se puedan realizar.

2. Especificación de Requisitos.

Para garantizar el perfecto funcionamiento de la aplicación y por ende que esta responda de manera adecuada a los usuarios que la están usando, se han establecido una serie de requisitos fundamentales. Estos requisitos se dividen en requisitos técnicos y en requisitos teóricos, que especifican aspectos relacionados con la implementación, arquitectura y desarrollo o en su defecto definen el comportamiento esperado de la aplicación y sus características operativas respectivamente.

2.1 Requisitos Teóricos.

1. Gestión de Usuarios:

- El sistema permite el registro de abogados y clientes.
- El registro de abogados debe ser aprobado por un administrador.
- El registro de el cliente debe ser aprobado por un abogados

2. Autenticación y Autorización:

- Es sistema implementará autenticación mediante JWT.
- Impedirá que un usuario sin autorización acceda a documentos no permitidos.
- El cliente irá relacionado con su abogados
- La validación será en tiempo real
-
- Navegación consistente entre endpoints

3. Gestión de Documentos:

- Los clientes pueden subir documentos a su abogado asignado.
- Los abogados deben poder descargar dichos documentos de sus abogados

4. Visualización de Administración:

- El administrador verá un listado de los abogados y sus clientes
- Podrá aprobar o rechazar las solicitudes de nuevos abogados

5. Visualización según Rol:

- El cliente ve sus documentos.
- El abogado debe ver sus clientes asociados y los documentos compartidos.
- El administrador unicamente verá los abogados y los clientes de estos pero no los datos.

2.2 Requisitos Técnicos:

1. Seguridad

- Todos los endpoints deben estar protegidos por la autenticación JWT.
- Los roles serán verificados en cada petición.

2. Mantenibilidad

- El código debe estar organizado en capas, ya sean controladores, daos, servicios etc.

3. Usabilidad

- La interfaz debe ser clara, con botones visibles y acciones intuitivas

3. Planificación Temporal y Evaluación de Costes:

3.1 Planificación Temporal

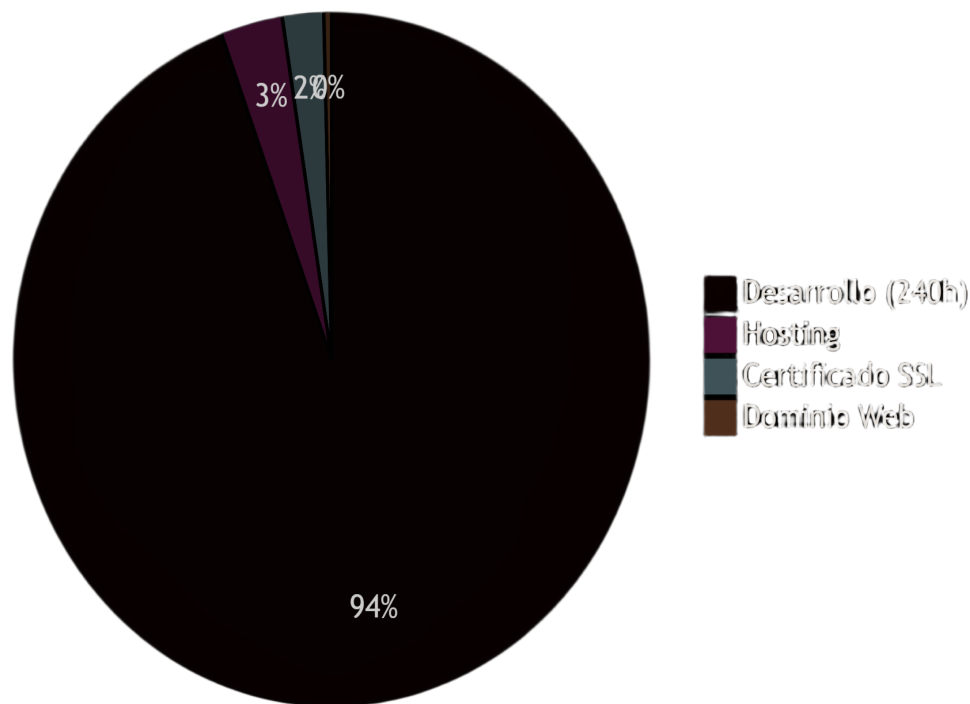
La planificación temporal de este proyecto se han dividido en cuatro fases distribuidas a lo largo de cuatro semanas:



3.2 Evaluación de Costes

A pesar de que el proyecto ha sido desarrollado como un trabajo final de un grado superior y por ende no tiene un costo real, a pesar de eso, se puede estimar unos costes aproximados para un entorno real:

tribución de Costes Anuales (Total: 3.815€)



En esta gráfica el 94% corresponde con el coste de un desarrollador Full-stack que cobraría 3.600€, en segundo lugar con un 3% entraría el hosting con 120€, en tercera posición con un 2% y 80€ anuales encontramos al SSL mientras que finalmente por el 0.5€ equivalente a 15€ encontraríamos el dominio de la aplicación.

Ha de tenerse en cuenta que se han tomado valores tipo para poder hacer los cálculos de una manera semi-realista.

4. Tecnologías Utilizadas

El presente proyecto ha sido realizado empleando un conjunto de tecnologías modernas usadas de forma amplia en el desarrollo profesional de aplicaciones web. Estas herramientas han sido seleccionadas ya sea por haber sido estudiadas a lo largo del curso o en su defecto tener mayor fama entre desarrolladores reales. De esta manera se garantiza un mínimo de profesionalidad dando un resultado óptimo y de calidad.

En cuanto a la base de datos, se ha optado por PostgreSQL como motor de base de datos relacional, administrado por la plataforma Railwind. Esta solución en la nube ofrece un entorno completo de despliegue y gestión de bases de datos, destacando por su facilidad de uso y escalabilidad. Durante el desarrollo, el coste de desarrollo ha sido mínimo gracias al plan gratuito inicial que ofrece la plataforma, manteniéndose dentro del presupuesto sin requerir inversión adicional.

Para el desarrollo de la interfaz de usuario, se ha empleado React.js en combinación con Vite como herramienta de construcción, utilizando JavaScript como lenguaje principal. Esta combinación tecnológica nos permite crear interfaces dinámicas y reactivas con un excelente rendimiento. Ha de mencionarse React Router (Routes), una librería que es fundamental puesto que facilita en gran medida la navegación entre varias rutas, funcionando de forma similar a los endpoints en Spring Boot. El diseño visual se ha implementado mediante hojas de estilo CSS personalizadas, aprovechando variables para mantener coherencia en toda la aplicación. La comunicación con el servidor es realizada mediante Fetch API, una tecnología nativa de el navegador que permite realizar peticiones HTTP de forma eficiente.

En cuanto al servidor, el proyecto utiliza principalmente tecnología Java que trabaja de manera conjunta.

Spring Boot actúa como el framework principal, proporcionando la base para el desarrollo de el backend de la aplicación. Por otra parte Maven se encarga de el control de dependencias y el proceso de construcción de el proyecto.

Para la persistencia de datos, se ha implementado JPA (Java Persistence API) junto con Hibernate, de esta forma podemos mapear objetos Java a tablas relacionales de forma transparente y simplificando en gran medida las operaciones de la base de datos.

La seguridad de la aplicación se ha reforzado mediante numerosos componentes:

- JWT (JSON Web Token) para autenticación y autorización de usuarios.
- Spring Security como framework de seguridad principal.
- PasswordEncoder para el cifrado seguro de contraseñas.
- Configuración CORS para garantizar la comunicación segura entre frontend y backend

Para poder enviar correos, parte fundamental en el proceso de registro, se ha implementado utilizando las librerías JavaMail y SimpleMailMessage, que permiten la integración de servidores SMTP de forma sencilla y comoda.

Como herramientas complementarias que han facilitado el proceso de desarrollo, debemos mencionar:

- Postman para pruebas API
- Los IDE VSCode e IntelliJ IDEA
- Github

Este conjunto de tecnologías no solo permiten los requisitos fundamentales de el proyecto, sino que también representa un currículo moderno y demandado en el mercado actual, demostrando las competencias profesionales obtenidas en el modulo.

5. Desarrollo e Implementación

Para ofrecer una comprensión detallada y estructurada sobre el proceso de desarrollo de esta aplicación. Se ha organizado este capítulo en tres apartados principales que permitirán analizar el sistema desde cada parte estructural de el proyecto.

En el primer apartado, se abordará exclusivamente la estructura de el proyecto, realizando un análisis tanto de los archivos como de los directorios correspondientes al backend y al frontend respectivamente. Este apartado incluirá un análisis de las partes importantes o destacables de el código, describiendo que funciones realiza cada componente dentro de esta gran estructura.

Adicionalmente, esta sección incorpora la explicación detallada de sobre el diseño de la base de datos de el sistema.

En el segundo apartado se examinará el funcionamiento práctico de la aplicación, describiendo como actúa la aplicación ante el uso óptimo de esta por parte de un usuario. Para realizar esto se explicarán como funcionan las diferentes opciones y como afecta el código en estos eventos. De esta forma estableceremos una comprensión completa del proyecto.

Finalmente, el tercer apartado expondrá un análisis comparativo entre dos métodos de despliegue de la aplicación en entornos de producción.

5.1 Estructura del Proyecto

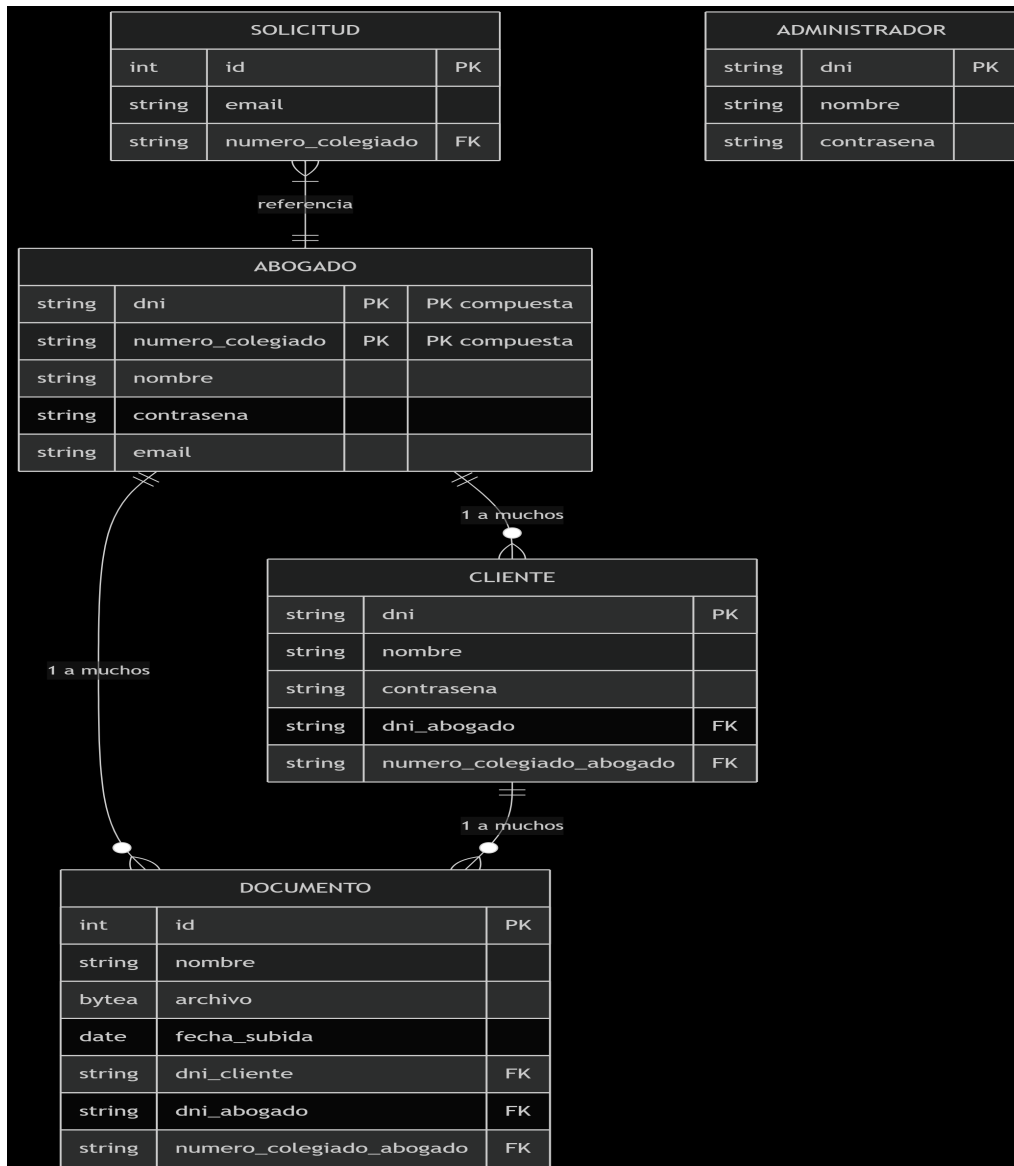
En este apartado hablaremos de forma detallada de la estructura de la base de datos, el backend y el frontend. Explicando como es su estructura y así de esta manera poder entender en mayor medida la explicación de el apartado 5.2

5.1.1 Base de Datos

Para la base de datos se usa PostgreSQL mediante Railwind, una aplicación web que permite hostear de manera gratuita durante los primeros cinco euros de saldo gestionar una base de datos relacional. Estructuralmente se encuentra conformada por las siguientes tablas:

- **Abogado:** Almacena los datos de este como el nombre, el dni, el numero colegiado, etc. Tiene una clave primaria doble de dni y numero colegiado.
- **Administrador:** Almacena los datos de el administrador tales como el dni, nombre, etc. Su clave primaria es dni
- **Ciente:** Almacena los datos de el cliente y tanto el dni como el numero colegiado de el abogado al que están registrados
- **Documento:** Almacena los datos de el documento que se ha subido. Tiene de clave ajena los elementos las claves primarias de cliente y abogado.
- **Solicitud:** Guarda los elementos de la solicitud siendo estos correo y numero colegiado, en cuanto a su clave primaria se genera de forma automatica.

En cuanto a las relaciones de la base de datos realmente solo existe una que sirve para limitar y gestionar los clientes con sus abogados. Esta relación entre Cliente y Abogado es One to Many, es decir, un abogado puede tener varios clientes pero cada cliente solo tiene un abogado.



5.1.2 Backend

Estructuralmente siguiendo la ruta `src/main/java/backend.backend/` accedemos a los directorios de el proyecto. En estos se almacenan los diferentes archivos .java que conforman el backend completamente. Estas carpetas son las siguientes:

Controller: Aquí se almacenan los controladores del proyecto es decir todos aquellos archivos donde se crea en endpoint y se le da diferentes funciones, por lo general estos controladores van relacionados con las entidades del proyecto, es decir, los endpoints relacionados a Abogado se encontrarían en `AbogadoController` aunque en este caso existen algunas excepciones siendo estas:

- **RegistroController:** Aquí se acceden a los registros tanto de Cliente como de Abogado junto con el registro de una solicitud.

```
public class RegistroController {

    @PostMapping(@PathVariable("solicitud"))
    public ResponseEntity<?> solicitud(@RequestBody Solicitud solicitud){
        Solicitud crearsolicitud = new Solicitud();
        crearsolicitud.setNumeroColegiado(solicitud.getNumeroColegiado());
        crearsolicitud.setMail(solicitud.getMail());
        crearsolicitud.setDni(solicitud.getDni());

        daoSolicitud.save(crearsolicitud);

        return ResponseEntity.ok().body("Solicitud enviada correctamente");
    }

    @RequestMapping(@PathVariable("/registro-abogado"))
    public ResponseEntity<?> registro(@RequestBody Abogado abogado){
        try{
            Abogado nuevoAbogado= abogadoService.save(abogado);
            return ResponseEntity.ok(nuevoAbogado);
        }catch(IllegalArgumentException e){
            return ResponseEntity.status(400).body("El abogado ya existe "+e.getMessage());
        }catch(Exception e){
            return ResponseEntity.status(500).body("Error al registrar el abogado "+e.getMessage());
        }
    }

    @RequestMapping(@PathVariable("/registro-cliente"))
    public ResponseEntity<?> registroCliente(
        @RequestBody Cliente cliente,
        @RequestHeader("Authorization") String authorizationHeader) {

        // Authorization: Bearer <token>
        String token = authorizationHeader.replace(target: "Bearer ", replacement: "");
        String dniAbogado = jwtUtils.getDniFromToken(token);

        Optional<Abogado> abogado = daoAbogado.findById_Dni(dniAbogado);
        if (abogado.isEmpty()) {
            return ResponseEntity.status(400).body("El abogado no existe");
        }
        cliente.setAbogado(abogado.get());
        Cliente newCliente = clienteService.save(cliente);
        return ResponseEntity.ok(newCliente);
    }
}
```

- **AuthController:** Actualmente no se puede acceder a este endpoint variando SecurityConfig se puede activar para añadir usuarios a mano.

```
@PostMapping(@PathVariable("/register/cliente"))
public ResponseEntity<?> registerCliente(@RequestBody Cliente cliente) {
    cliente.setContrasena(passwordEncoder.encode(cliente.getContrasena()));
    daoCliente.save(cliente);
    return ResponseEntity.ok().body("Cliente registrado con éxito");
}

@PostMapping(@PathVariable("/register/abogado"))
public ResponseEntity<?> registerAbogado(@RequestBody Abogado abogado) {
    abogado.setContrasena(passwordEncoder.encode(abogado.getContrasena()));
    daoAbogado.save(abogado);
    return ResponseEntity.ok().body("Abogado registrado con éxito");
}

@PostMapping(@PathVariable("/register/administrador"))
public ResponseEntity<?> registerAdministrador(@RequestBody Administrador administrador) {
    administrador.setContrasena(passwordEncoder.encode(administrador.getContrasena()));
    daoAdministrador.save(administrador);
    return ResponseEntity.ok().body("Administrador registrado con éxito");
}
```

- **LoginController:** Este controlador se dedica a gestionar el dni, la contraseña y el rol de el cliente para realizar las siguientes funciones. Primero permitirte iniciar sesión comprobando que la contraseña insertada coincide con la contraseña de el usuario y Segundo genera un token JWT donde se almacena el dni y el rol de el usuario para de esta manera poder acceder a sus endpoints reservados.

```

@Autowired
private ClienteService clienteService;
@Autowired
private PasswordEncoder passwordEncoder;
@Autowired
private JWUtils jwtUtils;
@PostMapping
public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest){
    String dni = loginRequest.getDni();
    String contrasenaLimpia = loginRequest.getContrasena();
    String rol = loginRequest.getRol(); // "ADMIN", "ABOGADO", "CLIENTE"

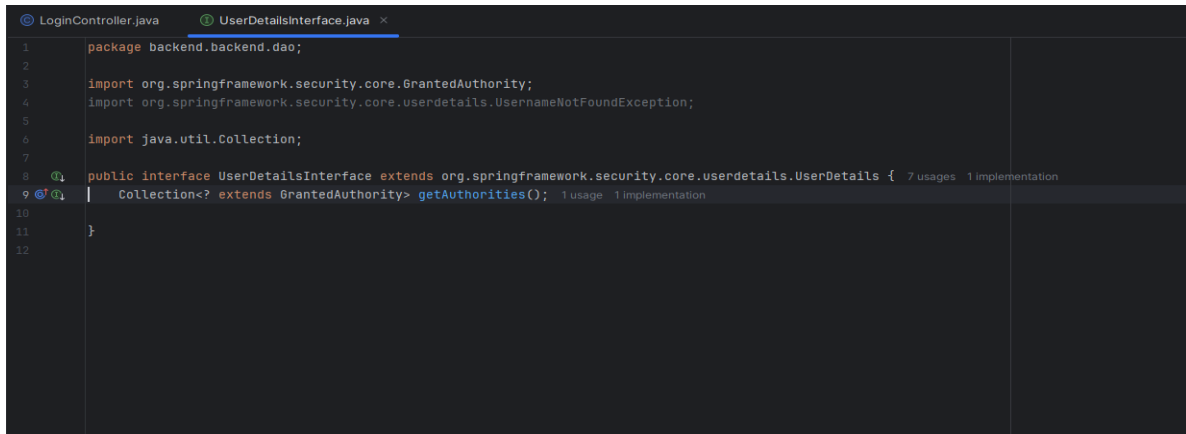
    switch (rol.toUpperCase()) {
        case "ADMIN":
            Optional<Administrador> admin = administradorService.findById(dni);
            if (admin.isPresent() && passwordEncoder.matches(contrasenaLimpia, admin.get().getContrasena())) {
                String token = jwtUtils.generateToken(dni, rol);
                return ResponseEntity.ok(Map.of("token", token));
            }
            break;
        case "ABOGADO":
            Optional<Abogado> abogado = abogadoService.findByIdDni(dni);
            if (abogado.isPresent() && passwordEncoder.matches(contrasenaLimpia, abogado.get().getContrasena())) {
                String token = jwtUtils.generateToken(dni, rol);
                return ResponseEntity.ok(Map.of("token", token));
            }
            break;
        case "CLIENTE":
            Optional<Cliente> cliente = clienteService.findById(dni);
            if (cliente.isPresent() && passwordEncoder.matches(contrasenaLimpia, cliente.get().getContrasena())) {
                String token = jwtUtils.generateToken(dni, rol);
                return ResponseEntity.ok(Map.of("token", token));
            }
            break;
    }

    return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Usuario o contraseña incorrectos");
}

```

Dao: En la carpeta dao se almacenan todas y cada una de las interfaces a las que luego los servicios buscarán ciertas funciones. Al igual que en controlador suelen encontrarse principalmente los daos, o repositorios, de las entidades. Esto sucede para que sea mas cómodo seguir la lógica de el código. A pesar de ello es necesario destacar el siguiente archivo:

- **UserDetailsInterface:** Esta interfaz extiende de security permitiendo usar todas sus funciones de forma mas organizada. Esta interfaz esta íntimamente relacionada con los JWT que explicaremos a fondo mas adelante.



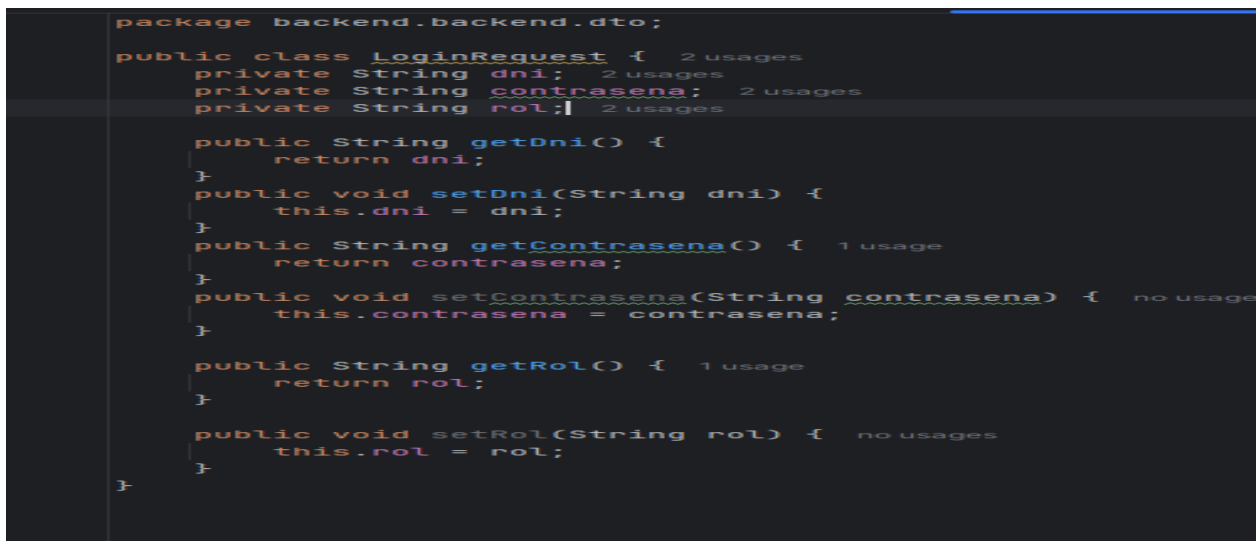
```

1 package backend.backend.dao;
2
3 import org.springframework.security.core.GrantedAuthority;
4 import org.springframework.security.core.userdetails.UsernameNotFoundException;
5
6 import java.util.Collection;
7
8 public interface UserDetailsInterface extends org.springframework.security.core.userdetails.UserDetails { 7 usages 1 implementation
9     Collection<? extends GrantedAuthority> getAuthorities(); 1 usage 1 implementation
10
11 }
12

```

Dto: Los dto son archivos que permiten maquillar la información que se obtiene de un usuario para que no muestre información sensible en el json. En este caso y debido a como se gestionan los usuarios solo existe ClienteDto como un dto clásico. Aún así hay otro archivo en la carpeta dto que vale la pena mencionar mas a fondo:

- **LoginRequest:** Este dto sirve para obtener en LoginController el usuario, contraseña y rol que el usuario facilita al iniciar sesión.



```

package backend.backend.dto;

public class LoginRequest { 2 usages
    private String dni; 2 usages
    private String contrasena; 2 usages
    private String rol; 2 usages

    public String getDni() {
        return dni;
    }
    public void setDni(String dni) {
        this.dni = dni;
    }
    public String getContrasena() { 1 usage
        return contrasena;
    }
    public void setContrasena(String contrasena) { no usage
        this.contrasena = contrasena;
    }

    public String getRol() { 1 usage
        return rol;
    }

    public void setRol(String rol) { no usages
        this.rol = rol;
    }
}

```

Entity: En esta carpeta se almacenan las entidades, archivos que se crean según la base de datos o viceversa y sirven para transformar programación orientada a objetos en comandos sql. Es destacable en este repositorio que abogado al tener una clave multiple posee dos entidades. Una llamada abogado que almacena todos los datos junto con abogadoid y abogadoid que almacena las claves primarias de abogado.

```
@Embeddable 5 usages
public class AbogadoId implements Serializable {
    private static final long serialVersionUID = 488612524010180022L; no usages
    @Column(name = "dni", nullable = false, length = 20) 5 usages
    private String dni;

    @Column(name = "numero_colegiando", nullable = false, length = 20) 5 usages
    private String numeroColegiando;

    public String getDni() { return dni; }

    public void setDni(String dni) { this.dni = dni; }

    public String getNumeroColegiando() { return numeroColegiando; }

    public void setNumeroColegiando(String numeroColegiando) { this.numeroColegiando = numeroColegiando; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(proxy, this) != Hibernate.getClass(o)) return false;
        AbogadoId entity = (AbogadoId) o;
        return Objects.equals(this.dni, entity.dni) && Objects.equals(this.numeroColegiando, entity.numeroColegiando);
    }

    @Override
    public int hashCode() { return Objects.hash(dni, numeroColegiando); }
```

```
@Entity
@Table(name = "abogado")
public class Abogado {
    @EmbeddedId 2 usages
    private AbogadoId id;

    @Column(name = "nombre", length = 100) 2 usages
    private String nombre;

    @Column(name = "telefono", length = 20) 2 usages
    private String telefono;

    @Column(name = "correo", length = 100) 2 usages
    private String correo;

    @Column(name = "contrasena", length = 100) 2 usages
    private String contrasena;

    public AbogadoId getId() { return id; }

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

    public String getNombre() { return nombre; }

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

    public String getTelefono() { return telefono; }

    public void setTelefono(String telefono) { this.telefono = telefono; }

    public String getCorreo() { return correo; }

    public void setCorreo(String correo) { this.correo = correo; }

    public String getContrasena() { return contrasena; }

    public void setContrasena(String contrasena) { this.contrasena = contrasena; }
```


Security: En este repositorio se almacenan todos los archivos java relacionados con la seguridad que vamos a explicar uno a uno:

- **CustomUserDetails:** Este archivo sirve para adaptar al usuario ya sea cliente, abogado o administrador al modelo que espera SpringSecurity internamente con la interfaz UserDetailsInterface.

```
public class CustomUserDetails implements UserDetailsInterface { 4 usages

    private String dni; 2 usages
    private String password; 2 usages
    private String rol; 3 usages

    public CustomUserDetails(String dni, String password, String rol) { 3 usages
        this.dni = dni;
        this.password = password;
        this.rol = rol;
    }

    @Override 1 usage
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(() -> "ROLE_" + rol);
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return dni;
    }

    public String getRol() { no usages
        return rol;
    }

    @Override no usages
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override no usages
    public boolean isAccountNonLocked() {
        return true;
    }
}
```

- **JWTAuthenticatorFilter:** Es un archivo que recoge de las peticiones el header que tiene la siguiente estructura “Bearer” + token, de este obtiene el token y se asegura de que es un token valido. Para realizarlo obtiene el dni de el token y se asegura que el usuario existe.

```
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
    final String authHeader = request.getHeader("Authorization");
    final String jwt;
    final String dni;

    if(authHeader == null || !authHeader.startsWith("Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    jwt = authHeader.substring(7);
    dni = jwtService.getDniFromToken(jwt);

    if(dni != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        UserDetails userDetailsInterface = userDetailsService.loadUserByUsername(dni);

        if(jwtService.validateToken(jwt, (UserDetailsInterface) userDetailsInterface)) {
            UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(
                    userDetailsInterface,
                    null,
                    userDetailsInterface.getAuthorities()
                );
            authenticationToken.setDetails(
                new WebAuthenticationDetailsSource().buildDetails(request)
            );
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        }
    }

    filterChain.doFilter(request, response);
}
```

- **JWUtils:** En este archivo creamos el token JWT, con el dni y el rol o solo con el dni. El proceso para realizarlo es el siguiente: tenemos una variable donde almacenamos la semilla en este caso SECRET_KEY_STRING, con Key obtenemos los bytes de la semilla y junto con EXPIRATION_TIME, que es el tiempo de vida util de el token, creamos mediante jwst.builder, el dni y el rol un token con la siguiente estructura {texto}. {texto}. {texto} que usaremos para identificarnos en la app

```
public class JWUtils {
    private static final String SECRET_KEY_STRING = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY1234567890abcdefghijklmnopqrstuvwxyz";
    private static final Key key = Keys.hmacShaKeyFor(SECRET_KEY_STRING.getBytes(StandardCharsets.UTF_8));
    private static final long EXPIRATION_TIME = 864000000; // 10 días 2 usages

    public String generateToken(String dni, String rol){ 3 usages
        return Jwts.builder()
            .setSubject(dni)
            .claim("rol", rol)
            .setIssuedAt(new java.util.Date(System.currentTimeMillis()))
            .setExpiration(new java.util.Date(System.currentTimeMillis() + EXPIRATION_TIME))
            .signWith(SignatureAlgorithm.HS512, key)
            .compact();
    }

    public String generateToken(String dni){ 1 usage
        return Jwts.builder()
            .setSubject(dni)
            .setIssuedAt(new java.util.Date(System.currentTimeMillis()))
            .setExpiration(new java.util.Date(System.currentTimeMillis() + EXPIRATION_TIME))
            .signWith(SignatureAlgorithm.HS512, key)
            .compact();
    }

    public String getDniFromToken(String token){ 2 usages
        return Jwts.parser().setSigningKey(key)
            .parseClaimsJws(token).getBody().getSubject();
    }

    public String getRolFromToken(String token){ no usages
        return Jwts.parser().setSigningKey(key)
            .parseClaimsJws(token).getBody().get("rol", String.class);
    }
}
```

- **SecurityConfig:** Finalmente en este archivo especificamos dos cosas. Por una parte en securityFilterChain forzamos quien puede acceder a que rutas y con corsConfigurationSource especificamos que se nos puede conectar y hacer peticiones. En esta caso puesto que no está en producción permitimos que nos realicen peticiones cualquiera para poder conectarnos con el frontend.

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

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .addFilterBefore(new JwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class)
        .cors().configurationSource(corsConfigurationSource())
        .csrf().disable()
        .authorizeHttpRequests((AuthorizationManagerRequestMatcher) authz -> authz
            .requestMatchers(@"/login").permitAll()
            .requestMatchers(@"/register/**").permitAll()
            .requestMatchers(@"/admin/**").hasRole("ADMIN")
            .requestMatchers(@"/abogado/**").hasRole("ABOGADO")
            .requestMatchers(@"/cliente/**").hasRole("CLIENTE")
            .anyRequest().authenticated()
        )
        .httpBasic(withDefaults())
        .formLogin().formLoginConfigurer((HttpSecurity) form -> form.disable());
    return http.build();
}

@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration config = new CorsConfiguration();
    config.addAllowedOriginPattern("*");
    config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
    config.setAllowCredentials(true);
    config.setAllowedHeaders(List.of("*"));

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", config);

    return source;
}
```

Services: Por ultimo en esta carpeta guardamos los servicios de la aplicación, aquí se encuentran las funciones y la lógica que Springboot con el webkit no proporcione, es decir los elementos que se encuentran en los daos. A pesar de ello cabe destacar el siguiente:

- **UserDetailsService:** Primero hay que mencionar que este nombre a pesar de poner que es una interfaz no lo es, lo cual es un error que se realizo en etapas altas de desarrollo y para evitar el mal funcionamiento se ha mantenido así. A pesar de eso su función reside en buscar un usuario especifico entre los diferentes tipos de usuarios posibles para luego compararlo con el que se obtiene con el token jwt.

```
private DaoAdministrador daoAdministrador;

@Override
@1 usage
public UserDetails loadUserByUsername(String dni) throws UsernameNotFoundException {
    // Buscar en Cliente
    Optional<Cliente> clienteOpt = daoCliente.findByDni(dni);
    if (clienteOpt.isPresent()) {
        Cliente cliente = clienteOpt.get();
        return new CustomUserDetails(cliente.getDni(), cliente.getContrasena(), rol: "CLIENTE");
    }

    // Buscar en Abogado
    Optional<Abogado> abogadoOpt = daoAbogado.findById_Dni(dni);
    if (abogadoOpt.isPresent()) {
        Abogado abogado = abogadoOpt.get();
        return new CustomUserDetails(abogado.getId().getDni(), abogado.getContrasena(), rol: "ABOGADO");
    }

    // Buscar en Administrador
    Optional<Administrador> adminOpt = daoAdministrador.findByDni(dni);
    if (adminOpt.isPresent()) {
        Administrador admin = adminOpt.get();
        return new CustomUserDetails(admin.getDni(), admin.getContrasena(), rol: "ADMIN");
    }

    throw new UsernameNotFoundException("Usuario no encontrado con DNI: " + dni);
}
```

Por otra parte siguiendo la ruta `src/main/resources` podremos acceder a `application.properties`. Es importante mencionarlo pues aquí se guardan ciertas configuraciones importantes como el puerto que se usa para mandar un correo por el protocolo smtp (Simple Mail Transfer Protocol) además de el mail por el que se manda el código necesario que se obtiene para que google gmail permita que una app mande correos, las credenciales de la base de datos, etc.

```
spring.application.name=backend
spring.datasource.url=jdbc:postgresql://gondola.proxy.rlwy.net:13491/postgres
spring.datasource.username=postgres
spring.datasource.password=YjBdYnwZJJcAKmszKDfJyfuiHmIEWJUt
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=tfq745111@gmail.com
spring.mail.password=gjpl xhyc loen ntnb
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

5.1.3 Frontend

En cuanto al frontend la estructura varía en comparación con el front por un lado tenemos `App.jsx` y luego dentro de `src` tenemos `pages` y `components`. Por lo que en este caso mencionaremos los archivos directamente ya que son menos que en backend:

App.jsx: Es el nexo de todas las páginas, aquí se almacenan las rutas y a qué páginas están enlazadas.

```
function App() {
  const [count, setCount] = useState(0)

  return (
    <Router>
      <Routes>
        <Route path="/" element={<RegistroPage />} />
        <Route path="/admin" element={<AdministradorPage/>} />
        <Route path="/abogado" element={<Abogado/>} />
        <Route path="/cliente" element={<ClientePage/>} />
        <Route path="/solicitud" element={<SolicitudPage/>} />
        <Route path="/registro-abogado" element={<RegistroAbogadoPage/>} />
        <Route path="/registro-cliente" element={<RegistroCliente/>} />
      </Routes>
    </Router>
  )
}
```

- **Pages:** Este directorio almacena todas las paginas de la aplicación junto con sus hojas de estilos Principalmente aquí encontramos formularios y las paginas de los usuarios. Por otra parte es aquí donde se utilizan también los use effect
- **Fetch.jsx:** El archivo que permite la conexión con el backend, esta llena de Fetch API con todos los endpoints del backend.

```
export async function registrarCliente(cliente, token) {
  const response = await fetch('http://localhost:8080/registro/registro-cliente', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${token}`
    },
    body: JSON.stringify(cliente),
  });

  if (!response.ok) {
    const errorText = await response.text();
    throw new Error(errorText || 'Error al registrar cliente');
  }

  return response.json();
}
```

- **Logout.jsx:** Solo tiene una función que permite realizar un logout. Para ello se borra el token de localStorage y se redirige a / donde esta el formulario de inicio de sesión.

```
1 export function logout(navigate) {
2   localStorage.removeItem('token');
3   navigate('/');
4 }
```

5.2 Funcionamiento de la aplicación

En este apartado se realizará un análisis detallado del funcionamiento básico de la aplicación. Abordando de manera ordenada todas las funcionalidades que componen dicho sistema. De esta manera se facilitará a comprensión además de ofrecer una visión mas clara de como es una iteración tipo en la platadorma. Para ello dividiremos este capitulo en tres, una división por cada rol y mostraremos cuales son las iteraciones posibles y con que deben interactuar para acceder a ellas.

Usando este enfoque por roles se permitirá, de manera mas clara y concisa, conocer como interactúan los diferentes usuarios con la aplicación, mostrando sus funcionalidades comunes como exclusivas.

5.2.1 Administrador

El administrador no tiene forma de registrarse esto se debe a el usuario que tuviese dicho rol recibiría las credenciales de forma externa.

Primero cuando acceda a la aplicación se encontrará un formulario donde deberá insertar su dni y contraseña ademas de seleccionar su rol, en este caso administrador. Una vez inicie sesión verá dos paneles y un botón.



The image shows a login form titled "Iniciar Sesión" on a light teal background. It contains three input fields: "DNI" with the value "1234", "Contraseña" with masked characters "....", and "Rol" with a dropdown menu showing "Administrador". Below these fields is a dark teal "Ingresar" button. At the bottom, there is a link that says "¿Eres abogado? Solicita un usuario".

El primer panel corresponde con los abogados, mostrando sus datos y otro botón extra para ver los clientes. Este botón si se aprieta por segunda vez los ocultará. En cuanto al segundo panel ahí verá las solicitudes y en caso que exista alguna podrá aceptarla o denegarla. Independientemente de que la acepte o no esa solicitud se borrará a parte de la acción que deba hacer.

Cerrar sesión

Saludos Administrador

Listado de Abogados

DNI	Nombre	Correo	Acciones
1235	Elisa	laura.perez@abogados.com	Ver Clientes
800973775	Daniel Moreno Ortiz de Urbina	dnl.moreno@gmail.com	Ver Clientes

Solicitudes de Registro

Email	Acciones
-------	----------

Listado de Abogados

DNI	Nombre	Correo	Acciones
1235	Elisa	laura.perez@abogados.com	Ocultar Clientes
800973775	Daniel Moreno Ortiz de Urbina	dnl.moreno@gmail.com	Ver Clientes

Clientes del abogado 1235 - 1235

DNI	Nombre	Correo	Teléfono
12345	Juan Pérez	juan.perez@example.com	600123456
1236	Elisa	laura.perez@abogados.com	123456789
123456	Juan Pérez	juan@example.com	600123456
12345678	daniel	dnl.moreno@gmail.com	

Saludos Administrador

Listado de Abogados

DNI	Nombre	Correo	Acciones
1235	Elisa	laura.perez@abogados.com	Ver Clientes
800973775	Daniel Moreno Ortiz de Urbina	dnl.moreno@gmail.com	Ver Clientes

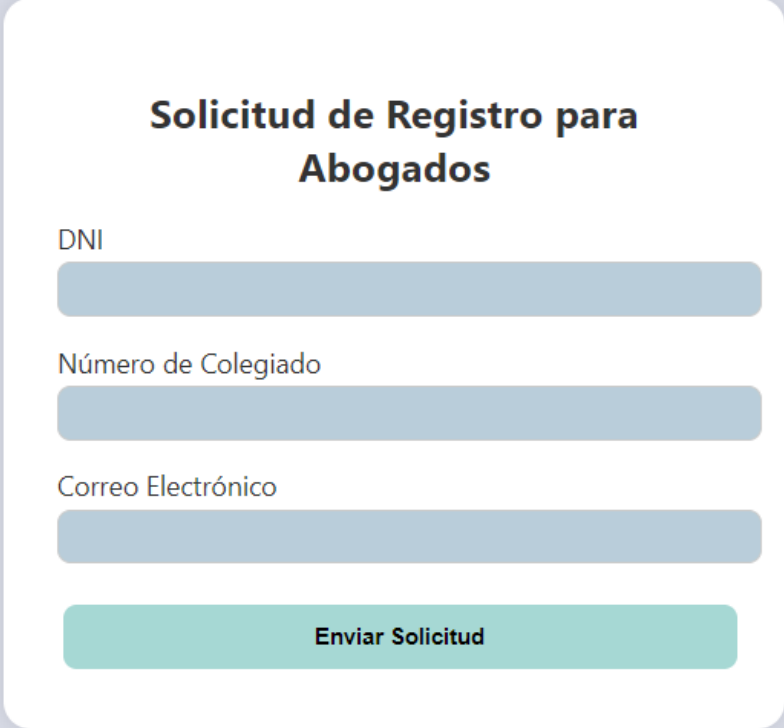
Solicitudes de Registro

Email	Acciones
urbn.apollo76@gmail.com	Aceptar Eliminar

Finalmente un botón arriba a la derecha le cerrará sesión y lo enviará a iniciar sesión.

5.2.2 Abogado

El abogado puede registrarse, para ello en el formulario de inicio de sesión puedes encontrar un enlace que redirige a otro formulario diferente que solo pide numero de colegiado y correo. Al rellenarlo y mandarlo el administrador recibirá la solicitud.



The image shows a web form titled "Solicitud de Registro para Abogados". It is set against a light purple background. The form itself is a white rounded rectangle. It contains three input fields with light blue borders and rounded ends, each preceded by a label. The first label is "DNI", the second is "Número de Colegiado", and the third is "Correo Electrónico". Below these fields is a teal-colored button with the text "Enviar Solicitud" in white.

Solicitud de Registro para Abogados

DNI

Número de Colegiado

Correo Electrónico

Enviar Solicitud

Si la solicitud es aceptada recibirás un correo de forma automática con un enlace que te redirigirá a un formulario donde podrás darte de alta. Una vez hecho si vas a iniciar sesión y metes bien tus credenciales accederás a la pagina de abogado.



Registro de Abogado

DNI

Número de Colegiado

Nombre Completo

Teléfono

Correo

Contraseña

Registrar

The image shows a registration form for a lawyer. It has a title 'Registro de Abogado' at the top. Below the title are six input fields: 'DNI', 'Número de Colegiado', 'Nombre Completo', 'Teléfono', 'Correo', and 'Contraseña'. Each field is represented by a light blue rounded rectangle. At the bottom of the form is a green button labeled 'Registrar'.

En la pagina de abogado el puede ver sus clientes, de la misma forma que un administrador podía ver los clientes de un abogado, el abogado puede ver los documentos de un cliente. Estos documentos puede descargarlos o mostrarlos como presentados. Para mostrarlo como presentados cada documento tiene un botón que al pulsarlo añade 15 días a la fecha actual, siendo esta una fecha aproximada de respuesta del juzgado.

Bienvenido, Abogado

Cientes asociados

Juan Pérez - 12345 - [juan.perez@example.com](#)

Ver documentos

Elisa - 1236 - [laura.perez@abogados.com](#)

Ver documentos

Juan Pérez - 123456 - [juan@example.com](#)

Ver documentos

daniel - 12345678 - [dnl.moreno@gmail.com](#)

Ver documentos

Documentos del cliente 12345678

ID	Nombre	URL	Fecha Entrega	Fecha Respuesta	Acciones
13	carta_ISFAS.jpg	Ver documento	2025-05-30	2025-06-14	<div>Descargar</div>

Registrar nuevo cliente

Por ultimo el abogado puede añadir el correo electrónico de un futuro cliente con un desplegable en la parte inferior de la pantalla. Al pulsar un botón se muestra un formulario de un campo para añadir el correo electrónico de un futuro cliente.

Ocultar formulario de registro

Correo del cliente

Enviar enlace de registro

Finalmente también tiene un botón para cerrar sesión.

5.2.3 Cliente

El cliente no puede darse de alta de forma automática sino que es el abogado quien debe insertar el correo como se ha dicho anteriormente.



Registro de Cliente

Nombre:

DNI:

Correo:

Contraseña:

Registrar

En caso que el correo sea enviado el cliente recibirá un mensaje con una url. Esta url es diferente a la del registro de el abogado por lo siguiente, viene relacionado con un token con el dni del abogado. De esta forma, al rellenar el formulario el cliente y darse de alta será relacionado directamente con su abogado.

El cliente al iniciar sesión podrá subir archivos, ver el estado de estos y cerrar sesión.

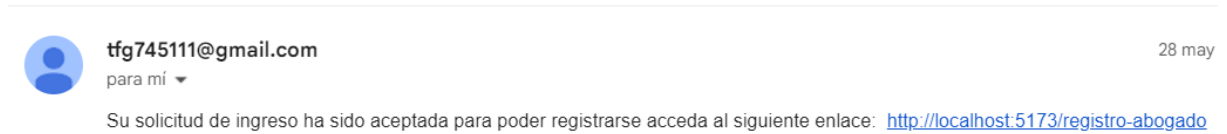


Mis Documentos

ID	Nombre	Fecha Entrega	Fecha Respuesta	Acciones
13	carta_ISFAS.jpg	2025-05-30	2025-06-14	Descargar

Seleccionar archivo | Ningún archivo seleccionado | Subir Documento

Independientemente de con cual rol te registres te llegará el mismo correo el cual tiene el siguiente formato.



5.3 Despliegue

A pesar que esta aplicación no tiene forma de desplegarse a priori, simplemente realizando un cambio estructural simple podemos hacer que se despliegue de forma sencilla. Este cambio a pesar de ser sencillo es bastante delicado puesto que es crear un conjunto de variables de entorno en un .env, de esta forma centralizamos las variables con información sensible como las que se encuentran en el propiedades o las url de los fetch en Fetch.jsx. Una vez explicado esto vamos a hablar de las dos formas de despliegue.

- **Despliegue Clásico:** Esta alternativa representa al corto plazo una solución mas sencilla y económica para entornos de tráfico moderado. Un punto a favor reside en que no son necesarios conocimientos avanzados en desarrollo. A pesar de eso tiene mero flexibilidad y escalamiento que Docker
- **Docker:** Es una opción a priori algo mas compleja pero muchísimo mas útil si tenemos en cuenta que los servidores principalmente se encuentran en Linux. Para poder realizar un despliegue en docker tenemos que crear un Dockerfile tanto para el frontend como para el backend y despues un DockerCompose que monte los dos Dockerfile creando tres contenedores, uno con la base de datos, otro que usa de imagen el backend y otro que hace lo mismo con el frontend. De esta forma tenemos una opción sencilla de escalar y desplegar nuestra aplicación.

6. Conclusiones y Lineas futuras

Para finalizar y antes de mencionar la bibliografía daremos paso a las conclusiones obtenidas tras la realización de el proyecto y las posibles lineas futuras que podría tomar.

6.1 Conclusiones

El proyecto a cumplido enormemente los objetivos planteados siendo el despliegue el único que no se ha podido cumplir. A pesar de ello considero un éxito la creación de una aplicación web funcional de gestión documental en el ámbito legal y jurídico, que implementa un sistema de autenticación por roles mediante JWT que diferencia las capacidades entre diferentes roles. Los resultados prácticos muestran un flujo de trabajo diseñado que responde adecuadamente a las necesidades básicas de el sector.

Sin embargo, tras realizar varias pruebas prácticas de funcionamiento, he descubierto puntos de mejora dentro de la aplicación como mensajes de error mas claros y oportunidades de mejora como la implementación de notificaciones entre roles o la validación de formato de los archivos que se suben.

6.2 Lineas Futuras

Como lineas futuras diferenciamos dos grupos, las lineas futuras inmediatas y las lineas futuras a largo plazo.

Dentro de las lineas futuras inmediatas cabe destacar la mejora de los mensajes de error junto con la implementación de docker para un despliegue mas rápido y escalable. Permitiendo que la aplicación en un futuro pueda ser accesible desde todos los dispositivos. Para poder realizarlo de forma eficiente también hay que mejorar los estilos del frontend para que puedan adaptarse a dispositivos móviles.

En cuanto a las lineas futuras a largo plazo veo interesante poder implementar Autofirma dentro del proyecto obteniendo incluso mas seguridad y optimización en el registro de los usuarios. También cabe destacar la implementación de notificaciones entre usuarios.

7. Bibliografía

Finalmente aquí se muestran los recursos bibliográficos usados en este proyecto.

Manuales Docker: <https://docs.docker.com/manuals/> 28/5/25

JavaMail: <https://javaee.github.io/javamail/> 15/5/25

PostgreSQL: <https://www.postgresql.org/docs/> 5/5/25

JWT:

https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html 7/5/25

Spring Security: <https://docs.spring.io/spring-security/reference/> 8/5/25

CORS: <https://docs.spring.io/spring-framework/reference/web/webmvc-cors.html> 12/5/25