

**SANTIAGO FERNÁNDEZ**  
**EHIMATIE**

**EDU-VAULT**





# Índice

1. **Introducción**
2. **Tecnologías utilizadas**
3. **Arquitectura de la aplicación**
4. **Componentes del Frontend**
5. **Componentes del Backend**
6. **Seguridad con JWT**
7. **Manual de Usuario**
8. **Propuestas de mejora**
9. **Requisitos técnicos**
10. **Conclusión**



Este proyecto es una aplicación web de tipo *Single Page Application (SPA)* que permite a los usuarios organizar y gestionar su contenido académico de forma intuitiva y centralizada. A través de esta plataforma, se podrá crear y administrar asignaturas, definir temas relacionados y adjuntar archivos que podrán visualizarse directamente en el navegador.

Esta herramienta está diseñada para estudiantes, docentes o cualquier persona interesada en mantener un control estructurado de su material de estudio. Facilita el acceso rápido a información académica, mejora la organización personal y permite visualizar recursos en línea sin necesidad de descargarlos.

Para conseguirlo se han utilizado diversas herramientas y frameworks, así como una base de datos sql.

Se han intentado seguir buenas prácticas en todos los casos según la herramienta utilizada.

A continuación se procede con la explicación de cómo se han manejado los distintos aspectos que componen la aplicación frontend, backend, seguridad...

## Frontend

Este aspecto se ha desarrollado con el framework/librería React.js junto con Vite.

### Vite

#### Funciones básicas de Vite en una aplicación React

Vite es una herramienta moderna de construcción y desarrollo de aplicaciones web que ha ganado popularidad por su rapidez y eficiencia, especialmente en proyectos basados en frameworks como React. A continuación, se describen las funciones fundamentales que Vite desempeña en una aplicación React, según su documentación oficial y su funcionamiento práctico:

- 1. Servidor de desarrollo rápido:**

Vite ofrece un entorno de desarrollo local que se inicia casi instantáneamente, facilitando la visualización de los cambios en la aplicación de manera inmediata. Esta característica se basa en el uso de módulos ES nativos del navegador, lo que evita la necesidad de empaquetar toda la aplicación antes de su ejecución (Vite, 2023).



2. **Recarga en caliente (Hot Module Replacement - HMR):**

La función HMR permite actualizar únicamente los módulos modificados sin recargar toda la página, lo que acelera el ciclo de desarrollo y mejora la experiencia del desarrollador, ya que mantiene el estado de la aplicación mientras se realizan cambios (Vite, 2023).

3. **Construcción optimizada para producción:**

Al generar la versión para producción, Vite realiza un empaquetado eficiente que minimiza y optimiza los archivos, mejorando el rendimiento y reduciendo el tamaño de los recursos que se envían al cliente (Vite, 2023).

4. **Soporte nativo para módulos ES:**

Durante el desarrollo, Vite aprovecha la compatibilidad del navegador con los módulos ES para cargar directamente los archivos JavaScript sin necesidad de un proceso previo de empaquetado, lo que contribuye a la rapidez del servidor de desarrollo (Vite, 2023).

5. **Configuración sencilla y flexible:**

Vite proporciona una configuración por defecto que es suficiente para la mayoría de los proyectos React, permitiendo una rápida puesta en marcha. Además, ofrece opciones para personalizar la configuración según las necesidades específicas del proyecto (Vite, 2023).

6. **Integración con plugins:**

La arquitectura de Vite soporta plugins que permiten extender sus funcionalidades, facilitando la integración con otras herramientas, librerías y optimizaciones específicas para el desarrollo web moderno (Vite, 2023).

## **React y su uso en una SPA**

React es una biblioteca/framework de JavaScript desarrollada por Facebook que permite construir interfaces de usuario de manera eficiente y declarativa. Se caracteriza por facilitar el desarrollo de aplicaciones web dinámicas y responsivas mediante la creación de componentes reutilizables.

En el contexto de una **Single Page Application (SPA)**, React juega un papel fundamental al gestionar el contenido que se muestra al usuario sin necesidad de recargar la página completa. Esto se logra mediante:



- **Renderizado basado en componentes:** React divide la interfaz en componentes independientes, cada uno con su propio estado y propiedades, lo que facilita la actualización y mantenimiento de la aplicación.
- **Virtual DOM:** React utiliza un DOM virtual que representa una copia ligera del DOM real. Cuando el estado de un componente cambia, React actualiza únicamente las partes necesarias en el DOM real, optimizando el rendimiento y mejorando la experiencia del usuario.

## Uso de React Router en la SPA

React Router es una librería de enrutamiento para aplicaciones React que permite gestionar la navegación dentro de una Single Page Application (SPA) sin recargar la página. Proporciona una estructura declarativa para definir las diferentes rutas y componentes asociados a cada una, facilitando una navegación fluida y dinámica.

En la aplicación desarrollada, se configura React Router de la siguiente manera:

### 1. Configuración principal con `<Router>`:

El componente `<Router>` envuelve toda la aplicación, habilitando el uso de rutas y gestionando la historia del navegador para permitir la navegación.

### 2. Definición de rutas con `<Routes>` y `<Route>`:

Dentro del `<Router>`, se definen las rutas mediante el componente `<Routes>`, que agrupa los diferentes `<Route>` que representan las URLs y los componentes asociados.

### 3. Rutas públicas y privadas:

- La ruta `/login` es pública y utiliza un layout específico (`<AuthLayout>`) que contiene la página de autenticación (`<AuthPage>`). Esto permite separar la interfaz de login del resto de la aplicación.
- Las demás rutas están protegidas mediante el componente `<ProtectedRoute>`, que actúa como un guardián, restringiendo el
- 

acceso a usuarios no autenticados y asegurando que solo usuarios autorizados puedan acceder al contenido principal.



```
<Router>
  You, 4 weeks ago • Feature/Fix: Arreglado apertura de archivos y a...
  <Routes >
    <Route path="/login" element={<AuthLayout/>} >
      <Route index element={<AuthPage/>} />
    </Route>

    <Route path="/" element={<ProtectedRoute ><Layout></Layout> </ProtectedRoute> }>

      <Route index path="subjects" element={<Subjects />} />
      <Route path="/calendar" element={<MyCalendar />} />
      <Route path="/subjects/:subjectUri/topics" element={<Topics />} />
      <Route path="subjects/:subjectUri/topics/:topicUri/files" element={<Files />} />
      <Route path="storage" element={<Storage></Storage>}></Route>

      <Route path="userSettings" element={<UserSettings />} />
      <Route path="favourites" element={<Favourites />} />
    </Route>
  </Routes>
</Router>
```

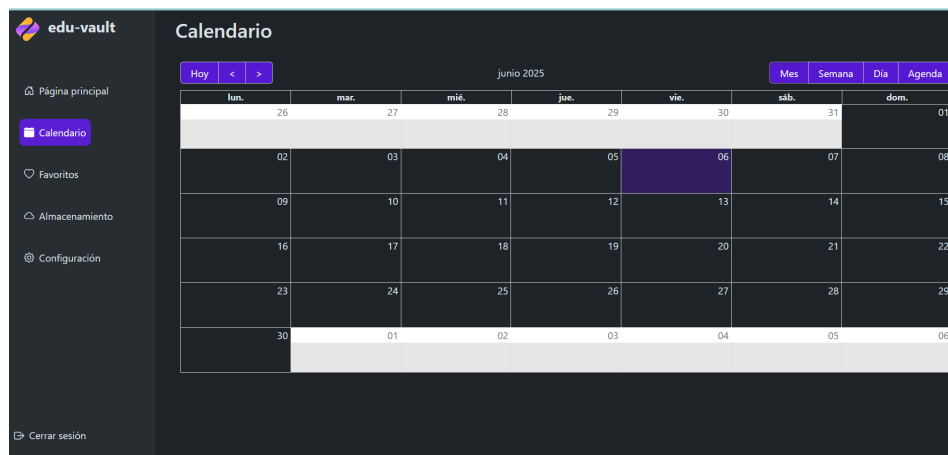
#### 4. Rutas anidadas y parámetros dinámicos:

- La ruta principal / muestra el componente <Subjects>.
  - La aplicación utiliza rutas con parámetros dinámicos para mostrar información específica, por ejemplo:
    - subject/:subjectId/topics/ carga el componente <Topics>, mostrando los temas de una asignatura concreta.
    - subject/:subjectId/topics/:topicId/files muestra los archivos asociados a un tema específico mediante el componente <Files>.
- Esta estructura permite una navegación detallada y contextual dentro de la SPA.

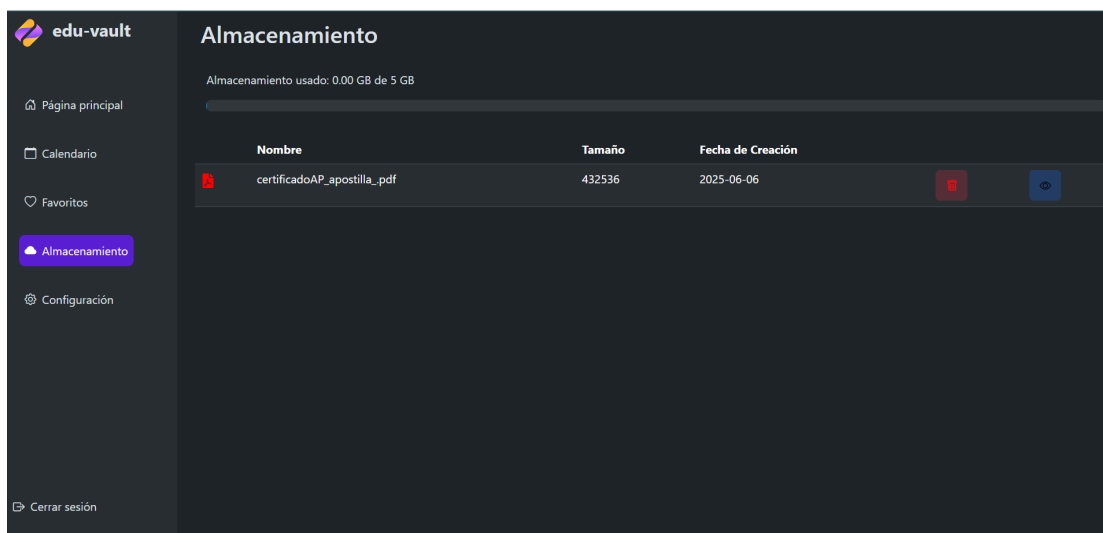


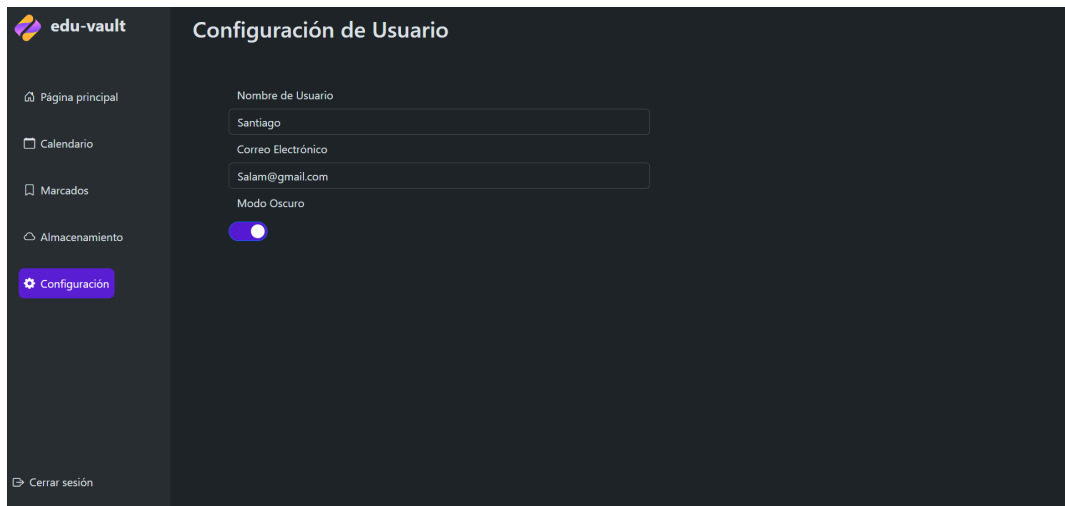
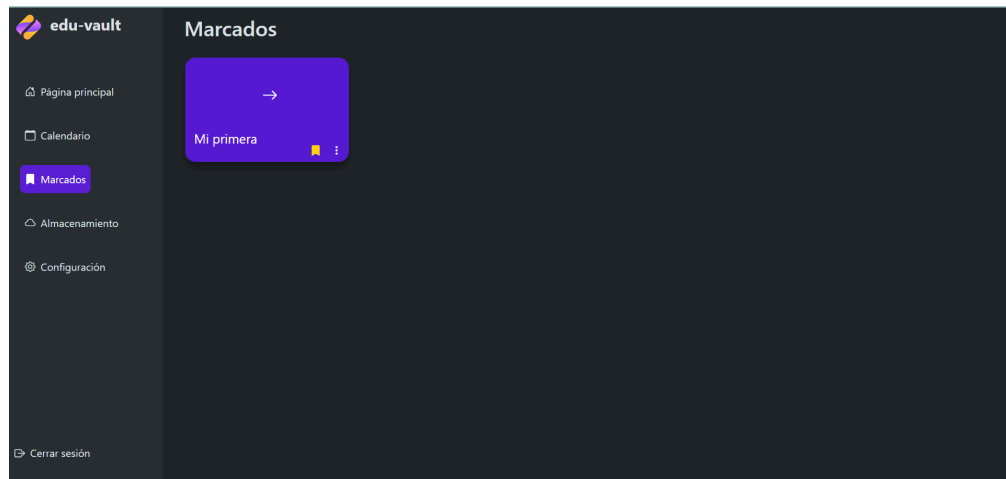
## 5. Otras rutas relevantes:

- /calendar muestra un calendario personalizado (<MyCalendar>).



- /storage, /userSettings y /favourites permiten al usuario acceder a almacenamiento personal, configuración y favoritos, respectivamente, cada uno asociado a su componente correspondiente.





El componente `<Outlet />` forma parte de la arquitectura de **rutas anidadas** en React Router y cumple una función esencial en el renderizado de componentes secundarios dentro de un diseño principal o layout.

Outlet actúa como un **marcador de posición** que indica **dónde se debe renderizar el componente hijo** correspondiente a una ruta anidada. Es decir, dentro de un componente de layout, `<Outlet />` es el lugar donde se insertará el contenido de las rutas hijas definidas en la configuración de rutas.





## Hooks en React

Los **Hooks** son una característica introducida en React a partir de la versión 16.8 que permiten utilizar el estado y otras funcionalidades de React en componentes funcionales, sin necesidad de recurrir a clases.

Con los Hooks, es posible manejar **estado local**, realizar **efectos secundarios**, gestionar **referencias** y reutilizar lógica entre componentes, todo dentro de funciones simples y concisas.

---

### Principales Hooks utilizados

- **useState**  
Permite gestionar el estado interno de un componente funcional, facilitando la definición y actualización de variables que pueden cambiar a lo largo del ciclo de vida del componente.
- **useEffect**  
Se emplea para ejecutar efectos secundarios en los componentes, como por ejemplo llamadas a servicios externos, sincronización con APIs o manipulación del DOM. Además, permite controlar el momento en el que dichos efectos deben ejecutarse, ya sea al montar, actualizar o desmontar el componente.
- **useContext**  
Facilita el acceso a valores globales definidos mediante el Context API de React. Esto resulta útil para compartir información como el estado de autenticación del usuario o preferencias generales entre múltiples componentes, sin necesidad de pasar props manualmente.
- **useRef**  
Proporciona una forma de crear referencias persistentes entre renderizados. Puede utilizarse, por ejemplo, para acceder directamente a elementos del DOM o para almacenar valores que no deben desencadenar una nueva renderización al cambiar.
- **useNavigate y useParams** (React Router)  
Son hooks específicos del sistema de enrutamiento. useNavigate permite redirigir al usuario de manera programática entre rutas, mientras que



useParams ofrece acceso a los parámetros dinámicos definidos en las URLs, como identificadores de recursos.

---

## Ventajas de los Hooks

- Promueven una **mayor modularidad** y la **reutilización de lógica** entre componentes.
  - Eliminan la necesidad de escribir componentes de clase, simplificando el código.
  - Mejoran la **legibilidad**, el mantenimiento y la organización del proyecto.
  - Permiten separar claramente la lógica de negocio de la presentación visual del componente.
- 

## Uso de Hooks en la SPA desarrollada

Durante el desarrollo de la SPA, los Hooks han sido esenciales para gestionar el estado de la interfaz, realizar peticiones asíncronas, acceder a rutas dinámicas, y mantener la lógica de autenticación de los usuarios. Su utilización ha permitido estructurar la aplicación de forma clara, eficiente y escalable.

## Gestión de datos asíncronos con TanStack Query en la SPA

En lugar de utilizar el Hook `useEffect` para gestionar las llamadas a servicios externos, en esta aplicación se ha optado por emplear **TanStack Query** (anteriormente React Query), una biblioteca especializada en la gestión de datos asíncronos en aplicaciones React.

Esta herramienta permite manejar el ciclo completo de obtención, actualización y sincronización de datos remotos de forma eficiente, declarativa y automática,



reduciendo la necesidad de gestionar manualmente estados de carga, errores o sincronización con el backend.

---

## Integración de TanStack Query: el caso de useSubjects

La lógica relacionada con la obtención, creación, edición y eliminación de asignaturas se encapsula en un **custom hook** llamado useSubjects, el cual permite centralizar todas las interacciones con la API relacionadas con las asignaturas.

Dentro de este hook:

- **useMutation** se utiliza para gestionar operaciones que modifican los datos (crear, editar, borrar o marcar como favorita una asignatura). Esta función proporciona estados como isLoading, isError o isSuccess y permite definir comportamientos específicos en los callbacks onSuccess y onError.
- **useQueryClient** se emplea para acceder al cliente global de consultas y, especialmente, para **invalidar queries** mediante invalidateQueries, forzando una nueva obtención de datos actualizados tras una mutación exitosa.
- Se define una serie de funciones handle\* (por ejemplo, handleAddSubject, handleDeleteSubject) que actúan como controladores lógicos y que son exportadas para ser utilizadas directamente desde los componentes React. Esto mantiene la lógica desacoplada de la interfaz.

Gracias a esta estructura, se mejora la **organización del código** y se evita la redundancia y la complejidad de gestionar múltiples useState y useEffect por separado.

---

## Relación con otros hooks personalizados (useTopics, useEvents, useFiles)

De forma análoga, se emplean otros hooks como useTopics, useFiles y useEvents para encapsular la lógica relacionada con los temas y eventos dentro de su propio contexto. Todos estos hooks siguen un patrón común:



- Acceso al **contexto de autenticación** mediante useContext para incluir el token en las cabeceras.
- Uso de **mutaciones controladas** con useMutation para modificar datos de forma segura.
- Actualización automática del estado global mediante **invalidación de queries** en el QueryClient.

Esta arquitectura modular facilita la **escalabilidad** y el **mantenimiento** de la aplicación, permitiendo reutilizar lógica, mejorar la legibilidad y centralizar las operaciones relacionadas con los datos.

## Context API en React

React Context es una herramienta incorporada en React que permite **compartir valores globales** entre componentes, sin necesidad de pasarlos explícitamente como props a lo largo de toda la jerarquía del árbol de componentes.

Es especialmente útil en aplicaciones grandes o de arquitectura SPA, donde se necesita acceder a datos como la sesión del usuario, temas de la interfaz, preferencias o configuración general desde múltiples puntos de la aplicación.

---

## Elementos principales del Context API

- **createContext()**  
Se utiliza para crear un contexto. Este objeto contiene dos componentes principales:
  - Un Provider, que permite definir el valor compartido.
  - Un Consumer (opcional), que permite acceder al valor cuando no se usa el hook useContext.



- **Provider**

Es el componente que **envuelve** la parte de la aplicación donde se desea que el contexto esté disponible. Recibe una propiedad `value` con el contenido que será accesible desde los componentes descendientes.

- **useContext()**

Es un Hook que permite a los componentes funcionales acceder directamente al valor del contexto sin necesidad de un Consumer tradicional.

---

## Implementación: TokenContext.js

En esta aplicación, se ha definido un contexto llamado `TokenContext`, encargado de **gestionar el token JWT de autenticación** del usuario. Este contexto es fundamental para mantener la sesión activa y validar el acceso a rutas protegidas o recursos restringidos.

### Principales responsabilidades de TokenContext

- **Almacenamiento y persistencia del token**

El token se obtiene desde el almacenamiento local (`localStorage`) al iniciar la aplicación y se almacena en el estado React mediante `useState`, lo cual

permite que todos los componentes accedan a su valor actualizado.

- **Verificación de validez del token**

Se definen funciones auxiliares (`isTokenValid`, `isTokenExpired`, `isTokenPresent`) que usan `jwtDecode` para interpretar el contenido del token y comprobar si ha expirado en base al tiempo (`exp`). Esto permite proteger partes de la aplicación y redirigir automáticamente si la sesión ya no es válida.

- **Actualización del token**

La función `setNewToken` permite establecer un nuevo token (por ejemplo, al iniciar sesión o renovar la sesión) y actualizar el almacenamiento.



- **Gestión de mensajes personalizados**

Se incluye también un estado `expiredMsg` que puede utilizarse para informar al usuario si su sesión ha caducado o si necesita volver a autenticarse.

### **Ejemplo de uso en la aplicación**

El `TokenProvider` se utiliza para **envolver el árbol de la aplicación** (normalmente desde el componente principal como `App.js`), lo que permite que cualquier componente descendiente acceda al valor del token o verifique su validez mediante el hook `useContext`.

Esto permite un sistema de autenticación **centralizado**, limpio y reutilizable, evitando tener que propagar el token manualmente o duplicar lógica de validación.

Esta arquitectura permite:

- Centralizar la lógica de seguridad.
- Facilitar el acceso a información global.
- Evitar la repetición de lógica en cada componente.

Este patrón se integra perfectamente con otras herramientas como **React Router** para proteger rutas o **TanStack Query** para añadir cabeceras de autorización en peticiones asíncronas.

## **Arquitectura de las vistas principales: Subjects, Topics y Files**

La interfaz de usuario de esta aplicación SPA (Single Page Application), desarrollada con React, está compuesta por varias vistas funcionales, entre las que destacan: Subjects, Topics y Files. Todas ellas comparten una arquitectura común basada en componentes reutilizables, gestión de estado global mediante Context API y acceso eficiente a datos mediante TanStack Query (anteriormente conocida como React Query).

### **Estructura general**

Cada vista sigue un patrón común que incluye:



- **Acceso controlado por autenticación** (verificación de token).
- **Obtención y gestión de datos desde el backend** usando hooks personalizados.
- **Renderizado condicional** según el estado de la petición (cargando, error o datos disponibles).

### Ejemplo:

```
src > pages > Subjects.jsx > Subjects
18  const Subjects = () => {
31    // Usamos React Query para hacer la petición a la API y manejar loading/error/data automáticamente
32    const { isLoading, isError, data, error } = useQuery({
33      queryKey: ["subjects"], // Clave para caché y refetching
34      queryFn: getSubjects,
35    });
36
37    // Mientras se cargan los datos, mostramos una página de carga
38    if (isLoading) {
39      return <LoadingPage />;
40    }
41
42    // Si ocurre un error en la consulta, mostramos el mensaje
43    if (isError) {
44      // Redirige a la página de login si hay un error
45      return (
46        <ErrorPage
47          errorTitle={error.message}
48          errorMessage={"Ha ocurrido un problema al cargar tus asignaturas"}
49        />
50      );
51    }
52  }
```

- **Representación visual mediante componentes reutilizables** como tarjetas, botones o listas.

A continuación, se detalla el funcionamiento de la vista Subjects, que sirve como modelo para el resto.



## Vista Subjects

Esta página se encarga de mostrar al usuario las asignaturas disponibles, permitiendo además añadir, eliminar o editar dichas asignaturas.

### 1. Validación de autenticación

Se emplea el `TokenContext`, implementado mediante la `Context API` de `React`, para acceder al token de autenticación del usuario y verificar su validez antes de realizar cualquier operación con la API. Esto garantiza que solo usuarios autenticados puedan acceder a los recursos protegidos.

### 2. Obtención de datos con `TanStack Query`

La lógica para obtener las asignaturas se abstrae mediante un **custom hook** (`useSubjects`), que define las funciones de acceso a la API. Este hook es utilizado junto con `useQuery`, que gestiona de forma automática el ciclo de vida de la consulta:

- `isLoading`: indica si los datos están siendo cargados.
- `isError`: se activa en caso de error en la petición.
- `data`: contiene la información devuelta por el backend (lista de asignaturas).
- `enabled`: permite condicionar la ejecución de la query, en este caso, solo si el token es válido.

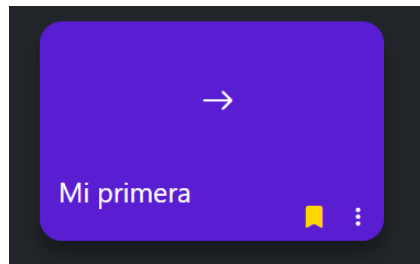
Esta integración permite una gestión eficiente del estado, el caché y el refetching de datos.

### 3. Interfaz dinámica

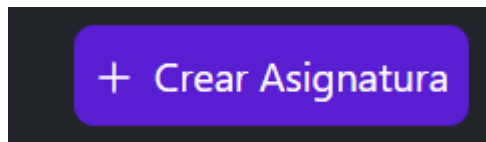
El contenido de la página varía según el estado de la consulta:

- Si los datos están en proceso de carga, se muestra una vista `LoadingPage`.
- Si ocurre un error, se muestra una página de error personalizada (`ErrorPage`) con el mensaje correspondiente.
- Si la consulta es exitosa, se renderiza la lista de asignaturas mediante el componente **SubjectCard**:





También se incluye un componente **AddIconButton** que permite al usuario añadir nuevas asignaturas, mostrando un modal contextual basado en el tipo de recurso.



## Reutilización de la estructura en otras vistas

Este mismo patrón de diseño se replica en las vistas Topics y Files, lo que permite mantener una arquitectura **modular, consistente y escalable**. Cambia únicamente la lógica del hook (useTopics, useFiles) y el componente visual utilizado (TopicCard, FileCard, etc.).

## Ventajas de este enfoque

- **Mantenibilidad:** Separar lógica de negocio y presentación facilita la depuración y evolución de la aplicación.
- **Reutilización:** Los hooks y componentes pueden ser utilizados en múltiples partes del proyecto sin duplicar código.
- **Escalabilidad:** Es sencillo añadir nuevas vistas o recursos que sigan el mismo patrón.
- **Seguridad y control de acceso:** La verificación del token protege los endpoints y permite controlar la experiencia del usuario.



## Vista del calendario con React Big Calendar

Una de las páginas más destacadas de la aplicación es el calendario interactivo, accesible desde la ruta /calendar. Esta vista permite al usuario visualizar, crear y eliminar eventos a través de una interfaz moderna y adaptativa.

### Librería utilizada: React Big Calendar

Para implementar esta funcionalidad, se ha utilizado la librería **React Big Calendar**, una solución ampliamente adoptada para gestionar calendarios dentro de aplicaciones React. Esta librería ofrece soporte completo para diferentes vistas (día, semana, mes, agenda) y permite interacción directa con los eventos y franjas horarias.

Para la gestión de fechas, en lugar de usar moment.js, se ha optado por **dayjs**, una librería más ligera y moderna. A través del adaptador dayjsLocalizer, se configura el calendario para que utilice dayjs como sistema de fechas y se adapta al idioma español.

```
const localizer = dayjsLocalizer(dayjs);
```

### Gestión de eventos

El comportamiento principal del calendario se centra en dos interacciones clave:

- **Seleccionar una franja vacía (slot):** activa un formulario para crear un nuevo evento.
- **Seleccionar un evento existente:** activa una confirmación para eliminarlo.

Ambas acciones están unificadas mediante un sistema de **modales contextuales**, que se muestran dinámicamente según la acción seleccionada.

### Uso de React Context y modales

La gestión del modal se realiza a través del CreationContext, un contexto React que controla el estado de los formularios emergentes en toda la aplicación. Esto permite abrir un modal con diferentes configuraciones (campos, mensajes, acciones) sin duplicar lógica.



La opción seleccionada (`selectedOption`) se actualiza dependiendo de si se desea crear o eliminar un evento, y se renderiza uno de los dos componentes:

- `ModalTemplate`: para introducir los datos del nuevo evento.
- `ConfirmActionModal`: para confirmar la eliminación de un evento.

## Obtención de eventos desde la API

La obtención de los eventos almacenados en el servidor se realiza mediante el hook `useEvents`, que a su vez es consumido con `useQuery` de `TanStack Query`.

Los datos obtenidos desde la API se transforman para adaptarse al formato requerido por `react-big-calendar`, que exige que los campos `start` y `end` sean objetos `Date`.

## Localización e interfaz

Se configura una traducción personalizada para que todos los textos del calendario aparezcan en español, utilizando la opción `messages` del componente `Calendar`. Esto mejora significativamente la experiencia de usuario y la adecuación a un entorno hispanohablante.

## Página Favourites

El componente `Favourites` tiene como función principal mostrar al usuario las asignaturas que ha marcado previamente como favoritas. Para ello, se realiza una consulta a las asignaturas disponibles, y posteriormente se filtran aquellas cuyo atributo `isFav` está activado.

## Lógica de funcionamiento

Cuando el componente se monta:

- Se comprueba que el usuario tenga una sesión válida mediante el contexto de autenticación.
- Se obtiene la lista de asignaturas desde un hook personalizado que centraliza la lógica de acceso a recursos.
- Si las asignaturas ya se han cargado previamente en otra parte de la aplicación, **se utilizarán directamente desde la caché**, lo que mejora el



rendimiento y evita peticiones innecesarias al servidor.

Una vez obtenidas las asignaturas:

- Se filtran las que tienen isFav activado.
- Se representan en pantalla mediante el componente SubjectCard, que muestra su nombre y permite realizar acciones sobre ellas.

### Beneficios

- **Experiencia personalizada:** solo se muestran asignaturas relevantes para el usuario.
- **Eficiencia:** se aprovechan datos en caché cuando están disponibles.
- **Modularidad:** tanto la lógica como la presentación están divididas en componentes y hooks reutilizables.
- **Escalabilidad:** si en el futuro se añaden más atributos o filtros, el sistema está preparado para adaptarse.

## Componente Layout: Estructura base de la interfaz

El componente **Layout** define la estructura principal y común para todas las páginas de la aplicación, asegurando una organización coherente y una experiencia de usuario uniforme.

### Organización visual y responsividad

- Se utiliza la **grilla de Bootstrap** para crear un diseño adaptable a diferentes tamaños de pantalla mediante clases como col-md-4, col-xl-2, col-lg-3, etc.
- El diseño está dividido en dos partes principales:
  - **Sidebar (barra lateral):** ubicado en la parte izquierda, ocupa una proporción menor del ancho según el tamaño de pantalla. Este
  -



componente contiene la navegación principal o menús.

- **Contenido principal:** ocupa el espacio restante y contiene el encabezado (Header) y el área donde se renderiza el contenido dinámico (Content).

```
//Contenedor definido con container-fluid de bootstrap para aplicar responsividad
<>
  <div className="row ">
    <aside className="col-md-4 col-xl-2 col-lg-3 col-sm-2 col-3">
      <Sidebar />
    </aside>
    <div className="col-9 col-md-8 col-xl-10 col-lg-9 col-sm-10 mt-1">
      <header >
        <Header />
      </header>
      <main className="">
        <Content /> { /* React Router renderizará aquí el contenido dinámico */}
      </main>
    </div>
  </div>
</div>
```

## Funcionalidad

- El **Sidebar** permanece visible en todo momento y permite cambiar entre secciones o funcionalidades de la aplicación.
- El **Header** proporciona una barra de búsqueda global.
- El componente **Content** es donde se renderizan los diferentes componentes o páginas según la ruta actual, aprovechando la capacidad de React Router para renderizado dinámico.

## Ventajas de usar un Layout

- Permite **reutilizar la misma estructura** sin repetir código en cada página.
- Facilita mantener una experiencia visual consistente.
- Aprovecha la **responsividad de Bootstrap** para que la interfaz se adapte a móviles, tablets y escritorios.



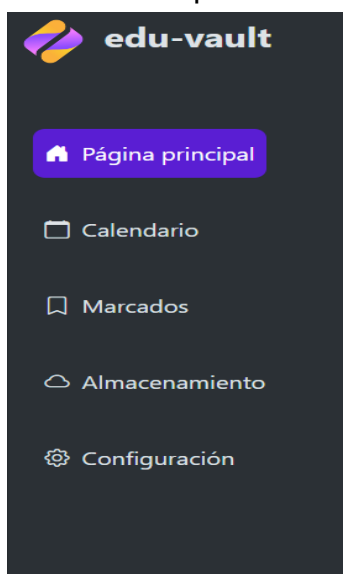
- Facilita la integración con React Router para cargar contenido dinámico sin recargar toda la página.

## Componente Sidebar: Navegación lateral de la aplicación

El **Sidebar** es un componente fundamental para la navegación dentro de la SPA (Single Page Application). Se encarga de mostrar un menú lateral con diferentes opciones que permiten al usuario desplazarse entre las distintas secciones de la aplicación.

### Funcionalidad principal

- **Navegación:** Cada elemento del menú es un enlace (Link) que apunta a una ruta específica usando React Router, lo que permite cambiar la vista sin recargar la página.
- **Botones personalizados:** Los botones son componentes reutilizables (Navbutton) que reciben propiedades para mostrar texto, iconos y gestionar el estado de selección.
- **Estado local para selección:** El componente mantiene en su estado interno qué botón está actualmente seleccionado (isSelected). Esto permite mostrar visualmente qué sección está activa.





## Persistencia del estado seleccionado

- Para mejorar la experiencia, el botón seleccionado se guarda en localStorage. Así, al recargar la página o volver a la aplicación, se mantiene marcada la última sección visitada.
- Este valor se carga al montar el componente con useEffect y se actualiza cada vez que el usuario hace clic en un botón.

## Integración con autenticación y navegación

- El componente utiliza un hook de autenticación personalizado (useAuth) para gestionar la sesión.
- En la parte inferior del menú, el botón **Cerrar sesión** ejecuta la función logout() que limpia la sesión y redirige al usuario a la página de login usando useNavigate.

## Organización visual y estilo

- El diseño usa clases de Bootstrap y estilos personalizados para posicionar y dar formato a los elementos del menú.
- El menú está estructurado en dos bloques:
  - Opciones principales de navegación (Página principal, Calendario, Favoritos, Almacenamiento).
  - Opciones de usuario (Configuración y Cerrar sesión), ubicadas en la parte inferior para fácil acceso.

## Ventajas del enfoque

- La navegación es rápida y fluida, típica de las SPA.
- La selección persistente mejora la usabilidad y ayuda al usuario a orientarse en la aplicación.



- Al centralizar la navegación en un componente, se facilita su mantenimiento y escalabilidad.
- La separación en componentes reutilizables como Navbutton ayuda a mantener el código limpio y modular.

## Componente Content: Gestión del contenido dinámico y control de sesión

El componente **Content** es un contenedor principal donde se renderizan las diferentes páginas o vistas hijas según la ruta actual en la aplicación. Utiliza React Router para manejar la navegación y el control del acceso basado en la validez del token de autenticación.

---

### Funcionalidad principal

- **Renderizado dinámico con <Outlet />:**  
El componente usa el componente <Outlet /> de React Router, que actúa como un placeholder donde se renderizan las rutas hijas definidas en la configuración del router. Esto permite que el contenido cambie dinámicamente sin recargar toda la página, manteniendo la estructura de la interfaz intacta.
- **Control de sesión y seguridad:**  
El componente usa el contexto de autenticación (TokenContext) para:
  - Comprobar si hay un token presente (isTokenPresent).
  - Verificar si el token ha expirado (isTokenExpired).
  - Redirigir al usuario a la página de login si el token es inválido o ha expirado, asegurando que solo usuarios autenticados puedan acceder al contenido protegido.





- Mostrar un mensaje de expiración de sesión mediante `setExpiredMsg` para informar al usuario que debe volver a iniciar sesión.

---

## Uso de hooks de React Router y React

- **useLocation:**  
Se usa para detectar cambios en la ruta actual (`location.pathname`). Esto es útil porque cada vez que cambia la ubicación (p.ej., el usuario navega a otra página), se ejecuta el efecto para comprobar el estado del token y proteger la ruta.
- **useEffect:**  
Cada vez que la ubicación cambia, el `useEffect` ejecuta la lógica para verificar la validez del token:
  - Si no hay token o está expirado, se redirige al usuario a `/login`.
  - Además, se establece un mensaje que indica que el token expiró para que se pueda mostrar un aviso en la interfaz.
- **useNavigate:**  
Hook que permite realizar navegación programática, en este caso para redirigir a la página de login.

---

## Ventajas y objetivos

- **Protección de rutas:** Este componente actúa como una "guardia" para las rutas hijas, evitando el acceso a usuarios no autenticados.
- **Renderizado eficiente:** Al usar `<Outlet />` el contenido cambia sin recargar la página, mejorando la experiencia de usuario.
- **Gestión centralizada:** Al controlar el token y la sesión en un solo lugar, se evita repetir esta lógica en cada página.
- **Mejora UX:** Al mostrar mensajes de expiración, se informa al usuario de forma clara sobre el estado de su sesión.



## Componente AuthPage: Gestión de inicio de sesión y registro

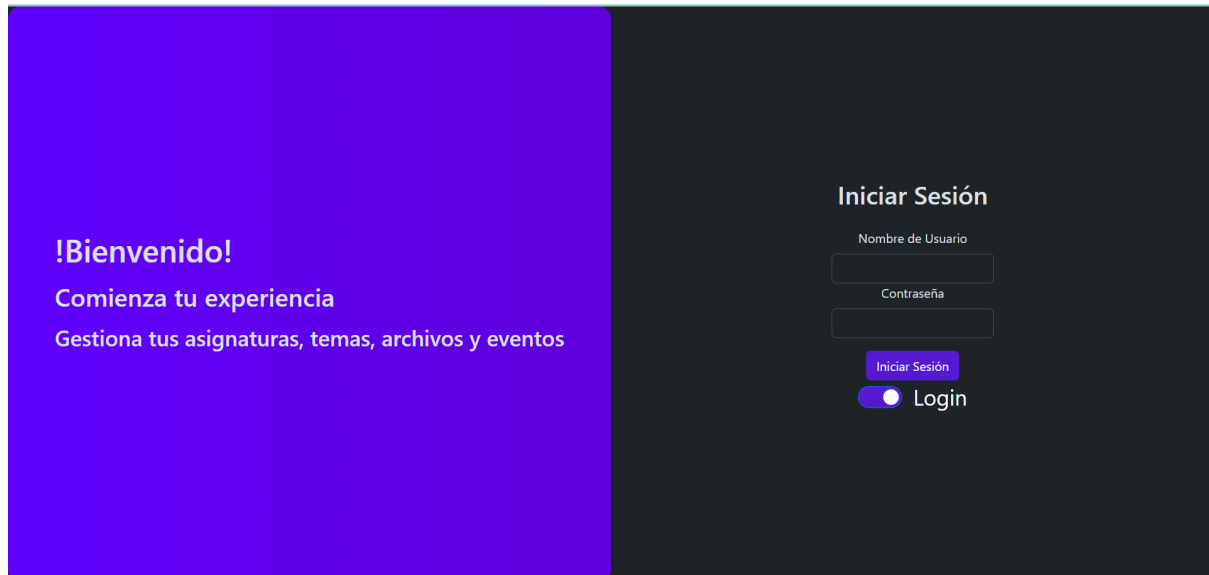
El componente **AuthPage** es la página principal encargada de gestionar la autenticación del usuario. En esta interfaz se permite tanto **iniciar sesión** como **registrarse**, integrando lógica de sesión, validaciones y gestión de tokens para proteger el acceso a la aplicación.

---

### Objetivo principal

Este componente busca proporcionar una **interfaz de autenticación unificada** que permite alternar entre los formularios de **login** y **registro**, ofreciendo una experiencia de usuario moderna, clara y visualmente separada en dos bloques.

The screenshot displays the AuthPage interface, which is split into two main sections. The left section has a solid blue background and contains the following text: **!Bienvenido!**, **Comienza tu experiencia**, and **Gestiona tus asignaturas, temas, archivos y eventos**. The right section has a dark gray background and contains a registration form titled **Registrarse**. The form includes three input fields labeled **Nombre de Usuario**, **Contraseña**, and **Correo electrónico**. Below these fields is a blue button labeled **Registrarse**. At the bottom of the right section, there is a toggle switch that is currently turned on, with the text **Sign up** next to it.



---

## Estructura y funcionamiento

### 1. Estado local form

- Se utiliza useState para controlar qué formulario se está mostrando.
  - true: se muestra el formulario de **login**.
  - false: se muestra el formulario de **registro**.
- El estado se modifica con un botón de alternancia (ToggleButton), permitiendo cambiar de vista con una sola acción.

---

### 2. Contexto de autenticación

Se consume el **contexto TokenContext** para detectar si el token ha expirado, mostrando un mensaje de error informativo:



```
const { expiredMsg } = useContext(TokenContext);
```

Si hay un mensaje de expiración (expiredMsg), se muestra una alerta con una clase de Bootstrap:

```
const alert = expiredMsg !== "" ? "alert alert-danger" : "";
```

---

### 3. Diseño visual dividido

La interfaz está dividida en **dos columnas**:

- **Columna izquierda:**
  - Fondo con degradado (bg-primary-gradient).
  - Mensajes de bienvenida.
  - Componente ToggleButton que alterna entre login y registro.
- **Columna derecha:**
  - Contiene el título y el formulario correspondiente.
  - Se muestra el mensaje de expiración de token si es necesario.
  - Condicionalmente muestra:
    - <LoginForm /> si form es true.
    - <RegisterForm /> si form es false.



#### 4. Componentes usados

- **ToggleButton:** Componente reutilizable que permite cambiar entre login y registro.
- **LoginForm:** Formulario con campos de usuario y contraseña, validado con react-hook-form.
- **RegisterForm:** Formulario de registro con validaciones adicionales.
- **AuthButton:** Botón reutilizable para confirmar acciones (login o registro).

#### Hook useForm – Gestión eficiente de formularios en React

En esta aplicación se utiliza la librería react-hook-form para gestionar los formularios de manera eficiente, optimizando el rendimiento y simplificando la validación y el manejo de errores. Su principal hook es useForm, el cual proporciona múltiples utilidades para controlar el ciclo de vida del formulario sin necesidad de re-renderizados innecesarios.

#### Objetivo

El hook useForm permite registrar campos, validar datos de entrada, capturar errores y manejar el envío de formularios de forma declarativa y reactiva.

#### Uso en el proyecto

En el contexto de este trabajo, useForm se emplea principalmente en el componente LoginForm (y también en RegisterForm) para:

- **Registrar los campos del formulario** usando register, conectándolos con el sistema de validación de la librería.
- **Validar los datos introducidos** utilizando reglas personalizadas o integradas.
- **Capturar errores** a través del objeto errors, que indica qué campos no cumplen las validaciones.



- **Manejar el envío del formulario** mediante `handleSubmit`, que invoca una función solo cuando el formulario es válido.
- **Generar errores manualmente**, por ejemplo, en caso de credenciales incorrectas, con `setError`.

### Estructura básica del hook

```
const {  
  register,  
  handleSubmit,  
  formState: { errors },  
  setError,  
} = useForm();
```

- `register`: Se utiliza en los inputs para enlazarlos con el sistema de validación.
- `handleSubmit`: Función que se pasa al atributo `onSubmit` del formulario.
- `errors`: Contiene los errores actuales de validación por campo.
- `setError`: Permite establecer errores personalizados desde código, por ejemplo, tras una respuesta negativa del servidor.

### Ejemplo práctico

En el componente `LoginForm`, `useForm` se emplea para validar que el nombre de usuario no esté vacío y que la contraseña cumpla una longitud mínima. Si el usuario introduce credenciales incorrectas, se usa `setError` para mostrar un mensaje de error específico bajo el campo correspondiente.



## Componente LoginForm – Formulario de inicio de sesión

El componente LoginForm forma parte de la vista de autenticación (AuthPage) y se encarga de gestionar el proceso de inicio de sesión del usuario.

### Funcionalidad general

Su objetivo principal es permitir que un usuario introduzca su nombre de usuario y contraseña, validar las credenciales y, en caso de que sean correctas, almacenar el token de autenticación para que pueda acceder a las funcionalidades protegidas de la aplicación.

### Detalles técnicos

#### 1. Gestión del formulario

Se utiliza la librería react-hook-form, que proporciona una forma eficiente de manejar formularios en React sin necesidad de controlarlos manualmente. A través de su hook principal useForm, se definen:

- register: para asociar los campos del formulario con el sistema de validación.
- handleSubmit: para manejar el envío del formulario.
- setError y errors: para gestionar y mostrar errores de validación.

#### 2. Validación personalizada

El componente importa y aplica validaciones definidas en el hook personalizado useFormValidations, que encapsula las reglas para los campos username y password, permitiendo su reutilización y separación de responsabilidades.

#### 3. Autenticación del usuario

Mediante el hook useAuth, se accede a funciones como validateUser (para comprobar que las credenciales sean correctas) y getToken (para obtener el token JWT del servidor en caso de éxito).

#### 4. Gestión del token

Si el inicio de sesión es exitoso (código de estado HTTP 200), se almacena el token en localStorage y se actualiza el estado global mediante setNewToken desde el contexto TokenContext. Esto garantiza que el token



esté disponible en otras partes de la aplicación. Posteriormente, se redirige al usuario a la página principal mediante `navigate("/")`.

#### 5. **Gestión de errores**

Si las credenciales no son válidas, se muestra un mensaje de error personalizado utilizando `setError` sobre el campo de la contraseña. Esto proporciona retroalimentación inmediata al usuario sin necesidad de recargar la página.

#### 6. **Renderizado del formulario**

El formulario se compone de dos campos de entrada (`input`) para el nombre de usuario y la contraseña. Cada uno está conectado con `react-hook-form` y muestra errores de validación cuando corresponde. Finalmente, el botón de envío activa la función `handleSubmit`, que ejecuta la lógica de autenticación.

### **Componente RegisterForm – Registro de nuevos usuarios**

El componente `RegisterForm` permite a nuevos usuarios registrarse en la aplicación proporcionando un nombre de usuario, contraseña y dirección de correo electrónico. Este componente sigue la misma estructura y lógica que el componente `LoginForm`, previamente descrito, reutilizando `react-hook-form` para la gestión del formulario y validaciones, así como funciones del hook personalizado `useAuth`.

Al igual que en `LoginForm`, los campos están validados individualmente. En este caso, se aplican validaciones adicionales al campo de correo electrónico a través del hook `useFormValidations`, lo que permite verificar su formato de manera modular y reutilizable.


La lógica de envío (`onSubmit`) llama a la función `createUser` proporcionada por `useAuth`, que se encarga de enviar la información al backend para crear el nuevo usuario. Si la operación es exitosa (`status 200`), se activa una señal visual informando del registro exitoso. Este comportamiento es similar al inicio de sesión, pero en lugar de generar y almacenar un token, el foco está en la creación inicial del usuario.

### **Componente Filetable – Visualización y gestión de archivos**

El componente `Filetable` es responsable de mostrar una tabla con los archivos almacenados por el usuario. Su estructura permite listar archivos con información relevante y ofrece funcionalidades básicas como visualización y eliminación.





Nombre	Tamaño	Fecha de Creación		
 certificadoAP_apostilla_.pdf	432536	2025-06-06		

## Funcionalidades principales:

### 1. Listado de archivos

Se recibe como propiedad (props) el array files, que contiene los objetos con información de cada archivo (nombre, tamaño, fecha de creación, etc.). Estos datos se presentan mediante una tabla con formato estilizado mediante clases de Bootstrap.

### 2. Visualización de archivos

El componente implementa la función handleOpenFile, que utiliza axios para realizar una petición HTTP GET al backend y recuperar el archivo correspondiente en formato binario (blob). Una vez recibido, se construye un objeto Blob, se genera una URL temporal (URL.createObjectURL) y se abre en una nueva pestaña del navegador. Esto permite al usuario previsualizar o descargar el archivo directamente desde la aplicación sin necesidad de guardarlo previamente.

### 3. Eliminación de archivos

La función handleDeleteFile, obtenida del hook personalizado useFiles, se encarga de ejecutar la lógica necesaria para eliminar un archivo mediante una llamada a la API. Esta función se asocia a un botón con icono de papelera, claramente identificable en la interfaz.

### 4. Cabeceras de autenticación

Para acceder a los archivos protegidos en el backend, las peticiones HTTP incluyen una cabecera Authorization con un token JWT extraído del localStorage. Este token identifica al usuario autenticado y valida su acceso a los recursos protegidos.

### 5. Componentes reutilizables

- SimpleIconButton: componente estilizado que encapsula botones con iconos de Bootstrap, utilizado tanto para la acción de eliminar como para abrir archivos.



- FileIcon: componente que muestra un icono representativo en función del tipo o nombre del archivo, ayudando a identificar visualmente cada ítem en la lista.

## BACKEND

### SPRING BOOT

Spring Initializr es una herramienta web proporcionada por el equipo de Spring que facilita la creación inicial de un proyecto con Spring Boot. Su objetivo principal es generar de forma rápida y personalizada la estructura base de un proyecto Spring Boot, evitando así la configuración manual inicial y permitiendo que el desarrollador comience a programar desde el primer momento.

---

Spring Initializr ofrece una interfaz simple donde el usuario puede:

1. **Seleccionar el lenguaje de programación:** Java, Kotlin o Groovy (siendo Java el más común).
2. **Elegir la versión de Spring Boot:** Se puede optar por la versión estable más reciente u otras versiones específicas.
3. **Configurar metadatos del proyecto:** como el nombre del grupo (Group), nombre del artefacto (Artifact), descripción, nombre del paquete base, etc.
4. **Elegir el tipo de proyecto:** normalmente un archivo .jar ejecutable, aunque también se puede generar un .war para servidores externos.
5. **Seleccionar dependencias necesarias:** Spring Boot permite seleccionar “starters” o módulos ya preconfigurados, como:
  - **Spring Web:** para construir APIs REST o aplicaciones web.
  - **Spring Data JPA:** para conectarse a bases de datos relacionales.
  - **Spring Security:** para gestionar autenticación y autorización.
  - **MySQL Driver, PostgreSQL Driver, H2 Database:** para trabajar con distintos motores de base de datos.
  - **Entre muchos otros módulos comunes.**



**Una vez seleccionadas las opciones, Spring Initializr genera un proyecto base comprimido en un archivo .zip, que puede ser importado directamente en un entorno de desarrollo como IntelliJ IDEA, Eclipse o VS Code.**

---

### **Ventajas de utilizar Spring Initializr**

- Ahorro de tiempo: Se evita tener que configurar manualmente el pom.xml (para Maven) o el build.gradle (para Gradle).
  - Estandarización: Los proyectos generados siguen una estructura coherente y recomendada por Spring, lo que facilita su mantenimiento.
  - Simplicidad: Reduce la curva de entrada para desarrolladores menos experimentados.
  - Flexibilidad: Se puede volver a generar el proyecto con nuevas dependencias si se desea, sin afectar el código existente.
- 

### **Aplicación en este proyecto**

En este proyecto, Spring Initializr se utilizó para generar la base del backend. Se seleccionaron dependencias como:

- Spring Web para exponer controladores REST.
- Spring Data JPA para gestionar el acceso a la base de datos.
- Spring Security para el sistema de autenticación basado en tokens.
- El driver correspondiente a la base de datos usada (por ejemplo, MySQL Driver).

**Esto permitió comenzar con un proyecto bien estructurado, funcional desde el primer momento, y fácilmente escalable a medida que se añadían nuevas funcionalidades.**



## Estructura del Backend – Arquitectura RESTful

El backend del proyecto ha sido desarrollado utilizando el framework Spring Boot, siguiendo una arquitectura basada en servicios REST. Esta estructura permite organizar el código en capas diferenciadas, lo que favorece la escalabilidad, el mantenimiento y la claridad del sistema.

A continuación, se describe brevemente cada una de las capas que componen el backend y su función:

### 1. Modelo (model)

La capa de modelo contiene las clases que representan las entidades del dominio, es decir, los objetos del mundo real que se desean gestionar y que se corresponden con tablas en la base de datos. Estas clases están anotadas con `@Entity` y utilizan anotaciones de JPA para indicar relaciones, claves primarias, restricciones y mapeo de columnas.

Cada entidad define los atributos persistentes (como identificadores, nombres, fechas, etc.) y las relaciones entre entidades (`@OneToMany`, `@ManyToOne`, etc.). Estas clases constituyen la base de la estructura de datos de la aplicación.

### 2. Repositorio (repository)

Los repositorios son interfaces que extienden de `JpaRepository` o `CrudRepository`, lo que permite acceder a la base de datos de forma automática sin necesidad de implementar consultas SQL directamente.

Esta capa se encarga de realizar operaciones básicas de persistencia (guardar, actualizar, buscar, eliminar) y, opcionalmente, consultas personalizadas mediante métodos nombrados (`findByEmail`, `findByNameContaining`, etc.). Es el punto de contacto entre la aplicación y la base de datos.

En el desarrollo del backend de mi aplicación web he utilizado Spring Boot junto con Spring Data JPA, y una de las herramientas fundamentales en la gestión de datos ha sido la interfaz `JpaRepository`.

Gracias a `JpaRepository`, he podido acceder y manipular la base de datos de forma sencilla, sin necesidad de escribir consultas SQL manuales. Esta interfaz me ha permitido definir repositorios específicos para cada entidad de mi sistema, como `UserRepository`, `SubjectRepository`, `FileRepository`, entre otros. Al extender



Santiago Fernandez Ehimatie

JpaRepository, automáticamente obtengo funcionalidades básicas como guardar, actualizar, eliminar o buscar entidades.

Por ejemplo, cuando necesito obtener todos los archivos subidos por un usuario, simplemente defino en el repositorio un método como `findByUserId(Long userId)` y Spring se encarga de generar la consulta correspondiente. Esto no solo reduce considerablemente la cantidad de código, sino que también mejora la claridad y mantenibilidad del proyecto.

Otro punto relevante es que en mi aplicación, cada recurso está asociado a un usuario específico, y para garantizar que un usuario solo accede a lo que le pertenece, utilizo el `SecurityContextHolder` de Spring Security, desde donde extraigo el ID del usuario autenticado (incluido en el JWT). Luego, a través del repositorio correspondiente, filtro los datos por ese ID. Por ejemplo, al listar asignaturas o archivos, hago una búsqueda como `findByUserIdAndSubjectId(...)` para asegurar que se devuelvan únicamente los datos del usuario que ha iniciado sesión.

### 3. Servicio (service)

La capa de servicios implementa la lógica de negocio del sistema. Aquí se centralizan todas las operaciones que deben llevarse a cabo al recibir una solicitud del cliente, como validaciones, transformaciones, cálculos o coordinaciones entre diferentes repositorios.

Las clases de esta capa están anotadas con `@Service` y permiten que el código del controlador permanezca limpio y enfocado exclusivamente en el tratamiento de las peticiones HTTP.

### 4. Controlador (controller)

La capa de controladores expone la funcionalidad del backend a través de endpoints REST. Estas clases están anotadas con `@RestController` y definen rutas y métodos HTTP (`@GetMapping`, `@PostMapping`, etc.) que permiten a los clientes (por ejemplo, el frontend) interactuar con el sistema.

Cada método del controlador se encarga de recibir una solicitud, delegar la lógica a la capa de servicios y devolver una respuesta adecuada, habitualmente en formato JSON.

### Relación entre las capas



Santiago Fernandez Ehimatie

La arquitectura sigue un patrón en capas donde las dependencias fluyen de manera descendente:

**[Controller] -> [Service] -> [Repository] -> [Base de datos]**

```
32 public class User implements UserDetails { // Implementa UserDetails para integrarse con Spring Security
45
46     @NotNull
47     private String password;
48
49     @NotNull
50
51     private Boolean enabled;
52
53     // Relaciones uno-a-muchos: un usuario puede tener muchos subjects, topics, files, events
54     @OneToMany(mappedBy = "user")
55     @JsonIgnore // Evita bucles infinitos al serializar a JSON
56     private List<Subject> subjects;
57
58     @OneToMany(mappedBy = "user")
59     @JsonIgnore
60     private List<Topic> topics;
61
62     @OneToMany(mappedBy = "user")
63     @JsonIgnore
64     private List<FileUpload> files;
65
66     @OneToMany(mappedBy = "user")
67     private List<Event> events;
68
69     // Métodos requeridos por Spring Security para la autenticación:
```

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

import com.example.backend.models.User;

You, yesterday | 1 author (You)
@Repository
// Repository para la entidad User, que extiende de JpaRepository para proporcionar métodos CRUD
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
    User findById(long id);
    User findByEmail(String email);
}
```



```
public class UserService implements UserDetailsService {  
    public boolean saveUser(User user){  
        User newUser = new User(user.getUsername(), user.getEmail(), encodePassword(user.getPassword()));  
        userRepository.save(newUser);  
  
        return true;  
    }  
    public UserDetails loadUserByEmail(String email){  
        UserDetails userDetails = userRepository.findByEmail(email);  
        return userDetails;  
    }  
  
    private String encodePassword(String password) {  
        return passwordEncoder.encode(password);  
    }  
  
    public UserDTO getUser(String username) {  
        Optional<User> user = userRepository.findByUsername(username);  
        if(user.isPresent()){  
            return toUserDTO(user.get());  
        }else{  
            return null;  
        }  
    }  
}
```

Este enfoque garantiza una separación clara de responsabilidades y favorece la reutilización del código, las pruebas unitarias y la escalabilidad del sistema.

## JSON Web Token (JWT)

El JSON Web Token (JWT) es un estándar abierto (RFC 7519) utilizado para el intercambio seguro de información entre dos partes, habitualmente entre el cliente y el servidor. Su principal uso en aplicaciones web modernas es la gestión de la autenticación y la autorización, especialmente en arquitecturas desacopladas, como las aplicaciones que cuentan con un frontend en React y un backend en Spring Boot.

Un JWT es un token compacto, auto-contenido y firmado digitalmente, que permite transmitir datos de manera verificable y segura. Se compone de tres secciones codificadas en Base64 y separadas por puntos. Estas secciones representan:



- Un encabezado que especifica el algoritmo de firma.
- Un contenido (payload) que incluye las "claims" o afirmaciones, es decir, los datos que se desean transmitir, como el identificador del usuario, el rol, o la fecha de expiración.
- Una firma digital generada a partir del contenido anterior y una clave secreta, que garantiza la integridad del token y permite al servidor verificar su autenticidad.

En una arquitectura típica, cuando un usuario se autentica correctamente mediante sus credenciales, el servidor emite un JWT que es devuelto al cliente. Este token se incluye en las solicitudes posteriores como prueba de autenticación, generalmente en la cabecera HTTP Authorization. De esta manera, el servidor puede identificar al usuario y autorizar el acceso a recursos protegidos sin necesidad de mantener una sesión activa en el servidor.

Entre las ventajas de JWT destacan su carácter autónomo, ya que contiene toda la información necesaria para validar la sesión; su ligereza, lo que lo hace ideal para sistemas distribuidos y microservicios; y su interoperabilidad, siendo compatible con múltiples lenguajes y plataformas.

Sin embargo, también implica ciertos desafíos en cuanto a seguridad, como el correcto almacenamiento del token en el cliente, la gestión de su expiración y la protección frente a ataques como el cross-site scripting (XSS). Es fundamental implementar medidas de protección adecuadas y emplear siempre conexiones seguras (HTTPS) en su uso.

En el proyecto desarrollado, JWT se utiliza como mecanismo central de autenticación, integrándose en el backend construido con Spring Boot. Gracias a ello, se permite una autenticación segura de los usuarios y un control de acceso eficiente a las distintas secciones de la aplicación web.





## JSON Web Token (JWT)

El JSON Web Token (JWT) es un estándar abierto (RFC 7519) utilizado para el intercambio seguro de información entre dos partes, habitualmente entre el cliente y el servidor. Su principal uso en aplicaciones web modernas es la gestión de la autenticación y la autorización, especialmente en arquitecturas desacopladas, como las aplicaciones que cuentan con un frontend en React y un backend en Spring Boot.

Un JWT es un token compacto, auto-contenido y firmado digitalmente, que permite transmitir datos de manera verificable y segura. Se compone de tres secciones codificadas en Base64 y separadas por puntos. Estas secciones representan:

- Un encabezado que especifica el algoritmo de firma.
- Un contenido (payload) que incluye las "claims" o afirmaciones, es decir, los datos que se desean transmitir, como el identificador del usuario, el rol, o la fecha de expiración.
- Una firma digital generada a partir del contenido anterior y una clave secreta, que garantiza la integridad del token y permite al servidor verificar su autenticidad.

En una arquitectura típica, cuando un usuario se autentica correctamente mediante sus credenciales, el servidor emite un JWT que es devuelto al cliente. Este token se incluye en las solicitudes posteriores como prueba de autenticación, generalmente en la cabecera HTTP Authorization. De esta manera, el servidor puede identificar al usuario y autorizar el acceso a recursos protegidos sin necesidad de mantener una sesión activa en el servidor.

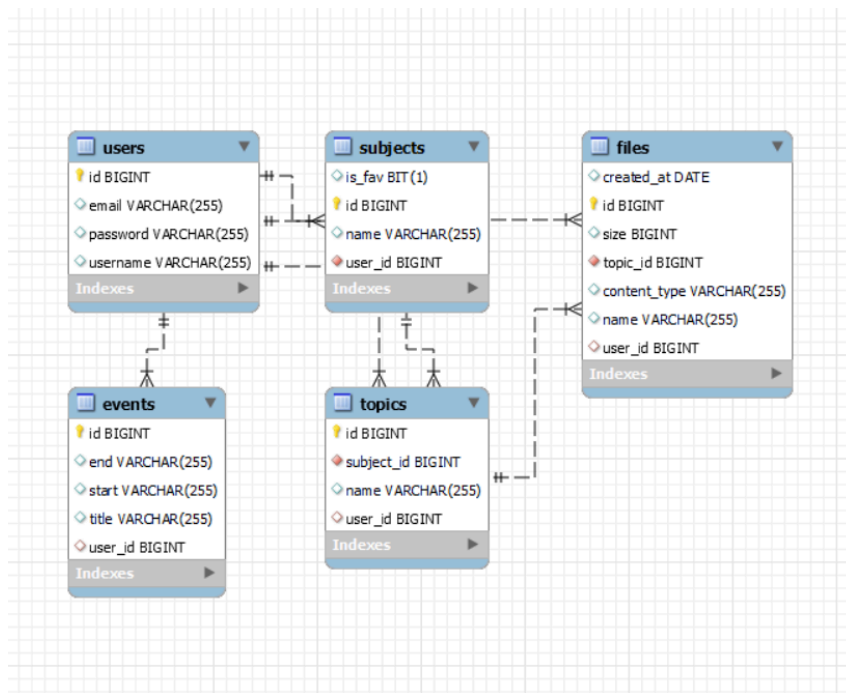
Entre las ventajas de JWT destacan su carácter autónomo, ya que contiene toda la información necesaria para validar la sesión; su ligereza, lo que lo hace ideal para sistemas distribuidos y microservicios; y su interoperabilidad, siendo compatible con múltiples lenguajes y plataformas.

Sin embargo, también implica ciertos desafíos en cuanto a seguridad, como el correcto almacenamiento del token en el cliente, la gestión de su expiración y la protección frente a ataques como el cross-site scripting (XSS). Es fundamental implementar medidas de protección adecuadas y emplear siempre conexiones seguras (HTTPS) en su uso.



En el proyecto desarrollado, JWT se utiliza como mecanismo central de autenticación, integrándose en el backend construido con Spring Boot. Gracias a ello, se permite una autenticación segura de los usuarios y un control de acceso eficiente a las distintas secciones de la aplicación web.

## Modelo ER



## Seguridad con JWT en SpringBoot

### 1. Clase utilitaria para JWT (JWT Utility)

Esta clase es fundamental para manejar todo lo relacionado con los tokens JWT. Se encarga de:

- **Generación del token:**  
A partir de los datos del usuario (principalmente el nombre de usuario y los roles o permisos), se crea un token firmado criptográficamente con una clave secreta. El algoritmo más usado es HMAC con SHA-256 (HS256), que garantiza la integridad y autenticidad del token. Además, se configura un tiempo de expiración para que el token no sea válido indefinidamente, reforzando la seguridad.



- Extracción de datos del token:  
Permite obtener la información almacenada en el token, como el nombre de usuario, roles, fecha de emisión y expiración. Esto es importante para validar el contexto del usuario durante las peticiones.
- Validación del token:  
Verifica que el token sea auténtico y válido, comprobando que la firma coincida con la clave secreta y que no haya expirado. También puede incluir comprobaciones adicionales como el issuer (emisor) y el audience (destinatario).

**Esta clase trabaja con bibliotecas estándar de JWT, como io.jsonwebtoken (JJWT), adaptándose a las mejores prácticas de seguridad.**

---

## 2. Filtro de Autenticación JWT (JWT Authentication Filter)

El filtro es un componente que se integra en la cadena de filtros de Spring Security y tiene la función de:

- Interceptar todas las peticiones HTTP entrantes:  
Antes de que se acceda a cualquier recurso protegido, el filtro verifica si la petición incluye un token JWT en la cabecera Authorization con el esquema Bearer.
- Extraer y validar el token:  
Si el token está presente, el filtro lo valida mediante la clase utilitaria mencionada anteriormente. En caso de que el token sea inválido o esté ausente, la petición puede ser rechazada o redirigida.
- Establecer el contexto de seguridad:  
Cuando el token es válido, el filtro crea un objeto de autenticación (Authentication) con los datos del usuario y lo establece en el contexto de seguridad de Spring (SecurityContextHolder). Esto permite que el resto de la aplicación reconozca al usuario como autenticado y pueda aplicar las reglas de autorización.



Este filtro debe ser registrado en la configuración de seguridad para que se ejecute antes de los filtros estándar de autenticación de Spring.

---

### 3. Configuración de Seguridad (Security Configuration)

En Spring Boot 3.x y Spring Security 6, la configuración tradicional que extendía `WebSecurityConfigurerAdapter` ha sido deprecada. En su lugar, se utiliza un bean de tipo `SecurityFilterChain` donde se define:

- Las rutas públicas y protegidas:  
Por ejemplo, endpoints para autenticación y registro pueden ser accesibles sin token, mientras que otros requieren validación JWT.
  - Integración del filtro JWT:  
El filtro de autenticación JWT se agrega explícitamente en la cadena de filtros para interceptar las peticiones.
  - Configuración sin estado (stateless):  
Debido a que JWT es un mecanismo de autenticación sin estado, se desactiva la gestión tradicional de sesiones HTTP para evitar almacenar estado en el servidor, aumentando escalabilidad y seguridad.
  - Manejo de excepciones:  
Se pueden definir manejadores personalizados para respuestas cuando un token no es válido o el usuario no está autorizado.
- 

### 4. Proveedor de Autenticación y Servicio de Usuario (UserDetailsService)

Para que Spring Security pueda cargar y validar usuarios, se implementa un servicio que:

- Carga usuario por nombre de usuario:  
Consulta la base de datos o cualquier otro repositorio para obtener los datos del usuario, incluyendo su contraseña (encriptada) y roles o permisos.



- Genera un objeto UserDetails:  
Que Spring Security utiliza internamente para manejar autenticación y autorización.
- Este servicio es utilizado tanto en la autenticación inicial (login) como en la validación del token JWT para asegurar que el usuario sigue activo y tiene los permisos necesarios.

## Mapa del sitio web

La aplicación web está organizada mediante un sistema de rutas que define las páginas existentes y los enlaces entre ellas, usando React Router. La estructura principal es la siguiente:

### Rutas públicas

- /login
  - Página de autenticación que contiene el formulario para iniciar sesión o registrarse.
  - Renderiza el layout de autenticación (AuthLayout) y el componente principal de autenticación (AuthPage).

### Rutas protegidas (requieren token válido)

Estas rutas están protegidas por un componente ProtectedRoute, que verifica si el usuario está autenticado antes de permitir el acceso:

- / (página principal)
  - Muestra la lista de asignaturas (Subjects).
- /calendar
  - Muestra un calendario con eventos (MyCalendar).
- /subject/:subjectId/topics/



- Muestra los temas relacionados con una asignatura específica (Topics).
  - /subject/:subjectId/topics/:topicId/files
    - Muestra los archivos relacionados con un tema específico (Files).
  - /storage
    - Página para la gestión del almacenamiento (Storage).
  - /userSettings
    - Página de configuración del usuario (UserSettings).
  - /favourites
    - Página que muestra las asignaturas marcadas como favoritas (Favourites).
- 

## Enlaces entre páginas

- Desde la página principal (/) el usuario puede navegar a temas específicos de asignaturas y desde ahí a los archivos relacionados.
- El calendario y la gestión de almacenamiento son accesibles desde rutas específicas en el menú principal.
- La configuración del usuario y las asignaturas favoritas también tienen su propio espacio, accesible desde el menú principal.



- El acceso a estas rutas está protegido, por lo que el usuario debe autenticarse primero mediante la página /login.

# Manual del Usuario

## 1. Introducción

Bienvenido a la aplicación Edu-vault. Esta herramienta te permite gestionar tus asignaturas, temas, archivos y eventos de forma sencilla y organizada.

---

## 2. Requisitos Previos

Para utilizar esta aplicación necesitas:

- Un navegador web actualizado (Google Chrome, Firefox, Edge, etc.).
  - Acceso a internet.
  - Una cuenta de usuario registrada.
- 

## 3. Acceso y Registro

### 3.1 Registro de Usuario

- Accede a la página de registro desde el enlace “Registrarse”.
- Completa los campos solicitados: nombre de usuario, correo electrónico y contraseña.
- Haz clic en “Registrarse”.



- Si el registro es exitoso, recibirás un mensaje de confirmación.

### 3.2 Inicio de Sesión

- Ingresa a la página de login.
- Introduce tu nombre de usuario y contraseña.
- Haz clic en “Iniciar Sesión”.
- Si las credenciales son correctas, accederás al panel principal.

---

## 4. Navegación Principal

La aplicación cuenta con las siguientes secciones principales accesibles desde el menú:

- **Asignaturas:** Lista de todas tus asignaturas. Puedes ver detalles y favoritos.
- **Temas:** Dentro de cada asignatura, accede a sus temas asociados.
- **Archivos:** Visualiza y administra archivos relacionados con los temas.
- **Calendario:** Visualiza eventos y fechas importantes.
- **Almacenamiento:** Consulta tus archivos almacenados.
- **Favoritos:** Accede rápidamente a asignaturas marcadas como favoritas.
- **Configuración:** Modifica tus datos personales y preferencias.





## 5. Funcionalidades Destacadas

### 5.1 Añadir Favoritos

- Desde la lista de asignaturas, marca una asignatura como favorita con el icono correspondiente.
- Las asignaturas favoritas aparecerán en la sección “Favoritos”.

### 5.2 Gestión de Archivos

- En la sección de archivos, puedes subir, descargar, abrir o eliminar documentos.
- Para abrir un archivo, haz clic en el icono “ojo” y se mostrará en una nueva pestaña.

### 5.3 Calendario

- Visualiza eventos programados.
- Añade o elimina eventos según sea necesario.

---

## 6. Solución de Problemas Comunes

- **No puedo iniciar sesión:** Verifica que el nombre de usuario y contraseña sean correctos. Si olvidaste tu contraseña, utiliza la opción de recuperación (si está disponible).
- **No se cargan las asignaturas:** Asegúrate de tener conexión a internet y que tu token de sesión no haya expirado. En caso de expiración, vuelve a iniciar sesión.



- **Error al abrir archivos:** Revisa que el archivo no esté corrupto y que tienes permisos adecuados.

---

## 7. Contacto y Soporte

Si tienes alguna duda o problema que no se resuelve con este manual, por favor contacta con el soporte técnico en:  
email@soporte.com

### PROPUESTAS DE AMPLIACIÓN

Dado que esta aplicación ha sido creada únicamente por mi persona, está claro que se podrían añadir numerosas funcionalidades con el fin de mejorar, ya sea el rendimiento, la lógica o la experiencia del usuario.

A continuación se presentan una serie de mejoras que podrían ser aplicadas en un futuro:

#### Envío de Correos Electrónicos de Confirmación

Actualmente, la aplicación permite que los usuarios se registren introduciendo diversos datos personales, entre ellos, una dirección de correo electrónico. No obstante, en la versión actual del sistema no se verifica la validez de dicha dirección ni se implementa un mecanismo que asegure que el correo introducido pertenece realmente al usuario.

#### Funcionalidad Propuesta

Una mejora relevante sería incorporar un sistema de envío de correos electrónicos de confirmación durante el proceso de registro. Esta funcionalidad tendría como objetivo principal verificar la identidad del usuario a través de su dirección de correo electrónico antes de permitirle el acceso completo a la plataforma.



### **El funcionamiento básico sería el siguiente:**

1. Registro inicial del usuario: El usuario completa el formulario de registro incluyendo su correo electrónico.
2. Generación de un token único de verificación: El sistema genera un identificador único y temporal asociado al usuario.
3. Envío de un correo de verificación: Se envía automáticamente un correo electrónico a la dirección proporcionada, incluyendo un enlace que contiene el token.
4. Activación de la cuenta: El usuario debe hacer clic en el enlace recibido para confirmar su dirección de correo electrónico. Hasta que lo haga, la cuenta permanecería en estado "pendiente de activación".
5. Validación del token: Al acceder al enlace, el backend verifica el token y activa la cuenta si es válido.

### **Beneficios**

- Aumenta la seguridad del sistema al asegurar que los registros están vinculados a correos electrónicos reales y controlados por los usuarios.
- Previene el uso de direcciones falsas o incorrectas, evitando registros anónimos o erróneos.
- Permite habilitar funcionalidades dependientes del correo, como la recuperación de contraseña, notificaciones automáticas, o incluso futuros sistemas de autenticación en dos pasos.
- Mejora la confianza del usuario en la plataforma al percibir que se siguen prácticas de validación comunes y seguras.



## Consideraciones Técnicas

- Para implementar esta funcionalidad sería necesario configurar un servicio de envío de correos electrónicos (por ejemplo, mediante SMTP o servicios como SendGrid, Mailgun, etc.).
- También se requiere la implementación de un sistema de gestión de tokens seguros y temporales en el backend, junto con rutas específicas para la verificación.

## Posibles Mejoras en la Página de Configuración del Usuario

La página de configuración del usuario constituye un espacio esencial para la personalización y administración de la experiencia individual dentro de la aplicación. Actualmente, esta sección incluye funcionalidades básicas, pero se han identificado diversas opciones adicionales que podrían incorporarse para mejorar significativamente la flexibilidad, usabilidad y adaptabilidad del sistema a las preferencias del usuario.

A continuación, se describen algunas de las funcionalidades que podrían añadirse:

### 1. Ajuste de Temas de la Interfaz

**Permitir al usuario modificar el tema visual de la aplicación (modo claro, oscuro u otros esquemas de color) ofrecería una experiencia más cómoda, especialmente en función del entorno de uso (día/noche) o de necesidades específicas de accesibilidad.**

- **Modo oscuro:** útil para entornos con poca luz o para reducir la fatiga visual.
- **Temas personalizados:** opción para elegir paletas de colores, fuentes o tamaños de texto.
- **Persistencia del tema:** configuración guardada en la base de datos o en localStorage para mantenerla en futuras sesiones.

### 2. Gestión de Avisos por Correo Electrónico



**Una funcionalidad altamente útil sería la posibilidad de configurar alertas o recordatorios automáticos por correo electrónico, relacionados con fechas próximas de eventos, entregas o tareas importantes.**

- Los usuarios podrían activar o desactivar este tipo de avisos.
- Configuración del intervalo de notificación (por ejemplo, 1 día antes, 1 hora antes).
- Integración con el sistema de eventos del calendario interno de la aplicación.

**Esta opción reforzaría la utilidad práctica de la plataforma como herramienta de planificación y gestión personal o académica.**

### **3. Posibilidad de Cerrar Sesión e Iniciar Sesión con Otro Usuario**

Actualmente, la mayoría de aplicaciones requieren reiniciar la sesión manualmente para cambiar de cuenta. Incluir una opción directa para cerrar sesión y volver al login permitiría agilizar este proceso, sobre todo en contextos donde un mismo dispositivo es utilizado por varias personas (por ejemplo, en entornos educativos o familiares).

- Se podría mostrar un botón visible de "Cambiar de usuario".
- Redirección automática a la página de autenticación.
- Limpieza de tokens y datos almacenados para preservar la seguridad y privacidad.

## **Gestión de Archivos**

En la versión actual del sistema, los archivos que los usuarios suben y gestionan a través de la plataforma se almacenan directamente en el mismo servidor donde se encuentra desplegado el backend. Este enfoque es funcional y sencillo de implementar en entornos de desarrollo o despliegues a pequeña escala, ya que permite una integración rápida entre el almacenamiento de archivos y la lógica de negocio del sistema.

No obstante, este tipo de gestión presenta algunas **limitaciones** importantes en términos de **escalabilidad, disponibilidad y mantenimiento**, especialmente si se



Santiago Fernandez Ehimatie

prevé un crecimiento significativo del número de usuarios o del volumen de archivos almacenados.

### Propuesta de Mejora: Almacenamiento en la Nube

Una posible evolución de esta funcionalidad sería **integrar un sistema de almacenamiento en la nube**, utilizando servicios especializados como Amazon S3, Google Cloud Storage, Azure Blob Storage, entre otros.

Este cambio permitiría:

- **Separar la lógica de negocio del almacenamiento físico de archivos**, facilitando un desarrollo más modular.
- **Mejorar la escalabilidad**, ya que las soluciones en la nube están preparadas para manejar grandes volúmenes de datos sin requerir intervenciones manuales sobre el servidor.
- **Incrementar la disponibilidad** y redundancia de los archivos, gracias a la infraestructura distribuida que ofrecen estos servicios.
- **Optimizar la seguridad**, mediante políticas de acceso basadas en roles, cifrado automático y autenticación directa con tokens o claves de acceso temporales.
- **Reducir la carga del servidor backend**, delegando la entrega de archivos al proveedor de almacenamiento.

### Consideraciones Técnicas

La implementación de esta mejora implicaría:

- Modificar la lógica de subida, descarga y eliminación de archivos en el backend para utilizar la API del proveedor de almacenamiento en la nube.
- Configurar la autenticación y gestión de permisos adecuados.
- Garantizar la compatibilidad con la arquitectura actual, sin afectar la experiencia del usuario final.



## Capacidad de Compartir Archivos entre Usuarios

Una posible mejora futura de la aplicación sería la implementación de un sistema de **compartición de archivos entre usuarios**. Esta funcionalidad permitiría que los usuarios pudieran acceder a los archivos que otros usuarios han decidido hacer públicos, fomentando la colaboración y el intercambio de recursos dentro de la plataforma.

### Funcionamiento Propuesto

El sistema se podría estructurar de la siguiente manera:

- Cada archivo almacenado por un usuario incluiría una opción para marcarlo como "público".
- Los archivos públicos serían accesibles por otros usuarios, pero siempre con permisos limitados: únicamente lectura o visualización, sin posibilidad de edición o eliminación.
- Para facilitar el acceso, se implementaría una funcionalidad de **búsqueda de usuarios**, permitiendo localizar a otros miembros de la plataforma.
- Una vez localizado un usuario, se abriría una vista reducida de su entorno de archivos, con una interfaz simplificada y restringida. Esta vista funcionaría de forma similar al espacio personal del usuario, pero con varias diferencias clave:
  - Solo se mostrarían los archivos públicos compartidos por dicho usuario.
  - El menú lateral (sidebar) podría estar ausente o reducido al mínimo necesario, para evitar confusión y reforzar la idea de que el espacio mostrado pertenece a otro usuario.
  - Todas las acciones estarían limitadas a visualización y descarga, si se habilita.



## **Beneficios**

- Fomenta la colaboración entre estudiantes o miembros de una misma organización.
- Permite compartir apuntes, recursos y documentos sin necesidad de recurrir a servicios externos.
- Refuerza el uso de la aplicación como un espacio centralizado de estudio y trabajo en equipo.

## **Consideraciones de Seguridad y Privacidad**

- Será fundamental implementar un sistema de control de acceso robusto que garantice que:
  - Solo los archivos explícitamente marcados como públicos sean accesibles por otros usuarios.
  - No se filtre ningún dato privado ni se permita ninguna acción no autorizada.
- También se podría considerar la incorporación de permisos más granulares, como compartir archivos con usuarios específicos o grupos concretos, aunque esta funcionalidad se podría abordar en fases posteriores.





## Requisitos de Hardware y Software

A continuación, se detallan los requisitos mínimos necesarios para el correcto funcionamiento del sistema tanto a nivel de software como de hardware. Estos requisitos hacen referencia tanto al entorno de desarrollo como al de ejecución.

### 1. Requisitos de Software

#### Sistema operativo compatible:

- **Cliente (Frontend):**
  - Windows 10 o superior / macOS 11.0 o superior / Linux (distribuciones basadas en Debian o RedHat).
  - Navegador moderno con soporte para ES6+ y React (Chrome, Firefox, Edge o Safari actualizados).
- **Servidor (Backend):**
  - Windows Server 2019 o superior / Ubuntu Server 20.04 o superior / macOS 11.0 o superior.
  - Java 17 o superior (para backend en Spring Boot).
  - Node.js 18+ y npm/yarn (para la parte frontend con React).
  - Docker (opcional para despliegue en contenedor).

#### Dependencias principales:

- **Frontend:**
  - React 18
  - React Router DOM
  - React Hook Form



- @tanstack/react-query
- Bootstrap 5+
- Axios
- **Backend:**
  - Spring Boot 3+
  - Base de datos MySQL/MariaDB 10.5+
  - JPA/Hibernate
  - Spring Security (para autenticación con JWT)

#### **Entorno de desarrollo recomendado:**

- Visual Studio Code (versión 1.70 o superior)
- IntelliJ IDEA / Eclipse (para el backend)
- Postman (para pruebas de API REST)
- Git (para control de versiones)
- MySQL Workbench (para administración de la base de datos)

---

## **2. Requisitos de Hardware**

### **Cliente (usuario final):**

- **Procesador:** Intel Core i3 de 6ª generación o AMD equivalente.
- **Memoria RAM:** 4 GB mínimo (8 GB recomendados para una experiencia fluida).



Santiago Fernandez Ehimatie

- **Espacio en disco:** 500 MB libres para uso en navegador y caché del frontend.

- **Conectividad:** Acceso a Internet para conexión al servidor y carga de recursos.

#### **Servidor (despliegue backend):**

- **Procesador:** Intel Core i5 de 8ª generación o superior / AMD Ryzen 5 o superior.
- **Memoria RAM:** 8 GB mínimo (recomendado 16 GB en entornos con múltiples usuarios simultáneos).
- **Espacio en disco:** 5 GB libres para el sistema, base de datos y archivos subidos por los usuarios.
- **Red:** Conectividad estable, al menos 10 Mbps de subida para manejar peticiones de usuarios.



## Bibliografía

Vite. (2023). *Vite Documentation*. Recuperado de <https://vitejs.dev/>

React. (2023). *React Documentation*. Recuperado de <https://reactjs.org/>

Auth0. (2022). *The JWT Handbook*. Recuperado de <https://auth0.com/resources/ebooks/jwt-handbook>

Bootstrap. (2023). *Bootstrap Documentation*. Recuperado de <https://getbootstrap.com/>

MySQL. (2023). *MySQL 8.0 Reference Manual*. Recuperado de <https://dev.mysql.com/doc/>

MySQL. (2023). *MySQL Workbench Manual - EER Diagrams*. Recuperado de <https://dev.mysql.com/doc/workbench/en/wb-eer-diagrams.html>

Pivotal Software. (2023). *Spring Boot Documentation*. Recuperado de <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>

React. (2023). *React Documentation*. Recuperado de <https://reactjs.org/>

React Hook Form. (2023). *React Hook Form Documentation*. Recuperado de <https://react-hook-form.com/>

TanStack. (2023). *React Query Documentation*. Recuperado de <https://tanstack.com/query/latest>

Vite. (2023). *Vite Documentation*. Recuperado de <https://vitejs.dev/>