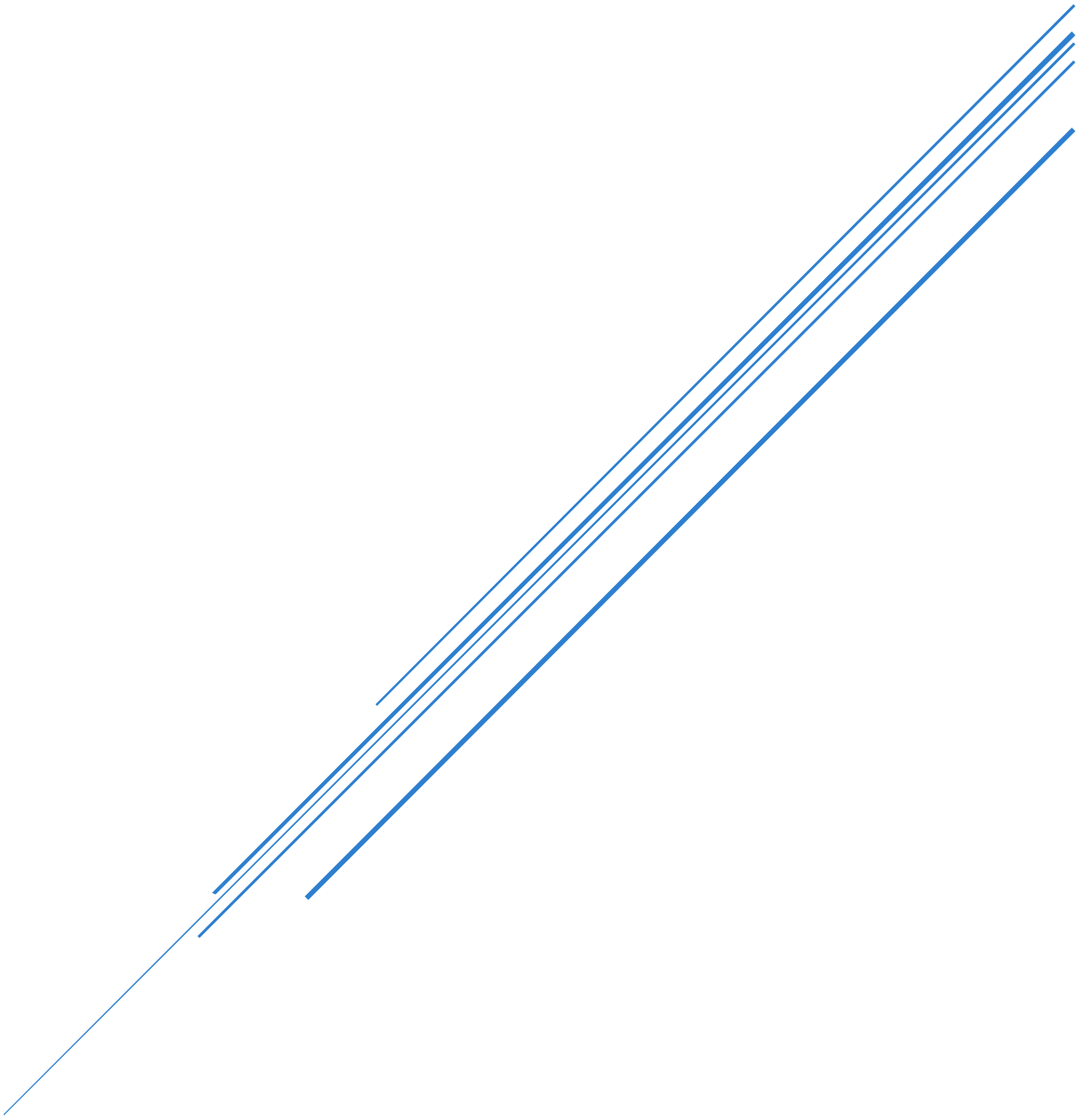
SISTEMA DE GESTIÓN DE USUARIOS, ROLES Y SEGURIDAD JWT

Cristian Alfredo Alas Castellanos

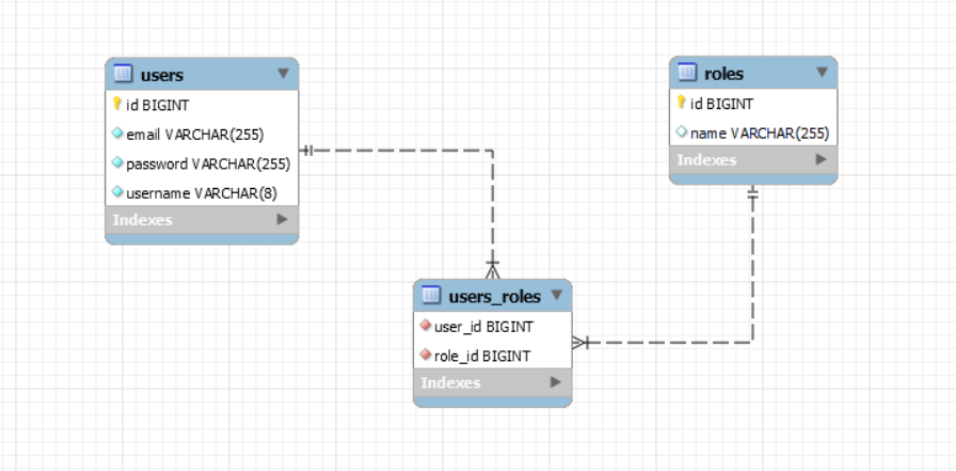


Índice

1. Base de datos

MySQL Workbench

1.1.Diagrama de base de datos



1.2.Estructura

Tabla roles:

Esta tabla almacena roles de usuario.

Campos:

- id: Identificador único auto incremental.
- name: Nombre del rol.
- PRIMARY KEY en id.
- UNIQUE KEY en name.
- Se insertan dos roles: ROLE_ADMIN con id 1 y ROLE_USER con id 2.

Tabla users:

Esta tabla almacena información de usuarios.

Campos:

- id: Identificador único auto incremental.
- email: Correo electrónico del usuario.
- password: Contraseña del usuario (encriptada).
- username: Nombre de usuario único.
- PRIMARY KEY en id.
- UNIQUE KEY en email y username.

Tabla users_roles:

Esta tabla relaciona usuarios con roles.

Campos:

- user_id: Clave foránea que referencia id en la tabla users.
- role_id: Clave foránea que referencia id en la tabla roles.
- UNIQUE KEY en (user_id, role_id).
- FOREIGN KEY que conecta user_id con id en users.
- FOREIGN KEY que conecta role_id con id en roles.

2. Backend

2.1.Herramientas utilizadas:

2.1.1. IDE:

Intellij idea

2.1.2. Versión de java

- 17

2.1.3. Versión de SpringBoot

- 3.3.0

2.1.4. Dependencias

- Spring Data JPA
- MySQL Driver
- Spring Boot DevTools
- Spring Web
- Lombok
- Actuator
- Validation
- Swagger

Pruebas Unitarias

- Junit5 (jupiter)
- Mockito

Spring Security y JWT

- spring-boot-starter-security (version 5.5)

io.jsonwebtoken

- jjwt-api (versión 0.11.5)
- jjwt-impl (versión 0.11.5)
- jjwt-jackson (versión 0.11.5)

2.1.5. Patrón de diseño

Patrón de Diseño Data Transfer Object (DTO)

El patrón DTO se utiliza para transferir datos entre distintas capas de la aplicación, reduciendo el número de llamadas a métodos remotos y simplificando la transferencia de datos complejos. En el código, se observa claramente en la clase UserDto y el uso de esta en los servicios y controladores.

Patrón Builder

El patrón Builder se utiliza para crear objetos complejos paso a paso. Se encuentra implementado en la clase DtoMapperUser, que permite construir un UserDto a partir de un UserEntity de manera fluida y clara.

Patrón de Diseño Repository

El patrón Repository se encarga de encapsular la lógica de acceso a datos y de proveer una interfaz hacia las operaciones de la base de datos. Se observa en las interfaces UserRepository y RoleRepository.

2.1.6. Principio solid

Single Responsibility Principle (SRP)

Cada clase tiene una única responsabilidad. Por ejemplo, UserEntity representa la entidad del usuario, UserService maneja la lógica de negocio y UserController gestiona las peticiones HTTP.

Open/Closed Principle (OCP)

Las clases están abiertas para la extensión, pero cerradas para la modificación. Por ejemplo, UserServiceImpl puede ser extendida o decorada sin modificar su implementación existente.

Liskov Substitution Principle (LSP)

Las instancias de una subclase (UserServiceImpl) pueden reemplazar instancias de la superclase (UserService) sin alterar el comportamiento del programa.

Interface Segregation Principle (ISP)

Las interfaces están bien definidas y no obligan a implementar métodos no necesarios. Por ejemplo, UserService define métodos específicos para las operaciones de usuarios.

Dependency Inversion Principle (DIP)

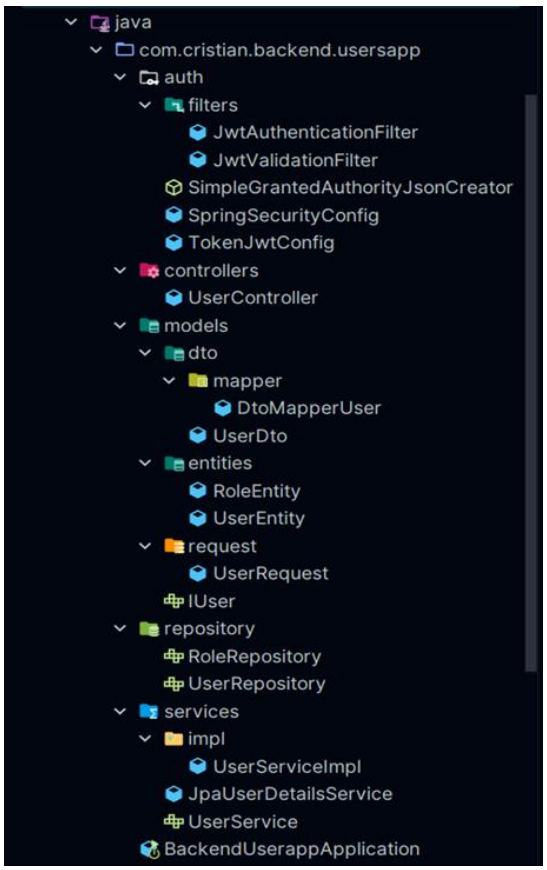
Las dependencias se inyectan en lugar de ser creadas dentro de la clase. Esto es evidente en el uso de @Autowired en UserServiceImpl y UserController.

Aspect-Oriented Programming (AOP)

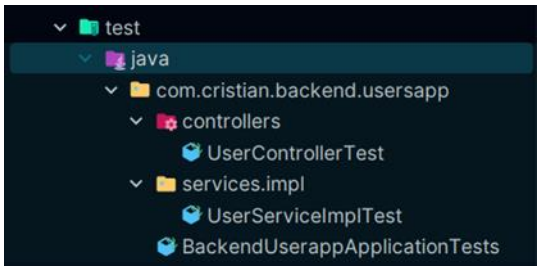
En el controlador UserController, se hace uso de aspectos para la validación:

Aquí se valida la entrada del usuario antes de proceder con la lógica de negocio, separando así las preocupaciones transversales del código principal.

2.1.7. Estructura del proyecto:



2.1.8. Estructura Test:



2.1.9. Explicación de la Api Rest

API REST con Gestión de Usuarios, Roles y Seguridad JWT

Esta API REST, construida con Spring Boot, JPA y seguridad JWT, ofrece un sistema de gestión de usuarios y roles, validando y garantizando un acceso seguro y controlado a los recursos. A continuación, se detalla el funcionamiento de cada componente.

Clases de Entidades

RoleEntity

- Representa los roles de usuario en la base de datos.
- Atributos: id (ID del rol), name (nombre único del rol).

UserEntity

Representa a los usuarios en la base de datos.

Atributos: id, username (nombre de usuario único), password (contraseña), email (correo electrónico único), roles (lista de roles asociados al usuario), admin (indica si el usuario tiene privilegios de administrador, es un atributo transitorio).

Clases de DTO y Mapeadores

UserDto

- Data Transfer Object para los usuarios.
- Atributos: id, username, email, admin.

DtoMapperUser

- Mapea una entidad UserEntity a un DTO UserDto.
- Implementa el patrón Builder para construir el DTO.

Clases de Repositorio

UserRepository

- Extiende JpaRepository para operaciones CRUD sobre UserEntity.
- Métodos adicionales: findByUsername (busca un usuario por su nombre de usuario), findAll(Pageable pageable) (paginación de usuarios).

RoleRepository

- Extiende JpaRepository para operaciones CRUD sobre RoleEntity.
- Método adicional: findByName (busca un rol por su nombre).

Servicios

UserService

- Interfaz que define los métodos del servicio de usuario: findAll, findAll(Pageable pageable), findById, save, update, deleteById.

UserServiceImpl

- Implementación de UserService.
- Utiliza UserRepository, RoleRepository y PasswordEncoder.

Métodos

- findAll: retorna todos los usuarios.
- findAll(Pageable pageable): retorna una página de usuarios.
- findById: busca un usuario por su ID.
- save: guarda un nuevo usuario, codificando la contraseña y asignando roles.
- update: actualiza un usuario existente.
- deleteById: elimina un usuario por su ID.
- getRoles: asigna roles a un usuario según sus privilegios de administrador.

Controlador

UserController

- Define los endpoints para la gestión de usuarios.
- GET /users: retorna todos los usuarios.
- GET /users/page/{page}: retorna una página de usuarios.
- GET /users/{id}: busca un usuario por su ID.
- POST /users: crea un nuevo usuario.
- PUT /users/{id}: actualiza un usuario existente.
- DELETE /users/{id}: elimina un usuario por su ID.
- POST /login: autenticación de los usuarios.

Métodos auxiliares:

- Validation: maneja los errores de validación y construye la respuesta apropiada.

Seguridad

Proceso de Login

- Punto de Entrada para el Login:
- La aplicación tiene un endpoint de autenticación donde los usuarios envían sus credenciales (nombre de usuario y contraseña).
- Maneja las solicitudes de login. Valida las credenciales del usuario y, si son correctas. genera un token JWT.

SpringSecurityConfig

- Configura la seguridad de la aplicación usando JWT.
- Define los permisos para cada endpoint.
- Implementa CORS.
- Define los filtros de autenticación y validación de JWT.

JwtAuthenticationFilter

- Filtro de autenticación que genera un token JWT al autenticarse correctamente.

Métodos

- attemptAuthentication: intenta autenticar al usuario.
- successfulAuthentication: genera y retorna un token JWT al autenticarse correctamente.
- unsuccessfulAuthentication: maneja los errores de autenticación

JwtValidationFilter

- Filtro que valida el token JWT en cada petición.
- Método doFilterInternal: extrae y valida el token JWT de las cabeceras de la petición.
- Otros Componentes.

JpaUserDetailsService

- Implementa UserDetailsService para cargar los detalles del usuario desde la base de datos por nombre de usuario.
- Convierte UserEntity en un objeto User de Spring Security con sus roles correspondientes.

TokenJwtConfig

- Configura los parámetros del token JWT (clave secreta, prefijo, cabecera de autorización).

SimpleGrantedAuthorityJsonCreator

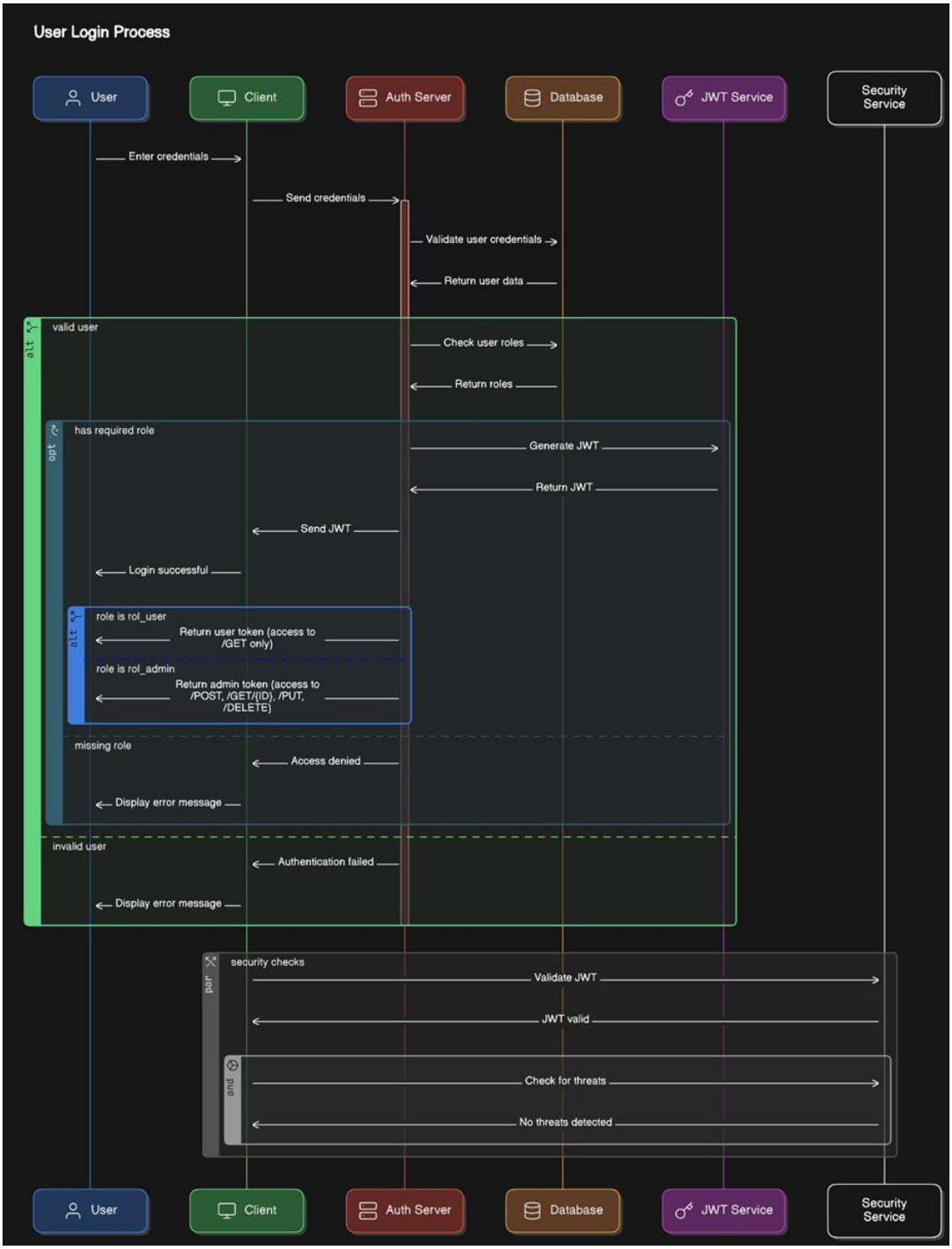
- facilita la deserialización de objetos JSON en instancias de SimpleGrantedAuthority mediante las anotaciones @JsonCreator y @JsonProperty.

Pruebas Unitarias

- Las pruebas unitarias para esta API se realizaron con JUnit5 y Mockito.
- JUnit5 para definir y ejecutar las pruebas.

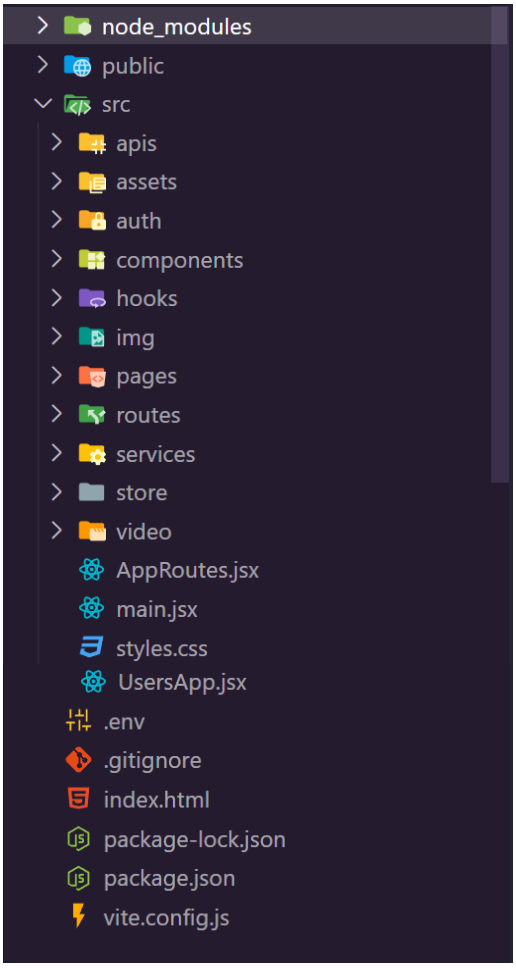
- Mockito para simular las dependencias (como repositorios y servicios) y verificar el comportamiento de los métodos de servicio y controladores.

3. Diagrama de caso de uso



4. Front End

4.1.Estructura del proyecto



4.2.Herramientas utilizadas

- React
- Axios

4.3.Custom Components:

- useAuth
- useUsers

- UserForm
- UserModalForm
- UserRow
- UsersList
- Paginator

4.4.Descripción de la Aplicación React

La aplicación React implementa un sistema completo de autenticación de usuarios y gestión de usuarios para administradores. A continuación, se detallan las funcionalidades principales de la aplicación.

4.4.1. Autenticación de Usuarios

- Registro de usuarios: Los usuarios pueden registrarse y crear una cuenta.
- Inicio de sesión: Los usuarios pueden iniciar sesión con su nombre de usuario y contraseña.
- Mantener sesión: La aplicación mantiene la sesión del usuario mientras el navegador esté abierto.
- Cierre de sesión: Los usuarios pueden cerrar sesión para finalizar su sesión.

Gestión de Usuarios (para Administradores)

- Visualización de usuarios: Los administradores pueden ver una lista de todos los usuarios registrados.
- Agregar usuarios: Los administradores pueden agregar nuevos usuarios a la aplicación.

- Editar usuarios: Los administradores pueden editar la información de los usuarios existentes.
- Eliminar usuarios: Los administradores pueden eliminar usuarios de la aplicación.

Funcionalidades Adicionales

- Nombre de usuario en la navegación: La aplicación muestra el nombre de usuario del usuario actualmente conectado en la barra de navegación.
- Autenticación con token de portador: La aplicación utiliza un token de portador para autenticar las solicitudes al servidor API.
- Almacenamiento de sesión: La aplicación almacena el token de portador y la información del usuario en el almacenamiento de sesión.

Explicación Detallada

Configuración del Inicio de Sesión de Usuario

- Uso de axios: Se utiliza axios para realizar solicitudes API.
- Configuración de usersApi: Este se configura para apuntar al endpoint /users de una API, con la URL base definida en import.meta.env.VITE_API_BASE_URL.
- Interceptor de solicitudes: Se agrega un interceptor a usersApi para incluir un encabezado Authorization con un token de portador en cada solicitud. Este token se recupera del almacenamiento de sesión.

Funcionalidad de Inicio de Sesión

- useAuth: Proporciona métodos para iniciar y cerrar sesión.

Función handlerLogin:

- Toma el nombre de usuario y la contraseña como argumentos.
- Envía una solicitud POST al endpoint /login de la API con las credenciales.
- Si el inicio de sesión es exitoso, los datos de respuesta (incluido el token) se almacenan en el almacenamiento de sesión.
- El estado de la aplicación se actualiza con la información del usuario recuperada del token.

Funcionalidad de Cierre de Sesión

- Función handlerLogout: Elimina el token y la información del usuario del almacenamiento de sesión y del estado de la aplicación.

Componentes de la Interfaz de Usuario

- LoginPage: Representa un formulario de inicio de sesión donde los usuarios pueden ingresar sus credenciales. Utiliza el gancho useAuth para acceder a handlerLogin para enviar el formulario de inicio de sesión.
- Navbar: Muestra el nombre de usuario si está conectado y proporciona un botón de cierre de sesión. Utiliza el gancho useAuth para acceder al estado login y la función handlerLogout.

Gestión de Usuarios (para Administradores)

- Renderización condicional: La aplicación renderiza funcionalidades basadas en el rol del usuario (administrador o no).

Componentes para administradores:

- UsersList: Muestra una lista de usuarios.
- UserForm: Proporciona un formulario para agregar o editar usuarios.
- useUsers: Proporciona métodos para obtener usuarios, agregar, eliminar y actualizar usuarios. Interactúa con la API (findAllPages, save, update, remove) para realizar estas acciones.

Configuración de la API

- Uso de axios: Para realizar solicitudes HTTP en aplicaciones JavaScript.
- URL base de la API: Apunta a un servidor separado que aloja los endpoints de autenticación y gestión de usuarios.
- Interceptor: Intercepta todas las solicitudes realizadas a través de usersApi y agrega un encabezado Authorization con un token de portador recuperado del almacenamiento de sesión.

Autenticación de Usuario

- LoginPage: Presenta un formulario de inicio de sesión donde el usuario ingresa su nombre de usuario y contraseña.

HandlerLogin

- Envía una solicitud POST al endpoint /login de la API con las credenciales.
- Si el inicio de sesión es exitoso, la API devuelve un token de portador.
- El token se almacena en el almacenamiento de sesión y el estado de la aplicación se actualiza con la información del usuario.

Gestión del Estado de la Aplicación

- Gancho useAuth: Administra el estado de autenticación del usuario.

Propiedades

- login: Indica si el usuario está conectado.
- user: Contiene información del usuario actual.
- token: Almacena el token de portador.

Funciones

- handlerLogin y handlerLogout actualizan el estado de la aplicación en consecuencia.
- Acceso al estado: Los componentes de la interfaz de usuario pueden acceder al estado utilizando el gancho useAuth.