Universidad de San Carlos de Guatemala
Centro Universitario de Occidente División de Ciencias de la Ingeniería
Manejo e implementación de Archivos
Ing. Christian Alberto López Quiroa
Sección: A

Manual Técnico

Carlos Raúl Alberto López Peláez 202031871

Quetzaltenango,31 de Octubre del 2024

Herramientas Utilizadas

Spring Framework

Spring es un marco de trabajo para el desarrollo de aplicaciones Java que proporciona una arquitectura robusta y escalable. Se basa en la inyección de dependencias, lo que facilita la creación de aplicaciones modulares. Las características clave de Spring incluyen:

Inversión de Control (IoC): Permite que el marco gestione la creación y el ciclo de vida de los objetos.

Aspect-Oriented Programming (AOP): Facilita la separación de preocupaciones, permitiendo manejar tareas transversales como la gestión de transacciones y la seguridad.

Spring Boot

Spring Boot es una extensión del Spring Framework que simplifica la configuración y el desarrollo de aplicaciones Spring. Sus características incluyen:

Configuración Automática: Reduce la necesidad de configuraciones manuales al detectar automáticamente las dependencias y configurarlas.

Aplicaciones Autónomas: Permite ejecutar aplicaciones como aplicaciones Java autónomas, eliminando la necesidad de un servidor de aplicaciones separado.

Empezar Rápido: Facilita la creación de microservicios y aplicaciones RESTful con un enfoque en el desarrollo ágil.

Vue.js

Vue.js es un marco de trabajo progresivo para construir interfaces de usuario. Se enfoca en la capa de vista del modelo MVC y ofrece características como:

Reactividad: Los cambios en el estado de la aplicación se reflejan automáticamente en la interfaz de usuario.

Componentes: Permite crear componentes reutilizables, lo que promueve la modularidad y la mantenibilidad del código.

Integración Sencilla: Se puede integrar fácilmente con otras bibliotecas o proyectos existentes.

MongoDB

MongoDB es una base de datos NoSQL orientada a documentos que almacena datos en un formato flexible y escalable. Sus características incluyen:

Modelo de Documentos: Los datos se almacenan en documentos BSON (Binary JSON), lo que permite un esquema dinámico.

Escalabilidad Horizontal: Facilita la distribución de datos a través de múltiples servidores para manejar grandes volúmenes de información.

Consultas Potentes: Proporciona un lenguaje de consulta robusto que permite realizar búsquedas complejas.

Flujo de Trabajo de la Aplicación

La integración de estas tecnologías sigue un flujo de trabajo típico en aplicaciones modernas:

Backend con Spring Boot y MongoDB:

Se configura un proyecto Spring Boot utilizando dependencias como Spring Data MongoDB para interactuar con la base de datos.

Se crean controladores REST que manejan las solicitudes HTTP y responden con datos en formato JSON.

Frontend con Vue.js:

Se desarrolla una interfaz de usuario interactiva utilizando Vue.js, que se comunica con el backend a través de solicitudes AJAX (generalmente usando Axios).

Los componentes de Vue.js se utilizan para manejar la visualización de datos y la interacción del usuario.

Comunicación entre el Frontend y el Backend:

La aplicación Vue.js envía solicitudes al backend de Spring Boot, que procesa los datos y realiza operaciones en MongoDB.

Los resultados se devuelven a la interfaz de usuario para su visualización.

Controladores

AdminController

El controlador AdminController se encarga de gestionar las operaciones del administrador relacionadas con archivos y usuarios dentro de la API. Proporciona varios endpoints para obtener directorios, archivos y archivos compartidos, así como para crear nuevos usuarios. Utiliza el patrón @RestController para manejar las solicitudes HTTP y devuelve respuestas en formato JSON.

1. Importaciones y Anotaciones

- Anotaciones principales:
- @RestController: Indica que esta clase es un controlador que responderá a las solicitudes HTTP y devolverá datos JSON.
- @RequestMapping("/admin"): Define la ruta base para todos los endpoints dentro de este controlador. Todas las rutas estarán precedidas por /admin.
- Inyección de Dependencias:
- @Autowired: Se utiliza para inyectar las dependencias AdminService, FileRepository y UserDetailServiceImpl, que proporcionan la lógica de negocio para este controlador.

2. Dependencias Inyectadas

- AdminService: Servicio que contiene la lógica de negocio para operaciones relacionadas con directorios, archivos y usuarios.
- FileRepository: Repositorio que maneja las operaciones CRUD (Create, Read, Update, Delete) para la entidad File en la base de datos.
- UserDetailServiceImpl: Servicio que gestiona la lógica relacionada con los detalles de los usuarios (no se detalla su uso específico en este controlador).

3. Endpoints del Controlador

1. getDirectorys (GET /admin/getDirectorys)

   Descripción:

   Recupera una lista de directorios para un ID de usuario específico.

   Parámetro:@RequestParam String id: El ID del usuario para obtener los directorios.

2. Lógica: Llama al método getDirectorys de AdminService, que retorna una lista de DirectoryDTOResponse representando los directorios.
3. Retorno:Una lista de objetos DirectoryDTOResponse en formato JSON.

3.2. getFiles (GET /admin/getFiles)

- Descripción:Recupera una lista de archivos que pertenecen a un directorio específico, filtrando aquellos que están marcados como eliminados.
- Parámetro:@RequestParam String id: El ID del directorio para el cual se desean obtener los archivos.
- Lógica:Valida si el id no es nulo, y busca los archivos correspondientes utilizando el método findAllByDirectoryIdAndIsDeletedTrue de FileRepository, que busca archivos en un directorio específico y que están marcados como eliminados.

  Llama a adminService.getFilesWithContent() para transformar la lista de entidades File a FileDTOResponse, incluyendo el contenido de cada archivo.
- Manejo de Excepciones:Si ocurre un IOException, retorna un error interno del servidor (500).
- Retorno:ResponseEntity<List<FileDTOResponse>> con la lista de archivos en formato JSON si la operación es exitosa.

3.3. getSharedFiles (GET /admin/getSharedFiles)

- Descripción:Obtiene una lista de archivos compartidos para un usuario específico.
- Parámetro:@RequestParam String id: El ID del usuario para obtener los archivos compartidos.
- Lógica:Llama al método adminService.getSharedFilesWithContent() para obtener una lista de FileDTOResponse que representan archivos compartidos con el usuario.
- Manejo de Excepciones:Si ocurre un IOException, retorna un error interno del servidor (500).
- Retorno:ResponseEntity<List<FileDTOResponse>> con la lista de archivos compartidos en formato JSON si la operación es exitosa.

3.4. createUser (POST /admin/create/User)

- Descripción:Crea un nuevo usuario utilizando la información proporcionada en la solicitud.
- Parámetro:@RequestBody CreateUserRequest createUserRequest: Un objeto que contiene los detalles necesarios para crear un usuario (nombre, correo, contraseña, etc.).
- Lógica:Llama al método adminService.createUser() que gestiona la creación del usuario y devuelve una respuesta de autenticación (AuthResponse).

- Retorno:ResponseEntity<AuthResponse> con el resultado de la creación del usuario (puede incluir un mensaje de éxito y un token de autenticación).

AuthController

1. **Anotaciones y Dependencias**

- @RestController: Indica que esta clase es un controlador de Spring que devuelve datos directamente en el cuerpo de la respuesta, en lugar de renderizar vistas.
- @RequestMapping("/auth"): Define que todas las rutas de los métodos en esta clase tendrán el prefijo /auth.
- @Autowired: Inyecta instancias de UserDetailServiceImpl y UserRepository, que son necesarias para realizar la autenticación y acceder a los datos de los usuarios.

2. **Endpoints del Controlador**

Método login:

- @PostMapping("/login"): Define un endpoint POST en la ruta /auth/login para manejar las solicitudes de inicio de sesión.
- Parámetro @RequestBody @Valid AuthLoginRequest userRequest: Recibe el cuerpo de la solicitud con los datos de login, que incluye el nombre de usuario y la contraseña. La anotación @Valid asegura que la validación de los datos se realice antes de procesarlos.
- this.userDetailService.loginUser(userRequest): Llama al método que autentica al usuario y devuelve un AuthResponse con la información del token JWT.
- userRepository.findUserEntityByUsername(): Busca al usuario en la base de datos por su nombre de usuario.
- orElseThrow(): Lanza una excepción UsernameNotFoundException si el usuario no existe.
- responseBody: Un mapa para estructurar la respuesta, incluyendo el rol del usuario, el mensaje de autenticación, el token JWT y el estado.
- ResponseCookie.from(...): Crea una cookie HTTP que contiene el token JWT. La cookie es httpOnly y secure, lo que significa que solo es accesible desde el servidor y requiere HTTPS para ser transmitida.

- ResponseEntity.ok().header(HttpHeaders.SET_COOKIE, jwtCookie.toString()).body(responseBody): Retorna una respuesta 200 OK con la cookie configurada y el cuerpo de la respuesta.

Manejo de excepciones:

- UsernameNotFoundException: Si el usuario no se encuentra, retorna un 404 (Not Found) con un mensaje de error.
- BadCredentialsException: Si las credenciales son incorrectas, retorna un 401 (Unauthorized).
- Exception: Para cualquier otro error inesperado, retorna un 500 (Internal Server Error).

Método logout:

- @PostMapping(value = "logout"): Define un endpoint POST en la ruta /auth/logout para manejar las solicitudes de cierre de sesión.
- logout(response): Llama al método logout en UserDetailServiceImpl, que se encarga de gestionar la lógica para cerrar la sesión del usuario.
- ResponseEntity<String>: Retorna la respuesta correspondiente, generalmente una confirmación del cierre de sesión.

DirectoryController

El DirectoryController es un componente de la API de Grafiles que maneja las solicitudes HTTP relacionadas con la creación, actualización, obtención, eliminación, copia y movimiento de directorios. Utiliza JWT (JSON Web Tokens) para la autenticación de usuarios y proporciona una interfaz RESTful para interactuar con el servicio de directorios.

1. Anotaciones y Dependencias

- @RestController: Indica que esta clase es un controlador de Spring que maneja solicitudes HTTP y devuelve respuestas en formato JSON.
- @RequestMapping("/directory"): Define la ruta base para todas las solicitudes en este controlador.

- **Dependencias:**
- DirectoryService: Servicio que contiene la lógica de negocio relacionada con los directorios.
- JwtUtils: Clase utilitaria para manejar la decodificación de JWT y la extracción de información del mismo.
- HttpServletRequest y HttpServletResponse: Objetos que representan la solicitud y la respuesta HTTP, respectivamente.

2. Métodos del Controlador

2.1. Crear un Directorio

Método: directoryCreated

- Ruta: POST /directory/created
- Descripción: Crea un nuevo directorio basado en la solicitud del cliente.
- Parámetros: DirectoryDTOResquest directoryDTOResquest: Contiene la información del directorio a crear.
- Flujo:Extrae el JWT de las cookies.
- Decodifica el JWT para obtener el ID del usuario.
- Llama al método createDirectory del servicio de directorios.

2.2. Actualizar un Directorio

Método: directoryUpdate

- Ruta: PUT /directory/updated
- Descripción: Actualiza un directorio existente.
- Parámetros:UpdateDirectoryDTORequest updateDirectoryDTORequest: Contiene la información actualizada del directorio.
- Flujo:Similar al método anterior, se extrae el JWT y el ID del usuario, luego se llama a updateDirectory.

2.3. Obtener Directorios

Método: getDirectorys

- Ruta: GET /directory/gets
- Descripción: Recupera una lista de directorios para un usuario específico.
- Parámetros:String id: ID del directorio que se desea obtener.
- Flujo:Extrae el JWT y el ID del usuario, luego llama a getDirectorys.

2.4. Obtener un Directorio Específico

Método: getDirectory

- Ruta: POST /directory/get
- Descripción: Obtiene la información de un directorio específico.
- Parámetros:DirectoryDTOResquest directoryDTOResquest: Contiene la información necesaria para recuperar el directorio.
- Flujo:Extrae el JWT y el ID del usuario, luego llama a getDirectory.

2.5. Eliminar un Directorio

Método: deleteDirectory

- Ruta: DELETE /directory/deleted
- Descripción: Elimina un directorio existente.
- Parámetros:String id: ID del directorio a eliminar.
- Flujo:Extrae el JWT y el ID del usuario, luego llama a deleteDirectory.

2.6. Copiar un Directorio

Método: copyDirectory

- Ruta: POST /directory/copy
- Descripción: Copia un directorio a otro lugar.
- Parámetros:DirectoryCopiDTORequest directoryCopiDTORequest: Contiene los IDs del directorio a copiar y su nuevo padre.
- Flujo:Extrae el JWT y el ID del usuario, llama a copyDirectory en el servicio y maneja posibles excepciones.

2.7. Mover un Directorio

Método: moveDirectory

- Ruta: POST /directory/move
- Descripción: Mueve un directorio a una nueva ubicación.
- Parámetros:DirectoryCopiDTORequest directoryCopiDTORequest: Contiene los IDs del directorio a mover y su nuevo padre.
- Flujo:Extrae el JWT y el ID del usuario, llama a moveDirectory y maneja excepciones.

3. Métodos Privados

3.1. Extraer JWT de la Cookie

Método: extractJwtFromCookie

- Descripción: Extrae el valor del token JWT de las cookies de la solicitud HTTP.
- Flujo:Itera sobre las cookies y devuelve el valor de la cookie llamada jwtToken.

3.2. Extraer ID del Usuario del Token

Método: extractUserIDFromToken

- Descripción: Decodifica el JWT y extrae el ID del usuario.
- Flujo:Utiliza JwtUtils para decodificar el token y retorna el valor del reclamo id_user.

FileController

El controlador FileController es responsable de manejar las operaciones relacionadas con los archivos en la aplicación de gestión de archivos. Este controlador utiliza JWT (JSON Web Tokens) para la autenticación y autorización de los usuarios, asegurando que solo los usuarios válidos puedan acceder y manipular sus archivos. Las operaciones incluyen obtener, crear, subir, descargar, eliminar, copiar, mover y compartir archivos.

1. Anotaciones y Dependencias

- JwtUtils: Para la gestión de la autenticación basada en JWT.
- FileService: Servicio para realizar operaciones de negocio relacionadas con archivos.
- GridFsTemplate: Para almacenar y recuperar archivos en GridFS (un sistema de almacenamiento de archivos de MongoDB).
- HttpServletRequest y HttpServletResponse: Para interactuar con las solicitudes y respuestas HTTP.2.

2. Endpoints del Controlador

1. **Obtener Archivos**
    - GET /file/gets
    - Descripción: Obtiene todos los archivos de un directorio específico para un usuario.
    - Parámetros:Id (ObjectId): ID del directorio.

- Respuesta: Devuelve una lista de archivos asociados al directorio.

2.Obtener Archivos Compartidos

- GET /file/get/shared
- Descripción: Recupera todos los archivos que han sido compartidos con el usuario autenticado.
- Respuesta: Devuelve una lista de archivos compartidos.

3. Subir Archivo

- PUT /file/upload
- Descripción: Sube un archivo a un directorio específico.
- Parámetros:file (MultipartFile): Archivo a subir.
- directoryId (String): ID del directorio donde se subirá el archivo.
- id (String): ID del archivo (si se está actualizando).
- Respuesta: Mensaje de éxito con el ID del archivo.

4. Crear Archivo

- POST /file/created
- Descripción: Crea y sube un nuevo archivo en un directorio específico.
- Parámetros:file (MultipartFile): Archivo a subir.
- directoryId (String): ID del directorio donde se subirá el archivo.
- Validación: Se verifica que el nombre del archivo solo contenga caracteres alfanuméricos, guiones y guiones bajos.
- Respuesta: Mensaje de éxito con el ID del archivo.

5. Descargar Archivo

- POST /file/download
- Descripción: Descarga un archivo basado en su ID.
- Parámetros:fileId (ObjectId): ID del archivo a descargar.
- Respuesta: Devuelve el archivo como un recurso de flujo de entrada.

6. Eliminar Archivo

- DELETE /file/deleted
- Descripción: Elimina un archivo específico.
- Parámetros:id (ObjectId): ID del archivo a eliminar.

- Respuesta: Mensaje de éxito o error.

7. Eliminar Archivo Compartido

- DELETE /file/deletedShare
- Descripción: Elimina un archivo compartido.
- Parámetros:id (ObjectId): ID del archivo compartido a eliminar.
- Respuesta: Mensaje de éxito o error.

8. Copiar Archivo

- POST /file/copy
- Descripción: Copia un archivo específico.
- Parámetros:id (ObjectId): ID del archivo a copiar.
- Respuesta: Mensaje de éxito o error.

9. Mover Archivo

- POST /file/move
- Descripción: Mueve un archivo a un nuevo directorio.
- Cuerpo de la Solicitud: Un objeto que contiene el ID del archivo y el ID del directorio de destino.
- Respuesta: Mensaje de éxito o error.

10. Compartir Archivo

- POST /file/share
- Descripción: Comparte un archivo con un usuario a través de su correo electrónico.
- Parámetros:id (String): ID del archivo a compartir.
- email (String): Correo electrónico del usuario con quien se comparte el archivo.
- Respuesta: Mensaje de éxito o error.

UserController

La clase UserController es un componente RESTful de Spring que se encarga de gestionar las operaciones relacionadas con el usuario en la aplicación. Proporciona endpoints para obtener información del usuario y cambiar su contraseña. La autenticación y la autorización se manejan mediante tokens JWT (JSON Web Tokens) que se extraen de las cookies en las solicitudes HTTP.

1. Anotaciones y Dependencias

- @RestController: Indica que esta clase es un controlador que gestiona solicitudes HTTP y responde con datos en formato JSON.
- @RequestMapping("/user"): Define la ruta base para los endpoints que maneja este controlador. Todas las rutas comenzarán con /user.
- private final JwtUtils jwtUtils: Utilidad para manejar la codificación y decodificación de tokens JWT. Se utiliza para validar y extraer información del token JWT presente en las cookies de la solicitud.
- @Autowired private UserService userService: Servicio que contiene la lógica de negocio relacionada con la gestión de usuarios. Se inyecta automáticamente por el contenedor de Spring.
- @Autowired private HttpServletRequest httpServletRequest: Proporciona acceso a la solicitud HTTP actual, permitiendo la extracción de cookies y otros datos de la solicitud.
- @Autowired private HttpServletResponse httpServletResponse: Proporciona acceso a la respuesta HTTP actual, utilizada principalmente para enviar respuestas al cliente.

2. Endpoints del Controlador

1. **@GetMapping("/get/info")**
   - Descripción: Endpoint para obtener información del usuario autenticado.
   - Retorno: Devuelve un objeto ResponseEntity<UserInfoResponse> que contiene los detalles del usuario.
   - Proceso:Extrae el token JWT de las cookies de la solicitud.
   - Extrae el ID del usuario del token.
   - Llama al servicio userService para recuperar la información del usuario basado en su ID.

2.@PostMapping("/change/password")

- Descripción: Endpoint para cambiar la contraseña del usuario autenticado.

- Parámetro: Recibe un objeto ChangePasswordDTORequest que contiene la nueva contraseña.
- Retorno: Devuelve un objeto ResponseEntity<String> que indica el resultado de la operación.
- Proceso:Extrae el token JWT de las cookies de la solicitud.
- Extrae el ID del usuario del token.
- Llama al servicio userService para cambiar la contraseña del usuario.

private String extractJwtFromCookie(HttpServletRequest request)

- Descripción: Método privado que extrae el token JWT de las cookies de la solicitud.
- Parámetro: HttpServletRequest request - la solicitud HTTP actual.
- Retorno: El valor del token JWT si se encuentra, de lo contrario, devuelve null.
- Proceso:Itera sobre las cookies de la solicitud y busca una llamada jwtToken.
- private String extractUserIDFromToken(String token)
- Descripción: Método privado que extrae el ID del usuario a partir del token JWT.
- Parámetro: String token - el token JWT.
- Retorno: El ID del usuario extraído del token.
- Proceso:Decodifica el token utilizando jwtUtils.
- Extrae los reclamos del token y devuelve el ID del usuario.

Servicios

AdminService

La clase AdminService es un componente de servicio en la aplicación que maneja la lógica de negocio relacionada con la gestión de archivos y directorios. Utiliza las capacidades de MongoDB para almacenar y recuperar datos, y se integra con varias capas de la aplicación, incluyendo repositorios y DTOs (Data Transfer Objects). Esta clase está anotada con @Service, lo que permite que Spring la gestione como un bean y la inyecte donde sea necesario.

1. Anotaciones y Dependencias

- La clase AdminService utiliza varias dependencias que son inyectadas a través de la anotación @Autowired:
-
- GridFsOperations y GridFsTemplate: Utilizados para interactuar con el sistema de archivos GridFS de MongoDB, que permite almacenar y recuperar archivos de gran tamaño.
- DirectoryShareRepository: Repositorio que proporciona métodos para interactuar con la base de datos en relación con los directorios compartidos.
- DirectoryRepository: Repositorio para gestionar operaciones sobre los directorios.
- UserDetailServiceImpl: Servicio que maneja la lógica de usuario, incluyendo la creación de usuarios.
- FileRepository: Repositorio que proporciona acceso a las operaciones de archivos.

2. Metodos

1. getDirectorys(String id)

- Descripción: Recupera todos los directorios que tienen como padre el directorio con el ID proporcionado, filtrando los que están marcados como eliminados.
- Parámetros:id: El ID del directorio padre, que es convertido a un objeto ObjectId.
- Retorno: Una lista de objetos DirectoryDTOResponse que representan los directorios recuperados.

2. getFilesWithContent(List<File> files)

- Descripción: Genera una lista de respuestas DTO para los archivos proporcionados, incluyendo el contenido de cada archivo.
- Parámetros:files: Una lista de objetos File.

- Retorno: Una lista de objetos FileDTOResponse que contienen los datos de los archivos, incluyendo su contenido.

3. buildFileDTOResponse(File file)

- Descripción: Construye un objeto FileDTOResponse a partir de un objeto File, recuperando el contenido del archivo desde GridFS.
- Parámetros:file: El objeto File del que se generará el DTO.
- Retorno: Un objeto FileDTOResponse que representa el archivo con su contenido.

4. buildFileDTOResponse(DirectoryShared directoryShared)

- Descripción: Similar a buildFileDTOResponse(File file), pero se utiliza para construir un DTO a partir de un objeto DirectoryShared.
- Parámetros:directoryShared: El objeto DirectoryShared del que se generará el DTO.
- Retorno: Un objeto FileDTOResponse que representa el archivo compartido con su contenido.

5. getSharedFilesWithContent(String id)

- Descripción: Recupera todos los archivos compartidos que pertenecen a un directorio específico, devolviendo el contenido de cada archivo.
- Parámetros:id: El ID del directorio cuyos archivos compartidos se recuperarán.
- Retorno: Una lista de objetos FileDTOResponse que representan los archivos compartidos.

6. createUser(CreateUserRequest createUserRequest)

- Descripción: Crea un nuevo usuario en el sistema utilizando los datos proporcionados en el DTO CreateUserRequest.
- Parámetros:createUserRequest: Objeto que contiene la información necesaria para crear un usuario.
- Retorno: Una respuesta HTTP que incluye el resultado de la creación del usuario y un código de estado 201 (CREATED).

DirectoryService

La clase DirectoryService es un componente central en la gestión de directorios y archivos en la aplicación, utilizando MongoDB como sistema de almacenamiento. Proporciona operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre directorios y archivos, así como funcionalidades adicionales como copiado y movimiento de directorios.

1. Anotaciones y Dependencias

- Repositories: Interactúa con tres repositorios:
- DirectoryRepository: Para la gestión de entidades de directorio.
- DirectoryInfoRepository: Para consultas específicas sobre la información del directorio.
- FileRepository: Para la gestión de archivos asociados a los directorios.
- GridFsTemplate: Utilizado para la manipulación de archivos almacenados en GridFS de MongoDB.
- Atributos
  - count: Un contador que se utiliza para rastrear la cantidad de operaciones de copia de directorios.

2. Métodos

1. createDirectory(DirectoryDTOResquest directoryDTOResquest, String id)

- Descripción: Crea un nuevo directorio si no existe uno con el mismo nombre y atributos para el usuario especificado.
- Parámetros:directoryDTOResquest: Objeto que contiene la información del nuevo directorio.
- id: ID del usuario que está creando el directorio.
- Retorno: ResponseEntity<String> que indica el resultado de la operación.

2. updateDirectory(UpdateDirectoryDTORequest updateDirectoryDTORequest, String id)

- Descripción: Actualiza el nombre de un directorio existente.
- Parámetros:updateDirectoryDTORequest: Contiene los datos del directorio a actualizar.
- id: ID del usuario que realiza la actualización.
- Retorno: ResponseEntity<String> que indica el resultado de la operación.

3. deleteDirectory(String id, String id_user)

- Descripción: Elimina un directorio y su contenido de forma recursiva.
- Parámetros:id: ID del directorio a eliminar.
- id_user: ID del usuario que posee el directorio.
- Retorno: String que indica el resultado de la eliminación.

4. getDirectorys(String id, String user_id)

- Descripción: Obtiene todos los directorios asociados a un usuario específico.
- Parámetros:id: ID del directorio padre.
- user_id: ID del usuario propietario de los directorios.
- Retorno: ResponseEntity<List<DirectoryDTOResponse>> que contiene la lista de directorios.

5. copyDirectory(ObjectId originalDirectoryId, ObjectId parentId, String user_id)

- Descripción: Realiza una copia de un directorio existente, incluyendo sus archivos y subdirectorios.
- Parámetros:originalDirectoryId: ID del directorio original.
- parentId: ID del directorio padre donde se almacenará la copia.
- user_id: ID del usuario que realiza la copia.
- Excepciones: Lanza FileNotFoundException si algún archivo asociado no se encuentra en GridFS.

6. moveDirectory(ObjectId originalDirectoryId, String user_id, ObjectId parentId)

- Descripción: Mueve un directorio a un nuevo padre.
- Parámetros:originalDirectoryId: ID del directorio a mover.
- user_id: ID del usuario propietario del directorio.
- parentId: ID del nuevo directorio padre.

FileService

El servicio FileService es un componente de Spring que gestiona las operaciones relacionadas con archivos en un sistema de gestión de archivos basado en MongoDB. Este servicio permite la carga, recuperación, actualización, eliminación, copia, movimiento y compartición de archivos.

1. Anotaciones y Dependencias

El servicio FileService utiliza varias dependencias clave, que se describen a continuación:

- GridFsTemplate: Proporciona métodos para almacenar y recuperar archivos desde GridFS.
- UserRepository: Interactúa con la base de datos para gestionar la información de los usuarios.
- FileRepository: Interactúa con la base de datos para gestionar la información de los archivos.
- DirectoryShareRepository: Maneja la información relacionada con archivos compartidos y directorios.
- GridFsOperations: Permite realizar operaciones sobre los archivos almacenados en GridFS.

2. Métodos

A continuación, se describen los métodos más relevantes de la clase FileService:

1. saveFile(MultipartFile file, String userId, ObjectId directoryId)

Este método permite guardar un archivo en GridFS y crear un registro en la base de datos.

- file: El archivo que se desea almacenar.
- userId: El ID del usuario que está subiendo el archivo.
- directoryId: El ID del directorio donde se almacenará el archivo.
- Retorno: Devuelve el archivo guardado o null si ya existe un archivo con el mismo nombre en el directorio.

2. updateFile(MultipartFile file, String userId, ObjectId directoryId, ObjectId id)

Este método actualiza un archivo existente en GridFS.

- file: El archivo actualizado.
- userId: El ID del usuario que está actualizando el archivo.
- directoryId: El ID del directorio donde se encuentra el archivo.
- id: El ID del archivo que se va a actualizar.

- Retorno: Devuelve el archivo actualizado o null si no coincide el ID.

3. getFilesWithContent(List<File> files)

Recupera una lista de archivos y su contenido en forma de FileDTOResponse.

- files: Una lista de objetos File.
- Retorno: Devuelve una lista de objetos FileDTOResponse que contienen los metadatos y el contenido de los archivos.

4. deleteFile(ObjectId fileId, ObjectId user)

Elimina un archivo físico y su registro en la base de datos, marcándolo como eliminado.

- fileId: El ID del archivo a eliminar.
- user: El ID del usuario que solicita la eliminación.

5. copyFile(ObjectId fileId, ObjectId userId)

Crea una copia de un archivo existente.

- fileId: El ID del archivo original.
- userId: El ID del usuario que realiza la copia.

6. moveFile(ObjectId originalDirectoryId, String userId, ObjectId parentId)

Mueve un archivo de un directorio a otro.

- originalDirectoryId: El ID del directorio original.
- userId: El ID del usuario que está moviendo el archivo.
- parentId: El ID del nuevo directorio donde se moverá el archivo.

7. shareFile(ObjectId originalDirectoryId, String userId, String email)

Permite compartir un archivo con otro usuario a través de su dirección de correo electrónico.

- originalDirectoryId: El ID del archivo que se va a compartir.
- userId: El ID del usuario que comparte el archivo.
- email: La dirección de correo electrónico del usuario con quien se comparte el archivo

UserService

La clase UserService es un componente del servicio de la aplicación api_grafiles_spring, que se encarga de la gestión de usuarios, incluyendo la recuperación de información del usuario y el cambio de contraseña. Esta clase utiliza los repositorios de usuario para realizar operaciones sobre la base de datos y está anotada con @Service, lo que la convierte en un componente de servicio gestionado por el contenedor de Spring.

1. Anotaciones y Dependencias

La clase depende de los siguientes componentes:

- UserInfoRepository: Repositorio para acceder a la información detallada de los usuarios.
- UserRepository: Repositorio principal para las operaciones relacionadas con los usuarios.
- PasswordEncoder: Componente de Spring Security que proporciona funcionalidades para codificar y verificar contraseñas.

2. Métodos

1. getInfoUser(String id)

Descripción: Recupera la información básica del usuario (nombre y correo electrónico) basado en su ID.

- String id: El identificador del usuario.
- Retorno:ResponseEntity<UserInfoResponse>: Devuelve un objeto que contiene la información del usuario. Si no se encuentra el usuario, retorna un ResponseEntity con un cuerpo nulo.
- Excepciones: Lanza una RuntimeException en caso de errores durante la recuperación de datos.

2. changePassword(String id_user, ChangePasswordDTORequest changePasswordDTORequest)

Descripción: Permite a un usuario cambiar su contraseña. Verifica si la contraseña actual es correcta antes de proceder con el cambio.

- String id_user: El identificador del usuario que solicita el cambio de contraseña.
- ChangePasswordDTORequest changePasswordDTORequest: Objeto que contiene la contraseña actual y la nueva contraseña.
- Retorno:ResponseEntity<String>: Devuelve un mensaje de éxito si el cambio fue exitoso o un mensaje de error si la contraseña actual es incorrecta.
- Excepciones: Lanza una RuntimeException en caso de errores durante la operación.

Repositorios

DirectoryRepository

DirectoryRepository es una interfaz que extiende MongoRepository para gestionar entidades de tipo Directory en una base de datos MongoDB. La interfaz proporciona métodos personalizados para realizar operaciones CRUD (Create, Read, Update, Delete) en documentos de directorios, con énfasis en la actualización de atributos y la recuperación de documentos no eliminados lógicamente. Se utiliza la anotación @Repository para indicar que esta es una clase de acceso a datos, y la anotación @Transactional para gestionar transacciones en algunos métodos.

1. Anotaciones y Dependencias

- MongoRepository: Proporciona las operaciones básicas de CRUD para trabajar con la base de datos MongoDB.
- Directory: Modelo de datos que representa la estructura de un directorio.
- ObjectId: Tipo de datos para manejar identificadores de objetos en MongoDB.

2. Métodos

1. updateNameByIdAndIsDeletedFalse(ObjectId id, String newName) Descripción: Actualiza el nombre de un directorio específico, siempre que el directorio no esté marcado como eliminado.

- ObjectId id: Identificador único del directorio.
- String newName: Nuevo nombre para el directorio.
- Retorno: No tiene retorno.
- Consulta: Utiliza una consulta personalizada @Query para encontrar el directorio por su ID y un @Update para modificar su nombre y la fecha de actualización.

2. findByDirectoryParentAndUserAndIsDeletedFalse(ObjectId directoryParentId, ObjectId user_id)

Descripción: Recupera una lista de directorios que tienen un directorio padre específico y pertenecen a un usuario, siempre que no estén eliminados.

- ObjectId directoryParentId: Identificador del directorio padre.
- ObjectId user_id: Identificador del usuario propietario.
- Retorno:List<Directory>: Lista de directorios que coinciden con los criterios.

3. findByNameAndUserAndDirectoryAndDirectoryParentAndIsDeletedFalse(String name, ObjectId user, int directory, ObjectId directoryParent)

Descripción: Encuentra un directorio específico basado en su nombre, usuario, un valor de tipo de directorio, y su directorio padre, siempre que no esté eliminado.

- String name: Nombre del directorio.
- ObjectId user: Identificador del usuario propietario.
- int directory: Tipo de directorio.
- ObjectId directoryParent: Identificador del directorio padre.
- Retorno:Directory: El directorio que cumple con los criterios de búsqueda.

4. findByNameAndUserAndDirectoryAndIsDeletedFalse(String name, ObjectId user, int directory)

Descripción: Encuentra un directorio por su nombre, usuario y tipo de directorio, siempre que no esté marcado como eliminado.

- String name: Nombre del directorio.
- ObjectId user: Identificador del usuario propietario.
- int directory: Tipo de directorio.
- Retorno:Directory: El directorio encontrado.

5. findAllByDirectoryParentAndIsDeletedTrue(ObjectId id)

Descripción: Recupera todos los directorios que tienen un directorio padre específico y que están marcados como eliminados.

- ObjectId id: Identificador del directorio padre.
- Retorno:List<Directory>: Lista de directorios eliminados.

6. findByIdAndIsDeletedFalse(ObjectId id)

Descripción: Encuentra un directorio por su ID, siempre que no esté marcado como eliminado.

- ObjectId id: Identificador del directorio.
- Retorno:Directory: El directorio encontrado.

7. findByNameAndUserAndDirectoryParentAndIsDeletedFalse(String name, ObjectId user, ObjectId parent_id)

Descripción: Busca un directorio basado en su nombre, usuario y directorio padre, siempre que no esté eliminado.

- String name: Nombre del directorio.
- ObjectId user: Identificador del usuario propietario.

- ObjectId parent_id: Identificador del directorio padre.
- Retorno:Directory: El directorio que cumple con los criterios.

8. newDeletedByIdAndUser(ObjectId id, ObjectId id_user)

- Descripción: Marca un directorio como eliminado, basándose en su ID y el ID del usuario propietario.
- ObjectId id: Identificador del directorio.
- ObjectId id_user: Identificador del usuario propietario.
- Retorno: No tiene retorno.
- Transacción: Utiliza @Transactional para asegurar la consistencia de la operación.

9. newupdateByIdAndDirectoryParentAndUserDirectoryAndAndIsDeletedFalse(ObjectId id, ObjectId id_user, ObjectId parent_id, int directory)

- Descripción: Actualiza el ID del directorio padre y el tipo de directorio, para un directorio específico perteneciente a un usuario y que no esté marcado como eliminado.
- ObjectId id: Identificador del directorio.
- ObjectId id_user: Identificador del usuario propietario.
- ObjectId parent_id: Nuevo identificador del directorio padre.
- int directory: Nuevo valor para el tipo de directorio.
- Retorno: No tiene retorno.
- Transacción: Utiliza @Transactional para asegurar la consistencia de la operación.

FileRepository

FileRepository es una interfaz que extiende MongoRepository para manejar la persistencia de objetos de tipo File en una base de datos MongoDB. Proporciona métodos personalizados para realizar operaciones CRUD (Create, Read, Update, Delete) sobre documentos de archivos, con un enfoque en la actualización de propiedades y la recuperación de archivos que no están eliminados de forma lógica. Se utiliza @Transactional para asegurar la consistencia en algunas de las operaciones de actualización y eliminación lógica.

1. Anotaciones y Dependencias

- MongoRepository: Proporciona las operaciones básicas de CRUD para interactuar con la base de datos MongoDB.
- File: Modelo de datos que representa la estructura de un archivo.
- ObjectId: Tipo de datos para manejar identificadores únicos de objetos en MongoDB.

2. Métodos

1. findAllByUserIdAndDirectoryIdAndIsDeletedFalse(ObjectId userId, ObjectId directoryId)

Descripción: Recupera todos los archivos que pertenecen a un usuario y están en un directorio específico, siempre que no estén eliminados.

- ObjectId userId: Identificador del usuario propietario de los archivos.
- ObjectId directoryId: Identificador del directorio en el que se encuentran los archivos.
- Retorno:List<File>: Lista de archivos que cumplen con los criterios de búsqueda.

2. newDeletedByIdAndUser(ObjectId id, ObjectId id_user)

Descripción: Marca un archivo como eliminado, basándose en su ID y el ID del usuario propietario.

- ObjectId id: Identificador del archivo.
- ObjectId id_user: Identificador del usuario propietario del archivo.
- Retorno: No tiene retorno.
- Transacción: Utiliza @Transactional para asegurar la consistencia de la operación.
- Consulta: Se define una consulta personalizada @Query que busca el archivo por su ID y user_id, y luego aplica un @Update para marcar el campo isDeleted como true.

3. findAllByDirectoryIdAndIsDeletedTrue(ObjectId id)

Descripción: Recupera todos los archivos que pertenecen a un directorio específico y que están marcados como eliminados.

- ObjectId id: Identificador del directorio.
- Retorno:List<File>: Lista de archivos eliminados que pertenecen al directorio especificado.

4. findByNameAndUserIdAndDirectoryIdAndIsDeletedFalse(String name, ObjectId user_id, ObjectId directory_id)

Descripción: Encuentra un archivo específico por su nombre, usuario y directorio, siempre que no esté marcado como eliminado.

- String name: Nombre del archivo.
- ObjectId user_id: Identificador del usuario propietario del archivo.
- ObjectId directory_id: Identificador del directorio al que pertenece el archivo.
- Retorno:File: El archivo que cumple con los criterios de búsqueda.

5. findByIdAndIsDeletedFalse(ObjectId fileId)

Descripción: Busca un archivo por su ID, siempre que no esté marcado como eliminado.

- ObjectId fileId: Identificador del archivo.
- Retorno:File: El archivo encontrado que no está eliminado.

6. newupdateByIdAndDirectoryParentAndUserDirectoryAndAndIsDeletedFalse(ObjectId id, ObjectId id_user, ObjectId parent_id)

Descripción: Actualiza el directorio padre de un archivo, siempre que el archivo no esté marcado como eliminado y pertenezca a un usuario específico.

- ObjectId id: Identificador del archivo.
- ObjectId id_user: Identificador del usuario propietario del archivo.
- ObjectId parent_id: Nuevo identificador del directorio padre.
- Retorno: No tiene retorno.
- Transacción: Utiliza @Transactional para garantizar la consistencia de la operación.
- Consulta: Se define una consulta personalizada @Query que busca el archivo por su ID y user_id, y luego aplica un @Update para cambiar el directory_id.

UserRepository

UserRepository es una interfaz que extiende MongoRepository para manejar la persistencia de objetos de tipo UserModel en una base de datos MongoDB. Proporciona métodos personalizados para realizar operaciones de búsqueda y actualización de usuarios, enfocándose en la recuperación de datos de usuarios por atributos clave como nombre de usuario y correo electrónico, así como en la actualización de contraseñas. Algunas operaciones utilizan transacciones para asegurar la consistencia de los datos.

1. Anotaciones y Dependencias

- MongoRepository: Proporciona las operaciones básicas de CRUD para interactuar con la base de datos MongoDB.
- UserModel: Modelo de datos que representa la estructura de un usuario.
- UserPasswordDTO: Objeto de transferencia de datos (DTO) que contiene información de la contraseña del usuario.
- ObjectId: Tipo de datos para manejar identificadores únicos de objetos en MongoDB.

2. Métodos

1. findUserEntityByUsername(String username)

Descripción: Busca un usuario específico por su nombre de usuario.

- String username: Nombre de usuario por el que se realiza la búsqueda.
- Retorno:Optional<UserModel>: Un Optional que puede contener el UserModel correspondiente al nombre de usuario proporcionado, o vacío si no se encuentra.

2. findByEmail(String email)

Descripción: Encuentra un usuario por su dirección de correo electrónico.

- String email: Correo electrónico del usuario.
- Retorno:Optional<UserModel>: Un Optional que puede contener el UserModel correspondiente al correo electrónico proporcionado, o vacío si no se encuentra.

3. findById(ObjectId id)

Descripción: Recupera un usuario por su ID, devolviendo un objeto UserPasswordDTO que incluye información de la contraseña.

- ObjectId id: Identificador único del usuario en la base de datos.
- Retorno:Optional<UserPasswordDTO>: Un Optional que puede contener el UserPasswordDTO correspondiente al ID proporcionado, o vacío si no se encuentra.

4. newPassword(ObjectId id, String password)

- Descripción: Actualiza la contraseña de un usuario, buscando el usuario por su ID.
- ObjectId id: Identificador único del usuario.
- String password: Nueva contraseña encriptada para el usuario.
- Retorno: No tiene retorno.
- Transacción: Utiliza @Transactional para asegurar que la operación de actualización sea atómica y se realice de manera consistente.
- Consulta: Se define una consulta personalizada @Query que busca el usuario por su ID y un @Update para actualizar el campo password.

Instalación

Instalación con docker

1. Primer Paso instalar docker

   Primero instala docker desde la pagina oficial https://www.docker.com/

2. Segundo Paso

   Clonar el siguiente repositorio https://github.com/117CarlosCoder/Proyecto_Final_GraFiles

3. Tercer Paso

   Al clonar el repositorio anterior ingrese dentro de la carpeta principal /Proyecto_Final_GraFiles , estando dentro de esta carpeta clone los dos repositorios siguientes:

   1. https://github.com/117CarlosCoder/Frontend_Vue_GraFiles
   2. https://github.com/117CarlosCoder/Backend_Spring_GraFiles

   En la carpeta tiene que tener la siguiente estructura de carpetas
   .

   ├── Backend

   ├── Backend_Spring_GraFiles

   ├── Documentos

   ├── Frontend

   └── Frontend_Vue_GraFiles

4. Cuarto Paso

   Remplace lo que esta dentro de la carpeta Backend con Backend_Spring_GraFiles y remplace lo que esta dentro de la carpeta Frontend con Frontend_Vue_GraFiles

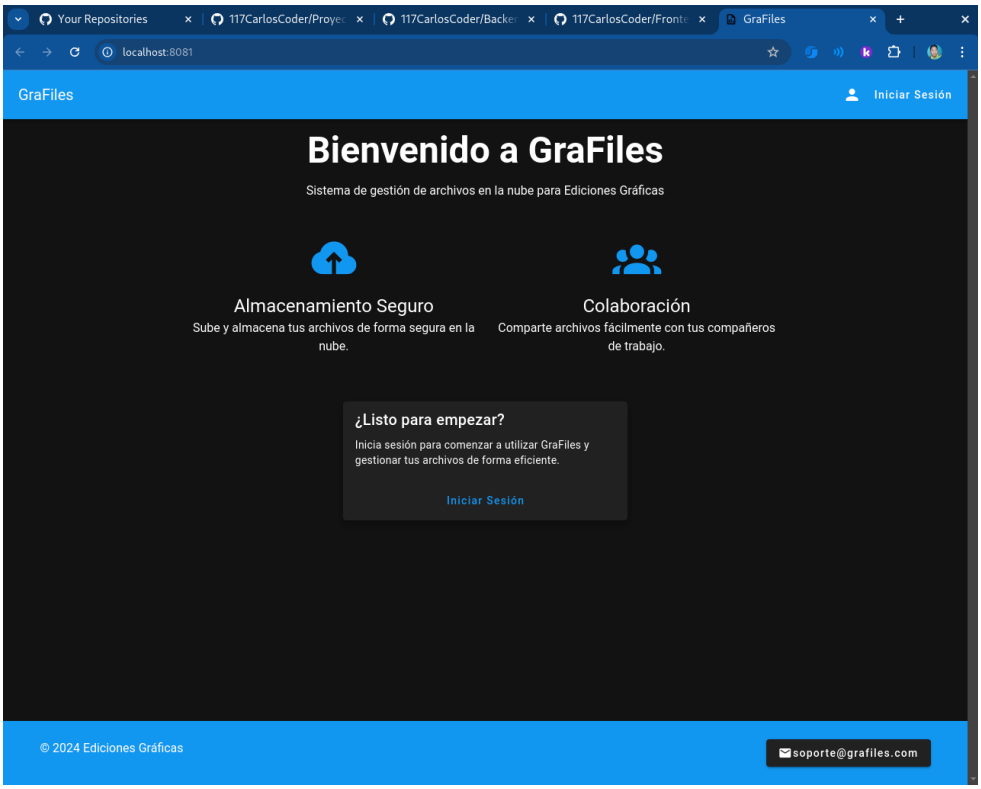
5. Quinto Paso

   Abra una terminal y ejecute el siguiente comando

   ```
   docker compose up -d
   ```

esto creara un compose en docker con el nombre proyecto_final_grafiles el cual también se va ejecutar con el mismo comando y sino hay ningún conflicto con la creación del compose y con su ejecución podrá entrar al local host puerto 8081

Tendria que ver lo siguiente:



6. Sexto paso

   Para poder iniciar sesión necesita cargar la información a la base de datos, para usando mongodb ello usando mongosh y conectese atreves de este comando

   mongosh --port 27018

   Después ingrese el siguiente comando , este cargara la base de datos con 3 usuario para usar la aplicación

   load("./Documentos/Script.js");

   El script se encuentra en la carpeta Documentos por si necesita modificar o agregar datos

7. Septimo paso

   Para iniciar sesión dentro de la aplicación estos son los usuario y contraseñas:

1.

usuario : carlos117

contraseña : 1234

2.

usuario : daniel
contraseña : 1234

3.

usuario : anyi
contraseña : 1234

Instalación sin docker

1.Primer Paso

Clonar los dos repositorios siguientes: en la misma carpeta

1. https://github.com/117CarlosCoder/Frontend_Vue_GraFiles

2. https://github.com/117CarlosCoder/Backend_Spring_GraFiles

2.Segundo Paso

Tener instalado npm

Ejecutar dentro del proyecto frontend vue el siguiente comando

npm -i

Cuando termine ejecutar el siguiente comando

npm run dev
Esto iniciara la aplicación en el localhost puerto 3000

3. Tercer Paso

Usar algún ide compatible con spring, si utiliza intellij idea lo único que tendrá que hacer es actualizar la versión de java para este proyecto, la versión que utiliza es java 21, automáticamente intellij hara los cambios y bastara solo con ejecutar el main prara iniciar la api en el puerto 8080

4. Cuarto Paso

Debe tener instalado mongodb y mongosh para poder conectarse con el siguiente comando

mongosh --port 27018

Después ingrese el siguiente comando , este cargara la base de datos con 3 usuario para usar la aplicación

load("./Documentos/Script.js");

El script se encuentra en la carpeta Documentos por si necesita modificar o agregar datos