Manual Técnico

GraFiles

Brandon Josué Pinto Méndez

201930236

Introducción:

La aplicación GraFiles suple la necesidad de una aplicación de guardado de documentos en la nube la cual cumple con las herramientas básicas que necesita una aplicación de esta índole para poder funcionar con el fin de apoyar al flujo de datos en un ambiente laboral.

Esta aplicacion consta de una arquitectura distribuida con tres componentes principales:

Frontend: Aplicación web desarrollada en Angular.

Backend: API REST construida en Java utilizando Maven, Tomcat y Servlets.

Base de Datos: MongoDB.

Dockerización.

Docker permite crear contenedores aislados para cada componente de la aplicación. La dockerización facilita el despliegue, administración y escalabilidad de las aplicaciones al encapsular el software junto con todas sus dependencias en un solo contenedor. Esto ayuda a crear un entorno homogéneo, eliminando problemas de compatibilidad entre entornos y simplificando el proceso de despliegue en distintos sistemas.

Configuración del Entorno Docker

A continuación, se describe el proceso de configuración de cada componente en contenedores Docker.

1. Frontend en Angular

```
FROM node:20 AS build
WORKDIR /app
COPY package.json /app
RUN npm install
```

```
COPY . /app
RUN npm run build --prod
FROM nginx:alpine
COPY --from=build /app/dist/frontend /usr/share/nginx/html
EXPOSE 4200
CMD ["nginx", "-g", "daemon off;"]
```

Explicación

Etapa de construcción:

Se utiliza la imagen de Node.js (node:20) para compilar el proyecto Angular.

Se crea el directorio /app y se configura como el directorio de trabajo.

Se copian los archivos de Angular, se instalan las dependencias y se compila el proyecto en modo de producción (npm run build --prod).

Etapa de despliegue:

Se utiliza la imagen nginx:alpine para servir los archivos estáticos de Angular.

El contenido compilado en dist/frontend se copia al directorio de Nginx /usr/share/nginx/html.

Se expone el puerto 80 para el acceso a la aplicación web. Explicación

Etapa de construcción:

Se utiliza la imagen de Node.js (node:20) para compilar el proyecto Angular.

Se crea el directorio /app y se configura como el directorio de trabajo.

Se copian los archivos de Angular, se instalan las dependencias y se compila el proyecto en modo de producción (npm run build --prod).

Etapa de despliegue:

Se utiliza la imagen nginx:alpine para servir los archivos estáticos de Angular.

El contenido compilado en dist/frontend se copia al directorio de Nginx /usr/share/nginx/html.

Se expone el puerto 4200 para el acceso a la aplicación web.

2. **Dockerfile para el Backend**

```
FROM maven:3-openjdk-17 AS build
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn clean package -DskipTests
RUN ls -l /app/target
FROM tomcat:9.0
COPY --from=build /app/target/GraFiles-1.0.war /usr/local/tomcat/webapps/
EXPOSE 8080
```

Etapa de construcción:

Se utiliza la imagen de Maven con OpenJDK 17 para compilar el proyecto.
El archivo pom.xml y el directorio src se copian al contenedor.
La aplicación se compila utilizando el comando mvn clean package -DskipTests, generando un archivo .war en el directorio target.
Etapa de despliegue:

Se usa la imagen tomcat:9.0 para desplegar la aplicación.
Se copia el archivo .war generado (GraFiles-1.0.war) al directorio webapps de Tomcat, que se encarga de desplegar automáticamente el proyecto.
Se expone el puerto 8080 para acceder al Backend.

Base de Datos: MongoDB

Para MongoDB, se utiliza la imagen mongodb/mongodb-enterprise-server:latest. A continuación, se muestra el comando docker run para ejecutar la base de datos:

```
docker run -d --name mongodb \
 -p 27017:27017 \
 --restart unless-stopped \
 mongodb/mongodb-enterprise-server:latest \
 mongod --bind_ip_all --auth false mongodb/mongodb-enterprise-server:latest
```

Se utiliza la imagen mongodb/mongodb-enterprise-server:latest.
Se define el nombre del contenedor (--name mongodb).
Se mapea el puerto 27017 del contenedor al puerto 27017 del host.
Se establecen las variables de entorno para el usuario y la contraseña de administración (admin/adminpassword).

Dockerizar cada componente de esta aplicación ayuda a crear entornos consistentes y portables, asegurando que se ejecute de forma homogénea sin importar el sistema subyacente. Además, Docker simplifica el despliegue y la escalabilidad, haciendo que sea fácil clonar y configurar entornos de desarrollo y producción.

Clase de manejo de archivos de la base de datos MongoDB

ObtenerArchivosPropio:

Obtiene archivos habilitados de un propietario específico (PropietarioId) y en una carpeta específica (PadreId).
Query: db.archivo.find({ PropietarioId: <Id>, Habilitado: true, PadreId: <PadreId> })

ObtenerArchivosAdmin:

Recupera archivos que están deshabilitados (Habilitado: false), útiles para la administración de archivos inactivos.
Query: db.archivo.find({ Habilitado: false })

ObtenerArchivosAbuelo:

Busca archivos en una estructura anidada: primero busca un archivo específico (_id: PadreId) y, luego, usa su ID como PadreId para buscar archivos habilitados del propietario (PropietarioId).
Query (ejecución en dos pasos):
db.archivo.find({ PropietarioId: <Id>, Habilitado: true, _id: <PadreId> })
db.archivo.find({ PropietarioId: <Id>, Habilitado: true, PadreId: ObjectId('<result_of_first_query>') })

CrearCarpeta:

Verifica si existe una carpeta con el mismo nombre y PadreId. Si no existe, inserta una nueva carpeta.
Query (verificación): db.archivo.find({ Nombre: '<Nombre>', PadreId: <PadreId>, Habilitado: true }).limit(1).count()
Query (inserción): db.archivo.insertOne({ Nombre: '<Nombre>', Extension: 'carpeta', PadreId: <PadreId>, FechaCreacion: '<fecha>', PropietarioId: <CreadorId>, Habilitado: true })

CrearTexto:

Similar a CrearCarpeta, verifica si el archivo ya existe y, si no, inserta un nuevo documento de tipo texto.

Query (verificación): db.archivo.find({ Nombre: '<Nombre>', PadreId: <PadreId>, Habilitado: true }).limit(1).count()
Query (inserción): db.archivo.insertOne({ Nombre: '<Nombre>', Extension: '<Extension>', PadreId: <PadreId>, FechaCreacion: '<fecha>', FechaModificacion: '<fecha>', Contenido: '<Contenido>', PropietarioId: <CreadorId>, Habilitado: true })

DesactivarArchivos:

Actualiza el estado de habilitación de un archivo y, si tiene subarchivos (PadreId), aplica la desactivación de forma recursiva.
Query (actualización): db.archivo.updateOne({ _id: <id>, Habilitado: true }, { $set: { Habilitado: false } })
Query (buscar hijos): db.archivo.find({ PadreId: <id>, Habilitado: true }).toArray()
Mover:

Actualiza el campo PadreId del archivo especificado (_id) para cambiar su ubicación.
Query: db.archivo.updateOne({ _id: <Id>, Habilitado: true }, { $set: { PadreId: <IdPadre> } })

ActualizarTexto:

Verifica si ya existe un archivo con el mismo nombre en el nuevo PadreId. Si no existe, actualiza los datos del archivo.
Query (verificación): db.archivo.find({ Nombre: '<Nombre>', PadreId: <PadreId>, Habilitado: true }).limit(1).count()
Query (actualización): db.archivo.updateOne({ _id: <Id>, Habilitado: true }, { $set: { Nombre: '<Nombre>', Extension: '<Extension>', FechaModificacion: '<fecha>', Contenido: '<Contenido>' } })

CompartirArchivo:

Consulta la ID de un usuario y, si es encontrado, copia un archivo específico (Nombre, Extension, PadreId) con los datos proporcionados.
Query (buscar usuario): db.usuario.find({ Username: '<NombreU>' }).limit(1)
Query (inserción): db.archivo.insertOne({ Nombre: '<Nombre>', Extension: '<Extension>', PadreId: <idPadre>, FechaCreacion: '<FechaC>', FechaModificacion: '<FechaM>', Contenido: '<Contenido>', PropietarioId: <id>, Habilitado: true })

Clase de manejo de usuarios de la base de datos MongoDB

comprobarContraseña(String usuario, String contra)
Descripción: Verifica si el usuario ingresado existe en la base de datos y si la contra (contraseña) coincide con el hash almacenado.

Proceso:
Consulta a MongoDB para obtener el documento del usuario con el nombre de usuario especificado.
Si el usuario existe, toma la contraseña almacenada (Password) y su rol (Rol).
Calcula el hash de la contraseña proporcionada usando SHA-256 y lo compara con el hash almacenado.
Devuelve un array con el estado de la autenticación, el nombre de usuario, el rol y el ID.
Query en MongoDB:
js
db.usuario.findOne({ Username: '<usuario>' });

2. bytesToHex(byte[] bytes)
Descripción: Convierte un arreglo de bytes en su representación hexadecimal, útil para mostrar el hash en un formato legible.
Propósito: Solo transforma el hash calculado en un string hexadecimal, no interactúa directamente con MongoDB.

3. CrearUsuario(String User, String password, String name)
Descripción: Crea un nuevo usuario en la base de datos con el nombre de usuario (User), la contraseña (password) y el nombre (name).
Proceso:
Verifica si el User ya existe en la base de datos.
Si no existe, calcula el hash SHA-256 de la contraseña y lo guarda en MongoDB junto con el nombre de usuario y el nombre.
Asigna el rol de "Empleado" por defecto.
Queries en MongoDB:
Para verificar la existencia del usuario:
db.usuario.find({ Username: '<User>' }).limit(1).count();
Para insertar el nuevo usuario:
db.usuario.insertOne({ Username: '<User>', Password: '<hashed_password>',