Teoría Java Project

Estructura proyecto back

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente con confianza media

Estructura proyecto Front (Vite-React)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Después he creado los repositos para cada projecto y conectado correctamente

Luego he creado un private Project con tickets/tasks que voy completando

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Se crea una rama feature/read-expenses y se hace un fetch all en Intellij para poder trabajar en esa rama

Estos son los pasos del desarrollo de la API

Texto

Descripción generada automáticamente con confianza baja

Primero para crear en la BD en Workbench

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez creada cambiamos la configuración del Project .java para la conexión y configuración en application.properties:

Texto

Descripción generada automáticamente

Ahora tenemos que añadir nuevas dependencias al proyecto

Texto

Descripción generada automáticamente

Una vez creadas y sincronizadas/descargadas (View > Tool Window > Maven > Sync) las dependencias con Maven se pueden crear los Objetos IO (Input Output)

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

después crearemos los request objects que con el @data nos crea varios métodos solo

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Por ultimo creamos el response object

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Crear un DTO Object para Expense, es un patron de diseño.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Imagen de la pantalla de un celular con texto

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Carta, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Después de definir los pasos a seguir, desglosamos punto por punto.

Primero creamos el GET API endpoint

Dentro de /src/main/java

En el archivo por defecto creamos un nuevo package llamado Controller y dentro una clase nueva llanada ExpenseController

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Ahora para probarlo, se puede probar en el documento por defecto RestapiApplication.java, nos debería devolver Reading the expenses from database.

Texto

Descripción generada automáticamente

En la terminar integrada se construye

Ahora tomcat te corre en el localhost:8080/ruta y yo ponía /expenses para que saliera pero estaba mal pq había añadido un prefijo /api/v1 en la URL

Texto

Descripción generada automáticamente

Ahora ya funciona con localhost:8080/expenses

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ahora vamos a intentar leer los expenses de la BD

Crearemos los IO Objects (Los objetos de petición y respuesta)

Dentro de src/main/java/in.bushansirgur.restapi, creamos otro package pero para io y dentro dos clases nuevas

Captura de pantalla de un celular

Descripción generada automáticamente

Creamos la primera clase con sus import necesarios

Texto

Descripción generada automáticamente

Se crea la segunda clase

Texto

Descripción generada automáticamente

Ahora vamos a crear los DTO (Data Transfer Object)

Creamos nuevo paquete dto y clase ExpenseDTO, que tiene el mismo contenido que ExpenseResponse

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Vamos a añadir las dependencias necesarias para que se pueda conectar correctamente con nuestra BD de MySQL. En Spring initializer añadimos estas dos dependencias:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Se copian y se añaden al pom.xml

Una vez añadidas crearemos la BD y la configuremos en nuestra app

En Workbench creamos DB expenesetransactions

(Aunque lo hice antes de tiempo se configura en application.properties). He añadido el prefijo de Url de nuevo

Texto

Descripción generada automáticamente

Ahora vamos a crear la entidad JPA para expense Object

Igual que otras veces, creamos paquete entity y dentro clase ExpenseEntity

El contenido de esta clase esta mas relacionado con la tabla y hay que indicar id, y restricciones

Texto

Descripción generada automáticamente

Ahora creamos el repositorio JPA para ExpenseEntity

Dentro de un nuevo paquete repository, creamos una interfaz llamada ExpenseRepository con sus correspondiente herencia

Captura de pantalla de computadora

Descripción generada automáticamente

Ahora configuraremos la dependencia del modelMapper (Biblioteca para mapear objetos de un tipo a otro, útil para convertir entidades en DTOs y viceversa )

Para eso vamos a modelmapper.org, copiamos la dependencia y la ponemos en el pom.xml

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ahora en el archivo principal creamos un nuevo método ModelMapper que creara un nuevo modelMapper

Texto

Descripción generada automáticamente

Ahora crearemos la interfaz para el servicio

Creamos nuevo paquete llamado service y una interfaz llamada ExpenseService

Captura de pantalla de computadora

Descripción generada automáticamente

Dentro del paquete service creamos otro paquete llamado impl (Implementation) con una clase dentro llamada ExpenseServiceImpl

Texto

Descripción generada automáticamente

Ahora llamaremos al servicio en nuestro controlador

Ahora actualizamos el ExpenseController

Texto

Descripción generada automáticamente

Ahora vamos a testear la API endpoint

Primero insertamos datos de prueba con workbench

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Cuando se hayan insertado correctamente, viajamos a localhost:8080/api/v1/expenses

Texto

Descripción generada automáticamente

Añadimos ahora javadocs a todos los métodos

Asi una vez añadido correctamente cuando hagamos hover sobre la función te dice que hace y parámetros y return

Texto

Descripción generada automáticamente

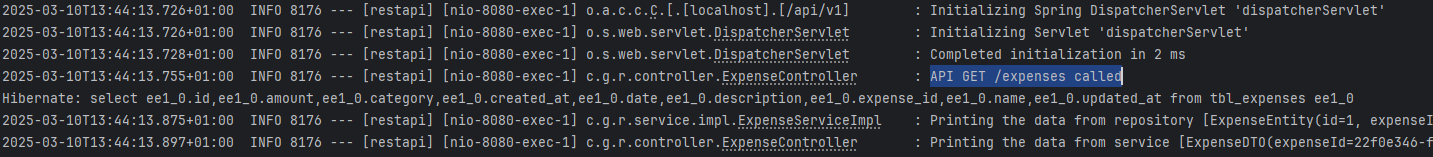
Ahora vamos a añadir los loggers (Añadirlo cada poco tiempo, asi cuando lo mandemos a producción podemos ver que fallo con los logs)

Usaremos la librería Slf4j para acceder a sus métodos de logs

Texto

Descripción generada automáticamente

Si corres la app de nuevo y haces una nueva petición (Refrescas), en la terminal se pueden ver los logs



Ahora ya podemos hacer un commit + push