Caja Control Pro

Requerimientos:

- javascript
- node 20.16.0

- prisma 6.18.0 (ORM)

- npm 9.2.0
- acorn 8.8.1
- tailwindcss 4.1.15
- vite 6.4.1
- react 19.2.2
- dotenv 17.2.3
- electron-builder 26.0.12

Tutorial

1. Clonar el repositorio

```Shell
cd
mkdir caja-control-pro
cd caja-control-pro
git clone https://github.com/jmpizza/no_informado.git
```

2. Instalar con npm

```Shell
npm install
npm run build
npm init -y
npm install prisma typescript tsx @types/node --save-dev
```

3. Montar la base de datos

```shell
npm install prisma
npx prisma init --datasource-provider postgresql --output ../generated/prisma
npx prisma migrate dev --name init
npx install @prisma/client
npx db pull
npx prisma generate
```

3.1. Agregar la URL de la base de datos a DATABASE_URL en el archivo .env
3.2. Modificar *schema.prisma* con el campo `" provider = “prisma-client-js”`.

Ejecución

Correr el siguiente comando en la carpeta:

```shell
npm run dev:all
```

Contenido del docker-composer.yml

```shell
version: "3.9"

services:

db:

container_name: ${DB_NAME}

image: postgres:17.6

restart: always

ports:

- "${DB_PORT}:5432"

environment:

POSTGRES_USER: ${DB_USER}

POSTGRES_PASSWORD: ${DB_PASSWORD}

POSTGRES_DB: ${DB_NAME}

volumes:

- ./initdb.d:/docker-entrypoint-initdb.d
```

Documentación de los patrones de diseño

DatabaseSingle.js

Crea un objeto *DatabaseSingle.js* que implementa el patrón singleton para la conexión de la base de datos. Al instanciar esta clase se crea una instancia del objeto *prisma* que es la conexión de la base de datos a través del ORM prisma y lo añade a la variable de instancia.

```
public static getInstance()
```

Crea una instancia del Singleton que crea la única conexión a la base de datos del proyecto a través del ORM. Se sugiere interactuar con la base de datos a través de la variable de instancia prisma para las operaciones CRUD
Ejemplo:

```
Shell
ConexionDB = DatabaseSingle.getInstance().prisma
```

AlertasObserver.js
Escenario de uso: Patrón *Observer*

Este patrón se adapta cómodamente al manejo de logs para movimientos de caja en el sistema, ya que no es necesario hacer modificaciones en la base de datos para su implementación. Podemos usar a los logs como observadores que cada vez que se realiza un movimiento de caja definiendo los movimientos como los eventos.

En cuanto a su implementación, cada vez que se realiza un movimiento de caja, el evento notifica a los observadores registrados que, en nuestro caso, es el sistema de logs. Este último, al recibir la notificación, registra la información correspondiente al movimiento en la tabla de logs de nuestra base de datos, permitiendo un registro de lo que pasa en la aplicación en caso de ser necesario.

De esta manera, obtenemos una serie de ventajas significativas a la hora de de desarrollar, añadir o corregir el código de la aplicación. Algunas de estas ventajas son:

- Centraliza la lógica de auditoría: Evita tener logs en cada momento del código donde se intenta realizar un movimiento.

- Escala fácilmente: Para nuestro modelo de datos, pueden introducirse nuevos eventos (cierres, alertas, etc.) que notifiquen al observador y registre sus logs según sea el caso.

- Consistencia: Es sencillo identificar errores relacionados, pues todo pasa por el mismo flujo.

Codigo:

En nuestro proyecto usamos prisma como sistema de ORM, el siguiente código muestra una posible implementación en nuestro código usando conexionBD (Nuestro singleton par manipular la base de datos):

JavaScript

```
import { PrismaClient } from "@prisma/client";

class DatabaseSingle {

static instance;

prisma;

constructor() {

this.prisma = new PrismaClient();

}

static getInstance() {

if (!DatabaseSingle.instance) {

DatabaseSingle.instance = new DatabaseSingle();

}

return DatabaseSingle.instance;

}

}

export default DatabaseSingle;
```

Interfaz gráfica demostrativa