Proyecto final: Taller de Patrones de diseño

Estructura general del proyecto:

```
template-nodejs/
├── .vscode/
├── db/
├── src/
├── .environment.example
├── Dockerfile
├── README.md
├── docker-compose.yml
├── environment.d.ts
├── package.json
├── tsconfig.app.json
├── tsconfig.json
├── tsconfig.spec.json
├── tslint.json
└── webpack.config.js
```

Cómo podemos observar en el diagrama anterior, hay 2 carpetas principales:

- db/ : Esta carpeta la podemos asociar a la capa de **infraestructura**. Puesto que en esta carpeta se encuentran contenidos dos archivos de migraciones de bases de datos.

- src/ : En esta carpeta encontraremos los los archivos pertenecientes a las capas de dominio más profundas tales cómo los servicios de **aplicación**, **servicios de dominio** y **dominio**.

Posteriormente, tenemos los siguientes documentos de configuración a nivel raíz del proyecto:

- .environment.example: declara las variables de entorno necesarias para la ejecución de la app.

- Dockerfile: Archivo de configuración de Docker para construir la imagen del contenedor.

- README.md: Documentación del proyecto.

- docker-compose.yml: Configuración de Docker Compose para orquestar contenedores.
- environment.d.ts: Declaraciones de tipos para variables de entorno.
- package.json: Dependencias y scripts del proyecto.
- tsconfig.app.json: Configuración de TypeScript para la aplicación.
- tsconfig.json: Configuración principal de TypeScript.
- tsconfig.spec.json: Configuración de TypeScript para pruebas.
- tslint.json: Configuración de TSLint.
- webpack.config.js: Configuración de Webpack para el empaquetado del proyecto.

Los archivos de configuración listados, pertenecen a la parte de la lógica de aplicación.

Carpeta src:

```
src/
├── common/
├── config/
├── constants/
├── controllers/
├── data-access/
├── domain/
├── interfaces/
├── platform/
├── routes/v1/
├── util/
└── index.ts
```

Common:

En este folder nos encontramos con dos **servicios de aplicación:**

- Auth-service: servicio de autenticación de usuarios.
- Cache-service: servicio de manejo de caché de la aplicación

En los componentes descritos en este folder encontramos:

- Declaración e implementación de la interfaz de IAuthService. Además, la AuthService (que implementa la interfaz) es declarada cómo una clase inyectable. Aquí es dónde vemos por primera vez la inyección de dependencias implementada por primera vez. En este caso me puedo decir que cada vez que declaramos una clase cómo inyectable, se puede calificar cómo una implementación de singleton, puesto que se busca crear un objeto estático dentro del contenedor de control de dependencias para evitar su repetición y que sea inyectado cuándo se necesite.

  En este caso, podemos ver polimorfismo

- Interfaz de CacheService

Config y constants:

En este folder nos encontramos archivos de configuración de Redis y postgresql, además los archivos de configuración de nuestro contenedor de inyección de dependencias de nuestra aplicación (Inversify). En este caso, el folder config y todos los archivos en él pertenecen a la capa de **infraestructura**.

Controllers:

En esta carpeta nos encontramos con los **servicios de entidad** de la aplicación. Se presenta una clase abstracta base controller cómo inyectable. Posteriormente se extiende en dos subclases: UserSigninController y UserSignupController, encargadas del login y la creación de nuevos usuarios. En estas clases podemos encontrar un patrón de diseño de Data Transfer Object para manejar la data del usuario.

Data access:

En esta carpeta encontramos dos implementaciones: Caché Service integrado con Redis y el user Repository para la base de datos de postgresql. Por ello, he de decir que las implementaciones ahí contenidas pertenecen a la **capa de infraestructura**.

Domain:

Aquí encontramos la interfaz de usecase, y sus implementaciones. Las siguientes implementaciones corresponden a la **capa de dominio**, es decir las reglas del negocio:

- **Entidades:**
  - User
- **Repositorio:**
  - User Repository
- **Servicios:**
  - Sign in
  - Sign up

En los servicios, además se declaran los Data Transfer Objects.

Los componentes que vemos declarados, son implementados en la carpeta de controllers.

En esta carpeta vemos más polimorfismo, herencia y DTO.

Interfaces:

En esta carpeta vemos dos interfaces que funcionan cómo estructuras de datos para objetos JSON y para los errores personalizados de la aplicación. Estas interfaces corresponden a **la capa de infraestructura.**

Platform:

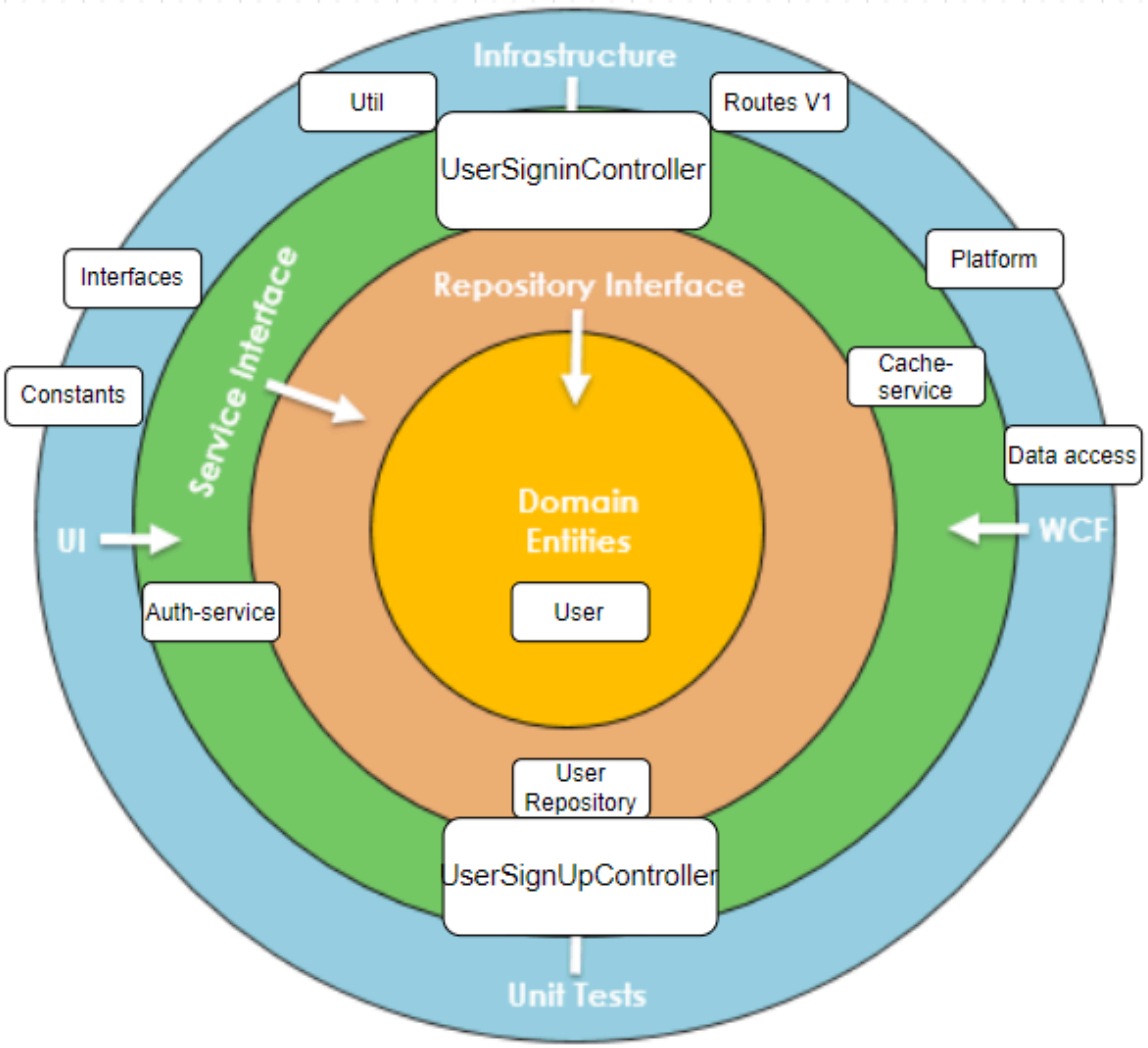
En esta carpeta se encuentra el middleware y la configuración del servidor express de nuestro proyecto. Es decir, pertenece a **la capa de infraestructura** y a la lógica de aplicación.

Routes v1:

Capa de infraestructura.

Util:

Capa de infraestructura.



Resumen:

A lo largo del proyecto se observan todos los principios SOLID implementados en las distintas entidades que interactúan.

En cuanto a patrones de diseño, cómo se menciona a lo largo del documento podemos observar el singleton y la inyección de dependencias implementados en la mayor parte del proyecto. Podemos decir también que el bridge para desacoplar algunas clases.