ZFCocteles: Patrón de Diseño

Patrón de diseño elegido: Builder

Se trata de un patrón de diseño creacional en el cual se permite la construcción de objetos complejos paso a paso, es decir, cuando es necesario crear un objeto con múltiples configuraciones o pasos pero a la vez se quiere evitar declarar constructores con muchos parámetros opcionales, para lograr esto se separa el código de construcción del objeto de la clase y se divide en varios segmentos independientes, cada uno de los cuales se encargará de añadir las características opcionales requeridas al objeto.

Justificación y contexto:

Un cóctel puede suponer una alta complejidad a la hora de definir su preparación ya que tienen una estructura variable como lo es: distintos ingredientes, decoración, alcohol sí/no, categoría, receta modificada o preexistente, etc. Por ello es útil una configuración que permita construir objetos en pasos que dividan el proceso de construcción de la receta en cuestión evitando un único constructor excesivamente largo lo que además permite controlar el paso a paso en la preparación, modificación y creación de un cóctel.

Implementación:

Se plantea a cada cóctel no sólo como una tabla en SQLite, sino un objeto con propiedades como nombre, ingredientes, categoría, imagen, instrucciones, etc. Para la construcción de este objeto serán necesarios una serie de pasos (setters encadenables o métodos específicos) en lugar de pasar todo de golpe a un constructor.

Uso:

De momento no es posible mostrar código de cómo está implementado el patrón en el estado actual del proyecto, sin embargo en la presente estructura del código se ha creado el `renderer/builders/`. Eso muestra que se está organizando el proyecto según este patrón ya que aquí se alojarán las clases Builder de las entidades que manejan la aplicación.

```
ZFCFANS/
├── .vscode/                    # Configuración del entorno de desarrollo.
├── ...
  electron-app/
  ├── docs                      # Documentación del proyecto
  ├── public/                   # Archivos estáticos opcionales
  ├── src/
  │   ├── main/                 # Proceso principal de Electron
  │   │   ├── db/               # Conexión y lógica de la base de datos (SQLite)
  │   │   ├── tests /           # Pruebas unitarias del proceso principal
  │   │   └── main.js           # Entrada principal (ESM)
  │   ├── preload/              # Preload scripts (IPC seguro)
  │   └── renderer/             # Aplicación React (Vista + lógica)
  │       ├── assets/           # Recursos estáticos (imágenes, estilos, etc.)
  │       ├── builders/         # Builders para construir entidades
  │       ├── components/       # Componentes React (Vista)
  │       ├── contexts/         # Contextos de React (Estado global)
  │       ├── controllers/      # Controladores (MVC)
  │       ├── hooks/            # Hooks personalizados de React
  │       ├── models/           # Modelos de dominio
  │       ├── services/         # Lógica de negocio y persistencia
  │       ├── tests/            # Pruebas unitarias (Vitest)
  │       │   ├── components/   # Pruebas de componentes
  │       │   ├── services/     # Pruebas de componentes de React
  │       │   └── utils/        # Pruebas de utilidades
  │       ├── utils/            # Utilidades y helpers
  │       ├── index.html        # Plantilla HTML principal
  │       └── main.jsx          # Entrada de la aplicación (React)
  ├── .gitignore                # Archivos y carpetas ignorados por Git
  ├── .prettierrc               # Configuración de Prettier
  ├── eslint.config.js          # Configuración de ESLint
  ├── package.json              # Configuración del proyecto
  ├── vite.config.js            # Configuración de Vite (ESM)
  └── vitest.config.mjs         # Configuración de Vitest (ESM)
```