Informe del Proyecto - Parcial Browntastic

1. Introducción

Este documento describe el desarrollo del proyecto Browntastic, explicando la arquitectura utilizada, la estructura del código y la implementación de la Programación Orientada a Objetos (POO).

2. Objetivo del Proyecto

El objetivo de este proyecto es desarrollar una aplicación en Flutter para gestionar artículos de brownies, permitiendo a los usuarios marcar artículos como favoritos y almacenarlos localmente en SQLite para su persistencia, sincronizándolos con una base de datos en PostgreSQL.

3. Estructura del Código

3.1 Backend (Node.js con Express y PostgreSQL)

El backend del proyecto está desarrollado en Node.js utilizando Express y PostgreSQL con Sequelize como ORM. A continuación, se describen las carpetas y archivos clave.

Carpetas y archivos del Backend:

• `models/`: Contiene los modelos de Sequelize para definir la estructura de las tablas.
• `routes/`: Define las rutas de la API para gestionar usuarios y artículos.
• `controllers/`: Contiene la lógica para manejar las solicitudes HTTP.
• `config/`: Incluye la configuración de la base de datos y las credenciales de conexión.
• `server.js`: Archivo principal que inicializa el servidor Express.

3.2 Frontend (Flutter)

El frontend está desarrollado en Flutter utilizando el patrón Provider para la gestión del estado y SQLite para el almacenamiento local.

Carpetas y archivos del Frontend:

• `lib/screens/`: Contiene las pantallas principales de la app (login, home, articles).
• `lib/providers/`: Define los proveedores de estado como `FavoritesProvider`.
• `lib/database/`: Implementa la lógica de persistencia local con SQLite.
• `lib/services/`: Gestiona las peticiones HTTP al backend.
• `main.dart`: Punto de entrada de la aplicación.

4. Uso de la Programación Orientada a Objetos (POO)

El proyecto hace un uso efectivo de los principios de POO mediante:

• Encapsulación : Se encapsulan los datos dentro de modelos como `Article` y `User`.

• Herencia : Se utilizan clases en Dart que extienden `ChangeNotifier` para la gestión del estado.

• Abstracción : Se separa la lógica de negocio en controladores y servicios.

• Polimorfismo : Se utilizan métodos en `DbHelper` para acceder a la base de datos de forma flexible.

5. Conclusión

El proyecto Browntastic integra un backend robusto con PostgreSQL y Express y un frontend en Flutter con SQLite para almacenamiento local. Se aplicaron principios de POO y buenas prácticas de desarrollo para garantizar un código modular y mantenible.