Arquitectura DDD (Domain-Driven Design)

Domain

Esta capa es el corazón de la aplicación y contiene la lógica de negocio más importante. En una arquitectura DDD, el **dominio** se encarga de representar conceptos del mundo real relacionados con el negocio. Esta capa está compuesta por:

- **Entities**: Representan los objetos del dominio con identidad propia, que pueden cambiar con el tiempo.
- **Value Objects**: Objetos que no tienen identidad propia y son inmutables.
- **Use Cases**: Lógica de negocio que define cómo las entidades interactúan entre sí.

Infrastructure

La capa de **infraestructura** se encarga de las implementaciones concretas de interfaces definidas en el dominio, tales como repositorios y fuentes de datos. Aquí se encuentran las clases que permiten la persistencia de datos, la integración con servicios externos, entre otros.

- **Repositories**: Implementaciones concretas de los repositorios que interactúan con los datos, ya sea a través de APIs, bases de datos locales, etc.
- **Data Sources**: Abstracciones y clases concretas para la obtención de datos desde fuentes externas, como APIs o bases de datos.

Presentation

La capa de **presentación** se encarga de manejar la interfaz de usuario y la lógica de interacción. Aquí se utilizan Providers para gestionar el estado de la aplicación y propagar los cambios a los widgets.

Uso de Provider para la Gestión de Estados

El paquete Provider es utilizado para la gestión de estados en Flutter debido a su simplicidad y capacidad para escalar en aplicaciones más complejas. Al utilizar Provider, se asegura que los datos y el estado se manejen de manera eficiente y reactiva, permitiendo que los cambios en los datos se reflejen automáticamente en la UI sin necesidad de un manejo manual del estado.

Fuente de Inspiración para el Diseño

El diseño de la interfaz de usuario fue inspirado por una página financiera que ofrece un panel de control para la gestión de transacciones y balances. Aunque el diseño no es una réplica exacta, se han tomado las ideas centrales para crear una interfaz funcional y visualmente atractiva, adaptada a las necesidades específicas de esta aplicación.

Conclusión

La elección de una arquitectura DDD en combinación con Provider permite una clara separación de responsabilidades, facilitando el mantenimiento y la escalabilidad de la aplicación. Además, la inspiración del diseño en una solución financiera real asegura que la interfaz sea intuitiva para los usuarios que manejan transacciones y balances financieros.