Arquitectura General

En el desarrollo de aplicaciones móviles, uno de los retos más comunes que enfrentan los equipos es la **desorganización del código** cuando el proyecto comienza a crecer. Esto dificulta el mantenimiento, las pruebas y la incorporación de nuevas funcionalidades.

En nuestro análisis inicial detectamos que muchos proyectos dependen en exceso de frameworks o librerías, lo que termina generando rigidez y alto costo en cambios futuros. Para resolver este problema, decidimos adoptar el uso de **la arquitectura CLEAN**.

A continuación, se detallan las principales ventajas que justifican su uso:

1. **Separación de responsabilidades**
   Cada capa de la arquitectura tiene un propósito claramente definido, lo que evita la mezcla desorganizada de lógicas. Esta división facilita la identificación de problemas y la comprensión del flujo del sistema.
2. **Facilidad para realizar pruebas**
   La lógica de negocio se encuentra aislada del resto de componentes, lo que permite la creación de pruebas unitarias e integradas sin depender de la interfaz gráfica o de la base de datos. Como resultado, se obtiene un desarrollo más seguro y eficiente.
3. **Escalabilidad y mantenimiento**
   CLEAN Architecture ofrece una estructura flexible que facilita la incorporación de nuevas funcionalidades sin afectar la estabilidad de las ya existentes. Además, en equipos de trabajo grandes, posibilita que varios desarrolladores colaboren de manera paralela en distintas capas, usamos CLEAN y no MVC porque el primero evita el típico “Massive ViewController” evitando un cuello de botella. Igualmente no optamos por VIPER, que también separa capas pero tiende a ser verboso y con muchos archivos por funcionalidad, puede generar sobrecarga o confusión en equipos pequeños.
4. **Independencia de frameworks**
   La aplicación no queda ligada a una librería, API o SDK en particular.
5. **Legibilidad y colaboración en equipo**
   El código estructurado en capas definidas sin redundancia mejora su legibilidad, permitiendo la rápida incorporación de nuevos miembros al equipo de trabajo . Esto

se traduce en un aumento de la productividad y en una mayor calidad del trabajo colaborativo.

Por otro lado, y siguiendo una línea de trabajo cuyo fin es la organización óptima del código optamos como patrón de diseño principal **Model–View–ViewModel (MVVM).**

A continuación, se presentan las principales ventajas que justifican su implementación:

1. **Separación de responsabilidades**
   MVVM divide el código en tres componentes principales:
   - **Model:** Encargado de la lógica de negocio y la gestión de datos.
   - **ViewModel:** Procesa y transforma la información del modelo para que la vista pueda mostrarla.
   - **View:** Responsable únicamente de la interfaz gráfica.
     Esta división mejora la organización del proyecto y evita dependencias innecesarias.
2. **Facilidad de pruebas**
   Al tener la lógica encapsulada en el ViewModel, es posible realizar pruebas unitarias sin necesidad de cargar la interfaz gráfica. Esto permite detectar errores de forma temprana y aumentar la calidad del software.
3. **Reducción del acoplamiento**
   La vista no se comunica directamente con el modelo, sino a través del ViewModel. Gracias a esta independencia, se pueden modificar o reemplazar las interfaces gráficas sin alterar la lógica de negocio. A diferencia de MVP, que requiere mucha comunicación Presenter-View, preferimos usar data binding para que la UI cambie automáticamente a cambios de estado.
4. **Mantenimiento y escalabilidad**
   En proyectos en crecimiento, como es nuestro caso, MVVM ofrece una estructura clara que facilita la incorporación de nuevas funcionalidades. Asimismo, permite que varios desarrolladores trabajen de manera paralela sin afectar otras partes del sistema.

Herramientas y tecnologías:

En el desarrollo de la aplicación se ha optado por un conjunto de herramientas y tecnologías modernas que permiten optimizar tiempos y garantizar la calidad del producto final.

- Desarrollo de iOS

  Se utiliza el lenguaje **Swift** junto con el framework **SwiftUI**, una combinación que permite desarrollar aplicaciones con un enfoque actual, seguro y eficiente.

  - **Lenguaje moderno:** Swift incorpora características avanzadas como *type safety*, *optionals* e *inferencia de tipos*, lo que contribuye a la estabilidad y robustez del código.
  - **Formato declarativo:** A diferencia de UIKit, donde se define paso a paso cómo construir la interfaz, SwiftUI adopta un enfoque declarativo en el que se describe **qué** se quiere mostrar, favoreciendo la simplicidad y la legibilidad del código.

  Por otro lado, el uso de **macros en lugar de wrappers** se traduce en un desarrollo más **eficiente, claro y alineado con la visión declarativa de SwiftUI**. Adoptar esta práctica en proyectos con iOS 17 garantiza mantener la aplicación al día con las recomendaciones de Apple, optimizando tanto el tiempo de desarrollo como la calidad del código.

  En cuanto a **reducción de tiempos de desarrollo**, SwiftUI introduce ventajas significativas frente a UIKit, entre ellas:

  - **Vista previa en tiempo real**, que permite validar cambios sin necesidad de compilar.
  - **Menor número de líneas de código**, lo que agiliza la implementación de interfaces.

  Esto se traduce en un ciclo de entrega más corto y en una mayor capacidad de iterar sobre el diseño y la experiencia de usuario.

- Desarrollo de Android
- Backend
  - Vapor
- Base de Datos

    - Sqlite ? si la usamos es que es muy veloz a la hora de trabajar en local para poder crear y eliminar datos fácilmente. No maneja grandes cantidades de información pero para el MVP no será necesario,
- Control de Versiones y colaboración
- Diseño UX/UI