Flutter Bootcamp

Clean Architecture

Emanuel López – emanuel.lopez@globant.com

Globant

Repaso

- Estado efímero y Estado de la aplicación
- `InheritedWidget`
- `Provider`
- Patrón de diseño Observer
- `ChangeNotifier` y `ChangeNotifierProvider`
- `StreamController` y `Stream`

Repositorio del Bootcamp: bit.ly/FlutterBootcampGlobant

Globant

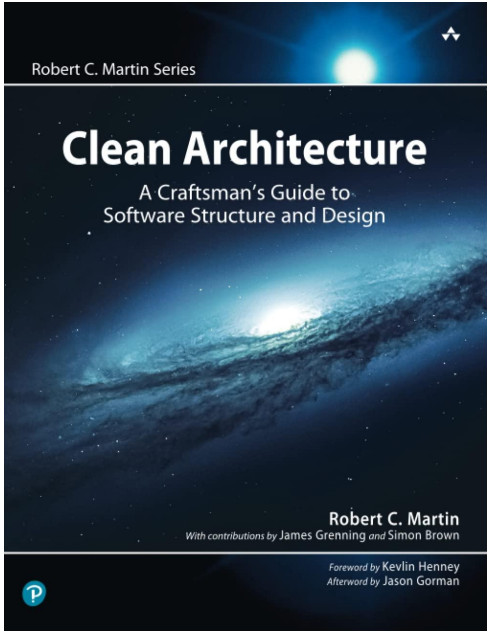
Clean Architecture

Robert C. Martin presentó por primera vez el concepto de Clean Architecture en una serie de conferencias y charlas en la década de 2010.

Uno de los artículos más influyentes sobre el tema es su publicación "The Clean Architecture", que fue publicado en el blog de Uncle Bob en 2012.

Desde entonces, el concepto ha ganado popularidad y se ha convertido en un enfoque preferido para desarrolladores y equipos que buscan crear software más mantenible y escalable.

La idea es separar las preocupaciones en diferentes capas bien definidas, con reglas estrictas sobre cómo deben interactuar entre sí.



Globant

Clean Architecture

El objetivo con esta arquitectura es lograr:

- **Independencia del Framework:** La arquitectura no debe depender de herramientas o frameworks específicos.
- **Independencia de la UI:** El sistema puede cambiar la interfaz de usuario sin afectar la lógica de negocio.
- **Testeabilidad:** La lógica de negocio debe ser fácilmente testeable sin interfaces de usuario, bases de datos u otros componentes externos.
- **Independencia de la base de datos:** La lógica de la aplicación no debe depender de detalles específicos de almacenamiento.

Clean Architecture

Las capas de una arquitectura limpia son:

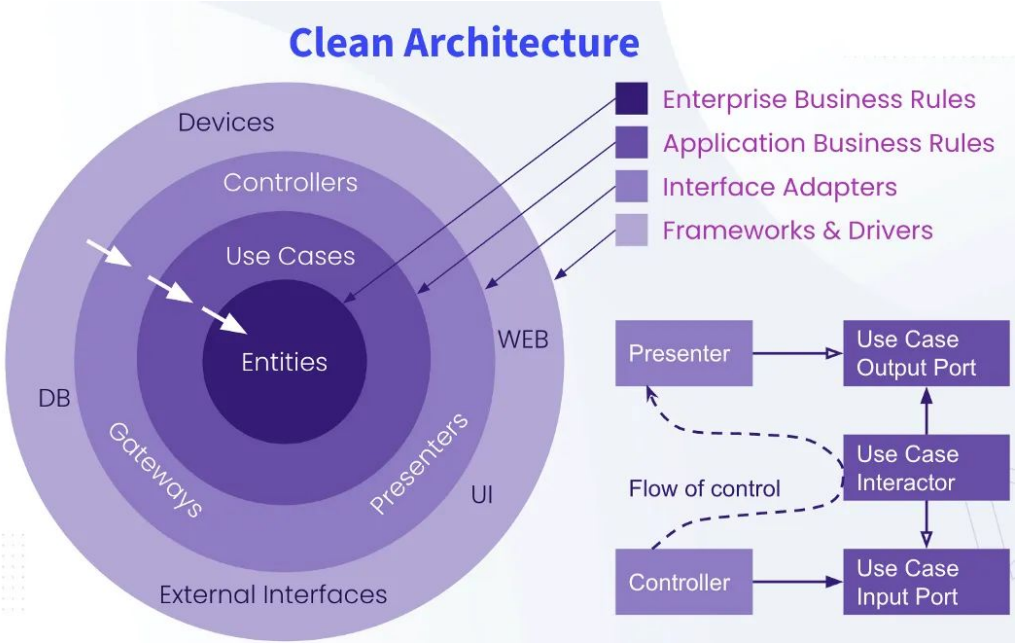
- **Dominio (Entities y Use Cases):** Representa el núcleo de la aplicación, donde se encuentran las reglas de negocio. Aquí es donde colocamos las entidades y casos de uso.
- **Datos (Frameworks & Drivers):** Aquí se manejan las interacciones con la base de datos, APIs externas o cualquier otra fuente de datos. Esta capa es donde implementamos los repositorios y las fuentes de datos.
- **Presentación (Interface Adapters):** En esta capa está la lógica que conecta la UI con el dominio. Se encarga de adaptar los datos que provienen del dominio para que puedan ser representados correctamente en la interfaz de usuario.

Clean Architecture

Las dependencias siempre fluyen hacia adentro, desde las capas externas hacia las internas.

Las capas externas conocen las internas, pero las internas son completamente independientes de las externas.

Esto permite que el núcleo de la aplicación sea robusto y fácil de modificar, mientras que las capas externas pueden cambiar sin afectar al núcleo.



Globant

WidgetsFlutterBinding.ensureInitialized()

Se utiliza para asegurarnos que los servicios del framework de Flutter estén completamente inicializados antes de ejecutar cualquier código que dependa de ellos.

Se suele usar al inicio del `main()` cuando necesitamos ejecutar código que interactúa con cosas como:

- Plugins de Flutter: Muchos plugins necesitan que el motor de Flutter esté completamente inicializado antes de usarlos.
- Operaciones asíncronas iniciales: Si necesitas ejecutar código asíncrono (por ejemplo, inicializar Firebase o cargar configuraciones) antes de que la aplicación se inicie.

Si intentamos realizar operaciones dependientes del framework de Flutter antes que este esté completamente inicializado, podríamos encontrarnos con errores inesperados o comportamientos indefinidos.

WidgetsBinding.instance.addPostFrameCallback()

Se utiliza para ejecutar un callback justo después de que el árbol de widgets haya sido completamente construido y renderizado en la pantalla.

A veces, necesitamos realizar ciertas acciones después de que se haya completado el renderizado inicial de la interfaz de usuario, por ejemplo:

- Mostrar un `SnackBar` o un `Dialog` justo después de que se haya mostrado una pantalla.
- Realizar cálculos basados en el tamaño de los widgets que ya están renderizados.
- Navegar a otra pantalla inmediatamente después de que la pantalla actual se haya construido.

A veces intentar hacer estas acciones dentro de los métodos `build` o `initState` puede causar problemas, ya que estos métodos pueden ejecutarse antes de que el widget esté completamente renderizado.

`addPostFrameCallback` asegura que el código dentro del callback se ejecute después de que se haya completado el renderizado inicial.

WidgetsBindingObserver

Se utiliza para observar y reaccionar a ciertos cambios en el estado del sistema o en el ciclo de vida de la aplicación, ej: cuando la aplicación pasa a estar en segundo plano, en primer plano, o cuando cambia la orientación de la pantalla, etc.

Este mixin es especialmente útil cuando necesitas escuchar estos eventos y actuar en consecuencia, ej:

- Pausar la reproducción de un video cuando la app va a segundo plano.
- Realizar un ajuste cuando cambia la orientación.

Proporciona un conjunto de métodos que podemos sobrescribir en nuestra clase y serán ejecutados automáticamente cuando ocurran estos eventos.

Barrel files

Un barrel file es simplemente un archivo Dart que utiliza la declaración `export` para agrupar otros archivos.

- **Simplicidad en las importaciones:** En lugar de importar cada archivo individualmente, importas solo el barrel file. Esto reduce la cantidad de declaraciones import en tu código.
- **Mejor organización:** Permite agrupar archivos relacionados, lo que mejora la estructura y el mantenimiento del código, sobre todo en proyectos grandes.
- **Facilita la refactorización:** Si cambias la ubicación de algunos archivos, solo necesitas actualizar el barrel file en lugar de modificar todas las importaciones en tu proyecto.

shared_preferences

El paquete **`shared_preferences`** se utiliza en Flutter para almacenar datos pequeños y simples de forma local en el dispositivo del usuario, utilizando un sistema de almacenamiento **clave-valor**. Es ideal para guardar datos no requieren una base de datos completa, como:

- Guardar las preferencias de usuario (por ejemplo, el tema oscuro o claro).
- Almacenar configuraciones de la app (idioma, opciones de notificación, etc.).
- Mantener el estado de inicio de sesión (un token de autenticación simple).
- Guardar información temporal que persiste entre sesiones de la app.

Características:

- Tipos de datos compatibles: **`int`**, **`double`**, **`bool`**, **`String`** y **`List<String>`**
- Persistencia simple: Los datos almacenados en **`shared_preferences`** permanecen incluso cuando la app se cierra y se vuelve a abrir.

Mas Información: pub.dev/packages/shared_preferences

Globant

http

El paquete `http` se utiliza en Dart y Flutter para realizar solicitudes HTTP y comunicarse con servidores web. Es una herramienta esencial para interactuar con APIs o servicios web RESTful, permitiéndonos enviar y recibir datos a través de la red.

Funciones principales del paquete http:

- Realizar solicitudes `HTTP` como `GET`, `POST`, `PUT`, `DELETE`, entre otras.
- Enviar datos al servidor (normalmente en formato `JSON`) y recibir respuestas.
- Autenticación mediante encabezados `HTTP` como `Authorization` para enviar tokens de acceso o credenciales.
- Manejo de respuestas en formato `JSON`, texto o binario.

Mas información: pub.dev/packages/http

dio

`dio` es un paquete para Dart y Flutter más avanzado y poderoso que `http`. Ofrece una amplia gama de funcionalidades realizar peticiones `HTTP` de manera más flexible y robusta. Es útil cuando nuestra aplicación requiere características avanzadas que el paquete `http` no ofrece de manera nativa.

Algunas características distintivas de dio son:

- **Interceptores:** Permite agregar interceptores para modificar las solicitudes o respuestas antes de que se envíen o reciban. Útil para agregar autenticación, registros, manejo de errores, etc.
- **Manejo de cancelaciones:** Podemos cancelar solicitudes en curso, algo que es difícil de hacer con el paquete `http`.
- **Manejo automático de errores:** Nos permite manejar errores y excepciones de manera más organizada.

Mas Información: pub.dev/packages/dio

Otras librerías utilizadas

- `cached_network_image`: Se utiliza para mostrar imágenes desde la web y almacenarlas en caché. La imagen descargada se guarda localmente en el dispositivo, y las futuras solicitudes para mostrar esa imagen no requerirán volver a descargarla, lo que mejora el rendimiento y la eficiencia de la aplicación.
- `flutter_launcher_icons`: Se utiliza para generar automáticamente los íconos de la aplicación en diferentes tamaños y formatos que se requieren en distintas plataformas, como Android e iOS. Este paquete facilita la tarea al crear todos los íconos necesarios con un solo comando, ahorrando tiempo y esfuerzo.
- `intl`: Se utiliza para manejar la internacionalización (i18n) y la localización (l10n) de aplicaciones. Ofrece una serie de utilidades para trabajar con formato de fechas, números, monedas y textos traducidos según la región y el idioma del usuario.

Otras librerías utilizadas

- `sqflite`: se utiliza para gestionar una base de datos SQLite local en dispositivos móviles.
- `path`: se utiliza para manejar y manipular rutas de archivos de forma segura y conveniente. Proporciona funciones que simplifican la tarea de trabajar con rutas de archivos, como construir rutas a partir de partes, unir rutas, obtener extensiones de archivos y más.
- `lottie`: Se utiliza para mostrar animaciones vectoriales en formato JSON, que han sido creadas con la herramienta **Lottie** de Airbnb. Estas animaciones están basadas en vectores y se pueden integrar fácilmente en aplicaciones Flutter, ofreciendo una forma de agregar animaciones complejas sin necesidad de usar imágenes de mapa de bits o animaciones pre-renderizadas.

Tarea

Desarrollar una aplicación de tipo **Shopping Cart**. Los productos a la venta se pueden obtener de APIs como las siguientes (no requieren autenticación):

- fakeapi.platzi.com/ (`https://api.escuelajs.co/api/v1/products`)
- fakestoreapi.com/docs (`https://fakestoreapi.com/products`)

La aplicación debe tener un mínimo de 3 pantallas:

1. **Listado de productos.** obtenidos de la API. (no es necesario implementar paginación)
2. `Detalle del producto.` con título, descripción, imagen y precio. Opción para agregar al carrito. (con mostrar el producto obtenido la pantalla de listado es suficiente)
3. **Checkout o finalización del pedido.** donde se listen todos los productos y se muestre el total del pedido.

Links adicionales

- Clean Architecture: blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- Clean Architecture Book: .amazon.com/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164
- Clean Architecture Talk: vimeo.com/97530863

Preguntas?

Globant

Gracias!

Globant