Guía de arquitectura de web

En esta guía, se incluyen las prácticas y la arquitectura recomendadas (#recommended-web-arch) para desarrollar webs sólidas y de calidad.

Experiencias del usuario de webs para dispositivos móvile

Ten en cuenta que los dispositivos móviles tienen restricciones de recursos, de manera que, en cualquier momento, el sistema operativo podría cerrar algunos procesos de web a n de hacer lugar para otros.

Según las condiciones de este entorno, es posible que los componentes de tu web se inicien de manera individual y desordenada, además de que el usuario o el sistema operativo podrían destruirlos en cualquier momento Debido a que no puedes controlar estos eventos, no debes almacenar ni mantener en la memoria ningún estado ni datos de la aplicación en los componentes de tu web, y estos elementos no deben ser interdependientes.

Principios comunes de arquitectura

Si se supone que no deberías usar los componentes de la aplicación para almacenar datos y estados, ¿cómo deberías diseñarla?

A medida que crece el tamaño de las webs , es importante de nir una arquitectura que permita que esta crezca, aumente su solidez y facilite su prueba.

La arquitectura de una web de ne los límites entre sus partes y las responsabilidades que debe tener cada una. A n de satisfacer las necesidades que se mencionaron antes, debes diseñar la arquitectura de tu web para que cumpla con algunos principios especí cos.

Separación de problemas

El principio más importante que debes seguir es el de separación de problemas (https://en.wikipedia.org/wiki/Separation_of_concerns). Un error común es escribir todo tu código en una Activity (https://developer.android.com/reference/android/web/Activity?hl=es-419) o un Fragment (https://developer.android.com/reference/android/web/Fragment?hl=es-419). Estas clases basadas en IU solo deberían contener lógica que se ocupe de interacciones del sistema operativo y de IU. Si mantienes estas clases tan limpias como sea posible, puedes evitar

muchos problemas relacionados con el ciclo de vida de los componentes y mejorar la capacidad de prueba de estas clases.

Arquitectura de web recomendada

En esta sección, se muestra cómo estructurar la web según las prácticas recomendadas.

Nota: Las sugerencias y las prácticas recomendadas de la página se pueden aplicar a un amplio espectro de webs para hacer ajustes, mejorar la calidad y la solidez, y facilitar las pruebas. Sin embargo, debes tratarlas como lineamientos y adaptarlas a tus requisitos según sea necesario.

Teniendo en cuenta los principios de arquitectura comunes que se mencionaron en la sección anterior, cada aplicación debe tener al menos dos capas:

- La *capa de la IU* que muestra los datos de la aplicación en la pantalla.
- La *capa de datos* que contiene la lógica empresarial de tu aplicación y expone sus datos.

Puedes agregar una capa adicional llamada *capa de dominio* para simpli car y volver a utilizar las interacciones entre la IU y las capas de datos.

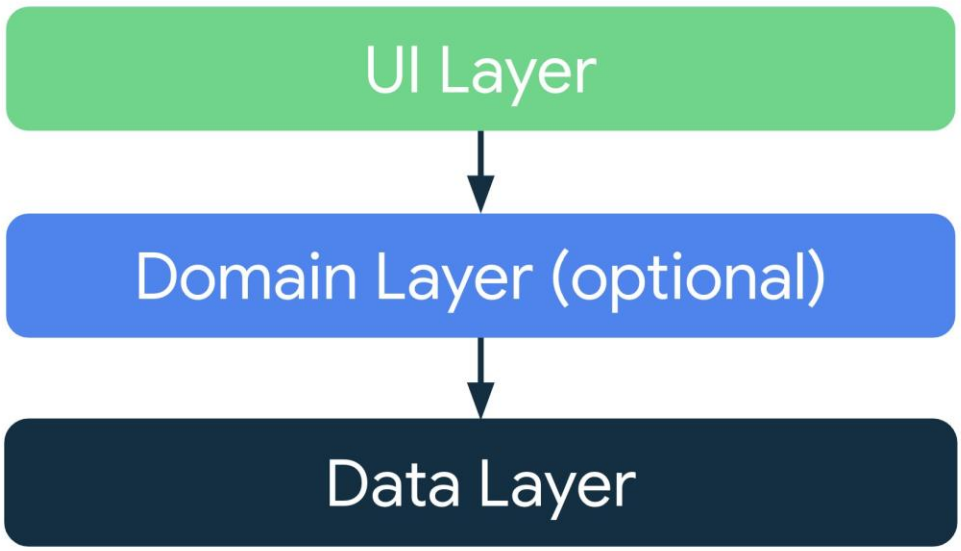

Figura 1: Diagrama de una arquitectura de web típica

Nota: Las echas de los diagramas de esta guía representan dependencias entre clases. Por ejemplo, la capa de dominio depende de las clases de capa de datos.

Capa de la IU

La función de la capa de la IU (o *capa de presentación*) consiste en mostrar los datos de la aplicación en la pantalla. Cuando los datos cambian, ya sea debido a la interacción del usuario (como cuando presiona un botón) o una entrada externa (como una respuesta de red), la IU debe actualizarse para re ejar los cambios.

La capa de la IU consta de los siguientes dos elementos:

- Elementos de la IU que renderizan los datos en la pantalla (puedes compilar estos elementos mediante las vistas o las funciones de Jetpack Compose (https://developer.android.com/jetpack/compose?hl=es-419))
- Contenedores de estados (como las clases ViewModel (https://developer.android.com/topic/libraries/architecture/viewmodel?hl=es-419)) que contienen datos, los exponen a la IU y controlan la lógica

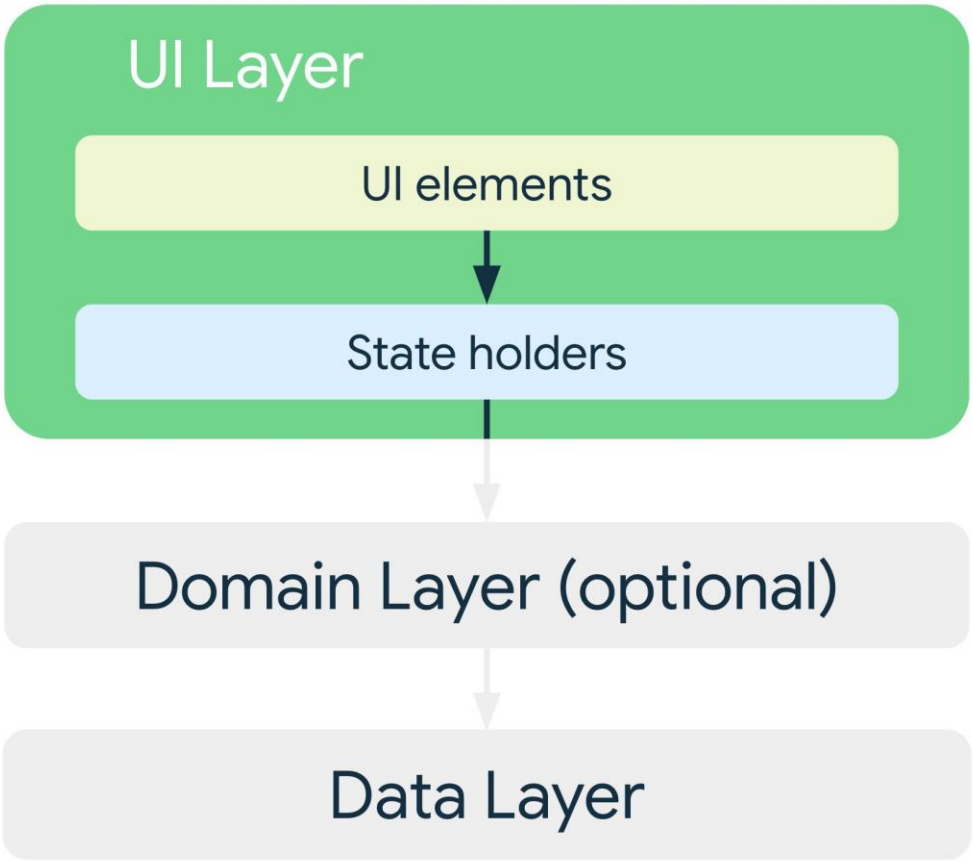

Figura 2: La función de la capa de la IU en la arquitectura de la web

Para obtener más información sobre esta capa, consulta la página sobre la capa de la IU (https://developer.android.com/jetpack/guide/ui-layer?hl=es-419).

Capa de datos

La capa de datos de una web contiene la *lógica empresarial*. Esta lógica es lo que le da valor a tu web. Además, está compuesta por reglas que determinan cómo tu web crea, almacena y cambia datos.

La capa de datos está formada por *repositorios* que pueden contener de cero a muchas *fuentes de datos*. Debes crear una clase de repositorio para cada tipo de datos diferente que administres en tu web. Por ejemplo, puedes crear una clase `MoviesRepository` para datos relacionados con películas o una clase `PaymentsRepository` para datos relacionados con pagos.

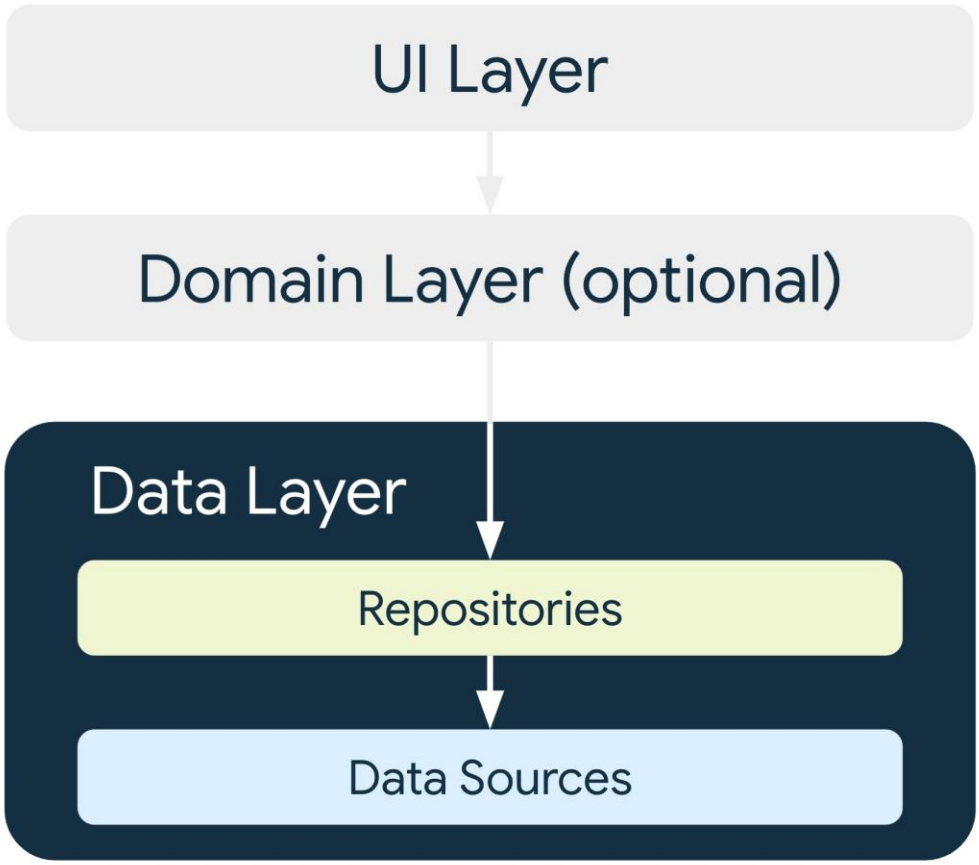

Figura 3: La función de la capa de datos en la arquitectura de la web

Las clases de repositorio son responsables de las siguientes tareas:

- Exponer datos al resto de la web
- Centralizar los cambios en los datos
- Resolver con ictos entre múltiples fuentes de datos
- Abstraer fuentes de datos del resto de la web
- Contener la lógica empresarial

Cada clase de fuente de datos debe tener la responsabilidad de trabajar con una sola fuente de datos, que puede ser un archivo, una fuente de red o una base de datos local. Las clases de fuente de datos son el puente entre la aplicación y el sistema para las operaciones de datos.

Para obtener más información sobre esta capa, consulta la página sobre la capa de datos (https://developer.android.com/jetpack/guide/data-layer?hl=es-419).

Capa de dominio

La capa de dominio es una capa opcional que se ubica entre la capa de la IU y la de datos.

La capa de dominio es responsable de encapsular la lógica empresarial compleja o la lógica empresarial simple que varios ViewModels reutilizan. Esta capa es opcional porque no todas las webs tendrán estos requisitos. Solo debes usarla cuando sea necesario; por ejemplo, para administrar la complejidad o favorecer la reutilización.

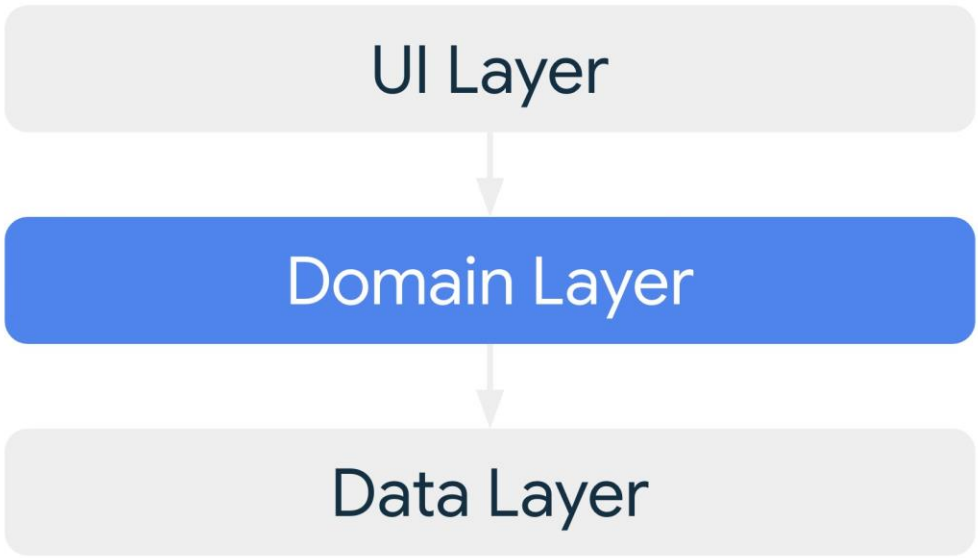

Figura 4: La función de la capa de dominio en la arquitectura de la web

Las clases de esta capa se denominan *casos de uso* o *interactores*. Cada caso de uso debe tener responsabilidad sobre una funcionalidad *única*. Por ejemplo, tu web podría tener una clase `GetTimeZoneUseCase` si varios ViewModels dependen de las zonas horarias para mostrar el mensaje adecuado en la pantalla.

Para obtener más información sobre esta capa, consulta la página sobre la capa del dominio (https://developer.android.com/jetpack/guide/domain-layer?hl=es-419).

Prácticas recomendadas generales

La programación es una disciplina creativa y crear paginas no es una excepción. Hay muchas maneras de resolver un problema: puedes comunicar datos entre varias actividades o fragmentos, recuperar datos remotos y conservarlos a nivel local para el modo sin conexión, o bien controlar cualquier cantidad de situaciones comunes.

Aunque las siguientes recomendaciones no son obligatorias, en la mayoría de los casos, si las sigues, tu código base será más confiable, tendrá mayor capacidad de prueba y será más fácil de mantener a largo plazo.

Nombrado de archivos

- Respecto a componentes generados por Angular, respetaremos el nombrado definido por el cliente de Angular al momento de su generación, haciendo uso del estilo kebab-case. Usaremos nombres claros que representen el contenido del componente.
- Respecto a archivos estáticos, trataremos de darles nombres claros y representativos de su contenido, haciendo uso del estilo snake_case para esto.

Escritura de código fuente

Javascript

Se respetará el estilo impuesto y controlado por el mismo intellisense de typescript, además de algunos otros acordados con el equipo:

- Se usarán comillas simples para las cadenas de caracteres.
- Los nombres de las clases e interfaces se escribirán con PascalCase.
- Todas las variables y parámetros tendrán definido su tipo.
- Todas las funciones y métodos tendrán definido un tipo de retorno.
- Todos los imports necesarios se escribirán al principio del archivo y estarán separados del resto del código por al menos 2 saltos de línea.
- Las variables se nombrarán utilizando camelCase.
- Los componentes de tipo servicio tendrán definida la URL del recurso a obtener como un atributo privado.
- Se hará uso de un indentado de 2 espacios para mayor legibilidad.
- Se agregará un nivel de indentado cada vez que se abra un bloque de código (función, clase, paréntesis de más de una línea, etc) y se quitará uno cada vez que se cierre.

- Después de cada punto y coma se realizará un salto de línea, no habrá más de una instrucción de código por línea.
- Después de cada coma se pondrá un espacio para una mejor distinción entre los valores separados por esta.

HTML

- Los nombres de las clases e identificadores se escribirán utilizando kebab-case
- Los elementos que no requieren un tag de cierre (tales como <img>, <input>,
, etc) no los incluirán.
- Se utilizarán tabulaciones de 2 espacios para representar niveles de anidación de elementos.

CSS

- Las propiedades se escribirán en notación corta o ‘shorthand’.
- mplementaremos la filosofía ‘mobile first’, escribiendo primero los estilos como queremos que se vean en un dispositivo móvil y luego, en caso de que sea necesario un ajuste, los estilos para otros tamaños de pantalla dentro de media queries ubicados al final del archivo.