**INSTITUTO TECNOLÓGICO DEL VALLE DE OAXACA**

**“Arquitecturas de software para Móviles”**

MATERIA:

Aplicaciones Móviles II

PRESENTA:

Luis Enrique Hernández Hernández

NÚMERO DE CONTROL:

16920360

CATEDRATICO:

I.S.C. Kevin David Molina Gómez

GRADO:

Verano

GRUPO:

Verano

INGENIERÍA INFORMÁTICA

FECHA DE ENTREGA: martes, 3 de agosto de 2021, 00:00

**ÍNDICE**

1------------------------------------------------------------------------------------------------------------Introducción.

2---------------------------------------------------------------------------------------------------------------Desarrollo.

2.1-------------------------------------------------------------------------------------------------Arquitectura MVVM

2.2------------------------------------------------------------------------------------------------Arquitectura CLEAN

2.3-------------------------------------------------------------------------------------------------Arquitectura VIPER

3---------------------------------------------------------------------------------------------------------------Conclusión

4-------------------------------------------------------------------------------------------------------------------Fuentes

1. **INTRODUCCIÓN**

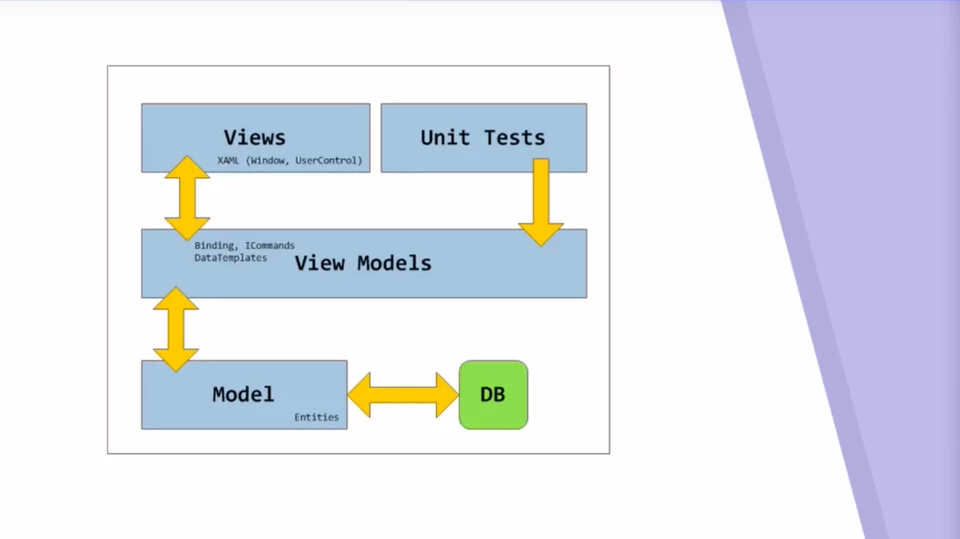
El desarrollo de software para las diferentes plataformas se ha convertido en estos últimos años en una parte importante dentro del sector del desarrollo de software. La velocidad a la que ha crecido la complejidad del software a desarrollar, ha provocado que los modelos de arquitectura utilizados hasta la fecha hayan quedado obsoletos para suplir las necesidades de un desarrollo de larga duración. En este trabajo se estudiarán y documentarán los modelos de arquitectura existentes para el desarrollo de aplicaciones tanto para IOS como Android. Tras ello, se utilizarán patrones de diseño de software para diseñar y documentar una propuesta de arquitectura con la que tratar de garantizar la sostenibilidad de un proyecto de software.

1. **DESARROLLO** 
   1. **Arquitectura MVVM**

El patrón modelo–vista–modelo de vista (en inglés, model–view–viewmodel, abreviado MVVM) es un patrón de arquitectura de software. Se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.

Esta arquitectura ha sido adoptada después por otros lenguajes y otras tecnologías, como pueden ser Java con Android o iOS con Apple, y resulta muy potente.

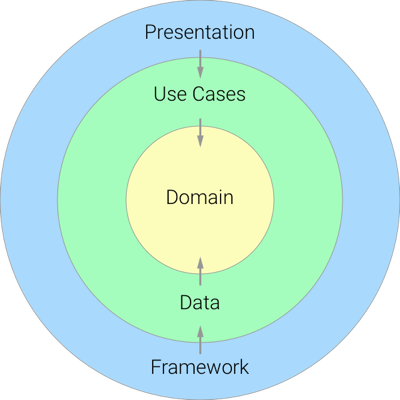
Podemos ver aquí el esquema inicial de MVVM, el esquema clásico, el cual lo que prima es el desacoplamiento de toda la parte visual, toda la parte de lógica de negocio y todo la parte que va a ir relacionada con nuestra entidad.

****

* La parte del modelo (Model), en la cual vamos a ver todo lo que serían los datos, dónde vamos a tener toda la lógica de datos. Representa la capa de datos y/o la lógica de negocio, también denominado como el objeto del dominio. El modelo contiene la información, pero nunca las acciones o servicios que la manipulan. En ningún caso tiene dependencia alguna con la vista.
* La vista modelo (View Models) que va a ser la encargada de interactuar tanto con el modelo como con la vista. El modelo de vista es un actor intermediario entre el modelo y la vista, contiene toda la lógica de presentación y se comporta como una abstracción de la interfaz. La comunicación entre la vista y el modelo de vista se realiza por medio los enlaces de datos.
* La vista (Views) que va a ser la parte visual. La misión de la vista es representar la información a través de los elementos visuales que la componen. Las vistas en MVVM son activas, contienen comportamientos, eventos y enlaces a datos que, en cierta manera, necesitan tener conocimiento del modelo subyacente.
  1. **Arquitectura CLEAN**

si no una serie de guías y buenas prácticas en el desarrollo de software. Fue definida por Rober C Martin (Uncle Bob) en su charla “Architecture the lost years”, donde exponía una serie de problemas y el alto acoplamiento de los desarrollos de software tanto a los modelos de datos como a la interfaz.

Clean define una serie de capas y otorga una responsabilidad a cada una, pero no entra en profundidad en los detalles de implementación y cómo se deben resolver los problemas.



El objetivo es escribir software que esté lo menos acoplado posible a nuestro modelo de datos, a la representación de este y al framework que estemos usando. Esto va a incrementar la estabilidad de nuestro código ya que va a hacer más fácil cambiar las partes dependientes del sistema. Va a facilitar la portabilidad a otros entornos (como iOS o Web) dado que gran parte del código es independiente del framework. También va a permitir postergar decisiones de implementación, como por ejemplo la persistencia o el uso de red

Dentro de esta arquitectura, existen cuatro capaz las cuales son:

* Capa de Modelo.

También se utiliza el nombre de Dominio o Lógica de Negocio. Está formada por las clases que representan la lógica interna de la aplicación y cómo representamos los datos con los que vamos a trabajar. Muchas clases de esta capa se conocen como POJO (Plain Old Java Object) al tratarse de clases Java puras. Ejemplos de POJO serían las clases Lugar o GeoPunto. No es conveniente que en estas clases se utilicen APIs externos. Si abres las clases que hemos indicado, podrás comprobar que no necesitan ningún import.

* Capa de Datos.

En esta capa estarían las clases encargadas de guardar de forma permanente los datos y cómo acceder a ellos. Suelen representar bases de datos, servicios Web, preferencias, ficheros JSON… También es conocida como capa de almacenamiento o persistencia.

* Capa de Casos de Uso.

Los casos de uso son clases que van a definir las operaciones que el usuario puede realizar con nuestra aplicación. Esta capa no sería estrictamente necesaria (por ejemplo, en Asteroides no la vamos a utilizar), pero resulta muy interesante para tener enumeradas las diferentes acciones que vamos a implementar. Además, va a permitir quitar mucha responsabilidad a las actividades. Los casos de uso también se conocen como interactors.

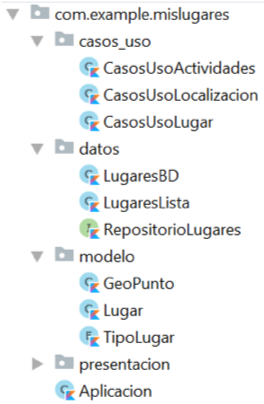
* Capa de Presentación.

Representa la interfaz de usuario, por lo que está formada por las actividades, fragments, vistas y otros elementos con los que interactúa el usuario.

Organizando las clases en paquetes.

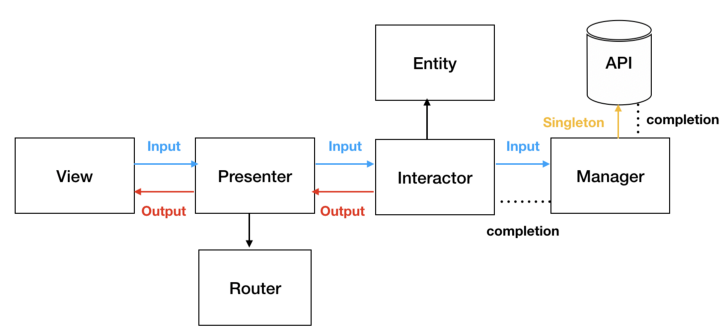
El número de clases de un proyecto Android también puede ser muy elevado, por lo que resulta complicado localizarlas. Para resolver este problema, resulta frecuente organizar las clases en diferentes paquetes. Podemos usar diferentes criterios, por ejemplo, por módulos del proyecto (como autentificación, visualización, mapas…) Otro criterio podría ser por entidades (como lugares, usuarios…) También podemos utilizar la función de la clase (como actividades, fragments, adaptadores…).

En este ejemplo utilizaremos como criterio la capa de la arquitectura (en concreto modelo, datos, casos de uso y presentación). La organización propuesta se muestra debajo. Pero eres libre de usar nombres en inglés o definir tu propio criterio. El paquete donde se encuentra cada clase no afectará a los ejercicios.



* 1. **Arquitectura VIPER**

VIPER es el acrónimo de las palabras View, Interactor, Presenter, Entity y Router, las cuales vienen a ser las diferentes áreas de responsabilidad que constituyen una aplicación móvil para el sistema operativo iOS.



A continuación se describen brevemente cada uno de los componentes de VIPER.

* View. la responsabilidad de la vista es de enviar las acciones del usuario al presentar y enseñar lo que le dice el presenter.
* Interactor. Esto es la columna vertebral de la aplicación, ya que contiene la lógica de negocio descrita por el use cases en la aplicación. El interactor es responsable de atraer data desde la capa de mode, y su implementación es totalmente independiente de la interface del usuario.
* Presenter. Su responsabilidad es de coger data desde el interactor en acciones de usuario, crear un ejemplo de modelo vista y llevarlo hacia el View para mostrarlo.
* Entity. Contiene el modelo básico de objetos usados por el interactor. Tiene parte de responsabilidades de la capa modelo en otras arquitecturas.
* Router. Tiene toda la lógica de navegación para describir lo que la pantalla quiere mostrar y cuando.

La arquitectura Viper tiene muchos beneficios, pero es importante mencionar que es mejor utilizarla en grandes proyectos de mucha complejidad. Debido al número de elementos involucrados, esta arquitectura causa gastos generales cuando se aplica en proyectos pequeños, así que la arquitectura Viper puede ser una medida exagerada para pequeños proyectos que buscan la escalabilidad.

En la arquitectura VIPER, cada bloque corresponde a un objeto con tareas específicas, inputs y outputs. Es muy parecido a trabajadores en una organización en línea: una vez un trabajador completa su tarea en un objeto, este se pasa al siguiente trabajador en la fila hasta que el producto se completa. Las conexiones entre los bloques representa la relación entre los objetos y el tipo de información que se transmiten entre ellos. La comunicación de una entidad hacia otra se transmite a través de los protocols.

La idea detrás de este patrón de arquitectura es separar las dependencias de tu app, balanceando la delegación de responsabilidades entre las entidades. Básicamente la arquitectura Viper divide tu lógica de aplicación en capas de funcionalidad más pequeñas, cada una de ellas con un strict, con una responsabilidad predefinida. Esto hace que testear las interacciones en los límites de las capas sea más fácil. Encaja bastante bien con pruebas unitarias, y hace que tu codigo se pueda reutilizar.

1. **CONCLUSIÓN**

En esta investigación abarcamos tres arquitecturas de desarrollo las cuales son MVVM, CLEAN y VIPER. Cada una de ellas tiene un propósito particular pero eso no signifique que uno sea más importante que otro.

Hoy en día el acceso móvil a los sistemas ya no es algo opcional, sino que es algo ubicuo y es necesario pensar en clientes móviles desde que se comienza a diseñar la aplicación. El acceso móvil es la forma como la gente interactúa hoy en día con computación y, por ello, los usuarios tienen expectativas al respecto, como por ejemplo poder realizar las mismas actividades que se hacen en la máquina de la oficina a través de un teléfono inteligente.

La arquitectura de software juega un papel fundamental en el desarrollo de aplicaciones móviles y creemos que el darle un énfasis mayor al que se está dando actualmente podría aportar grandes beneficios.

Al implementar las arquitecturas todo está más encapsulado, permitiendo con esto que nuestra aplicación sea más escalable, más fácil de leer, de mantener, de probar y por tanto más segura.

**FUENTES**

<https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93modelo_de_vista>

<https://openwebinars.net/blog/la-arquitectura-mvvm-y-sus-componentes/>

<http://www.androidcurso.com/index.php/932>

<https://www.paradigmadigital.com/dev/introduccion-los-componentes-arquitectura-android-go-clean/>

[**https://apiumhub.com/es/tech-blog-barcelona/arquitectura-viper/**](https://apiumhub.com/es/tech-blog-barcelona/arquitectura-viper/)

[**http://diposit.ub.edu/dspace/bitstream/2445/120321/3/memoria.pdf**](http://diposit.ub.edu/dspace/bitstream/2445/120321/3/memoria.pdf)