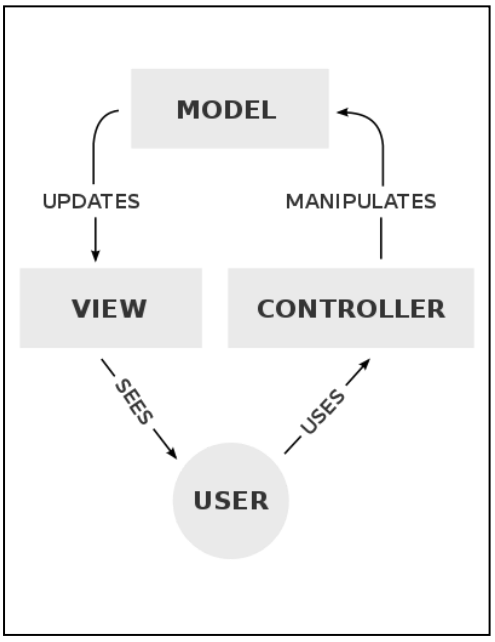
PASO A PASO 9:
MVC: Model View Controller



MVC es una forma de estructurar nuestro código. Se trata de un patrón de arquitectura de software, y divide la aplicación en tres capas interconectadas, favoreciendo particularmente la separación entre la representación interna de la información y la forma en esa misma información se presenta al usuario. De esta forma nos permite mantener los componentes de nuestra aplicación conectados, pero sus responsabilidades separadas.
Esto va a ser particularmente útil cuando sea necesario modificar nuestro código, porque en el 95% de los casos va a ser necesario modificar nuestro código. Al tener un sistema modularizado y estructurado en capas, esos cambios son mucho más fáciles de ejecutar, afectando sólo a las partes que debemos en verdad alterar.

VISTA: refiere a la capa de la vista, la capa que se encarga de presentarle cosas al usuario; sólo se tiene que preocupar de mostrar cosas, es lo único que sabe hacer.
Si el usuario quiere hacer login, esta capa no va a saber (ni le interesa saber) si el nombre y la contraseña son correctas, si hay internet o no, etc. A la vista solo le preocupa obtener un nombre de usuario y una contraseña y pasárselo al controlador.

CONTROLADOR o SERVICIO: refiere a la capa de la lógica, la capa que va a manejar toda la lógica del negocio, las validaciones; solo se preocupa de procesar información y dársela a la Vista para que la presente.
Se va a encargar de validar que algo se pueda hacer siguiendo las lógicas de negocio de la aplicación, validar el contexto (ej si hay o no internet, posición de GPS, etc) y procesar las operaciones lógicas de la información.

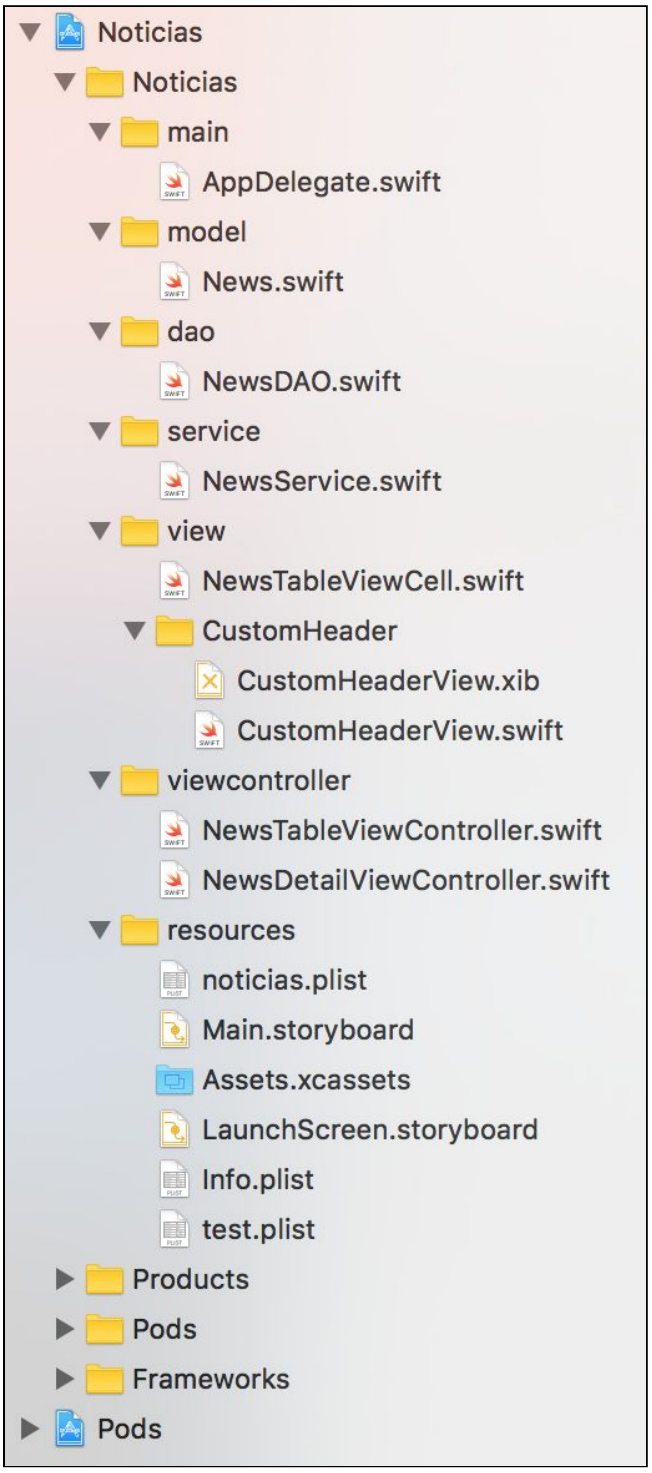
MODELO: refiere a la capa de estructura de los datos. Por un lado se preocupa de definir la estructura de los datos (también llamados DTOs -Data Transfer Objects-), es decir las clases que representan los modelos del negocio. Por otro lado, se contienen los DAOs (Data Access Objects), objetos que se encargan de obtener información (remota, de un archivo, al azar, etc) y generar DTOs a partir de ese contenido.

EJEMPLO

1. Si tenemos que obtener el listado de contactos de un usuario y mostrarlos en una tabla, la capa de la Vista no sabe de donde se va a obtener la información. Lo único que le importa es que se la a pedir a la capa de Servicios y luego presentársela al Usuario.
2. La capa de Servicios no tiene interés en cómo se va a mostrar la información, no le importa si se va mostrar con una tableView o una collectionView o de qué forma. Sólo sabe que tiene que obtener la información, procesarla y enviarsela a la capa de la Vista. Una vez que le solicita los datos a la capa del DAO, puede que tenga que validarla de alguna forma, según la lógica de negocio: por ejemplo ordenar por órden alfabético, y eliminar los contactos que hace más de un año que no se llaman.
3. Finalmente, la capa del DAO va a obtener la información, no le interesan las validaciones, de igual forma que a las otras capas no les interesa si la información viene de un plist local o de un endpoint remoto. Simplemente el DAO correspondiente va a obtener la información, y luego de parsearla generará un DTO que luego enviará a la capa de Servicios.
4. El DTO generado por la capa del DAO será enviado a la capa de Servicios, que luego se lo enviará a la capa de la Vista, quien finalmente lo presentará al usuario.

Importante: si fuera necesario modificar algo, solo modifico una capa del sistema.

ESTRUCTURA DE UN PROYECTO



A partir de ahora, nuestros proyectos van a estar estructurados siguiendo los lineamientos de MVC.
Esto significa que en nuestro proyecto, los archivos van a estar agrupados (en grupos) de la siguiente forma:

- **main**
  Sólo el AppDelegate.
- **model**
  Todos los DTOs.
- **dao**
  Todos los DAOs.
- **service**
  Todos los SERVICEs.
- **view**
  Todas las VIEWs que no sean ViewControllers.
- **viewcontroller**
  Todos los VIEWCONTROLLERs.
- **resources**
  Todos los archivos de recursos, como Storyboards, Assets, plists, etc.