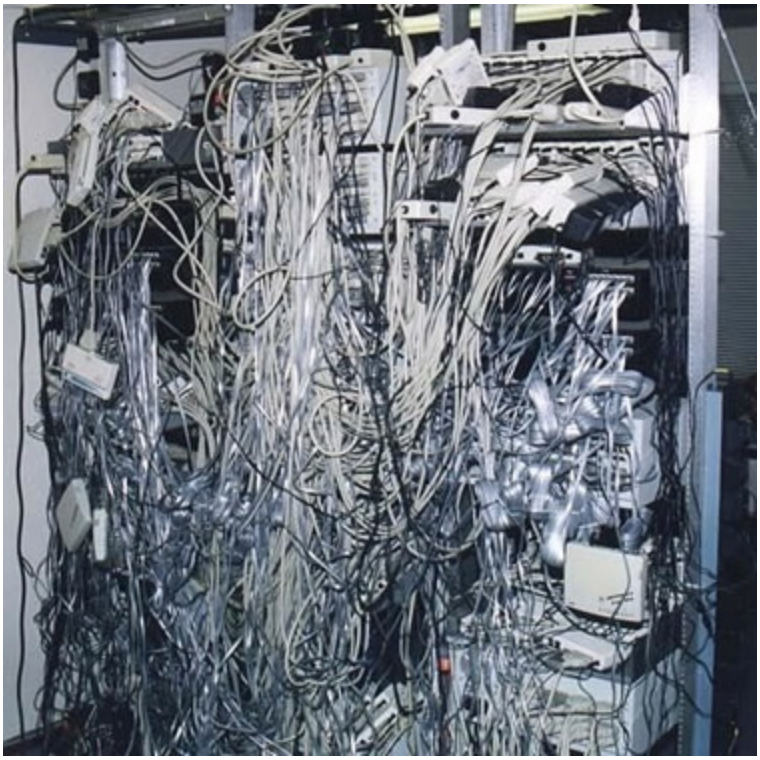
🎯

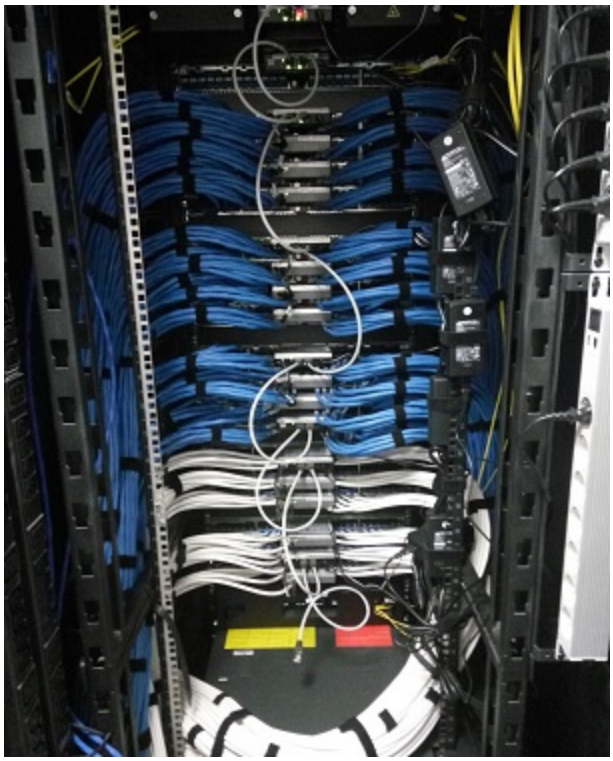
MVVM-C

Хотя каждый из паттернов имеет довольно много отличий, их цели похожи: отделить UI-код (View) от кода логики (Presenter, Controller, ViewModel и т. д.) и кода обработки данных (Model). Это позволяет каждому из них развиваться самостоятельно. Например, вы сможете изменить внешний вид и стиль приложения, не затрагивая логику и данные.

Код без MV*-паттерна



Код с MV*-паттерном



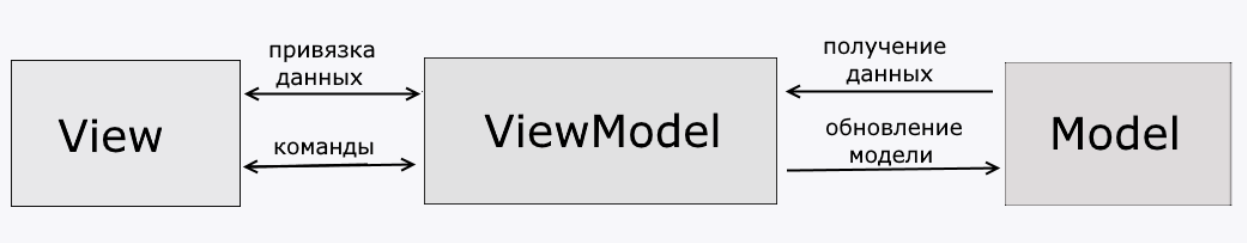
Признаки хорошей архитектуры приложения:

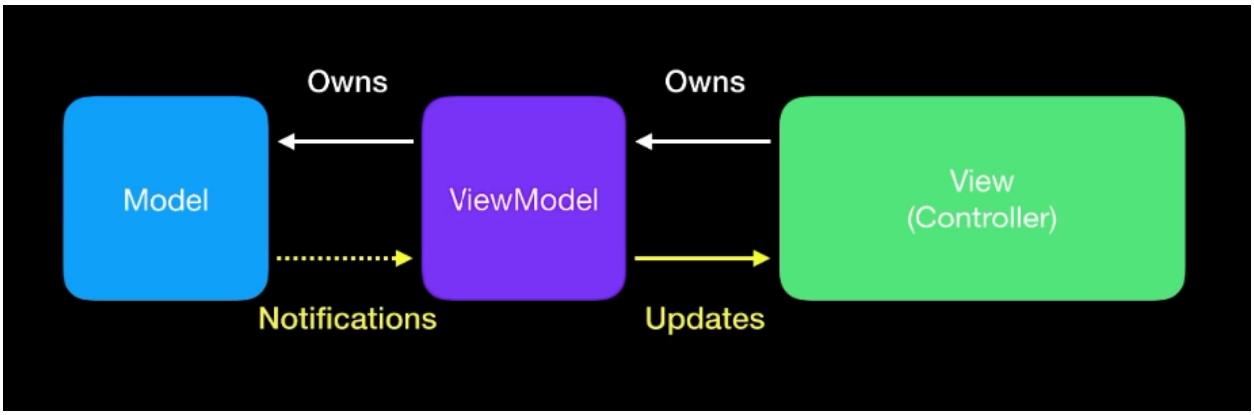
- сбалансированное **распределение** обязанностей между сущностями с жесткими ролями;
- **тестируемость**
- **простота использования** и низкая стоимость обслуживания.

MVVM - шаблон, который появился в 2005 году усилиями компании Microsoft для обхода ограничений паттернов MVC и MVP, и объединяющий некоторые из их сильных сторон.

Основная проблема Apple MVC — смешанная ответственность, что приводит к появлению некоторых проблем, таких как **Massive-View-Controller**.

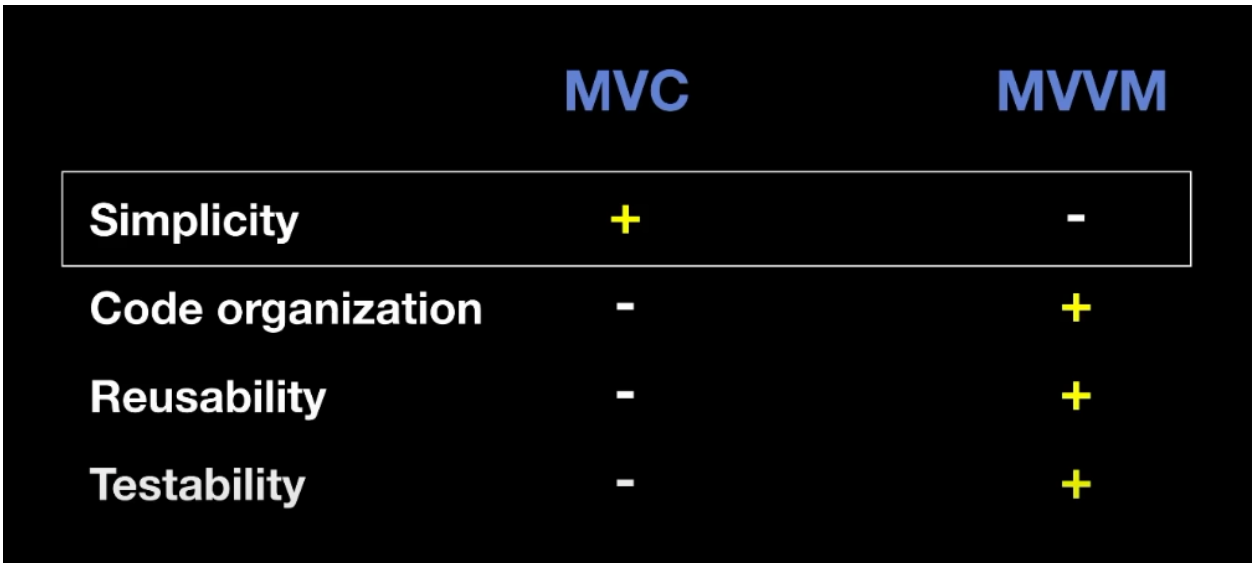
MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).





(источник видео от SwiftBook https://www.youtube.com/watch?v=1eWgghSDlS8)

Ключевые отличия паттерна MVC от MVVM



(источник видео от SwiftBook https://www.youtube.com/watch?v=1eWgghSDlS8)

Слои паттерна MVVM

Model

Model - чистая информация, которая интерпретируется посредством ViewModel.

Может содержать логику работы с данными (вычисления, запросы и т.п.),

Не содержит логики отображения данных и взаимодействия с визуальными элементами управления.

View

View - вид или представление, определяет визуальный интерфейс, через который пользователь взаимодействует с приложением.

Обычно в понятиях UIKit это наследники классов UIView или UIViewController.

View должен быть максимально "глуп", ничего не знает и не ссылается на другие составляющие паттерна.

ViewModel

ViewModel - модель представления, которая служит прослойкой между View и Model, связывает модель и представление через механизм привязки данных.

Содержит логику по получению данных из модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели.

> ViewModel — полное представление данных view. Каждый View должен содержать только один экземпляр ViewModel. Как правило, ViewModel использует диспетчер для извлечения данных и преобразования его в необходимый формат. источник https://apptractor.ru/info/articles/osvoenie-mvvm-na-ios.html

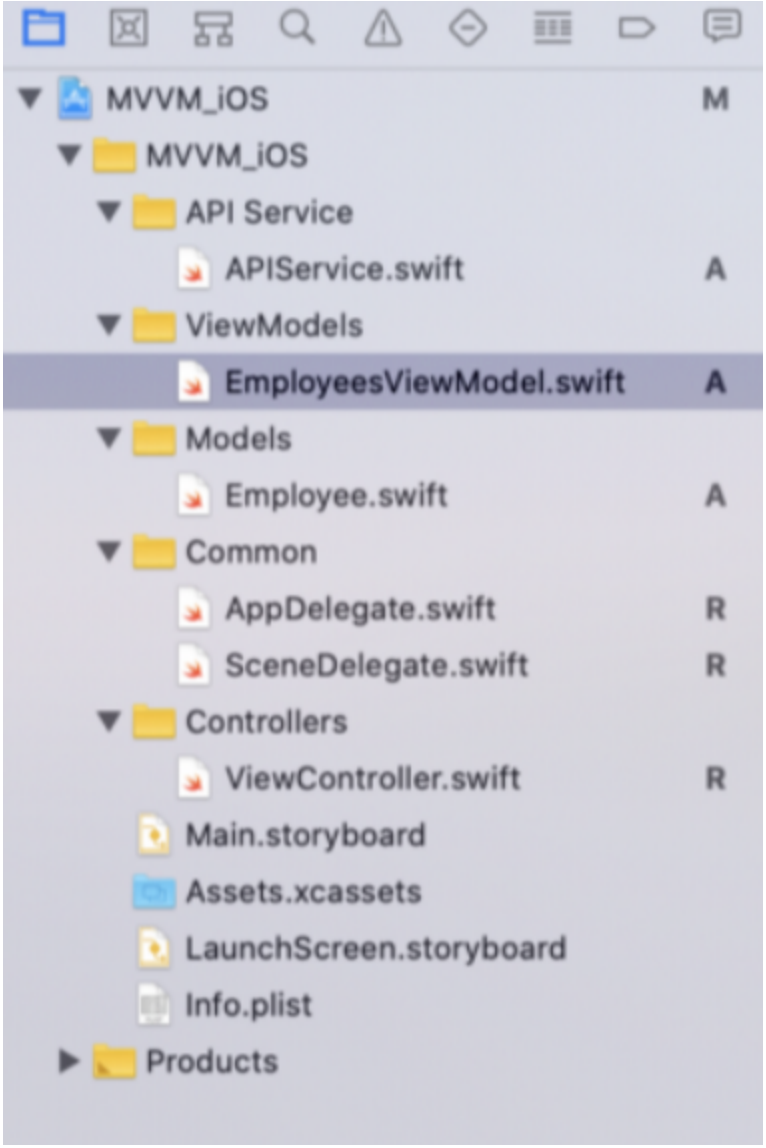
Таким образом, имея разделенные сущности, мы можем:

- лучше понимать их;
- повторно их использовать (в основном применимо к **View** и **Model**);

- тестировать их отдельно друг от друга.

Почему проект, построенный по MVVM легче тестировать?

- Более мелкие кусочки кода.
-



Жизненный пример

Этот паттерн можно разобрать на примере из реального мира. В главных ролях: знаменитость, PR-менеджер и пресса.

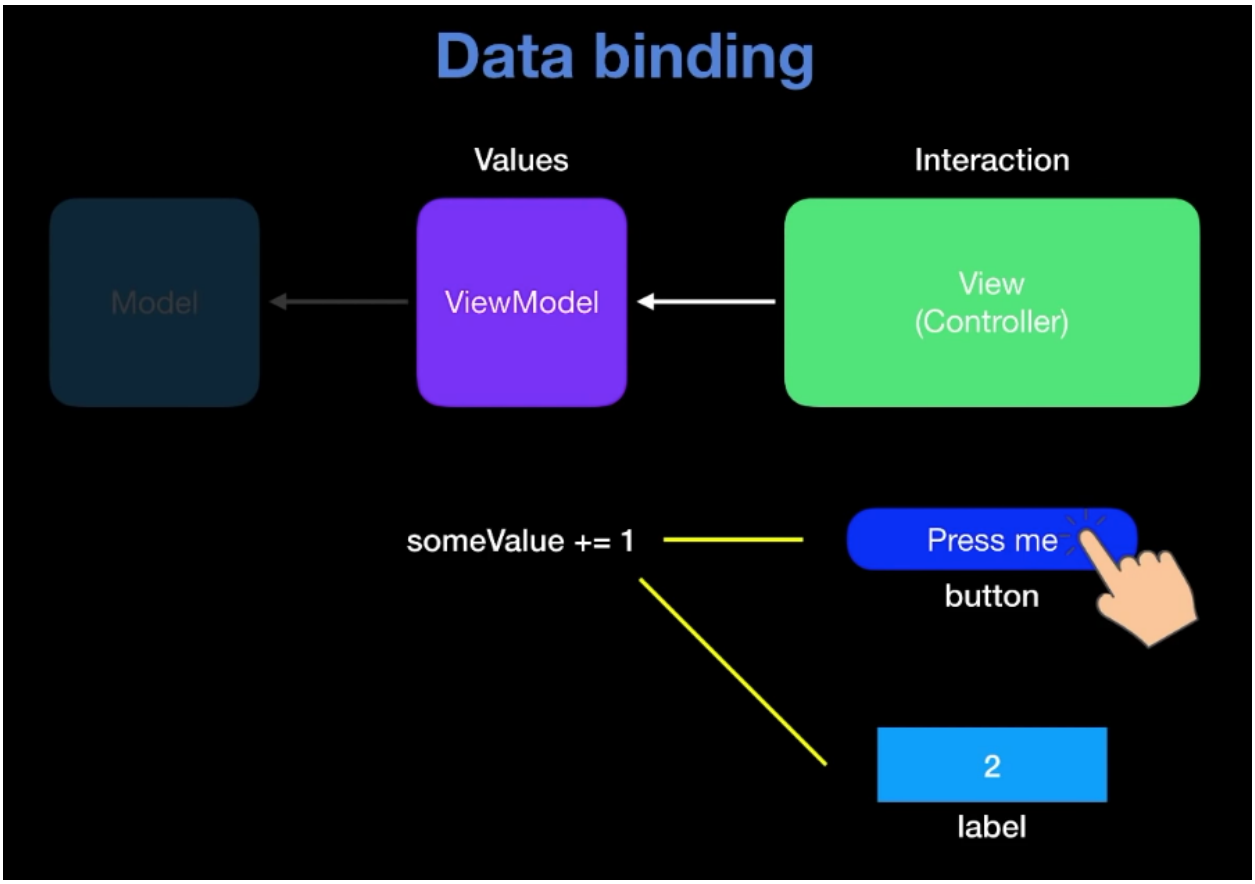
Знаменитость (Model)	**PR-менеджер (ViewModel)**	**Пресса (View)**
Занимается своей непосредственной работой, не отвлекаясь на продвижение. Если нужно, сообщает своему менеджеру, что произошло что-то, о чём нужно рассказать прессе.	Получает информацию от знаменитости и передаёт её прессе. Также может передать своему работодателю запрос от какой-нибудь газеты на проведение интервью или предложение сотрудничества.	Пишет публикации основываясь на данных, полученных от PR-менеджера знаменитости.

Внутренняя кухня каждого компонента никак не связана с другими и им не важно, как это делается. Пресса может поменять редакцию, изменить макет газеты, нанять новых авторов или перейти к другому владельцу. Это никак не влияет не действия PR-менеджера и знаменитости. Каждый из них автономен.

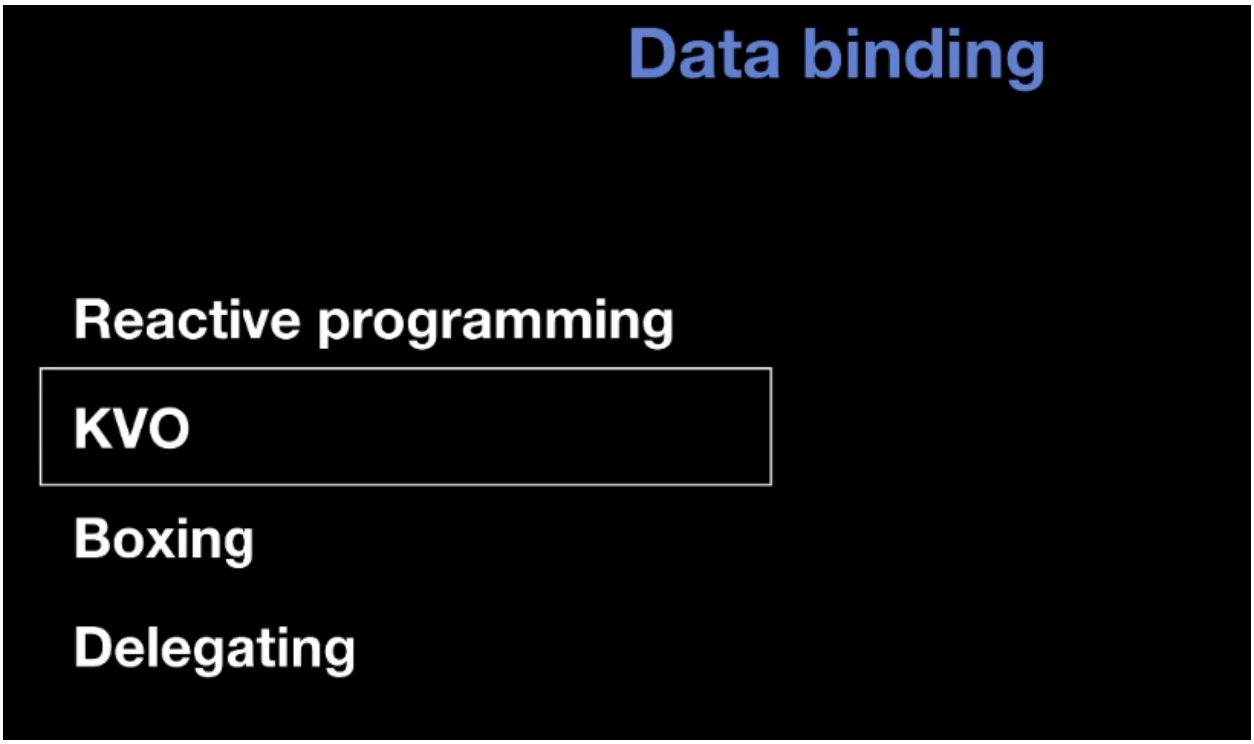
Программист, используя этот паттерн может менять отдельные части приложения, не затрагивая другие.

Недостатки MVVM

- требуется data binding (связывание данных) viewmodel с view, чтобы происходило обновление
- писать больше шаблонного кода



(источник видео от SwiftBook https://www.youtube.com/watch?v=1eWgghSDlS8)



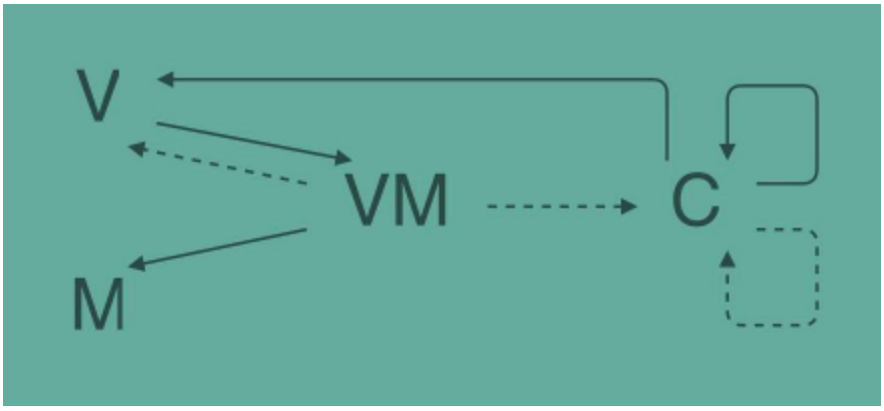
Coordinator

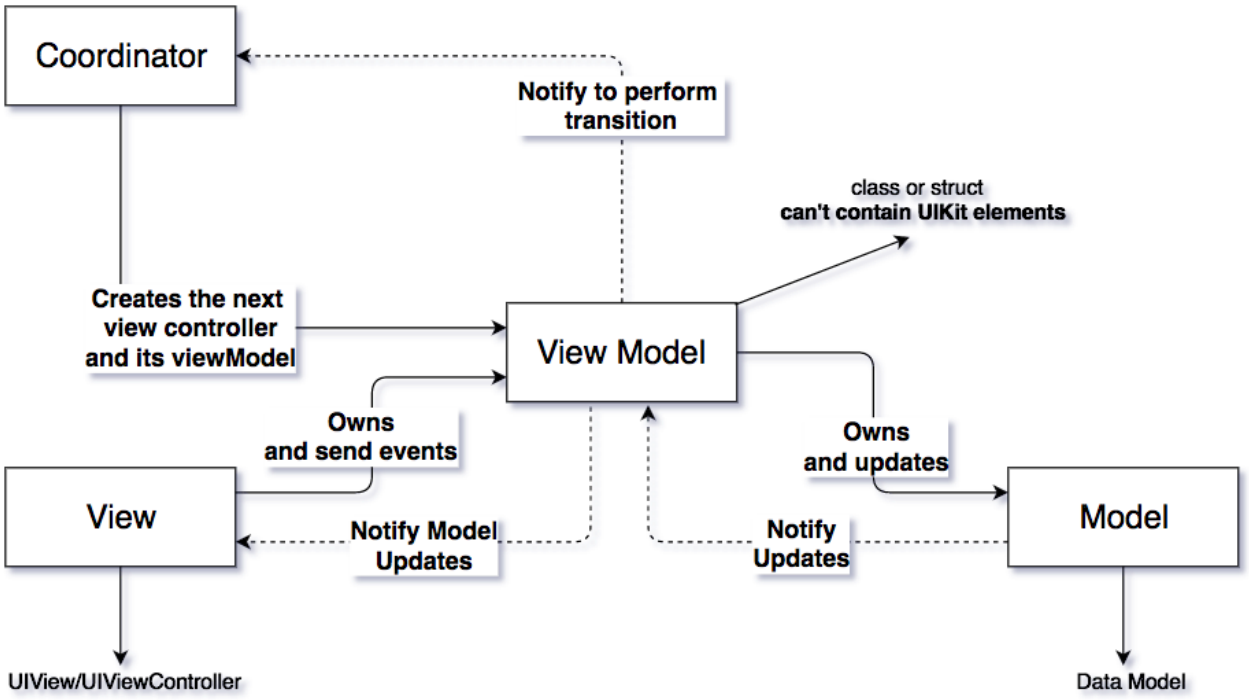
Coordinator (или *ApplicationCoordinator*)- дополнительный компонент, который образует паттерн MVVM-C.

Наиболее распространенные методы построения архитектуры в iOS (MVC, MVP, MVVM) описывают то, как построить один экран-модуль. Модули могут знать друга друга, общаться, но почти не уделяется внимание вопросу перехода между модулями.

Coordinator помогает разрешить проблему взаимозависимостей между ViewControllers в вопросе навигации между экранами.

Чтобы код, связанный с навигацией не был разбросан по всему коду, паттерн Coordinator обеспечивает инкапсуляцию навигационной логики.





Полезные ссылки

https://medium.com/flawless-app-stories/data-binding-in-mvvm-on-ios-714eb15e3913

https://medium.com/flawless-app-stories/how-to-use-a-model-view-viewmodel-architecture-for-ios-46963c67be1b

https://medium.com/@giovannyorozco24/mvvm-and-coordinator-pattern-together-8920fc0f1f55