курс development 12 недель

iOS: разработка приложений с 0

r_d

iOS программа курса 20 занятий

1
Swift: начало

2
ООП: основы

3
Создание iOS-прило-
жения в Xcode

4
Создание интерфейса iOS-приложения

5
Динамические интерфейсы, часть 1

6
Динамические интерфейсы, часть 2

iOS программа курса 20 занятий

7

Динамические интерфейсы, часть 3

8

Навигация в приложении, часть 1

9

Навигация в приложении, часть 2

10

Анимации в iOS

11

Работа с памятью в iOS

12

Многозадачность в iOS, часть 1

iOS программа курса 20 занятий

13

Многозадачность в iOS, часть 2

14

Дебаг iOS-приложения

15

Тестирование

16

Хранение данных в приложении

17

Работа с сетью в приложении

18

Сборка приложения

iOS программа курса 20 занятий

19

Современные архитектуры для iOS приложений

20

Защита курсовых проектов

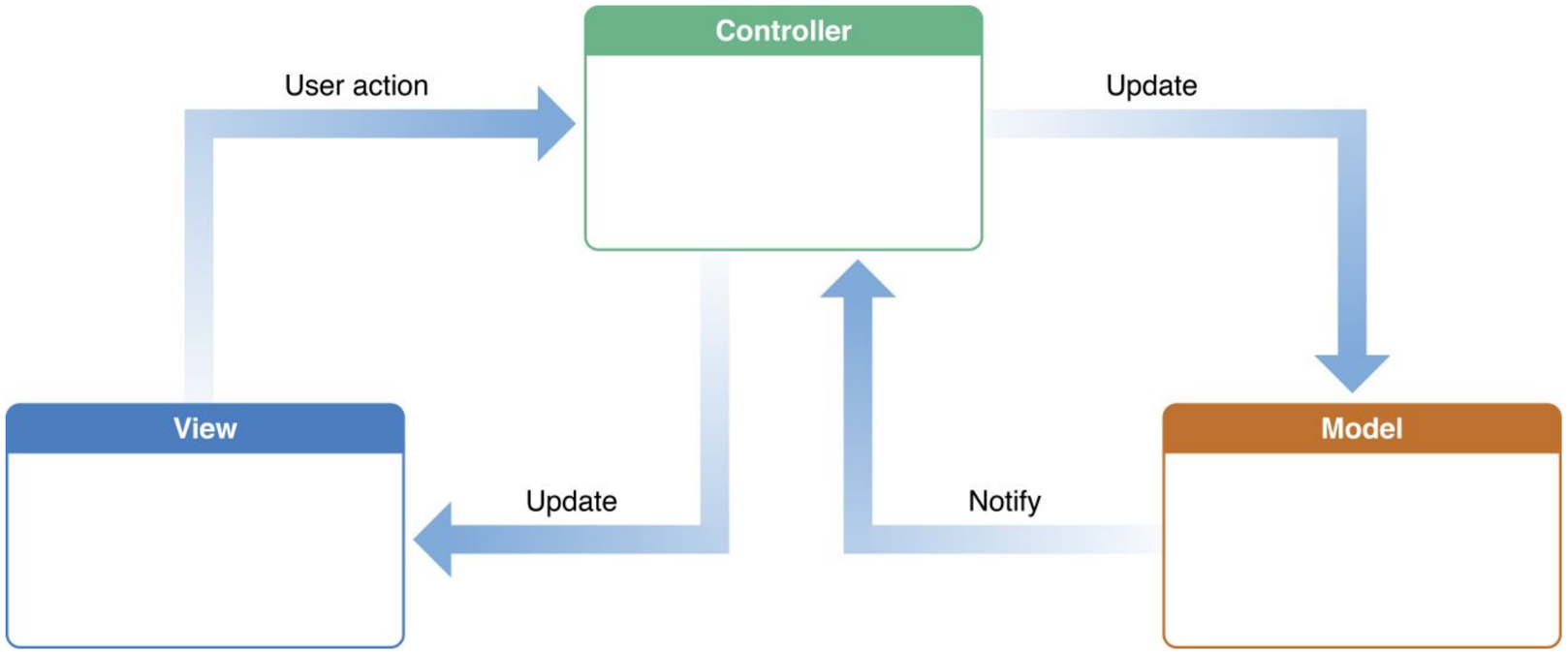
Современные архитектуры iOS приложений

- Архитектура приложения Apple MVC
- Жизнь после MVC
- Решение всех проблем – архетиктура MVVM

Архитектура приложения Apple MVC

- Model-View-Controller (MVC) дизайн паттерн определяет в iOS приложении общую архитектуру коммуникации между её компонентами.

- MVC архитектура предлагает разделение каждого отдельного MVC-модуля на 3 основных компонента: model, view и controller.

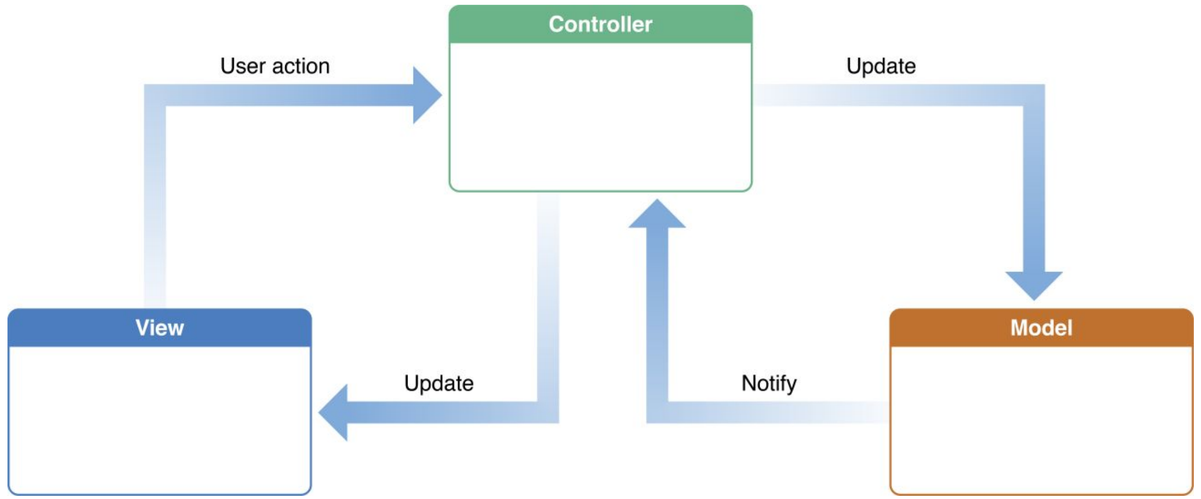


Архитектура приложения Apple MVC

- Кроме определения ролей и разделения ответственности между её объектами, MVC архитектура также определяет способы коммуникации между объектами.

- Важно понимать, что коммуникация происходит не между двумя любыми объектами. MVC определяет кто с кем общается.

- Например, объект view и model могут общается с controller, а контроллер также умеет общаться с view и model. Но view не может общаться с model.



Архитектура приложения Apple MVC

- Слово “коммуникация” (между модулями) также для разных объектов означает разные действия.

- Объект view “коммуницирует” с controller с помощью механизма определения и получения события от пользователя (напр., пользователь нажал на кнопку) и последующей передачи этого события в controller.

- Объект controller содержит в себе конкретный объект модели и общается с model через вызов конкретных методов объекта. Также работает и с view.

- Model не должна знать о том, кто управляет ею, поэтому один из способов “обратной связи” использование механизма нотификаций. Для этого в Apple разработала такие механизмы, как делегирование и Notification Center.

Архитектура приложения Apple MVC

Ответственность view:

- правильное расположение
- отрисовка и показ себя
- реакция на действия пользователя

Детальнее об ответственности view смотрите лекцию №4. Там я подробно прохожусь по тому, чем view занимается.

Архитектура приложения Apple MVC

Ответственность model:

- хранит необходимый набор данных (для отрисовки различных состояний пользовательского интерфейса)

- реализует логику по управлению этими данными (управляет зависимостями, которые хранят и предоставляют данные)

Примеры можно найти в лекциях 16, 17 и 18, где мы рассматривали работу с сетью, хранение данных и использование этих инструментов в проекте

Архитектура приложения Apple MVC

При реализации классов типа model необходимо учитывать ряд правил:

- хранение переменных данных и экземпляров классов зависимостей (инкапсуляция)
- описание бизнес-логики конкретного экземпляра класса (инкапсуляция)
- правильная работа с памятью (инициализация / деинициализация зависимостей)
- правильный “encoding / copying” model-объекта

Архитектура приложения Apple MVC

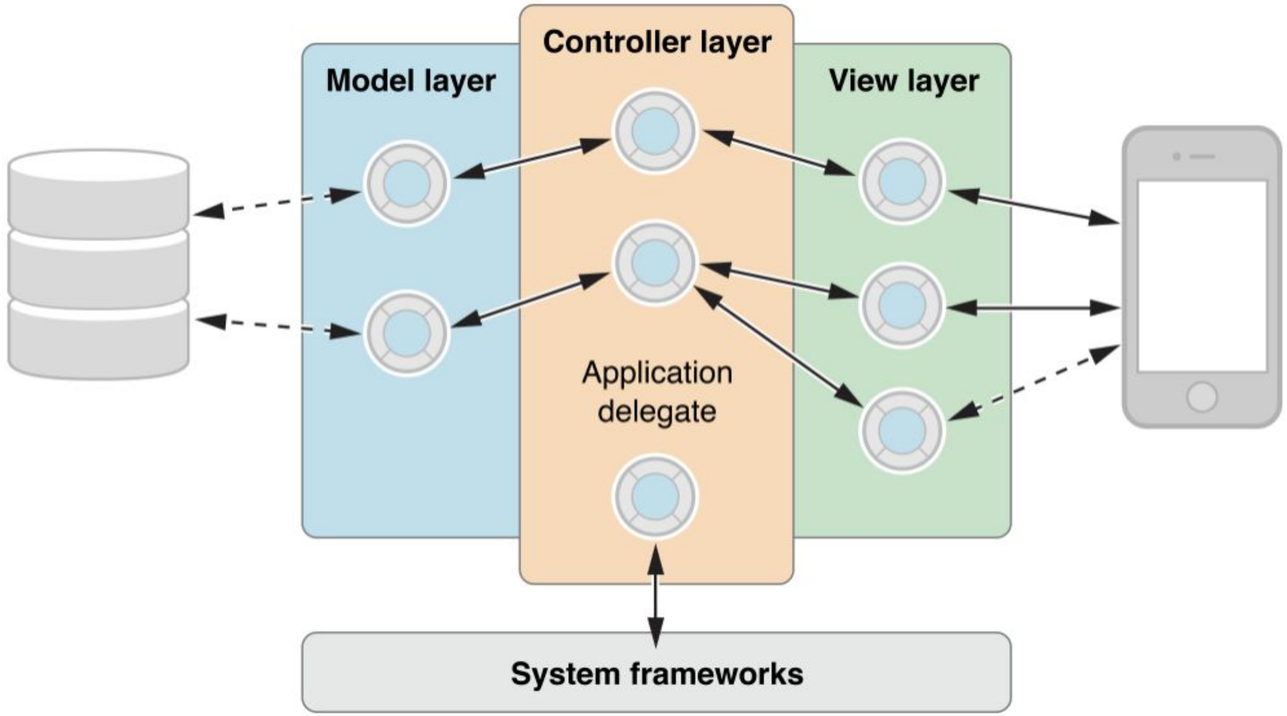
Ответственность controller:

- реагировать на события от пользователя (нажатия, жесты и прочее), пришедшие из view
- создавать / обновлять / удалять данные из model (используя бизнес-логику из model)
- сохранять полученный результат операций (после работы с view) и соответствующе обновлять view

Архитектура приложения Apple MVC

Помним, что существует 2 вида controllers:

- coordination controllers
- view controllers



Архитектура приложения Apple MVC

- Coordinate Controllers отвечают за навигацию и хранение необходимых данных для корректной реализации навигации в приложении.

- Занимаются отправкой данных и нотификаций в нужные view controller.

- Часто coordinate controller занимаются созданием, управлением и распределением зависимостей между model объектами.

- Часто такие coordinate controller объекты являются наследниками класса NSObject. Одним из неплохих примеров можно найти реализацию паттерна Coordinator.

Архитектура приложения Apple MVC

Про view controllers мы довольно детально говорили, начиная с лекции №4 и заканчивая лекцией №7.

Жизнь после MVC

И вроде бы все хорошо, но что-то всегда идёт не так … И вот в один прекрасный день вы открываете проект, а там примерно следующая картина:

- у вас существует один или более классов с десятками методов и свойств, а самое страшное что этот класс – наследник UIViewController
- данные, которые должны храниться в model, хранятся в controller, а иногда и во view
- и при этом классы uiview ни за что не отвечают, максимум что в них есть – хранение IBOutlets
- model хранит только данные, никакой бизнес-логики

Жизнь после MVC

… и при этом всём у вас код не покрыт тестами 👻

Жизнь после MVC

Конечно, сразу винить во всём Apple MVC не стоит. Для начала давайте пройдемся по пунктам, выделяющим хорошую архитектуру:

- сбалансированное распределение ответственности между сущностями, жёсткое определение ролей
- тестируемость, возможность за минимальное время и усилия покрыть ваш модуль тестами
- простая в использовании и поддерживаемости

Жизнь после MVC

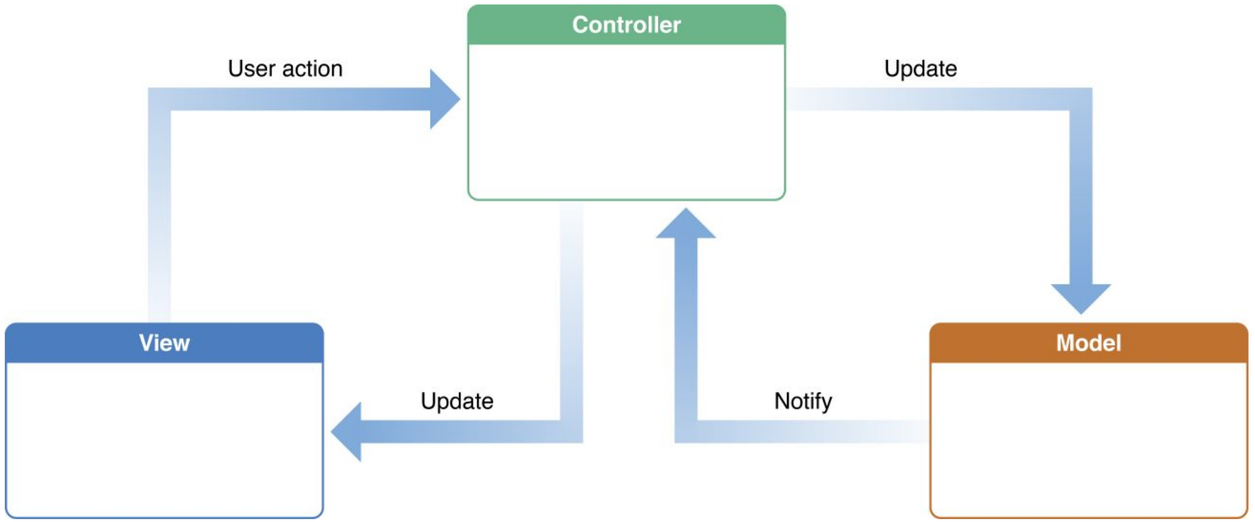
- Забегая немного вперёд, из всего выше перечисленного MVC имеет:

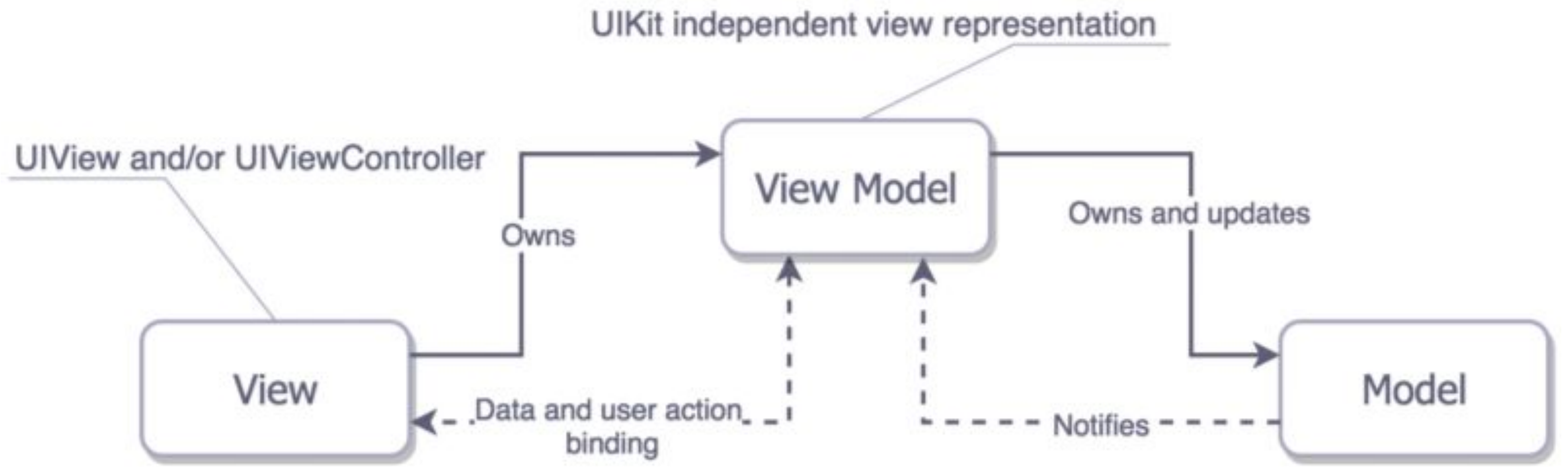
- баланс в распределении ответственностей, но нет довольно строгого разделения ролей. К тому же сам Apple в своих примерах нарушает кучу правил

- тестировать UIView / UIViewController сущности довольно трудоёмко, в особенности если в них находится какая-то бизнес-логика

- всего три роли позволяют нам быстро и легко поддерживать ранее написанный код, но при неправильном распределении бизнес-логики код резко становится трудно поддерживаемым

Решение всех проблем – архетиктура MVVM





Решение всех проблем – архетиктура MVVM

- Какие мы видим отличия?
- MVVM рассматривает View Controller как View
- в нем нет тесной связи между View и Model
- делает биндинг между View и View Model

Решение всех проблем – архетиктура MVVM

- Основная особенность архитектуры MVVM – обратная связь между view и controller (точнее view и view model).

- В классической схеме – view controller знает про view, а view может про view controller ничего и не знать.

- В варианте MVVM используется обратный подход – view знает про view model, а view model может (не должна) не знать про view.

Решение всех проблем – архетиктура MVVM

Если рассмотреть основные пункты хорошей архитектуры, то MVVM:

- имеет лучшую распределяемость ролей между объектами, т.к. учитывает прикладное применение таких объектов как view и view controller
- имеет лучшую тестируемость за счёт отделения view model от UIKit фреймворка, а значит легче написать unit-тесты
- имеет более высокий порог входа, чем MVC но легче поддерживается в дальнейшем за счет более четкого распределения ответственности между view, controller и model

Demo Time 🎉 🎉 🎉