MVC
(Model - View - Controller)

Занятие №4

Дмитрий Тараев

Организационная часть

- Отметиться!
- На прошлом занятии
  - База по Xcode
- **Сегодня**
  - MVC и другие архитектурные паттерны
  - Самостоятельная работа (приложение)
- Оставить отзыв (после занятия)

Вопросы к аудитории

- Кто скачал из репозитория код примера?
- Кто начал работу над своим проектом?
- У кого нет темы?
- Кто ещё не разбился на группы?
- Вопросы?

План лекции

- Повторение (мать учения)
- Про Objective-C
  - посмотрим прошлый пример, переписанный на нём
- MVC
  - Model
  - View
  - Controller
- Другие архитектурные паттерны (кратко, подробнее в лекции «Углублённые темы» 15 мая)

Objective-C

- Нет строгой типизации
  - тип id
  - вызвано тем, что это надстройка над C
    - структуры, указатели на функции
- Runtime
- Отправка сообщения объекту vs. Вызов метода

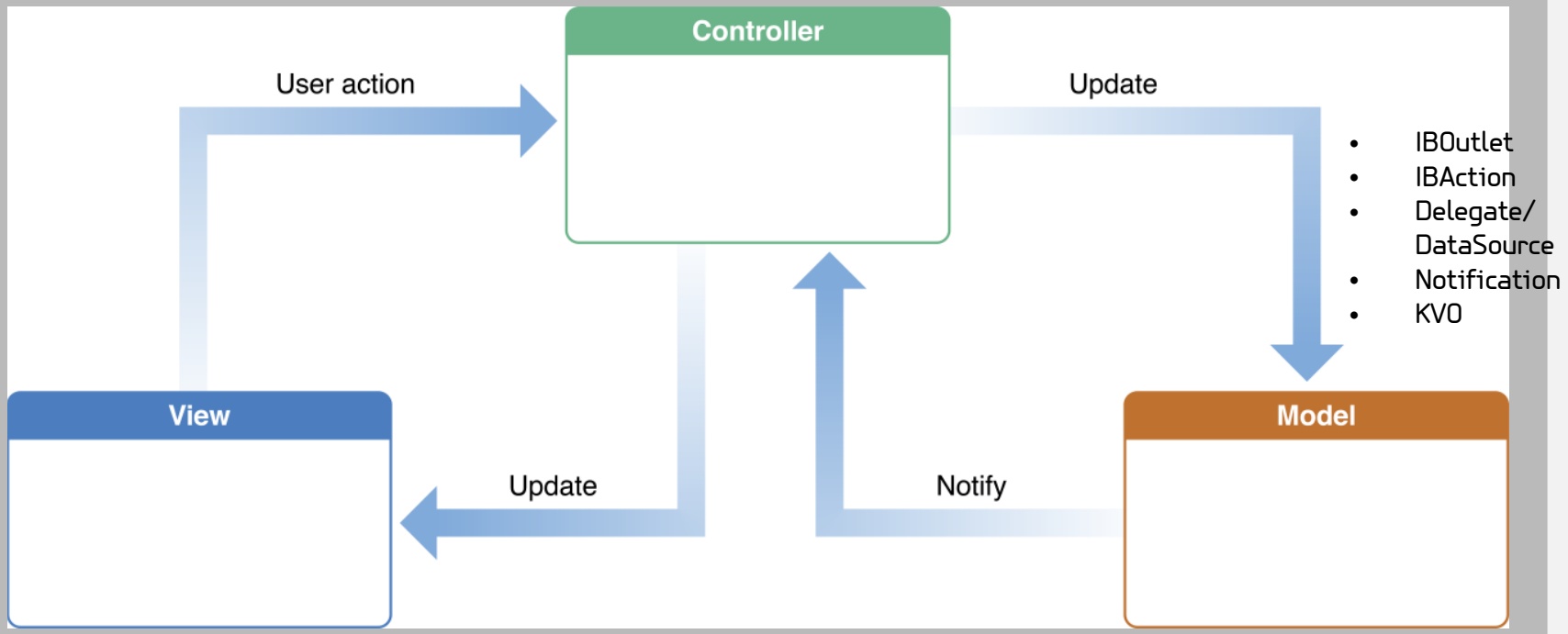
Понадобится для примера:

- Жизненный цикл приложения (состояния приложения)
- Жизненный цикл контроллера
- Контейнеры
  - UITabBarController
  - UINavigationController
- UIView + наследники (label, button…)
- UIScrollView + наследники (table view, collection view)
- Auto layout
- Создание UI с помощью
  - Interface Builder
  - кодом
- Segue vs. работа с Navigation Controller напрямую

Архитектуры приложений

- Предпосылки к возникновению разных архитектур
  - MVC не до конца учитывает специфику мобильных приложений
  - MVC может трактоваться кучей способов, и это плохо
- Что решают другие архитектуры?
  - Унификация кода для того, чтобы можно было легче в нём разобраться
  - Уменьшение неопределённости при написании кода

Схема MVC



https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Model

- сами данные
- обеспечение доступа к данным
- логика для изменения данных

- Ещё называют реализацией бизнес-логики приложения

- Не зависит от UI
- Чаще всего потомок NSObject (в Objective-C) или базового класса базы данных

View

- Стандартные элементы (view, label и т.п.)
- Показывают данные
- Общаются с пользователем
- Ничего не знают о модели

Controller

Координирует взаимодействие между View и Model (посредник)

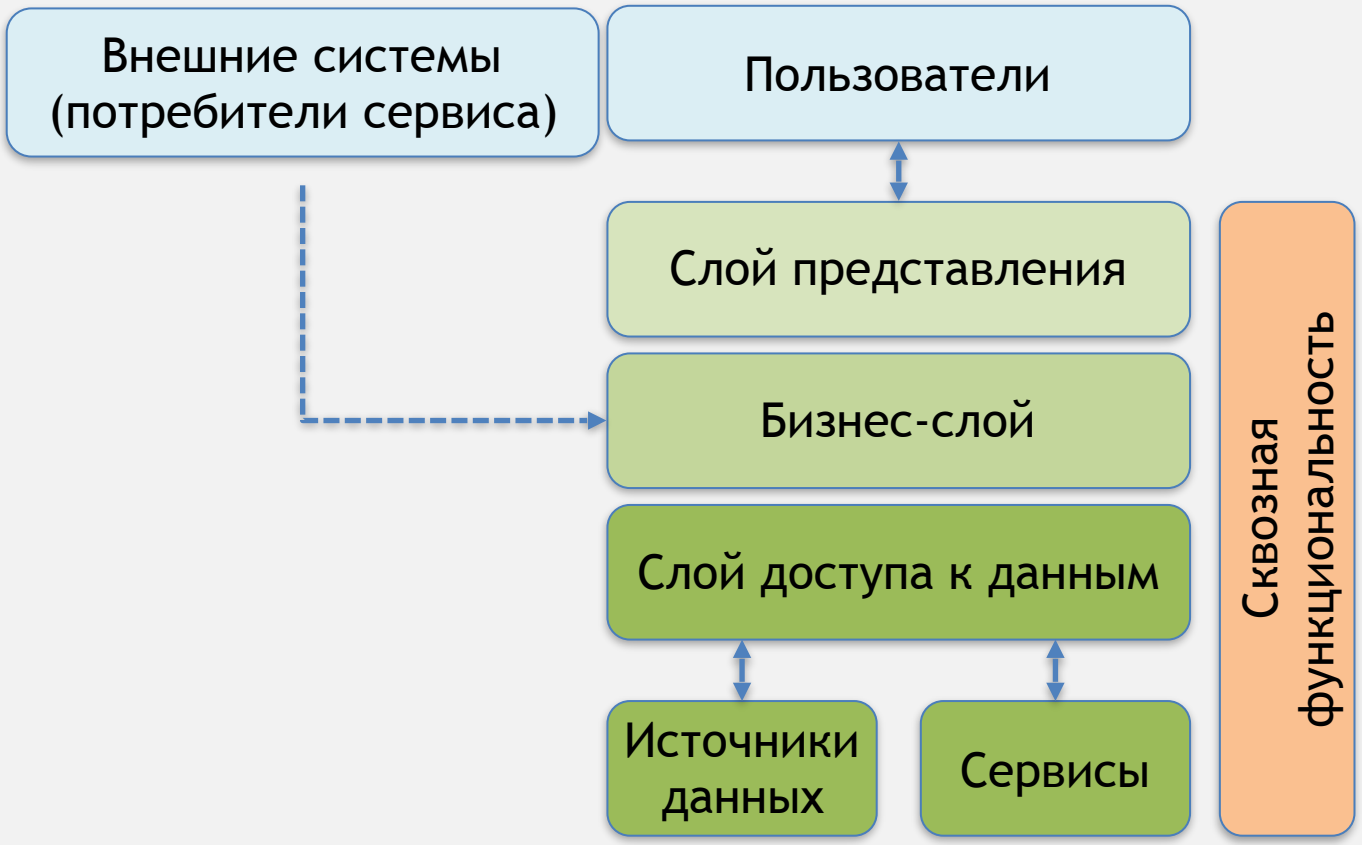
- Заполняет view данными
- Обрабатывает события от view

- Передаёт данные в модель
- Берёт данные из модели (модель уведомляет об изменении данных)

Типовые составляющие приложения

1. Пользовательский интерфейс (UI)
2. Бизнес-логика
3. Сетевое взаимодействие
4. Хранение/кеширование данных

Layered архитектура приложения



Способы коммуникации между слоями

1. Делегат
2. Block / closure
3. Notifications
4. KVO
5. IBOutlet / IBAction (между View и Controller)

Основные архитектурные паттерны

1. MVC
   - чем плох?
2. MVP
   - похож на MVC, но
     - presenter не занимается layout
     - вместо view можно использовать mock-объекты
     - view controller относится к view
3. MVVM
   - похож на MVP, но связь не view и model, а view и view model (биндинг)
4. VIPER
   - ещё большее разделение обязанностей
     - view - UI
     - presenter - бизнес-логика, связанная с UI (всё инициализирует в модуле)
     - interactor - бизнес-логика, связанная с данными
     - router - переходы между модулями
     - entity - объекты данных

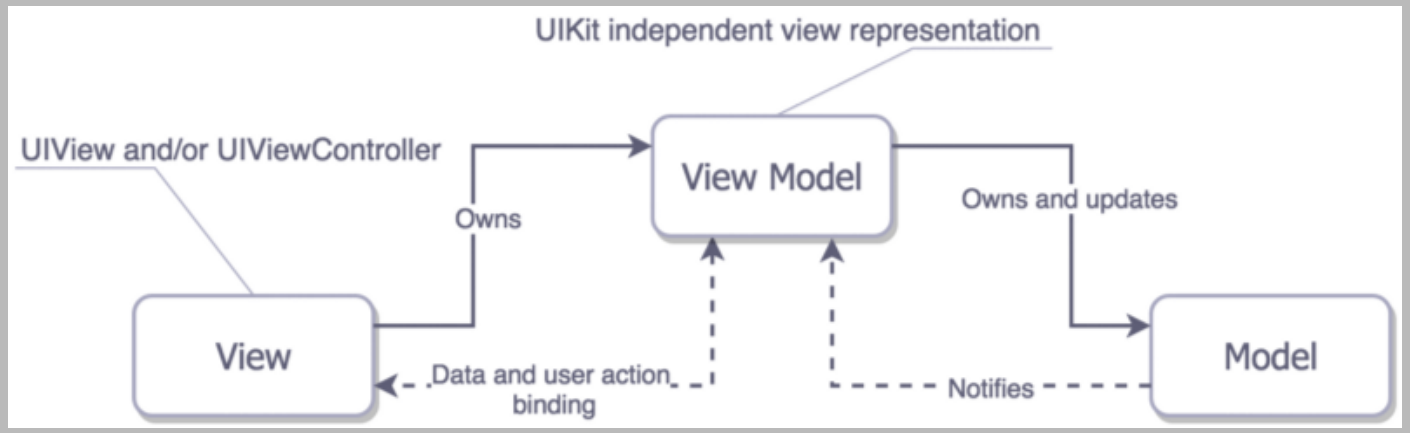
https://habrahabr.ru/company/badoo/blog/281162/

Архитектуры приложений

- Предпосылки к возникновению разных архитектур
  - MVC не до конца учитывает специфику мобильных приложений
  - MVC может трактоваться кучей способов, и это плохо
- Что решают другие архитектуры?
  - Унификация кода для того, чтобы можно было легче в нём разобраться
  - Уменьшение неопределённости при написании кода

Архитектуры приложений (MVVM)

- MVVM (Model View ViewModel)
  - Отлично работает в связке с Rx* штуками
  - Отчасти решает проблему работы со сложными данными

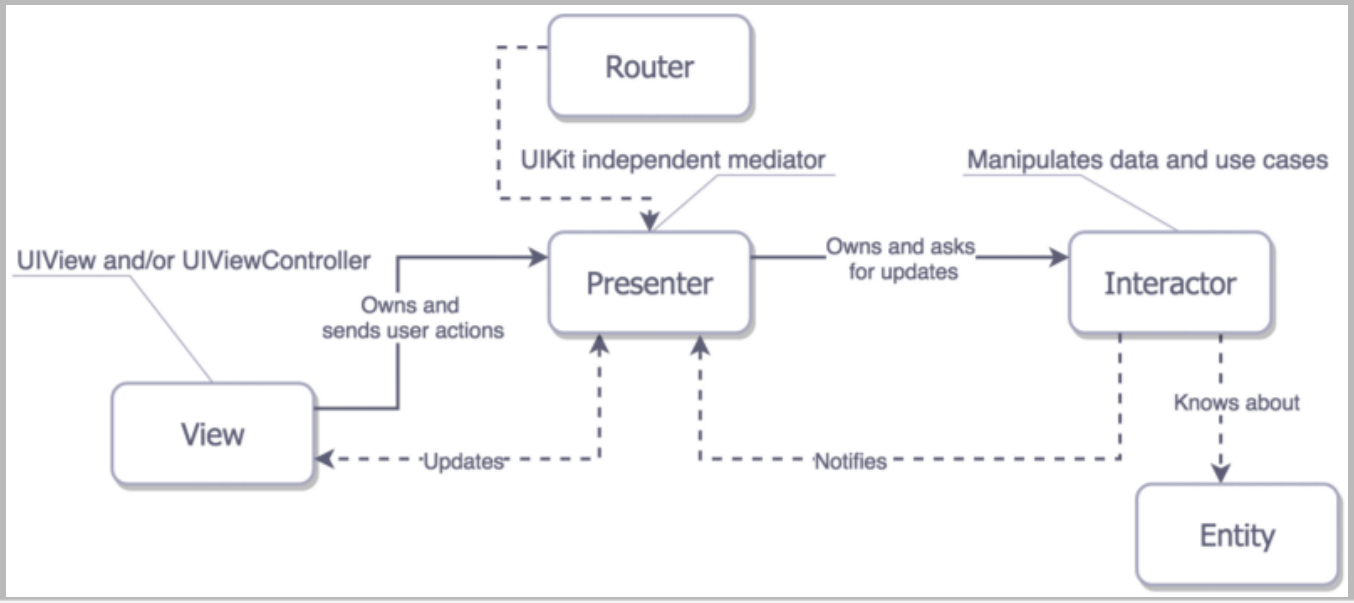


Архитектуры приложений

- VIPER
  - Стильно, модно, молодёжно
  - V - view (представление)
  - I - interactor (бизнес-логика)
  - P - presenter (связь всего со всем)
  - E - entity (сущности)
  - R - router (маршрутизатор)

Архитектуры приложений (VIPER)

- Основная единица — модуль
- Модули независимы друг от друга
- Модуль это не обязательно == экран, на сложном экране может быть несколько модулей
- Взаимодействие между модулями осуществляется через интерфейсы ModuleInput и ModuleOutput



Архитектуры приложений

- VIPER
  - Плюсы
    - Переиспользуемость модулей (хотя на практике это не часто применяется)
    - Тестируемость
  - Минусы
    - Многословность, на экран надо писать минимум 5 классов и кучу интерфейсов (отчасти решается кодогенерацией)

Заключение

- **Сегодня**
  - MVC и другие архитектурные паттерны
  - Самостоятельная работа
- **Следующая лекция**
  - Коллекции (TableView, CollectionView)
- **Отзыв**
- **Вопросы?**
  - по лекции
  - по проектам

Ссылки

- Про «тяжёлые» view controller'ы
  https://www.objc.io/issues/1-view-controllers/

- Про структуру проекта в Xcode:
  https://habrahabr.ru/post/261907/

- «Архитектурный дизайн мобильных приложений»
  часть 1: https://habrahabr.ru/company/redmadrobot/blog/246551/
  часть 2: https://habrahabr.ru/company/redmadrobot/blog/251337/

- Про архитектурные паттерны:
  https://habrahabr.ru/company/badoo/blog/281162/