1.

Всем привет. Меня зовут Виктор Пузаков и я хочу вам рассказать о способе организации экранов со сложным контентом который мы используем на нашем проекте. Он основан на принципах композиции и модульности.

2.

В мобильных приложениях мы очень часто используюем таблицы и коллекции. Как правило они нужны для отображения большого количества повторяющегося контента. И нередко экраны, которые мы должны отобразить пользователю состоят из нескольких типов ячеек.

Препарируем самый обычный экран отзывов из АппСтор.

Он состоит из лейбла и таблицы с 2 типами ячеек.

Как мы сделаем подобную таблицу? Первое что приходит на ум – сделать таблицу с помощью добавления if-else внутрь методов UITableViewDataSource. Которые потом так и останутся в конечном продукте и никуда не денутся.

В рамках подобной задачи нет смысла продумывать какое-то более элегантное решение и тратить на это время.

3.

Намного сложнее, если типов ячеек большое количество. Для примера рассмотрим экран раздела игр из АппСтор..

Представленная на этом экране таблица состоит уже из 6 разных типов ячеек.

При этом, у ячеек таблицы может быть несколько состояний, они могут показываться или скрываться в зависимости от каких-то условий.

4.

Одинаковых блоков на одном экране может быть по несколько штук. Многие из них представляют собой содержимое таблицы или коллекции. Реализовывать всю логику в одном контроллере было бы некрасиво и неудобно, поддерживать такой экран в дальнейшем было бы очень сложно.

5.

К тому же если мы посмотрим на приложение в целом, то можем обнаружить что некоторые элементы используются на других экранах.

6.

Основная идея - реализовать некий «datasource»-объект, который бы мог составляться из любого числа других «datasource»-объектов в единое целое.

Фактически этом массив “datasource” объектов. Каждый элемент этого массива является небольшим модулем конечного экрана – провайдером. Внутри каждого провайдера может быть реализована логика отображения и взаимодействия с одним или несколькими типами контента, в зависимости от конкретных условий и задач.

При этом решение, очевидно, должно быть обобщенным, подходить для любого числа составляющих (включая ноль и один) и не зависеть от конкретных типов.

7.

Наши цели при разработке заключаются в следующем.

Система должна позволять отобразить:

Одну секцию с ячейками

(1 секция = 1 провайдер)

8.

Несколько провайдеров в одной секции

(1 секция = N провайдеров)

9.

Несколько секций в одном провайдере, например если у нас есть хедер секции и 1 тип ячейки и мы не хотим создавать лишние объекты

(N секций = 1 провайдер)

10.

Это протокол для наших будущих провайдеров, они должны быть датасорсами и делегатами таблицы и должны хранить типы используемых ячеек, которые мы потом зарегистрируем. Они будут предоставлять композитору информацию об отображаемом контенте и обрабатывать действия пользователя. Таким провайдером может быть как обычный класс, который реализует этот протокол, так и вью контроллер, поэтому для удобства мы также назначим псевдоним.

11.

Композитор это класс который будет хранить наши провайдеры и управлять потоком данных от них. Мы хотим чтобы таблицу можно было создавать в коде или из ксиба, поэтому мы будем ее инжектить извне.

12.

Композитор должен уметь добавлять и удалять провайдеры. Здесь у нас простейший пример того как это должно работать. Композитор должен уметь регистрировать в таблице типы ячеек, которые предоставляют провайдеры. Также он должен уметь удалять провайдеры по одному, либо все вместе.

13.

Если посмотреть на все методы UITableViewDataSource можно заметить, что в качестве аргументов они принимают только ссылку на таблицу и номер секции или IndexPath. Поэтому все задачи можно свести к двум дженерик функциям, которые будут принимать ссылку на конкретный провайдер и номер секции или IndexPath. Для удобства мы назначим им псевдонимы.

14.

Эти две функции тоже вспомогательные, они берут таски которые мы только что видели и перенаправляют их нашим конкретным провайдерам с соответствующим номером секции либо IndexPath.

15.

Все эти приготовления и помощники дают неимоверное преимущество в реализации, собственно, методов протокола. Таким образом сводиться к минимуму работа нашего композитора. Как и обсуждалось ранее, он также должен быть датасорсом и делегатом таблицы. Здесь  мы должны реализовать все методы, которые мы хотим использовать.

16.

Соответственно для методов делегата таблицы мы должны сделать то же самое.

17.

Мы объявляем протокол для композитного контроллера таблицы и говорим, что контроллер который хочет управлять провайдерами должен иметь ссылку на таблицу и ссылку на композитор.

18.

Так как провайдеры могут быть не только обычными наследниками nsobject, но и вью контроллерами, наш композитный контроллер должен уметь регистрировать и удалять оба вида провайдеров. В первом случае мы просто говорим композитору что нужно сделать.

19.

А во втором мы еще добавляем и удаляем провайдеры в качестве дочерних контроллеров, чтобы у нас правильно отрабатывали системные вызовы, например навигация.

20.

Вот так примерно может выглядеть наш композитный контроллер. Он реализует протокол TableViewCompositeController. У него есть ссылка на композитор, на таблицу и на контроллеры провайдеров ячеек для таблицы.

21.

Здесь же мы можем инициализировать композитор, передать ему Таблицу, сконфигурировать все настройки, в каком порядке должны быть зарегистрированы и отображены провайдеры, при каких условиях. Настроить Хедер и Футер для Таблицы, если они нужны.

22.

Таким образом может выглядеть простой Провайдер. Он обязан хранить массив с типами ячеек.

23.

А в остальном все так же как при обычной работе с таблицей.

24.

Собственно, самое время вспомнить о плюсах и минусах

Используя этот подход мы можем реализовать небольшие экраны, например экраны регистрации и авторизации. При этом можно сделать так чтобы поля на нескольких экранах были взаимозаменяемыми, мы можем их переиспользовать их на схожих экранах, например при редактировании профиля.

Мы можем создавать большие экраны с множеством разнородного контента, с несколькими видами ячеек, при этом отдельная логика будет скрыта внутри отдельного провайдера.

При этом мы используем механизм декьюинга ячеек, это огромный плюс потому что мы не держим все вью в памяти, как это было бы при работе со scrollview.

Порог вхождения очень низкий и даже начинающие разработчики смогут быстро понять что к чему, потому что таблицы есть практически в любых приложениях и зачастую это один из первых компонентов при знакомстве с разработкой мобильных приложений.

Главным минусом можно отметить тот факт, что система требует доработки. Например, для реализации множественных связей между секциями и модулями. Или возможность перетаскивания строк.

(это невозможно реализовать при помощи только лишь UITableViewDataSource. Методы этого протокола определяют, можно ли «перетаскивать» конкретную строку и получают «коллбэк» по окончании перетаскивания. А обработка самого события подразумевается внутри методов UITableViewDelegate)

Теоретически, данный подход можно использовать и для экранов на основе CollectionView.