АРХИТЕКТУРА «МОБИЛЬНОГО ИНТЕРВЬЮЕРА»: ОТ MVC К VIPER

Кинаятов Д.К., студент кафедры АОИ

Научный руководитель: Сидоров А.А., канд. экон. наук, доцент кафедры АОИ

Разработка мобильных приложений состоит из множества этапов: анализа требований, проектирования, реализации, тестирования, внедрения и поддержки. Выбор архитектуры влияет на осуществление большинства этапов жизненного цикла. Де-факто классическим способом проектирования мобильных приложений является использование шаблона MVC – Model-View-Controller (модель-представление-контроллер). При использовании указанной методологии для архитектуры приложения каждый класс – это или модель, или представление, или контроллер. Поскольку значительная часть логики приложения не входит в модель или представление, она обычно оказывается в контроллере. Этот факт приводит к проблеме, известной как Massive View Controllers (массивные контроллеры представлений), где контроллеры в итоге выполняют слишком много задач [1]. Если вся логика встроена в контроллер, это приводит к тестированию логики через графический интерфейс приложения, в свою очередь, это является неправильным способом проектирования логики, так как отдельный разработчик не сможет независимо написать тесты для своей части.

С данной проблемой сталкиваются все проекты, за исключением небольших. Проект мобильного клиента «Интервьюер» (рабочее название «CSM» – «CodeSurveyMobile») системы облачного сервиса подготовки и проведения социологических опросов не стал исключением. Разрабатываемая система должна упростить всю работу путём избавления от сбора исходных данных на бумажном носителе и их ручного анализа. В целом разрабатываемая система облачного сервиса подготовки и проведения социологических опросов представляет собой совокупность следующих элементов:

– сайт, на котором организация или индивид создаёт свои опросы;
– сервер, служащий хранилищем и обработчиком опросов;
– мобильное приложение для платформ Android и iOS, которое является клиентской частью для прохождения опросов, сформированных на сайте, и отправки данных на сервер.

Используя паттерн MVC, программные классы мобильного приложения «CSM» выполняли множество обязанностей, поэтому чтобы найти нужный фрагмент кода может уйти немало времени, а чтобы разобраться с кодом незнакомый с проектом разработчик должен долго изучать классы. Для решения проблемы необходим иной подход к проектированию.

Альтернативами архитектуры MVC, по-иному разделяющими обязанности компонентов программы, являются MVP – Model-View-Presenter, MVVM – Model-View-ViewModel, VIPER. Из них особенно интересной для разработчиков является VIPER, так как она не принадлежит к семейству прежде упомянутых методологий – MV(X). Именно VIPER наиболее широко описывает обязанности классов, предоставляя пять слоев модуля вместо привычных трёх. Было решено перевести мобильный клиент для операционной системы Android на архитектуру VIPER.

Архитектура VIPER была разработана американской компанией Mutual Mobile, занимающейся разработкой и тестированием мобильных приложений. Компания искала способы для улучшения тестирования их программных продуктов, но написание тестов оказалось довольно сложным. Тогда, чтобы улучшить способ тестирования программного обеспечения разработчики из Mutual Mobile придумали иной способ проектирования своих приложений – VIPER.

VIPER – это подход к архитектуре мобильных приложений, основанный на идеях Роберта Мартина, изложенных им в статье The Clean Architecture [2]. Clean Architecture делит логическую структуру приложения на различные уровни обязанностей. Это упрощает изолирование зависимости и тестирование взаимодействия на границах между уровнями. Изначально архитектура VIPER являлась реализацией Clean Architecture для приложений под

операционную систему компании Apple – iOS, но позже нашла применение и в остальных операционных системах, в том числе Android.

Слово VIPER – аббревиатура, складывающаяся из следующих слов: View (Вид), Interactor (Интерактор), Presenter (Презентатор), Entity (Сущность) и Routing (Маршрутизация). Архитектуру можно разделить на три слоя ответственностей (рисунок 1): слой представления, слой бизнес-логики и слой данных. К слою представления относятся следующие части: View, Presenter и Router. Слой бизнес-логики представляет Interactor, а слой данных – Entity.

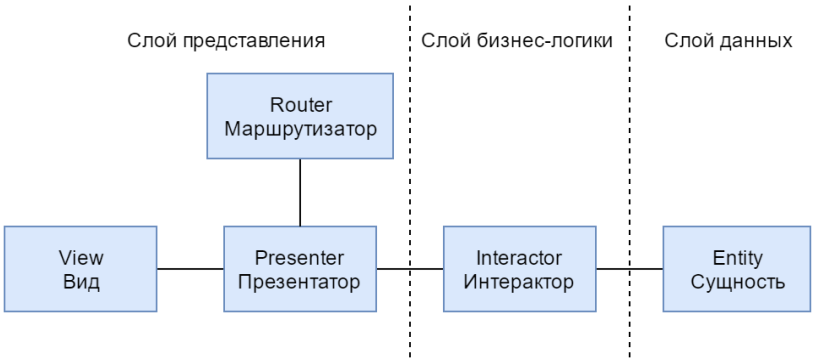

Рисунок 1 – Слои ответственностей архитектуры VIPER

Любое приложение можно поделить на модули, выполняющие свой ряд задач. Для их реализации требуется реализовать бизнес-логику модуля, работу с сетью и базой данных, отрисовку пользовательского интерфейса. За все это должны отвечать отдельные компоненты, и VIPER описывает роль каждого компонента и способы их взаимодействия между собой [3]. Итак, VIPER-модуль состоит из следующих частей:

– View: отвечает за отображение данных на экране и оповещает Presenter о действиях пользователя. Пассивен, сам никогда не запрашивает данные, только получает их от Presenter'а;

– Interactor: содержит всю бизнес-логику, необходимую для работы текущего модуля;

– Presenter: получает от View информацию о действиях пользователя и преображает её в запросы Router'у или Interactor'у, а также получает данные от Interactor'а, подготавливает их и отправляет View для отображения;

– Entity: представляет объекты модели, не содержащие никакой бизнес-логики;

– Router: отвечает за навигацию между модулями.

На таблицах 1 и 2 приведены выполняемые функции в различных частях архитектур MVC и VIPER.

Таблица 1 – Функции различных частей архитектуры MVC

Model	View	Controller
Взаимодействие с базой данных	Визуализация экрана	Управление событиями экранных элементов
Взаимодействие с сетью	Управление состоянием экранных элементов	Обновление Model
		Навигация между экранами приложения
		Взаимодействие с системными компонентами
		Управление системными событиями
		Обновление представления в ответ на системные события

Таблица 2 – Функции различных частей архитектуры VIPER

<table>
<tr><th>View</th><th>Interactor</th><th>Presenter</th><th>Entities</th><th>Router</th></tr>
<tr><td rowspan="2">Визуализация экрана</td><td>Взаимодействие с сущностями</td><td rowspan="2">Управление событиями экранных элементов</td><td rowspan="2">Предоставление данных из источников данных</td><td rowspan="2">Навигация между экранами</td></tr>
<tr><td>Взаимодействие с базами данных</td></tr>
<tr><td rowspan="2"></td><td>Взаимодействие с системными компонентами</td><td>Обновление состояния экрана</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>Управление системными событиями</td><td>Управление состоянием экранных элементов</td></tr>
</table>

В результате анализа было выяснено, что архитектура VIPER даёт больше возможностей при построении кода для дальнейшего тестирования и сопровождения, так как независимые модули могут быть написаны разными разработчиками, что предотвращает коллизии в коде, и протестированы отдельно. Самый важный недостаток VIPER – резкое увеличение количества классов, но это не так критично, так как эти классы будут отвечать за конкретные функции, и иметь меньше кода, по сравнению с MVC, в которой классов хоть и меньше, но эти классы будут массивнее и запутаннее.

При миграции мобильного приложения с MVC на VIPER решаются следующие задачи:

- разбитие приложения на модули по экранам;
- анализ классов с целью выявления отдельных частей VIPER;
- создание схемы классов для проекта;
- создание первоначальных компонентов: частей View всех экранов;
- написание и отладка программного кода всех частей;
- тестирование всех программных компонентов.

При анализе классов из контроллеров были выделены части кода, взаимодействующие с интернетом, базой данных и внутренним хранилищем данных приложения. Главная часть создания схемы классов – выделение верных зависимостей объектов классов. Написание программного кода, его последующая отладка и тестирование заняла основной ресурс времени.

В рамках архитектуры MVC прежде были разработаны классы, в которых View и Controller были совмещены в связи с малым разграничением обязанностей. Схема классов представлена на рисунке 2: синие – классы модели данных, фиолетовые – классы контроллеров отображения, зеленый – интерфейс констант, оранжевые – классы контроллеров данных, жёлтые – классы адаптеров экранных элементов.

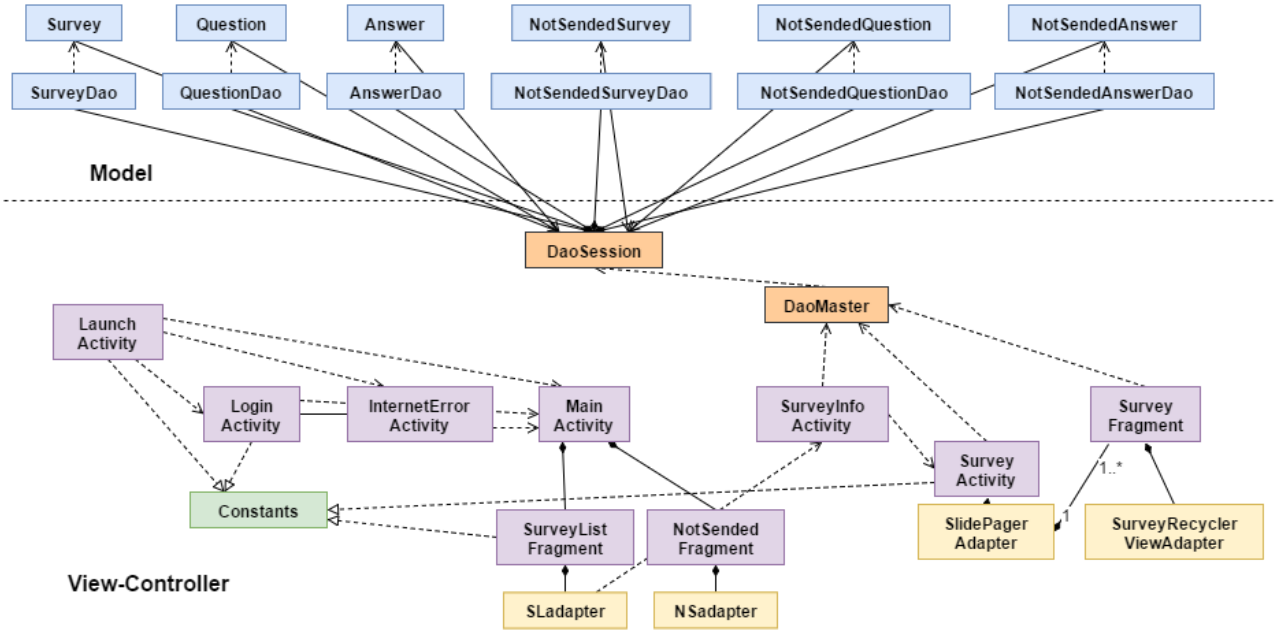

Рисунок 2 – Схема классов на архитектуре MVC

Проанализировав классы, созданные на основе архитектуры MVC, была создана схема классов для VIPER, изображённая на рисунке 3: синим цветом обозначены Entities, жёлтым – Interactors, зелёным – Presenters, красным – Routers, фиолетовым – Views, оранжевым – сервисы для работы с сервером и локальной базой данных.

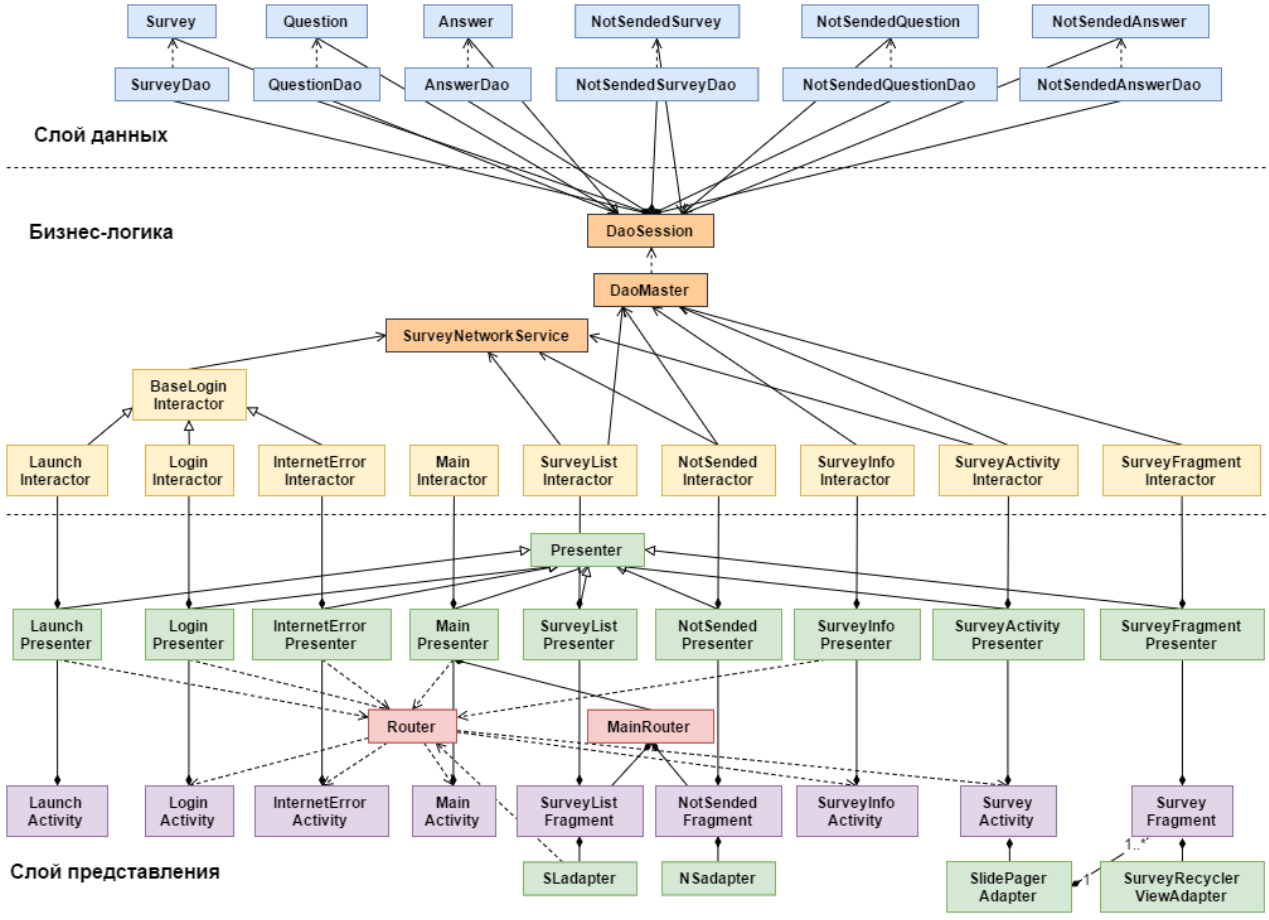

Рисунок 3 – Схема классов на архитектуре VIPER

Созданная схема классов упрощает понимание компонентов программы и значительно упрощает написание кода, несмотря на резкое увеличение количества программных классов. Незнакомому с VIPER разработчику достаточно прочесть о частях модели, чтобы появилось базовое понимание схемы классов на этой архитектуре.

В итоге потраченный временной ресурс для перехода на новую архитектуру компенсирован логичной структурой проекта и поддержкой кода разработчиками. VIPER помогает нам быть более точными, что касается разделения проблем, разделяя большое количество кода одного класса на несколько меньших классов. За счет поддержания единственной ответственности в каждом классе это упростит разработку классов и позволит нам более быстро реагировать на изменяющиеся требования.

Система облачного сервиса подготовки и проведения социологических опросов находится на этапе разработки, поэтому элементы системы часто изменяются. Своевременный переход мобильного приложения на архитектуру VIPER позволил сделать программный код более гибким: вносить изменения не теряя времени на поиски фрагментов кода в массивных классах, выявлять проблемы в конкретной части модуля при тестировании. Эта методология позволит легко влиться в проект новому разработчику. Целесообразно использовать VIPER и в будущих проектах.

Список литературы:

1. Введение в VIPER. [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/273061/ (дата обращения: 05.11.2016)
2. Robert Martin. The Clean Architecture. [Электронный ресурс]. – Режим доступа: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html (дата обращения: 05.11.2016)
3. Компания Рамблер. The Book of VIPER. [Электронный ресурс]. – Режим доступа: https://github.com/rambler-digital-solutions/The-Book-of-VIPER (дата обращения: 06.11.2016)