АРХИТЕКТУРА ANDROID ПРИЛОЖЕНИЯ «УЧЕТ ЛИЧНЫХ РАСХОДОВ»

Расули Рамин Мохаммадович

магистрант, Федеральное государственное бюджетное образовательное учреждение высшего образования Восточно-Сибирский Государственный Университет Технологий и управления, Россия, Улан-Удэ

Аннотация. В работе описана программная реализация и архитектуры Android приложения «Учет личных расходов». Программа позволяет сформировать сводную таблицу расходов по определённым датам.

Ключевые слова: MVP, нативное приложение, presenter, dao, activity, services, adapter.

Введение

Приложение «Учет личных расходов», рассчитан на тех людей, которые стремятся экономить личные средства и знать куда они уходят. При этом предполагается что пользователь может вести записи в любое время, поэтому приложение разрабатывается на мобильные устройства под управлением ОС Android.

Приложения под ОС Android можно разрабатывать на языках программирования Java, C#, и с 2017 года на языке Kotlin. В настоящее время предпочтительным является язык Java, так как он имеет различные библиотеки для реализации приложений, поэтому данное приложение будет реализовано на данном языке.

Архитектура приложения

В связи с тем, что проект не имеет финансовых ресурсов для реализации клиент-серверного приложения, было решено разрабатывать нативное приложение. В таких приложениях все записи хранятся на устройстве пользователя, т.е. база данных является локальной.

Нативное приложение рассчитано на параметры и свойства ОС и технические характеристики самого мобильного устройства. Преимущество нативного приложения – то, что они оптимизированы под определенные ОС, а значит будут работать корректно и быстро. Также они могут получать доступ к аппаратной части устройства, то есть могут использовать в своем функционале камеру смартфона, микрофон, геолакацию, адресную книгу и т.д. Можно настроить получение push-уведомлений.

Рассматриваемое приложение разработано на основе **MVP** шаблона (Model-View-Presenter pattern, рис. 1). [1]

Рисунок 1. Шаблон проектирования MVP

1. Модель (англ. **Model**) – хранит в себе всю бизнес-логику, при необходимости получает данные из хранилища;

2. Вид (англ. **View**) – реализует отображение данных из Model, обращается к **Presenter** за обновлением;

3. Представитель (англ. **Presenter**) – реализует взаимодействие между моделью и представителем.

Структура приложения будет состоять из двух модулей: *App-модуль* и *Core-модуль.*

App-модуль принимает данные из внутреннего хранилища и отображает их в понятом виде для конечного пользователя. Данная часть является **View** компонентом из шаблона проектирования **MVP**. Данный модуль содержит следующие основные компоненты:

− *Activity* – это набор шаблонов для предоставления пользовательского интерфейса.

− *Adapter* – посредник, который получает результат запросов из *Presenters*, и передает их на определённое представление (*Activity*).

Core-модуль содержит большую часть бизнес-логики приложения. Структура данного модуля, основывается на архитектуре обычного мобильного приложения, и содержит следующие компоненты:

− *DAO* – это слой объектов которые, обеспечивают доступ к данным, которые связывают хранилище с концепцией объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

− *Presenters* – это компоненты, которые получают запросы и передает их всем остальным компонентам приложения. В конце формирует ответ на первичный запрос и отправляет на нужный представление (*Adapter*).

− *Services* – это компоненты, которые предоставляют основную бизнес-логику. Которые расширяют готовые слои *DAO*, для того чтобы предоставить конечные данные для пользователя.

Взаимодействие рассмотренных модулей приложения приведено на рисунке 2.

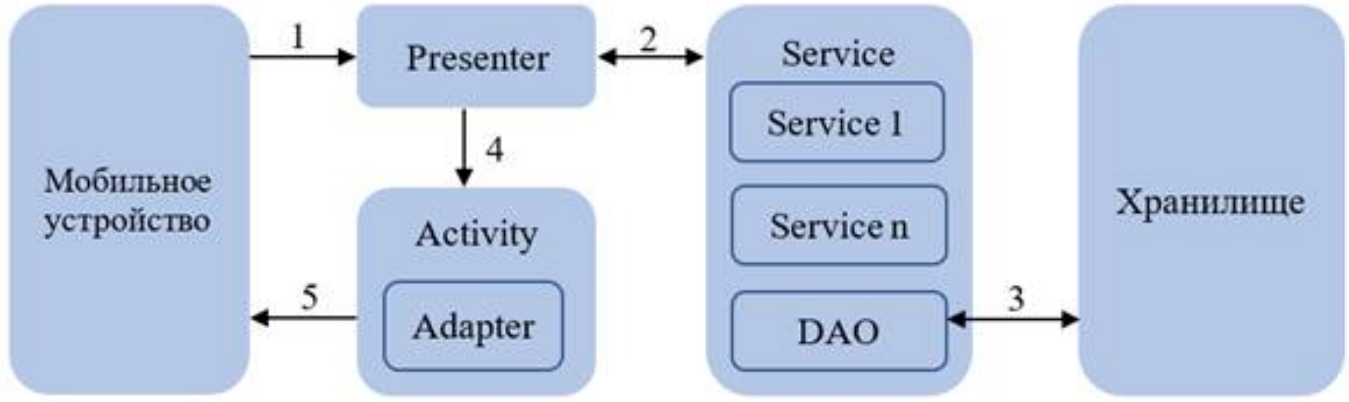

Рисунок 2. схема взаимодействия модулей приложения

Взаимодействие происходит по следующему алгоритму:

1. Presenter получает данные о том, какие кнопки были нажаты пользователем, и решает, как отреагировать на эти нажатия. В зависимости от какого действия срабатывает нужный Presenter.

2. Presenter передает параметры в нужный Service. Каждый Service отвечает за определенную обработку или бизнес-функцию приложения.

3. Если Service нужны данные из хранилища, то он обращается к слою DAO, а тот в свою очередь получает данные из хранилища и хранит их в памяти.

4. Activity по наименованию выбирает определенный готовый Adapter и наполняется данными из модели.

5. Готовый Activity, заполненный данными, отображается на мобильном устройстве.

На рисунке 3 изображена подробная структура проекта.

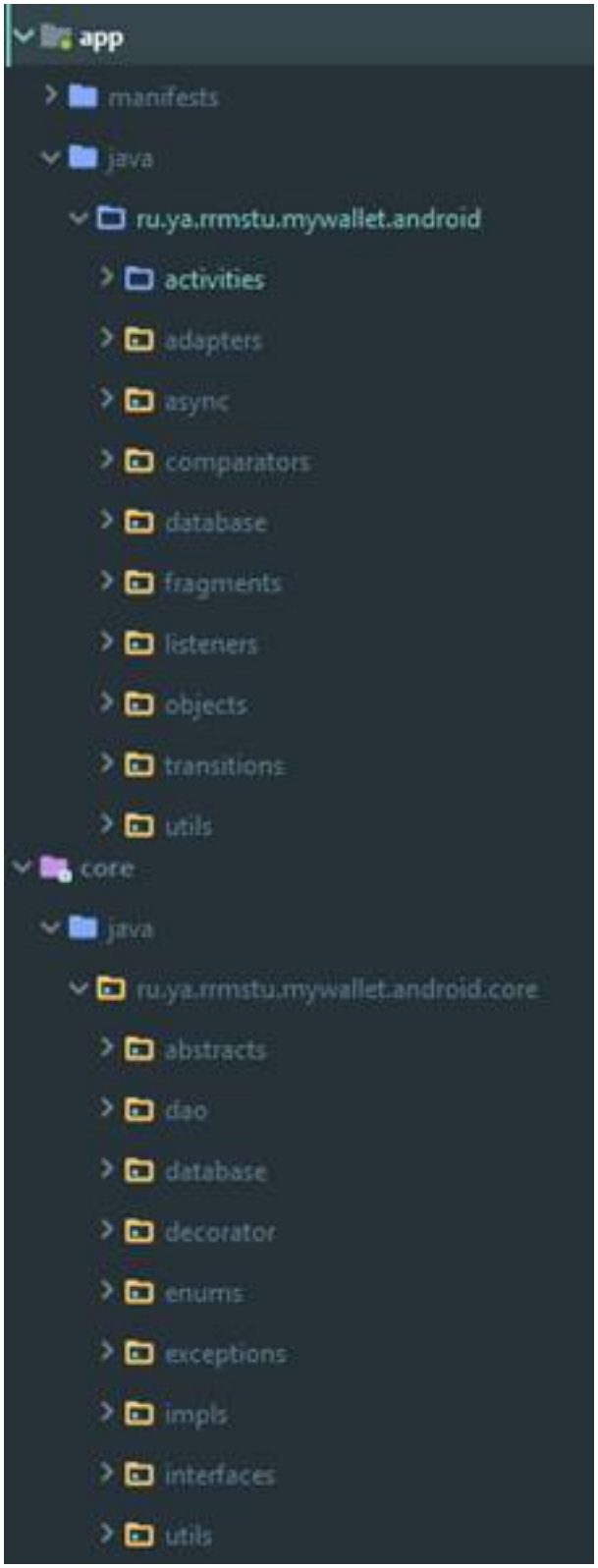

Рисунок 3. подробная структура проекта

Заключение

В статье рассмотрен стандартный метод шаблонного проектирования Android приложения MVP, а также его функциональная модель. Данный подход позволит легко написать тесты как для UI составляющей, так и для функционала, а также повысит так называемую читаемость кода. А для пользователя ускорит взаимодействие с приложением, так как скорость приложения возрастет.

Список литературы:

1. Шаблон проектирования Model-View-Presenter (MVP). – [Электронный ресурс] – Режим доступа. – URL: https://en.wikipedia.org/wiki/Model–view–presenter (Дата обращения 20.05.2018).

2. Билл Филлипс, К. Стюарт, Кристин Марсикано, Android. Программирование для профессионалов, 2017. – 688 с.

3. Джошуа Блох, Java. Эффективное программирование, 2002. – 224 с.