Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 09.03.04 Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2019г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____А.А.Замышляева
«____»_____2019 г.

Разработка Android приложения для клиентов федеральной сети секций робототехники «Лига роботов»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2019.115.ПЗ ВКР

Руководитель работы, доцент
_____/А.К. Демидов
«____»_____2019 г.
Автор работы
Студент группы ЕТ-414
_____/ Н.А. Горбачевских
«____»_____2019 г.
Нормоконтролер, доцент
_____/Н.С. Мидоночева
«____»_____2019 г.

Челябинск

2019

АННОТАЦИЯ

Горбачевских Н. А. Разработка Android приложения для клиентов федеральной сети секций робототехники «Лига роботов». – Челябинск: ЮУрГУ, ЕТ-414, 60 с., 36 ил., 1 табл., библиогр. список – 20 наим., 2 прил.

Данная работа посвящена разработке Android приложения для клиентов федеральной сети секции робототехники «Лига роботов».

В работе выполнен обзор существующих приложений подобного типа, рассмотрены требования к программе, а также выполнен обзор операционных систем подходящих и выбран инструментарий для разработки.

Спроектирована и разработана архитектура системы, включающая в себя диаграмму вариантов использования. Выполнен анализ предметной области и спроектирован пользовательский интерфейс приложения.

Программа реализована на языке программирования Kotlin. В приложении приведен текст программы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Федеральная сеть секций робототехники «Лига Роботов» занимается тем, что преподаёт робототехнику детям в возрасте 5 до 13 лет. Клиенты «Лиги Роботов» в первую очередь это родители детей, которые учатся в секции. Так как посещение секции робототехники платная услуга, то родителям важно понимать, за, что они отдают деньги. Им важно знать, как проходит процесс обучения, следить за успеваемостью своего ребенка. Этот процесс крайне сложен без прикладного программного обеспечения в виде CRM системы.

«Лига Роботов» для автоматизации большинства бизнес процессов использует «Альфа-CRM», где находится база данных детей, отчетность по оплате клиентов и оплате преподавателей. Но такая система не транслирует большинство информации своим клиентам.

Исходя из этого было предложено сделать мобильное приложение под Android, через которое можно было бы транслировать всю нужную информацию для клиентов «Лиги Роботов». Приложение может показывать информацию о филиалах секции по Челябинску, показывать полный спектр всех возможных курсов, которые проводит «Лига Роботов». Также в приложении содержится информация об успеваемости ребенка.

Таким образом такое приложение упрощает и улучшает взаимодействие работы с клиентами «Лиги Роботов».

1. ПРОСМОТР И АНАЛИЗ ТРЕБОВАНИЙ К СИСТЕМЕ. ОБЗОР ТЕХНОЛОГИЙ

1.1 Постановка задачи

Целью данной работы является создание Android приложения для клиентов федеральной сети секций робототехники «Лига роботов».

Данная работа является заказным проектом от Челябинского отделения «Лиги роботов».

Для достижения поставленной цели необходимо решить следующие задачи:

– разработать требования к приложению;

– изучить и проанализировать современные технологии и инструментарий для создания мобильных систем;

– разработка программного интерфейса для взаимодействия «Альфа-CRM»;

– разработать архитектуру приложения;

– реализовать и отладить программу.

1.2 Обзор существующих решений

1.2.1 «Дневник.ру»

«Дневник.ру» – закрытая информационная система со строгим порядком регистрации образовательных учреждений и пользователей. В системе учтены все требования безопасности и федерального закона №152 «О персональных данных», а для работы в ней потребуется только компьютер с доступом в интернет. «Дневник.ру» доступен 24 часа в сутки, 7 дней в неделю, 365 дней в году.

Для родителей в данной информационной системе существуют некоторый функционал.

1.2.1.1 Дневник ученика

Электронный дневник – это доступ в режиме реального времени к любой учебной информации. Следите за изменениями в расписании, текущими и итоговыми достижениями вашего ребёнка (см. рисунок 1.1).

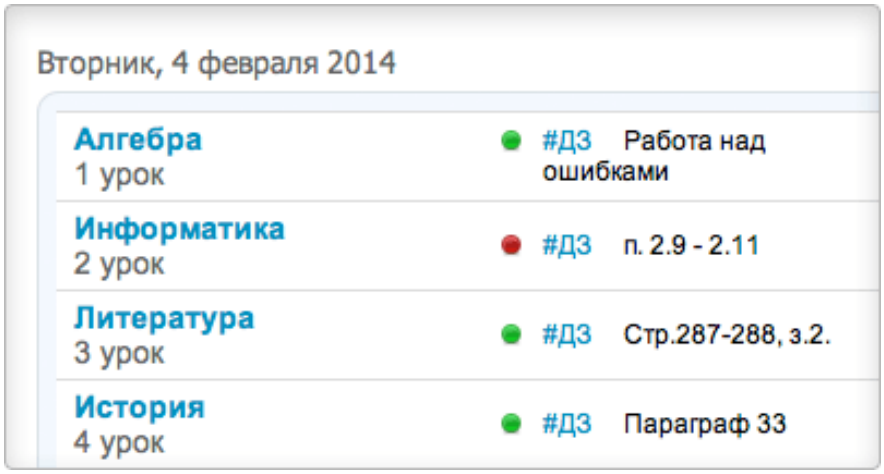

Рисунок. 1.1 – Дневник ученика

1.2.1.2 Домашние задания

Полезный инструмент для контроля и отслеживания выполнения домашних заданий вашего ребёнка (см. рисунок 1.2).

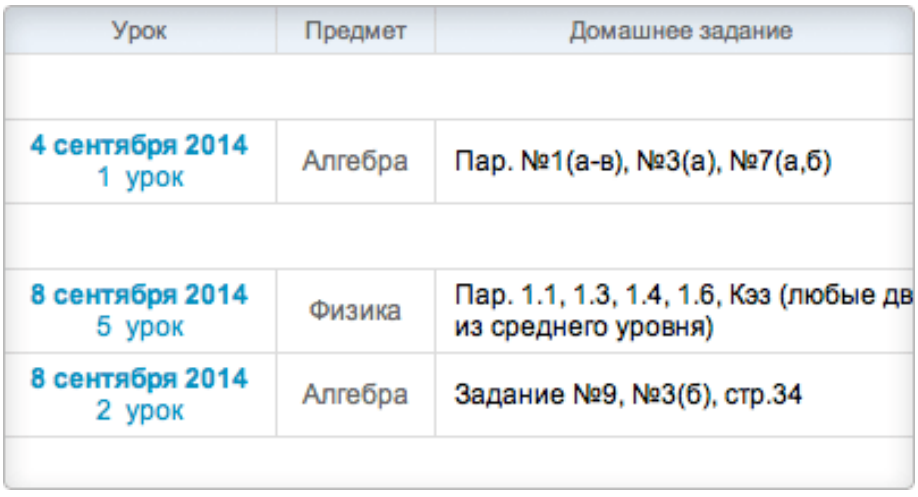

Рисунок 1.2. – Домашние задания ученика

1.2.1.3 Уведомления об учёбе

Ежедневные оповещения о ходе учебного процесса и всех важных событиях школьной жизни ребёнка, поступающие на ваши мобильный телефон или email (см. рисунок 1.3).

Рисунок 1.3 – Домашние задания ученика

1.2.1.4 Общение

Дневник.ру – это закрытая и современная среда для общения. Можно создавать тематические форумы, обмениваться фотографиями с мероприятий, связываться в удобное время с любым родителем или преподавателем (см. рисунок 1.4).

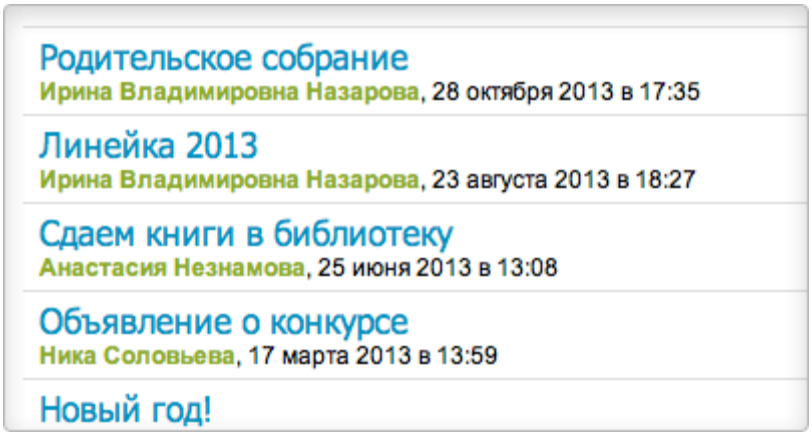

Рисунок 1.4 – Общение родителей с преподавателями

1.2.2 «SmileS.Школьная карта»

«SmileS.Школьная карта» – система безопасности школы и коммуникаций между учениками, учителями и родителями. Разработчик продукта компания «Смайлс.Образование». Проект «SmileS.Школьная карта» ориентирован на родителей, чтобы сделать процесс обучения более контролируемым и управляемым с их стороны (см. рисунок 1.5).

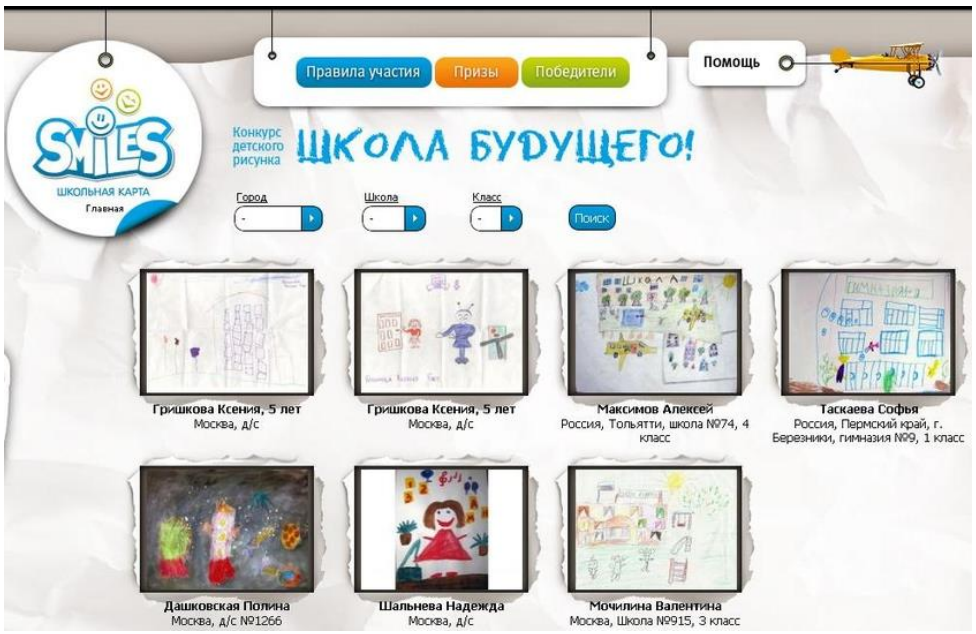

Рисунок 1.5 – Иллюстрация приложения «SmileS.Школьная карта»

1.3 Требования к программе

1.3.1 Требования к функциональным характеристикам

Приложение должно представлять совокупность программных средств для решения следующих задач:

– получение всей информации о занятиях по робототехнике;

– получение информации о местонахождении филиалов;

– получение информации о преподавателях, которые преподают в различных филиалах;

– извещение об оплате, задолженности;

– возможность произвести оплату за занятия.

1.3.2 Требования к надежности

Основные требования к надежности следующие:

– обеспечение сохранности и конфиденциальности информации, хранящейся на клиенте и сервере;

– осуществление корректной авторизации и блокировки некорректных действий пользователя при работе с системой.

– обеспечение корректного вывода информации с сервера.

1.3.3 Требования к составу и параметрам технических средств

Система должна работать на мобильных устройствах, совместимых с операционной системой Android 4.2 и выше, с установленным приложением Google Services.

1.3.4 Требования к программной документации

Разрабатываемое приложение должно включать справочную информацию о работе системы и подсказки пользователю.

1.4 Обзор мобильных операционных систем

Поскольку разрабатываемое приложение является мобильным, необходимо определиться с выбором операционной системы, на которой будет выполняться разработка клиентской части данного приложения.

Рассмотрим самые широко распространенные мобильные операционные системы.

1.4.1 Операционная система Android

Android – операционная система, разработанная на базе Linux компанией Open Handset Alliance при поддержке Google.

В числе преимуществ Android можно назвать гибкость, открытые исходные коды, высокое быстродействие, отсутствие затруднений при взаимодействии с сервисами Google и многозадачность.

Одним из недостатков Android является наличие множества актуальных версий, поскольку для версии для многих устройств выходят с серьезным опозданием, из-за чего разработчикам приходится создавать приложения, ориентированные на старые и устаревающие версии. Открытость кода имеет и свою обратную сторону – снижение уровня безопасности.

1.4.2 Операционная система iOS

iOS – операционная система, разработанная компанией Apple.

Преимуществами iOS являются удобство пользования, качественная служба поддержки, регулярные обновления.

Одним из основных недостатков iOS является ее распространение только лишь на продуктах компании Apple. Следует также упомянуть заблокированный характер этой операционной системы и отсутствие многозадачности.

1.5 Инструментарий разработки

Для написания приложения было принято решение использовать язык программирования Kotlin. Так как требуется разработать мобильное приложение под Android, для этого используется специальная IDE Android Studio. Для хранения и управления версиями проекта был использован GitLab – сайт и система управления репозиториями кода для Git. А также для красивого и приемлемого вида приложения был разработан и нарисован дизайн в специальном онлайн сервисе для разработки интерфейсов Figma.

1.5.1 Figma

Figma – онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени. Сервис имеет широкие возможности для интеграции с корпоративным мессенджером Slack и инструментом для высокоуровневого прототипирования Framer. Позиционируется создателями как основной конкурент программным продуктам компании Adobe.

Figma подходит как для создания простых прототипов и дизайн-систем, так и сложных проектов (мобильные приложения, порталы). В 2018 году платформа стала одним из самых быстро развивающихся инструментов для разработчиков и дизайнеров.

1.5.2 Android Studio

Android Studio – это интегрированная среда разработки (IDE) для работы с платформой Android. IDE находилась в свободном доступе начиная с версии 0.1, опубликованной в мае 2013, а затем перешла в стадию бета-тестирования, начиная с версии 0.8, которая была выпущена в июне 2014 года. Первая стабильная версия 1.0 была выпущена в декабре 2014 года, тогда же прекратилась поддержка плагина Android Development Tools (ADT) для Eclipse.

Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, – официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, OS X и Linux. 17 мая 2017, на ежегодной конференции Google I/O, Google анонсировал поддержку языка Kotlin, используемого в Android Studio, как официального языка программирования для платформы Android в добавление к Java и C++.

1.5.3 Язык Kotlin

Kotlin – это статически типизированный язык программирования, работающий поверх JVM и разрабатываемый компанией JetBrains. Также компилируется в JavaScript, и в исполняемый код ряда платформ через инфраструктуру LLVM. Язык назван в честь острова Котлин в Финском заливе, на котором расположен город Кронштадт.

1.5.4 JSON

Следует отметить, что строка в JSON является практически полным аналогом одноименного типа данных в языках программирования C и Java. Это же касается и числа, с тем отличием, что в JSON оно представлено только в десятеричном формате. Использование пробелов допустимо между двумя любыми элементами синтаксиса.

1.5.5 Git

Git – распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы.

Для удобства работы с системой Git было решено использовать GitLab. GitLab - это сайт и система управления хранилищем кода для Git, обладающая дополнительными функциями: собственной вики и системой отслеживания ошибок. Программное обеспечение доступно в системе управления пакетами Omnibus.

1.6 Выбор средств разработки

При выборе мобильной операционной системы доля на рынке мобильных устройств является основным фактором.

Операционная система Android лидирует на мировом рынке мобильных телефонов с долей 87%, что делает её наиболее привлекательной для разработки приложения массового использования (см. рисунок 1.6).

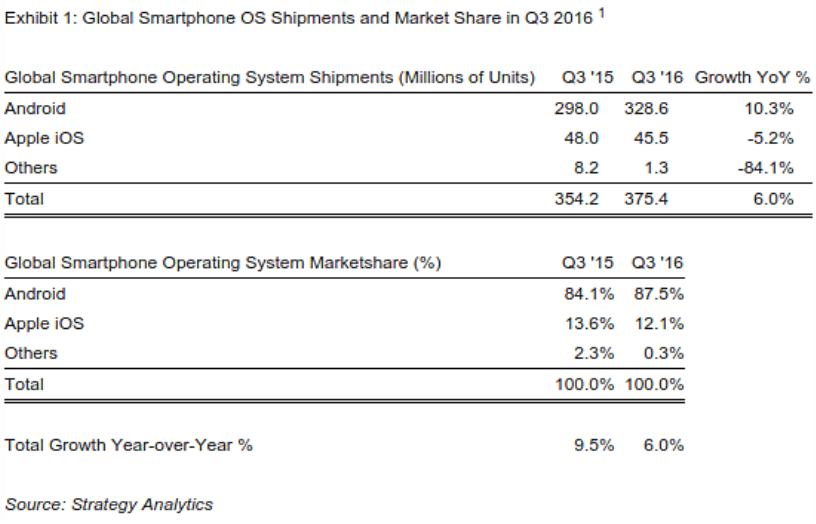

Exhibit 1: Global Smartphone OS Shipments and Market Share in Q3 2016 [1]

Global Smartphone Operating System Shipments (Millions of Units)	Q3 '15	Q3 '16	Growth YoY %
Android	298.0	328.6	10.3%
Apple iOS	48.0	45.5	-5.2%
Others	8.2	1.3	-84.1%
Total	354.2	375.4	6.0%

Global Smartphone Operating System Marketshare (%)	Q3 '15	Q3 '16
Android	84.1%	87.5%
Apple iOS	13.6%	12.1%
Others	2.3%	0.3%
Total	100.0%	100.0%

Total Growth Year-over-Year %	9.5%	6.0%

Source: Strategy Analytics

Рисунок 1.6 – Статистика, отображающая долю рынка каждой операционной системы и размеры поставок мобильных устройств

Количество пользователей в России на момент сентября 2017 года Android – 67.98%, iOS – 28.72%, Windows – 1.71%.

Этих данных достаточно чтобы понимать картину потребностей большинства пользователей. Мобильные устройства гораздо проще для взаимодействия с различными рабочими процессами. В связи с этим встала большая потребность в производстве программных продуктов на гаджеты и мобильные устройства.

1.7 Выводы по разделу

В данном разделе была проведен обзор мобильных операционных систем актуальных для данного момента времени, а также для дальнейшей работы с приложением. Выбор остался за операционной системой android и дальнейшей разработке под android. Были рассмотрены другие приложения со схожей тематикой, таких как «Дневник.ру» и «Smiles.Школьная карта». Была проделана работа по составлению требований к программе:

– требования к функциональности.

– требования к надежности.

– требования к составу и параметрам технических средств.

– требования к программной документации.

Также в разделе был описан весь используемый инструментарий разработки, который необходим для достижения цели.

2. РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ

2.1 Диаграмма вариантов использования приложения

Для построения диаграмм архитектуры приложения использовался язык графического описания UML (Unified Model Language). Данный язык очень удобен для представления общих понятий, таких как, класс, обобщение, поведение.

На рисунке 2.1 приведена диаграмма вариантов использования системы в целом, раскрывающая, какие взаимодействия и типы пользователей предусмотрены в приложении.

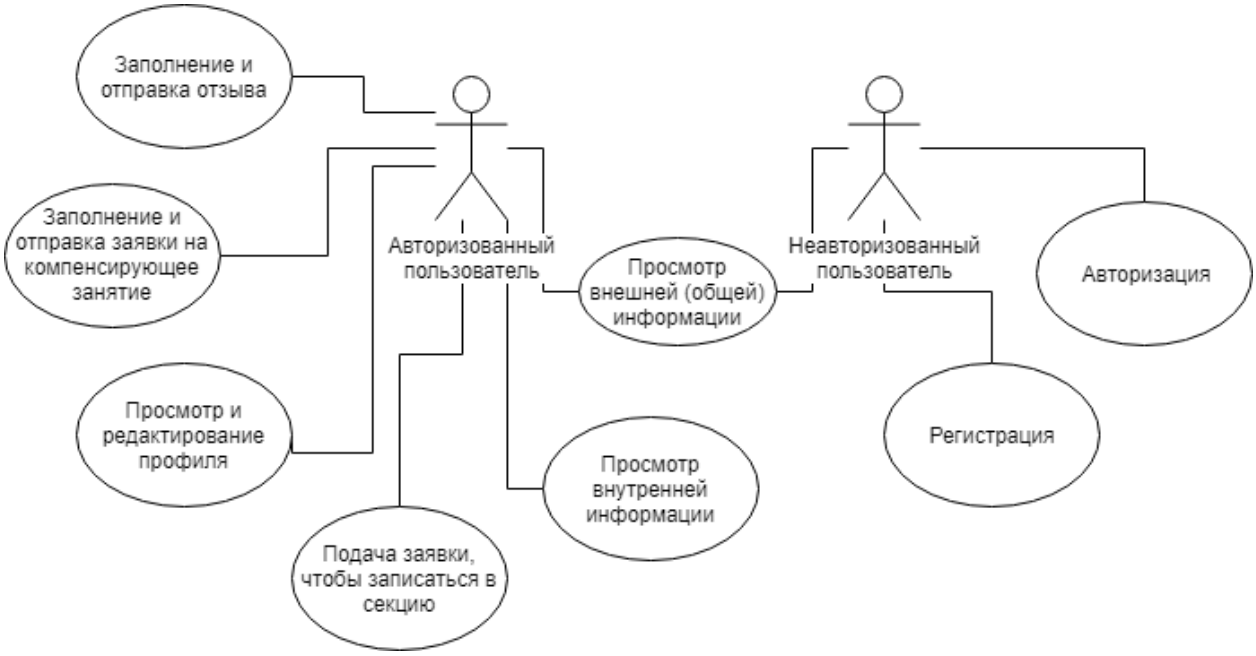

Рисунок 2.1 – Диаграмма вариантов использования приложения

2.1.1 Вариант использования: «Авторизация»

Неавторизованный пользователь имеет доступ только к просмотру общей информации, а также у него закрыта большая часть функционала. Чтобы получить доступ к этому функционалу, нужно пройти процедуру авторизации. Пользователь должен ввести в предложенное поле свой номер телефона, после чего ему отправиться смс-уведомление с кодом, код нужно

будет ввести в специальное предложенное поле. После этих манипуляций пользователь войдет в приложение.

2.1.2 Вариант использования: «Просмотр внешней (общей) информации»

Неавторизованный и авторизованный пользователь может в приложении нажать на кнопку с информацией о «Лиге Роботов», он попадет на специальное окно с исчерпывающей информацией о секции.

2.1.3 Вариант использования: «Просмотр внутренней информации»

Будучи авторизованным, пользователю будет предложено множество информации, которую он может изменять и просматривать. Например: информация о балансе, информация об услугах «Лиги Роботов» или информация о профиле.

2.1.4 Вариант использования: «Просмотр и редактирование профиля»

Авторизованный пользователь может изменять данные о своем профиле. В окне с профилем для этого есть специальная кнопка редактирования профиля.

2.1.5 Вариант использования: «Подача заявки, чтобы записаться в секцию»

Авторизованный и не авторизованный пользователь имеет возможность через приложение подать заявку на посещение секции по робототехнике.

2.1.6 Вариант использования: «Заполнение и отправка заявки на компенсирующее занятие»

Часто бывает так, что ребенок пропускает занятие по какой-либо причине. Авторизованный пользователь имеет возможность заполнить и отправить заявку на компенсирующее занятие. Компенсирующее занятие –

это занятие, которое нацелено на восстановление знаний и материала, пройденного на обычном занятии.

2.1.7 Вариант использования: «Заполнение и отправка отзыва»

Федеральная сеть секций робототехники «Лига Роботов» всегда собирает отзывы у своих клиентов, для дальнейшего улучшения различных областей работы. Для этого в приложении создано специальная форма. Авторизованный пользователь может заполнить и отправить эту форму.

2.2 Диаграмма архитектуры

Для разработки архитектуры приложения было принято решение использовать шаблон Redux (см. рисунок 2.2). Redux – шаблон проектирования для JavaScript, предназначенный для управления состоянием приложения. Чаще всего используется в связке с React или Angular для разработки клиентской части. Содержит ряд инструментов, позволяющих значительно упростить передачу данных хранилища через контекст.

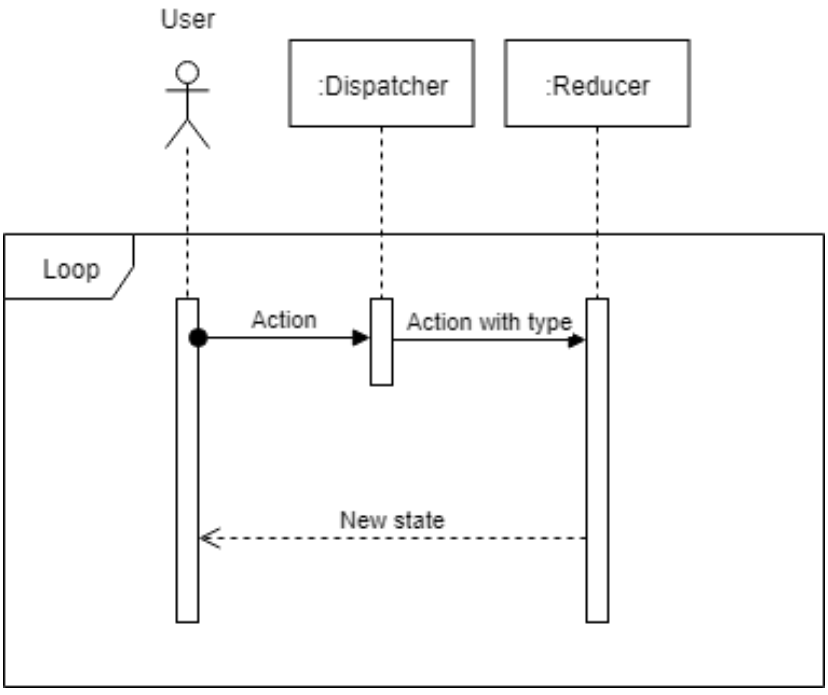

Рисунок 2.2 – Диаграмма последовательности Redux архитектуры

2.3 Проектирование интерфейса приложения

В ходе работы было спроектировано некоторое количество экранов интерфейса программы:

2.3.1 Стартовое окно приложения.

Стартовое или основное окно приложения – это окно, которое видит пользователь перед собой, запуская приложение (см. рисунок 2.3). Оно предлагает авторизоваться в приложении, а также перейти на окно с информацией о школе робототехники.

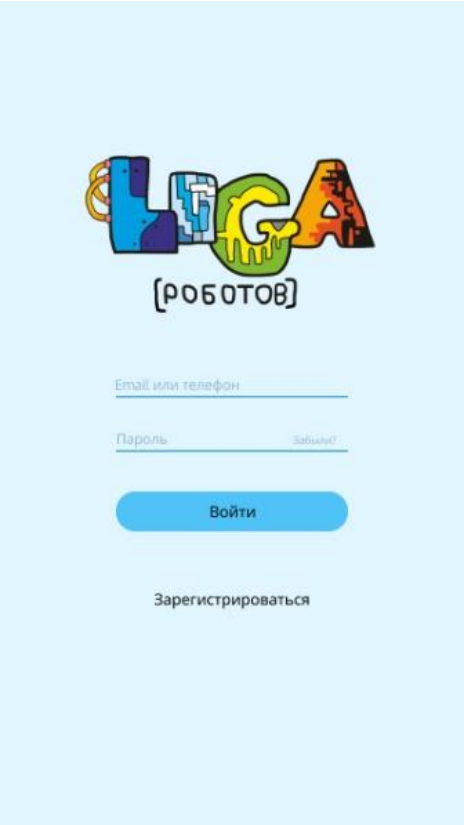

Рисунок 2.3 – Стартовое окно приложения

2.3.2 Окно информации приложения.

В окне информации приложения отображается основная информация о «Лиге Роботов», представлены иллюстрации с занятий робототехники, а также существует возможность записаться в секцию (см. рисунок 2.3).

Рисунок 2.4 – Окно информации приложения

2.3.3 Окно клиента

Окно клиента в приложении предлагает ознакомиться пользователю со своим профилем. Конкретно ему предлагается такая информация как: ФИО ребенка, филиал, расписание, связь с администратором, пропуски ребенка, информация о задолженности по оплате, информация по компенсирующим занятиям, возможность оставить отзыв, а также прикрепить фото своего ребенка (см. рисунок 2.4).

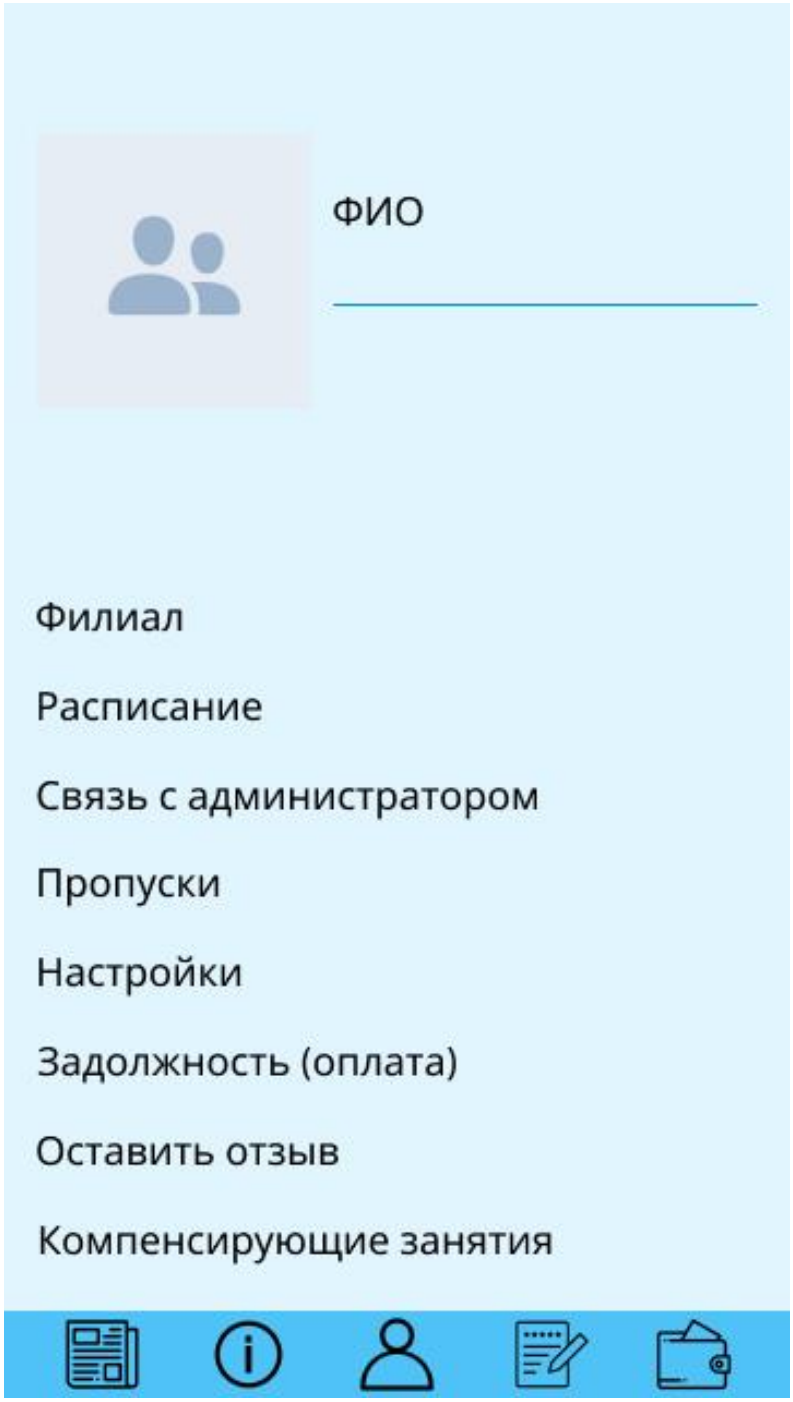

Рисунок 2.5 – Окно клиента

2.3.4 Окно контактов

Окно контактов в приложении нужно для представления данных, о контактах относительно выбранного филиала. Пользователь может выбрать нужный ему филиал, он отобразиться на карте. После выборки будет представлена информация об адресе филиала, преподавателе, который ведет занятие в филиале и его номер телефона (см. рисунок 2.5).

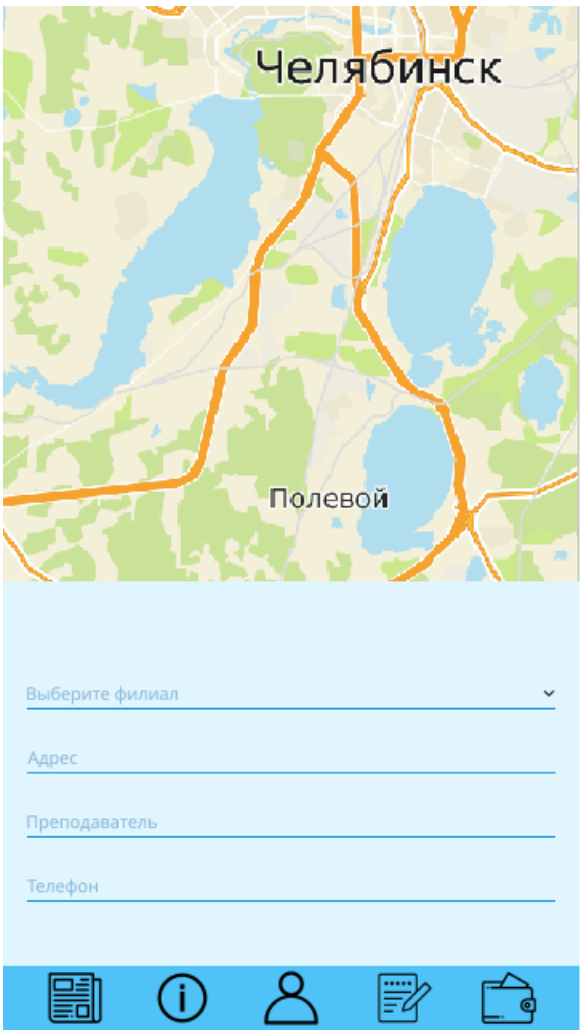

Рисунок 2.6 – Окно контактов

2.3.5 Окно платежей и документов

В этом окне пользователю предлагается просмотреть информацию о договоре и услугах, которые предоставляются секцией, возможность просмотреть список оплат, текущий баланс, а также возможность заплатить сразу через приложение (см. рисунок 2.6).

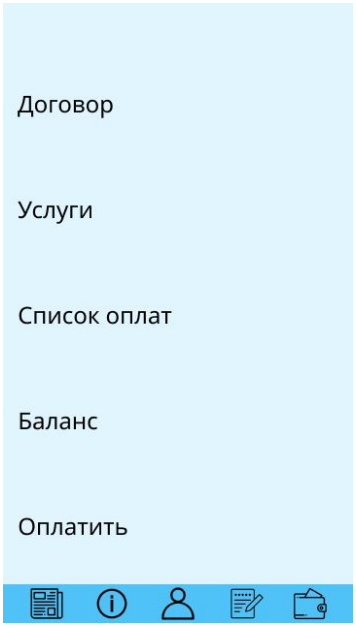

Рисунок 2.7 – Окно платежей и документов

2.3.6 Окно новостей

Окно новостей представляет обычную новостную ленту. Новости робототехники и самой секции (см. рисунок 2.7).

Рисунок 2.8 – Окно новостей

2.3.7 Окно расписания занятий

В этом окне пользователь может узнать информацию о расписании своих занятий на весь месяц. В окне помечаются дни, когда будут занятия или когда были или отдельные какие-то мероприятия. Также есть вся информация по определенному занятию (см. рисунок 2.8).

Рисунок 2.9 – Окно расписания занятий

2.3.8 Окно компенсирующих занятий

В данном окне пользователь может записаться на компенсирующие занятия (см. рисунок 2.9). Компенсирующие занятия – это занятия, на которые родитель записывает своего ребенка, в случае пропуска обычного занятия.

Рисунок 2.10 – Окно компенсирующих занятий

2.3.9 Окно заполнения отзыва

В данном окне пользователь может оставить отзыв о секции робототехники. В окне пользователю представлена возможность выбора темы отзыва и написать сам отзыв (см. рисунок 2.10).

Рисунок 2.11 – Окно заполнения отзыва

2.4 Выводы по разделу

В данном разделе разработана архитектура системы, включающая в себя диаграмму вариантов использования, диаграмму архитектуры. Был спроектирован и разработан интерфейс приложения.

Диаграмма вариантов использования (прецедентов) включает в себя все возможные варианты использования системы; диаграмма архитектуры описывает внутреннее взаимодействие основных блоков приложения.

Диаграммы описаны с помощью языка UML. При разработке системы был использован паттерн Redux.

3. РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Файловая структура проекта

Первоначально для разработки приложения был создан проект с «Empty Activity». «Activity» – это компонент приложения, который выдает экран, и с которым пользователи могут взаимодействовать для выполнения каких-либо действий, например, набрать номер телефона, сделать фото, отправить письмо или просмотреть карту. Каждой операции присваивается окно для прорисовки соответствующего пользовательского интерфейса. Обычно окно отображается во весь экран, однако его размер может быть меньше, и оно может размещаться поверх других окон (см. рисунок 3.1).

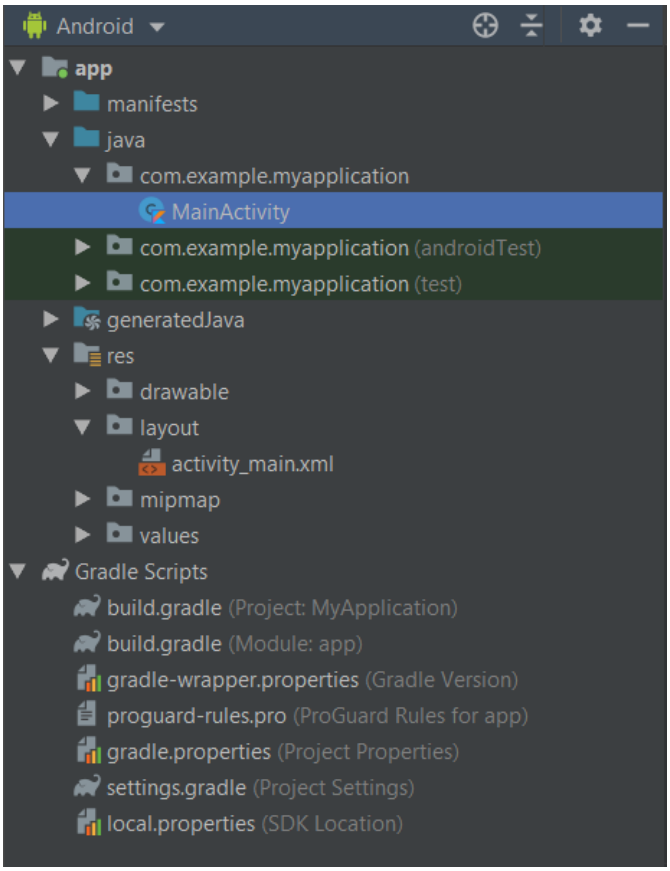

Рисунок 3.1 – Файловая структура после создания проекта

Каталог res нужен для хранения всех ресурсов, которые относятся к проекту. В нем находятся иконки приложения, файлы с расширением xml,

относящиеся к экрану приложения, различная графика, которая располагается в проекте, а также заданные цвета и шрифты.

Каталог manifests содержит AndroidManifest.xml. Файл манифеста содержит важную информацию о приложении, которая требуется системе Android. Только получив эту информацию, система может выполнить какой-либо код приложения. Среди прочего файл манифеста выполняет следующие действия:

– он задает имя пакета Java для приложения. Это имя пакета служит уникальным идентификатором приложения;

– он описывает компоненты приложения – операции, службы, приемники широковещательных сообщений и поставщиков контента, из которых состоит приложение. Он содержит имена классов, которые реализуют каждый компонент, и публикует их возможности (указывает, например, какие сообщения Intent они могут принимать). На основании этих деклараций система Android может определить, из каких компонентов состоит приложение и при каких условиях их можно запускать;

– он определяет, в каких процессах будут размещаться компоненты приложения;

– он объявляет, какие разрешения должны быть выданы приложению, чтобы оно могло получить доступ к защищенным частям API-интерфейса и взаимодействовать с другими приложениями;

– он также объявляет разрешения, требуемые для взаимодействия с компонентами данного приложения;

– он содержит список классов Instrumentation, которые при выполнении приложения предоставляют сведения о профиле и прочую информацию. Эти объявления присутствуют в файле манифеста только во время разработки и отладки приложения и удаляются перед его публикацией;

– он объявляет минимальный уровень API-интерфейса Android, который требуется приложению;

– он содержит список библиотек, с которыми должно быть связано приложение.

Для того чтобы сделать архитектуру redux, нужно подготовить специальную файловую структуру в проекте (см. рисунок 3.2). Для этого созданы следующие каталоги.

1. Redux

В каталоге Redux содержится основа архитектуры: классы Action и Store, интерфейс Middleware и функция Reducer.

2. State

Содержит основные состояния приложения.

3. Navigation

Содержит все используемые фрагменты приложения, класс MainActivity и класс BaseFragment, переопределяющий класса фрагмента.

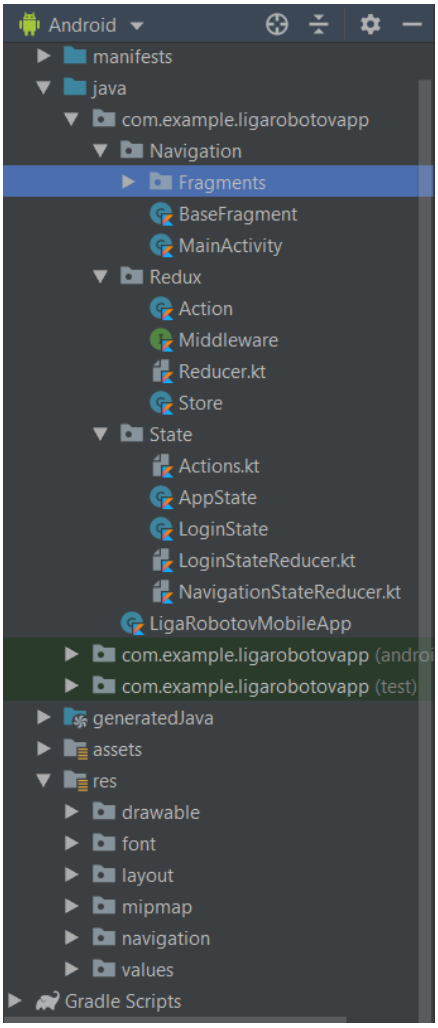

Рисунок 3.2 – Итоговая файловая структура проекта

3.2 Сборка проекта в APK файл

Во время запуска проекта Android studio собирает весь проект в файл формата apk. Выполняет эту функцию специальный сборщик Gradle. Gradle – система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но предоставляющая DSL на языках Groovy и Kotlin вместо традиционной XML-образной формы представления конфигурации проекта. Общая схема сборки apk файла показана на рисунке 3.3.

3.2.1 Ресурсы приложения

Начинается сборка с компиляции ресурсов приложения. Компиляцией ресурсов Сборка осуществляется с помощью специальной утилит AAPT2 и AAPT (Android Asset Packaging Tool). Очень важно следовать файловой структуре каталога res, так как показано на рис.3.2., потому как AAPT выдаст ошибку. Задача утилиты упаковать ресурсы в красивые и понятные для дальнейшей системы сборки файлы с расширением .flat и создать класс R, который используется для обращения к ресурсам. На этом этапе сформируется APK файл из ресурсов и AndroidManifest.xml, но работать пока не будет, так как много частей ещё отсутствует.

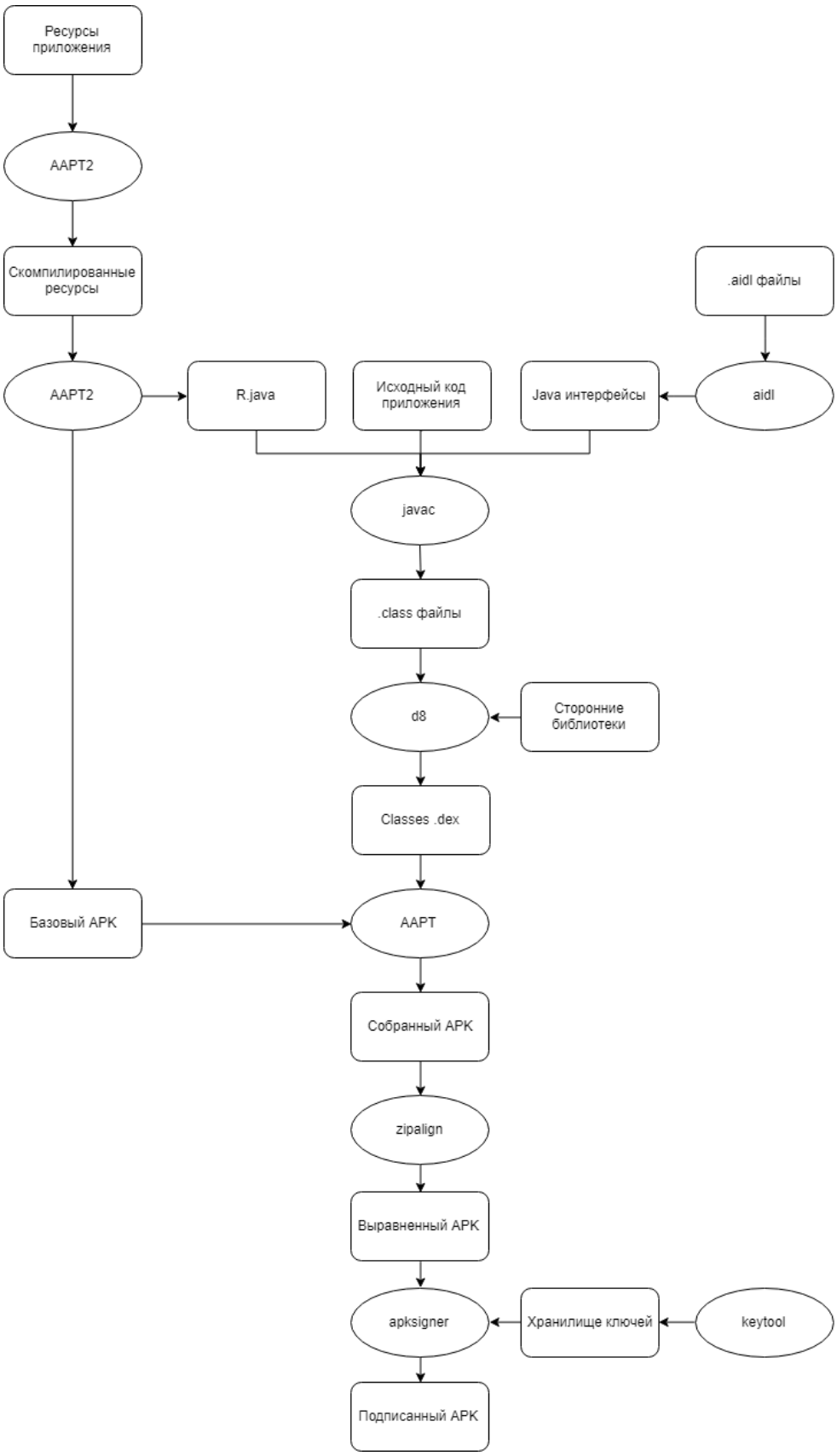

Рисунок 3.3 – Общая схема сборки APK файла

3.2.2 AIDL

Это интерфейсы, которые вы можете написать на Java с немного необычным синтаксисом. Такие интерфейсы нужны для описания взаимодействия между различными процессами, например, когда нужно, чтобы сервис (Service) приложения работал не просто в отдельном потоке, а именно в отдельном процессе. С помощью утилиты aidl соответствующие классы компилируются в Java интерфейсы. Их программисты редко используют напрямую, так как AIDL не потокобезопасны, тем более есть более простая альтернатива (Messenger) этим классам, которая уменьшает как количество написанных вами строчек кода, так и увеличивает надёжность вашего приложения.

3.2.3 Компиляция Java

На этом этапе обычный javac из вашего JDK компилирует исходные данные в JVM bytecode. На вход компилятор принимает весь ваш исходный код вместе со сгенерированными AIDL интерфейсами и R классом. На выходе видно привычные .class файлы, но на этом процесс компиляции не заканчивается, так как Dalvik VM не может интерпретировать java bytecode, так как использует dex bytecode.

3.2.4 Сторонние библиотеки

Чтобы использовать в приложении карты, специальные файлы для навигации и различные сторонние решения, нужно подключать их через специальный файл сборщика gradle (см. рисунок 3.4). Это могут быть файлы с расширением .jar и .aar, которые скачивает сам сборщик после указания специальных указаний в зависимостях. С помощью утилиты d8 вы можете преобразовать ваши входные бинарные файлы в исполняемый .dex файл. После подключения библиотек сгенерированный бинарный файл .dex включается в архив APK файла.

Рис.3.4 – Пример подключения сторонних библиотек

3.2.5 Выравнивание APK

Когда APK уже почти готов к выпуску, остаётся его, по сути, только подписать. Но прежде есть один важный момент. Каждый раз, когда происходит запуск приложения, после нажатия на кнопку “Run”, среда разработки Android Studio не только генерирует APK архив, но и немножко его оптимизирует с помощью утилиты zipalign. Задача этой утилиты состоит в том, чтобы все несжатые данные начинались с определенного выравнивания относительно начала файла. В частности, это приводит к выравниванию всех несжатых данных по 4-байтовым границам. Преимущество заключается в уменьшении используемого объема оперативной памяти при запуске приложения. Выравнивание APK архивов положительно сказывается на производительности самой операционной системе Android.

3.2.6 Генерация ключа

С помощью утилиты keytool можно сгенерировать уникальный ключ. Каждый APK файл должен быть подписан уникальным ключом из хранилища (файл формата keystore или .jks). Подпись подтверждает автора данного приложения, а если кто-то добавит в приложение вредоносную часть, то система Android не разрешит установить это приложение, так как ключ злоумышленника не будет подходить под имя пакета.

3.2.7 Подпись приложения

Это очень простая операция, которая происходит посредством утилиты apksigner. Эта программа создает в apk файле папку META-INF, в которую помещается три файла: ANDROIDD.RSA, ANDROIDD.SF, MANIFEST.MF. Это те файлы, которые необходимы для идентификации вас как истинного разработчика этого приложения.

3.3 Реализация архитектуры Redux

Во время использования приложения пользователь сталкивается с большим количеством динамики. Так как архитектурным решением был выбран шаблон Redux, было описаны следующие классы: store, action и appstate, которые служат для выполнения должной динамики приложения. Также был написан специальный интерфейс middleware и функция reducer

3.3.1 Класс «AppState»

Класс «AppState» содержит в себе два поля navigationStack и loginState (см. рисунок 3.5). Сам класс отвечает за состояние класса приложения. Состояние класса меняется в зависимости от того как использует приложение пользователь. Поле navigationStack может принимать несколько действий пользователя связанных с навигацией по приложению. Например, если пользователь переходит с одного окна на другое navigationStack принимает значение определенного действия отвечающего за переход на новое окно приложения. Поле loginState отвечает за состояние – авторизован пользователь или нет и принимает соответствующие значения.

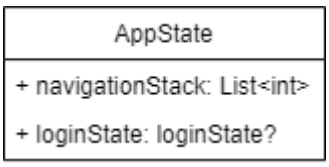

Рисунок 3.5 – Класс «AppState»

3.3.2 Класс «Store»

Класс «Store» один из самых основных классов в архитектурном паттерне redux (см. рисунок 3.6). Он выступает в роли хранилища. «Store» содержит в себе начальное состояние и редюсер, который из начального состояния делает другое в зависимости от того как пользователь взаимодействует с приложением. Внутри класса описаны два метода subscribe, но эти методы разные. Один метод subscribe позволяет подписаться на обновление состояния, другой метод позволяет подписаться на обновление подсостояния. Метод middleware нужен для того чтобы добавить переменную-перехватчик, она нужна для того чтобы забирать данные с внешних хранилищ.

Рисунок 3.6 – Класс «State»

3.3.3 Класс «Action»

Этот класс отвечает за действия пользователя (см. рисунок 3.7). Например, пользователь нажимает на меню навигации на кнопку перехода на одно профиля, приложение просто меняет окно. Но так видит только пользователь. На самом деле внутри приложения это выглядит немного иначе. Пользователь своим действием посылает определенный action, которое далее отправляется в reducer. У класса «Action» существует два поля, первое поле отвечает за тип действия, второе за объект. Также для этого класса описаны специальные действия, например действие NAVIGATE_TO, нужно для перехода на следующее окно.

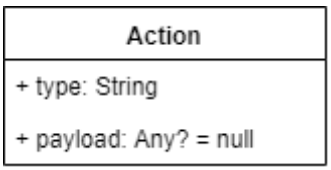

Рисунок 3.7 – Класс «Action»

3.3.4 Функция «Reducer»

Функция «Reducer» одна из важнейших функций паттерна архитектуры redux (см. рисунок 3.8). Она отвечает за переработку всех действий, совершаемые пользователем, например: нажатие кнопки, переход на новое окно и т.д. Работает функция по следующему принципу: функция принимает значение предыдущего состояния приложения и применяемое действия и возвращает новое состояние, которое учитывает совершенное действие пользователя.

```
fun <State : Any> combineReducers(vararg reducers: Reducer<State>) : Reducer<State> {
    return { startState: State, action: Action ->
        reducers.fold(startState) { state, reducer ->
            reducer(state, action)
        }
    }
}
```

Рисунок 3.8 – Функция «Reducer»

3.3.5 Интерфейс «Middleware»

«Middleware» или промежуточный слой нужен для того, чтобы предоставлять способ взаимодействия с действиями, которые отправляются в хранилище, но не проходят через редюсер (см. рисунок 3.9). Примеры различного использования промежуточного слоя включают ведение журнала действий, сообщения об ошибках, выполнение асинхронных запросов, а также отправку новых действий. Но это лишь часть того когда может понадобиться данный интерфейс. Для приложения он нужен, чтобы в дальнейшем забирать данные с CRM-системы, которую использует «Лига Роботов».

```
interface Middleware<State : Any> {
    fun dispatch(store: Store<State>, nextDispatcher: (action: Action) -> Unit): (action: Action) -> Unit
}
```

Рисунок 3.9 – Интерфейс «Middleware»

3.4 Реализация навигации приложения

Навигация приложения неотъемлемая часть, без которой приложение не будет корректно работать. В проекте используется некоторое количество фрагментов между которыми взаимодействует пользователь, навигация настраивает пути взаимодействия между этими фрагментами. Например, всегда имеется стартовое окно приложения, после него можно перейти на другое окно. Все это описывается с помощью Navigation Architecture Component.

3.4.1 Добавление зависимостей

Первым делом для правильной работоспособности навигации нужно добавить зависимости в блок «dependencies» в файл сборщика модуля «build.gradle». Чтобы добавить зависимость от навигации, требуется добавить репозиторий Google Maven в проект (см. рисунок 3.10).

```
allprojects {
    repositories {
        google()
        maven {
            url 'https://dl.google.com/dl/android/maven2/'
        }
    }
}
```

Рисунок 3.10 – Добавление репозитория Google Maven

Далее нужно добавить актуальные зависимости, относящиеся к навигации (см. рисунок 3.11). Все зависимости берутся с официальных источников android. Для версии проекта, который работает на языке программирования «kotlin» следует указать в конце строки «-ktx».

```
def nav_version = "2.1.0-alpha04"
implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

Рисунок 3.11 – Добавление зависимостей навигации

3.4.2 Создание фрагментов

Фрагменты – это окна будущего приложения. Кликнув правой кнопкой на каталоге с файлами классов, командой «New => Fragment => Fragment(Blank)» можно создать фрагмент (см. рисунок 3.12).

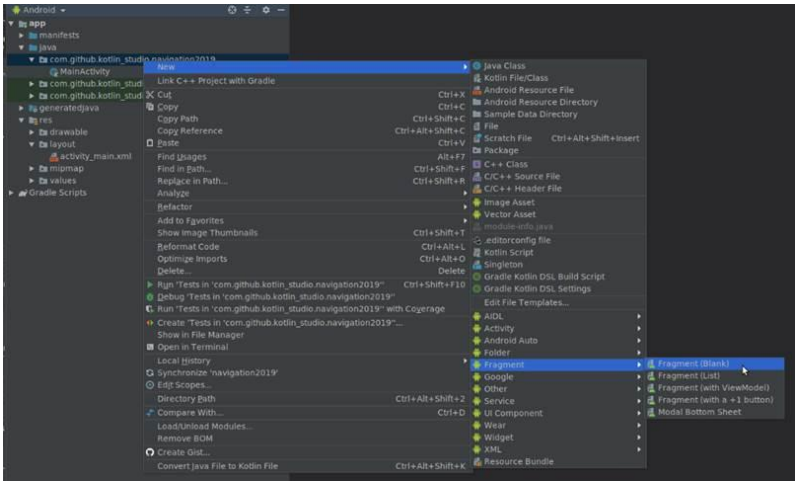

Рисунок 3.12 – Создание фрагмента

При создании фрагмента его нужно сконфигурировать: указать его имя, выбрать создавать ли к нему xml разметку, в случае если да, то выбрать имя разметки (см. рисунок 3.13).

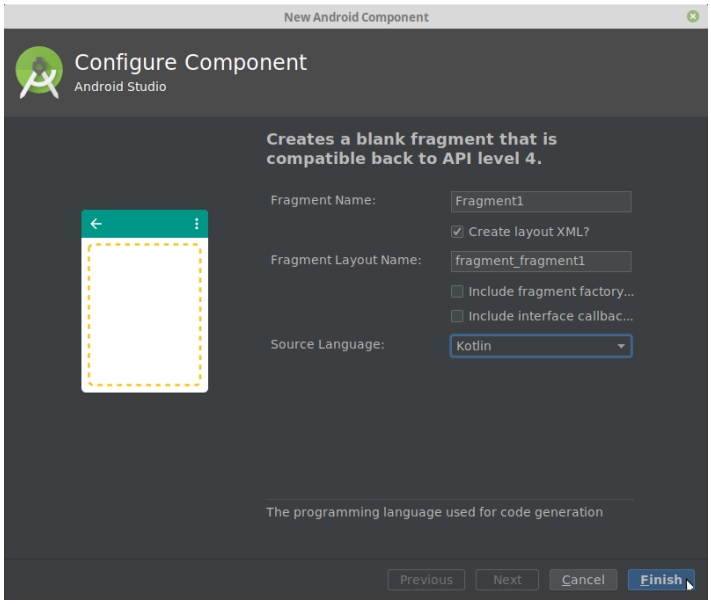

Рисунок 3.13 – Конфигурация фрагмента

IDE Android Studio создаст файл с классом фрагмента и xml-файл с макетом фрагмента. Таким образом, было создано нужное количество фрагментов.

3.4.3 Создание файла ресурсов типа «Navigation»

После подключения нужных зависимостей и создания фрагментов идет самая важная часть реализации навигации. Нужно создать, как показано на рисунке 3.14 специальный файл в каталоге ресурсов и выбрать тип «Navigation»

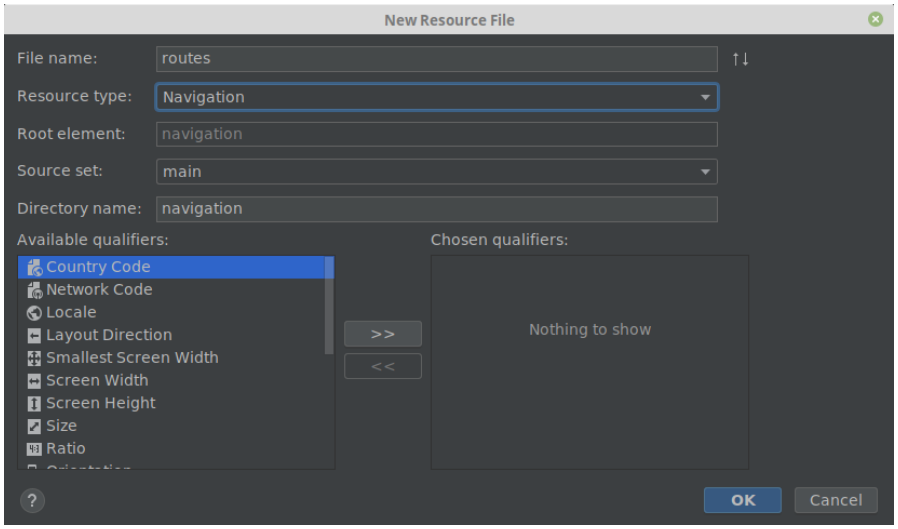

Рисунок 3.14 – Создание файла «routes» типа «Navigation»

После создания файла «routes» в нем нужно разместить, нужны фрагменты и соединить их в соответствии со спроектированной схемой. «Хватаясь» курсором мыши за точку в середине правой стороны фрагмента, можно соединять фрагменты между собой тем самым выстраивая навигационную иерархию (см. рисунок 3.15).

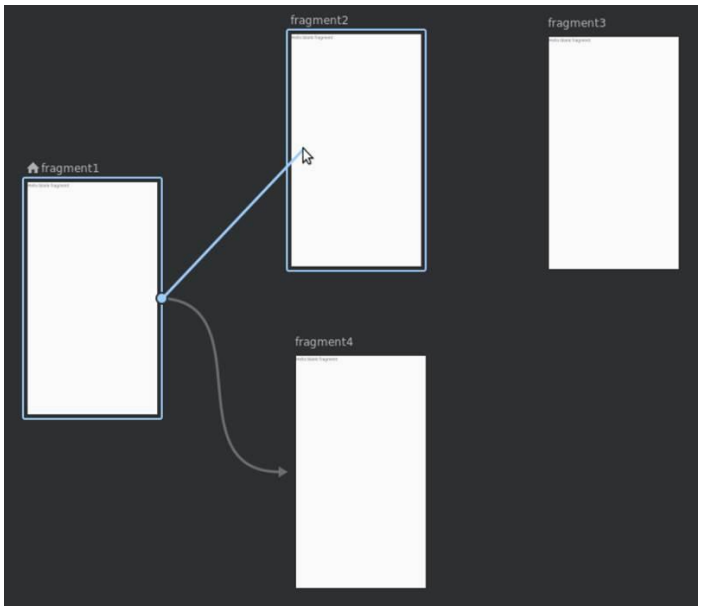

Рисунок 3.15 – Соединение фрагментов

3.4.4 Добавление виджета «NavHostFragment»

Виджет «NavHostFragmet» выступает в роли фрейма, в котором будут выводиться все фрагменты приложения. Иными словами фрагменты будут являться отображением основной активности приложения (см. рисунок 3.16).

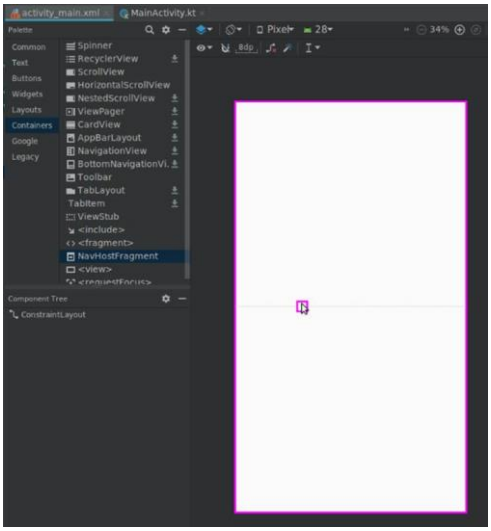

Рисунок 3.16 – Добавление виджета «NavHostFragmet» на «MainActivity» приложения

Последнее что остается сделать, это изменить идентификаторы фрагментов, задать им имена и нужные действия, применяемые внутри этих фрагментов (см. рисунок 3.17).

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:app="http://schemas.android.com/apk/res-auto"
            xmlns:tools="http://schemas.android.com/tools"
            android:id="@+id/navigation"
            app:startDestination="@+id/startFragment">
    <fragment
            android:id="@+id/startFragment"
            android:name="com.example.ligarobotovapp.Navigation.Fragments.StartFragment"
            android:label="fragment_start"
            tools:layout="@layout/fragment_start">
        <action android:id="@+id/action_startFragment_to_infoFragment"
                app:destination="@+id/infoFragment"/>
        <action android:id="@+id/action_startFragment_to_loginFragment"
                app:destination="@+id/loginFragment"/>
    </fragment>
    <fragment
            android:id="@+id/infoFragment"
            android:name="com.example.ligarobotovapp.Navigation.Fragments.InfoFragment"
            android:label="fragment_info"
            tools:layout="@layout/fragment_info">
    </fragment>
```

Рисунок 3.17 – Пример описания файла навигации

3.4.5 Реализация карт «MapBox»

Для отображения филиалов «Лиги роботов» на карте был использован сервис «MapBox». «MapBox» предлагает услуги коммерческой картографии. Чтобы подключить на фрагменте карты, первым делом нужно подключить внешние зависимости в файл сборщика проекта (см. рисунок 3.17).

```
implementation 'com.mapbox.mapboxsdk:mapbox-android-sdk:8.0.1'
implementation 'com.mapbox.mapboxsdk:mapbox-android-plugin-locationlayer:0.5.0'
```

Рисунок 3.18 – Подключение зависимостей сервиса «MapBox»

После подключения зависимостей сборщик соберет проект и можно будет воспользоваться подключенными компонентами карт на фрагменте. В основном классе фрагмента, в котором была реализована карта, был подключен API ключ. Для понимания, где находиться какой филиал на карте были созданы определенные метки с координатами филиала (см. рисунок 3.19).

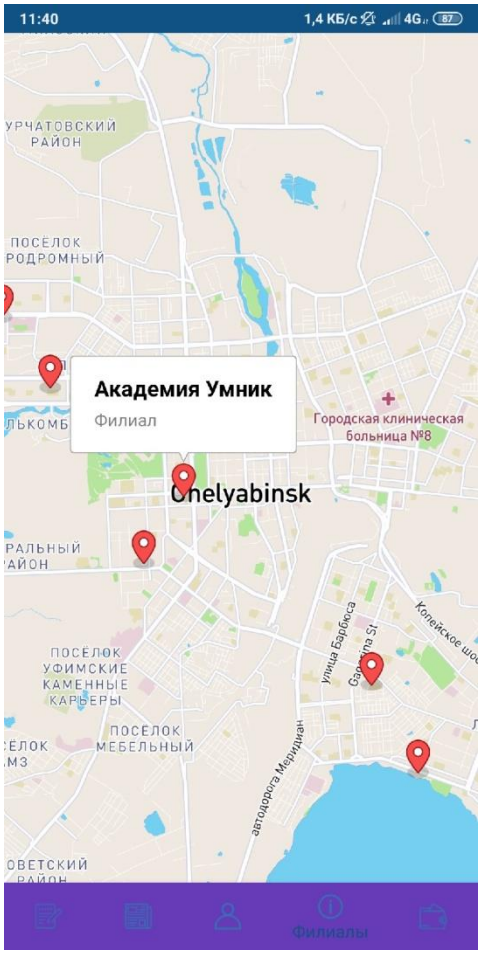

Рисунок 3.19 – Реализация карт в приложении

ЗАКЛЮЧЕНИЕ

Данная работа посвящена реализации мобильного приложения для клиентов федеральной сети секций по робототехнике «Лига Роботов». В «Лиге роботов» стояла проблема в упрощении представления информации предоставляемой клиентам секции робототехники.

В ходе проделанной работы было проанализировано несколько похожих приложений, которые могут являться конкурентами данному приложению. Из них были выявлены преимущества и недостатки. Была произведена работа с большим количеством инструментария разработки, таким как: интегрированная среда разработки «Android Studio», система контроля версий «GitLab». Был изучен язык программирования «Kotlin» и его взаимодействие с мобильными приложениями, а также был изучен архитектурный паттерн «Redux».

В ходе работы над приложением были разработаны:

– архитектура приложения;

– диаграмма использования приложения;

– пользовательский интерфейс приложения;

– первичный дизайн приложения;

– окна приложения;

– навигация приложения и взаимодействие между окнами.

По итогу разработки были выполнены все поставленные задачи по разработке приложения. Приложение решило большинство проблем связанных с упрощение представления информации клиентам «Лиге Роботов».

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Жемеров, Д. Kotlin в действии / Д. Жемеров, С. Исакова : пер. с англ. Киселев А.Н. – М.:, «ДМК Пресс», 2018. – 406 с.

2 Голощапов, А. Google Android. Программирование мобильных устройств – М.:, «БХВ-Петербург-Москва», 2012. – 448 с.

3 Дейтел, П. Android для разработчиков / П. Дейтел, Х. Дейтел : пер. с англ. Изд-во «Питер», 2016. – 512 с.

4 Харди, Б. Android программирование для профессионалов / Б. Харди, К. Стюарт, Б. Филлипс : пер. с англ. Изд-во «Питер», 2016. – 640 с.

5 Рето, М. Android 4. Программирование приложений для планшетных компьютеров и смартфонов – М.:, «Эксмо», 2013. – 816 с.

6 Гриффитс, Д. Head First. Программирование для Android – СПб.: «Питер», 2016. – 704 с.

7 Фелкер, Д. Android: разработка приложений для чайников. : пер. с англ. – М. : ООО «И.Д. Вильямс», 2012. – 336 с.

8 Медникс, З. Программирование под Android. 2-е изд. – СПб.: Питер, 2013. – 560 с.

9 Голощапов, А. Google Android: системные компоненты и сетевые коммуникации. – СПб. : «БХВ-Петербург», 2012. – 384 с.

10 Дейтел, П. Android для программистов / П. Дейтел – СПб. : «Питер», 2013. – 560 с.

11 Бирман, И. Пользовательский интерфейс. – М.: «Бюро Горбунова», 2017. – 405 с.

12 Бэнкс, А. React и Redux: функциональная веб-разработка. – СПб.: «Питер», 2018. – 336 с.

13 Уроки по Android программированию – Дата обновления: 05.07.2014. URL: https://startandroid.ru/ru/uroki.html (дата обращения: 27.03.2019)

14 Руководство по языку Kotlin. – Дата обновления: 20.05.2015. URL: https://kotlinlang.ru (дата обращения: 14.03.2019)

15 Keddit–Intro: Learn Kotlin while developing an Android App – Дата обновления: 27.01.2016. URL: https://android.jlelse.eu/learn-kotlin-while-developing-an-android-app-introduction-567e21ff9664 (дата обращения: 14.03.2019)

16 Get Started with Kotlin on Android – Дата обновления: 08.06.2014. URL: https://developer.android.com/kotlin/get-started.html (дата обращения: 09.02.2019)

17 Android Testing with Kotlin – Дата обновления: 03.02.2017. URL: https://fernandocejas.com/2017/02/03/android-testing-with-kotlin

18 Material Design: на Луну и обратно – Дата обновления: 11.03.2015. URL: https://habr.com/ru/company/redmadrobot/blog/252773 (дата обращения: 16.04.2019)

19 Material Design 2.0 – Дата обновления: 17.05.2018. URL: https://medium.com/slashdesigner/material-2-814b64c62441 (дата обращения: 24.05.2019)

20 Leiva, A. Kotlin for Android Developers – Lean Publishing, 2019. – 607с.

ПРИЛОЖЕНИЕ 1

ОПИСАНИЕ ПРОГРАММЫ

Разработка Android приложения для клиентов федеральной сети секций робототехники «Лига роботов»

2019

ОГЛАВЛЕНИЕ

П1.1 Общие сведения

Продукт мобильное Android приложение для федеральной сети секций робототехники «Лига роботов», которой будут пользоваться клиенты «Лиги роботов».

П1.2 Функциональное назначение

Данное приложение обладает следующими возможностями:

– предоставление информации пользователю о секции робототехники;

– вывод информации о профиле пользователя;

– дает возможность заполнить отзыв о секции;

– дает возможность оставить заявку, чтобы записаться в секцию;

– дает возможность оставить заявку, чтобы записаться на компенсирующее занятие в секцию;

– дает возможность получить информацию о филиалах секции;

П1.3 Используемые технические средства

Для обеспечения функционирования приложения требуютсяследующие технические средства:

– мобильный телефон на базе операционной системы Android, минимальная версия Android 4.2;

Необходимые программно-аппаратные ресурсы для запуска локального сервера представлены в таблице 1.3.

Таблица 1.3 – Программно-аппаратная поддержка

Составляющие	Требования
Операционная система	Android 4.2 и выше
Оперативная память, Гб	1
Пространство на жестком диске, Мб	10
Рабочая частота ЦПУ, МГц	1000

П1.4 Входные и выходные данные

Входными данными для приложения являются текстовая информация, вводимая пользователем в предназначенные для этого поля и информация о выборе предлагаемых опций, элементов меню, нажатии кнопок при управлении программой при помощи мыши.

На выходе программа предоставляет текстовую информацию, относящуюся к определенному запросу пользователя.

ПРИЛОЖЕНИЕ 2

ТЕКСТ ПРОГРАММЫ

2019

ОГЛАВЛЕНИЕ

П2.1 LigaRobotovMobileApp.kt класс приложения

```
package com.example.ligarobotovapp

import android.app.Application
import android.util.Log
import com.example.ligarobotovapp.Redux.Action
import com.example.ligarobotovapp.Redux.Middleware
import com.example.ligarobotovapp.Redux.Store
import com.example.ligarobotovapp.Redux.combineReducers
import com.example.ligarobotovapp.State.AppState
import com.example.ligarobotovapp.State.loginStateReducer
import com.example.ligarobotovapp.State.navigationStateReducer


class LigaRobotovMobileApp: Application() {
    val TAG = "LigaRobotovMobileApp"

    val store = Store(
        AppState(emptyList(), null),
        combineReducers(
            AppState::navigationStack, navigationStateReducer,
            AppState::loginState, loginStateReducer,
            ::AppState
        )
    )

    override fun onCreate() {
        super.onCreate()

        store.addMiddleware(object : Middleware<AppState> {
            override fun dispatch(
                store: Store<AppState>,
                nextDispatcher: (action: Action) -> Unit
            ) = { action: Action ->
                Log.d(TAG, "State: ${store.state}")
                Log.d(TAG, "Action: ${action.type}, payload: ${action.payload}")
                nextDispatcher(action)
            }
        })
    }
}
```

П2.2 Reducer.kt редюсер

```
    package com.example.ligarobotovapp.Redux

    typealias Reducer<State> = (state: State, action: Action) -> State

    fun <State : Any> combineReducers(vararg reducers: Reducer<State>) :
Reducer<State> {
```

```
        return { startState: State, action: Action ->
            reducers.fold(startState) { state, reducer ->
                reducer(state, action)
            }
        }
    }

    fun <State : Any,
            SubState1 : Any?> combineReducers(mapper1: (State) -> SubState1,
reducer1: Reducer<SubState1>,
                                              builder: (SubState1) -> State) :
Reducer<State> {
        return { startState: State, action: Action ->
            builder(
                reducer1(mapper1(startState), action)
            )
        }
    }

    fun <State : Any,
            SubState1 : Any?,
            SubState2 : Any?> combineReducers(mapper1: (State) -> SubState1,
reducer1: Reducer<SubState1>,
                                              mapper2: (State) -> SubState2,
reducer2: Reducer<SubState2>,
                                              builder: (SubState1, SubState2) ->
State) : Reducer<State> {
        return { startState: State, action: Action ->
            builder(
                reducer1(mapper1(startState), action),
                reducer2(mapper2(startState), action)
            )
        }
    }

    fun <State : Any,
            SubState1 : Any?,
            SubState2 : Any?,
            SubState3 : Any?> combineReducers(mapper1: (State) -> SubState1,
reducer1: Reducer<SubState1>,
                                              mapper2: (State) -> SubState2,
reducer2: Reducer<SubState2>,
                                              mapper3: (State) -> SubState3,
reducer3: Reducer<SubState3>,
                                              builder: (SubState1, SubState2,
SubState3) -> State) : Reducer<State> {
        return { startState: State, action: Action ->
            builder(
                reducer1(mapper1(startState), action),
                reducer2(mapper2(startState), action),
                reducer3(mapper3(startState), action)
            )
        }
    }

    fun <State : Any,
            SubState1 : Any?,
            SubState2 : Any?,
            SubState3 : Any?,
            SubState4 : Any?> combineReducers(mapper1: (State) -> SubState1,
reducer1: Reducer<SubState1>,
```

```
                                            mapper2:    (State)    ->    SubState2,
reducer2: Reducer<SubState2>,
                                            mapper3:    (State)    ->    SubState3,
reducer3: Reducer<SubState3>,
                                            mapper4:    (State)    ->    SubState4,
reducer4: Reducer<SubState4>,
                                            builder:     (SubState1,     SubState2,
SubState3, SubState4) -> State) : Reducer<State> {
        return { startState: State, action: Action ->
            builder(
                reducer1(mapper1(startState), action),
                reducer2(mapper2(startState), action),
                reducer3(mapper3(startState), action),
                reducer4(mapper4(startState), action)
            )
        }
    }

    fun <State : Any,
            SubState1 : Any?,
            SubState2 : Any?,
            SubState3 : Any?,
            SubState4 : Any?,
            SubState5   :   Any?>   combineReducers(mapper1:   (State)   ->   SubState1,
reducer1: Reducer<SubState1>,
                                            mapper2:    (State)    ->    SubState2,
reducer2: Reducer<SubState2>,
                                            mapper3:    (State)    ->    SubState3,
reducer3: Reducer<SubState3>,
                                            mapper4:    (State)    ->    SubState4,
reducer4: Reducer<SubState4>,
                                            mapper5:    (State)    ->    SubState5,
reducer5: Reducer<SubState5>,
                                            builder:     (SubState1,     SubState2,
SubState3, SubState4, SubState5) -> State) : Reducer<State> {
        return { startState: State, action: Action ->
            builder(
                reducer1(mapper1(startState), action),
                reducer2(mapper2(startState), action),
                reducer3(mapper3(startState), action),
                reducer4(mapper4(startState), action),
                reducer5(mapper5(startState), action)
            )
        }
    }
```

П2.3 Store.kt класс хранилища

```
    package com.example.ligarobotovapp.Redux

    import java.lang.NullPointerException

    class Store<State : Any> (initialState: State, val reducer: Reducer<State>) {
        var state = initialState
            private set
        private val middlewares = mutableListOf<Middleware<State>>()
        private  val  subscriptions  =  mutableMapOf<Any,  MutableList<Pair<(State)  ->
Any, Any >>>()
        private val actions = mutableListOf<Action>()
```

```
fun addMiddleware(middleware: Middleware<State>){
    middlewares.add(middleware)
}

fun <Context : Any, M : Any> subscribe (context: Context,
                                        mapper: (State) -> M,
                                        subscriber: (Context, M) -> Unit){
    val subscriptionsList = subscriptions[context] ?: mutableListOf()
    val pair = Pair(mapper, subscriber)
    subscriptionsList.add(pair)
    subscriptions[context] = subscriptionsList
    subscriber(context, mapper(state))
}

fun <Context : Any> subscribe(context: Context,
                              subscriber: (Context, State) -> Unit) =
        subscribe(context, {state -> state}, subscriber)

fun unsubscribe(context: Any){
    subscriptions.remove(context)
}

fun dispatch(action: Action) {
    actions.add(action)
    if (actions.size == 1) {
        processAction(action)
    }
}

fun processAction(action: Action){
    val initialDispatcher = {resulingAction : Action ->
        val prevState = state
        state = reducer(prevState, resulingAction)
        for ((context, subscripitionList) in subscriptions){
            for (subscription in subscripitionList){
                try {
                    val value = subscription.first.invoke(state)
                    val prevValue = subscription.first.invoke(prevState)
                    if (value !== prevValue){
                        callSubcription(context, value, subscription.second)
                    }
                } catch (error: NullPointerException){

                }
            }
        }
        Unit
    }
    middlewares.foldRight(initialDispatcher) { middleware, dispatcher ->
        middleware.dispatch(this, dispatcher)
    }.invoke(action)
    actions.removeAt(0)
    if (actions.size > 0) {
        processAction(actions[0])
    }
}

@Suppress("UNCHECKED_CAST")
private fun <Context : Any, M : Any> callSubcription(context: Context,
                                                     m: M,
                                                     subscription: Any){
    (subscription as (Context, M) -> Unit).invoke(context, m)
```

```
        }
    }
```

П2.4 MainActivity.kt класс основного окна

```
    package com.example.ligarobotovapp.navigation


    import android.os.Bundle
    import android.support.v7.app.AppCompatActivity
    import android.view.MenuItem
    import android.view.View
    import androidx.navigation.NavController
    import androidx.navigation.Navigation
    import com.example.ligarobotovapp.*
    import com.example.ligarobotovapp.Redux.Action
    import com.example.ligarobotovapp.State.AppState
    import com.example.ligarobotovapp.State.NAVIGATE_BACK
    import com.example.ligarobotovapp.State.NAVIGATE_TO
    import com.example.ligarobotovapp.State.NAVIGATE_TO_REPLACE
    import kotlinx.android.synthetic.main.activity_main.*
    import kotlin.math.min

    class MainActivity : AppCompatActivity() {
        private var navController: NavController? = null
        private var cacheNavigationStack: List<Int> = emptyList()

        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
            bottom_menu.setOnNavigationItemSelectedListener(::bottomMenuItemSelected)
            navController            =            Navigation.findNavController(this,
R.id.navigation_host_fragment)

            getStore().subscribe(this,    AppState::navigationStack)    {    activity,
navigationStack ->
                if (navigationStack.isEmpty()) {
                    cacheNavigationStack = navigationStack
                    //   особый   случай   при   старте   приложения   -   стек   ещё   не
проинициализирован
                    if     (navController?.currentDestination     !=     null     &&
navController?.currentDestination?.id != 0) {
                        activity.getStore().dispatch(Action(NAVIGATE_TO,
navController?.currentDestination?.id))
                    } else {
                        activity.getStore().dispatch(Action(NAVIGATE_TO,
R.id.startFragment))
                    }
                }  else  if  (navigationStack.size  ==  1  &&  navigationStack[0]  ==
navController?.currentDestination?.id) {
                    // Особый случай при старте приложения: фрагмент уже фактически
на стеке (виден),
                    // но его ещё не было в состоянии
                    cacheNavigationStack = navigationStack
                } else {
                    var      matchFragmentsCount      =      min(navigationStack.size,
cacheNavigationStack.size)
                    for ((i, targetId) in cacheNavigationStack.withIndex()) {
                        if (i >= navigationStack.size) {
                            navController?.popBackStack(targetId, true)
```

```
                        break
                    } else if (targetId != navigationStack[i]) {
                        navController?.popBackStack(targetId, true)
                        matchFragmentsCount = i
                        break
                    }
                }
                for (i in (matchFragmentsCount until navigationStack.size)) {
                    navController?.navigate(navigationStack[i])
                }
                cacheNavigationStack = navigationStack
            }
            toggleBottomMenuIfNeeded(cacheNavigationStack)
        }
    }
    override fun onDestroy() {
        getStore().unsubscribe(this)
        navController = null
        cacheNavigationStack = emptyList()
        super.onDestroy()
    }
    override fun onBackPressed() {
        if (cacheNavigationStack.size > 1) {
            getStore().dispatch(Action(NAVIGATE_BACK))
        } else {
            super.onBackPressed()
        }
    }
    private fun getStore() = (applicationContext as LigaRobotovMobileApp).store
    private fun toggleBottomMenuIfNeeded(navigationStack: List<Int>){
        if (navigationStack.isEmpty()){
            bottom_menu.visibility = View.GONE
            return
        }
        bottom_menu.visibility = View.VISIBLE
        when(navigationStack.last()){
            R.id.scheduleFragment       ->      bottom_menu.selectedItemId      =
R.id.scheduleItem
            R.id.consumerFragment       ->      bottom_menu.selectedItemId      =
R.id.profileItem
            R.id.contactsFragment       ->      bottom_menu.selectedItemId      =
R.id.contactsItem
            R.id.documentsFragment      ->      bottom_menu.selectedItemId      =
R.id.documentsItem
            R.id.newNewsFragment -> bottom_menu.selectedItemId = R.id.newNewsItem
            else -> bottom_menu.visibility = View.GONE
        }

    }
    private fun bottomMenuItemSelected(menuItem: MenuItem) : Boolean {
        when(menuItem.itemId){
            R.id.scheduleItem  ->  getStore().dispatch(Action(NAVIGATE_TO_REPLACE,
R.id.scheduleFragment))
            R.id.profileItem   ->  getStore().dispatch(Action(NAVIGATE_TO_REPLACE,
R.id.consumerFragment))
            R.id.contactsItem  ->  getStore().dispatch(Action(NAVIGATE_TO_REPLACE,
R.id.contactsFragment))
            R.id.documentsItem -> getStore().dispatch(Action(NAVIGATE_TO_REPLACE,
R.id.documentsFragment))
            R.id.newNewsFragment                                                ->
getStore().dispatch(Action(NAVIGATE_TO_REPLACE, R.id.newNewsFragment))
        }
```

```
            return true
        }
    }
```

П2.5 Navigation.xml файл навигационный архитектурный компонент

```
    <?xml version="1.0" encoding="utf-8"?>
    <navigation
            xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:app="http://schemas.android.com/apk/res-auto"
            xmlns:tools="http://schemas.android.com/tools"
            android:id="@+id/navigation"
            app:startDestination="@+id/startFragment">

        <fragment
                android:id="@+id/startFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.StartFragment"
                android:label="fragment_start"
                tools:layout="@layout/fragment_start">
            <action android:id="@+id/action_startFragment_to_infoFragment"
                    app:destination="@+id/infoFragment"/>
            <action android:id="@+id/action_startFragment_to_consumerFragment"
                    app:destination="@id/consumerFragment"/>
        </fragment>
        <fragment
                android:id="@+id/infoFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.InfoFragment"
                android:label="fragment_info"
                tools:layout="@layout/fragment_info">
        </fragment>
        <fragment android:id="@+id/otpFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.OtpFragment"
                  android:label="fragment_otp"
                  tools:layout="@layout/fragment_otp"/>
        <fragment android:id="@+id/consumerFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.ConsumerFragment"
                  android:label="fragment_consumer"
                  tools:layout="@layout/fragment_consumer"/>
        <fragment android:id="@+id/contactsFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.ContactsFragment"
                  android:label="fragment_contacts"
                  tools:layout="@layout/fragment_contacts"/>
        <fragment android:id="@+id/compensFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.CompensFragment"
                  android:label="fragment_compens"
tools:layout="@layout/fragment_compens"/>
        <fragment android:id="@+id/recallFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.RecallFragment"
                  android:label="RecallFragment"/>
        <fragment android:id="@+id/newNewsFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.NewNewsFragment"
```

```
                android:label="fragment_new_news"
tools:layout="@layout/fragment_new_news"/>
        <fragment android:id="@+id/scheduleFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.ScheduleFragment"
                android:label="ScheduleFragment"/>
        <fragment android:id="@+id/documentsFragment"

android:name="com.example.ligarobotovapp.navigation.fragments.DocumentsFragment"
                android:label="DocumentsFragment"/>
    </navigation>
```