МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет
«Московский институт электронной техники»

Кафедра информатики и программного обеспечения вычислительных систем

Фаткуллин Осип Андреевич

Бакалаврская работа
по направлению 09.03.04 «Программная инженерия»

Разработка мобильного приложения сопровождения учебного процесса студентов в системе ОРИОКС (Шифр МП СУПС)

Студент _____ О.А. Фаткуллин

Руководитель,
доцент, к.п.н. _____ Е.Л. Федотова

Москва 2018

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МП СУПС — мобильное приложение сопровождения учебного процесса студентов в системе ОРОИКС.

ОРИОКС — организация распределенного информационного обмена в корпоративных средах.

ПК — персональный компьютер.

API — application programming interface — внешний интерфейс взаимодействия с приложением.

БД — база данных.

HTML — hypertext markup language — язык гипертекстовой разметки.

ОС — операционная система.

ИЛМ — инфологическая модель предметной области.

ER — entity-relationship (сущность-связь).

VCS — version control system — система контроля версий.

NDK — native development kit.

SVG — scalable vector graphics — масштабируемая векторная графика.

DI — dependency injection — внедрение зависимости.

GoF — gang of four — банда четырёх (Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес).

APK — Android package kit.

DSL — domain-specific language — язык, специфический для предметной области.

XML — extensible markup language — расширяемый язык разметки.

CI — continuous integration — непрерывная интеграция.

YAML — YAML Ain’t Markup Language — YAML – не язык разметки.

CLI — command line interface — интерфейс командной строки.

TDD — Test-Driven Development — разработка через тестирование.

ВВЕДЕНИЕ

В современных учебных заведениях всё чаще практикуют использование систем сопровождения учебного процесса, которые позволяют учащимся видеть свои оценки, получать домашние задания, узнавать о событиях учебного процесса. В МИЭТе такой системой является ОРИОКС (Организация Распределенного Информационного Обмена в Корпоративных Средах).

Слабым местом ОРИОКС является невозможность оперативного оповещения учащихся о событиях учебного процесса. Это обусловлено тем, что ОРИОКС представляет из себя веб-приложение и подразумевает регулярное посещение через браузер. Если же студент долгое время не заходит в систему, что бывает довольно часто, то он может пропустить информацию о важных событиях, например о пересдачах. Так же, в ОРИОКС невозможна установка напоминаний о предстоящих событиях.

Еще одна проблема веб-приложений – неудобство использования на мобильных устройствах. Большая часть студентов посещает ОРИОКС именно с мобильных устройств, и, хотя, дизайн сайта оптимизирован для работы на небольших экранах, всё же мобильные устройства накладывают свои ограничения. Скорость интернет соединения, чаще всего, ниже чем на стационарных ПК, кроме того, соединение может часто разрываться, что приводит к дискомфорту при использовании веб-приложения, т.к. для каждой страницы браузер должен загрузить, помимо данных, разметку и таблицу стилей.

Все эти недостатки можно устранить, создав мобильное приложение, которое будет получать данные от сервера ОРИОКС через API). Это позволит запрашивать и получать только ту информацию, которая нужна для работы приложения в конкретный момент времени, так как приложению не нужно скачивать таблицу стилей и разметку. В виду малого потребления трафика приложение сможет работать на смартфоне в фоновом режиме и отображать уведомления и напоминания в тот момент когда они только пришли.

Удобство работы с системой ОРИОКС при нестабильном интернет соединении тоже повысится за счёт меньшего объёма пересылаемых данных. Кроме того, приложение может кэшировать данные и использовать их даже при отсутствии интернет соединения.

На данный момент нет ни одного приложения работающего с ОРИОКС через API, обладающего полным функционалом и предоставляющего возможности push-уведомлений, поэтому задача является актуальной. Целью данной работы является создание такого приложения, чтобы повысить оперативность оповещения студентов о событиях учебного процесса.

Задачи выпускной квалификационной работы:

— сравнительный анализ существующих программных решений;
— выбор платформы;
— выбор языка и среды программирования;
— разработка алгоритма;
— разработка схемы данных;
— разработка пользовательского интерфейса;
— отладка и тестирование;
— разработка руководства оператора.

Пояснительная записка состоит из введения, исследовательского, конструкторского и технологического разделов, заключения, списка использованных источников и трёх приложений.

Исследовательский раздел содержит обзор существующих решений, исследование структуры ОРИОКС, описание входных и выходных данных, постановку целей и задач. Конструкторский раздел содержит обзор и выбор языка программирования, среды разработки, выбор стека технологий, описание алгоритма работы, схему данных и макеты пользовательского интерфейса. Технологический раздел включает в себя описания применяемых технологий, нюансы разработки под Android, описание процессов сборки, публикации, тестирования, отладки и сопровождения.

В приложении А содержится техническое задание на разработку МП СУПС. Приложение Б содержит фрагменты исходного кода приложения. В приложении В — руководство оператора.

1 ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

В этом разделе проводится исследование предметной области, обзор существующих решений и постановка общих задач.

1.1 Актуальность разработки мобильного приложения

Мобильные приложения могут быть инструментом для быстрой доставки информации, чего нельзя добиться при помощи обычного веб-приложения.

Индустрия мобильных устройств очень быстро развивается. На смену старым устройствам приходят более новые, современные и обладающие большим спектром возможностей. Количество пользователей с каждым годом растёт. сейчас смартфоны есть почти у всех студентов и они редко с ними расстаются надолго. Конечно, важную роль играет программная составляющая — мобильные приложения. Они существуют совершенно разной направленности:

— развлекательные (игры, музыкальные и видео проигрыватели и т.д.);
— коммуникационные (мессенджеры, навигаторы и т.д.);
— справочные (словари, базы знаний);
— прикладные (все остальные от графического редактора до калькулятора).

Кроме того, популярность мобильных приложений повлекла за собой появление мощных инструментов разработки, большого количества библиотек и фреймворков, что, в свою очередь, сделало разработку приложений быстрой, лёгкой и продуктивной.

По возможностям оповещений с приложениями могут сравниться только боты для мессенджеров или социальных сетей. Боты, как правило, представляют собой программу, “общение” с которой происходит посредством текстовых сообщений. Сообщения могут содержать команды или запросы, которые бот обрабатывает и возвращает результат тоже в виде сообщения. Боты хороши тем, что работают везде, где работает платформа для которой они написаны. То есть бот для Telegram будет работать и на iOS, и на Android, и на PC, потому что приложение Telegram есть для всех этих платформ. Но, в отличие от приложений у ботов есть следующие слабые места:

— отсутствие пользовательского интерфейса, что приводит к невозможности создания сложных систем;

— боты становятся неудобными, если выполняют большое количество функций, т.к. резко увеличивается количество команд;

— для бота обязательно нужен сервер, на котором он будет размещаться, а значит нужны дополнительные траты;

— бота нельзя использовать в режиме оффлайн, потому что если не работает мессенджер, то не работает и бот.

В случае с ОРИОКС бот неудобен, т.к. невозможна работа в условиях отсутствия интернета и невозможность создания графического интерфейса накладывает ограничения на возможный функционал.

1.2 Обзор аналогичных решений

Был произведен поиск существующих решений в Windows Phone Store, Google Play, Apple App Store, на GitHub, и было найдено три аналога. К ним можно, так же, добавить сам сайт ОРИОКС.

Рассмотрим преимущества и недостатки каждого решения по отдельности. В качестве критериев будем брать:

— способ получения данных;
— возможность push-уведомлений;
— возможность просмотра расписания;
— возможность просмотра текущих предметов;
— возможность просмотра успеваемости;
— возможность просмотра списка долгов;
— возможность просмотра списка пересдач;
— наличие графического интерфейса;
— оффлайн доступ.

1.2.1 Сайт ОРИОКС

Платформа: Браузер.

Из плюсов: сайт ОРИОКС получает информацию напрямую из БД. Это позволяет позволяет запрашивать только ту информацию, которая нужна в данный момент для отображения страницы. Можно, так же, отметить наличие всех перечисленных выше возможностей, за исключением просмотра расписания (но его можно

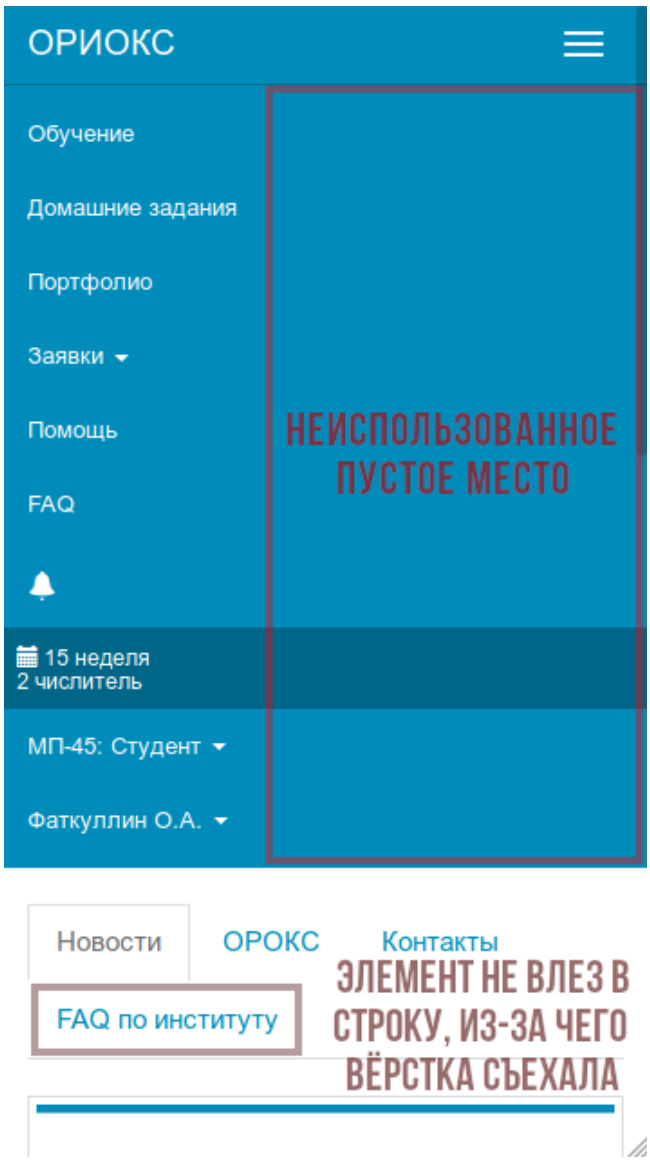

Рисунок 1.1 — Главная страница ОРИОКС с открытым меню

посмотреть на сайте МИЭТ) [2]. Графический интерфейс присутствует, но не полностью адаптирован для мобильных устройств, то есть в некоторых местах элементы интерфейса не помещаются на экране, а в других — наоборот слишком много неиспользованного места (см. рис. 1.1).

К минусам можно отнести отсутствие push-уведомлений, невозможность просмотра информации без интернет соединения и необходимость загружать таблицы стилей и HTML разметку для просмотра страницы.

1.2.2 Приложение “Расписание для МИЭТ”

Платформа: Android. Дополнительная информация: около тысячи установок; рейтинг на Google Play — 4.7 из 5; дата последнего обновления — 22.04.2018.

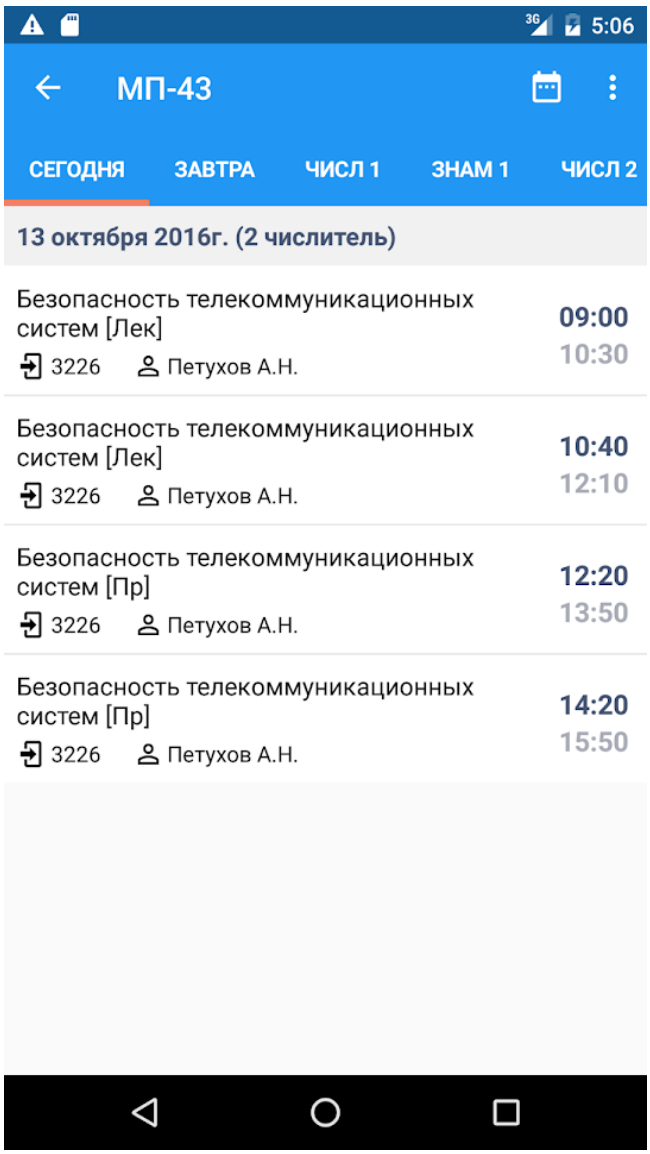

Рисунок 1.2 — Экран расписания в приложении “Расписание для МИЭТ”

Приложение предназначено для просмотра новостей МИЭТ и расписания любой группы с возможностью скачать его и использовать в оффлайн-режиме. Раньше присутствовал функционал просмотра успеваемости, но из-за изменений на сайте ОРИОКС этот функционал стал недоступен [3].

Получение расписания реализовано через API, это плюс. Для получения текущей успеваемости использовался синтаксический анализ сайта. При таком подходе любое изменение в таблице стилей или HTML разметке сайта приводит к неработоспособности приложения, что и случилось.

Из требуемых возможностей присутствует просмотр и кэширование расписания, это позволяет просматривать его без интернет-соединения Просмотр текущих предметов, успеваемости, списка долгов и пересдач невозможен.

Графический интерфейс присутствует, но не соответствует требованиям Material Design. Например, слишком маленькие отступы от краёв экрана (см. рис. 1.2).

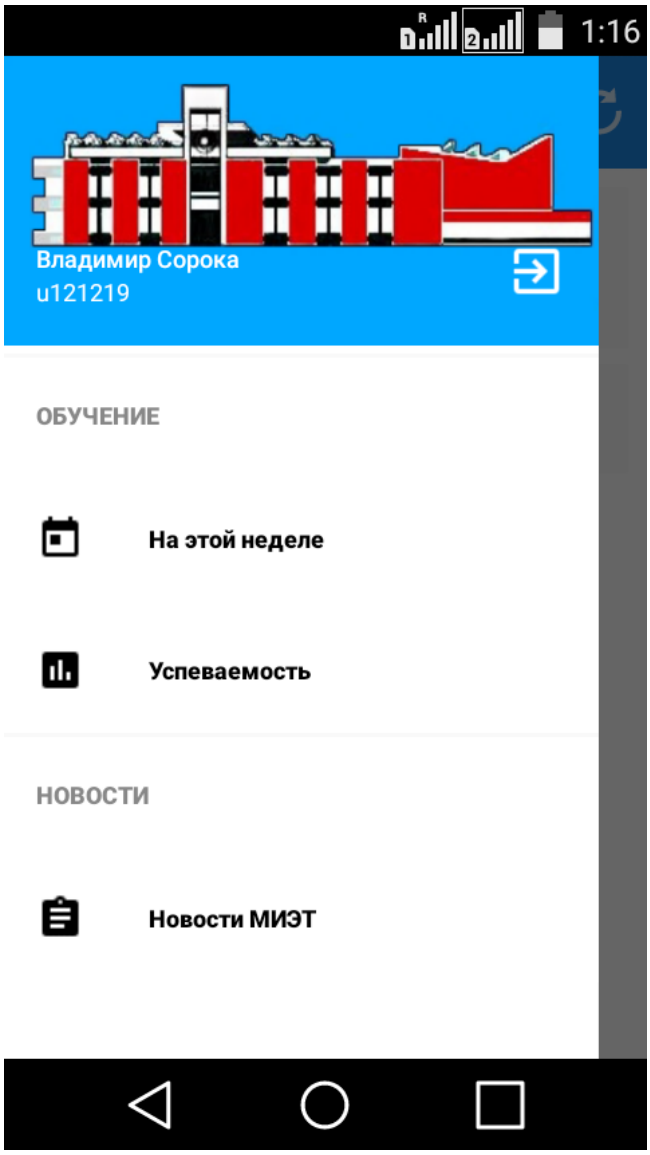

Рисунок 1.3 — Главный экран с открытым меню в приложении “Ориокс Live”

1.2.3 Приложение “ОРИОКС Live”

Платформа: Android. Дополнительная информация: около тысячи установок; рейтинг на Google Play — 3.9 из 5; дата последнего обновления — 30.09.2015.

Приложение предназначено для просмотра успеваемости, списка контрольных мероприятий и информации о преподавателях [4].

Данные приложение получает при помощи непубличного программного интерфейса, но после обновления ОРИОКС до версии 3.0, интерфейс перестал работать. Приложение не обновлялось с 2015 года и на данный момент не работает.

Заявленный функционал проверить не удалось из-за невозможности авторизации, поэтому все эти возможности отметим как отсутствующие. Рабочей осталась только возможность просмотра новостей.

Графический интерфейс присутствует, но не соответствует требованиям Material Design. Неправильные отступы, слишком контрастные цвета (см. рис. 1.3).

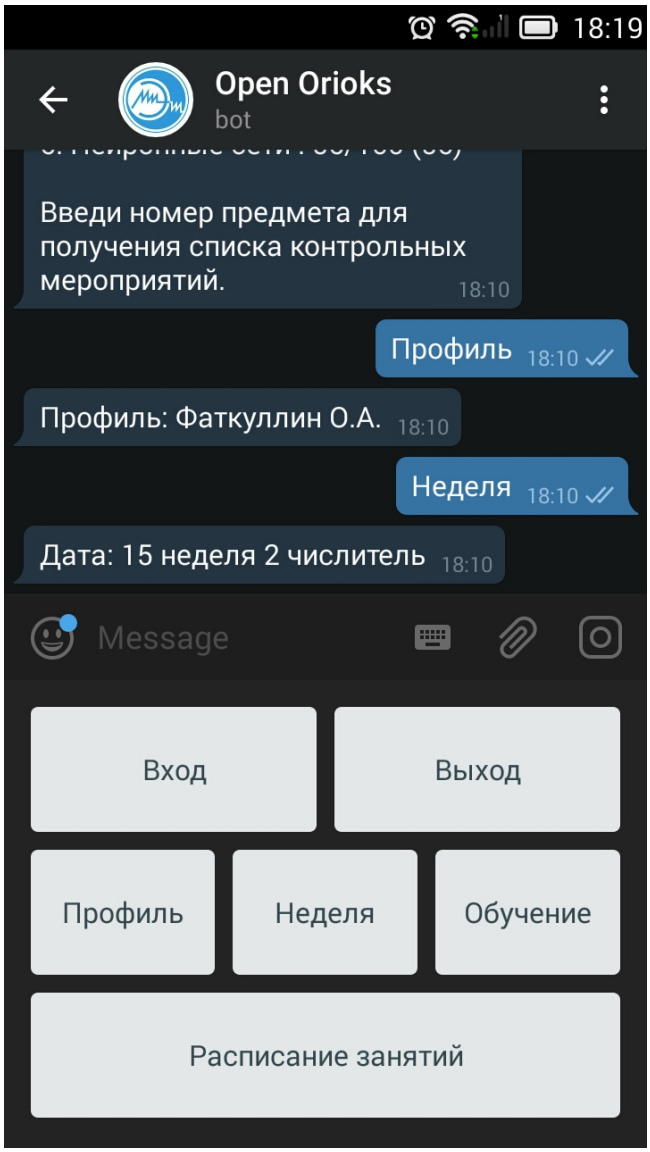

Рисунок 1.4 — Интерфейс управления Telegram-ботом “Open Orioks”

1.2.4 Telegram-бот “Open Orioks”

Платформа: Telegram. Дополнительная информация: последнее обновление — 12.10.2017.

Бот позволяет просматривать расписание на день, текущую успеваемость и список контрольных мероприятий по каждому предмету.

Для получения данных используется синтаксический анализ, но за счёт того, что данные всех студентов обновляются раз в полчаса и сохраняются в хранилище бота, скорость получения данных конечным пользователем сравнима со скоростью получения данных из БД [5].

Собственного графического интерфейса нет. Взаимодействие с ботом производится через текстовые сообщения в мессенджере Telegram. Использование при отсутствии интернета невозможно, но можно просматривать предыдущие ответы бота, что можно считать частичным кэшированием.

По итогом обзора аналогичных решений была построена таблица 1.1. Как видно из таблицы, нет ни одного решения, которое бы соответствовало всем необходимым критериям, что еще раз доказывает актуальность задачи.

Таблица 1.1 — Сравнение аналогичных решений

Критерий	Сайт ОРИОКС [2]	Приложение “Расписание для МИЭТ” [3]	Приложение “ОРИОКС Live” [4]	Бот “Open Orioks” [5]	МП СУПС
Скорость получения информации	Средняя	Высокая	Низкая	Средняя	Высокая
Push-уведомления	−	−	−	+	+
Расписание	±	+	−	±	+
Текущие предметы	+	−	−	+	+
Успеваемость	+	−	−	+	+
Долги	+	−	−	−	+
Пересдачи	+	−	−	−	+
Графический интерфейс	+	−	−	−	+
Оффлайн доступ	−	+	−	±	+

\+ – указанная возможность присутствует

± – указанная возможность частично присутствует

− – указанная возможность отсутствует

В приложении “Расписание для МИЭТ” есть кэширование, но работает оно только для расписания, так как остальные функции недоступны. Только Telegram-бот “Open Orioks” предоставляет возможность push-уведомлений, но не обладает всеми требуемыми возможностями, т.к. при увеличении количества выполняемых задач, бот становится неудобен в использовании.

1.3 Обзор мобильных платформ

Смартфоны, в виде, похожем на нынешний, появились в 2007 году, когда Стив Джобс на выставке “Macworld Conference & Expo” показал миру iPhone. Все существующие телефоны мгновенно стали устаревшими. Рынок смартфонов был практически пуст и было понятно, что один iPhone с iOS его не покроет. В Google в это время только думали о выпуске своего телефона, но о смартфоне речи не шло.

За четыре года до этого, в 2003 году, Энди Рубин, Ник Сир, Крис Уайт и Рич Майнер решили создать операционную систему для носимых устройств, которые могли бы подстраиваться под нужды пользователя. Они основали компанию Android Inc. и назвали свою операционную систему Android. Никто тогда не оценил этот стартап и в 2005 году компания была на грани банкротства, но Энди Рубин смог убедить Google, которая занималась скупкой инновационных проектов, в том, что у Android есть будущее. Так и оказалось.

В 2007 году Google вспоминает, что они покупали стартап, направленный на разработку операционной системы для переносных устройств. Команда Android была только рада заняться разработкой ОС, но нужно было найти производителя смартфонов, который был бы готов выпустить на рынок устройство с совершенно новой ОС. У Nokia уже была своя ОС — Symbian. Motorola в это время была ослеплена успехом Razr и вряд ли обратила бы внимание на Android. Оставались еще LG и HTC, но LG уже решили развивать Windows Mobile вместе с Microsoft, поэтому выбор пал на HTC. HTC была рада сотрудничеству с большой компанией и могла обеспечить быстрый выпуск прототипов устройств. В 2007–2008 году Google и HTC интенсивно работали над первым смартфоном на базе Android — HTC Dream. 22 октября 2008 года устройство уже поступило в продажу.

TODO: Добавить инфографику с историей развития мобильных платформ.

В 2009 году Microsoft тоже попыталась выйти на рынок смартфонов и выпустила Windows Phone 7. Проект оказался неудачным, своего рода “мобильная Vista”.

Продажи устройств на Windows Mobile и Symbian упали и остались только две развивающиеся ОС — Android и iOS. Apple не позволяла сторонним компаниям использовать iOS, поэтому все взгляды обратились на Android.

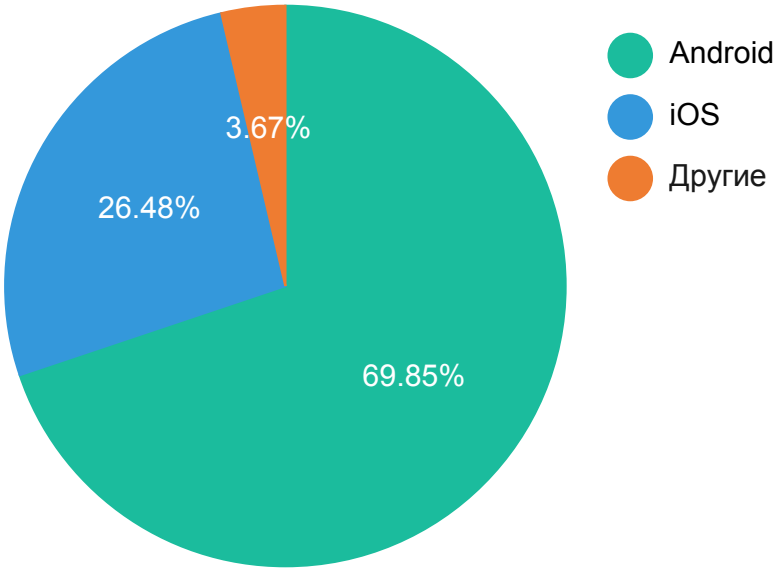

Рисунок 1.5 — Доля устройств (данные по России)

Гонка Apple и Google продолжалась. Компании улучшали свои ОС чтобы превзойти друг друга по быстродействию, безопасности, интерфейсу, функциональности, удобству использования и т.д. В данный момент обе ОС достигли высокого уровня по всем направлениям, но в силу того, что Android — открытая система и лицензии (Apache v2 и GNU GPL v2) не запрещают устанавливать адаптировать её под любые устройства, она гораздо популярнее чем iOS.

Исходя из доли устройств с каждой мобильной ОС в России (см. рис. 1.5), будем выбирать из двух вариантов: iOS и Android. Разработка для iOS ведется на языке Swift или Objective-C, для работы среды разработки требуется устройство под управлением MacOS [6]. Для разработки под Android можно использовать Java или Kotlin, среда разработки не накладывает ограничений на ОС разработчика, так как доступна для Windows, Linux и MacOS [7].

Для сравнения была составлена таблица 1.2. Исходя из покрытия устройств, открытости исходного кода, ограничений на ОС для разработки и наличия опыта программирования на языках Java и Kotlin, была выбрана платформа Android.

Таблица 1.2 — Сравнение мобильных платформ

Название	Поддерживаемые языки программирования	Доля устройств (Россия) [8]	Исходный код	ОС для разработки
iOS [6]	Swift, Objective-C	26.48%	Закрытый	MacOS
Android [7]	Kotlin, Java	69.85%	Открытый	GNU\Linux, Windows, MacOS

1.4 Исследование структуры ОРИОКС, концептуальная модель

Функционал ОРИОКС делится на несколько частей, мы будем рассматривать только часть которая касается студентов, и говоря, например, “инфологическая модель ОРИОКС” будем иметь ввиду инфологическую модель студенческой части ОРИОКС.

На главной странице есть доступ к новостям, FAQ, портфолио, текущей успеваемости. При открытии страницы успеваемости отображается список предметов, текущий балл по каждому из них и индикатор контрольного мероприятия (если индикатор активен, значит на этой неделе есть контрольное мероприятие по этому предмету). Список дисциплин одинаков для всех студентов из одной группы. Но если студент обучается по индивидуально плану, список его предметов будет отличаться. Кроме того, у каждого студента может быть собственный список долгов, который хранится и отображается отдельно от текущих дисциплин.

Нажатие на строку предмета открывает страницу с подробной информацией, где содержится список преподавателей, форма зачёта, список ресурсов, название кафедры и список контрольных мероприятий. Для каждого мероприятия выводится:

— номер недели когда проходит мероприятие;

— название контрольного мероприятия;

— форма контроля;

— максимальное количество баллов;

— текущее количество баллов;

— индикатор является контрольное мероприятие обязательным или дополнительным.

Расписание не представлено в ОРИОКС, но оно есть на сайте МИЭТ. Для каждой записи в расписании указывается:

— предмет;
— день недели;
— тип недели (каждую неделю, 1/2 числитель/знаменатель);
— номер аудитории;
— номер пары;
— тип занятия (лабораторная работа, лекция, семинар);
— преподаватель.

Можно выделить следующие сущности: группа, студент, план на семестр, ресурс, дисциплина, контрольное мероприятие, расписание, занятие (одна запись из расписания). Для них была построена ИЛМ (рис. 1.6) и ER-диаграмма (рис. 1.7).

1.5 Входные и выходные данные

В студенческой части ОРИОКС нет полей ввода, кроме формы запроса справки и формы портфолио. Ввод осуществляется при помощи мыши и заключается в нажатии на активные элементы сайта (пункты меню, кнопки), в результате студент получает страницу, отображающую данные, которые запросил, сформированные в удобный для восприятия вид. В приложении ввод данных должен осуществляться тоже путём нажатия на активные элементы, на выходе пользователь будет получать экран с отображенными на нём запрошенными данными.

В силу того, что приложение рассчитано на использование с одного аккаунта и в ОРИОКС нет взаимодействия студентов между друг другом, в локальной БД можно хранить только об одном студенте. В случае, если понадобится использовать приложение с нескольких аккаунтов, можно создать отдельные базы данных для каждого пользователя. Это не только упростит работу с БД (т.к. связь Группа-Студент станет 1:1 вместо 1:N), но и изолирует данные одного пользователя от данных другого, что позволит при желании зашифровать каждую БД пин-кодом, индивидуальным для каждого пользователя. Для хранения будет использоваться база данных SQLite.

1.6 Постановка задачи

Можно выделить следующие задачи:

а) Нужно разработать приложение, которое предоставит студентам полный функционал ОРИОКС. На данный момент такого приложения не существует;

б) Приложение должно иметь возможность push-уведомлений оповещать студента как только такое уведомление было получено;

в) Приложение должно обеспечивать возможность работы в оффлайн режиме. Для этого нужно сохранять все полученные данные в локальную базу данных;

г) Для поддержания актуальности сохранённых данных нужно обновлять их в фоновом режиме.

Выводы по разделу

TODO: Написать выводы

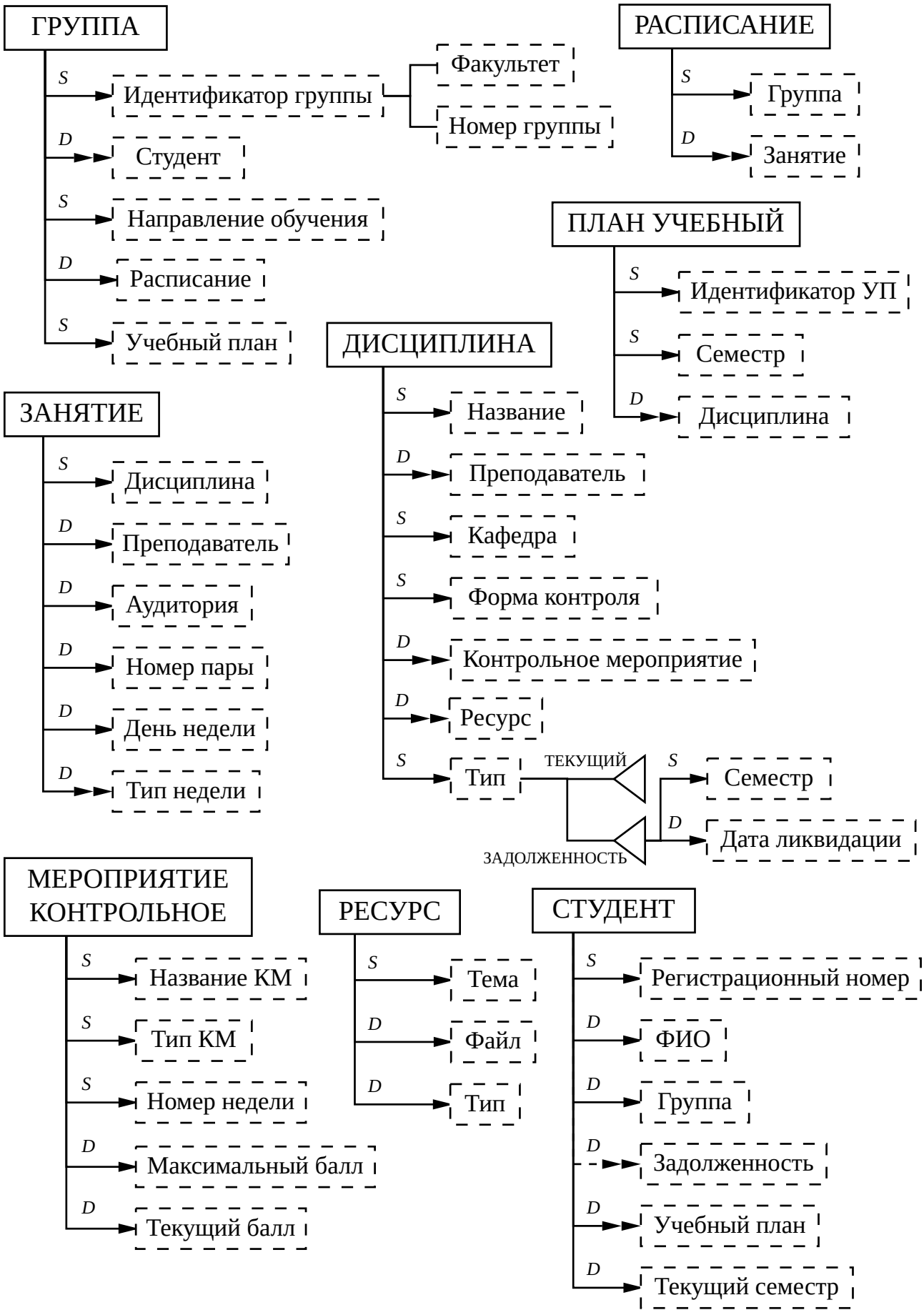

Рисунок 1.6 — Инфологическая модель ОРИОКС

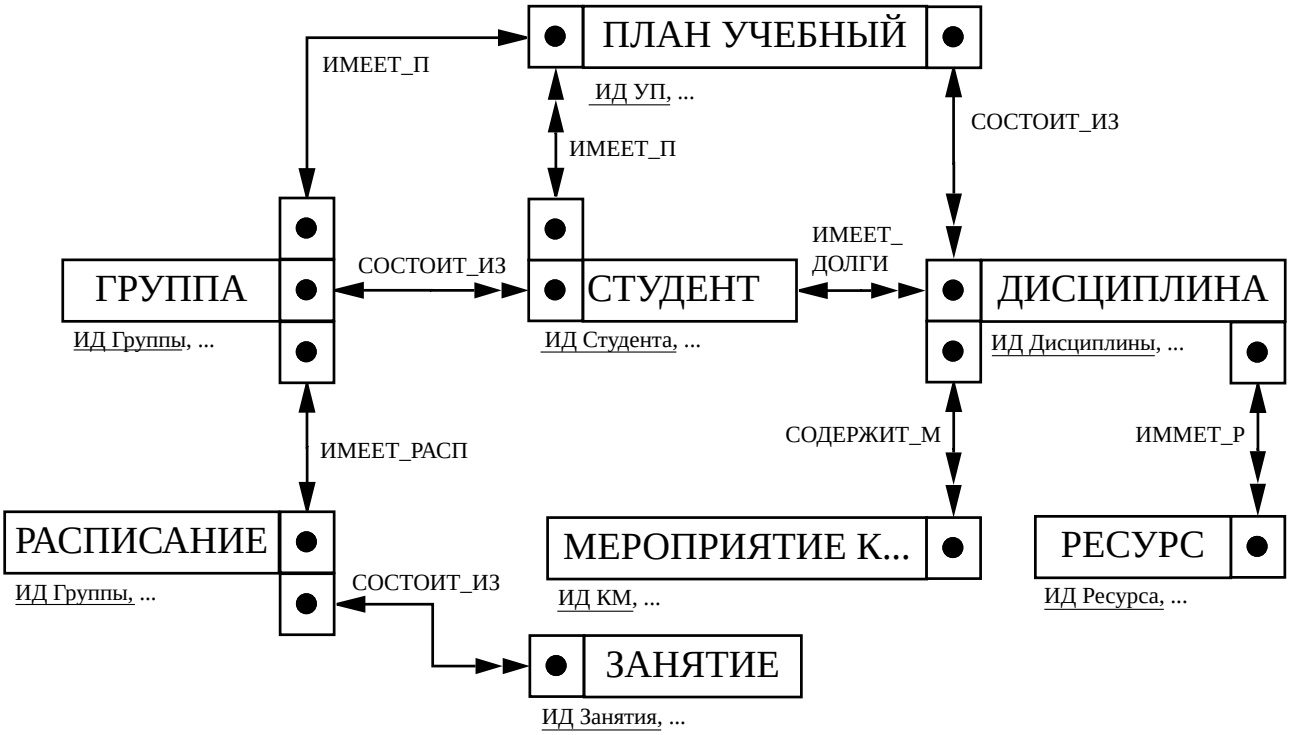

Рисунок 1.7 — ER-диаграмма ОРИОКС

2 КОНСТРУКТОРСКИЙ РАЗДЕЛ

2.1 Выбор языка программирования

Для разработки приложений под Android используют, в основном, три языка программирования: Kotlin, Java и C++.

Kotlin стоит на первом месте не с проста. В 2017 году компания Google объявила Kotlin новым официальным языком для Android и на то есть причины. Java очень медленно развивалась, новые версии выходили с периодичностью раз в два-три года и новый функционал добавлялся крайне осторожно. Например, в Java нет атрибутов класса, которые есть во многих современных языках, и приходится писать слишком много однотипного кода (геттеров и сеттеров) для тривиальной задачи — хранение внутреннего состояния объекта.

Еще одной большой проблемой Java является отсутствие null-безопасности. То есть никогда нельзя быть уверенным, что в определенном месте кода объект не является null и есть вероятность получить NullPointerException в самом неожиданном месте.

Даже сейчас, когда Oracle внесла изменения в цикл релизов Java и выпускает их каждые полгода, это не спасёт Java для Android. Причина тому — обратная совместимость приложений со старыми версиями Android API. Например, поддержка Java 7 была добавлена в Android 4.4 (API 19, 2013 год), и эта версия считается стандартной целевой версией для широкой аудитории, так как на этой версии всё еще работает около 10% устройств. На более старых версиях суммарно работает меньше пяти процентов, поэтому именно эта версия позволяет достигнуть покрытия аудитории 95%. Поддержка Java 8 была добавлена только в Android O (API 26, 2017 год), но эта версия еще не скоро станет целевой, и трудно представить когда, при таких темпах обновления целевой версии, разработчики смогут использовать функции недавно вышедшей Java 10 или Java 11, которая выйдет через полгода.

Разработчикам на Android требовался современный, обновляющийся и простой язык, но при этом, чтобы можно было использовать существующие библиотеки, написанные на Java. Сейчас существует много JVM языков, которые в той или иной мере совместимы с Java, но ни один из них, кроме Kotlin, не обрёл широкой популярности. Scala — слишком сложна в освоении. Groovy — допускает слишком много вольностей (даже самую простую задачу, например, объявление метода, мож-

но решить несколькими способами), а динамическое выведение типов приводит к ошибкам во время выполнения. И, что немаловажно, для этих языков нет хорошей IDE, которая бы поддерживала все их функции. Kotlin же постарался вобрать в себя все положительные стороны каждого из языков, но и добавить своего и полностью исключить неоднозначности. Тут есть удобные функциональные и nullable типы, как в Groovy, data-классы как в Scala, extension-функции и перегрузка операторов, как в C# и т.д. Kotlin разработан компанией JetBrains, которая занимается разработкой IntelliJ IDEA и поэтому проблем с поддержкой в IDE у него нет [10].

Например, чтобы объявить простой класс с несколькими внутренними состояниями и параметрами конструктора по умолчанию в Java потребуется такой код:

```
public class Student {

    private final String id;

    private String fullName;
    private String group;
    private List<Subject> debts;

    public Student(String id, String fullName, String group) {
        this(id, fullName, group, new ArrayList<Subject>());
    }

    public Student(String id,
                   String fullName,
                   String group,
                   List<Subject> debts) {
        this.id = id;
        this.fullName = fullName;
        this.group = group;
        this.debts = debts;
    }

    public String getId() {
        return id;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
```

```
        this.fullName = fullName;
    }

    public List<Subject> getDebts() {
        return debts;
    }

    public void setDebts(List<Subject> debts) {
        this.debts = debts;
    }

    public String getGroup() {
        return group;
    }

    public void setGroup(String group) {
        this.group = group;
    }
}
```

Листинг 2.1 — Пример data-класса на Java

В Kotlin же для того чтобы выполнить ту же самую задачу потребуется гораздо меньше кода:

```
data class Student(
        val id: String,
        var fullName: String,
        var group: String,
        var debts: List<Subject> = arrayListOf()
)
```

Листинг 2.2 — Пример data-класса на Kotlin

В Kotlin почти невозможно получить `NullPointerException` благодаря тому, что различаются nullable и non-nullable типы. Если переменная может принимать значение null, то после её типа дописывается вопросительный знак. Кроме того, для работы с nullable значениями в Kotlin предусмотрено множество удобных операторов. Вот пример null-безопасного кода на Kotlin:

```kotlin
// Переменным, которые могут принимать значение null после типа добавляется
// вопросительный знак
class Foo(val bar: String?)

// Допустим, что метод getFoo может вернуть объект Foo или null
val buz: Foo? = getFoo()

// Если хотя бы один объект в цепочке будет null, то длина будет равняться нулю
val length = buz?.bar?.length ?: 0
// Кроме того, тип переменной length будет автоматически выведен как Int
```

Листинг 2.3 — Пример null-безопасного кода на Kotlin

К слову, на Java подобный код будет выглядеть следующим образом:

```java
public class Foo {
    private final String bar;

    public Foo(String bar) {
        this.bar = bar;
    }

    public String getBar() {
        return bar;
    }
}

// Допустим, что метод getFoo может вернуть объект Foo или null
Foo buz = getFoo();

int length = 0;
if (buzz != null && buzz.getBar() != null) {
    length = buz.getBar().length;
}
```

Листинг 2.4 — Пример null-безопасного кода на Java

Даже если не учитывать объявление класса, на Java код получается сложнее и длиннее.

Kotlin очень быстро развивается и, в отличие от Java, для того чтобы использовать новую версию, достаточно лишь её подключить. Kotlin полностью совместим с Java, а значит с ним можно использовать все библиотеки, написанные на Java, коих для Android очень много.

Среди языков разработки был упомянут C++. Его используют только в специфических случаях, когда надо написать высокопроизводительный код. Обычно это работа с графикой, звуком и т.д. Минусом является то, что на C++ нельзя напрямую работать с Android SDK и нельзя использовать библиотеки, написанные на Java.

Таблица 2.1 — Сравнение языков программирования для Android

Критерий	Kotlin [10]	Java [11]	C++ [12]
Прямая работа с Android SDK	+	+	−
Наличие большого количества библиотек под Android	+	+	−
Возможность управления памятью напрямую	−	−	+
Простой синтаксис	+	−	−
Data-классы, null-безопасность	+	−	−

+ – указанная возможность присутствует

− – указанная возможность отсутствует

По итогам сравнения этих трёх языков программирования была составлена таблица 2.1. Был выбран язык Kotlin, как видно из таблицы, он имеет преимущество над остальными языками.

2.2 Выбор среды разработки

Были рассмотрены три среды программирования: Eclipse, Android Studio и IntelliJ IDEA.

2.2.1 Eclipse

Свободная IDE для разработки модульных кросплатформенных приложений. В стандартный пакет Eclipse не входят инструменты для создания Android приложений, но их можно добавить путём установки плагина. Тем же путём можно добавить поддержку Kotlin. Для Eclipse есть плагины для работы с Git, Gradle и прочими инструментами, полезными при разработке приложений.

Плюсом можно считать большое количество плагинов, возможность быстро и просто подключать новые языки, фреймворки и т.д. Плагин для работы с Git ре-

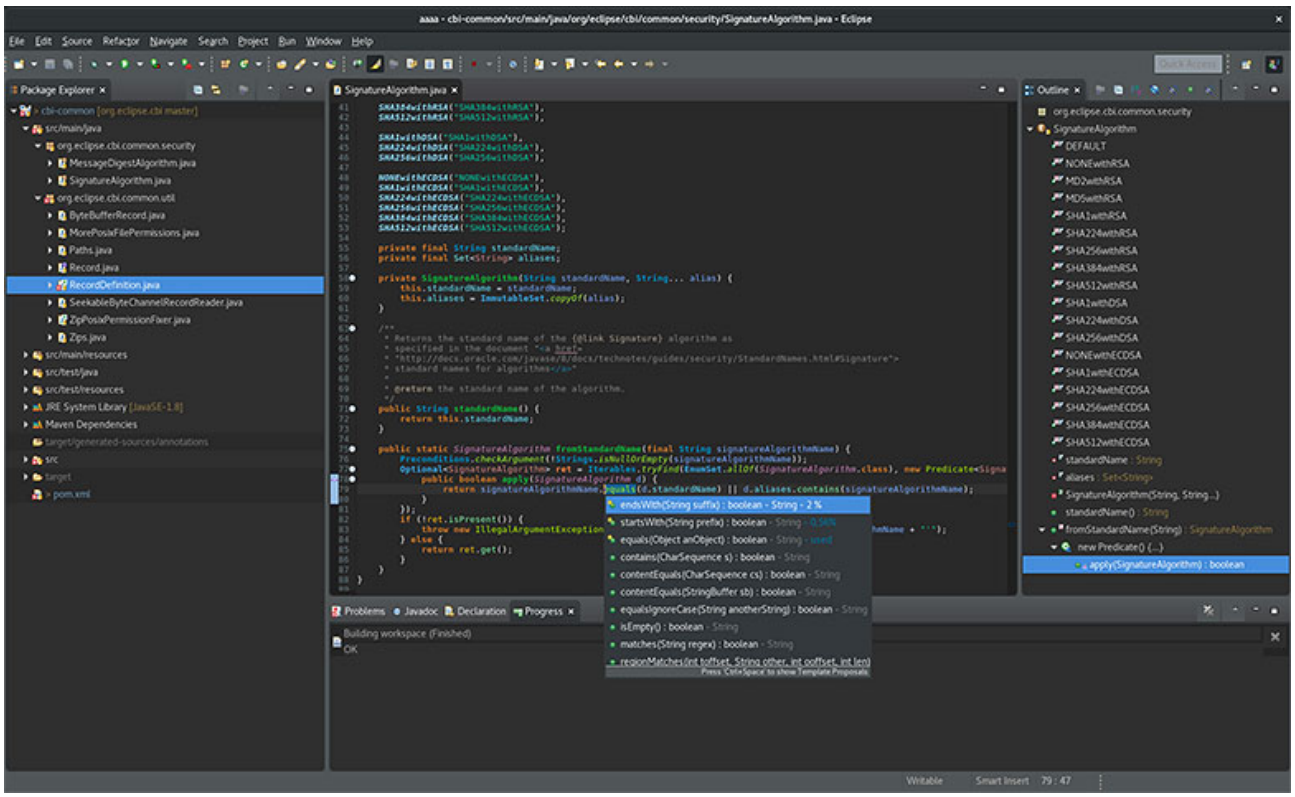

Рисунок 2.1 — Интерфейс Eclipse Oxygen

позиториями обладает большим количество функции и удобен в использовании. Из минусов — Eclipse не понимает контекст. Вернее понимает не так хорошо как IntelliJ IDEA или Android Studio. Это проявляется, например, в подсказках при написании кода. Eclipse просто подсказывает наиболее часто используемый вариант метода, поля и тому подобное, не глядя на контекст и зачастую это неудобно. Редактор экранных форм не очень удобен [13].

2.2.2 IntelliJ IDEA

Это интегрированная среда разработки от компании JetBrains. Её отличительными чертами являются: понимание контекста, наличие большого количества автоматических шаблонов рефакторинга, инспекции кода (инструменты анализа качества кода).

Есть две редакции: Community, с открытым исходным кодом и Ultimate, включающая все возможности редакции Community и дополнительные функции. Бесплатная версия предлагает следующие возможности:

— светлая и тёмная тема оформления;

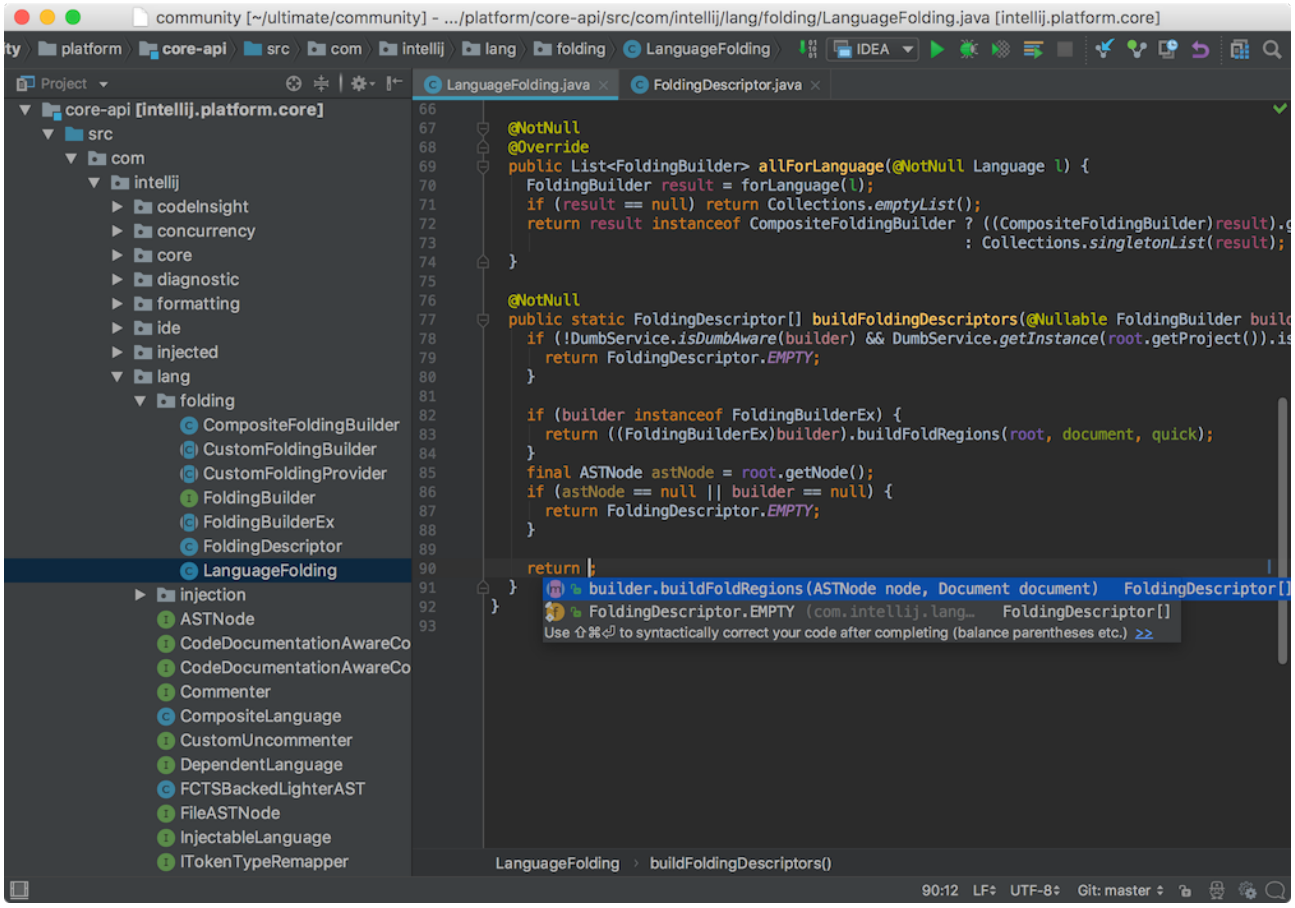

Рисунок 2.2 — Интерфейс IntelliJ IDEA 2018

— умное автодополнение, анализ качества кода, рефакторинги, форматирование кода ”на лету”для Java, Kotlin и др.;

— набор инструментов для разработки приложений под Android;

— отличная интеграция с VCS, с поддержкой таких полезных функций как частичные коммиты и сохранение контекста (открытых файлов, пакетов) для каждой ветки;

— интеграция с системами автоматической сборки: Gradle, Maven, Ant и др.;

— инструменты тестирования, с поддержкой множества библиотек и инструментов, таких как JUnit, Espresso, Spek, JaCoCo и др.

IDEA лучше других сред разработки поддерживает Kotlin потому как Kotlin тоже написан компанией JetBrains и новые возможности тут же добавляются в IDE.

В Ultimate версии добавляется множество функций, но из наиболее полезных при разработке мобильных приложений можно отметить:

— инструмент работы с базами данных и SQL файлами, удобный редактор схем БД, клиент и интеграция консоли для выполнения запросов в базу данных;

— поиск дубликатов кода;

— построение диаграмм классов и зависимостей.

Эта версия платная, но компания JetBrains предоставляет бесплатную лицензию для студентов и преподавателей [14].

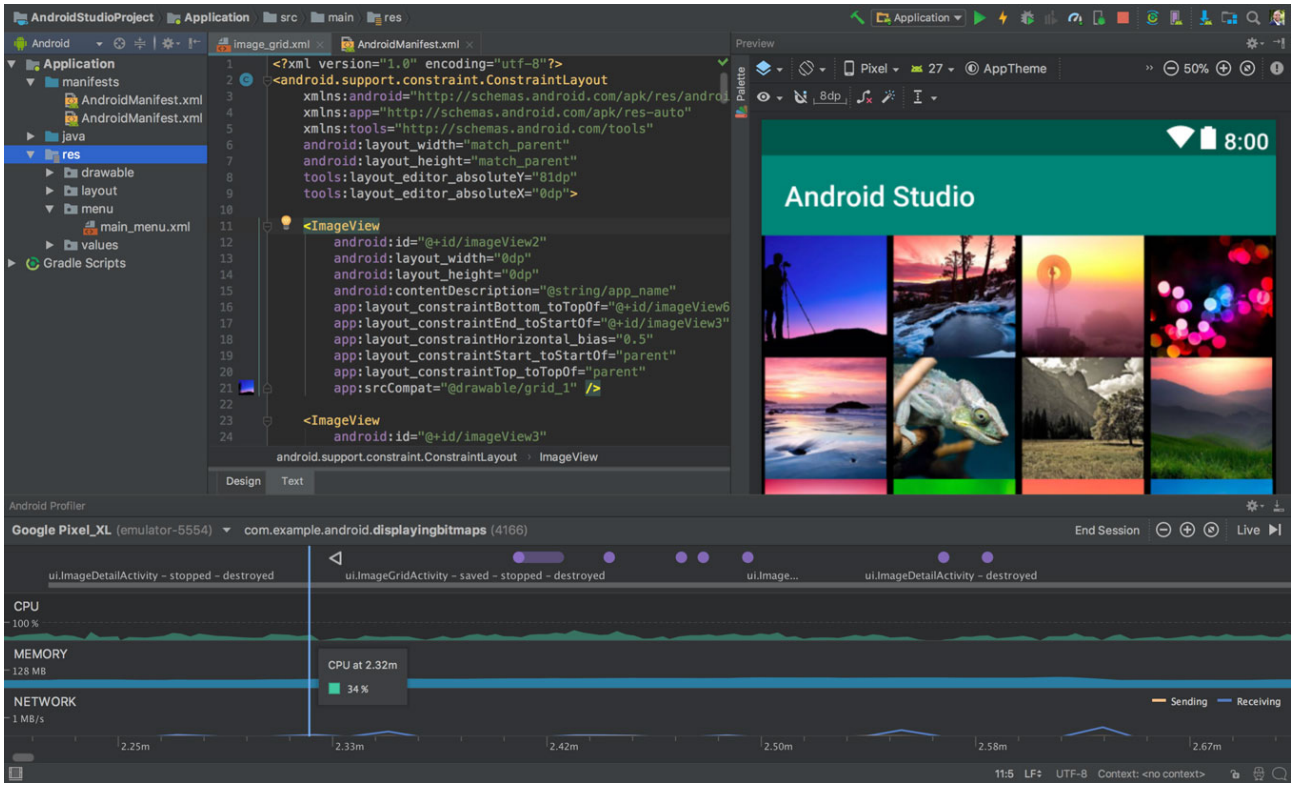

Рисунок 2.3 — Интерфейс Android Studio 3.0

2.2.3 Android Studio

Android Studio — это официальная интегрированная среда разработки для Android, на базе IntelliJ IDEA Community. Она обладает всеми преимуществами и возможностями IntelliJ IDEA и добавляет следующие инструменты для написания Android-приложений:

— эмулятор Android;

— шаблоны часто используемого при разработке приложений кода;

— поддержка C++ и NDK;

— редактор экранных форм;

— анализатор APK;

— профилировщик и инструменты отладки;

— возможность применения измененного кода без переустановки приложения (Instant Run) [15].

В настоящее время, Android Studio основывается на IntelliJ IDEA 2017.3.3, а значит не включает в себя новых функций, доступных в IDEA 2018. Это и есть основная проблема Android Studio. Изменения Android Studio и IntelliJ IDEA сливаются друг в друга перекрёстно, то есть команда Android отвечает за доработку инструментов Android-разработки, а JetBrains за улучшение самой IDE. IDEA часто обновляется и функционал из Android Studio сливают в неё быстрее, чем функционал новых версий IDEA в Android Studio.

Таблица 2.2 — Сравнение сред разработки для Android

Критерий	Eclipse [13]	Android Studio [15]	IntelliJ IDEA [14]
Поддержка Kotlin	±	+	+
Инструменты для Android-разработки	±	+	+
Понимание контекста	−	+	+
Частые обновления	+	−	+
Инструменты для работы с БД	−	−	+

\+ – указанная возможность присутствует

± – указанная возможность может быть добавлена при помощи плагина

− – указанная возможность отсутствует

По результатам проведенного обзора сред разработки составлена таблица 2.2. Выбрана IntelliJ IDEA, в силу того, что она содержит все возможности Android Studio, чаще обновляется и можно использовать возможности Ultimate версии.

2.3 Выбор целевой версии Android API

На официальном портале Android разработчиков размещена статистика, отображающая количество устройств, работающих под той или иной версией Android [16]. По этим данным составлена таблица 2.3, на основе которой построена круговая диаграмма (рис. 2.4), наглядно показывающая долю устройств для каждой версии Android.

Таблица 2.3 — Доля устройств для каждой версии Android API

Версия	Название	API	Доля устройств
2.3.3–2.3.7	Gingerbread	10	0.3%
4.0.3–4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.5%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	Kit Kat	19	10.3%
5.0	Lollipop	21	4.8%
5.1		22	17.6%
6.0	Marshmallow	23	25.5%
7.0	Nougat	24	22.9%
7.1		25	8.2%
8.0	Oreo	26	4.9%
8.1		27	0.8%

Данные, собранные за неделю 1–7 мая 2018 года

Версии с долей устройств меньше 0.1% не показаны

Android API обладает свойством обратно совместимости, а значит, приложение, написанное для старой версии API, будет работать на любой версии новее. Например, приложение с целевой версией API 19 будет работать на API 27. Исходя их этого можно посчитать, что целевая версия 19 даст 95% покрытие устройств, 21 — 84.7%, а 23 — всего 62.3%. В России эти показатели немного ниже, потому что доля устройств, с Android старее 4.4, выше, чем по миру [17].

Версии 19, 21 и 24 выбраны не случайно. Версию 19 чаще всего используют как целевую, т.к. она обеспечивает 95% покрытия устройств (в России 92.2%). Версия 21 тоже часто является целевой, потому что в ней была добавлена поддержка Material Design и SVG, что позволяет добиться одинакового вида приложения на всех устройствах. В версии 23 были добавлены функции, улучшающие безопасность: хранилища ключей, запрос разрешений в реальном времени и т.д.

По итогам сравнения целевых версий, представленных в таблице 2.4, выбрана версия 4.4, т.к. она обеспечивает максимальное покрытие устройств и есть возможность поддержки Material Design и SVG при помощи библиотек.

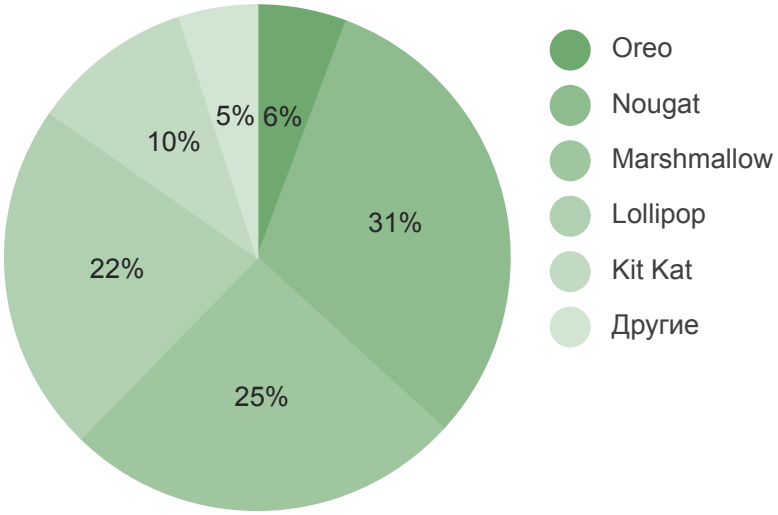

Рисунок 2.4 — Доля устройств для каждой версии Android

Таблица 2.4 — Сравнение целевых версий API

Версия	Покрытие устройств [17]	Material Design [18]	Поддержка SVG [18]	Безопасность: хранилища ключей, запрос разрешений в реальном времени и т.д. [18]
4.4 (19)	92.21%	±	±	−
5.0 (21)	80.76%	+	+	−
6.0 (23)	55.67%	+	+	+

\+ – указанная возможность присутствует

± – указанная возможность может быть добавлена при помощи библиотеки

− – указанная возможность отсутствует

2.4 Архитектура и алгоритм работы МП СУПС

Пользователь может запросить загрузку списка предметов, информации о текущей успеваемости, списка долгов, расписания и т.д. Чтобы максимально ускорить получение данных, используется двойное кэширование: кэш в памяти и кэш в БД. Более высокий приоритет у кэша в памяти. Если данных в нём нет, то происходит получение данных из БД, полученные данные сохраняются в память, но только при условии, если они достаточно свежие. Если в БД данных нет или они устарели, то производится запрос к API, и полученные данные сохраняются в БД и в память. Таким образом время ожидания данных сведено к минимуму.

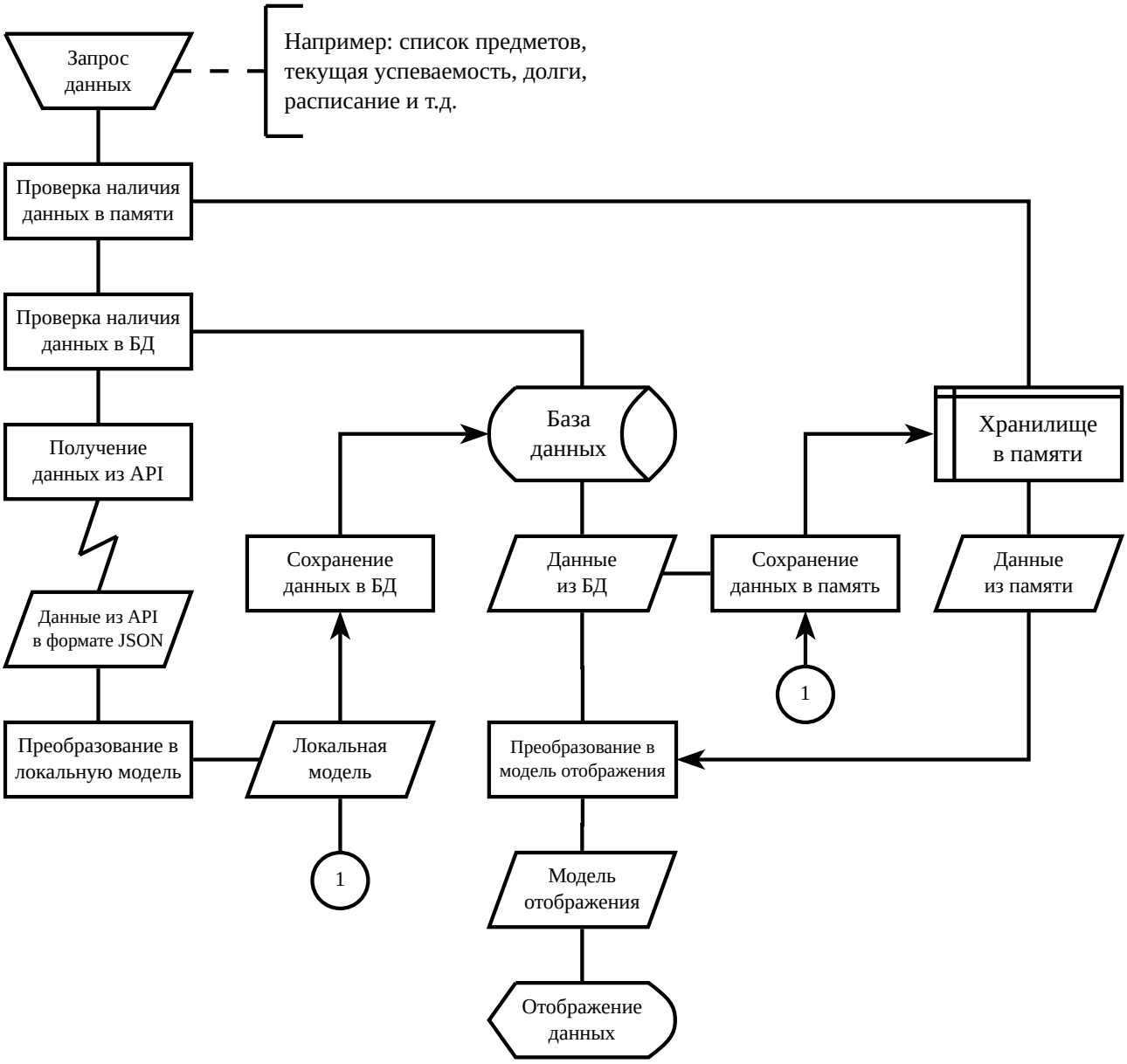

Рисунок 2.5 — Схема данных МП СУПС

2.5 Выбор стека технологий

Для разработки приложений нужно определиться с набором используемых инструментов, библиотек и фреймворков. Нужно выбрать:

— фреймворк или каркас приложения;

— DI фреймворк;

— библиотеки для работы с данными;

— библиотеку для навигации между экранами;

— технологию для выполнения асинхронных задач;

— библиотеки для тестирования.

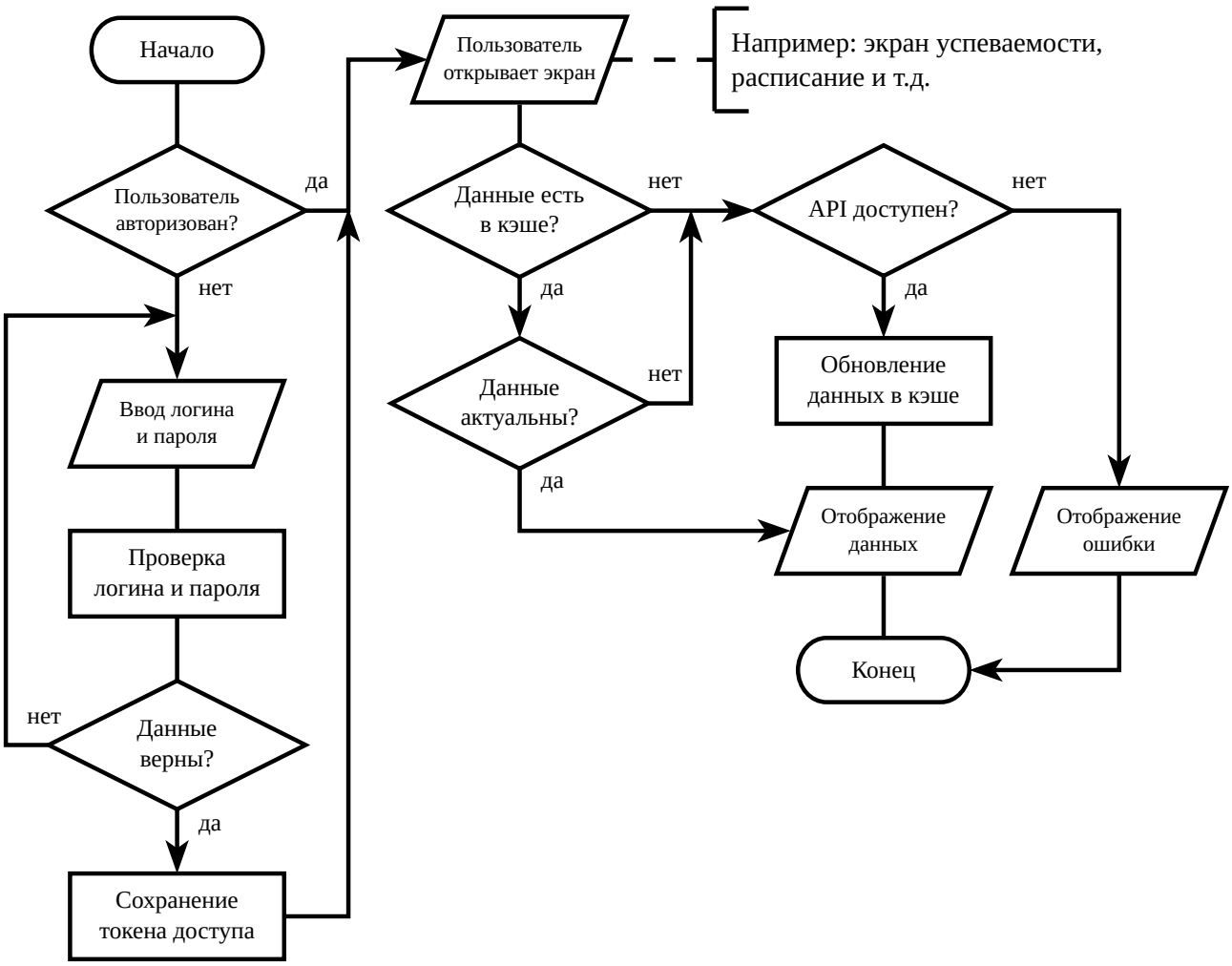

Рисунок 2.6 — Схема алгоритма МП СУПС

2.5.1 Каркас приложения

В Android разработке популярен паттерн проектирования, производный от MVC (Model–View–Controller) — MVP (Model–View–Presenter). Ответственность между слоями разделена следующим образом:

а) Model — содержит всю безнес-логику, получает данные из хранилища при необходимости;

б) View — отвечает за отображение данных модели;

в) Presenter — отвечает за взаимодействие слоёв View и Model, отвечает за обновление состояния View.

Этот паттерн решает часто возникающую проблему, когда Activity или Fragment обладает слишком большим количеством функций (антипаттерн God Object). Но в Android есть еще одна большая проблема, связанная с жизненным цик-

лом Activity и Fragment. Они не бессмертны и при изменении параметров, например при повороте экрана, пересоздаются заново, теряя своё состояние.

Лучше всего из существущих библиотек эту задачу решает Moxy. Это библиотека с открытым исходным кодом, разработанная Юрием Шмаковым (Aerllo-Mobile) и Александром Блиновым (RedMadRobot) в 2016 году [19]. Она полностью решает проблемы с жизненным циклом компонентов Android и предоставляет MVP каркас для приложения.

TODO: Добавить диаграмму работы Moxy

Для того, чтобы обеспечить сохранение и восстановление состояния, в Moxy используется ViewState. Он хранит в себе список команд, которые были применены Presenter'ом ко View и при появлении новой View применяет к ней все эти команды, что позволяет добиться восстановления состояния.

Итак, в качестве каркаса будет использована библиотека Moxy.

2.5.2 Внедрение зависимостей

Для соблюдения пятого принципа SOLID, а именно инверсии зависимостей, все необходимые для работы класса зависимости нужно передавать извне, а не создавать внутри. Передавать их вручную в конструктор не всегда удобно и гораздо удобнее делегировать задачу создания и предоставления зависимостей внешней библиотеке. Существует множество библиотек для решения этой задачи, но среди Android разработчиков наибольшей популярностью обладают Dagger2 и Toothpick.

Dagger2 — свободная библиотека разработанная компанией Google. Она строит дерево зависимостей и генерирует код для их создания и внедрения, то есть рефлексия для внедрения не используется, что является плюсом, так как рефлексия медленная [20]. Большим плюсом Dagger является обработка ошибок еще при компиляции, поэтому нет риска получить исключение во время выполнения. Основной претензией к библиотеке Dagger2 является сложность её освоения и сложность создания и контроля скоупов, из-за чего приходится писать свой менеджер скоупов. Стоит отметить, что менеджер скоупов можно написать один раз и потом переиспользовать с небольшими изменениями во всех проектах.

Библиотека Toothpick разрабатывается независимой командой разработчиков и тоже имеет открытый исходный код. Основной идеей является выстраивание дерева скоупов. Скоуп — это область действия той или иной зависимости. Создание но-

вых скоупов и зависимостей максимально упрощено. Принцип работы так же как и в Dagger основан на кодогенерации [21]. Из минусов: нет обработки ошибок на этапе компиляции, из-за чего есть риск получить ошибку во время выполнения, невозможно использовать Toothpick вместе со включённой функцией минификации APK.

При разработке приложения будет использоваться Dagger2, так как уже есть опыт работы с ним, есть реализация менеджера скоупов и нужно, чтобы работала минификация APK.

2.5.3 Работа с данными

Работа с данными разделяется на два типа: работа с API и работа с локальным хранилищем.

Для работы с API будет использоваться библиотека Retrofit 2 от команды Square. Это безопасный HTTP клиент для Android, позволяющий легко работать с сетью. Достаточно всего лишь создать интерфейс, методы в котором будут правильно аннотированы [22]. Например, вот код интерфейса для получения списка репозиториев пользователя из API GutHub:

```
interface GitHubService {
    @GET("users/{user}/repos")
    fun listRepos(@Path("user") user: String): Call<List<Repo>>
}
```

Листинг 2.5 — Пример интерфейса при использовании Retrofit

Для работы с локальным хранилищем есть две популярные библиотеки: Realm и Room. Realm — NoSQL база данных, Room же — абстракция над SQLite, позволяющая работать с БД как напрямую при помощи запросов, так и в режиме ORM [23, 24]. Room разрабатывается компанией Google и входит в набор официальных архитектурных компонентов, представленных на конференции Google I/O 2017. Room работает быстрее Realm примерно в два раза, кроме того Realm добавляет к весу APK целых 2.5 мегабайта [25]. Выбрана библиотека Room.

2.5.4 Навигация между экранами

Навигация между экранами без использования библиотек достаточно сложна, особенно если использовать подход “одна `Activity`, много `Fragment`’ов”. Кроме то-

го, для перехода на новый экран в Android нужен `Context`, а значит из слоя Presenter инициировать такую навигацию небезопасно, т.к. есть риск получить утечку памяти, если забыть удалить ссылку на контекст. Немаловажно, что наличие зависимости от контекста сильно усложняет тестирование кода.

Для упрощения задачи навигации существует библиотека Cicerone, написанная Константином Цховребовым (RedMadRobot). Это легковесная библиотека, которая спроектирована для использования с паттерном MVP.

TODO: Добавить диаграмму работы Cicerone

Есть несколько базовых классов: `Router`, `CommandBuffer`, `Navigator`, `Command`. Принцип работы заключается в том, что навигацией занимается отдельный класс — `Router`. Для навигации программист вызывает его методы (`navigateTo(screenKey)`, `exit()`, `replaceScreen(screenKey)` и другие). У роутера есть буфер команд в который добавляются все команды, посылаемые роутеру, обёрнутые в абстракцию Command, и в последствии все они применяются к навигатору. Это сделано для того, чтобы иметь возможность посылать команды роутеру даже до того, как он привязан к навигатору. `Navigator` — это класс ответственный за применение команд к `FragmentManager`'у или другой сущности, которая отвечает за навигацию. Навигатор может быть в любой момент привязан или отвязан от роутера, в навигаторе содержится логика какой ключ какому экрану соответствует.

2.5.5 Асинхронные задачи

Для асинхронных задач можно использовать функции обратного вызова (callbacks), реактивный подход (библиотеку RxJava) или сопрограммы (экспериментальная возможность Kotlin).

Проблема использования функций обратного вызова в том, что при увеличении их количества и количества кода внутри, читаемость кода сильно ухудшается. Это явление называется callback hell, и для него характерна большая вложенность.

RxJava — это библиотека для составления асинхронных цепочек. Она основывается на системе событий и слушателей, по сути расширяет паттерн GoF Наблюдатель (Observer) для создания цепочек данных или событий [26]. Цепочки можно воспринимать как потоки данных, которые можно комбинировать между собой, применять к ним фильтры и другие операторы. Цепочкам можно назначать контекст выполнения, причем можно часть цепочки выполнить в одном, а часть в другом

контексте. Каждый контекст — отдельный поток. Например, в контексте IO можно выполнять долгую операцию получения данных из API, а отображение результата должно делаться в UI потоке. Проблемой и реактивных цепочек является сложность их отладки и сложность кода, который получается в итоге. Код разбросан по разным звеньям цепи и зачастую нет возможности охватить всю цепочку.

TODO: Картинка фильтра Rx цепочки

Сопрограммы (coroutines) — экспериментальная возможность Kotlin. Они более легковесны чем потоки, то есть операция приостановки и запуска сопрограммы гораздо дешевле по ресурсам чем остановка и запуск потока. Достигается это зачёт того, что несколько сопрограмм работают в одном потоке, и вместо блокировки потока, сопрограмма приостанавливается, не мешая при этом работать остальным сопрограммам. Переключение между сопрограммами производится при помощи стейт-машины, переключение состояния в которой происходит при достижении точки приостановки. Такими точками являются функции приостановки (suspending functions). Пример такой функции — функция ожидания `delay(time: Int)`. Сопрограммы позволяют писать асинхронный код в императивном стиле [27].

Для примера, сравним эти три подхода. Например, у нас есть задача отображения информации о пользователе из API по нажатию на кнопку.

```
button.setOnClickListener {
    // Получаем данные пользователя из API (долгая операция)
    val rawData = getUserData("test")
    // Десериализуем пользователя (долгая операция)
    val user = deserializeUser(rawData)
    showUserData(user)
}
```

Листинг 2.6 — Изначальный код, который нужно сделать асинхронным

Применим первый подход. Добавим в методы `getUserData(...)` и `deserializeUser(...)` функции обратного вызова:

```
button.setOnClickListener {
    getUserData("test") { rawData ->
        deserializeUser(rawData) { user ->
            showUserData(user)
        }
    }
}
```

Листинг 2.7 — Асинхронный код с функциями обратного вызова

Применим второй подход, используем библиотеку RxJava. При этом подходе `getUserData(...)` будет возвращать тип `Single<String>` и дальше будет выстраиваться цепочка, которая сначала десериализует пользователя, а потом отобразит его данные:

```
button.setOnClickListener {
    getUserData("test")
            .flatMap { rawData ->
                deserializeUser(rawData)
            }
            .subscribe { user ->
                showUserData(user)
            }
}
```

Листинг 2.8 — Асинхронный код с реактивной цепочкой

И, наконец, применим третий подход. Используем механизм запуска неблокирующей сопрограммы при помощи функции `launch(context: Context)`:

```
button.setOnClickListener {
    launch(UI) {
        val rawData = getUserData("test")
        val user = deserializeUser(rawData)
        showUserData(user)
    }
}
```

Листинг 2.9 — Асинхронный код с применением сопрограмм

Как видно из примера, код с применением сопрограмм наиболее понятен и похож на исходный. Кроме того, не требуется вносить изменения в функции `getUserData(...)` и `deserializeUser(...)`.

При разработке МП СУПС будет использоваться комбинация сопрограмм и Rx. Причина этому — отсутствие в некоторых библиотеках интеграции с сопрограммами и наличие интеграции с RxJava. Типы RxJava могут быть конвертированы в каналы, используемые в сопрограммах, поэтому эти технологии можно использовать вместе.

2.5.6 Тестирование

Для юнит-тестирования будет использоваться Junit 5. Это самый популярный фреймворк для юнит-тестирования на языке Java. Распространяется под свободными лицензиями: Apache License v2.0 и Eclipse Public License v2.0 (разные модули распространяются под разными лицензиями). Для его работы требуется Java 8, но энтузиасты уже написали адаптацию Junit 5 для Android [28] и Java 7.

Для того, чтобы тесты были понятнее, будет использоваться библиотека Spek, которая позволяет писать тесты в виде исполняемых спецификаций [29]. Библиотека появилась недавно, но подобная библиотека RSpec для Ruby уже существуетс 2007 года и имеет успех [30]. Типичный тест JUnit выглядит следующим образом:

```
class CalculatorTest {
    private calculator: Calculator

    @Before
    fun setUp() {
        this.calculator = Calculator()
    }

    @Test
    fun testSum_whenPassedTwoNumbers_shouldReturnResultOfSum() {
        val result = calculator.sum(200, 10)
        assertEquals(210, result)
    }

    @Test
    fun testSubtract_whenPassedTwoNumbers_shouldReturnResultOfSubtract() {
        val result = calculator.subtract(200, 10)
        assertEquals(180, result)
    }
}
```

Листинг 2.10 — Пример теста JUnit

Этот код имеет несколько проблем. Длинные названия методов плохо читаются, но даже при такой длине названия можно его толковать по разному. В случае с тестом калькулятора кривотолки вряд ли возникнут, но в более сложных тестах их вероятность высока. Если называть методы не так длинно, то не будет понятно что должен делать метод и как именно он должен это делать. При увеличении количества тестовых сценариев возрастает количество методов и часто возникает дублирование кода. Spek решает все эти проблемы. В листинге 2.11 представлены те же тесты для калькулятора, написанные с применением библиотеки Spek.

```
object CalculatorSpec : Spek({
    given("a calculator") {
        val calculator = Calculator()

        on("addition") {
            val actual = calculator.sum(200, 10)
            it("should return the result of adding the first number to the
            ↪ second number") {
                assertEquals(210, actual)
            }
        }

        on("subtraction") {
            val actual = calculator.subtract(200, 10)
            it("should return the result of subtracting the second number from
            ↪ the first number") {
                assertEquals(180, actual)
            }
        }
    }
})
```

Листинг 2.11 — Пример теста с использованием Spek

Со Spek не составит труда дописать дополнительные тесты для каждого метода, избежав дублирования кода.

Для тестирования пользовательского интерфейса будет использоваться библиотека Espresso. Она позволяет имитировать нажатия, касания и прочие действия над интерфейсом и проверять состояние графических элементов.

2.6 Разработка пользовательского интерфейса

TODO: Написать

Выводы по разделу

TODO: Написать

3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

В данном разделе описываются использованные технологии разработки, сборки и публикации, отладки и тестирования Android-приложений.

3.1 Разработка под Android

В данном подразделе рассматриваются популярные архитектурные подходы при написании приложений для Android, а так же описывается структура проекта.

3.1.1 Архитектура приложения

Чаще всего Android-разработчиками используется подход под названием “Clean Architecture” или “чистая архитектура”. Изначально подход был сформулирован Бобом Мартином еще в 2012 году[31] и использовался не только в Android-разработке. Основная идея подхода состоит в том, чтобы написать приложение, которое:

— не зависит от пользовательского интерфейса;

— не зависит от внешних феймворков, хранилища информации и библиотек;

— легко тестируется.

Таким образом, в хорошо спроектированном приложении можно “откладывать” решения до того момента, когда их действительно необходимо принять. Если вместо одной технологии хранения появится другая более привлекательная, или если используемая технология перестанет справляться с нагрузкой — возможность лёгкой смены решения сыграет на руку. В итоге получается слоистая и гибкая архитектура, единый подход в осмыслении всего приложения [32].

Изначально Роберт Мартин изобразил архитектуру как четырёхслойную “луковицу”, у которой в центре находится слой Entity, а снаружи UI, Web, DB, но удобнее представлять чистую архитектуру в виде линейной диаграммы, изобрженной на рисунке 3.1.

Стрелки показывают потоки данных между слоями. Например, событие, инициированное пользователем, идет в Presenter, тот образается к Use Case. Use Case делает запрос в Repository, который получает данные, создает Entity и передает его в UseCase. Так Use Case получает все нужные ему Entity. Затем, применив их и свою

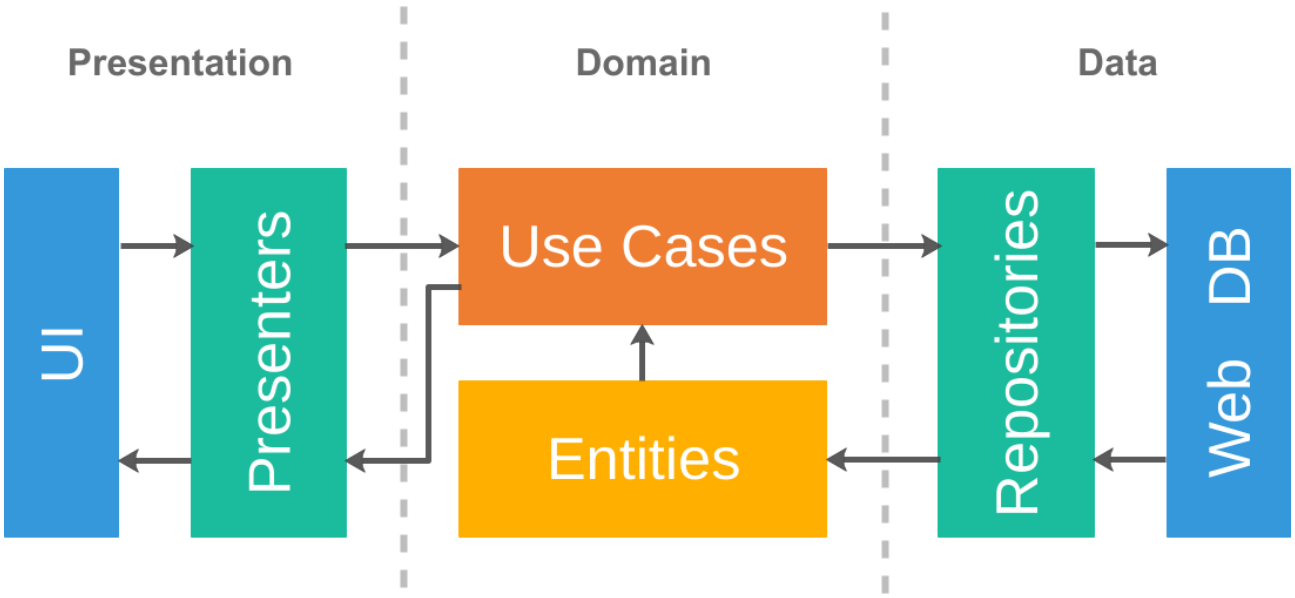

Рисунок 3.1 — Линейная диаграмма, изображающая слои чистой архитектуры

логику, получает результат, который передает обратно в Presenter. А тот, в свою очередь, отображает результат в UI [33].

Разработку приложения удобнее вести сверху вниз, то есть сначала реализовывать внешние слои и постепенно углубляться в внутренним. Это позволяет максимально быстро увидеть результат, кроме того, всегда удобно отталкиваться от того, что должен видеть пользователь.

Как видно из рисунка 3.1, проект удобно делить на три части: presentation, domain и data. Модуль presentation зависит от Android API, так как взаимодействует с графическим интерфейсом и прочими функциями системы. Слой data, так же, частично зависит от Android API т.к. в нём происходит взаимодействие с сетью, файлами и базами данных. Слой domain напротив полностью независим от Android и в нём содержится вся бизнес-логика приложения.

3.1.2 Структура проекта

Есть два типа разбиения структуры проекта: по слоям и по возможностям (или функциям). При разбиении проекта по слоям возникает риск захламлённости пакетов, т.к. при использовании чистой архитектуры количество классов увеличивается довольно быстро. Разбиение по возможностям, мало того, что лишено этого недостатка, но еще и обладает дополнительными преимуществами:

а) Можно узнать о возможностях приложения только посмотрев на структуру папок;

б) Легко добавлять новые возможности в приложение;

в) Легко удалять возможности, для этого достаточно лишь удалить директорию;

г) Легко ориентироваться в коде, когда он организован таким образом;

д) Позволяет переиспользовать код в других проектах;

е) Разработчики могут работать независимо друг от друга в разных пакетах [34].

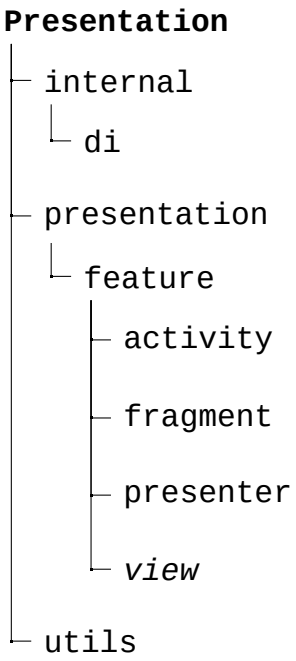

Рисунок 3.2 — Структура модуля Presentation

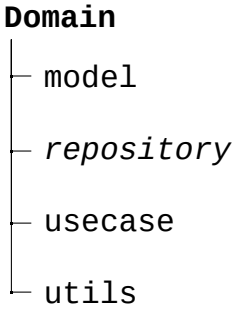

Рисунок 3.3 — Структура модуля Domain

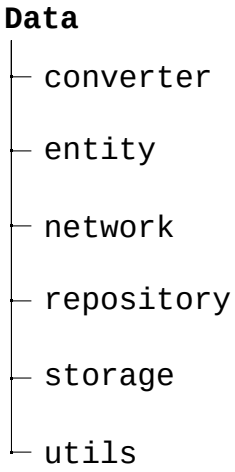

Рисунок 3.4 — Структура модуля Data

При разбивке проекта по возможностям, структура проекта имеет вид как на рисунках 3.2–3.4. Жирным шрифтом выделены названия модулей, а курсивом директории, которые содержат только абстракции.

3.2 Сборка и публикация приложения

В этом подразделе описываются:

— процессы сборки и публикации приложения;

— работа с Gradle в Android-проектах;

— подпись APK файла приложения;

— настройка сервиса непрерывной интеграции для работы с Android-проектом.

3.2.1 Система автоматической сборки Gradle

Для сборки приложения в APK файл при Android-разработке обычно используется Gradle. Это система автоматизации сборки с открытым исходным кодом, ориентированная на гибкость и производительность. В отличие от аналогов (Maven и Ant), сценарии сборки пишутся на Groovy или Kotlin DSL, а не на XML. Это позволяет максимально снизить порог вхождения для программиста [35].

Gradle является официальным инструментом сборки для Android и поддерживает многие популярные языки и технологии. Время сборки сильно уменьшается за счёт того, что при сборке используются результаты прошлых сборок, выполняется сборка только измененных файлов (инкрементальная сборка) и независимые между собой задачи выполняются параллельно. Gradle поддерживается всеми IDE для Android-разработки, в том числе IntelliJ IDEA.

При создании нового проекта в IDEA можно выбрать какой DSL использовать для сценариев сборки. В процессе разработки МП СУПС используется Kotlin DSL, т.к. язык проекта — Kotlin. После указания основных настроек (название приложения, домен, целевая версия и т.д.) создаются стандартные сценарии сборки для всего проекта и для модуля 'app', в котором содержится код приложения. Это обусловлено тем, что в проекте может быть несколько модулей, например, модули data, domain и presentation при использовании чистой архитектуры. Gradle и Kotlin DSL обновляются чаще чем инструменты разработки Android, поэтому созданные по умолчанию сценарии сборки могут не работать в силу того, что Kotlin DSL находится в стадии активной разработки и не гарантирует обратную совместимость, а шаблоны рассчитаны на старые версии.

Помимо сценариев сборки в Gradle есть еще файлы настроек:

— `gradle.properties` содержит настройки Gradle (здесь, например, можно выставить ограничение потребления памяти);

— `settings.gradle.kts` содержит список модулей, которые нужно подключить;

— `gradle-wrapper.properties` (находится по пути `gradle/wrapper/`) содержит настройки обёртки Gradle (Gradle wrapper).

Gradle позволяет дописывать собственные вспомогательные классы, которые можно будет использовать при написании сценария сборки. Такие классы помещают в модуль `buildSrc`, который подключён по умолчанию и этот собирается перед сбор-

кой остальных модулей. Еще одна возможность организации структуры сценариев сборки — разбивка сценария на несколько файлов, отвечающих за разные аспекты сборки. Например, выделить отдельный файл, в котором будет собрана вся работа с зависимостями. В Gradle для подключения внешних файлов сценариев предусмотрена функция:

`project.apply { from(”path/to/script.gradle.kts”) }`

Все необходимые задачи, такие как настройка проекта для работы с Android, сборка приложения в APK, подпись его ключом и пр. выделены в отдельный плагин — Android Gradle Plugin. Подключение этого плагина осуществляется в сценарии уровня проекта в секции `buildscript.dependencies` (листинг 3.1).

```
buildscript {
    ...
    dependencies {
        classpath("com.android.tools.build:gradle:3.0.1")
    }
}
```

Листинг 3.1 — Подключение Android Gradle Plugin версии 3.0.1

После того, как плагин подключен, в сценарии уровня модуля, нужно указать настройки сборки Android приложения, а именно, целевую версию, минимальную допустимую версию, версию инструментов сборки, ID и версию приложения (листинг 3.2).

```
android {
    compileSdkVersion(27)
    buildToolsVersion("27.0.3")

    defaultConfig {
        minSdkVersion(19)
        targetSdkVersion(27)

        applicationId = "ru.endlesscode.miet.orioks"
        versionCode = 1
        versionName = "0.1.0"
    }
}
```

Листинг 3.2 — Настройка сборки Android-приложения

3.2.2 Управление зависимостями

Ни одно Android-приложение не обходится без зависимостей. Проект всегда использует бибилиотеки, а так же может быть разбит на отдельные модули, которые зависят друг от друга. Управление зависимостями — это метод декларирования, разрешения и использования зависимостей, требуемых проектом, в автоматическом режиме.

В Gradle есть инструменты для управления зависимостями, которые покрывают все типичные сценарии, встречающиеся в современных проектах по разработке ПО.

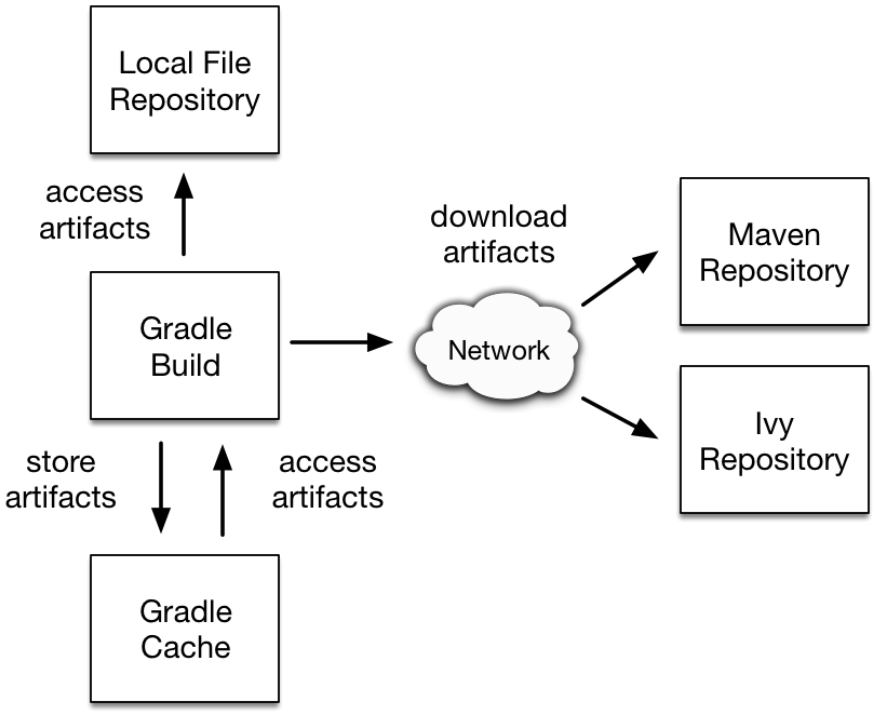

Рисунок 3.5 — Управление зависимостями в Gradle

Например есть проект, написанный на Java. В некоторых файлах используются классы из Google Guava (библиотека с открытым исходным кодом, предоставляющая множество полезных функций). Кроме того, проекту нужен JUnit для компиляции и выполнения тестов.

Guava и JUnit представляют собой зависимости этого проекта. Программист, при написании сценария сборки, может определять зависимости для разных областей (scopes), например, для компиляции всего кода или для компиляции и выполнения тестов. В Gradle область действия зависимости называется “конфигурацией”.

Обычно зависимости подключаются в виде модулей. Необходимо указать Gradle, где найти эти модули, чтобы их можно было использовать во время сборки. Место хранения модулей называется репозиторием. После указания репозиториев для сборки, Gradle будет знать, как находить и получать модули. Репозитории могут быть разных типов: локальный каталог и удаленный или локальный репозиторий. Для обеспечения совместимости с популярными хранилищами модулей Gradle может использовать Maven и Ivy репозитории

Gradle определяет требуемые зависимости во время выполнения сценария сборки. Зависимости могут быть загружены из удаленного репозитория, извлечены из локального каталога или могут потребовать сборки другого проекта при многопроектной конфигурации. Этот процесс называется разрешением зависимостей.

После разрешения Gradle хранит файлы зависимости в локальном кэше, называемом кэшем зависимостей. При повторной сборке будут повторно использованы файлы, хранящиеся в кэше, чтобы избежать ненужных сетевых вызовов и ускорить сборку.

Модули могут предоставлять метаданные, то есть данные, которые дополнительно описывают модуль, например, координаты для нахождения его в репозитории, актуальную версия модуля, информацию об авторах и т.д. В рамках метаданных модуль может объявить, что для его работы необходимы другие модули, а Gradle автоматически разрешает эти дополнительные зависимости, которые называют “транзитивными зависимостями”.

Проекты с большим количеством зависимостей могут пострадать от антипаттерна, называемого “ад зависимостей” (dependency hell). Для борьбы с этим Gradle предоставляет инструментарий для визуализации и анализа графа зависимостей проекта.

Объявление зависимостей

Типичным примером библиотеки в Java-проекте является JUnit 4 (в Kotlin-проекте эта библиотека может быть заменена модулем `kotlin-test-junit`).

```
1  plugins {
2      java
3  }
4
5  repositories {
6      mavenCentral() // Подключение репозитория
7  }
8
9  dependencies {
10     testImplementation("junit:junit:4.12") // Подключение зависимости
11 }
```

Листинг 3.3 — Подключение JUnit 4 в качестве зависимости

В листинге 3.3 приведен пример кода, который подключает JUnit в проект как зависимость для сборки и выполнения тестов. В строке 6 подключается репозиторий Maven Central, в котором находится модуль JUnit. В строке 10 происходит объявление самого JUnit, для этого указываются координаты артефакта в формате `<group>:<module>:<version>`. В начале строки указана область действия зависимости или, в терминологии Gradle, конфигурация — `testImplementation`. Существует много различных конфигураций, но самые часто используемые это `implementation` и `testImplementation`. В первом случае зависимость используется для компиляции всего проекта, во втором только для компиляции тестового кода.

Проекты могут использовать более агрессивный подход объявления зависимостей. Например, подключать всегда последнюю версию модуля. Динамическая версия позволяет подключить последнюю версию или последнюю подверсию версии модуля. Стоит помнить, что использование динамических версий несет потенциальный риск нарушения сборки. Код перестанет компилироваться как только новая версия библиотеки не будет обратно совместима с использованной до этого. Чтобы минимизировать риск можно динамически разрешать только минорную или патч версию и только при условии, что разработчик подключаемого модуля использует семантическое версионирование. Gradle проверяет наличие новой версии библиотеки по прохождении 24 часов.

```
plugins {
    java
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation("junit:junit:4.+")
}
```

Листинг 3.4 — Подключение JUnit 4 в качестве зависимости с динамически разрешаемой версией

В сценарии, показанном в листинге 3.4 версия `4.+` будет разрешена в версию `4.12`, но как только появится новая минорная или патч версия, будет использоваться она.

Бывает, что разработчики публикуют “снимки” (snapshots) библиотеки при разработке новой версии. В таком случае версия библиотеки не меняется, но сама библиотека обновляется. Для разрешения такой ситуации в Gradle достаточно добавить к версии постфикс `-SNAPSHOT`. По умолчанию, наличие измененной версии проверяется раз в 24 часа.

3.2.3 Подпись приложения

После того как приложение собирается, нужно его подписать цифровой подписью с сертификатом, иначе его будет невозможно установить на Android-устройство. В Android Gradle Plugin по умолчанию есть две конфигурации сборки: отладчная (debug) и релизная (release). При отладочной сборке приложение подписывается общим отладочным ключом и доступно для профилировки. При релизной сборке приложение должно быть подписано собственным сертификатом.

Сертификат открытого ключа, также известный как цифровой сертификат, содержит открытый ключ из пары открытого-закрытого ключей, а также некоторые другие метаданные, идентифицирующие владельца ключа (например, имя, название компании, город проживания). Только владелец сертификата имеет соответствующий закрытый ключ.

При подписи, инструмент подписи прикрепляет к APK сертификат открытого ключа, который служит “отпечатком пальца”, однозначно связывающим APK с владельцем, обладающим приватным ключом. Это помогает Android гарантировать, что обновления приложения являются подлинными и исходят от оригинального автора. Ключ, используемый для создания этого сертификата, называется ключом подписи. Хранилище ключей — это бинарный файл, содержащий один или несколько закрытых ключей.

Так как ключ подписи используется для подтверждения, что обновления выпущены действительно разработчиком, сохранение его в тайне является важной задачей как для разработчика, так и для пользователей его приложений. Можно использовать технологию Google Play App Signing для безопасного управления и хранения ключа подписи подписи приложения при помощи инфраструктуры Google или же самостоятельно хранить ключ подписи в надёжном месте.

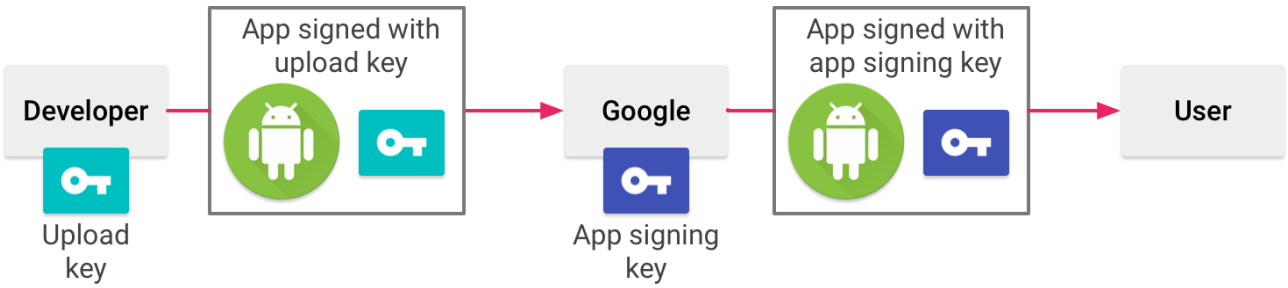

Рисунок 3.6 — Подпись приложения при помощи Google Play App Signing

В случае использования Google Play App Signing, есть два ключа: ключ загрузки и ключ подписи. Ответственность за сохранность в тайне ключа подписи лежит на Google, а ключ загрузки находится у разработчика. Ключом загрузки разработчик подписывает приложения для загрузки в Google Play, для того чтобы подтвердить, что обновление подлинно. Ключ подписи же загружается в Google Play один раз и приложение переподписывается им при публикации.

Отличие системы двухступенчатой подписи (рисунок 3.6) от одноступенчатой (рисунок 3.7) в том, что если ключ загрузки будет скомпрометирован, то его можно сменить сохранив ключ подписи. А в силу того, что ключ подписи хранится только на серверах Google, можно не беспокоиться, что он будет скомпрометирован.

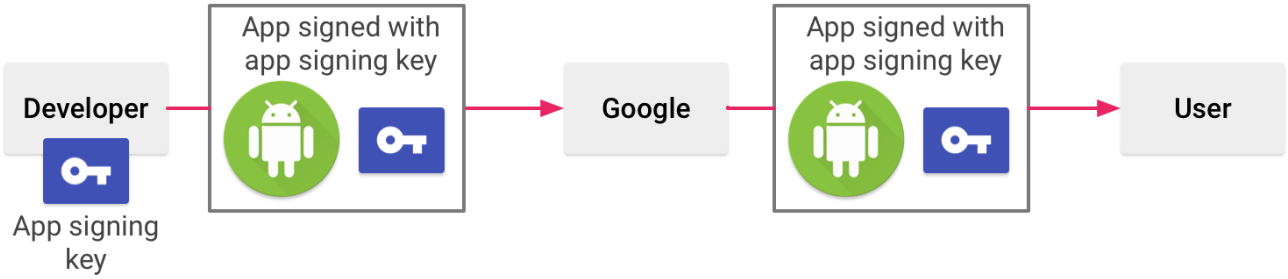

Рисунок 3.7 — Подпись приложения при самостоятельном управлении ключом подписи

Процесс подписи APK

Вне зависимости от того какой способ хранения ключей был выбран, необходимо создать хранилище ключей и ключ. Для этого можно использовать встроенные в Android Studio и IntelliJ IDEA инструменты.

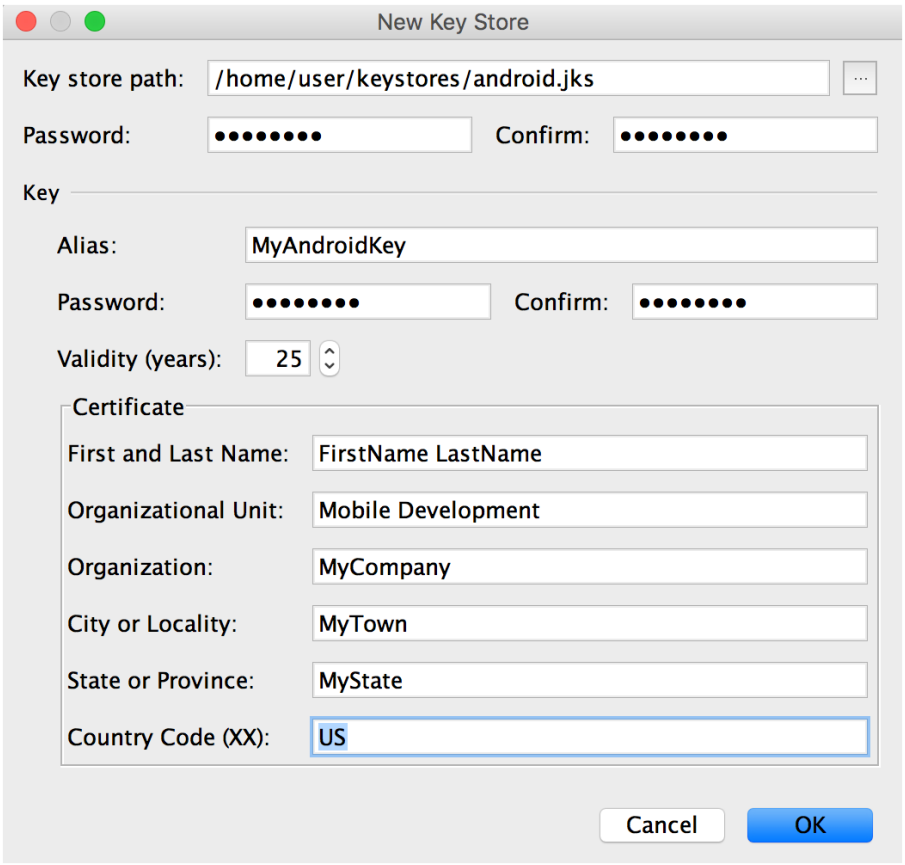

Рисунок 3.8 — Создание нового ключа и хранилища ключей

В верхнем меню нужно выбрать `Build > Generate Signed APK`, после чего выбрать модуль в выпадающем меню и нажать `Next`. Если нужно создать но-

вый ключ и хранилище, нужно нажать `Create new`. Откроется окно `New Key Store` (см. рисунок 3.8), в котором нужно заполнить все поля и нажать `OK`.

После того как хранилище ключей и ключ созданы, нужно настроить подпись приложения в Gradle. Для этого сначала надо добавить конфигурацию подписи, а после, ссылку на неё в релизную конфигурацию сборки как это сделано в листинге 3.5.

```
android {
    ...
    signingConfigs {
        create("release") {
            keyAlias = "keyAlias"
            keyPassword = "keyPassword"
            storeFile = "storeFile"
            storePassword = "storePassword"
        }
    }

    buildTypes {
        getByName("release") {
            signingConfig = signingConfigs.getByName("release")
        }
    }
}
```

Листинг 3.5 — Конфигурация подписи APK в Gradle

Чтобы сохранить в тайне пароль от ключа и хранилища ключей при использовании VCS, можно вынести эти настройки в отдельный файл `.properties`, который не будет включён в систему контроля версий, и считывать их из него перед подписью. Важно, чтобы файл хранилища ключей ни в коем случае не был включён в систему контроля версий.

3.2.4 Публикация приложения

Публикация приложения — это основная цель разработки, т.к. это процесс, который делает приложение доступным для пользователей. До, непосредственно, публикации приложение нужно подготовить. Помимо электронной подписи, как минимум нужно удалить из приложения логирование и отладочные атрибуты из манифеста, а так же добавить атрибуты `android:versionCode` и `android:versionName`. Android Gradle Plugin делает это автоматически при переключении в режим релизной

сборки. Так же, необходимо тщательно протестировать приложение как минимум на одном смартфоне и одном планшете с целевой версией Android. Если приложение зависит от внешних сервисов, то нужно убедиться, что сервисы доступны, безопасны и готовы к работе [36].

Приложение можно опубликовать несколькими способами. Обычно для публикации используют рынки приложений, но можно, так же, опубликовать приложение на своём сайте или рассылать его непосредственно пользователям.

Публикация на рынках приложений позволяет добиться максимального охвата аудитории, но это не всегда бывает нужно, например, на стадии разработки удобнее рассылать промежуточные сборки заказчику.

Главным рынком приложений является Google Play. Это надёжная платформа для публикации, которая позволяет продавать или распространять бесплатно Android-приложения по всему миру. При публикации приложения на Google Play, разработчик получает набор инструментов, позволяющих анализировать продажи, тенденции на рыке, следить за ошибками во время работы приложения и т.д. Для публикации приложения на Google Play необходим аккаунт разработчика, для регистрации которого нужно заплатить 25 USD.

3.2.5 Непрерывная интеграция

Непрерывная интеграция (CI) — это практика разработки, когда интеграция между членами команды происходит часто и при небольших изменениях кода, а не только в конце цикла разработки. Целью является создание более здорового программного обеспечения путем разработки и тестирования с небольшими приращениями.

В качестве платформы непрерывной интеграции выбран Travis CI, так как он поддерживает сборку Android-проектов, является открытым, бесплатным для Open Source проектов и не требует установки на собственный сервер. Travis CI поддерживает процесс разработки, автоматически собирая и тестируя приложения, как только обнаруживает изменения кода в VCS репозитории и обеспечивает немедленную обратную связь об успехе или неудаче изменений [37].

```
language: android # Язык проекта выставляется не java, а android
env:
  # Переменные, которые могут быть использованы позже
  global:
  - TARGET_API_LEVEL=27
  - MIN_API_LEVEL=19
  - BUILD_TOOLS_VERSION=27.0.3
android:
  # Список необходимых компонентов
  components:
  - tools
  - platform-tools
  - tools
  - build-tools-$BUILD_TOOLS_VERSION
  - android-$TARGET_API_LEVEL
  - android-$MIN_API_LEVEL
  - extra-google-m2repository
  - extra-android-m2repository
  licenses:
  - android-sdk-preview-license-.+
  - android-sdk-license-.+
  - google-gdk-license-.+
before_install:
- chmod +x gradlew # Необходимо сделать скрипт запуска Gradle исполняемым
- "./gradlew dependencies || true" # Разрешение зависимостей перед сборкой
```

Листинг 3.6 — Файл конфигурации Travis CI для сборки Android-проекта

Travis CI хранит настройки в файле `.travis.yml` в формате YAML. Для сборки Android-приложения нужно указать настройки, аналогично тому как они указаны в листинге 3.6.

Сборка может быть долгой из-за того, что Travis CI каждый раз собирает приложение “с чистого листа”. То есть каждый раз создаётся виртуальное окружение, в котором устанавливается JDK, подтягиваются зависимости и т.д., а значит кэширование Gradle не работает. Чтобы ускорить сборку, можно включить кэширование, для этого нужно указать в настройке `cache.directories` директории, которые используются для кэша (см. листинг 3.7).

Удобно использовать сервисы CI для сборки и публикации промежуточных версий приложения при разработке. Для того, чтобы настроить автоматическую публикацию приложения в GitHub Releases нужно немного изменить сценарий сборки, а именно, нужно организовать подпись приложения, при этом не скомпрометировав

```
# Удаление данных, которые не нужно сохранять в кэше
before_cache:
- rm -f  $HOME/.gradle/caches/modules-2/modules-2.lock
- rm -fr $HOME/.gradle/caches/*/plugin-resolution/
cache:
  # Директории, используемые для хранения кэша
  directories:
  - "$HOME/.gradle/caches/"
  - "$HOME/.gradle/wrapper/"
  - "$HOME/.android/build-cache"
```

Листинг 3.7 — Настройки Travis CI для сохранения кэша Gradle и Android

ключ подписи. Для этого в Travis CI предусмотрены инструменты шифрования, которые позволяют безопасно добавлять чувствительные данные в VCS репозиторий, не боясь утечки.

Travis CI использует асимметричный алгоритм шифрования и создаёт пару RSA ключей для каждого репозитория. Публичный ключ доступен всем, а приватный ключ Travis CI хранит в тайне, таким образом любой пользователь может зашифровать данные публичным ключом, но расшифровать их сможет только Travis CI.

Для упрощения шифрования данных есть утилита Travic CLI. Чтобы зашифровать файл достаточно авторизоваться и написать одну команду:

```
$ travis encrypt-file <имя_файла> --add
```

После чего файл будет зашифрован и настройки для его расшифровки будут автоматически добавлены в файл конфигурации `.travis.yml`.

Может понадобиться зашифровать не только файл, но и строковое значение, которое нужно указать в конфиге, например токен доступа к API GitHub для публикации релиза. Для такого случая вместо команды `encrypt-file` нужно использовать команду `encrypt`, например:

```
$ travis encrypt "токен_доступа"
```

Полученное значение можно использовать в настройках, оно будет автоматически расшифровано.

Другой способ безопасного добавления строковых констант — добавление их через веб-интерфейс Travis CI (см. рисунок 3.9), где достаточно написать название

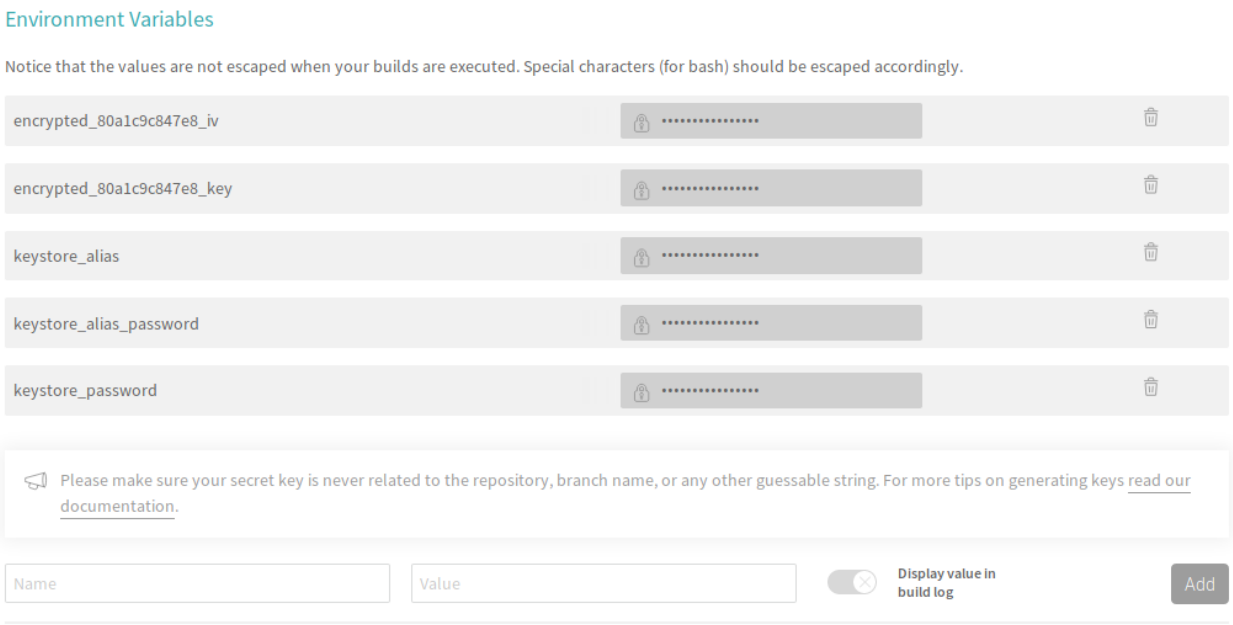

Рисунок 3.9 — Веб-интерфейс Travis CI для добавления переменных окружения

переменной окружения, её значение и отключить отображение переменной в логах. Этот способ проще, чем шифровка значений при помощи утилиты.

Для подписи приложения, нужно алиас ключа, пароль от ключа и пароль от хранилища ключей поместить в переменные окружения, а сам файл хранилища ключей зашифровать и добавить в систему контроля версий. После того как эти действия выполнены, надо изменить конфигурацию подписи, чтобы при сборке из Travis CI конфигурации подписи брались из переменных окружения. Результат можно увидеть в листинге 3.8.

После того как приложение успешно подписывается, можно настроить автоматическую публикацию его в GitHub Releases. Для этого в Travis CI есть секции `deploy` и `before_deploy`. Пример настроек с пояснениями приведен в листинге 3.9

3.3 Тестирование и отладка в Android разработке

В данном подразделе рассматривается тестирование и отладка Android-приложений.

3.3.1 Методы тестирования программного обеспечения

Методы тестирования ПО классифицируют по нескольким признакам [38]. По объекту тестирования:

```
android {
    ...
    signingConfigs {
        create("release") {
            val isRunningOnCI = System.getenv("CI") == "true"

            if (isRunningOnCI) {
               // Если сборка из CI, тополучаем значения из переменных окружения
                storeFile = file("../keystore.jks")
                storePassword = System.getenv("keystore_password")
                keyAlias = System.getenv("keystore_alias")
                keyPassword = System.getenv("keystore_alias_password")
            } else {
                // Иначе считываем значения из файла
                val secretProperties = Properties()
                val fis = FileInputStream(project.file("secret.properties"));
                secretProperties.load(fis)
                keyAlias = secretProperties.getProperty("keyAlias")
                        ?: error("'keyAlias' should be specified")
                keyPassword = secretProperties.getProperty("keyPassword")
                        ?: error("'keyPassword' should be specified")
                storeFile = file(secretProperties.getProperty("storeFile")
                        ?: error("'storeFile' should be specified"))
                storePassword = secretProperties.getProperty("storePassword")
                        ?: error("'storePassword' should be specified")
            }
        }
    }
    ...
}
```

Листинг 3.8 — Конфигурация подписи приложения

а) Функциональное тестирование — тестирование, основанное на проверке соответствия ПО функциональным требованиям;

б) Тестирование юзабилити — тестирование с целью определения понятности и простоты использования программного обеспечения;

в) Тестирование пользовательского интерфейса — проверка соответствия требованиям к пользовательскому интерфейсу;

г) Тестирование безопасности — проверка уязвимости ПО к различным типам атак;

```
before_deploy:
# Переименование релизного APK и перемещение в корневую директорию
- cp presentation/build/outputs/apk/release/presentation-release.apk ORIOKS.apk
# Реквизиты Travis CI и создание тэга билда
- git config --local user.name "Travis CI"
- git config --local user.email "builds@travis-ci.org"
- git tag "build-$TRAVIS_BUILD_NUMBER" -a -m "Build \#$TRAVIS_BUILD_NUMBER"
deploy:
  provider: releases # Билд будет загружен в GitHub Releases
  skip_cleanup: true # Не очищать результаты сборки перед публикацией
  overwrite: true # Разрешить перезапись билда
  api_key:
    secure: <зашифрованный_токен_доступа>
  file: ORIOKS.apk # Файл, который будет загружен в качестве билда
  on:
    repo: OsipXD/orioks-app # Репозиторий, в который загружать билд
  branch: master # Ветка, пуш в которую инициирует создание билда
```

Листинг 3.9 — Настройки автоматической публикации приложения из Travis CI

д) Тестирование локализации — проверка точности перевода элементов пользовательского интерфейса, правильности перевода сопроводительной документации.

е) Тестирование совместимости — тестирование с целью проверить правильность работы ПО в конкретной среде (аппаратная платформа, ОС, браузер и т.д.);

ж) Тестирование производительности — тестирование для определения производительности программного продукта;

1) Нагрузочное тестирование — определение скорости выполнения операций при разной нагрузке;

2) Стресс-тестирование — оценивает надёжность и устойчивость системы в условиях превышения пределов нормального функционирования;

и) Тестирование установки — предназначено для для проверки успешности установки, настройки, а так же обновления и удаления программного обеспечения.

По знанию внутреннего строения системы:

а) Тестирование белого ящика — тестирование на основе анализа внутренней структуры компонента или системы;

б) Тестирование черного ящика — тестирование без каких-либо знаний о внутренней структуре компонента или системы;

в) Тестирование серого ящика — комбинация тестирования белого и черного ящика.

По степени автоматизации:

а) Ручное тестирование — тестирование, производимое тестировщиком вручную, без использования средств автоматизации;

б) Автоматизированное тестирование — тестирование выполняется при помощи специализированных средств для автоматизации тестов без участия тестировщика в непосредственном тестировании;

в) Полуавтоматизированное тестирование — тестирование, выполняющееся частично вручную, частично автоматически.

По степени изолированности:

а) Юнит-тестирование — тестирование отдельных компонентов ПО;

б) Интеграционное тестирование — тестирование взаимодействия между связанными компонентами;

в) Системное тестирование — процесс тестирования системы в целом.

По времени проведения тестирования:

а) Альфа-тестирование — тестирование ранней версии ПО небольшой группой потенциальных пользователей или независимой группой разработчиков;

б) Бета тестирование — интенсивное использование почти готового продукта с целью выявления и устранения максимального числа ошибок;

в) Регрессионное тестирование — тестирование уже протестированной программы после модификации для того чтобы убедиться, что изменения работают так как было задумано;

г) Приёмочное тестирование — тестирование, проводимое при приёме программного обеспечения, с целью проверки соответствия программы критериям принятия;

д) Дымовое тестирование — минимальный набор тестов, направленных на проверку основного функционала программы.

По признаку позитивности сценария:

а) Позитивные тесты — проверка правильности работы программы при входных данных, соответствующих требованиям;

б) Негативные тесты — проверка, что программа работает правильно в случае неожиданных входных данных.

3.3.2 Разработка через тестирование

Разработка через тестирование (TDD) — это метод разработки ПО, при котором код пишется небольшими итерациями и предварительно для него пишется тест. Написание тестов перед написанием кода позволяет перейти от тестирования белого ящика к тестированию чёрного ящика и заставляет разработчика думать не “как сломать что-нибудь в этой программе”, а “как эта программа должна работать”.

При разработке через тестирования следует соблюдать три правила TDD.

а) Новый рабочий код пишется только после того, как будет написан юнит-тест, который не проходит.

б) Нужно писать ровно такой объём кода для юнит-теста, чтобы тест не проходил.

в) Нужно писать ровно такой объём рабочего кода, который необходим для прохождения юнит-теста, который в данный момент не проходит.

Эти три правила заставляют использовать коротки рабочий цикл продолжительностью примерно полминуты. Объём кода растёт постепенно и код уже покрыт тестами примерно на 85%. Высокая степень покрытия кода позволяет разработчику не бояться экспериментировать, добавлять новые возможности и изменять старые и не бояться при этом, что что-то сломается. Кроме того, разработчик уверен, что программа работает так как он задумал.

TDD поощряет хорошую архитектуру, так как для того, чтобы модуль было легко тестировать, он должен быть изолирован. Не получится ситуации, когда код сложно тестировать из-за большого количества зависимостей и статического кода, т.к. этого кода еще просто нет и разработчик будет писать код так, чтобы он проходил, написанные тесты.

При разработке через тестирование, полученную базу тестов можно использовать как документацию. Каждый тестовый случай — демонстрация использования возможностей модуля, а так как код пишется только при наличии теста для него, то такая документация есть для всех возможностей всех модулей.

Конечно, TDD — не панацея и не гарантирует, что разработчик получит все перечисленные выше преимущества, т.к. даже при таком подходе к разработке можно продолжать писать плохой код и плохие тесты. И, конечно, TDD, как и любой другой подход, обладает недостатками [39].

а) Написанные тесты нужно поддерживать. То есть при изменении логики работы модуля нужно сначала изменить тест под новую логику, и уже после этого вносить изменения в рабочий код.

б) На первых порах TDD замедляет разработку (но в долгосрочной перспективе разработка ускоряется).

в) Сложно начать работать по методологии TDD, особенно, если до этого есть многолетний опыт работы по-другому.

г) Необходимость создания тестов для всех случаев сбоев утомляет разработчика, но в долгосрочной перспективе окупается.

д) Приходится писать большое количество тестов (что одновременно является и плюсом).

3.3.3 Тестирование пользовательского интерфейса

Тестирование пользовательского интерфейса позволяет убедиться, что приложение удовлетворяет функциональным требованиям и является качественным. Один из подходов к тестированию UI — ручное тестирование. Когда тестировщик самостоятельно проходит все сценарии, которые необходимо протестировать. Однако, ручной подход трудоёмкий и подвержен ошибкам, поэтому гораздо лучше, если тестирование UI будет выполняться автоматически.

Для кода автоматизированных тестов пользовательского интерфейса в Android предусмотрен каталог `src/androidTest/java`. Android Gradle Plugin собирает тестовое приложения, опираясь на тестовый код и затем запускает тестовое приложение на устройстве, аналогичном целевому. Устройство может быть как реальным, так и виртуальным [40].

В тестовом коде можно использовать UI фреймворки для эмуляции взаимодействия пользователей с приложением. Это позволяет писать и воспроизводить тестовые сценарии для пользовательского интерфейса. Тестирование пользовательских взаимодействий в рамках приложения позволяет гарантировать, что пользователи не столкнутся с неожиданными результатами взаимодействии с приложением.

Фреймфорк тестирования Espresso, предоставляемый командой Android, представляет из себя API для написания тестов UI и имитации взаимодействия пользователя с приложением. Тесты Espresso могут выполняться на устройствах с версией Android 2.3.3 (API 10) или выше. Ключевым преимуществом Espresso пе-

ред другими фреймфорками является то, что он обеспечивает автоматическую синхронизацию пользовательских действий. То есть, фреймворк начинает взаимодействие с интерфейсом только тогда, когда обнаруживает, что главный поток (в котором производится отрисовка интерфейса) находится в режиме ожидания. Благодаря этой особенности нет необходимости добавлять ненадёжные решения (например, `Thread.sleep()`) для ожидания отображения результата [41].

3.3.4 Отладка в IntelliJ IDEA

Выводы по разделу

ЗАКЛЮЧЕНИЕ

Результатом выпускной квалификационнй работы является рабочее мобильное приложение для сопровождения учебного процесса студентов в системе ОРИОКС.

Мобильное приложение позволяет оповещать студентов с помощью их смартфонов о событиях учебного процесса, за счёт чего повышается удобство использования ОРИОКС и повышается оперативность получения оповещений студентами. Целевая аудитория приложения — студенты МИЭТ, которые заинтересованы в упрощении взаимодействия с ОРИОКС и оповещении о событиях учебного процесса.

В рамках ВКР были решены следующие задачи:

- проведен сравнительный анализ существующих программных решений;
- выбрана платформа МП;
- выбран язык и среда программирования;
- разработан алгоритм МП;
- разработана схема данных МП;
- разработан пользовательский интерфейс МП;
- проведены отладка и тестирование МП;
- разработан руководства оператора МП.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические указания по подготовке выпускной квалификационной работы по направлению подготовки бакалавров 09.03.04 «Программная инженерия» / Гагарина Л.Г., Коваленко Д.Г., Федотова Е.Л и др. ; Под ред. Б.В. Черникова. — МИЭТ, 2016. — 20 с.

2. ОРИОКС. ОРИОКС. — 2018. — Режим доступа: https://orioks.miet.ru/ (дата обращения: 20.04.2018).

3. Google. Приложения в Google Play — Расписание для МИЭТ. — 2018. — Режим доступа: https://play.google.com/store/apps/details?id=com.alex.miet.mobile (дата обращения: 25.04.2018).

4. Google. Приложения в Google Play — ОРИОКС Live. — 2018. — Режим доступа: https://play.google.com/store/apps/details?id=com.eva.oriokslive (дата обращения: 23.04.2018).

5. GitHub. dzmuh97/OpenOrioks — Уведомления из ОРИОКС и ОРОКС для студентов. — 2018. — Режим доступа: https://github.com/dzmuh97/OpenOrioks (дата обращения: 22.04.2018).

6. Apple. Develop - Apple Developer. — 2018. — Режим доступа: https://developer.apple.com/develop/ (дата обращения: 22.04.2018).

7. Android Developers. Application Fundamentals | Android Developers. — 2018. — Режим доступа: https://developer.android.com/guide/components/fundamentals (дата обращения: 23.04.2018).

8. Statista. Russia mobile OS market share 2012-2017 | Statistic. — 2018. — Режим доступа: https://www.statista.com/statistics/262174/market-share-held-by-mobile-operating-systems-in-russia/ (дата обращения: 23.04.2018).

9. В.М. Илюшечкин. Основы использования и проектирования баз данных : учебник для академического бакалавриата. Бакалавр. Академически курс. — ИД Юрайт, 2014. — 213 с.

10. Kotlin. Comparison to Java – Kotlin Programming Language. — 2018. — Режим доступа: https://kotlinlang.org/docs/reference/comparison-to-java.html (дата обращения: 27.04.2018).

11. Oracle. Oracle Technology Network for Java Developers | Oracle Technology Network | Oracle. — 2018. — Режим доступа: http://www.oracle.com/

technetwork/java/index.html (дата обращения: 27.04.2018).

12. Standard C++ Foundation. Standart C++. — 2018. — Режим доступа: https://isocpp.org/ (дата обращения: 27.04.2018).

13. Eclipse Foundation. Eclipse Oxygen | The Eclipse Foundation. — 2018. — Режим доступа: https://www.eclipse.org/oxygen/ (дата обращения: 29.04.2018).

14. JetBrains. Features – IntelliJ IDEA. — 2018. — Режим доступа: https://www.jetbrains.com/idea/features/ (дата обращения: 29.04.2018).

15. Android Developers. Meet Android Studio | Android Developers. — 2018. — Режим доступа: https://developer.android.com/studio/intro/ (дата обращения: 29.04.2018).

16. Android Developers. Distribution Dashboard | Android Developers. — 2018. — Режим доступа: https://developer.android.com/about/dashboards/ (дата обращения: 10.05.2018).

17. StatCounter. Mobile & Tablet Android Version Market Share Russian Federation | StatCounter Global Stats. — 2018. — Режим доступа: http://gs.statcounter.com/android-version-market-share/mobile-tablet/russian-federation (дата обращения: 13.05.2018).

18. Android Developers. Android Platform | Android Developers. — 2018. — Режим доступа: https://developer.android.com/about/ (дата обращения: 08.04.2018).

19. GitHub. Aerllo-Mobile/Moxy: Moxy is MVP library for Android. — 2018. — Режим доступа: https://github.com/Arello-Mobile/Moxy (дата обращения: 13.05.2018).

20. Dagger. Dagger. — 2018. — Режим доступа: https://google.github.io/dagger/ (дата обращения: 14.05.2018).

21. GitHub. stephanenicolas/toothpick: A scope tree based Dependency Injection (DI) library for Java. — 2018. — Режим доступа: https://github.com/stephanenicolas/toothpick (дата обращения: 13.05.2018).

22. Square. Retrofit. — 2018. — Режим доступа: https://square.github.io/retrofit/ (дата обращения: 12.05.2018).

23. Realm. Realm Database. — 2018. — Режим доступа: https://realm.io/products/realm-database (дата обращения: 14.05.2018).

24. Android Developers. Room Persistence Library | Android Developers. — 2018. — Режим доступа: https://developer.android.com/topic/libraries/architecture/room (дата обращения: 15.05.2018).

25. GitHub. AlexeyZatsepin/Android-ORM-benchmark: Performance comparison of Android ORM Frameworks. — 2017. — Режим доступа: https://github.com/AlexeyZatsepin/Android-ORM-benchmark (дата обращения: 13.05.2018).

26. ReactiveX. ReactiveX — Intro. — 2018. — Режим доступа: http://reactivex.io/intro.html (дата обращения: 15.05.2018).

27. GitHub. Kotlin/kotlinx.coroutines: Library support for Kotlin coroutines. — 2018. — Режим доступа: https://github.com/kotlin/kotlinx.coroutines (дата обращения: 15.05.2018).

28. GitHub. mannodermaus/android-junit5: Testing with JUnit 5 for Android. — 2018. — Режим доступа: https://github.com/mannodermaus/android-junit5 (дата обращения: 18.03.2018).

29. Spek. Spek User Guide. — 2018. — Режим доступа: http://spekframework.org/docs/latest/ (дата обращения: 15.05.2018).

30. RSpec. About RSpec. — 2016. — Режим доступа: http://rspec.info/about/ (дата обращения: 15.05.2018).

31. Uncle Bob. The Clean Architecture | 8th Light. — 2012. — Режим доступа: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html (дата обращения: 16.05.2018).

32. GitHub. AndroidArchitecture/AndroidArchitectureBook. — 2018. — Режим доступа: https://github.com/AndroidArchitecture/AndroidArchitectureBook (дата обращения: 16.03.2018).

33. Jeevuz. Заблуждения Clean Architecture / Блог компании MobileUp / Хабр. — 2017. — Режим доступа: https://habr.com/company/mobileup/blog/335382/ (дата обращения: 16.05.2018).

34. Karmakar Avijit. PBF(Package by Feature), no more PBL(Package by Layer). — 2017. — Режим доступа: https://medium.com/mindorks/pbf-package-by-feature-no-more-pbl-package-by-layer-50b8a9d54ae8 (дата обращения: 16.05.2018).

35. Gradle Inc. Gradle User Manual. — 2018. — Режим доступа: https://docs.gradle.org/ (дата обращения: 18.05.2018).

36. Android Developers. Publish Your App | Android Developers. — 2018. — Режим доступа: https://developer.android.com/studio/publish/ (дата обращения: 17.05.2018).

37. Travis CI. Travis CI User Documantation. — 2018. — Режим доступа: https://docs.travis-ci.com/ (дата обращения: 18.05.2018).

38. Wikipedia. Тестирование программного обеспечения — Википедия. — 2018. — Режим доступа: https://ru.wikipedia.org/wiki/Тестирование_программного_обеспечения#Классификации_видов_и_методов_тестирования (дата обращения: 20.05.2018).

39. Кент Бек. Экстремальное программирование. Разработка через тестирование. Библиотека программиста. — Питер, 2017. — 224 с.

40. Android Developers. Automating UI tests | Android Developers. — 2018. — Режим доступа: https://developer.android.com/training/testing/ui-testing/ (дата обращения: 21.05.2018).

41. Paul Blundell, Diego Torres Milano. Learning Android Application Testing. — Pakt, 2015. — 332 с.

ПРИЛОЖЕНИЕ А
ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

ПРИЛОЖЕНИЕ Б
РУКОВОДСТВО ОПЕРАТОРА

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

ПРИЛОЖЕНИЕ В
ТЕКСТ ПРОГРАММЫ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.