

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Мобильное приложение по контролю
состояния окружающей среды**

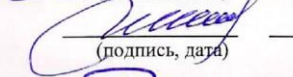
Студент


(подпись, дата)

Федосеев В.И.

(ФИО полностью)

Руководитель


(подпись, дата)

Шашков Б.Д.

(фамилия, инициалы)

Нормоконтролёр

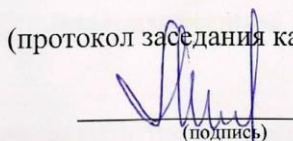

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

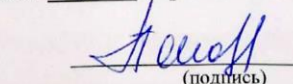

(подпись)

Макарычев П.П.

(фамилия, инициалы)

Работа защищена с отметкой 3 (протокол заседания ГЭК от 29.06.21 № 17)

Секретарь ГЭК


(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2021

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

П.П. Макарычев

«12» _____ 2021 г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Федосеев Вячеслав Игоревич гр.17ВИ1 факультета ФВТ

2. Тема работы Мобильное приложение по контролю состояния
окружающей среды Тема утверждена приказом ПГУ № _____ от
" ____ " _____ 20 ____ г.

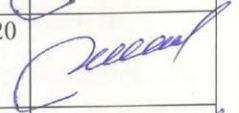
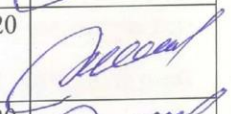
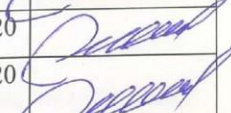
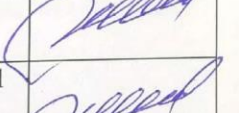
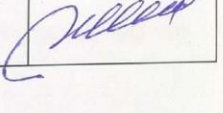
3. Руководитель работы к.т.н. доцент каф «МОиПЭВМ» Шашков Б.Д.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Мобильное приложение, предназначенное для контроля состояния
окружающей среды.

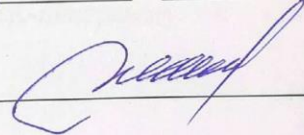
5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметной области, выбрать среду
разработки, определить интерфейс приложения, спроектировать приложение,
реализовать приложение, отладить и провести тестирование разработанного
мобильного приложения.

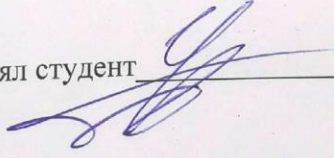
6. Календарный график выполнения работы

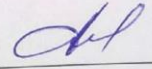
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10%	13.04.21-20.04.21	
2	Определение требований к системе	20%	20.04.20 – 04.05.20	
3	Проектирование приложения	40%	04.05.20 – 27.05.20	
4	Написание программы	10%	27.05.20 – 08.06.20	
5	Отладка и тестирование приложения	10%	08.06.20 – 16.06.20	
6	Оформление выпускной работы	10%	16.06.21-23.06.21	

Дата выдачи задания "28" декабря 2020 г.

Руководитель бакалаврской работы  Шашков Б.Д.

(подпись, дата)

Задание к исполнению принял студент  Федосеев В.И.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата) 28.06.2021

Реферат

Пояснительная записка содержит 92 страниц, 24 рисунка, 11 таблиц, 8 источников, 2 приложения.

Объект разработки – мобильное приложение по контролю состояния окружающей среды.

ANDROID STUDIO, JAVA SCRIPT, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ.

Целью разработки является разработка мобильного приложения по контролю состояния окружающей среды. Это достигается путём регистрации пользователем персональной учётной записи и дальнейшей публикацией зафиксированных происшествий в качестве новостного поста.

В результате проведенной работы создан проект, протестированы программные средства, удовлетворяющие поставленной задаче.

В пояснительной записке приведен анализ предметной области, обзор технологии разработки и отладки программы.

Графическая часть содержит ER-диаграмму и UML-диаграммы программы.

					ПГУ 09.03.02–8ВР171.25 ПЗ					
Изм.	Лист	№ докум.	Подпись	Дата	«Мобильное приложение по контролю состояния окружающей среды»	Лит.	Лист	Листов		
Разраб.		Федосеев В.И.								
Провер.		Шашков Б.Д.					3	92		
И. Контр.		Такташкин Д.В.		22.06.17		Группа 17ВИ1				
Утверд.		Макарычев П.П.		28.06						

Содержание

Введение.....	6
1 Анализ предметной области	8
1.1 Описание предметной области	8
1.2 Обзор существующих решений	8
1.3 Актуальность исследуемой предметной области	11
1.4 Выбор средств разработки	11
1.5 Архитектура приложения.....	13
1.6 Общая структура мобильного клиент-серверного приложения.....	14
2 Описание требований к системе	15
2.1 Требования к надёжности	15
2.2 Требования к пользователю ИС	15
2.3 Требования к информационной и программной совместимости.....	15
2.4 Требования к программной документации	16
3 Проектирование программных средств и структур данных разрабатываемой системы.....	17
3.1 Описание проектирования	17
3.2 Процесс разработки мобильного приложения	21
3.3 Структура мобильного приложения.....	22
4 Реализация проекта	24
4.1 Процесс разработки	24
4.1.2 Реализация Интерфейса приложения.....	25
4.2 Отладка и тестирование.....	38
4.3 Тестирование программного обеспечения	41
4.3.1 Структурное тестирование.....	41
4.3.2 Метод покрытия операторов.....	43
4.3.3 Метод покрытия условий	43

4.3.4 Метод покрытия маршрутов	44
4.3.5 Качественный анализ кода	45
4.3.6 SLOC-метрика	46
4.3.7 Метрика Холстеда.....	47
4.3.8 Цикломатическая метрика Мак-Кейба	48
5. Планирование и расчёт бюджета разрабатываемого мобильного приложения	52
5.1 Этапы разработки.....	52
5.2 Диаграмма Ганта	53
5.3 Проектирование приложения.....	55
5.4 Дизайн приложения	55
5.5 Разработка приложения	55
5.6 Тестирование и отладка.....	56
5.7 Техническая поддержка.....	56
5.8 Формирование бюджета разработки	56
5.9 Аренда помещения.....	58
5.10 Проектирование и дизайн.....	58
5.11 Этапы разработки.....	58
5.12 Тестирование и публикация.....	59
Заключение	60
Список используемых источников.....	61
Приложение А. Глоссарий	60
Приложение Б. Листинг перехода к блоку кода	62
Приложение В. Листинг программы.....	65

Введение

В нынешнее время практически любой человек полностью зависим от информации различного рода. Также непосредственное и самое главное влияние на жизнь человека оказывает среда, окружающая его на протяжении всего процесса жизнедеятельности. Вкупе со всем вышеперечисленным человек с помощью средств ознакомления с информацией и её обработкой в силах контролировать благосостояние, окружающей его среды.

Уже в течении последних нескольких лет мобильные технологии ощутимо для всех сфер развиваются, за счет непосредственного совершенствования беспроводной связи, различных мобильных платформ для разработок мобильных приложений различного назначения, появления более современных моделей мобильных устройств и расширения их функциональности, а также появления различных мобильных платформ с открытым исходным кодом.

Мобильное приложение подразумевает под собой программное обеспечение, которое разрабатывается с целью функционирования на различных мобильных устройствах, а именно таких как: смартфоны, планшеты, электронные книги, умные часы, умные браслеты и т.д.

Разработанное мобильное приложение для рассматриваемой предметной области несёт под собой цель – предоставить пользователю возможность фиксировать те или иные происшествия, происходящие в окружающей его среде и контролировать её состояние.

Цель и задачи подразумевают под собой анализирование и изучение современных приложений в области контроля состояния окружающей среды, сравнительный анализ аналогичных мобильных приложений, выделение их функциональных требований и определение функциональных возможностей, анализ актуальности разработки конкретного мобильного приложения по контролю состояния окружающей среды.

Посредством всего вышеперечисленного, имеется возможность сформулировать цели и задачи:

- анализ основных популярнейших приложений по контролю состояния окружающей среды
- анализ функциональных требований
- определение и выбор средств разработки
- анализ актуальности разработки приложения подобного рода
- конструктивное обоснование необходимости подобных приложений
- анализ конкурирующих решений
- анализ всех функциональных возможностей приложений
- анализ эффективности работы приложений
- анализ непосредственного влияния приложений на ту сферу деятельности, на которую они нацелены.

1 Анализ предметной области

1.1 Описание предметной области

Разрабатываемое мобильное приложение предназначено для контроля состояния окружающей среды, которое посредством своих функциональных возможностей предоставляет пользователю принимать участие в решении тех или иных вопросов окружающей среды относительно того места, в котором он находится или проживает.

Конкретная анализируемая и изучаемая предметная область подразумевает под собой совокупность мобильных приложений, нацеленных на контроль состояния окружающей среды.

Программное обеспечение данной предметной области – это процесс пользования мобильным приложением, которое представляет из себя аналог новостной ленты из какой-либо социальной сети. Пользователю предоставляется возможность завести себе учётную запись непосредственно внутри приложения, где он сможет с имеющегося аккаунта делать публикации и фиксировать те или иные происшествия. Такое вариант использования подобного приложения является актуальным, удобным и современным решением в данной конкретной предметной области.

Необходимо разработать мобильное клиент-серверное приложение, которое предоставлять пользователю возможность создания учётной записи, создание и редактирование сделанных им публикаций с сопутствующими функциональными возможностями.

1.2 Обзор существующих решений

Характеристика степени разработанности проблемы отечественными и зарубежными исследователями и разработчиками: Жизнь в большом городе накладывает ограничения, и одно из главных среди них – экология: качество воды и воздуха, как правило, оставляет желать лучшего. Хорошая новость в том, что появилось много ресурсов, которые помогут если не решить

проблему, то хотя бы минимизировать последствия – можно, например, не выходить на пробежку в те дни, когда воздух оставляет желать лучшего.

Приложение от Швейцарских разработчиков – AirVisual. Приятное глазу приложение, которое помогает отследить, как меняется качество воздуха в вашем городе в течение недели. В его основе – индекс качества воздуха: помимо конкретной цифры, программа использует цвет – зелёный, жёлтый, оранжевый или красный, в зависимости от обстановки. Кроме того, есть раздел с картой, в котором можно посмотреть ситуацию в разных городах – конечно, он неполный (например, почти нет данных об Африке), но может дать общую картину. В приложении также есть раздел с подробной информацией о том, что может влиять на загрязнение воздуха (правда, только на английском), и рейтинг городов по загрязнённости воздуха – может пригодиться, например, тем, кто планирует поездку.

Приложение от новозеландских разработчиков – UVLens. Это простое приложение на английском языке, которое подскажет индекс ультрафиолетового излучения: чтобы им пользоваться, знать язык совсем необязательно – всё и так понятно из интерфейса. Конечно, смотреть УФ-индекс можно и в стандартном приложении с погодой для смартфона, но UVLens даст дополнительные рекомендации: подскажет, как будет меняться уровень излучения в течение дня сегодня и завтра, а если вы введёте данные о себе, объяснит, через сколько минут вы обгорите и какая защита вам нужна – например, посоветует надеть шляпу и солнечные очки и по возможности прятаться в тени.

Приложение от разработчиков из США – Light Pollution Map. Световое загрязнение – проблема, уровень которой понимает, наверное, каждый житель большого (и не очень) города: просто вспомните, когда вам в последний раз удалось разглядеть ночью звёзды. Следить за обстановкой в вашем городе можно с помощью приложений для астрономов-любителей. Это приложение как раз из таких – в нём есть большая карта (раздел «The

Light Pollution Map»), с помощью которой можно оценить уровень загрязнения или, например, найти места, где можно наблюдать за звёздами. Помимо этого, в приложении есть и другие функции – например, фото дня NASA. Его авторы отмечают, что дело не только в звёздах на небе, но и в ресурсах, которые человечество тратит на освещение города; чтобы понять, что происходит там, где живёт человек, нужно несколько ночей отмечать, видны ли созвездия, на которые предлагает посмотреть приложение.

Приложение от итальянских разработчиков – Earthquake. Это приложение не касается экологической ситуации в прямом смысле, но помогает отследить очень важную проблему. Поначалу дизайн может показаться неочевидным, но всё просто: карта показывает информацию обо всех недавних землетрясениях в мире и границы тектонических плит, а в боковом меню можно открыть список всех недавних происшествий и отфильтровать их, например, по материку или конкретной стране. Можно посмотреть все землетрясения или только происшествия определённой магнитуды, например, пять баллов и выше. Можно настроить push-уведомления: они будут сообщать о происшествии поблизости или о землетрясениях определённой магнитуды, когда о них станет известно официально – можно, например, узнавать обо всех землетрясениях в России магнитудой выше трёх с половиной баллов.

Наиболее важное российское приложение в этом плане – Наш Санкт-Петербург. Приложение позволяет пользователю зафиксировать конкретную проблему на улице города, посредством чего привлекается внимание должностных лиц. Приложение также позволяет с помощью классификатора выбрать тему и категорию пользовательского сообщения, указать местоположение проблемы, используя карту и прикрепить фотографию. Имеется возможность описать детали и возможность отслеживания статуса сообщения. Пользователь может отслеживать

количество статистики сообщений по районам, вести мониторинг пяти важнейших причин.

1.3 Актуальность исследуемой предметной области

Актуальность мобильных приложений по контролю состояния окружающей среды обусловлена тем, что они достаточно хорошо популярны среди пользователей мобильных устройств, что является серьёзным аргументом в высокой значимости этих приложений. Мобильные приложения подобного рода необходимы как минимум потому, что пользователь имеет возможность посредством такого приложения с лёгкостью отслеживать и фиксировать то или иное состояние окружающей его среды. Также позволяет отслеживать качество воздуха, индекс ультрафиолетового излучения, световое загрязнение и также встречающиеся на улицах города проблемы, связанные с распространением мусора, заброшенных аварийных зданий, открытых люков и т.п. Пользователь может зафиксировать какую-либо замеченную им проблему на улицах или на окраине города, которая так или иначе вредит окружающей среде или является чрезвычайно опасной, тем самым обратив внимание жителей и конкретных должностных лиц, отвечающих за решение подобных проблем.

Также всё вышеописанное можно отнести к мобильным приложениям по контролю состояния окружающей среды, если рассматривать их как объект исследования, функциональные возможности которых, позволяют рационально использовать такие мобильные приложения в практической деятельности, которые тем самым несут собой пользу окружающей среде и благоустройству города.

1.4 Выбор средств разработки

В ходе анализа предметной области были проанализированы различные средства, технологии и языки программирования.

Таким образом был выбран ряд программных средств, использованных в данной работе:

- среда разработки Android Studio для работы с платформой Android;
- объектно-ориентированный язык программирования Java.

Выбор средств разработки исходил из функциональных возможностей, высокой производительности и бесплатного использования.

Чтобы выбрать средства разработки, вначале необходимо определиться с целью его использования. Если нужно создать простейшее мобильное приложение, не требующее каких-либо сложных манипуляций как со стороны разработчика, так и с точки зрения пользователя, то для этого идеально подходят онлайн - конструкторы мобильных приложений.

Если разработка приложения имеет коммерческий характер и требования функциональных возможностей приложения таковы, что они должны предоставлять пользователю обширный и детальный спектр возможностей в плане пользования приложением, то для таких целей подойдут такие средства разработки как: различные платформы, нацеленные исключительно на разработку мобильных приложений под операционные системы Android, IOS, BlackBerry, Windows Mobile; также подойдут среды разработки Microsoft Visual Studio, Android Studio, свободная интегрированная среда разработки модульных кроссплатформенных приложений Eclipse, интегрированная среда разработки программного обеспечения IntelliJ IDEA.

Следующим критерием для выбора средств разработки – это аппаратные требования. Для вышеописанных средств разработки могут подойти любые офисные и начальные решения персональных настольных компьютеров, особенно с учётом того, что рынок подобных средств разработки может предложить мощные средства решения ресурсоёмких задач.

Из рассмотренных средств разработки было выбрано такое средство, как Android Studio, так как оно обладает всеми необходимыми функциональными возможностями для обеспечения и работоспособности мобильного приложения, поддерживает визуальную технологию создания объектов и интерфейса приложения, используя языки программирования Java, Java Script, C#.

1.5 Архитектура приложения

Прежде, чем начать разработку мобильного приложения, необходимо изучить архитектуру создания ПО.

Проектирование и разработка мобильного приложения по контролю состояния окружающей среды осуществляется на основе архитектуры мобильного клиент – серверного приложения. Описание архитектуры клиент-серверного приложения изображена на рисунке 1.

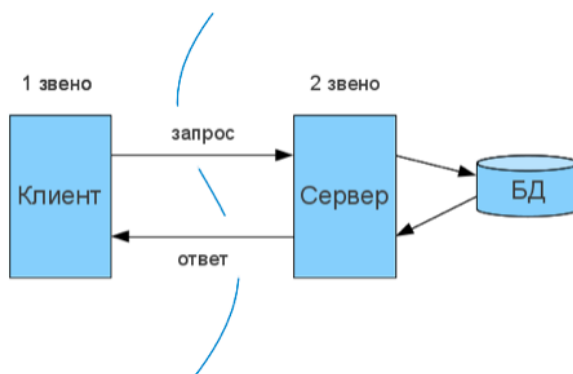


Рисунок 1 – Архитектура мобильного клиент-серверного приложения

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций. Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия).

В разработке конкретного рассматриваемого мобильного приложения применяется двухзвенная архитектура. Она используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса. Описание двухзвенной архитектуры представлено на рисунке 2.

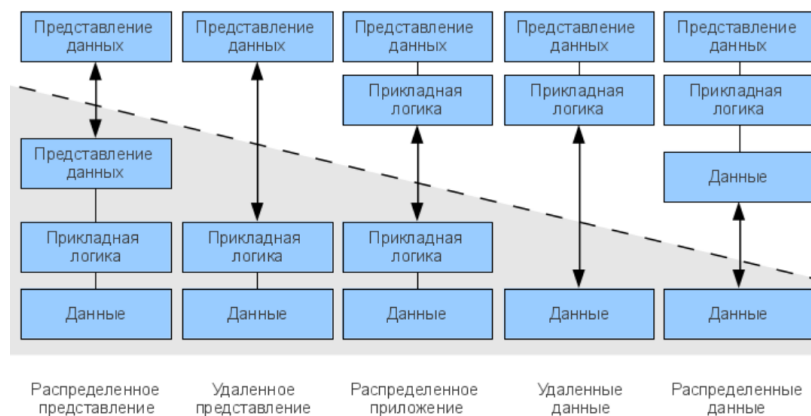


Рисунок 2 – Двухзвенная архитектура

Двухзвенная архитектура – распределение трех базовых компонентов между двумя узлами (клиентом и сервером).

1.6 Общая структура мобильного клиент-серверного приложения

На самом примитивном уровне абстракции приложение, ориентированное на работу с сервером, состоит из следующих архитектурных слоев:

- ядро приложения, которое включает в себя компоненты системы, не доступные для взаимодействия с пользователем;
- графический пользователь интерфейс;
- компоненты повторного использования: библиотеки, визуальные компоненты и другое;
- ресурсы приложения: графические файлы, звуки, необходимые сопутствующие файлы.

2 Описание требований к системе

2.1 Требования к надёжности

Основными требованиями к надёжности системы являются: бесперебойное функционирование мобильного приложения и возможность комфортного использования для пользователя с учётом корректного функционирования всех имеющихся функциональных возможностей.

В мобильном приложении должна быть реализована возможность регистрации персональной учётной записи посредством электронной почты и пароля, ввод данных от аккаунта при запуске приложения. Должна иметься возможность навигации по основным разделам приложения: новостной ленте, персональной учётной записи и по разделу контактов. В приложении можно осуществлять редактирование персональной учётной записи, публиковать новостные посты. Также реализована возможность осуществлять навигацию по вкладкам непосредственной внутри раздела новостной ленты.

2.2 Требования к пользователю ИС

Разработанным мобильным приложением может пользоваться без какого-либо дискомфорта даже не уверенный пользователь, так как у приложения просто, понятный и интуитивный интерфейс, простота в ознакомлении с информацией, в вводе данных и их использовании. Приложение предназначено как для пользователя, так и для модератора приложения с целью отслеживании публикации новостных постов и их модерации.

2.3 Требования к информационной и программной совместимости

Мобильное приложение для контроля состояния окружающей среды должно быть установлено непосредственно на мобильное устройство (смартфон, планшет).

Мобильное устройство пользователя должно удовлетворять следующим требованиям:

- операционная система Android не старше версии 4.4;
- возможность мобильного устройства отслеживать геоданные;
- постоянное подключение к сети интернет IPv4.

2.4 Требования к программной документации

В состав пояснительной записки дипломного проекта должны быть включены следующие разделы и документы:

- техническое задание;
- описание программы;
- программа и методика испытаний;
- описание применения.

3 Проектирование программных средств и структур данных разрабатываемой системы

3.1 Описание проектирования

В ходе реализации мобильного Android-приложения, изначально была выбрана архитектура, на основе которой будет спроектировано приложение. В разработке конкретного рассматриваемого мобильного приложения применяется двухзвенная архитектура. Грамотно выбранная архитектура позволит в дальнейшем без каких-либо проблем вносить изменения, добавляя и улучшая функциональные возможности приложения.

Также приложение должно быть комфортным в плане опыта пользования, пользовательский интерфейс должен быть интуитивно понятным, позволять осуществлять какие-либо манипуляции с постами новостной ленты приложения без каких-либо трудностей.

Возможные варианты взаимодействия пользователя с мобильным приложением представлены на UML – диаграмме вариантов использования на рисунке 3.

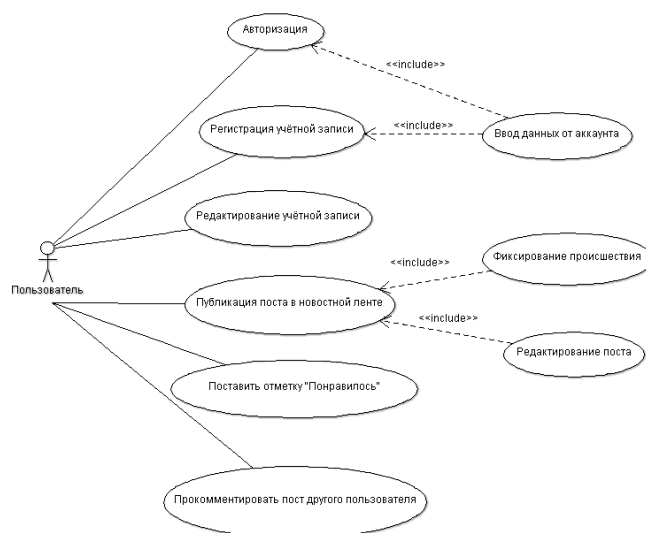


Рисунок 3 – Диаграмма вариантов использования

Диаграмма вариантов использования – диаграмма, на которой отражены отношения, существующие между актёрами

вариантами использования системы. Краткое описание вариантов использования описано в таблице 1.

Таблица 1 - Краткое описание вариантов использования

Действующее лицо	Цель	Краткое описание
Пользователь	Авторизация	Пользователь авторизуется в мобильном приложении
Пользователь	Регистрация учётной записи	Пользователь осуществляет регистрацию своей учётной записи в мобильном приложении
Пользователь	Редактирование учётной записи	Пользователь осуществляет редактирование своей учётной записи в мобильном приложении
Пользователь	Публикация поста в новостной ленте	Пользователь осуществляет публикацию поста в новостной ленте
Пользователь	Поставить отметку «Понравилось»	Пользователь может по своему желанию поставить отметку «Понравилось» на пост другого пользователя
Пользователь	Прокомментировать пост другого пользователя	Пользователь может по своему желанию прокомментировать пост другого пользователя
Пользователь	Ввод данных от аккаунта	При авторизации пользователю необходимо ввести данные от своей учётной записи
Пользователь	Фиксирование происшествия	Пользователь посредством своего мобильного устройства может зафиксировать то или иное происшествие и тем сама сделать публикацию в приложении
Пользователь	Редактирование поста	Пользователь может отредактировать, сделанный им ранее пост в мобильном приложении

Действующие лица диаграммы вариантов использования описаны в таблице 2.

Таблица 2 - Действующие лица

Название	Профиль, подготовка и навыки
Пользователь	Посредством мобильного приложения имеет возможность за счёт его функциональных возможностей, зарегистрироваться в нём и осуществлять публикации постов внутри приложения, тем самым фиксируя те или иные ситуации и происшествия.

Действующее лицо/Цель диаграммы вариантов использования описано в таблице 3.

Таблица 3 - Действующее лицо/Цель

Действующее лицо	Цель уровня задачи	Приоритет
Пользователь	Регистрация учётной записи	1
	Авторизация	1
	Редактирование учётной записи	2
	Публикация поста в новостной ленте	1
	Поставить отметку «Понравилось»	3
	Прокомментировать пост другого пользователя	3
	Ввод данных от аккаунта	1
	Фиксирование происшествия	1
	Заполнение учетных данных клиента	2
	Редактирование поста	2

Схема алгоритма создания и публикации новостного поста в мобильном приложении описана на рисунке 4.

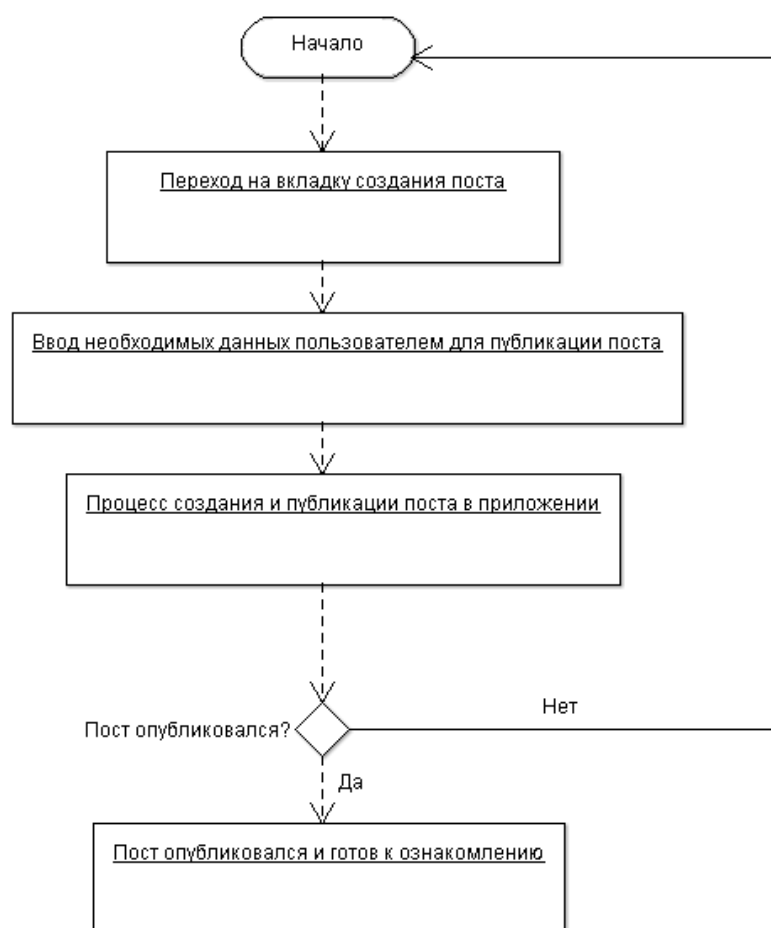


Рисунок 4 – Диаграмма деятельности

Обрабатываемые приложением результаты изображены с помощью UML – диаграммы состояний на рисунке 5.

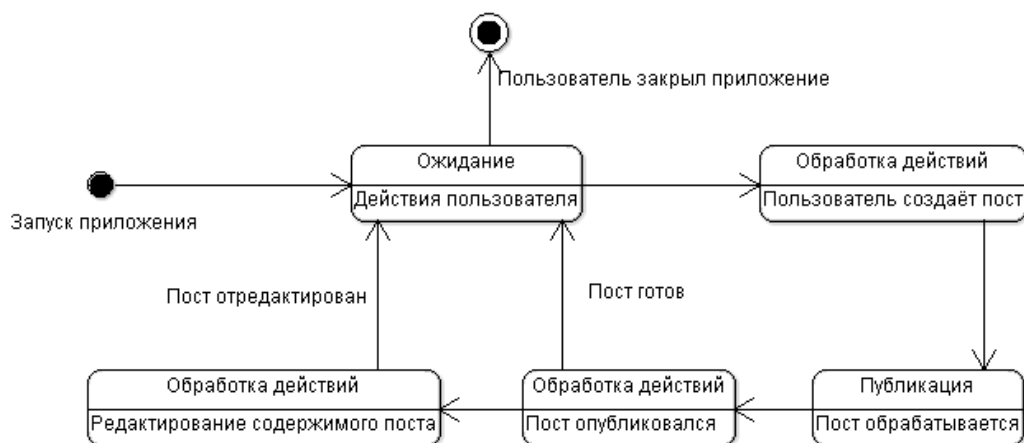


Рисунок 5 – Диаграмма состояний

Диаграмма состояний показывает, как объект переходит из одного состояния в другое.

3.2 Процесс разработки мобильного приложения

Каким бы ни было мобильное приложение, вначале процесс создания всегда включает оценку, аналитику, дизайн, разработку, тестирование, отладку, релиз и возможную поддержку после релиза. Все вышеперечисленные аспекты также имели отношение к разработанному мобильному приложению. В итоге на выходе имеются:

- описание базовых функций мобильного приложения;
- выбор методологии: гибкая или каскадная;
- выбор платформы операционной системы: Android, iOS или кросс-платформа.

В данном разрабатываемом приложении использовалась гибкая методология – приложение разрабатывается постепенно итерациями, каждая из которых объединяет в себе все рассмотренные и перечисленные стадии разработки мобильного приложения.

Был пройден процесс оценки, при котором стало известно представление о трудозатратах, необходимых на разработку мобильного приложения.

Важным этапом был анализ всего процесса разработки мобильного приложения. Также имел место быть анализ бизнес-процесса относительно функциональных возможностей приложения. После чего имеется:

- спецификация функциональных требований;
- план проекта разработки;
- определение графического интерфейса.

Одним из важнейших этапов является разработка дизайна мобильного приложения. Разработка происходила посредством графического редактора, интегрированного в среду разработки Android Studio. По итогу дизайн приложения полностью отвечал требованиям к приложению.

В итоге есть:

- полностью готовый интуитивно понятный интерфейс;

- карта экранов приложения;
- дизайн приложения.

Следующим, самым важным и необходимым этапом является разработка рассматриваемого приложения, процесс которой происходил в среде разработки Android Studio. Посредством написания кода приложения, была необходимость реализовать поведение запланированного приложения и не допустить ошибок в его логике. В конце реализации имеется:

- версия приложения, готовая к тестированию;
- корректировка дизайна, посредством кода.

Завершающим этим во всём процессе разработки мобильного приложения является тестирование и отладка. В следствии тестирования были выявлены ошибки как в коде самого приложения, так и в других аспектах, которые нарушали работу приложения или делали процесс пользования не комфортным и проблематичным. После прохождения тестирования имеется:

- сведённые к минимуму баги и ошибки или их полное устранение;
- предрелизная версия мобильного приложения.

3.3 Структура мобильного приложения

Мобильные приложения, разрабатываемые под ОС Android, могут быть сложными или простыми, но структура по своей сути всегда будет одинаковой. Есть необходимые и обязательные элементы, но также есть и опциональные – они используются по мере необходимости.

Приложение состоит из определённого количества основных компонентов: манифест приложения, набор различных ресурсов и исходный код мобильного приложения.

Обязательные и возможные составляющие структуры мобильного приложения: `gen` – группа файлов, которая сгенерирована самой Java. В них находится важный и необходимый файл как `R.java`; `AndroidManifest.xml` –

файл манифеста предоставляет системе основную информацию о программе. Абсолютно каждое приложение должно и необходимо иметь свой файл манифеста; `src` – это каталог, который содержит в себе группу файлов, представляющих из себя исходный код мобильного приложения, также необходим; `assets` – это произвольное собрание каталогов и файлов, не имеет какой-либо строгой необходимости; `res` – это каталог, который содержит необходимые ресурсы приложения. В данном каталоге могут находиться такие подпапки, как `drawable`, `anim`, `layout`, `menu`, `values`, `xml` и `raw`.

4 Реализация проекта

4.1 Процесс разработки

Необходимо спроектировать и разработать мобильное приложение по контролю состояния окружающей среды. Анализируя данную предметную область, можно составить описание бизнес - процессов информационной системы.

В таблице 4 представлено описание всех необходимых бизнес-процессов в разрабатываемом мобильном приложении.

Таблица 4 – Описание бизнес - процессов

п/п	Бизнес-процесс	Исполнитель	Входные данные		Выходные данные		Ограничения
			Поставщик	Содержание	Потребитель	Содержание	
1	Регистрация аккаунта в приложении	Пользователь	Приложение	Информация о персональных данных пользователя	Пользователь	Информация, свидетельствующая о успешной регистрации	Отсутствуют
2	Публикация новостного поста	Пользователь	Приложение	Информация о происшествии в новостном посте	Пользователь	Информация, свидетельствующая о успешной публикации поста	Не допустима публикация запрещённого контента
3	Модерация публикуемых новостных постов	Модератор	Приложение	Корректировка и проверка информации	Пользователь	Информация, свидетельствующая о успешной проверке поста	Отсутствуют

В таблице 5 представлено описание всех необходимых сущностей разрабатываемого мобильного приложения.

Таблица 5 - Описание типов сущностей

№	Наименование	Описание
1	Пользователь	id, Пользователя, id Геолокации, фамилия, имя, почта, пароль
2	Новостная лента	id, Новостной ленты
3	Карта	id Карты, Карта Пензы
4	Геолокация	id Геолокации, id Карты, координаты, районы
5	Новостные посты	id Поста, id Геолокации, id, Новостной ленты, дата, кол-во понравилось, кол-во комментариев

В таблице 6 также имеется описание всех необходимых типов связей между типами сущностей предметной области разрабатываемого мобильного приложения.

Таблица 6 - Описание типов связей между типами сущностей предметной области

№	Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность	Ограничения
1	Пользователь	Геолокация	Пользователь входит в местоположение	1:M	Обязательно	Отсутствуют
2	Новостные посты	Лента	Новостные посты содержатся в новостной ленте	1:M	Обязательно	Недопустимо повторение одного и того же новостного поста
3	Новостные посты	Геолокация	Новостные посты входят в местоположение	1:M	Обязательно	Отсутствуют
4	Геолокация	Карта	Геолокация содержится в разделе карты	1:M	Обязательно	Отсутствуют

4.1.2 Реализация Интерфейса приложения

Для работы с мобильным приложением были реализованы следующие экраны интерфейса:

- экран регистрации пользователей;

- экран авторизации пользователей;
- интерфейс главного экрана, в котором происходит взаимодействие с главными разделами;
- раздел новостной ленты;
- раздел аккаунта пользователя;
- раздел с контактами.

Экран регистрации пользователя.

Пользователю предоставляется возможность ввести свои имя, фамилию, электронную почту и пароль (рисунок 6). Все эти данные необходимы для регистрации персональной учётной записи. Также можно ознакомиться с политикой конфиденциальности.

Мой аккаунт

Имя

Имя

Фамилия

Фамилия

Email

Email

Пароль

Пароль

☐ Я прочитал и согласен с политикой конфиденциальности. *

Сохранить

Уже есть аккаунт? Войти!

Рисунок 6 – Экран регистрации учётной записи

Если пользователь уже имеет персональную учётную запись, то ему предлагается в неё войти.

Экрана авторизации пользователей.

Существует две ситуации, исходя из которых пользователь может попасть на этот экран. Первая – это когда пользователь заходит в приложение впервые: для работы в данном программном продукте ему необходимо авторизоваться. И вторая ситуация – когда во время работы необходимо сменить пользователя.

Стоит отметить, что при вводе пользователем пароля, символы закрашиваются точками, что необходимо для безопасности, как показано на рисунке 7.

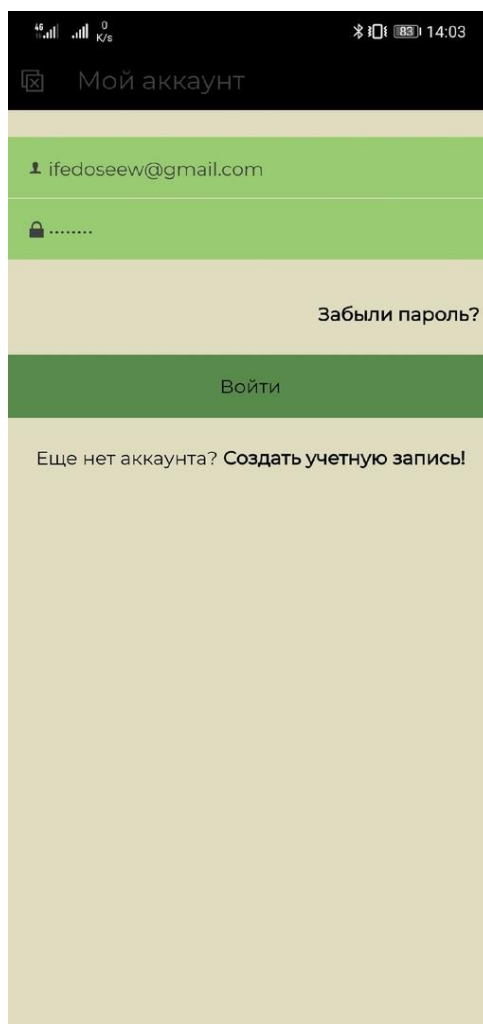


Рисунок 7 – Экран авторизации пользователя

Пользователю также предоставлена возможность восстановить пароль в случае его утери. Если ещё нет аккаунта, то с страницы входа в приложения пользователь может создать себе учётную запись.

Интерфейс главного экрана.

На главном экране пользователь может выбрать в какой раздел он может попасть: новостная лента, аккаунт пользователя и контакты. Результат представлен на рисунке 8.

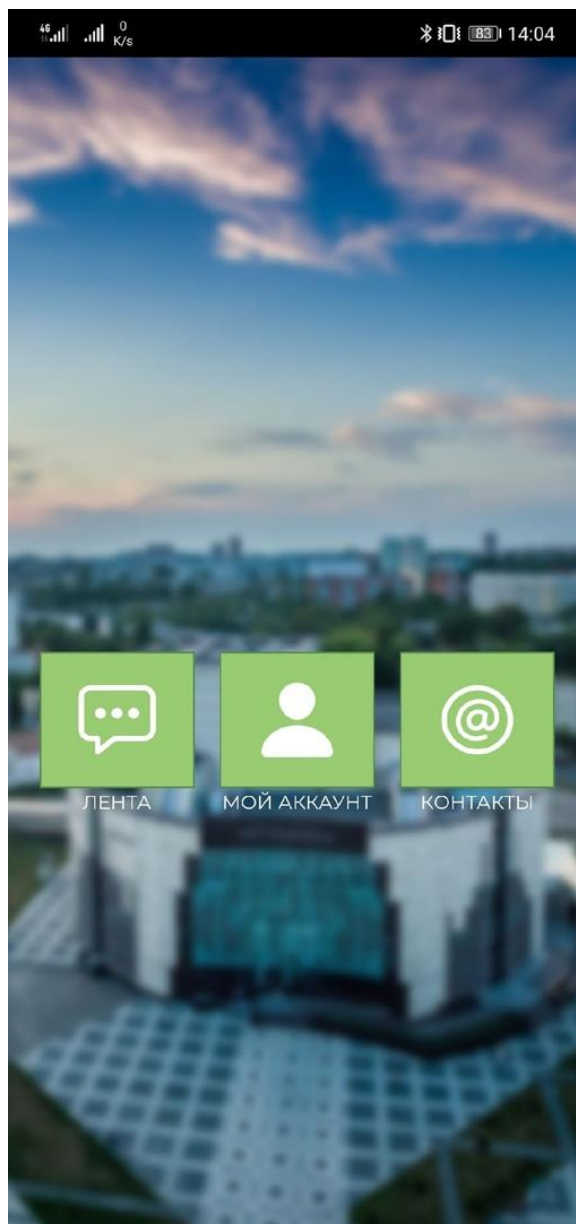


Рисунок 8 – Главный экран приложения

На главном экране предоставлены 3 главных раздела мобильного приложения.

Экран новостной ленты.

В разделе новостной ленты осуществляется навигация посредством прокрутки. Отображаются посты, опубликованные пользователями. Любой зарегистрированный пользователь имеет возможность поставить отметку «Понравилось» и прокомментировать опубликованный пост другого пользователя. Результат представлен на рисунках 9, 10.



Рисунок 9 – Раздел новостной ленты с опубликованными постами

При нажатии на кнопку параметров появляется контекстное меню, в котором представлены дополнительные функции для взаимодействия с новостным постом.

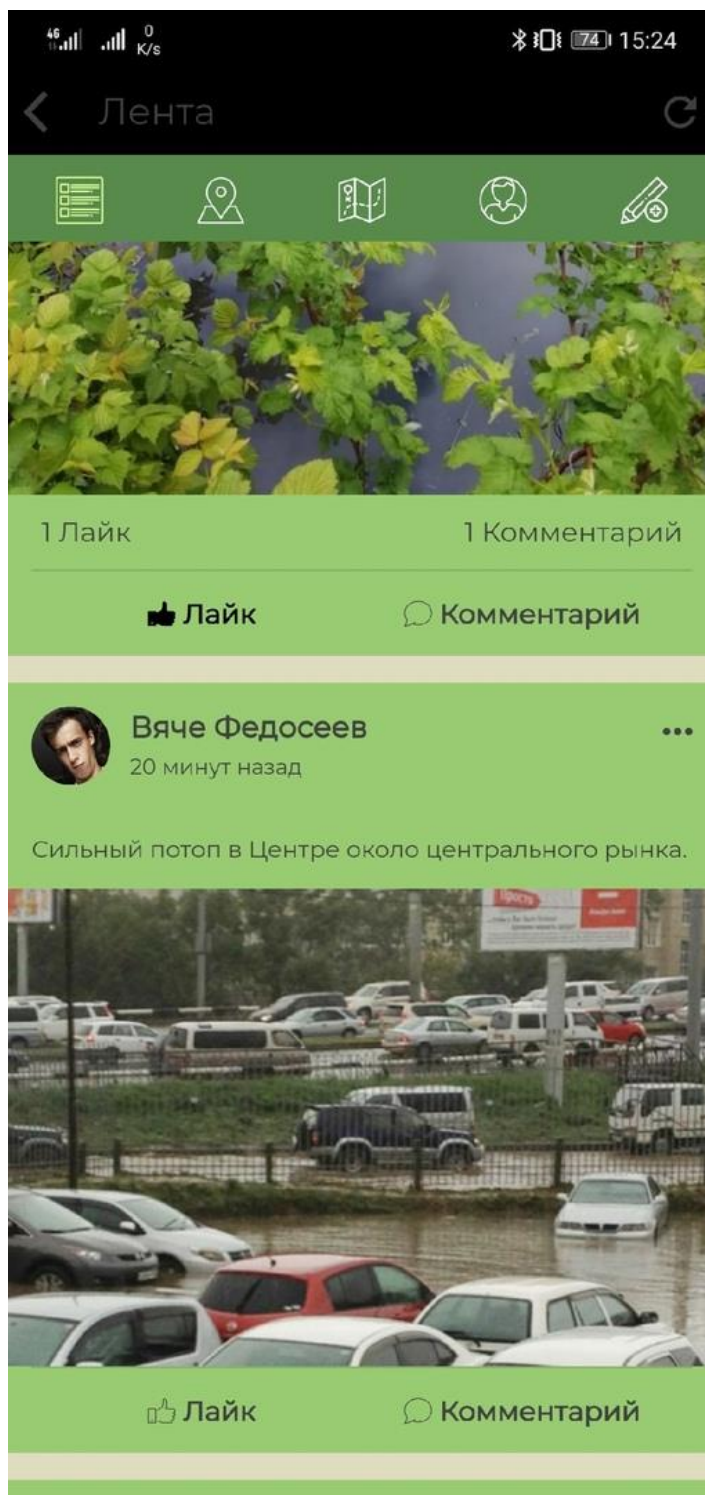


Рисунок 10 – Раздел новостной ленты с опубликованными постами

Новостная лента представляет из себя список всех опубликованных новостных постов в хронологическом порядке. На рисунке 11 представлена страница с комментариями поста.

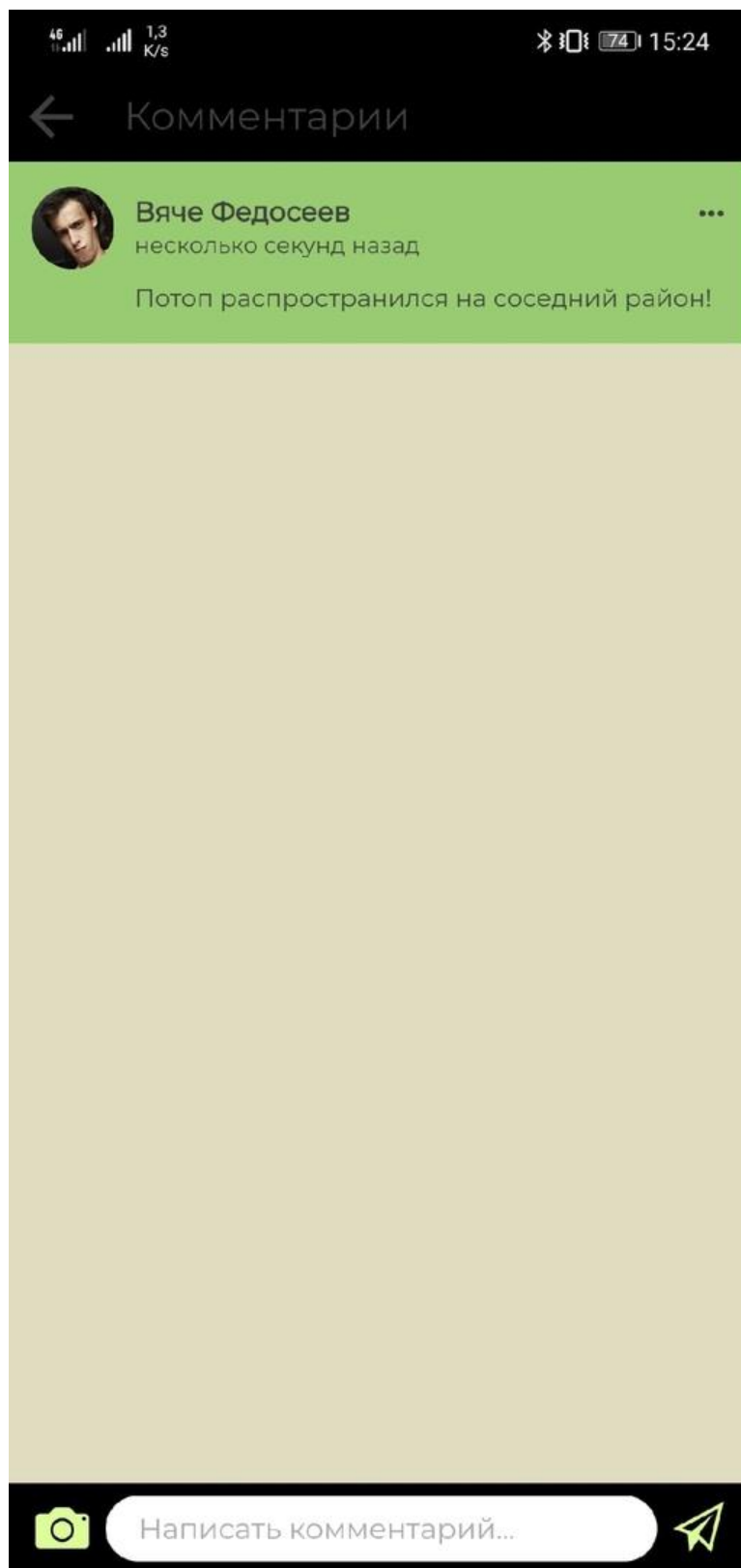


Рисунок 11 – Комментарии под опубликованным постом

Любой пользователь имеет возможность прокомментировать любой опубликованный новостной пост. Также в комментарии имеется

возможность прикрепить любое изображение. На рисунке 12 изображён раздел новостной ленты с опубликованными постами.



Рисунок 12 – Раздел новостной ленты с опубликованными постами

Также абсолютно любой пользователь, заметивший не допустимый или запрещённый контент, может посредством функции обратной связи пожаловаться на опубликованный пост. Результат представлен на рисунке 13.

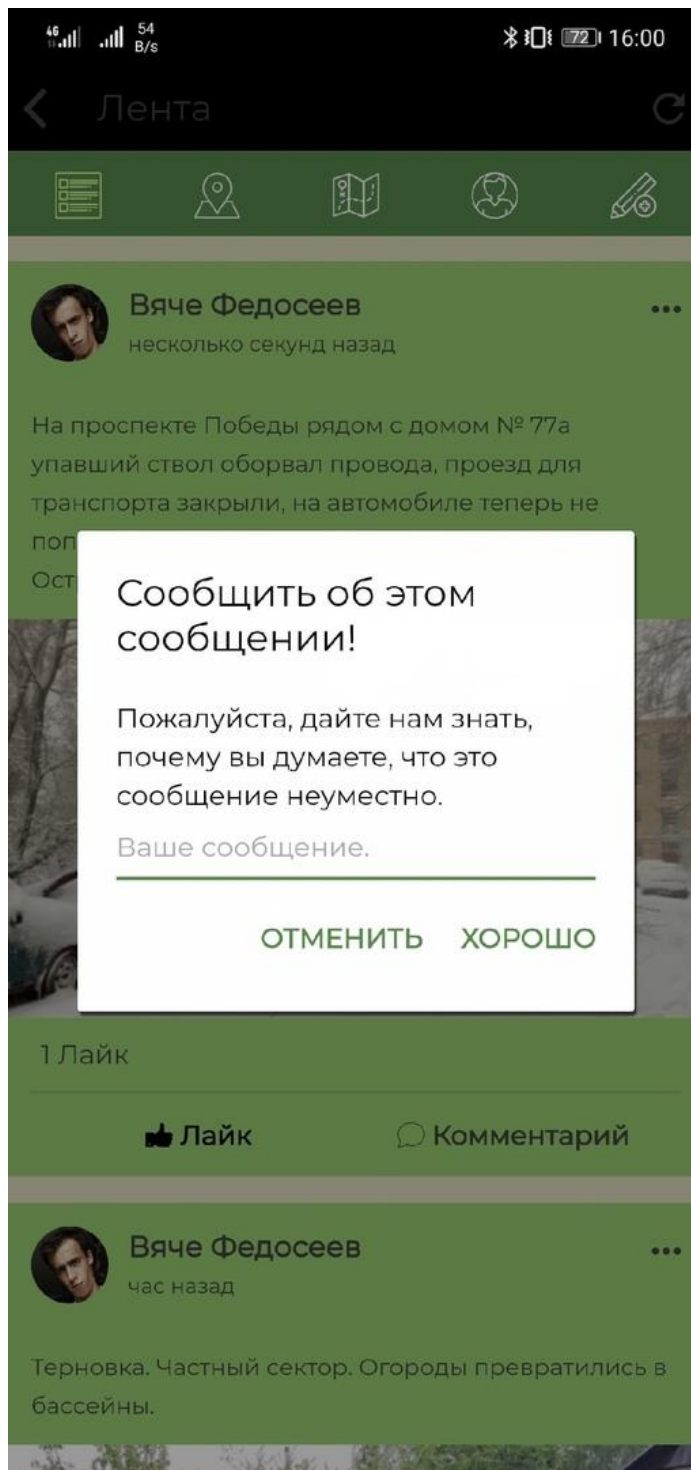


Рисунок 13 – Раздел новостной ленты. Функция жалобы на опубликованный пост

При содержании в новостном посте недопустимого контента, любой пользователь имеет возможность сообщить об этом разработчику приложения, чтобы решить дальнейшую судьбу новостного поста.

Экран с вкладкой «Карта».

На данном экране отображается карта, на которой отмечены все опубликованные рядом новостные посты другими пользователями в пределах 15 километров. Результат представлен на рисунках 14, 15 и 16.

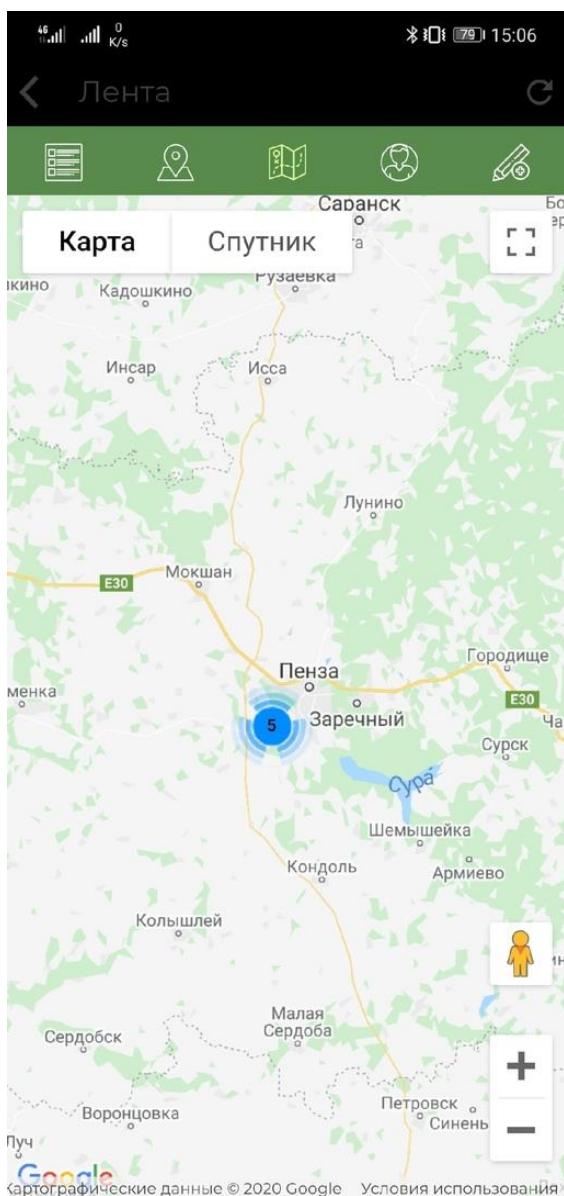


Рисунок 14 – Вкладка «Карты», на которой отображены опубликованные рядом постов

Посты определяются на карте посредством данных о геолокации. Пользователь указывает местоположение зафиксированного происшествия, информация о котором потом будет в новостном посте.

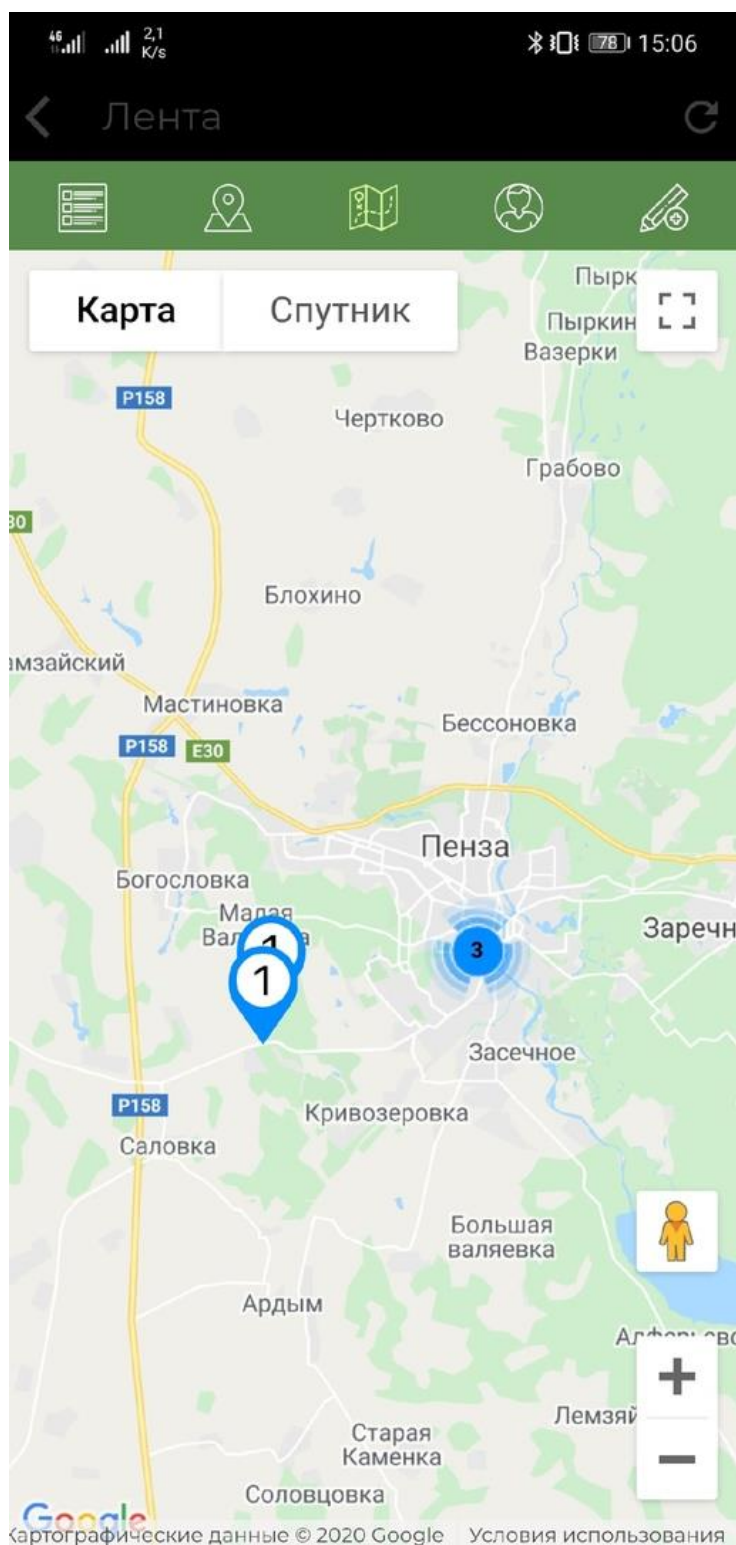


Рисунок 15 – Вкладка «Карты», на которой изображено отображение опубликованных рядом постов в пределах 15 километров

Мобильное приложение позволяет отследить и найти на карте опубликованные новостные посты в пределах 15 километров.

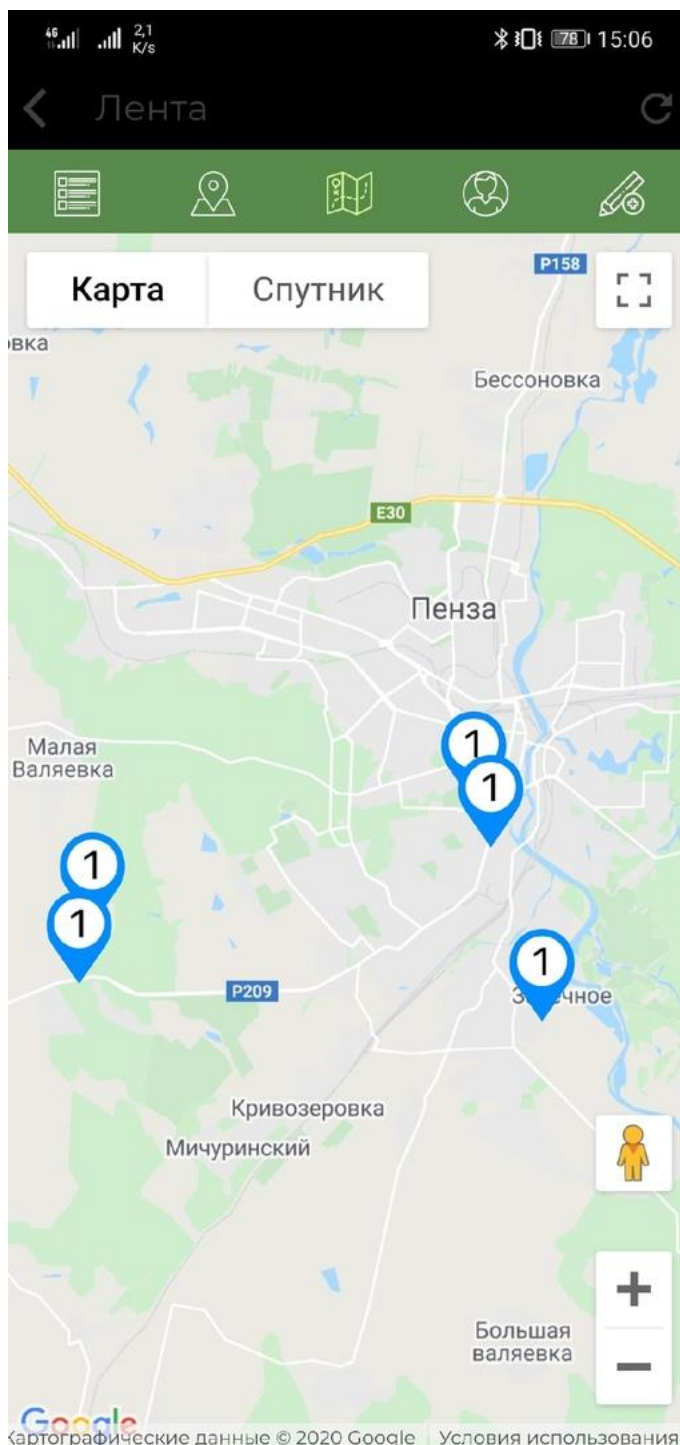


Рисунок 16 – Вкладка «Карты». Отображение опубликованных рядом постов в пределах 15 километров

На скриншоте отображены все опубликованные новостные посты с геолокации в пределах 15 километров. Если несколько новостных постов

обозначены в одном и том же месте, то при приближении изображения на карте они отображаются обособленно друг от друга.

Экран с вкладкой «Мой профиль».

На вкладке с профилем персональной учётной записи пользователя можно отредактировать данные аккаунта, а именно: изображение, имя и фамилию. Также имеется возможность заблокировать страничку другого пользователя, чтобы не видеть его публикации. В нижней части вкладки отображаются новостные посты, опубликованные пользователем данного профиля. Результат представлен на рисунках 17 и 18.

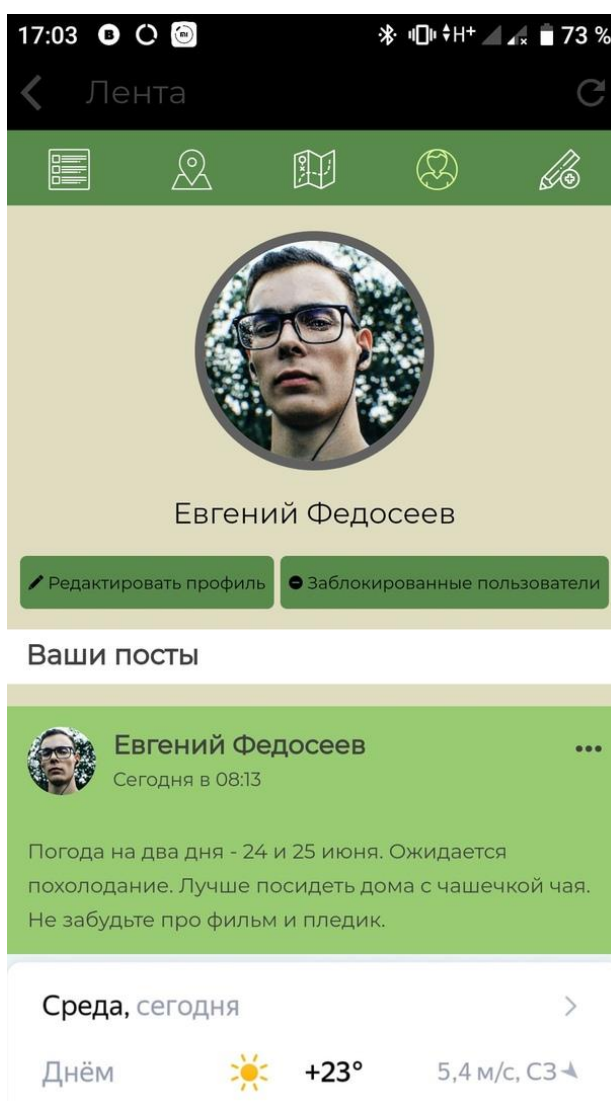


Рисунок 17 - Вкладка профиля учётной записи пользователя

Любой зарегистрированный пользователь имеет возможность ознакомиться с профилем любого другого зарегистрированного пользователя.

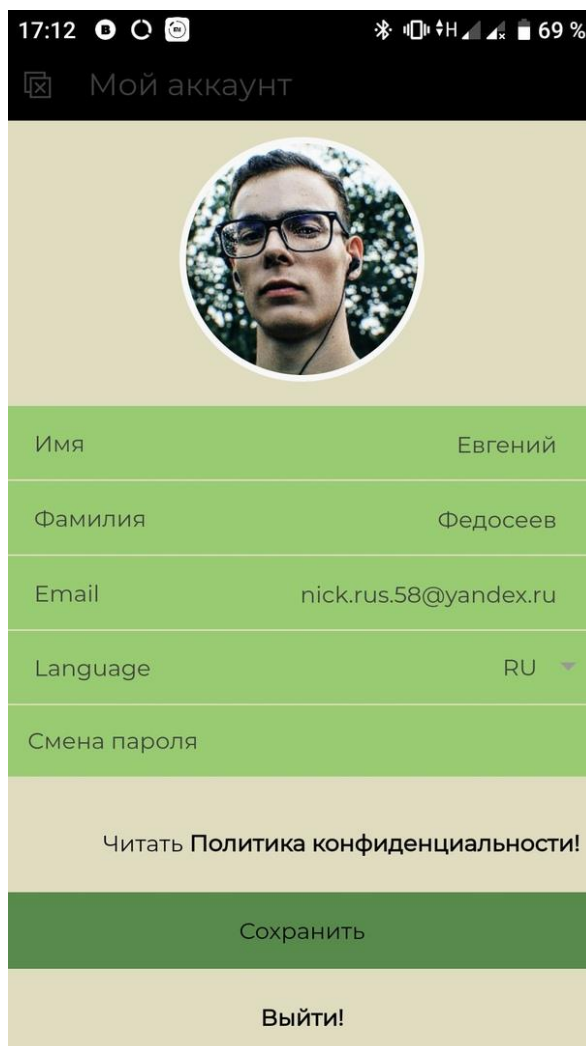


Рисунок 18 – Вкладка профиля учётной записи пользователя. Раздел редактирования данных

При редактировании своего профиля имеется возможность отредактировать любую характеристику профиля пользователя.

4.2 Отладка и тестирование

Программа написана и отлажена с использованием среды разработки и программирования Android Studio.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было

выявлено и исправлено множество проблем, связанных с синтаксисом конструируемых в программе элементов кода, с функциональными возможностями самого приложения и его интерфейса.

Произведём тестирование раздела «Новостная лента», где будет протестирована возможность публикации новостного поста. Перейдём на вкладку создания поста, опишем зафиксированное происшествие в поле описания к посту, по возможности зададим местоположение и прикрепим к посту изображение самого происшествия. Результат теста представлен на рисунках 19 и 20.



Рисунок 19 – Текст публикации новостного поста

На странице приложения имеется возможность приложить несколько изображений из галереи мобильного устройства.



Рисунок 20 – Тест публикации новостного поста

Только что опубликованный новостной пост содержит в себе всю необходимую информацию, указанную при его создании.

4.3 Тестирование программного обеспечения

Тестирование является способом определения качества программного продукта. Искусство тестирования находится в прямой зависимости от умения программиста разрабатывать эффективные тесты.

Суть тестирования программного продукта заключается не в том, чтобы показать, что она работает, а скорее наоборот – тестирование начинается с предположения, что в программном продукте есть ошибки, а затем уже обнаруживаются их максимально возможное число. Таким образом, тестирование – это процесс исполнения программы с целью обнаружения ошибок.

В пояснительной записке данной выпускной квалификационной работы рассматривается структурное тестирование и проводится анализ качества программного кода путём определения различных метрик.

4.3.1 Структурное тестирование

Структурное тестирование (тестирование по принципу «белого ящика») характеризуется степенью, в которой тесты покрывают логику (исходный текст) программы. Данная методология тестирования предполагает выполнение каждого пути в программе.

Структурное тестирование основывается на детальном изучении логики программы и подборе тестов, обеспечивающих максимально возможное число проверяемых операторов, логических ветвлений и условий.

Под маршрутом понимают последовательности операторов программы, которые выполняются при конкретном варианте исходных данных.

Тестирование по принципу белого ящика включает в себя метод покрытия операторов, метод покрытия условий и метод покрытия маршрутов.

Тестирование может быть проведено на примере диаграммы вариантов использования «Публикация поста». Блок-схема алгоритма представлена на рисунке 21. Управляющий граф алгоритма, отображающий последовательность операторов, представлен на рисунке 22.

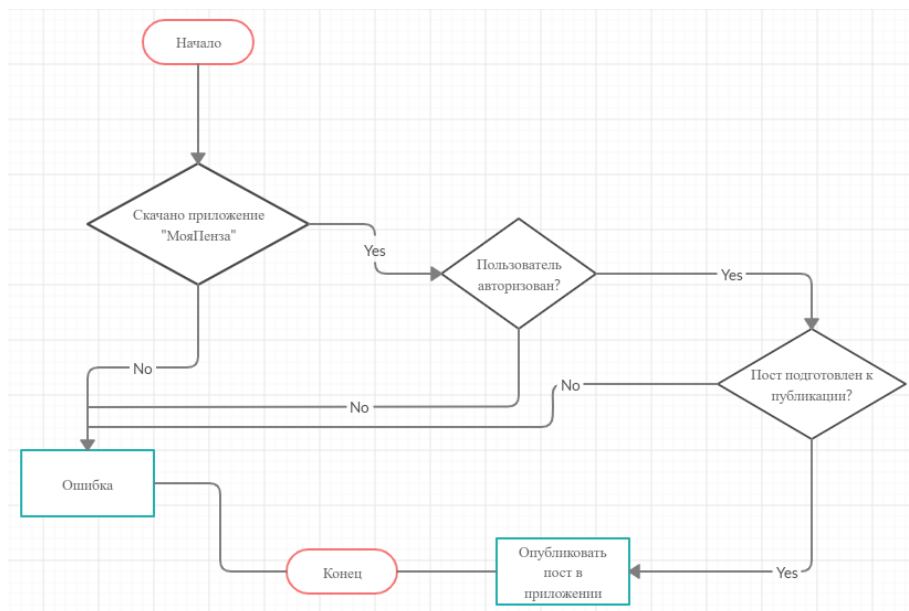


Рисунок 21 – Блок-схема прецедента «Публикация поста»

Описан процесс публикации новостного поста пользователем мобильного приложения.

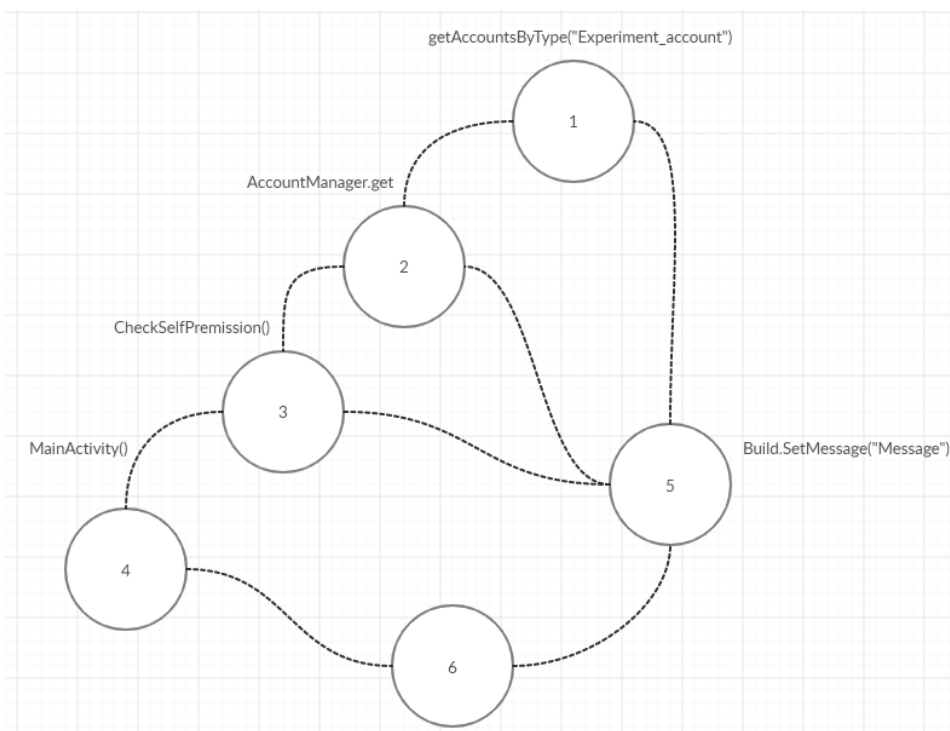


Рисунок 22 – Управляющий граф прецедента «Публикация поста»

Элемент кода алгоритма прецедента «Публикация поста» представлен в Приложении В. Алгоритм перехода.

4.3.2 Метод покрытия операторов

Метод покрытия операторов подразумевает под собой выполнение каждого оператора программы, по крайней мере, один раз.

Для выполнения каждого оператора были составлены тесты. Результаты приведены в таблице 7.

Таблица 7 – Результат тестирования методом покрытия операторов

Операторы	Тест	Фактический результат	Результат тестирования
4	Приложение скачано, Пользователь авторизован, пост подготовлен к публикации	Пост успешно опубликован	Успешно.
5, 6	Приложение не скачано.	Пост не опубликован	Успешно.

Посредством метода покрытия операторов, все тесты были успешно пройдены. Ожидаемые результаты полностью соответствуют получившимся.

4.3.3 Метод покрытия условий

Суть метода покрытия условий заключается в том, чтобы составить число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз. Также должно соблюдаться условие выполнения каждого оператора, по крайней мере, один раз.

Метод покрытия условий состоит в таком подборе тестов, когда каждое условие (элементарное суждения в условных операторах) принимает как истинное, так и ложное значение.

Для выполнения каждого условия метода покрытия условий было составлено четыре теста.

Результаты тестирования посредством метода покрытия условий приведены в таблице 8.

Таблица 8 – Результат тестирования методом покрытия условий

Условия	Тест	Фактический результат	Результат тестирования
1 – нет	Приложение не скачано.	Вывод ошибки приложения.	Успешно.
1 – да, 2 – нет	Приложение скачано, Пользователь не авторизован.	Вывод ошибки авторизации.	Успешно.
1 – да, 2 – да, 3 – нет	Приложение скачано, Пользователь авторизован, Пост не подготовлен	Вывод ошибки в получении прав.	Успешно.
1 – да, 2 – да, 3 – да	Приложение скачано, Пользователь авторизован, Пост подготовлен к публикации.	Пост опубликован	Успешно.

Посредством метода покрытия условий все тесты благополучно пройдены. Если пользователь не авторизован, выводится соответствующее сообщение об ошибке.

4.3.4 Метод покрытия маршрутов

Метод покрытия маршрутов является наиболее полным и результативным. Он предполагает прохождение по каждому возможному пути программы.

Согласно этому методу тесты должны составляться так, чтобы, по крайней мере, один раз выполнились все возможные результаты каждого

условия и все результаты каждого решения, и каждому оператору управление передавалось, по крайней мере, один раз.

Для выполнения всех маршрутов было составлено четыре теста. Результаты приведены в таблице 9.

Таблица 9 – Результат тестирования методом покрытия условий

Маршрут	Тест	Фактический результат	Результат тестирования
1, 5, 6	Приложение не скачано.	Вывод ошибки приложения.	Успешно.
1, 2, 5, 6	Приложение скачано, Пользователь не авторизован.	Вывод ошибки авторизации.	Успешно.
1, 2, 3, 5, 6	Приложение скачано, Пользователь авторизован, Пост не подготовлен	Вывод ошибки прав.	Успешно.
1, 2, 3, 4, 6	Приложение скачано, Пользователь авторизован, Пост подготовлен к публикации.	Пост опубликован	Успешно.

Посредством метода покрытия маршрутов, все тесты были успешно пройдены. Полученные результаты полностью соответствуют ожидаемым.

4.3.5 Качественный анализ кода

Определение качества кода является важной частью в мире программирования. В частности, от сложности кода системы зависит производительность труда разработчиков, а, следовательно, трудоемкость и сроки разработки. Между сложностью и надежностью программной системы также существует тесная связь. Очевидно, что чем выше сложность программной системы, тем труднее обеспечить ее надежность.

Сложность – это одна из главных причин ненадежности программного обеспечения. Сложность почти не поддается ни точному определению, ни

измерению. Однако можно сказать, что мерой сложности объекта является количество интеллектуальных усилий, необходимых для понимания этого объекта. Важным правилом программирования является соблюдение простоты кода. Более простой считается та программа, которая легче для понимания, отладки, сопровождения и модификации.

При оценке сложности программ в данной выпускной квалификационной работе выделены два типа метрик:

- 1) метрика размера программ;
- 2) метрика сложности потока управления программ.

Качественный анализ кода был проведен для листинга 4.1, реализующего публикацию поста.

4.3.6 SLOC-метрика

Количество строк кода (англ. Source Lines of Code — SLOC) – это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода. Как правило, этот показатель используется для прогноза трудозатрат на разработку конкретной программы на конкретном языке программирования, либо для оценки производительности труда уже после того, как программа написана.

Физическими строками считаются все непустые строки текстового файла. Применим SLOC-метрику к нашему элементу кода для публикации поста. Результаты данной метрики представлены в таблице 10.

Таблица 10 – SLOC-метрики

Число физических строк	83
Число логических строк	56
Число пустых строк	0
Число строк-комментариев	9

Измерены логические строки кода. Попытка посчитать количество операторов в программе.

4.3.7 Метрика Холстеда

Была рассмотрена метрика Холстеда. В данной метрике за базу принят подсчет количества операторов и операндов, используемых в программе, т.е. определение размера программы. Основу метрики Холстеда составляют четыре измеряемые характеристики программы: $n1$ – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций; $n2$ – число уникальных операндов программы; $N1$ – общее число операторов в программе; $N2$ – общее число операндов в программе. Подсчет каждой измеряемой характеристики представлен в таблице 11.

Таблица 11 – Операнды и операторы

Операторы	Число вхождений	Операнды	Число вхождений
=	7	@Override	1
.	62	accountManager	2
()	44	savedInstanceState	1
->	1	Permissions[]	1
[]	8	accounts	3
!=	2	requestCode	1
==	1	grantResults	4
&&	1	builder	4
onCreate()	1	intent	2
NextLevel()	2	dialog	3
onClick()	1	id	1
:	2		
>	1		
If()	4		
else	4		
new	8		
return	1		
$n1 = 17$	$N1 = 153$	$n2 = 11$	$N2 = 23$

На основании этих характеристик рассчитываются следующие оценки.

Длина программы рассчитывается по формуле:

$$N = N1 + N2.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2.$$

Вычислительная сложность рассчитывается по формуле:

$$HV = N * \log(2)n. \text{ (бит).}$$

Под битом подразумевается логическая единица информации - символ, оператор, операнд. Объем программы по Холстеду определяет объем кода алгоритма. Размер метрики для одной функции должен находиться в пределах от 20 до 1000.

В результате вычисления измеряемых характеристик метрики Холстеда были получены следующие значения:

$$N = 171; n = 28; HV = 833,09.$$

Полученный показатель объема программы по Холстеду находится в пределах нормы.

4.3.8 Цикломатическая метрика Мак-Кейба

В качестве метрики сложности потока управления (цикломатической сложности) программы была рассмотрена метрика Маккейба.

Программа представляется в виде управляющего ориентированного графа $G(v, e)$, где v – вершины, соответствующие операторам, а e – дуги, соответствующие переходам.

Основной метрикой сложности считается цикломатическая сложность графа программы, или цикломатическое число Маккейба, характеризующее трудоемкость тестирования программы.

Для вычисления цикломатического числа Маккейба $Z(G)$ применяется формула:

$$Z(G) = e - v + 2p,$$

где:

e - число дуг ориентированного графа G ;

v - число вершин;

p - число компонентов связности графа.

Цикломатическое число Маккейба определяет количество тестовых прогонов программы, необходимых для тестирования по критерию "работает каждая ветвь".

Подсчет цикломатической сложности был проведён на основе управляющего графа прецедента «Публикация поста» диаграммы вариантов использования. Показатели для данного графа:

$$e = 8; v = 6; p = 1.$$

В результате вычисления цикломатического числа метрики Маккейба на основе данных показателей был получено значение 4. Так как цикломатическая сложность метода не должна превышать 10, полученное значение является нормой.

Такой показатель, как индекс сопровождаемости рассчитывается как метрика относительной сложности поддержки, но без слагаемого, учитывающего количество комментариев, и может принимать значения от 0 до 100:

Индекс сопровождаемости кода определяется по формуле:

$$MI = \max(0, (171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LoC)) * 100 / 171),$$

где

HV – вычислительная сложность;

CC – структурная сложность кода, количество ветвей в коде;

LoC – количество строк кода.

$$CC = 4, LOC = 83, HV = 833,09.$$

В результате вычисления получено значение индекса сопровождаемости кода, равное 38. Таким образом, алгоритм кода публикация поста имеет приемлемый индекс сопровождаемости.

5 Планирование и расчёт бюджета разрабатываемого мобильного приложения

5.1 Этапы разработки

Необходимая и фундаментальная часть начала разработки мобильного приложения – идея. После рождения идеи необходимо составить план работы над проектом. Его проще всего начать с анализа конкурентов. Найти на рынке похожие мобильные приложения и проанализировать следующие факторы:

- количество загрузок – если приложение загружают, значит, это кому-нибудь нужно;
- оценки и отзывы – помогут предусмотреть слабые места и адаптировать «фишки», которые нравятся пользователям, под свой продукт.

Благодаря анализу конкурентов получается поймать сразу двух зайцев. Во-первых, имеется возможность выучиться на чужих ошибках, причём бесплатно, без траты лишнего времени и ощущения собственной неполноценности. Во-вторых, можно начать понимать, на что придётся пойти, чтобы выжить на рынке мобильной разработки.

Что именно пользователь хочет увидеть в своём смартфоне? Есть ли для вашего приложения свободная ниша? Ответы на эти вопросы в процессе разработки приложения, помогут найти баланс между возможностями на разработку и потребностями рынка. Если идея, разрабатываемого приложения, настолько уникальна, что подобных приложений ещё нет в природе, поэтому можно проанализировать то, как разработчики-первопроходцы из других сфер презентовали себя и свои проекты аудитории.

Также разработка приложений не ограничивается написанием кода. Она состоит из нескольких важнейших этапов, на протяжении которых определяется используемая предметная область, средства разработки, бюджет и соответственно сроки:

- планирование и оценка – знакомство с документацией заказчика, составление плана работ;
- аналитика – выявление требований и способов их реализации;
- дизайн – отрисовка экранов, подготовка материалов для вёрстки, иконок, скриншотов для магазинов, создание интерактивных прототипов;
- разработка – верстка, разработка API, подключение (иногда интеграция со сторонними сервисами);
- тестирование – проверка всех возможных сценариев использования в различных условиях;
- релиз – публикация приложения в магазинах Apple App Store и Google Play;
- сопровождение – поддержка и развитие проекта после релиза.

5.2 Диаграмма Ганта

Описание плана разработки и диаграммы Ганта представлены на рисунках 23 и 24 соответственно.

☰	Задача	Начало	Завершение
		27.04.2020	08.06.2020
☐	Android-приложение	27.04.2020	08.06.2020
	Планирование и оценка	27.04.2020	28.04.2020
	Аналитика	28.04.2020	08.05.2020
	Дизайн	08.05.2020	18.05.2020
	Разработка	18.05.2020	21.05.2020
	Тестирование	21.05.2020	25.05.2020
	Релиз	25.05.2020	01.06.2020
	Сопровождение	01.06.2020	08.06.2020

Рисунок 23 – План разработки

План разработки диаграммы Ганта представляет из себя хронологическое описание процессов производства мобильного приложения.

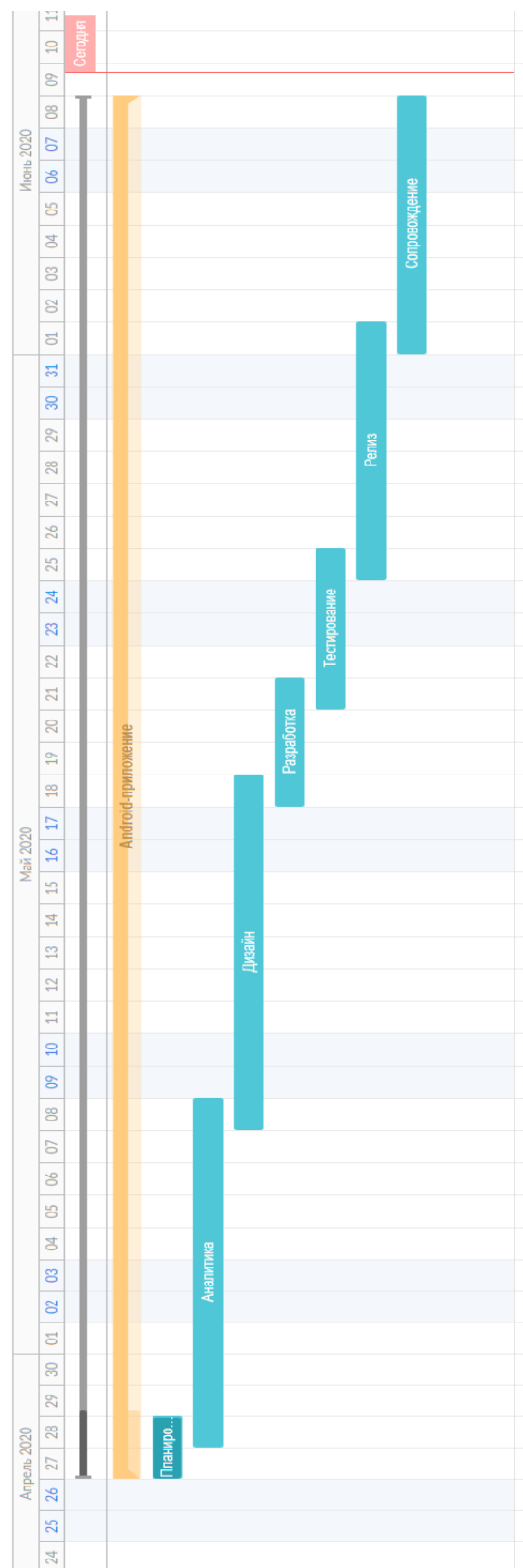


Рисунок 24 – Диаграмма Ганта

Диаграмма Ганта представляет из себя хронологическое описание процессов производсива мобильного приложения.

5.3 Проектирование приложения

Как правило, проект, в котором участвует Android-приложение, состоит из нескольких модулей (обычно приложение – это основной модуль).

Это и административная панель, в которой осуществляется управление контентом или пользователями, и веб-сайт проекта, и версия приложения для iPhone.

Чем больше, модулей – тем более тщательное проектирование требуется для успешной разработки и последующего функционирования, и развития проекта.

Необходимо спроектировать базы данных, интерфейсы между модулями, а также архитектуру самих модулей. Как правило, проектировщик – это более опытный и высокооплачиваемый специалист, чем разработчик, и стоимость проектирования может составлять до 30% стоимости всего проекта.

Ошибки в проектировании могут дорого стоить в будущем, поэтому, закладывая в стоимость проекта стоимость разработки архитектуры, мы косвенно экономим деньги и улучшаем качество проекта.

5.4 Дизайн приложения

Хороший дизайн – лицо проекта, на нём экономить не рекомендуется. Как правило, чем конкретнее ставить задачу дизайнеру – тем дешевле обходятся работы по дизайну. Самое дорогое – это идеи.

5.5 Разработка приложения

Написание кода, которое обычно только и принимается во внимание при оценке проекта, по трудоёмкости и стоимости составляет лишь половину бюджета проекта. Только в совсем небольших проектах весь бюджет идёт на разработку.

На стоимость разработка Android-приложения влияет количество разработчиков. Вообще-то, чем меньше разработчиков, тем лучше. Если по

каким-то причинам это невозможно – то взаимодействие разработчиков приводит к дополнительным затратам. Мы стараемся минимизировать и регламентировать взаимодействие разработчиков путем правила "один разработчик – один модуль" и описания интерфейсов взаимодействия модулей перед разработкой.

5.6 Тестирование и отладка

Ошибки – неотъемлемая часть отладки приложения. Это нормально для любого сложного продукта, которым и является Android-приложение. Количество ошибок минимизируется с помощью тестирования после окончания разработки, до ввода в коммерческую эксплуатацию.

5.7 Техническая поддержка

При вводе проекта в коммерческую эксплуатацию требуется его техническая поддержка. Она может производиться как силами заказчика (в этом случае очень важна качественная техническая документация и обучение сотрудников заказчика), так и силами разработчика (те же нормо-часы).

5.8 Формирование бюджета разработки

Цена продукта складывается из стандартных составляющих, некоторые из которых можно ликвидировать или заменить более дешевыми аналогами.

Если нужно узнать ориентировочную стоимость разработки, мы оцениваем техническую составляющую и в течение 1–3 дней имеется примерный план бюджета на разработку.

Если необходима более детальная оценкой, подготовка к разработке коммерческого предложения занимает от трех дней до 2–3 недель, в зависимости от объема проекта, его особенностей и особенностей бизнеса выбранной предметной области.

В течение этого периода студия или команда разработчиков анализирует бизнес-модель, проясняет технические детали, выявляет те или иные нюансы, прогнозирует различные исходы событий, которые могут возникнуть в ходе процесса разработки. Затем разбивается весь проект на итерации, оцениваются материальные и временные затраты на каждый этап проекта и рассчитываются примерные сроки разработки.

Чаше всего, если решение не очевидно, то имеется возможность рассмотреть ещё несколько возможных вариантов разработки на выбор, также рассматривается то, какие функции включать в первую версию, а какие лучше внедрять впоследствии, на что сделать акцент в приложении, чтобы привлечь пользователей.

Этап оценки представляет собой непрерывный диалог с клиентом, если разработка имеет коммерческий характер, целью которого является более глубокое понимание проблемы клиента для наиболее эффективного ее решения. В результате клиент получает не только смету проекта, но и примерную разбивку по срокам, а также предложения по оптимальным вариантам разработки и свежие идеи.

Так как приложение не является сложным приложением в плане разработки и проектирования для студии разработки приложений или какой-либо группы разработчиков, то стоит отталкиваться от средних цен на разработку простых Android-приложений.

Обычно под простыми приложениями понимают те, которые содержат около 5 экранов, не обмениваются данными ни с какими сторонними и чужими базами (сами не обновляют списки свободных столов, оставшихся билетов), не собирают аналитику действий пользователей, а личный кабинет в них если и есть, то имеет примитивный вид, то есть с регистрацией только по email.

Простые приложения – это, как правило, или визитки, то есть информация о компании, адрес, карта проезда, или программы, созданные

для одного несложного процесса вроде оформления заказа без выбора товара и добавления в корзину.

Приложение на одной платформе можно сделать быстрее и дешевле – в рамках месяца (150 часов) – но немногие клиенты, если заказ на приложение имеет коммерческий характер, согласны ограничиваться одним устройством. Для них и тех, кто хочет сэкономить время и деньги, есть решение в виде кроссплатформенной разработки, но там много подводных камней вроде ограниченности функциональности, визуализации и производительности, правда, для некоторых заказов это не критично.

Даже при раздельном написании код для каждой платформы не должен быть абсолютно уникальным. То есть, если программу сначала писать для одной платформы и потом адаптировать под другую, то это не увеличит стоимость разработки в два раза. Зато немного удлинит сроки.

5.9 Аренда помещения

Ориентировочная цена – 30 тыс. рублей в месяц. Также учитываются расходы на коммунальные платежи.

5.10 Проектирование и дизайн

Этим может заниматься и команда от трех специалистов (UX-дизайнер, UI-дизайнер и графический дизайнер), и один мастер на все руки. В первом варианте качество выше (цена тоже), но обычно студии используют второй. Ориентировочная цена – 15 – 25 тыс. рублей.

5.11 Этапы разработки

Делится на backend и frontend разработку, но мобильные программисты, специализирующиеся на определенной ОС, занимаются и тем, и другим. Исходя из среднего оклада сотрудника в 100 000 в месяц, получаем стоимость разработки простого приложения примерно в 120 тыс. рублей.

5.12 Тестирование и публикация

Тестированием занимается менеджер проекта или специальные тестировщики ПО, но зарплаты у них примерно одинаковые – около 80 тысяч в среднем. По практике, на тестирование и выкладывание в магазины приложений уходит от 2 дней до 2 недель непрерывной работы, то есть от 16 до 80 часов. Для простого приложения это будет стоить 8 000 рублей.

Прибавим сюда управление проектом, которое, скорее всего, будет осуществлять тот же менеджер. По опыту, он потратит на это около 3 дней непрерывной работы в простом варианте, 7 – в среднем и от 14 – в сложном. В деньгах такие траты – это 12 000 рублей. Ориентировочно получается – 20 тыс. рублей.

Вопрос, который требует не меньшего внимания – это планы по развитию приложения после релиза. Приложения в AppStore и GooglePlay требуют постоянной актуализации и поддержки, реакции на отзывы пользователей. Поэтому мы рекомендуем делить проект на несколько релизов и формируем цену каждого с учетом перечисленных факторов.

Резюмируя, стоит отметить, что очень редко можно оценить проект на этапе первичной оценки всего плана разработки мобильного приложения и изучения проблемы.

Чаще всего в той или иной степени приходится пересматривать бюджет на разработку проекта, и даже не раз. Но не стоит воспринимать это как проблему или неточность со стороны разработчиков: это нормальное состояние любого развивающегося проекта.

В совокупности всех вышеописанных этапов, можно прийти к выводу, что разработка мобильного приложения подобного рода суммарно может обойтись ориентировочно в 195 тыс. рублей.

Заключение

Результатом работы является мобильное приложение с интуитивно понятным пользовательским интерфейсом. Приложение предоставляет пользователю возможность зафиксировать то или иное наблюдаемое происшествие, которое описывается и в дальнейшем публикуется непосредственно внутри мобильного приложения в качестве новостного поста.

Разработанное мобильное приложение позволяет пользователю беспрепятственно и комфортно осуществлять навигацию по разделам, публиковать новостные посты, редактировать ту или иную информацию, связанную с персональной учётной записью или с уже опубликованным новостным постом. Приложение имеет простой, понятный и интуитивный интерфейс.

Отладка и тестирование мобильного приложения состоит из процесса прохождения теста по созданию и публикации новостного поста. Также проводилось тестирование программного обеспечения, структурное тестирование и тестирование связанное с использованием различных метрик и методов.

Работа над проектом позволила расширить знания и улучшить навыки создания мобильных приложений.

Программа может быть подвержена совершенствованию и оптимизации в дальнейшем, в соответствии с потребностями пользователей. Например, можно реализовать более приятный для восприятия пользовательский интерфейс или какие-либо функциональные возможности, связанные с более точным описанием при публикации новостного поста.

Список используемых источников

1. Ивлиева А.Г. Направления и перспективы экологического регулирования. / Под редакцией А.Г. Ивлиева, 2003. – 269 с.
2. Тимошенко Ю.А. Ответственность за экологические преступления. Теория и практика / Ю.А. Тимошенко, 2018. – 148 с.
3. Половцев, С.И., Калмыков; ФГОУ ВПО «Саратовский ГАУ». Экология и окружающая человека среда: Учебное пособие О.П. Половцев, С.И., Калмыков; ФГОУ «Саратовский ГАУ» / С.И. Калмыков, 2002. – 432 с.
4. Ян Ф.Дарвин. Android. Сборник рецептов. Задачи и решения для разработчиков приложений / Я.Ф. Дарвин, 2013. – 139 с.
5. Дэвид Гриффитс, Дон Гриффитс. Программирование для Android / Г. Дэвид, 2001. – 456 с.
6. Greg Nudelman. Android Design Patterns: Interaction Design Solutions for developers / G. Nudelman, 2019. – 356 с.
7. Контроль за состоянием окружающей среды [Электронный ресурс]. URL: <https://helpiks.org/9-2351.html>
8. Экологический контроль [Электронный ресурс]. URL: <https://www.ecoindustry.ru/literature.html?global=1>.

Приложение А
ГЛОССАРИЙ
(справочное)

Android Studio – это интегрированная среда разработки для работы с платформой Android.

Программное обеспечение (ПО) – это программа или множество программ, используемых для управления компьютером.

База данных (БД) – это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объемов.

Системы управления базами данных (СУБД) – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

JavaScript – это мультипарадигменный язык программирования, который обычно применяется в качестве встраиваемого инструмента для программного доступа к различным объектам приложений.

«Клиент - сервер» – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер – это программное обеспечение.

Android – это мобильная операционная система, основанная на модифицированной версии ядра Linux и другого программного обеспечения с открытым исходным кодом, разработанная в первую очередь для мобильных устройств с сенсорным экраном, таких как смартфоны и планшеты.

Приложение Б

ЛИСТИНГ ПЕРЕХОДА К БЛОКУ КОДА
(рекомендуемое)

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_authorization);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    if (ContextCompat.checkSelfPermission(Authorization.this,
Manifest.permission.GET_ACCOUNTS) != PackageManager.PERMISSION_GRANTED) {
        // Отобразить пояснение?
        if (ActivityCompat.shouldShowRequestPermissionRationale(Authorization.this,
Manifest.permission.GET_ACCOUNTS)) {
            ActivityCompat.requestPermissions(Authorization.this,
new String[]{Manifest.permission.GET_ACCOUNTS},
MY_PERMISSIONS_REQUEST_GET_ACCOUNTS);
            ActivityCompat.requestPermissions(Authorization.this,
new String[]{Manifest.permission.READ_CONTACTS},
MY_PERMISSIONS_REQUEST_GET_ACCOUNTS);
        } else {
            // Никаких объяснений не требуется
            ActivityCompat.requestPermissions(Authorization.this,
new String[]{Manifest.permission.GET_ACCOUNTS},
MY_PERMISSIONS_REQUEST_GET_ACCOUNTS);
            ActivityCompat.requestPermissions(Authorization.this,
new String[]{Manifest.permission.READ_CONTACTS},
MY_PERMISSIONS_REQUEST_GET_ACCOUNTS);
            // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
            // определяемая приложением константа int. The callback method gets the
            // результат запроса.
        }
    } else {
        NextLevel();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String permissions[],
int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_GET_ACCOUNTS: {
            // Если запрос отменяется, то массивы с результатами остаются пустыми
            if (grantResults.length > 0
&& grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                NextLevel();
            } else {
                AlertDialog.Builder builder = new AlertDialog.Builder(Authorization.this);

```

```

        builder.setTitle("Внимание!")
        .setMessage("К сожалению, пост не может быть опубликован")
        .setCancelable(false)
        .setNegativeButton("OK",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                    finishAffinity();
                }
            });
        AlertDialog alert = builder.create();
        alert.show();
    }
    return;
}
// другие строки "case" для проверки на наличие других
}
}
private void NextLevel() {
    AccountManager accountManager = AccountManager.get(Authorization.this);
    Account[] accounts =
        accountManager.getAccountsByType("experiment_account");
    if (accounts.length != 0) {
        Intent intent = new Intent(Authorization.this, MainActivity.class);
        intent.putExtra("userToken", accountManager.getUserData(accounts[0],
            "token"));
        startActivity(intent);
        finish();
    } else {
        AlertDialog.Builder builder = new AlertDialog.Builder(Authorization.this);
        builder.setTitle("Внимание!")
            .setMessage(R.string.link)
            .setCancelable(false)
            .setNegativeButton("Хорошо",
                (dialog, id) -> {
                    dialog.cancel();
                    finishAffinity();
                });
        AlertDialog alert = builder.create();
        alert.show();}}}

```

Приложение В
ЛИСТИНГ ПРОГРАММЫ
(рекомендуемое)

```

function() {
var PLATFORM_VERSION_BUILD_LABEL = '7.1.0';
var require;
var define;
(function () {
    var modules = {};
    // Stack of moduleIds currently being built.
    var requireStack = [];
    // Map of module ID -> index into requireStack of modules currently being built.
    var inProgressModules = {};
    var SEPARATOR = '.';
    function build (module) {
        var factory = module.factory;
        var localRequire = function (id) {
            var resultantId = id;
            // Its a relative path, so lop off the last portion and add the id (minus
            id;
            if (id.charAt(0) === '.') {
                resultantId = module.id.slice(0, module.id.lastIndexOf(SEPARATOR)) +
                SEPARATOR + id.slice(2);
            }
            return require(resultantId);
        };
        module.exports = {};
        delete module.factory;
        factory(localRequire, module.exports, module);
        return module.exports;
    }
    require = function (id) {
        if (!modules[id]) {
            throw "module " + id + " not found";
        } else if (id in inProgressModules) {
            var cycle = requireStack.slice(inProgressModules[id]).join('->') + '->' +
            id;
            throw 'Cycle in require graph: ' + cycle;
        }
        if (modules[id].factory) {
            try {
                inProgressModules[id] = requireStack.length;
                requireStack.push(id);
                return build(modules[id]);
            } finally {
                delete inProgressModules[id];
                requireStack.pop();
            }
        }
        return modules[id].exports;
    };
    define = function (id, factory) {
        if (modules[id]) {
            throw 'module ' + id + ' already defined';
        }
        modules[id] = {
            id: id,
            factory: factory
        };
    };
    define.remove = function (id) {
        delete modules[id];
    };
    define.moduleMap = modules;
})();

```

```

// Export for use in node
if (typeof module === 'object' && typeof require === 'function') {
    module.exports.require = require;
    module.exports.define = define;
}

// file: src/cordova.js
define("cordova", function(require, exports, module) {

    // Workaround for Windows 10 in hosted environment case
    // http://www.w3.org/html/wg/drafts/html/master/browsers.html#named-access-on-the-window-object
    if (window.cordova && !(window.cordova instanceof HTMLElement)) { // eslint-disable-line no-undef
        throw new Error('cordova already defined');
    }

    var channel = require('cordova/channel');
    var platform = require('cordova/platform');
    var m_document_addEventListener = document.addEventListener;
    var m_document_removeEventListener = document.removeEventListener;
    var m_window_addEventListener = window.addEventListener;
    var m_window_removeEventListener = window.removeEventListener;
    var documentEventHandlers = {};
    var windowEventHandlers = {};
    document.addEventListener = function (evt, handler, capture) {
        var e = evt.toLowerCase();
        if (typeof documentEventHandlers[e] !== 'undefined') {
            documentEventHandlers[e].subscribe(handler);
        } else {
            m_document_addEventListener.call(document, evt, handler, capture);
        }
    };
    window.addEventListener = function (evt, handler, capture) {
        var e = evt.toLowerCase();
        if (typeof windowEventHandlers[e] !== 'undefined') {
            windowEventHandlers[e].subscribe(handler);
        } else {
            m_window_addEventListener.call(window, evt, handler, capture);
        }
    };
    document.removeEventListener = function (evt, handler, capture) {
        var e = evt.toLowerCase();
        // If unsubscribing from an event that is handled by a plugin
        if (typeof documentEventHandlers[e] !== 'undefined') {
            documentEventHandlers[e].unsubscribe(handler);
        } else {
            m_document_removeEventListener.call(document, evt, handler, capture);
        }
    };
    window.removeEventListener = function (evt, handler, capture) {
        var e = evt.toLowerCase();
        // If unsubscribing from an event that is handled by a plugin
        if (typeof windowEventHandlers[e] !== 'undefined') {
            windowEventHandlers[e].unsubscribe(handler);
        } else {
            m_window_removeEventListener.call(window, evt, handler, capture);
        }
    };
    function createEvent (type, data) {
        var event = document.createEvent('Events');
        event.initEvent(type, false, false);
        if (data) {

```

```

        for (var i in data) {
            if (data.hasOwnProperty(i)) {
                event[i] = data[i];
            }
        }
    }
    return event;
}
var cordova = {
    define: define,
    require: require,
    version: PLATFORM_VERSION_BUILD_LABEL,
    platformVersion: PLATFORM_VERSION_BUILD_LABEL,
    platformId: platform.id,
    addWindowEventHandler: function (event) {
        return (windowEventHandlers[event] = channel.create(event));
    },
    addStickyDocumentEventHandler: function (event) {
        return (documentEventHandlers[event] = channel.createSticky(event));
    },
    addDocumentEventHandler: function (event) {
        return (documentEventHandlers[event] = channel.create(event));
    },
    removeWindowEventHandler: function (event) {
        delete windowEventHandlers[event];
    },
    removeDocumentEventHandler: function (event) {
        delete documentEventHandlers[event];
    },
    getOriginalHandlers: function () {
        return {'document': {'addEventListener': m_document_addEventListener,
'removeEventListener': m_document_removeEventListener},
'window': {'addEventListener': m_window_addEventListener,
'removeEventListener': m_window_removeEventListener}};
    },
    fireDocumentEvent: function (type, data, bNoDetach) {
        var evt = createEvent(type, data);
        if (typeof documentEventHandlers[type] !== 'undefined') {
            if (bNoDetach) {
                documentEventHandlers[type].fire(evt);
            } else {
                setTimeout(function () {
                    // Fire deviceready on listeners that were registered before
cordova.js was loaded.
                    if (type === 'deviceready') {
                        document.dispatchEvent(evt);
                    }
                    documentEventHandlers[type].fire(evt);
                }, 0);
            }
        } else {
            document.dispatchEvent(evt);
        }
    },
    fireWindowEvent: function (type, data) {
        var evt = createEvent(type, data);
        if (typeof windowEventHandlers[type] !== 'undefined') {
            setTimeout(function () {
                windowEventHandlers[type].fire(evt);
            }, 0);
        } else {
            window.dispatchEvent(evt);
        }
    }
};

```

```

    }
    callbackId: Math.floor(Math.random() * 2000000000),
    callbacks: {},
    callbackStatus: {
        NO_RESULT: 0,
        OK: 1,
        CLASS_NOT_FOUND_EXCEPTION: 2,
        ILLEGAL_ACCESS_EXCEPTION: 3,
        INSTANTIATION_EXCEPTION: 4,
        MALFORMED_URL_EXCEPTION: 5,
        IO_EXCEPTION: 6,
        INVALID_ACTION: 7,
        JSON_EXCEPTION: 8,
        ERROR: 9
    },
    callbackSuccess: function (callbackId, args) {
        cordova.callbackFromNative(callbackId, true, args.status, [args.message],
args.keepCallback);
    },

    /**
     * Called by native code when returning error result from an action.
     */
    callbackError: function (callbackId, args) {
        // TODO: Deprecate callbackSuccess and callbackError in favour of
callbackFromNative.
        // Derive success from status.
        cordova.callbackFromNative(callbackId, false, args.status, [args.message],
args.keepCallback);
    },
    callbackFromNative: function (callbackId, isSuccess, status, args, keepCallback) {
        try {
            var callback = cordova.callbacks[callbackId];
            if (callback) {
                if (isSuccess && status === cordova.callbackStatus.OK) {
                    callback.success && callback.success.apply(null, args);
                } else if (!isSuccess) {
                    callback.fail && callback.fail.apply(null, args);
                }
                /*
                else
                    Note, this case is intentionally not caught.
                    this can happen if isSuccess is true, but callbackStatus is
NO_RESULT
                    which is used to remove a callback from the list without calling the
callbacks
                    typically keepCallback is false in this case
                    if (!keepCallback) {
                        delete cordova.callbacks[callbackId];
                    }
                */
            }
        } catch (err) {
            var msg = 'Error in ' + (isSuccess ? 'Success' : 'Error') + ' callbackId: '
+ callbackId + ' : ' + err;
            console && console.log && console.log(msg);
            cordova.fireWindowEvent('cordovacallbackerror', { 'message': msg });
            throw err;
        }
    },
    addConstructor: function (func) {
        channel.onCordovaReady.subscribe(function () {
            try {
                func();
            }

```

```

        } catch (e) {
            console.log('Failed to run constructor: ' + e);
        }
    });
}
};
module.exports = cordova;
});
var nativeApi = this._cordovaNative || require('cordova/android/promptbasednativeapi');
var currentApi = nativeApi;
module.exports = {
    get: function() { return currentApi; },
    setPreferPrompt: function(value) {
        currentApi = value ? require('cordova/android/promptbasednativeapi') :
nativeApi;
    },
    // Used only by tests.
    set: function(value) {
        currentApi = value;
    }
};
});
module.exports = {
    exec: function(bridgeSecret, service, action, callbackId, argsJson) {
        return prompt(argsJson, 'gap:'+JSON.stringify([bridgeSecret, service, action,
callbackId]));
    },
    setNativeToJsBridgeMode: function(bridgeSecret, value) {
        prompt(value, 'gap_bridge_mode:' + bridgeSecret);
    },
    retrieveJsMessages: function(bridgeSecret, fromOnlineEvent) {
        return prompt(+fromOnlineEvent, 'gap_poll:' + bridgeSecret);
    }
};
});
define("cordova/argscheck", function(require, exports, module) {
var utils = require('cordova/utils');
var moduleExports = module.exports;
var typeMap = {
    'A': 'Array',
    'D': 'Date',
    'N': 'Number',
    'S': 'String',
    'F': 'Function',
    'O': 'Object'
};
function extractParamName (callee, argIndex) {
    return (/.*?\(((.*?)\))/).exec(callee)[1].split(', ')[argIndex];
}
function checkArgs (spec, functionName, args, opt_callee) {
    if (!moduleExports.enableChecks) {
        return;
    }
    var errMsg = null;
    var typeName;
    for (var i = 0; i < spec.length; ++i) {
        var c = spec.charAt(i);
        var cUpper = c.toUpperCase();
        var arg = args[i];
        if (c === '*') {
            continue;
        }
        typeName = utils.typeName(arg);
    }
    return errMsg;
}

```

```

        if ((arg === null || arg === undefined) && c === cUpper) {
            continue;
        }
        if (typeName !== typeMap[cUpper]) {
            errMsg = 'Expected ' + typeMap[cUpper];
            break;
        }
    }
    if (errMsg) {
        errMsg += ', but got ' + typeName + '.';
        errMsg = 'Wrong type for parameter "' + extractParamName(opt_callee ||
args.callee, i) + '" of ' + functionName + ': ' + errMsg;
        if (typeof jasmine === 'undefined') {
            console.error(errMsg);
        }
        throw TypeError(errMsg);
    }
}
function getValue (value, defaultValue) {
    return value === undefined ? defaultValue : value;
}
moduleExports.checkArgs = checkArgs;
moduleExports.getValue = getValue;
moduleExports.enableChecks = true;
});
define("cordova/base64", function(require, exports, module) {
var base64 = exports;
base64.fromArrayBuffer = function (arrayBuffer) {
    var array = new Uint8Array(arrayBuffer);
    return uint8ToBase64(array);
};
base64.toArrayBuffer = function (str) {
    var decodedStr = typeof atob !== 'undefined' ? atob(str) : Buffer.from(str,
'base64').toString('binary'); // eslint-disable-line no-undef
    var arrayBuffer = new ArrayBuffer(decodedStr.length);
    var array = new Uint8Array(arrayBuffer);
    for (var i = 0, len = decodedStr.length; i < len; i++) {
        array[i] = decodedStr.charCodeAt(i);
    }
    return arrayBuffer;
};
var b64_6bit = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
var b64_12bit;

var b64_12bitTable = function () {
    b64_12bit = [];
    for (var i = 0; i < 64; i++) {
        for (var j = 0; j < 64; j++) {
            b64_12bit[i * 64 + j] = b64_6bit[i] + b64_6bit[j];
        }
    }
    b64_12bitTable = function () { return b64_12bit; };
    return b64_12bit;
};

function uint8ToBase64 (rawData) {
    var numBytes = rawData.byteLength;
    var output = '';
    var segment;
    var table = b64_12bitTable();
    for (var i = 0; i < numBytes - 2; i += 3) {
        segment = (rawData[i] << 16) + (rawData[i + 1] << 8) + rawData[i + 2];
        output += table[segment >> 12];
        output += table[segment & 0xfff];
    }

```

```

    }
    if (numBytes - i === 2) {
        segment = (rawData[i] << 16) + (rawData[i + 1] << 8);
        output += table[segment >> 12];
        output += b64_6bit[(segment & 0xfff) >> 6];
        output += '=';
    } else if (numBytes - i === 1) {
        segment = (rawData[i] << 16);
        output += table[segment >> 12];
        output += '==';
    }
    return output;
}

});
var utils = require('cordova/utils');
function each (objects, func, context) {
    for (var prop in objects) {
        if (objects.hasOwnProperty(prop)) {
            func.apply(context, [objects[prop], prop]);
        }
    }
}
function clobber (obj, key, value) {
    exports.replaceHookForTesting(obj, key);
    var needsProperty = false;
    try {
        obj[key] = value;
    } catch (e) {
        needsProperty = true;
    }
    // Getters can only be overridden by getters.
    if (needsProperty || obj[key] !== value) {
        utils.defineGetter(obj, key, function () {
            return value;
        });
    }
}
function assignOrWrapInDeprecateGetter (obj, key, value, message) {
    if (message) {
        utils.defineGetter(obj, key, function () {
            console.log(message);
            delete obj[key];
            clobber(obj, key, value);
            return value;
        });
    } else {
        clobber(obj, key, value);
    }
}
function include (parent, objects, clobber, merge) {
    each(objects, function (obj, key) {
        try {
            var result = obj.path ? require(obj.path) : {};

            if (clobber) {
                // Clobber if it doesn't exist.
                if (typeof parent[key] === 'undefined') {
                    assignOrWrapInDeprecateGetter(parent, key, result, obj.deprecated);
                } else if (typeof obj.path !== 'undefined') {
                    // If merging, merge properties onto parent, otherwise, clobber.
                    if (merge) {
                        recursiveMerge(parent[key], result);
                    }
                }
            }
        }
    });
}

```

```

        } else {
            assignOrWrapInDeprecateGetter(parent, key, result,
obj.deprecated);
        }
    }
    result = parent[key];
} else {
    // Overwrite if not currently defined.
    if (typeof parent[key] === 'undefined') {
        assignOrWrapInDeprecateGetter(parent, key, result, obj.deprecated);
    } else {
        // Set result to what already exists, so we can build children into
it if they exist.
        result = parent[key];
    }
}

    if (obj.children) {
        include(result, obj.children, clobber, merge);
    }
} catch (e) {
    utils.alert('Exception building Cordova JS globals: ' + e + ' for key "' +
key + '"');
}
});
}
function recursiveMerge (target, src) {
    for (var prop in src) {
        if (src.hasOwnProperty(prop)) {
            if (target.prototype && target.prototype.constructor === target) {
                // If the target object is a constructor override off prototype.
                clobber(target.prototype, prop, src[prop]);
            } else {
                if (typeof src[prop] === 'object' && typeof target[prop] === 'object') {
                    recursiveMerge(target[prop], src[prop]);
                } else {
                    clobber(target, prop, src[prop]);
                }
            }
        }
    }
}

exports.buildIntoButDoNotClobber = function (objects, target) {
    include(target, objects, false, false);
};
exports.buildIntoAndClobber = function (objects, target) {
    include(target, objects, true, false);
};
exports.buildIntoAndMerge = function (objects, target) {
    include(target, objects, true, true);
};
exports.recursiveMerge = recursiveMerge;
exports.assignOrWrapInDeprecateGetter = assignOrWrapInDeprecateGetter;
exports.replaceHookForTesting = function () {};

});

// file: src/common/channel.js
define("cordova/channel", function(require, exports, module) {

var utils = require('cordova/utils');
var nextGuid = 1;

```

```

var Channel = function (type, sticky) {
  this.type = type;
  // Map of guid -> function.
  this.handlers = {};
  // 0 = Non-sticky, 1 = Sticky non-fired, 2 = Sticky fired.
  this.state = sticky ? 1 : 0;
  // Used in sticky mode to remember args passed to fire().
  this.fireArgs = null;
  // Used by onHasSubscribersChange to know if there are any listeners.
  this.numHandlers = 0;
  // Function that is called when the first listener is subscribed, or when
  // the last listener is unsubscribed.
  this.onHasSubscribersChange = null;
};

var channel = {
  /**
   * Calls the provided function only after all of the channels specified
   * have been fired. All channels must be sticky channels.
   */
  join: function (h, c) {
    var len = c.length;
    var i = len;
    var f = function () {
      if (!(--i)) h();
    };
    for (var j = 0; j < len; j++) {
      if (c[j].state === 0) {
        throw Error('Can only use join with sticky channels.');
```

```

Channel.prototype.subscribe = function (eventListenerOrFunction, eventListener) {
  checkSubscriptionArgument(eventListenerOrFunction);
  var handleEvent, guid;

  if (eventListenerOrFunction && typeof eventListenerOrFunction === 'object') {
    // Received an EventListener object implementing the handleEvent interface
    handleEvent = eventListenerOrFunction.handleEvent;
    eventListener = eventListenerOrFunction;
  } else {
    // Received a function to handle event
    handleEvent = eventListenerOrFunction;
  }

  if (this.state === 2) {
    handleEvent.apply(eventListener || this, this.fireArgs);
    return;
  }

  guid = eventListenerOrFunction.observer_guid;
  if (typeof eventListener === 'object') {
    handleEvent = utils.close(eventListener, handleEvent);
  }

  if (!guid) {
    // First time any channel has seen this subscriber
    guid = '' + nextGuid++;
  }
  handleEvent.observer_guid = guid;
  eventListenerOrFunction.observer_guid = guid;

  // Don't add the same handler more than once.
  if (!this.handlers[guid]) {
    this.handlers[guid] = handleEvent;
    this.numHandlers++;
    if (this.numHandlers === 1) {
      this.onHasSubscribersChange && this.onHasSubscribersChange();
    }
  }
};

Channel.prototype.unsubscribe = function (eventListenerOrFunction) {
  checkSubscriptionArgument(eventListenerOrFunction);
  var handleEvent, guid, handler;

  if (eventListenerOrFunction && typeof eventListenerOrFunction === 'object') {
    // Received an EventListener object implementing the handleEvent interface
    handleEvent = eventListenerOrFunction.handleEvent;
  } else {
    // Received a function to handle event
    handleEvent = eventListenerOrFunction;
  }

  guid = handleEvent.observer_guid;
  handler = this.handlers[guid];
  if (handler) {
    delete this.handlers[guid];
    this.numHandlers--;
    if (this.numHandlers === 0) {
      this.onHasSubscribersChange && this.onHasSubscribersChange();
    }
  }
};

Channel.prototype.fire = function (e) {
  var fail = false; // eslint-disable-line no-unused-vars

```

```

var fireArgs = Array.prototype.slice.call(arguments);
// Apply stickiness.
if (this.state === 1) {
    this.state = 2;
    this.fireArgs = fireArgs;
}
if (this.numHandlers) {
    // Copy the values first so that it is safe to modify it from within
    // callbacks.
    var toCall = [];
    for (var item in this.handlers) {
        toCall.push(this.handlers[item]);
    }
    for (var i = 0; i < toCall.length; ++i) {
        toCall[i].apply(this, fireArgs);
    }
    if (this.state === 2 && this.numHandlers) {
        this.numHandlers = 0;
        this.handlers = {};
        this.onHasSubscribersChange && this.onHasSubscribersChange();
    }
}
};
module.exports = channel;
});
// file: /Users/steveng/repo/cordova/cordova-android/cordova-js-src/exec.js
define("cordova/exec", function(require, exports, module) {
var cordova = require('cordova'),
    nativeApiProvider = require('cordova/android/nativeapiprovider'),
    utils = require('cordova/utils'),
    base64 = require('cordova/base64'),
    channel = require('cordova/channel'),
    jsToNativeModes = {
        PROMPT: 0,
        JS_OBJECT: 1
    },
    nativeToJsModes = {

        ONLINE_EVENT: 2,
        EVAL_BRIDGE: 3
    },
    jsToNativeBridgeMode, // Set lazily.
    nativeToJsBridgeMode = nativeToJsModes.EVAL_BRIDGE,
    pollEnabled = false,
    bridgeSecret = -1;
var messagesFromNative = [];
var isProcessing = false;
var resolvedPromise = typeof Promise == 'undefined' ? null : Promise.resolve();
var nextTick = resolvedPromise ? function(fn) { resolvedPromise.then(fn); } :
function(fn) { setTimeout(fn); };
function androidExec(success, fail, service, action, args) {
    if (bridgeSecret < 0) {
        // If we ever catch this firing, we'll need to queue up exec()s
        // and fire them once we get a secret. For now, I don't think
        // it's possible for exec() to be called since plugins are parsed but
        // not run until after onNativeReady.
        throw new Error('exec() called without bridgeSecret');
    }
    // Set default bridge modes if they have not already been set.
    // By default, we use the failsafe, since addJavascriptInterface breaks too often
    if (jsToNativeBridgeMode === undefined) {
        androidExec.setJsToNativeBridgeMode(jsToNativeModes.JS_OBJECT);
    }
}

```

```

// If args is not provided, default to an empty array
args = args || [];

// Process any ArrayBuffers in the args into a string.
for (var i = 0; i < args.length; i++) {
    if (utils.typeName(args[i]) == 'ArrayBuffer') {
        args[i] = base64.fromArrayBuffer(args[i]);
    }
}
var callbackId = service + cordova.callbackId++,
    argsJson = JSON.stringify(args);
if (success || fail) {
    cordova.callbacks[callbackId] = {success:success, fail:fail};
}
var msgs = nativeApiProvider.get().exec(bridgeSecret, service, action, callbackId,
argsJson);
// If argsJson was received by Java as null, try again with the PROMPT bridge mode.
// This happens in rare circumstances, such as when certain Unicode characters are
passed over the bridge on a Galaxy S2. See CB-2666.
if (jsToNativeBridgeMode == jsToNativeModes.JS_OBJECT && msgs === "@Null
arguments.") {
    androidExec.setJsToNativeBridgeMode(jsToNativeModes.PROMPT);
    androidExec(success, fail, service, action, args);
    androidExec.setJsToNativeBridgeMode(jsToNativeModes.JS_OBJECT);
} else if (msgs) {
    messagesFromNative.push(msgs);
    // Always process async to avoid exceptions messing up stack.
    nextTick(processMessages);
}
}
}
androidExec.init = function() {
    //CB-11828
    //This failsafe checks the version of Android and if it's Jellybean, it switches it
to
    //using the Online Event bridge for communicating from Native to JS
    //
    //It's ugly, but it's necessary.
    var check = navigator.userAgent.toLowerCase().match(/android\s[0-9].[0-9]/);
    var version_code = check && check[0].match(/4.[0-3].*/);
    if (version_code != null && nativeToJsBridgeMode == nativeToJsModes.EVAL_BRIDGE) {
        nativeToJsBridgeMode = nativeToJsModes.ONLINE_EVENT;
    }
    bridgeSecret = +prompt('', 'gap_init:' + nativeToJsBridgeMode);
    channel.onNativeReady.fire();
};
function pollOnceFromOnlineEvent() {
    pollOnce(true);
}
function pollOnce(opt_fromOnlineEvent) {
    if (bridgeSecret < 0) {
        // This can happen when the NativeToJsMessageQueue resets the online state on
page transitions.
        // We know there's nothing to retrieve, so no need to poll.
        return;
    }
    var msgs = nativeApiProvider.get().retrieveJsMessages(bridgeSecret,
!!opt_fromOnlineEvent);
    if (msgs) {
        messagesFromNative.push(msgs);
        // Process sync since we know we're already top-of-stack.
        processMessages();
    }
}
}

```

```

function pollingTimerFunc() {
    if (pollEnabled) {
        pollOnce();
        setTimeout(pollingTimerFunc, 50);
    }
}
function hookOnlineApis() {
    function proxyEvent(e) {
        cordova.fireWindowEvent(e.type);
    }
    // The network module takes care of firing online and offline events.
    // It currently fires them only on document though, so we bridge them
    // to window here (while first listening for exec()-related online/offline
    // events).
    window.addEventListener('online', pollOnceFromOnlineEvent, false);
    window.addEventListener('offline', pollOnceFromOnlineEvent, false);
    cordova.addWindowEventHandler('online');
    cordova.addWindowEventHandler('offline');
    document.addEventListener('online', proxyEvent, false);
    document.addEventListener('offline', proxyEvent, false);
}

hookOnlineApis();
androidExec.jsToNativeModes = jsToNativeModes;
androidExec.nativeToJsModes = nativeToJsModes;
androidExec.setJsToNativeBridgeMode = function(mode) {
    if (mode == jsToNativeModes.JS_OBJECT && !window._cordovaNative) {
        mode = jsToNativeModes.PROMPT;
    }
    nativeApiProvider.setPreferPrompt(mode == jsToNativeModes.PROMPT);
    jsToNativeBridgeMode = mode;
};
androidExec.setNativeToJsBridgeMode = function(mode) {
    if (mode == nativeToJsBridgeMode) {
        return;
    }
    if (nativeToJsBridgeMode == nativeToJsModes.POLLING) {
        pollEnabled = false;
    }
    nativeToJsBridgeMode = mode;
    // Tell the native side to switch modes.
    // Otherwise, it will be set by androidExec.init()
    if (bridgeSecret >= 0) {
        nativeApiProvider.get().setNativeToJsBridgeMode(bridgeSecret, mode);
    }

    if (mode == nativeToJsModes.POLLING) {
        pollEnabled = true;
        setTimeout(pollingTimerFunc, 1);
    }
};
function buildPayload(payload, message) {
    var payloadKind = message.charAt(0);
    if (payloadKind == 's') {
        payload.push(message.slice(1));
    } else if (payloadKind == 't') {
        payload.push(true);
    } else if (payloadKind == 'f') {
        payload.push(false);
    } else if (payloadKind == 'N') {
        payload.push(null);
    } else if (payloadKind == 'n') {
        payload.push(+message.slice(1));
    }
}

```

```

    } else if (payloadKind == 'A') {
        var data = message.slice(1);
        payload.push(base64.toArrayBuffer(data));
    } else if (payloadKind == 'S') {
        payload.push(window.atob(message.slice(1)));
    } else if (payloadKind == 'M') {
        var multipartMessages = message.slice(1);
        while (multipartMessages !== "") {
            var spaceIdx = multipartMessages.indexOf(' ');
            var msgLen = +multipartMessages.slice(0, spaceIdx);
            var multipartMessage = multipartMessages.substr(spaceIdx + 1, msgLen);
            multipartMessages = multipartMessages.slice(spaceIdx + msgLen + 1);
            buildPayload(payload, multipartMessage);
        }
    } else {
        payload.push(JSON.parse(message));
    }
}
// Processes a single message, as encoded by NativeToJsMessageQueue.java.
function processMessage(message) {
    var firstChar = message.charAt(0);
    if (firstChar == 'J') {
        // This is deprecated on the .java side. It doesn't work with CSP enabled.
        eval(message.slice(1));
    } else if (firstChar == 'S' || firstChar == 'F') {
        var success = firstChar == 'S';
        var keepCallback = message.charAt(1) == '1';
        var spaceIdx = message.indexOf(' ', 2);
        var status = +message.slice(2, spaceIdx);
        var nextSpaceIdx = message.indexOf(' ', spaceIdx + 1);
        var callbackId = message.slice(spaceIdx + 1, nextSpaceIdx);
        var payloadMessage = message.slice(nextSpaceIdx + 1);
        var payload = [];
        buildPayload(payload, payloadMessage);
        cordova.callbackFromNative(callbackId, success, status, payload, keepCallback);
    } else {
        console.log("processMessage failed: invalid message: " +
JSON.stringify(message));
    }
}
function processMessages() {
    // Check for the reentrant case.
    if (isProcessing) {
        return;
    }
    if (messagesFromNative.length === 0) {
        return;
    }
    isProcessing = true;
    try {
        var msg = popMessageFromQueue();
        // The Java side can send a * message to indicate that it
        // still has messages waiting to be retrieved.
        if (msg == '*' && messagesFromNative.length === 0) {
            nextTick(pollOnce);
            return;
        }
        processMessage(msg);
    } finally {
        isProcessing = false;
        if (messagesFromNative.length > 0) {
            nextTick(processMessages);
        }
    }
}

```

```

    }
}

function popMessageFromQueue() {
    var messageBatch = messagesFromNative.shift();
    if (messageBatch == '*') {
        return '*';
    }

    var spaceIdx = messageBatch.indexOf(' ');
    var msgLen = +messageBatch.slice(0, spaceIdx);
    var message = messageBatch.substr(spaceIdx + 1, msgLen);
    messageBatch = messageBatch.slice(spaceIdx + msgLen + 1);
    if (messageBatch) {
        messagesFromNative.unshift(messageBatch);
    }
    return message;
}
module.exports = androidExec;

});

// file: src/common/exec/proxy.js
define("cordova/exec/proxy", function(require, exports, module) {
// internal map of proxy function
var CommandProxyMap = {};
module.exports = {

    // example: cordova.commandProxy.add("Accelerometer",{getCurrentAcceleration:
function(successCallback, errorCallback, options) {...},...);
    add: function (id, proxyObj) {
        console.log('adding proxy for ' + id);
        CommandProxyMap[id] = proxyObj;
        return proxyObj;
    },
    // cordova.commandProxy.remove("Accelerometer");
    remove: function (id) {
        var proxy = CommandProxyMap[id];
        delete CommandProxyMap[id];
        CommandProxyMap[id] = null;
        return proxy;
    },

    get: function (service, action) {
        return (CommandProxyMap[service] ? CommandProxyMap[service][action] : null);
    }
};
});

// file: src/common/init.js
define("cordova/init", function(require, exports, module) {
var channel = require('cordova/channel');
var cordova = require('cordova');
var modulemapper = require('cordova/modulemapper');
var platform = require('cordova/platform');
var pluginloader = require('cordova/pluginloader');
var utils = require('cordova/utils');

var platformInitChannelsArray = [channel.onNativeReady, channel.onPluginsReady];

function logUnfiredChannels (arr) {
    for (var i = 0; i < arr.length; ++i) {
        if (arr[i].state !== 2) {
            console.log('Channel not fired: ' + arr[i].type);
        }
    }
}

```

```

    }
  }
}
window.setTimeout(function () {
  if (channel.onDeviceReady.state !== 2) {
    console.log('deviceready has not fired after 5 seconds.');
```

logUnfiredChannels(platformInitChannelsArray);

logUnfiredChannels(channel.deviceReadyChannelsArray);

```

  }
}, 5000);
// Replace navigator before any modules are required(), to ensure it happens as soon as
possible.
// We replace it so that properties that can't be clobbered can instead be overridden.
function replaceNavigator (origNavigator) {
  var CordovaNavigator = function () {};
  CordovaNavigator.prototype = origNavigator;
  var newNavigator = new CordovaNavigator();
  // This work-around really only applies to new APIs that are newer than
Function.bind.
  // Without it, APIs such as getGamepads() break.
  if (CordovaNavigator.bind) {
    for (var key in origNavigator) {
      if (typeof origNavigator[key] === 'function') {
        newNavigator[key] = origNavigator[key].bind(origNavigator);
      } else {
        (function (k) {
          utils.defineGetterSetter(newNavigator, key, function () {
            return origNavigator[k];
          });
        })(key);
      }
    }
  }
  return newNavigator;
}
if (window.navigator) {
  window.navigator = replaceNavigator(window.navigator);
}
if (!window.console) {
  window.console = {
    log: function () {}
  };
}
if (!window.console.warn) {
  window.console.warn = function (msg) {
    this.log('warn: ' + msg);
  };
}
// Register pause, resume and deviceready channels as events on document.
channel.onPause = cordova.addDocumentEventHandler('pause');
channel.onResume = cordova.addDocumentEventHandler('resume');
channel.onActivated = cordova.addDocumentEventHandler('activated');
channel.onDeviceReady = cordova.addStickyDocumentEventHandler('deviceready');
```

// Listen for DOMContentLoaded and notify our channel subscribers.

```

if (document.readyState === 'complete' || document.readyState === 'interactive') {
  channel.onDOMContentLoaded.fire();
} else {
  document.addEventListener('DOMContentLoaded', function () {
    channel.onDOMContentLoaded.fire();
  }, false);
}
// _nativeReady is global variable that the native side can set
```

```

// to signify that the native code is ready. It is a global since
// it may be called before any cordova JS is ready.
if (window._nativeReady) {
    channel.onNativeReady.fire();
}
modulemapper.clobbers('cordova', 'cordova');
modulemapper.clobbers('cordova/exec', 'cordova.exec');
modulemapper.clobbers('cordova/exec', 'Cordova.exec');
// Call the platform-specific initialization.
platform.bootstrap && platform.bootstrap();
// Wrap in a setTimeout to support the use-case of having plugin JS appended to
cordova.js.
// The delay allows the attached modules to be defined before the plugin loader looks
for them.
setTimeout(function () {
    pluginloader.load(function () {
        channel.onPluginsReady.fire();
    });
}, 0);
/**
 * Create all cordova objects once native side is ready.
 */
channel.join(function () {
    modulemapper.mapModules(window);

    platform.initialize && platform.initialize();

    // Fire event to notify that all objects are created
    channel.onCordovaReady.fire();

    // Fire onDeviceReady event once page has fully loaded, all
    // constructors have run and cordova info has been received from native
    // side.
    channel.join(function () {
        require('cordova').fireDocumentEvent('deviceready');
    }, channel.deviceReadyChannelsArray);

}, platformInitChannelsArray);
});
// file: src/common/init_b.js
define("cordova/init_b", function(require, exports, module) {
var channel = require('cordova/channel');
var cordova = require('cordova');
var modulemapper = require('cordova/modulemapper');
var platform = require('cordova/platform');
var pluginloader = require('cordova/pluginloader');
var utils = require('cordova/utils');
var platformInitChannelsArray = [channel.onDOMContentLoaded, channel.onNativeReady,
channel.onPluginsReady];
// setting exec
cordova.exec = require('cordova/exec');
function logUnfiredChannels (arr) {
    for (var i = 0; i < arr.length; ++i) {
        if (arr[i].state !== 2) {
            console.log('Channel not fired: ' + arr[i].type);
        }
    }
}
}

window.setTimeout(function () {
    if (channel.onDeviceReady.state !== 2) {
        console.log('deviceready has not fired after 5 seconds.');
```

```

        logUnfiredChannels(channel.deviceReadyChannelsArray);
    }
}, 5000);
// Replace navigator before any modules are required(), to ensure it happens as soon as
// possible.
// We replace it so that properties that can't be clobbered can instead be overridden.
function replaceNavigator (origNavigator) {
    var CordovaNavigator = function () {};
    CordovaNavigator.prototype = origNavigator;
    var newNavigator = new CordovaNavigator();
    // This work-around really only applies to new APIs that are newer than
    Function.bind.
    // Without it, APIs such as getGamepads() break.
    if (CordovaNavigator.bind) {
        for (var key in origNavigator) {
            if (typeof origNavigator[key] === 'function') {
                newNavigator[key] = origNavigator[key].bind(origNavigator);
            } else {
                (function (k) {
                    utils.defineGetterSetter(newNavigator, key, function () {
                        return origNavigator[k];
                    });
                })(key);
            }
        }
    }
    return newNavigator;
}
if (window.navigator) {
    window.navigator = replaceNavigator(window.navigator);
}
if (!window.console) {
    window.console = {
        log: function () {}
    };
}
if (!window.console.warn) {
    window.console.warn = function (msg) {
        this.log('warn: ' + msg);
    };
}
// Register pause, resume and deviceready channels as events on document.
channel.onPause = cordova.addDocumentEventHandler('pause');
channel.onResume = cordova.addDocumentEventHandler('resume');
channel.onActivated = cordova.addDocumentEventHandler('activated');
channel.onDeviceReady = cordova.addStickyDocumentEventHandler('deviceready');

// Listen for DOMContentLoaded and notify our channel subscribers.
if (document.readyState === 'complete' || document.readyState === 'interactive') {
    channel.onDOMContentLoaded.fire();
} else {
    document.addEventListener('DOMContentLoaded', function () {
        channel.onDOMContentLoaded.fire();
    }, false);
}
// _nativeReady is global variable that the native side can set
// to signify that the native code is ready. It is a global since
// it may be called before any cordova JS is ready.
if (window._nativeReady) {
    channel.onNativeReady.fire();
}
// Call the platform-specific initialization.
platform.bootstrap && platform.bootstrap();

```

```

// Wrap in a setTimeout to support the use-case of having plugin JS appended to
cordova.js.
// The delay allows the attached modules to be defined before the plugin loader looks
for them.
setTimeout(function () {
    pluginloader.load(function () {
        channel.onPluginsReady.fire();
    });
}, 0);

/**
 * Create all cordova objects once native side is ready.
 */
channel.join(function () {
    modulemapper.mapModules(window);

    platform.initialize && platform.initialize();

    // Fire event to notify that all objects are created
    channel.onCordovaReady.fire();

    // Fire onDeviceReady event once page has fully loaded, all
    // constructors have run and cordova info has been received from native
    // side.
    channel.join(function () {
        require('cordova').fireDocumentEvent('deviceready');
    }, channel.deviceReadyChannelsArray);

}, platformInitChannelsArray);
});
// file: src/common/modulemapper.js
define("cordova/modulemapper", function(require, exports, module) {
var builder = require('cordova/builder');
var moduleMap = define.moduleMap; // eslint-disable-line no-undef
var symbolList;
var deprecationMap;
exports.reset = function () {
    symbolList = [];
    deprecationMap = {};
};
function addEntry (strategy, moduleName, symbolPath, opt_deprecationMessage) {
    if (!(moduleName in moduleMap)) {
        throw new Error('Module ' + moduleName + ' does not exist.');
```

```

function prepareNamespace (symbolPath, context) {
  if (!symbolPath) {
    return context;
  }
  var parts = symbolPath.split('.');
  var cur = context;
  for (var i = 0, part; part = parts[i]; ++i) { // eslint-disable-line no-cond-assign
    cur = cur[part] = cur[part] || {};
  }
  return cur;
}

exports.mapModules = function (context) {
  var origSymbols = {};
  context.CDV_origSymbols = origSymbols;
  for (var i = 0, len = symbolList.length; i < len; i += 3) {
    var strategy = symbolList[i];
    var moduleName = symbolList[i + 1];
    var module = require(moduleName);
    if (strategy === 'r') {
      continue;
    }
    var symbolPath = symbolList[i + 2];
    var lastDot = symbolPath.lastIndexOf('.');
    var namespace = symbolPath.substr(0, lastDot);
    var lastName = symbolPath.substr(lastDot + 1);
    var deprecationMsg = symbolPath in deprecationMap ? 'Access made to deprecated
symbol: ' + symbolPath + '. ' + deprecationMsg : null;
    var parentObj = prepareNamespace(namespace, context);
    var target = parentObj[lastName];

    if (strategy === 'm' && target) {
      builder.recursiveMerge(target, module);
    } else if ((strategy === 'd' && !target) || (strategy !== 'd')) {
      if (!(symbolPath in origSymbols)) {
        origSymbols[symbolPath] = target;
      }
      builder.assignOrWrapInDeprecateGetter(parentObj, lastName, module,
deprecationMsg);
    }
  }
};

exports.getOriginalSymbol = function (context, symbolPath) {
  var origSymbols = context.CDV_origSymbols;
  if (origSymbols && (symbolPath in origSymbols)) {
    return origSymbols[symbolPath];
  }
  var parts = symbolPath.split('.');
  var obj = context;
  for (var i = 0; i < parts.length; ++i) {
    obj = obj && obj[parts[i]];
  }
  return obj;
};

exports.reset();
});
// file: src/common/modulemapper_b.js
define("cordova/modulemapper_b", function(require, exports, module) {
var builder = require('cordova/builder');
var symbolList = [];
var deprecationMap;
exports.reset = function () {
  symbolList = [];
  deprecationMap = {};
};

```

```

    });
    function addEntry (strategy, moduleName, symbolPath, opt_deprecationMessage) {
        symbolList.push(strategy, moduleName, symbolPath);
        if (opt_deprecationMessage) {
            deprecationMap[symbolPath] = opt_deprecationMessage;
        }
    }
    // Note: Android 2.3 does have Function.bind().
    exports.clobbers = function (moduleName, symbolPath, opt_deprecationMessage) {
        addEntry('c', moduleName, symbolPath, opt_deprecationMessage);
    };
    exports.merges = function (moduleName, symbolPath, opt_deprecationMessage) {
        addEntry('m', moduleName, symbolPath, opt_deprecationMessage);
    };
    exports.defaults = function (moduleName, symbolPath, opt_deprecationMessage) {
        addEntry('d', moduleName, symbolPath, opt_deprecationMessage);
    };
    exports.runs = function (moduleName) {
        addEntry('r', moduleName, null);
    };
    function prepareNamespace (symbolPath, context) {
        if (!symbolPath) {
            return context;
        }
        var parts = symbolPath.split('.');
        var cur = context;
        for (var i = 0, part; part = parts[i]; ++i) { // eslint-disable-line no-cond-assign
            cur = cur[part] = cur[part] || {};
        }
        return cur;
    }
    exports.mapModules = function (context) {
        var origSymbols = {};
        context.CDV_origSymbols = origSymbols;
        for (var i = 0, len = symbolList.length; i < len; i += 3) {
            var strategy = symbolList[i];
            var moduleName = symbolList[i + 1];
            var module = require(moduleName);
            // <runs/>
            if (strategy === 'r') {
                continue;
            }
            var symbolPath = symbolList[i + 2];
            var lastDot = symbolPath.lastIndexOf('.');
            var namespace = symbolPath.substr(0, lastDot);
            var lastName = symbolPath.substr(lastDot + 1);

            var deprecationMsg = symbolPath in deprecationMap ? 'Access made to deprecated
symbol: ' + symbolPath + '. ' + deprecationMsg : null;
            var parentObj = prepareNamespace(namespace, context);
            var target = parentObj[lastName];

            if (strategy === 'm' && target) {
                builder.recursiveMerge(target, module);
            } else if ((strategy === 'd' && !target) || (strategy !== 'd')) {
                if (!(symbolPath in origSymbols)) {
                    origSymbols[symbolPath] = target;
                }
                builder.assignOrWrapInDeprecateGetter(parentObj, lastName, module,
deprecationMsg);
            }
        }
    };

```

```

exports.getOriginalSymbol = function (context, symbolPath) {
    var origSymbols = context.CDV_origSymbols;
    if (origSymbols && (symbolPath in origSymbols)) {
        return origSymbols[symbolPath];
    }
    var parts = symbolPath.split('.');
    var obj = context;
    for (var i = 0; i < parts.length; ++i) {
        obj = obj && obj[parts[i]];
    }
    return obj;
};
exports.reset();
});
// file: /Users/steveng/repo/cordova/cordova-android/cordova-js-src/platform.js
define("cordova/platform", function(require, exports, module) {
// The last resume event that was received that had the result of a plugin call.
var lastResumeEvent = null;
module.exports = {
    id: 'android',
    bootstrap: function() {
        var channel = require('cordova/channel'),
            cordova = require('cordova'),
            exec = require('cordova/exec'),
            modulemapper = require('cordova/modulemapper');

        // Get the shared secret needed to use the bridge.
        exec.init();

        // TODO: Extract this as a proper plugin.
        modulemapper.clobbers('cordova/plugin/android/app', 'navigator.app');

        var APP_PLUGIN_NAME = Number(cordova.platformVersion.split('.')[0]) >= 4 ?
        'CoreAndroid' : 'App';
        var backButtonChannel = cordova.addDocumentEventHandler('backbutton');
        backButtonChannel.onHasSubscribersChange = function() {

            exec(null, null, APP_PLUGIN_NAME, "overrideBackbutton", [this.numHandlers ==
1]);
        };
        cordova.addDocumentEventHandler('menubutton');
        cordova.addDocumentEventHandler('searchbutton');

        function bindButtonChannel(buttonName) {
            var volumeButtonChannel = cordova.addDocumentEventHandler(buttonName +
        'button');
            volumeButtonChannel.onHasSubscribersChange = function() {
                exec(null, null, APP_PLUGIN_NAME, "overrideButton", [buttonName,
this.numHandlers == 1]);
            };
        }
        bindButtonChannel('volumeup');
        bindButtonChannel('volumedown');
        document.addEventListener = function(evt, handler, capture) {
            cordovaAddEventListener(evt, handler, capture);

            if (evt === 'resume' && lastResumeEvent) {
                handler(lastResumeEvent);
            }
        };
        channel.onCordovaReady.subscribe(function() {
            exec(onMessageFromNative, null, APP_PLUGIN_NAME, 'messageChannel', []);
            exec(null, null, APP_PLUGIN_NAME, "show", []);
        });
    }
};

```

```

    });
  }
};
function onMessageFromNative(msg) {
  var cordova = require('cordova');
  var action = msg.action;
  switch (action)
  {
    // Button events
    case 'backbutton':
    case 'menubutton':
    case 'searchbutton':
    // App life cycle events
    case 'pause':
    // Volume events
    case 'volumedownbutton':
    case 'volumeupbutton':
      cordova.fireDocumentEvent(action);
      break;
    case 'resume':
      if(arguments.length > 1 && msg.pendingResult) {
        if(arguments.length === 2) {
          msg.pendingResult.result = arguments[1];
        } else {
          // The plugin returned a multipart message
          var res = [];
          for(var i = 1; i < arguments.length; i++) {
            res.push(arguments[i]);
          }
          msg.pendingResult.result = res;
        }
        lastResumeEvent = msg;
      }
      cordova.fireDocumentEvent(action, msg);
      break;
    default:
      throw new Error('Unknown event action ' + action);
  }
}
});
// file: /Users/steveng/repo/cordova/cordova-android/cordova-js-
src/plugin/android/app.js
define("cordova/plugin/android/app", function(require, exports, module) {
var exec = require('cordova/exec');
var APP_PLUGIN_NAME = Number(require('cordova').platformVersion.split('.')[0]) >= 4 ?
'CoreAndroid' : 'App';
module.exports = {
  /**
   * Clear the resource cache.
   */
  clearCache:function() {
    exec(null, null, APP_PLUGIN_NAME, "clearCache", []);
  },
  loadUrl:function(url, props) {
    exec(null, null, APP_PLUGIN_NAME, "loadUrl", [url, props]);
  },
  cancelLoadUrl:function() {
    exec(null, null, APP_PLUGIN_NAME, "cancelLoadUrl", []);
  },
  clearHistory:function() {
    exec(null, null, APP_PLUGIN_NAME, "clearHistory", []);
  },
  backHistory:function() {
    exec(null, null, APP_PLUGIN_NAME, "backHistory", []);
  }
};

```

```

    },
    overrideBackbutton:function(override) {
        exec(null, null, APP_PLUGIN_NAME, "overrideBackbutton", [override]);
    },
    overrideButton:function(button, override) {
        exec(null, null, APP_PLUGIN_NAME, "overrideButton", [button, override]);
    },

    /**
     * Exit and terminate the application.
     */
    exitApp:function() {
        return exec(null, null, APP_PLUGIN_NAME, "exitApp", []);
    }
};
});
// file: src/common/pluginloader.js
define("cordova/pluginloader", function(require, exports, module) {
var modulemapper = require('cordova/modulemapper');
// Helper function to inject a <script> tag.
// Exported for testing.
exports.injectScript = function (url, onload, onerror) {
    var script = document.createElement('script');
    // onload fires even when script fails loads with an error.
    script.onload = onload;
    // onerror fires for malformed URLs.
    script.onerror = onerror;
    script.src = url;
    document.head.appendChild(script);
};
function injectIfNecessary (id, url, onload, onerror) {
    onerror = onerror || onload;
    if (id in define.moduleMap) { // eslint-disable-line no-undef
        onload();
    } else {
        exports.injectScript(url, function () {
            if (id in define.moduleMap) { // eslint-disable-line no-undef
                onload();
            } else {
                onerror();
            }
        }, onerror);
    }
}
}
function onScriptLoadingComplete (moduleList, finishPluginLoading) {
    for (var i = 0, module; module = moduleList[i]; i++) { // eslint-disable-line no-cond-assign
        if (module.clobbers && module.clobbers.length) {
            for (var j = 0; j < module.clobbers.length; j++) {
                modulemapper.clobbers(module.id, module.clobbers[j]);
            }
        }
        if (module.merges && module.merges.length) {
            for (var k = 0; k < module.merges.length; k++) {
                modulemapper.merges(module.id, module.merges[k]);
            }
        }
        if (module.runs) {
            modulemapper.runs(module.id);
        }
    }

    finishPluginLoading();
}
}

```

```

function handlePluginsObject (path, moduleList, finishPluginLoading) {    var
scriptCounter = moduleList.length;
    if (!scriptCounter) {
        finishPluginLoading();
        return;
    }
    function scriptLoadedCallback () {
        if (--scriptCounter) {
            onScriptLoadingComplete(moduleList, finishPluginLoading);
        }
    }
    for (var i = 0; i < moduleList.length; i++) {
        injectIfNecessary(moduleList[i].id, path + moduleList[i].file,
scriptLoadedCallback);
    }
}
function findCordovaPath () {
    var path = null;
    var scripts = document.getElementsByTagName('script');
    var term = '/cordova.js';
    for (var n = scripts.length - 1; n > -1; n--) {
        var src = scripts[n].src.replace(/\?.*$/, ''); // Strip any query param (CB-
6007).
        if (src.indexOf(term) === (src.length - term.length)) {
            path = src.substring(0, src.length - term.length) + '/';
            break;
        }
    }
    return path;
}
exports.load = function (callback) {
    var pathPrefix = findCordovaPath();
    if (pathPrefix === null) {
        console.log('Could not find cordova.js script tag. Plugin loading may fail.');
```

pathPrefix = '';

```

    }
    injectIfNecessary('cordova/plugin_list', pathPrefix + 'cordova_plugins.js', function
() {
        var moduleList = require('cordova/plugin_list');
        handlePluginsObject(pathPrefix, moduleList, callback);
    }, callback);
};
});
// file: src/common/pluginloader_b.js
define("cordova/pluginloader_b", function(require, exports, module) {
var modulemapper = require('cordova/modulemapper');
function handlePluginsObject (moduleList) {
    // if moduleList is not defined or empty, we've nothing to do
    if (!moduleList || !moduleList.length) {
        return;
    }
    for (var i = 0, module; module = moduleList[i]; i++) { // eslint-disable-line
no-cond-assign
        if (module.clobbers && module.clobbers.length) {
            for (var j = 0; j < module.clobbers.length; j++) {
                modulemapper.clobbers(module.id, module.clobbers[j]);
            }
        }
        if (module.merges && module.merges.length) {
            for (var k = 0; k < module.merges.length; k++) {
                modulemapper.merges(module.id, module.merges[k]);
            }
        }
        if (module.runs) {
            modulemapper.runs(module.id);
        }
    }
}

```

```

    }
  }
}
exports.load = function (callback) {
  var moduleList = require('cordova/plugin_list');
  handlePluginsObject(moduleList);

  callback();
};
});
// file: src/common/urlutil.js
define("cordova/urlutil", function(require, exports, module) {
exports.makeAbsolute = function makeAbsolute (url) {
  var anchorEl = document.createElement('a');
  anchorEl.href = url;
  return anchorEl.href;
};
});
// file: src/common/utils.js
define("cordova/utils", function(require, exports, module) {
var utils = exports;
utils.defineGetterSetter = function (obj, key, getFunc, opt_setFunc) {
  if (Object.defineProperty) {
    var desc = {
      get: getFunc,
      configurable: true
    };
    if (opt_setFunc) {
      desc.set = opt_setFunc;
    }
    Object.defineProperty(obj, key, desc);
  } else {
    obj.__defineGetter__(key, getFunc);
    if (opt_setFunc) {
      obj.__defineSetter__(key, opt_setFunc);
    }
  }
};
utils.defineGetter = utils.defineGetterSetter;
utils.indexOf = function (a, item) {
  if (a.indexOf) {
    return a.indexOf(item);
  }
  var len = a.length;
  for (var i = 0; i < len; ++i) {
    if (a[i] === item) {
      return i;
    }
  }
  return -1;
};
utils.arrayRemove = function (a, item) {
  var index = utils.indexOf(a, item);
  if (index !== -1) {
    a.splice(index, 1);
  }
  return index !== -1;
};
utils.typeName = function (val) {
  return Object.prototype.toString.call(val).slice(8, -1);
};
utils.isArray = Array.isArray ||
  function (a) { return utils.typeName(a) === 'Array'; };

```

```

utils.isDate = function (d) {
    return (d instanceof Date);
};
utils.clone = function (obj) {
    if (!obj || typeof obj === 'function' || utils.isDate(obj) || typeof obj !==
'object') {
        return obj;
    }
    var retVal, i;
    if (utils.isArray(obj)) {
        retVal = [];
        for (i = 0; i < obj.length; ++i) {
            retVal.push(utils.clone(obj[i]));
        }
        return retVal;
    }
    retVal = {};
    for (i in obj) {
        in
        interface supported" exception
            if ((!(i in retVal) || retVal[i] !== obj[i]) && typeof obj[i] !== 'undefined' &&
typeof obj[i] !== 'unknown') { // eslint-disable-line valid-typeof
                retVal[i] = utils.clone(obj[i]);
            }
        }
        return retVal;
    }
};
utils.close = function (context, func, params) {
    return function () {
        var args = params || arguments;
        return func.apply(context, args);
    };
};
function UUIDcreatePart (length) {
    var uuidpart = '';
    for (var i = 0; i < length; i++) {
        var uuidchar = parseInt((Math.random() * 256), 10).toString(16);
        if (uuidchar.length === 1) {
            uuidchar = '0' + uuidchar;
        }
        uuidpart += uuidchar;
    }
    return uuidpart;
}
utils.createUUID = function () {
    return UUIDcreatePart(4) + '-' +
        UUIDcreatePart(2) + '-' +
        UUIDcreatePart(2) + '-' +
        UUIDcreatePart(2) + '-' +
        UUIDcreatePart(6);
};
utils.extend = (function () {
    // proxy used to establish prototype chain
    var F = function () {};
    // extend Child from Parent
    return function (Child, Parent) {

        F.prototype = Parent.prototype;
        Child.prototype = new F();
        Child.__super__ = Parent.prototype;
        Child.prototype.constructor = Child;
    };
});

```

```
}());  
utils.alert = function (msg) {  
  if (window.alert) {  
    window.alert(msg);  
  } else if (console && console.log) {  
    console.log(msg);}  
}});  
window.cordova = require('cordova');  
// file: src/scripts/bootstrap.js  
require('cordova/init');
```