РЕФЕРАТ

БГУИР ДП 1-39 02 033 ПЗ

Козинец, Г.Ю. *Android*-клиент для веб-сервиса *reddit.com* : пояснительная записка к дипломному проекту / Г.Ю. Козинец. – Минск: БГУИР, 2019. – 77 с.

Пояснительная записка 77 с., 33 рис., 7 табл., 31 источник, 3 приложения

ANDROID-КЛИЕНТ, ВЕБ-СЕРВИС REDDIT.COM, МОБИЛЬНОЕ ПРОГРАММНОЕ СРЕДСТВО, УДАЛЁННЫЙ ДОСТУП К ВЕБ-СЕРВИСУ, REDDIT.COM, ПРОГРАММНОЕ СРЕДСТВО ДЛЯ СИСТЕМЫ ANDROID, СОЦИАЛЬНАЯ СЕТЬ, ПРОГРАММА ДЛЯ МОБИЛЬНОГО УСТРОЙСТВА.

Цель проектирования: создание эффективного средства взаимодействия пользователя с веб-сервисов путём предоставления удобного и быстрого доступа к необходимым функциям используя ресурсы смартфона пользователя.

Задачи проектирования: разработка *Android*-приложения с необходимым функционалом и удобным интерфейсом, позволяющим взаимодействовать с сервисом, просматривать размещаемую на нём информацию и взаимодействовать с ней.

Методология проведения работы: после проведения анализа необходимых требований в качестве операционной системы для разработки приложения была выбрана ОС *Android*, для реализации приложения была выбрана архитектура *MVC*, в качестве методологии пользовательского интерфейса был выбран *Material Design*, для реализации основного функционала были использованы следующие инструменты: *Realm*, *ButterKnife*, *Retrofit*.

Результаты работы: произведён анализ предметной области, изучены существующие реализации приложений и проведён анализ возможных недостатков. На основе полученных данных был разработан интерфейс и способы взаимодействия с программой, и алгоритмы работы. На основе подготовленных данных была разработана программная часть средства. Было разработано руководства пользования и произведено технико-экономическое обоснование разработки веб-сервиса.

Область применения результатов: Программа может быть установлено на устройство с операционной системой *Android* версии не младше 6.0 после установки пользователь может сразу приступать к использованию программного средства.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире социальные сети являются неотъемлемой частью жизни для большой части населения. По сути, они являются одними из наиболее популярных и посещаемых ресурсов в интернете – люди проводят значительную часть своего времени именно на этих веб-сайтах. Такая популярность обусловлена большим разнообразием возможных реализаций и простотой получения желаемой информации в режиме реального времени.

При первом рассмотрении социальных сетей, может сложится впечатление о том, что существование всех социальных сетей направлено на распространение развлекательного контента, однако это не так. В интернете существует большое количество специализированных ресурсов, которые могут быть направлены на создание различных сообществ. В их число входят веб-сайты для поиска работы, размещения художественных произведений, решения проблем и многого другого. На всех перечисленных ресурсах люди могут делится опытом, публиковать свои работы или оказывать помощь в достижении определённой цели. Также существуют варианты, которые не имеют строгой направленности и могут содержать в себе элементы социальных сетей перечисленных ранее.

Так как взаимодействие между людьми в интернете может происходить в реальном времени, становится актуальным вопрос об использовании веб-сервиса, на котором размещается социальная сеть, вдали от дома или персонального компьютера. Для решения этой проблемы многие пользователи используют смартфоны у которых есть беспроводной доступ к сети Интернет. Для повышения удобства использования веб-сервиса, многие социальные сети имеют специальные приложения для мобильных устройств, которые в значительной степени упрощают взаимодействие с функционалом веб-сайта через экран смартфона.

Объектом исследования являются существующие варианты приложений для мобильных устройств, позволяющие использовать основные функции веб-сервисов.

Предметом исследования являются способы улучшения взаимодействия пользователя с веб-сайтом через программное средство.

Целью данной работы стала разработка мобильного приложения для операционной системы *Android*, которое позволило бы взаимодействовать с основным функционалом веб-сервиса через экран смартфона и предоставляло более удобный интерфейс, чем веб-страница выбранной социальной сети, открытая во встроенном браузере *Android*-смартфона.

Поставленная цель потребовала решения следующих задач:

– исследовать процессы и способы взаимодействия пользователей с различными вариантами приложений;

– изучить существующие приложения для мобильных устройств;

– разработка алгоритмов функционирования программного средства;

– определение функциональных возможностей программного средства;

– разработка пользовательского интерфейса программного средства;

– разработать программное средство;

– произвести технико–экономическое обоснование разработанного программного средства.

Для достижения поставленной цели высокоуровневый язык программирования *Java,* официальный набор инструментов для разработки приложений под операционную систему *Google – Android Studio* и ряд сторонних библиотек для реализации дополнительного функционала.

Аналитики и разработчики ПО прогнозируют продолжение развития рынка социальных сетей в ближайшие годы, как в области решений, так и в области использования. Для пользователей важны в первую очередь простой и привычный интерфейс, простота и доступность сервисов. От того, насколько эффективно система будет улучшать взаимодействие пользователей, зависит коммерческий успех продукта.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет 76,45%. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников».

Скриншот приведён в приложении А.

1 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И ПОСТАНОВКА ЗАДАЧ

1.1 Анализ и описание функциональных возможностей программного средства

Существующие социальные сети обладают большой вариативностью размещаемого на них контента, однако основной функционал веб-сервиса, на котором размещается социальная сеть, является аналогичным для большинства из них. Большинство мобильных приложений, реализующих способ взаимодействия с веб-сервисов, имеют следующий функционал:

– просмотр размещённых публикаций;

– оценка публикаций;

– просмотр комментариев, оставленных другими пользователями;

– возможность оставить собственный комментарий;

– просмотр профиля пользователя;

– размещение публикаций.

В нашем случаем будет производится разработка мобильного приложения под операционную систему *Android* для веб-сервиса *reddit.com*. Данный веб-сервис является одной из самых популярных социальных сетей в Интернете и объединяет пользователей из стран по всему миру. Имея такую большую базу пользователей, на сайте размещается огромное количество различной информации. Чтобы информация разной тематики не смешивалась в одном месте, на сайте были придуманы потоки. Каждый поток имеет свою тематику, правила размещения публикаций и свои правила общения пользователей в разделе комментариев.

Такой способ позволяет каждому пользователи просматривать только выбранные им потоки, и не засорять ленту нежелательными публикациями.

Учитывая существующие аналоги мобильных приложений и особенности веб-сервиса, под который будет разрабатываться наше приложение нам необходимо включить в него следующий функционал:

– просмотр публикаций из выбранного потока;

– просмотр комментариев к выбранной публикации;

– возможность оставить собственный комментарий к выбранной публикации;

– просмотр рейтинга публикаций;

– возможность оценить публикацию.

В перечисленный функционал также входит создание удобного пользовательского интерфейса, который будет использовать преимущества сенсор-

ного экрана смартфона, так как большое количество пользователей хотят пользоваться данным веб-сервисов вне дома и в дали от персонального компьютера, а дизайн веб-страницы данной социальной сети плохо подходит для использования на экране мобильного устройства, появляется острая необходимость с создании программного средства под мобильные устройства на операционной системе *Android*.

1.2 Обоснование выбора языка программирования и средств разработки

При разработке сложных программных средств под мобильные операционные системы следует с особым вниманием подходить к выбору средств разработки, так как от этого зависит скорость и качество процесса создания программного средства.

Для выбора подходящей среды разработки необходимо учесть ряд определённых факторов, которые могут оказать сильное влияние на разработку:

– интерфейс. Этот первый компонент, с которым сталкивается пользователь после установки и который формирует первое впечатление о программе, и, на основании которого, может быть осуществлен окончательный выбор. Здесь оценивается не только общий дизайн, хотя, безусловно, он тоже сказывается определенным образом, но и удобство расположения и настройки таких компонент как окно исходного кода, окно проекта;

– настройка. Соответственно после установки и первого запуска среды разработки выполняется ее настройка, т.е. указываются пути, по которым располагаются установленные *SDK, DOCS, J2EE*. В этом компоненте, как правило, существенных различий не наблюдается. Более того, современные средства, как правило, самостоятельно определяют установленные компоненты;

– редактор кода. Настройка отображения исходных кодов, как правило, тоже не отличается разнообразием, в любом средстве легко можно настроить кеглю и ее размер, а также цвет. Немаловажным преимуществом является наличие помощника, когда, при «зависании» мышки на любой переменной или методе всплывает довольно подробный *ToolTip* (контекстное окно) в котором развернуто, описаны все параметры объекта. Также есть масса приятных мелочей, вывод нумерации строк, отображение структуры класса, показ символов абзаца, проверка орфографии.

В настоящее время хорошо зарекомендовавшим себя на рынке средств разработки под систему *Android* является *Android Studio*.

Android Studio это официальная интегрированная среда разработки (*IDE*) для разработки *Android*-приложений, основанная среде разработки *IntelliJ IDEA* [1].

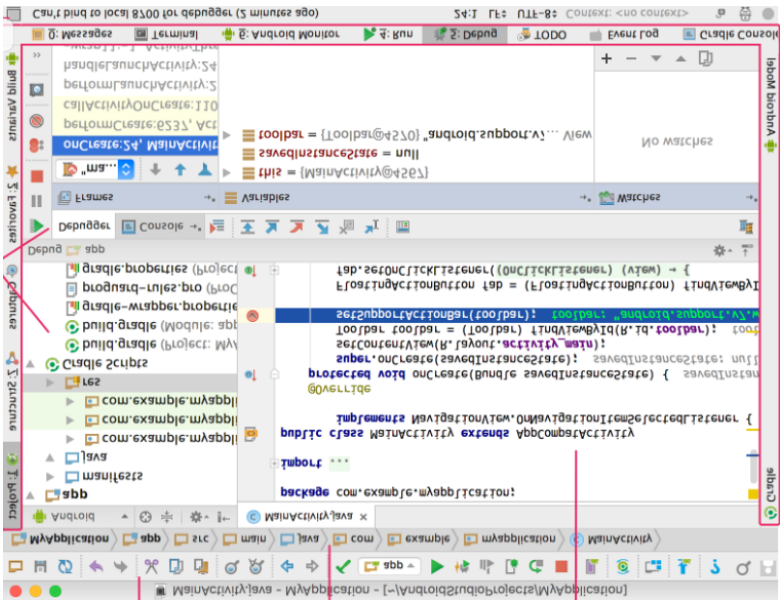

Рисунок 1.1 – Интерфейс среды разработки *Android Studio*[2]

Android Studio находилась в свободном доступе начиная с версии 0.1, опубликованной в мае 2013, а затем перешла в стадию бета-тестирования, начиная с версии 0.8, которая была выпущена в июне 2014 года. Первой стабильной версией была 1.0, выпущенная в 2014 году, сразу после этого прекратилась поддержка плагина *Android Development Tools* (*ADT*) для *IDE Eclipse*. Среда основана на программном обеспечении *IntelliJ IDEA* от компании *JetBrains*, одном из самым популярных сред разработки на языке *Java*. Данная среда разработки доступна для *Windows*, *OS X*, *Linux*. Интерфейс среды разработки представлен на рисунке 1.1.

Для *Android Studio* регулярно выходят обновления, улучшающие стабильность программы и её функционал. Вот список основного функционала доступного на данный момент:

– гибкая система сборки приложений на *Gradle*;
– быстрый и многофункциональный эмулятор *Android*;
– единое окружение для разработки под любые *Android*-устройства;
– возможность вносить изменения в запущенное приложение;
– шаблоны и интеграция с системой сборок *GitHub*;
– обширные инструменты для тестирования;
– средства отслеживания проблем совместимости, производительности;
– поддержка *C++* и *NDK*;
– встроенная поддержка для *Google Cloud Platform* [3].

К обязательным инструменты среды разработки относятся *Android SDK*, а также инструменты, входящие в состав *Android SDK*:

– *SDK Manager* - это инструмент, позволяющий загрузить компоненты *Android SDK*. Показывает пакеты *Android SDK* и их статус: установлен

(*Installed*), не установлен (*Not Installed*), доступны обновления (*Update available*);

– *Android Emulator* (*emulator*) - это виртуальное мобильное устройство, которое создается и работает на компьютере разработчика, используется для разработки и тестирования мобильных приложений без привлечения реальных устройств;

– *AVD Manager* предоставляет графический интерфейс для создания виртуальных *Android* устройств (*AVDs*), предусмотренных *Android Emulator*, и управления ими;

– *Android Debug Bridge* (*adb*) - это гибкий инструмент, позволяющий управлять состоянием эмулятора или реального *Android* устройства, подключенного к компьютеру. Также может использоваться для установки *Android* приложения на реальное устройство.

Для разработки приложений для *Android* существует несколько языков программирования, одним из них является кроссплатформенный язык *Java*.

Объектно-ориентированный язык *Java*, разработанный в *Sun Microsystems*, предназначен для создания переносимых на различные платформы и операционные системы программ. Язык *Java* нашел широкое применение в Интернет-приложениях, добавив на статические и клиентские *Web*-страницы динамическую графику, улучшив интерфейсы и реализовав вычислительные возможности. Но объектно-ориентированная парадигма и кроссплатформенность привели к тому, что уже буквально через несколько лет после своего создания язык практически покинул клиентские страницы и перебрался на сервера [4]. Программы на *Java* транслируются в байт-код *Java*, выполняемый виртуальной машиной *Java* (*JVM*) – программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор. Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять *Java*-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии *Java* является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Язык *Java* активно используется для создания мобильных приложений под операционную систему *Android*. При этом программы компилируются в нестандартный байт-код, для использования их виртуальной машиной *Dalvik* (начиная с *Android* 5.0 *Lollipop* виртуальная машина заменена на *ART*). Для такой компиляции используется дополнительный инструмент, а

именно *Android SDK* (*Software Development Kit*), разработанный компанией *Google*.

1.3 Обзор существующих программных средств по теме дипломного проекта

В настоящее время в сети Интернет наиболее востребовано всего около пяти продуктов.

Linked In

Интерфейс веб-сервиса изображён на рисунке 1.2.

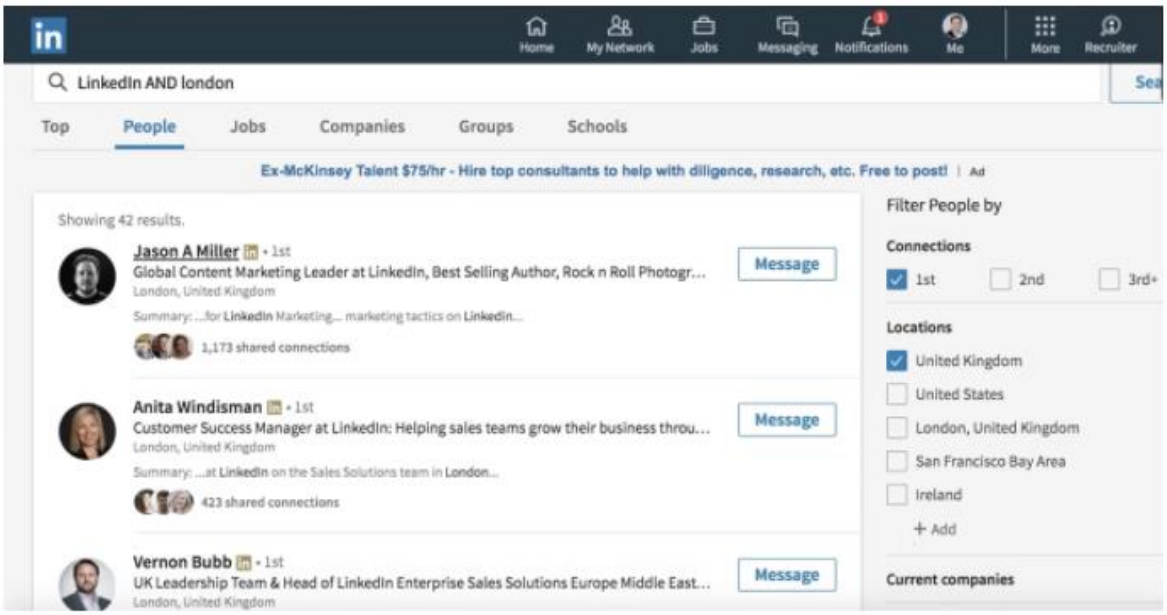

Рисунок 1.2 – Интерфейс сайта «*Linked In*»[5]

Эта социальная сеть является инструментом для создания собственного резюме и поиска работы. Данный веб-сервис может сам подбирать подходящих работодателей и вакансии на основе навыков указанных в вашем резюме.

Эта социальная сеть предназначена для общения профессионалов, а не для развлечения. Это и отличает её от других соцсетей, таких как «*Facebook*», «Одноклассники», «ВКонтакте». Здесь можно наладить деловые связи с коллегами, найти вакансии. В ней *HR*-менеджерам удобно находить специалистов для привлечения, а пользователям – быть в курсе новостей, касающихся профессиональных вопросов. Бизнес-страницы компаний, со статьями и новостями, тематическое общение, совместное решение сложных вопросов – всё это варианты использования этого портала[6].

Пользоваться данным веб-сервисом можно на бесплатной основе.

Мобильное приложение для данной социальной сети умеет удобный и понятный интерфейс, который позволяет пользоваться основным функционалом сайта.

Интерфейс приложения изображён на рисунке 1.3.

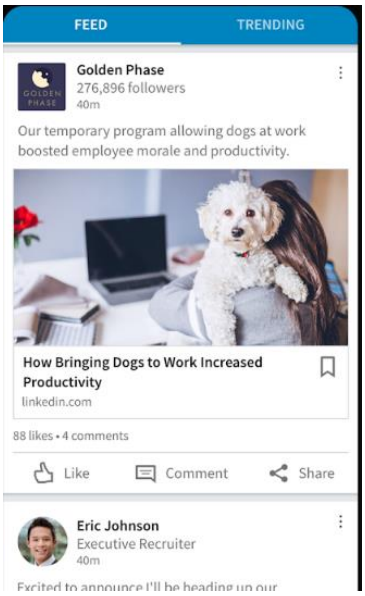

Рисунок 1.3 – Интерфейс приложения «*Linked In*»

По внешнему виду интерфейса можно заметить, что приложение имеет приятный пользовательский интерфейс на котором в понятной форме представлен доступный функционал.

Pikabu

Интерфейс веб-сервиса изображён на рисунке 1.4.

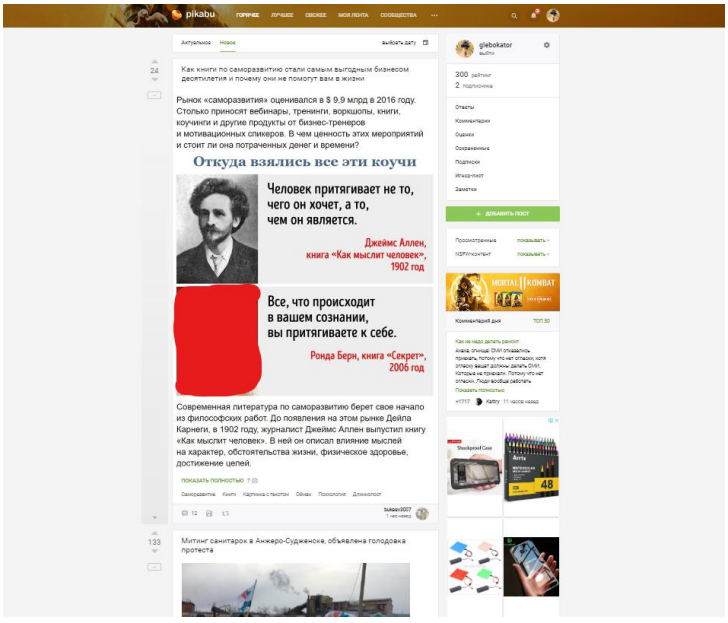

Рисунок 1.4 – Интерфейс сайта «*Pikabu*»

Пикабу является российской разработкой, которая обрела большую популярность благодаря большому количеству качественных публикаций и удобному интерфейсу. Одной из мер фильтрации размещаемого контента является система для определения уникальности этого контента и, если на сайте будут найдены публикации с одинаковым содержанием, пользователю будет предупреждён о том, что похожие публикации уже есть на сайте.

Мобильное приложение для данной социальной сети умеет удобный и понятный интерфейс, который позволяет пользоваться основным функционалом сайта.

Интерфейс приложения изображён на рисунке 1.5.

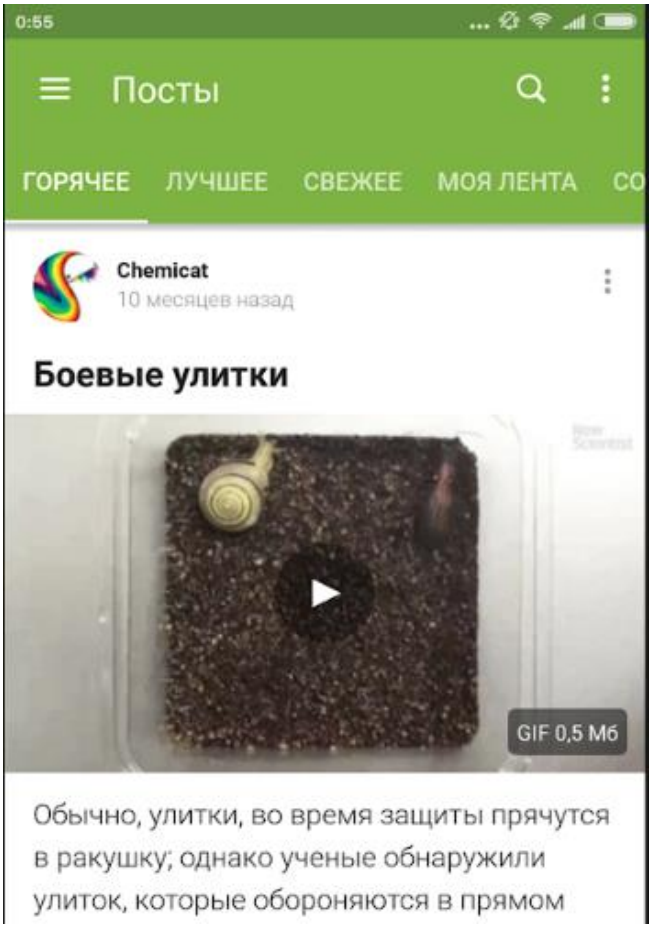

Рисунок 1.5 – Интерфейс приложения «*Pikabu*»

По внешнему виду интерфейса можно заметить, что приложение имеет приятный пользовательский интерфейс. Благодаря грамотно составленному интерфейсу, пользователь может получить представление о имеющемся функционале при первом же запуске программы, даже если ранее пользователь не ней не встречался.

Вконтакте

Внешний вид системы приведён на рисунке 1.6.

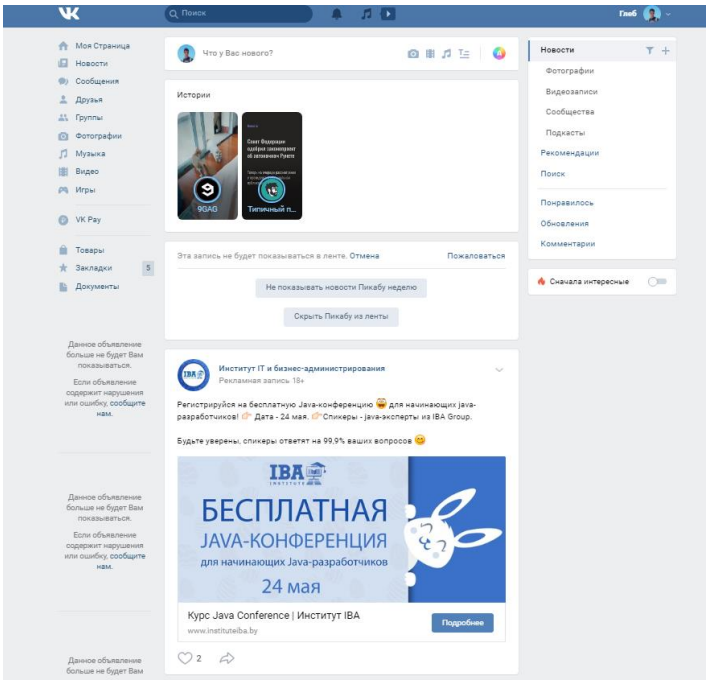

Рисунок 1.6 – Внешний вид сайта Вконтакте

Данная социальная сеть является крупнейшей в русскоязычном сегменте всемирной сети. Веб-сервис является типичным представителем социальных сетей и имеет огромный набор функций для обмена сообщениями, поиска пользователей, создания и просмотра публикаций.

Интерфейс приложения изображён на рисунке 1.7.

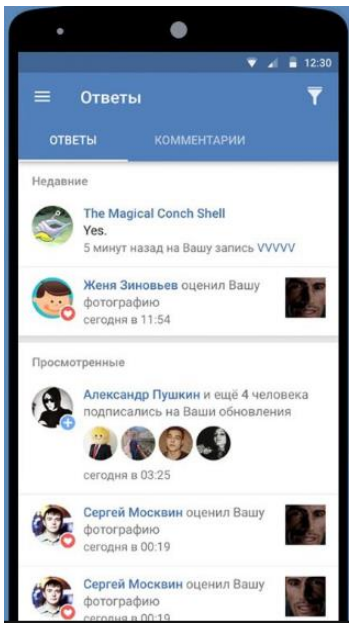

Рисунок 1.7 – Интерфейс приложения «*Vkontakte*»

Несмотря на первоначальную простоту интерфейса может показаться, что данный портал имеет скудный набор функций, однако такое мнения является ошибочным, так как многие функции скрыты от пользователя, для того чтобы не захламлять рабочее пространство. По мере сервиса скрытый функционал будет становиться доступным в зависимости от раздела сайта в котором находится пользователь, благодаря этому им не нужно запоминать расположение всех кнопок и полей ввода, так как веб-сайт будет сам направлять и подсказывать пользователю путь к необходимым функциям.

По внешнему виду интерфейса можно заметить, что приложение имеет приятный пользовательский интерфейс на котором в понятной форме представлен доступный функционал.

Twitter

Интерфейс системы изображён на рисунке 1.8.

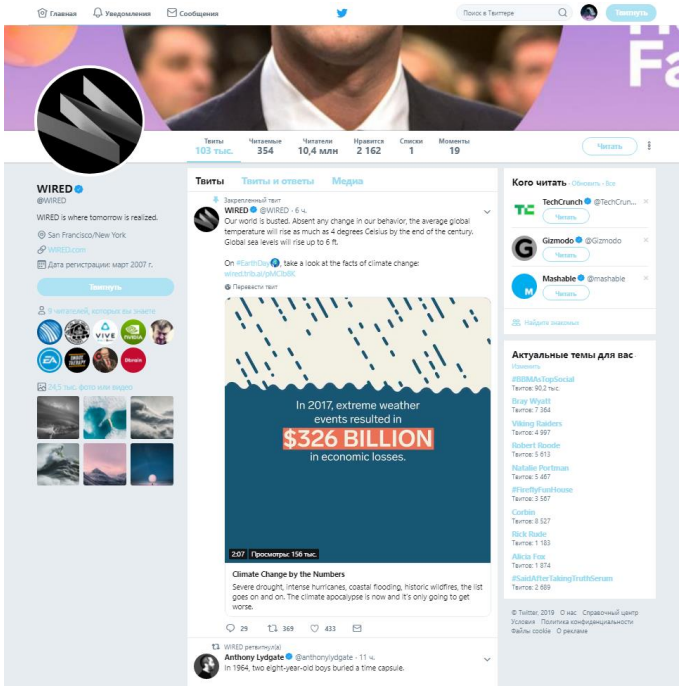

Рисунок 1.8 – Внешний вид системы *Twitter*

На данный момент эта социальная сеть является одним из самых влиятельных и эффективным инструментом распространения информации. Вебсервис получил большое распространение по всему миру и у многих влиятельных людей и компаний есть собственные официальные аккаунты, через которые они могут в удобной форме общаться и делиться новостями со всем миром [7].

Мобильное приложение для данной социальной сети умеет удобный и понятный интерфейс, который позволяет пользоваться основным функционалом сайта.

Интерфейс приложения изображён на рисунке 1.9.

Рисунок 1.9 – Интерфейс приложения «*Twitter*»

По внешнему виду интерфейса можно заметить, что приложение имеет темную тему оформления, которая позволяет использовать программное средство длительное время и при это пользователь не будет ощущать боль в глазах. Приятный пользовательский интерфейс имеет удобное расположение элементов на котором в понятной форме представлен доступный функционал.

Далеко не все существующие решения для мобильных приложений являются эффективными. Многие из них не дают удобного способа взаимодействия с веб-сервисом и предлагают неэффективные пути взаимодействия с функционалом приложения.

Довольно острой проблемой является постоянное разрастание и утяжеление приложений из-за увеличения реализованных функций. Данная проблема становится особенной острой, когда на мобильном устройстве установлено большое количество подобных программных средств, что может приводить к замедлению работы устройства и заметному снижению срока работы от аккумулятора до необходимости повторной его зарядки.

1.4 Постановка задач по разработке программного средства

Анализ предметной области и существующих решений показал, что большинство похожих продуктов имеют некоторые недостатки, например:

– перегруженность функциями и сложный интерфейс;

– специализация под классические «офисные» корпорации, например, сфера продаж, и отсутствие необходимых IT-специалистам особенностей;

– неудобные модели ценообразования, в результате которых продукт не подходит некоторым категориям клиентов.

Разрабатываемое приложение должно иметь удобный, интуитивно понятный интерфейс, в котором самые важные функции находятся в ближайшей доступности, а более глубокие возможности не должны перегружать пользовательский интерфейс. Боковое меню должно предоставлять пользователю в удобной форме перечень всех имеющихся категорий, по которым он может осуществлять навигацию.

Программный продукт будет распространяться через официальный сервис распространения приложения для операционной системы *Android*: *Google play*[8].

Google Play является сертифицированным сервисом, на котором размещаются только проверенные приложения от разработчиков со всего мира.

Использование официальных сервисов для распространения нашего программного средства гарантирует нам конкурентно способную среду, так как сервис будет рекламировать разработанное приложение на главной странице, что позволит большему количеству пользователей заметить наше приложение и в последствии произвести его установку на своё мобильное устройство.

Также в *Google Play* существует удобная система отзывов от приложении, что позволят разработчикам своевременно получать информацию о неисправностях или получать запросы о добавлении нового или изменении существующего функционала.

В результате сформирован следующий список требований к разрабатываемой системе:

– возможность авторизации в системе;

– возможность просмотра профиля пользователей;

– возможность просматривать публикации в выбранном тематическом потоке;

– возможность комментировать публикации;

– просмотр рейтинга публикации;

– возможность оценивать публикации и влиять на их рейтинг;

– просмотр существующих публикаций.

2 ОПИСАНИЕ РАЗРАБАТЫВАЕМОЙ СИСТЕМЫ

2.2 Обоснование выбора компонентов и технологий для реализации системы

Для реализации программного продукта был выбран следующий стек технологий:

– *Android SDK;*

– *Java SDK;*

– СУБД *Realm;*

– *ButterKnife;*

– *Retrofit;*

– *RestAPI.*

Рассмотрим каждую из технологий подробнее.

Для взаимодействия с серверной частью приложения будет использоваться технология *RestAPI*. *Android SDK* – универсальный фреймворк с открытым исходным кодом для *Java*-платформы. В качестве системы управления базами данных выбрана *Realm Database*. Для упрощения создания функционала для интерфейса будет использоваться библиотека *ButterKnife*. При работе с *JSON* данными и *HTTP* запросами будет использоваться фреймворк *Retrofit*.

Realm – датский проект основанный в 2011 году. Ранее проект назывался *tight.db*. За время существования привлечено 29 миллионов долларов инвестиций. Зарабатывать компания планирует на основе *Realm Mobile Platform*, сама же база данных бесплатная и с открытым исходным кодом. *Realm* под *Android* появился в 2014 году и с тех пор постоянно развивается [9]. В отличие от реляционных баз данных *Realm Database* предлагает *NoSQL* модель данных, благодаря чему *Realm* работает быстрее, обладает лучшей масштабируемостью. Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных.

Преимущества *Realm* баз данных:

– до 10 раз быстрее, чем *SQLite*;

– проста в использовании;

– удобно для создания и хранения данных на лету;

– хорошая документация и поддержка.

При создании функционала для большого количества элементов интерфейса или *activity* часто требуется большое количество кода для привязки функций к различным элемента интерфейса, это могут быть кнопки, диалоговые окна, списки и прочие объекты, с которыми может взаимодействовать

пользователь. Чтобы избежать эту проблему, будет использоваться библиотека *ButterKnife*.

Butter Knife – популярная библиотека для инициализации *Views*, разработанная Джейком Уортоном (*Square Inc*.). Данная библиотека использует аннотации, на основе которых генерируется повторяющийся код, которого мы избегаем. В спецификации написано, что это никак не влияет на производительность приложения, ведь генерация кода происходит как раз во время компиляции [10].

Retrofit – это *REST* клиент для *android* и *Java* от компании *Square*. Он может относительно легко получать и разбирать *JSON* или другие структуированные данные через веб-сервисы, использующие *REST*. В *Retrofit* для сериализации или десериализации данных используются конверторы, которые необходимо указывать вручную. Типичным конвертором для *JSON* формата является библиотека *GSon*. Для *HTTP* запросов *Retrofit* использует *OkHttp* библиотеку [11]. *Retrofit* имеет обширный функционал, достаточного для создания полноценного *REST*-клиента, который будет выполнять *POST, GET, PUT, DELETE* запросы. Тип и другие аспекты запроса обозначаются при помощи аннотаций. Например, для того, чтобы обозначить, что требуется *GET* запрос, нам нужно написать перед методом *GET*, для *POST* запроса *POST*. В скобках к типу запроса вводится необходимый веб-адрес.

Для отправки запросов и получения данных от веб-сервиса, нужно знать определённый список команд составленных в особом формате. Во время появления первых веб-сайтов не существовало определённого стандарта для взаимодействия, что сильно затрудняло создание программ для работы с ними. На сегодняшний день практически все веб-сайты и сервисы пользуются стандартом *REST API* [12].

REST – это общие принципы организации взаимодействия приложения или сайта с сервером посредством протокола *HTTP*. Особенность *REST* в том, что сервер не запоминает состояние пользователя между запросами - в каждом запросе передаётся информация, идентифицирующая пользователя (например, *token*, полученный через *OAuth*-авторизацию) и все параметры, необходимые для выполнения операции[13].

Наконец, в качестве окружения для разработки продукта была выбрана среда разработки *Android Studio*.

Android Studio – это расширяемая платформа для разработки *Android* приложений.

В состав *Android Studio* включены самые производительные инструменты для создания качественных и эффективных приложений для разных типов устройств *Android*, включая телефоны, планшеты, а также устройства

Android Auto, Android Wear и *Android TV*. Так как это официальная среда разработки от *Google*, в *Android Studio* есть все, что нужно для создания приложения: интеллектуальный редактор, отладчик, а также средства анализа характеристик, эмуляторы и многое другое.

Основные возможности:

– интеллектуальный редактор с расширенным автодополнением, рефакторингом и анализом кода;

– функция «Мгновенный запуск», позволяющая быстро проверять изменения, задавать параметры и запускать рабочие циклы путем введения кода и изменения ресурсов, доступных приложению, на устройстве или в эмуляторе;

– быстрый и многофункциональный эмулятор *Android* с виртуальным акселерометром, измерителем рабочей температуры, магнитометром и другими датчиками;

– поддержка всех платформ *Android*: телефонов и планшетов, а также устройств *Android Wear, Android Auto* и *Android TV*;

– гибкая система сборки на основе *Gradle* с автоматизацией процессов формирования кода приложений, управления зависимостями и настраиваемыми конфигурациями файлов *APK*;

– шаблоны кода для реализации стандартных функций;

– удобный редактор макетов с возможностью перетаскивания элементов и режимом прототипирования для разработки приложений на интуитивном уровне;

– новый менеджер ограничений макетов (обратно совместимый с *API Android* уровня 9) для разработки больших и сложных макетов в одноуровневой, упрощенной иерархии;

– анализаторы исходного кода для выявления в коде приложения проблем, связанных с производительностью, удобством использования, совместимостью версий и т. п;

– поддержка *C*/*C++* в режиме изменения кода и возможность отладки с использованием низкоуровневого набора команд (*LLDB*), что позволяет использовать в приложении компоненты интерфейса для прямого доступа из *Java*;

– встроенная поддержка *Firebase SDK, Firebase Test Lab, Firebase App Indexing* и *Google Cloud Platform*;

– анализатор *APK* для просмотра файлов *APK* и определения удельных долей отдельных компонентов приложения в его общем объеме;

– модуль записи тестов *Espresso* (бета версия) для создания тестов пользовательского интерфейса путем регистрации его взаимодействий с приложением, а также последующего вывода программного кода тестов[14];

– инспектор макета для просмотра иерархии представлений приложения во время его прогона;

– отладчик графического процессора (бета версия) для захвата потока команд *OpenGL ES* на устройстве *Android* и его запуска в *Android Studio* для последующего анализа [15].

После выбора технологий можно перейти к непосредственной разработке архитектуры программного средства.

2.2 Разработка алгоритмов функционирования программного средства

Система данного программного средства будет состоять из нескольких составных частей: веб-сервис, пользовательский интерфейс, обработчик *API*-запросов, обработчик *JSON*-данных и база данных. Для лучшего понимания работы приложения можно рассмотреть его работу в виде отдельных шагов. Рассмотрим алгоритм аутентификации пользователя для получения доступа к веб-сервису:

1 Для входа в свою учётную запись пользователю необходимо открыть боковое меню и добавить нового пользователя.

2 При нажатии кнопки добавления пользователя программное средство переводит пользователя на экран для ввода данных пользователя.

3 После ввода данных пользователь нажимает на кнопку подтверждения данных.

4 После программное средство отправляет запрос с данным пользователя на сервер веб-сервиса для валидации данных.

5 При успешном прохождении валидации веб-сервис отправляет пользователю сообщение об успешном прохождении аутентификации и специальный сессионный ключ.

6 После прохождения аутентификации программное средство переносит пользователя на главный экран.

Рассмотрим алгоритм отправки комментария к выбранной публикации:

Пользователь выбирает на главном экране желаемую публикацию.

1 Программа начинает загрузку содержимого публикации и оставленные к ней комментарии.

2 Пользователь нажимает кнопку для добавления комментария.

3 Программное средство проверяет наличие сессионного ключа, с помощью которого веб-сервис будет валидировать пользователя и отправленный им комментарий.

4 После того как пользователь ввёл текст комментария и нажал на кнопку отправки, программа отправит запрос на веб-сервис для добавления комментария к публикации.

5 После успешной отправки пользователю будет выведено сообщение о завершённой операции.

6 Далее программа обновляет список существующих комментариев.

Рассмотрим алгоритм загрузки списка публикаций:

1 При запуске программного средства на веб-сервис отправляется *API*-запрос для получения списка публикаций.

2 Обработчик *JSON*-данных получает набор данных для публикаций и переводит их в *Java*-объекты.

3 Классы, отвечающие за интерфейс программы, выводят полученные данные на экран устройства в соответствии с настройками вывода, сохраненными в базе данных программы.

4 При выборе определённой публикации, обработчик *JSON*-данных формирует *API*-запрос для получения содержимого публикации.

5 Во время получения данных о публикации класс отображает на экране устройства содержимое полученных и обработанных данных.

Рассмотрим подробнее пример такого алгоритма.

Для получения публикаций в выбранном разделе, клиент осуществляет *API*-запрос *Reddit.getHot*, который загружает публикации отсортированные по популярности, для других сортировок существуют приставки *New, Controvertial* и *Rising*.

Класс *RedditClient* вызовет метод *getSubredditPosts*(*String sorting, String baseUrl*), первый параметр метода отвечает за выбор типа сортировки, второй параметр указывает ссылку на раздел веб-сервиса. *RedditClient* используя класс-утилиту *RedditPostDeserializer,* обработает полученный *JSON*-запрос и с помощью библиотеки *Retrofit* переведёт данные в класс *PostItem*. Далее полученные и обработанные публикации передаются в класс *PostsListView*, который отвечает за отображение публикаций в виде списка, после этого содержимое *PostsListView* помещается в класс *HomeView* и передаётся в *MainActivity* для отображения публикаций на экране устройства. При нажатии на желаемую публикацию отправляется *GET*-запрос с ссылкой на данную публикацию, которая хранится в ранее созданной *PostItem*, далее клиент получает ответ от веб-сервиса в виде *JSON*-данных, которые обрабатываются классом *RedditPostContentDes* и сохраняются в виде объекта класса *PostContent*, который затем передаётся в класс *PostContentActivity* для отображения содержимого публикации на экране смартфона.

2.3 Проектирование архитектуры программного средства

Программный код дипломного проекта будет разрабатываться в архитектурном стиле *Model-View-Controller* (*MVC*, «Модель-Представление-Контроллер»).

MVC – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления, не знает, как визуализировать данные и контроллера, не имеет точек взаимодействия с пользователем, просто предоставляя доступ к данным и управлению ими. В роли модели будут выступать классы отвечающие за сущности приложения, а также за его бизнес логику.

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя. В роли представления будут выступать *XML* файлы разметки пользовательского интерфейса.

Контроллер обеспечивает связи между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия. В роли контроллеров будут выступать такие классы как *Activity, Fragment, Dialog, Navigation Drawer*.

Activity — это компонент приложения, который выдает экран, и с которым пользователи могут взаимодействовать для выполнения каких-либо действий, например набрать номер телефона, сделать фото, отправить письмо или просмотреть карту. Каждой операции присваивается окно для прорисовки соответствующего пользовательского интерфейса. Обычно окно отображается во весь экран, однако его размер может быть меньше, и оно может размещаться поверх других окон[16].

Главными контроллерами будут выступать активити. Вот список основных активити созданных в ходе разработки:

– *MainActivity*;
– *ThreadActivity*;
– *PostFragment*;
– *AddLesson*;
– *SubscriptionActivity*;
– *AuthActivity*.

Для эффективной работы системы *Android*, в ней были созданный жизненные циклы различных элементов программного средства.

У активити есть свой жизненный цикл, который показан на рисунке 2.1.

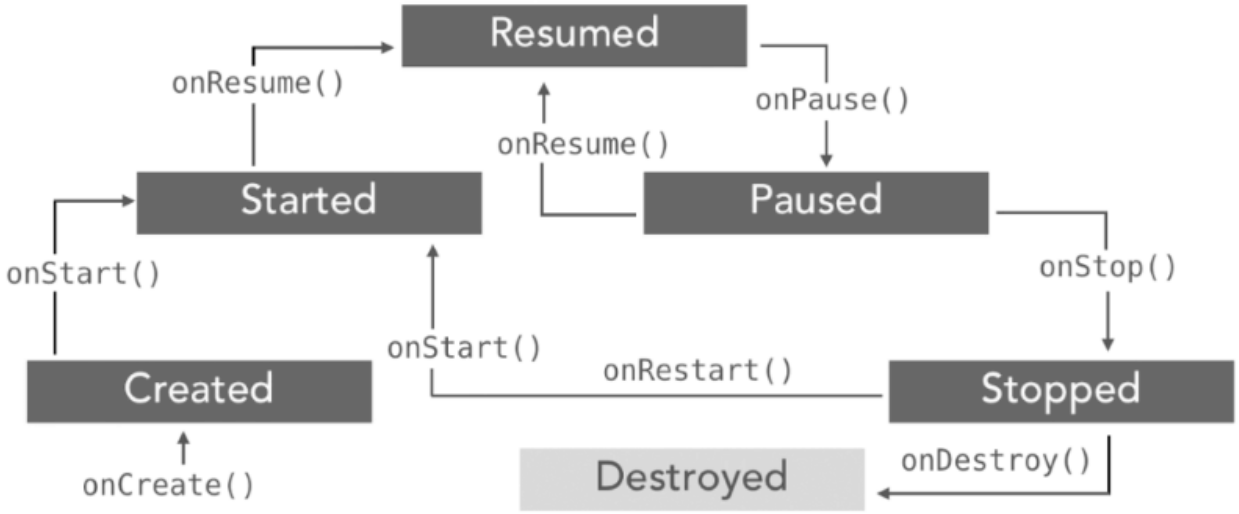

Рисунок 2.1 – Жизненный цикл *Activity*

Жизненный цикл *Android*-приложения жёстко контролируется системой и зависит от нужд пользователя, доступных ресурсов и производительности устройства. Например, когда пользователь хочет запустить браузер, решение о запуске приложения принимает система. Хоть последнее слово и остаётся за системой, она подчиняется заданным и логическим правилам, позволяющим определить, можно ли загрузить или приостановить приложение, прекратить его работу. Когда в определённый момент пользователь работает с конкретным окном, система даёт приоритет соответствующему приложению. И наоборот, если окно невидимо, и система решает, что работу приложения необходимо остановить, чтобы освободить дополнительные ресурсы, будет остановлена работа приложения, имеющего более низкий приоритет. В *Android* ресурсы на мобильном устройстве ограниченны, поэтому он более жёстко контролирует работу приложений[17].

Метод *onCreate*() вызывается при создании или перезагрузки активности. Система может запускать и приостанавливать текущие окна в зависимости от происходящих событий. Внутри данного метода настраивают статический интерфейс окна. В методе инициализируются статические данные активности, связывают данные со списками и так далее. Связывает с необходимыми данными и ресурсами. Задаёт внешний вид через метод *setContentView*(). В этом методе загружаем пользовательский интерфейс, размещаем ссылки на свойства класса, связываем данные с элементами управления, создаем сервисы и потоки. Метод *onCreate*() принимает объект *Bundle*, содержащий состояние пользовательского интерфейса, сохранённое в последнем вызове обработчика *onSaveInstanceState*. Для восстановления графического интерфейса в его предыдущем состоянии нужно задействовать эту переменную: внутри

onCreate() или переопределив метод *onRestoreInstanceState*(). В методе можно сделать проверку, запущено ли приложение впервые или восстановлено из памяти. Если значение переменной *savedInstanceState* будет *null*, приложение запускается первый раз.

Листинг кода метода *onCreate* класса *MainActivity*:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

if(getSession() == null || DatabaseHelper.getUserById(getRealm(), "Guest") ==
null){
//launch auth activity with specific flags and create a guest user
startActivity(AuthActivity.intent(this, true));
}else{

ButterKnife.inject(this);

mDrawerLayout.setStatusBarBackgroundColor(
getResources().getColor(R.color.material_red600));

if (!isTaskRoot()) {
   final Intent intent = getIntent();
   if (intent.hasCategory(Intent.CATEGORY_LAUNCHER) &&
   Intent.ACTION_MAIN.equals(intent.getAction())) {
   finish();
   return;
   }
}

setUpToolbar();
setup(savedInstanceState);
setUpPanel();
if(getSession() != null){
   setupNavUserLayout(getSession().getUser());
   }

 }
}
```

За *onCreate*() всегда следует вызов *onStart*(), но перед *onStart*() не обязательно должен идти *onCreate*(), так как *onStart*() может вызываться и для возобновления работы приостановленного приложения (приложение останавливается методом *onStop*()). При вызове *onStart*() окно еще не видно пользователю, но вскоре будет видно. Вызывается непосредственно перед тем, как активность становится видимой пользователю. Сопровождается вызовом метода *onResume*(), если активность получает передний план, или вызовом метода *onStop*(), если становится скрытой.

Метод *onResume*() вызывается после *onStart*(), даже когда окно работает в приоритетном режиме и пользователь может его наблюдать. В этот момент

пользователь взаимодействует с созданным вами окном. Приложение получает монопольные ресурсы. Запускает воспроизведение анимации, аудио и видео. Также может вызываться после *onPause*().

Система вызывает этот метод каждый раз, когда активность идет на переднем плане, в том числе, при первом создании. Таким образом, мы должны реализовать *onResume*() для инициализации компонентов, регистрации любых широковещательных приемников или других процессов, которые освободили или приостановили в *onPause*() и выполнять любые другие инициализации, которые должны происходить, когда активность вновь активна.

Необходимо пытаться размещать относительно быстрый и легковесный код, чтобы приложение оставалось отзывчивым при скрытии с экрана или выходе на передний план.

Когда пользователь решает перейти к работе с новым окном, система вызовет для прерываемого окна метод *onPause*(). По сути происходит свёртывание активности. Сохраняет незафиксированные данные. Деактивирует и выпускает монопольные ресурсы. Останавливает воспроизведение видео, аудио и анимацию. От *onPause*() можно перейти к вызову либо *onResume*(), либо *onStop*().

Метод *onStop*() вызывается, когда окно становится невидимым для пользователя. Это может произойти при её уничтожении, или если была запущена другая активность (существующая или новая), перекрывшая окно текущей активности. Всегда сопровождает любой вызов метода *onRestart*(), если активность возвращается, чтобы взаимодействовать с пользователем, или метода *onDestroy*(), если эта активность уничтожается.

Для передачи данных между активити, в *Android SDK* существуют интенты. *Intent* представляет собой объект обмена сообщениями, с помощью которого можно запросить выполнение действия у компонента другого приложения. Несмотря на то, что объекты *Intent* упрощают обмен данными между компонентами по нескольким аспектам. В разрабатываемом приложении интенты будут использоваться для запуска активностей и получения результатов с других компонентов. Компонент *Activity* представляет собой один экран в приложении. Для запуска нового экземпляра компонента *Activity* необходимо передать объект *Intent* методу *startActivity*(). Объект *Intent* описывает операцию, которую требуется запустить, а также содержит все остальные необходимые данные.

Если после завершения операции от нее требуется получить результат, вызовите метод *startActivityForResult*(). Ваша операция получит результат в виде отдельного объекта *Intent* в обратном вызове метода *onActivityResult*() операции.

Листинг кода метода *onClick* класса *ThreadActivity*:

```
@OnClick(R.id.upFab)
 public void onUpVote(){
     if (!TRUE.equalsIgnoreCase(postItem.getLikes())){
         threadPresenter.vote(postItem, UP_VOTE);
     }else{
         threadPresenter.vote(postItem, UN_VOTE);
     }
 }
```

Фрагмент (класс *Fragment*) представляет поведение или часть пользовательского интерфейса в операции (класс *Activity*). Разработчик может объединить несколько фрагментов в одну операцию для построения многопанельного пользовательского интерфейса и повторного использования фрагмента в нескольких операциях. Фрагмент можно рассматривать как модульную часть операции. Такая часть имеет свой жизненный цикл и самостоятельно обрабатывает события ввода. Кроме того, ее можно добавить или удалить непосредственно во время выполнения операции. Это нечто вроде вложенной операции, которую можно многократно использовать в различных операциях.

Фрагмент всегда должен быть встроен в операцию, и на его жизненный цикл напрямую влияет жизненный цикл операции. Например, когда операция приостановлена, в том же состоянии находятся и все фрагменты внутри нее, а когда операция уничтожается, уничтожаются и все фрагменты. Однако пока операция выполняется (это соответствует состоянию возобновлена жизненного цикла), можно манипулировать каждым фрагментом независимо, например, добавлять или удалять их. Когда разработчик выполняет такие транзакции с фрагментами, он может также добавить их в стек переходов назад, которым управляет операция. Каждый элемент стека переходов назад в операции является записью выполненной транзакции с фрагментом. Стек переходов назад позволяет пользователю обратить транзакцию с фрагментом (выполнить навигацию в обратном направлении), нажимая кнопку назад.

У фрагментов жизненный цикл схож с активити, добавляются только пара методов:

– *onAttach*;
– *onCreateView*;
– *onActivityCreated*;
– *onDestroyView*;
– *onDetach*.

Фрагменты не могут существовать без активити, и их методы жизненного цикла вызываются последовательно. Полный жизненный цикл фрагментов показан на рисунке 2.2.

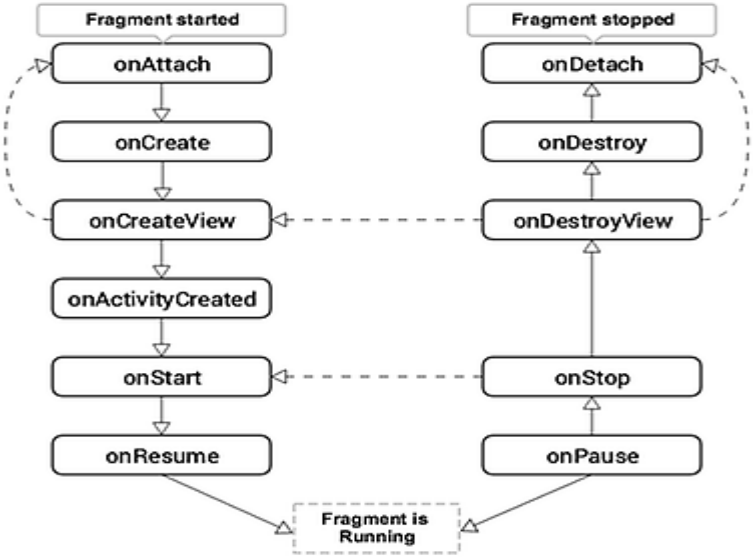

Рисунок 2.2 – Жизненный цикл *Fragment*

Метод *onAttach*() вызывается, когда фрагмент связывается с операцией (ему передается объект *Activity*), *onCreateView*() для создания иерархии представлений, связанной с фрагментом, *onActivityCreated*(), когда метод *onCreate*(), принадлежащий операции, возвращает управление, *onDestroyView*() при удалении иерархии представлений, связанной с фрагментом, *onDetach*() при разрыве связи фрагмента с операцией.

В разрабатываемом приложении создан один фрагмент. Массив объектов этого фрагмента будем создаваться на главном экране приложения и инициализировать *ViewPager*, для переключения движением пальцем по экрану между днями. *ViewPager* позволяет нам организовать удобный и красивый просмотр данных с возможностью перелистывания влево-вправо. Сам *ViewPager* отвечает за показ и прокрутку. Но ему нужен еще *PagerAdapter*, который отвечает за предоставление данных.

PagerAdapter – это базовый абстрактный класс, для которого разработчик дописывает реализацию так, как ему надо. Существует распространенная стандартная реализация *PagerAdapter*, которая работает с фрагментами – это *FragmentPagerAdapter*.

Листинг кода класса *PostFragment*:

```
public class PostFragment extends BaseFragment implements PostView{

Activity activity;
PostItem postItem;

@InjectView(R.id.content)
TextView mContent;

@Override
```

```
      public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {

      View view = inflater.inflate(R.layout.preview_post_fragment, container,
false);
      ButterKnife.inject(this, view);

      String                  json                  =                  getAr-
guments().getString(getResources().getString(R.string.main_data_key));
      postItem = new Gson().fromJson(json, PostItem.class);
      String content = postItem.getText();

      if(content.length() > 0){
      mContent.setText(content);
      }else{
      FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(FrameLay-
out.LayoutParams.WRAP_CONTENT, FrameLayout.LayoutParams.WRAP_CONTENT);
      params.gravity = Gravity.CENTER_HORIZONTAL;
      params.setMargins(0, 100, 0, 0);
      mContent.setLayoutParams(params);
      }

      return view;
      }

      @Override public void onActivityCreated(Bundle savedInstanceState) {
      super.onActivityCreated(savedInstanceState);
      this.activity = getActivity();
      }
```

В некоторых случаях требуется показать диалоговое окно, где пользователю нужно сделать какой-нибудь выбор или показать сообщение об ошибке. Безусловно, можно создать собственное окно, расположить в нем нужные кнопки и обрабатывать их нажатия. Но, в *Android* уже есть собственные встроенные диалоговые окна, которые гибко настраиваются под задачи. Использование диалоговых окон для простых задач позволяет сократить число классов *Activity* в приложении, экономя ресурсы памяти. Ведь нет необходимости регистрировать активность в манифесте, думать над компоновкой элементов на экране и так далее.

Диалоговые окна в *Android* представляют собой полупрозрачные «плавающие» активности, частично перекрывающие родительский экран, из которого их вызвали. Как правило, они затеняют родительскую активность позади себя с помощью фильтров размывания или затемнения. Можно установить заголовок с помощью метода *setTitle*() и содержимое с помощью метода *setContentView*().

Класс *Dialog* является базовым для всех классов диалоговых окон. Поскольку *ProgressDialog*, *TimePickerDialog* и *DatePickerDialog* – расширение класса *AlertDialog*, они также могут иметь командные кнопки.

Каждое диалоговое окно должно быть определено внутри активности, в которой будет использоваться. Диалоговое окно можно открыть один раз или несколько раз.

Для отображения диалогового окна необходимо вызвать метод *showDialog*() и передать ему в качестве параметра идентификатор диалога (константа, которую надо объявить в коде программы), который вы хотите отобразить.

Метод *dismissDialog*() прячет диалоговое окно (но не удаляет), не отображая его на экране. Окно остается в пуле диалоговых окон данной активности. При повторном отображении при помощи метода *showDialog*() будет использована кэшированная версия окна.

Метод *removeDialog*() удаляет диалоговое окно из пула окон данной активности. При повторном вызове метода *showDialog*() диалоговое окно придется создавать снова.

Метод *onCreateDialog*() вызывается один раз при создании окна. После начального создания при каждом вызове метода *showDialog*() будет срабатывать обработчик *onPrepareDialog*(). Переопределив этот метод, вы можете изменять диалоговое окно при каждом его выводе на экран. Это позволит привнести контекст в любое из отображаемых значений. Если требуется перед каждым вызовом диалогового окна изменять его свойства (например, текстовое сообщение или количество кнопок), то можно реализовать внутри этого метода. В этот метод передают идентификатор диалога и сам объект *Dialog*, который был создан в методе *onCreateDialog*().

Файл манифеста *AndroidManifest.xml* хранит основную информацию о системе программы. У каждого приложения имеется свой файл *AndroidManifest.xml.*

Редактировать файл манифеста можно вручную, изменяя *XML*-код или через визуальный редактор *Manifest Editor* (Редактор файла манифеста), который позволяет производить визуальное и текстовое редактирование файла манифеста приложения.

Назначение файла:

– объявление имени *Java*-пакета приложения, который служит уникальным идентификатором;

– описание компонентов приложения: деятельности, службы, приемники широковещательных намерений и контент-провайдеры, что позволяет вызывать классы, которые реализуют каждый из компонентов, и объявляет их намерения;

– содержит список необходимых разрешений для обращения к защищенным частям *API* и взаимодействия с другими приложениями;

– объявляет разрешения, которые сторонние приложения обязаны иметь для взаимодействия с компонентами данного приложения;

– объявляет минимальный уровень *API Android*, необходимый для работы приложения;

– перечисляет связанные библиотеки.

Файл манифеста инкапсулирует всю архитектуру *Android*-приложения, его функциональные возможности и конфигурацию. В процессе разработки приложения приходится постоянно редактировать данный файл, изменяя его структуру и дополняя новыми элементами и атрибутами.

Корневым элементом манифеста является <*manifest*>. Помимо данного элемента обязательными элементами является теги <*application*> и <*uses-sdk*>. Элемент <*application*> является основным элементом манифеста и содержит множество дочерних элементов, определяющих структуру и работу приложения. Порядок расположения элементов, находящихся на одном уровне, произвольный. Все значения устанавливаются через атрибуты элементов. Кроме обязательных элементов, упомянутых выше, в манифесте по мере необходимости используются другие элементы.

Элемент <*application*> один из главних элементов файла манифеста, он содержит описание компонентов приложения, доступных в пакете: стили, значок, строки и др. Содержит дочерние элементы, которые объявляют каждый из компонентов, входящих в состав приложения. Также в файле может быть только один элемент <*application*>.

Элемент <*activity*> объявляет активность. Если приложение содержит несколько активностей, то нужно обязательно объявлять их в манифесте, создавая для каждой из них свой элемент <*activity*>. Если активность не объявлена в манифесте, она не будет доступна системе и не будет запущена при выполнении приложения или будет выводиться сообщение об ошибке. Каждый тег <*activity*> поддерживает вложенные узлы <*intent-filter*>. Элемент <*intent-filter*> определяет типы намерений, на которые могут ответить деятельность, сервис или приемник намерений. Фильтр намерений определяет возможности его родительского компонента, в нём описано, что могут сделать деятельность или служба и какие типы рассылок получатель может обработать. Фильтр намерений предоставляет для компонентов-клиентов возможность получения намерений объявляемого типа, отфильтровывая те, которые не значимы для компонента, и содержит дочерние элементы <*action*>, <*category*>, <*data*>.

Элемент <*service*> объявляет службу как один из компонентов приложения. Все службы должны быть представлены элементом <*service*> в файле манифеста. Службы, которые не были объявлены, не будут обнаружены системой и никогда не будут запущены. Этот элемент имеет много атрибутов, определяющих имя, доступность, разрешения, процесс и т. д. Поддерживает вложенные узлы <*intent-fiiter*> [18].

Листинг кода файла *AndroidManifest.xml*:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="by.diplom.bsuir.gleb.myapplication">

    <uses-permission android:name="android.permission.INTERNET" />
```

```
    <uses-permission android:name="android.permission.ACCESS_NET-
WORK_STATE" />

    <application
        android:name=".ui.App"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <receiver android:name=".data.network.connection.Network-
StateReceiver">
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIV-
ITY_CHANGE" />
            </intent-filter>
        </receiver>

        <activity
            android:name=".ui.common.main.MainActivity"
            android:label="@string/app_name"
            android:launchMode="standard"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.cate-
gory.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <category android:name="android.intent.category.DEFAULT"
/>
                <category android:name="android.intent.category.BROWSA-
BLE" />

                <action android:name="android.intent.action.VIEW" />

                <data android:scheme="by.diplom.bsuir.gleb.myapplica-
tion" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ui.subcription.SubscriptionActivity"
            android:label="@string/fragment_subreddits"
            android:screenOrientation="portrait" />
        <activity
            android:name=".ui.thread.ThreadActivity"
            android:label="@string/title_activity_comments"
            android:screenOrientation="portrait"
            android:theme="@style/Slider" />
        <activity
            android:name=".ui.common.auth.AuthActivity"
            android:configChanges="orientation|keyboardHid-
den|screenSize"
            android:label="@string/title_activity_auth"
            android:screenOrientation="portrait"
            android:theme="@style/FullscreenTheme" />
        <activity
            android:name=".ui.settings.SettingsActivity"
            android:label="@string/title_activity_settings" />
        <activity android:name=".ui.profile.ProfileActivity"></activity>
    </application>
</manifest>
```

Чтобы вручную не собирать проект и не прописывать множество команд, разработчики создали системы сборки проектов. В нашем случае используется *Gradle*.

Gradle – система автоматической сборки, построенная на принципах *Apache Ant* и *Apache Maven*, но предоставляющая *DSL* на языке *Groovy* вместо традиционной *XML*-образной формы представления конфигурации проекта [19]. В нём мы указываем наши зависимости, версии минимальной и максимальной *SDK*, версию приложения и много других настроек.

Листинг кода файла *build.gradle*:

```
      apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.3"
    defaultConfig {
        applicationId " by.diplom.bsuir.gleb.myapplication"
        minSdkVersion 16
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUni-
tRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
    sourceSets {
        main.java.srcDirs += 'src/main/kotlin'
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:design:25.3.1'
    compile 'org.jetbrains.anko:anko-sdk15:0.9.1'
    compile 'org.jetbrains.anko:anko-support-v4:0.9.1'
    compile 'org.jetbrains.anko:anko-appcompat-v7:0.9.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.0-beta1'
    compile 'com.google.firebase:firebase-ads:9.8.0'
    compile 'com.google.android.gms:play-services-ads:9.8.0'
    compile 'com.vk:androidsdk:1.6.8'
    testCompile 'junit:junit:4.12'
    androidTestCompile 'junit:junit:4.12'
}
repositories {
```

```
    mavenCentral()
}
```

Разрабатывая по архитектурному шаблону *MVC* (*Model-View-Controller*), в первую очередь мы реализовали слой *Model* и *Controller*. После этого можно приступить к разработке слоя *View*, который будет отвечать за представление.

2.4 Информационная модель системы

В данном проекте для хранения данных используется нереляционная база данных *Realm*. Особенность этой системы является в том, что таблицы данных не ограниченны связями, что позволяет без затруднений расширять набор данных в будущем, также данная система может обрабатывать запросы быстрее по сравнению с реляционными СУБД [20].

Для работы приложения был создан набор сущностей, которые покрывают имеющийся функционал и позволяют использовать их для реализации новых функций. Ниже приведён список основных сущностей:

– сущность *User* хранит в себе основные данные пользователя после авторизации. В эти данные входит: имя пользователя, уникальный номер пользователя, тип пользователя, специальный ключ текущей сессий, ссылка на объект с настройками приложения;

– сущность *Preferences* хранит данные о настройках приложения. В эти данные входит: показ популярных подразделов, отображение недавних публикаций, показ публикаций содержащих спорных публикаций;

– сущность *Subreddit* хранит данные о подразделах, на которые подписан пользователь. В эти данные входит: название подраздела, описание, количество подписчиков, количество активных пользователей, тип подраздела;

– сущность *Token* хранит данные специального ключа (токена), он является уникальным для каждого пользователя, благодаря этому приложению пользователя не нужно иметь постоянно открытую сессию с веб-сервисом. В данные этой сущности входит: уникальный номер ключа, сгенерированный код, который является ключом, тип ключа, тип пользователя, срок действия.

Такой набор сущностей будет хранить необходимый набор информации для работы программного средства.

2.5 Проектирование пользовательского интерфейса

Пользовательский интерфейс является принципиально важной частью продукта. Неудобный пользовательский интерфейс может оттолкнуть пользователя, снизить эффективность использования веб-сервиса. Для разработки пользовательского интерфейса обычно привлекают дизайнера.

В связи с особенностями программного продукта пользовательский интерфейс также должен быть прост в использовании – расположение некоторых элементов, цветовая гамма и т.п. должны соответствовать сценария использования приложения через сенсорный экран мобильного устройства.

Пользовательский интерфейс разрабатываемого продукта использует уже существующие решения. Основные компоненты были заимствованы из свободного шаблонов *Material Design*. Изображение приведено на рисунке 2.3.

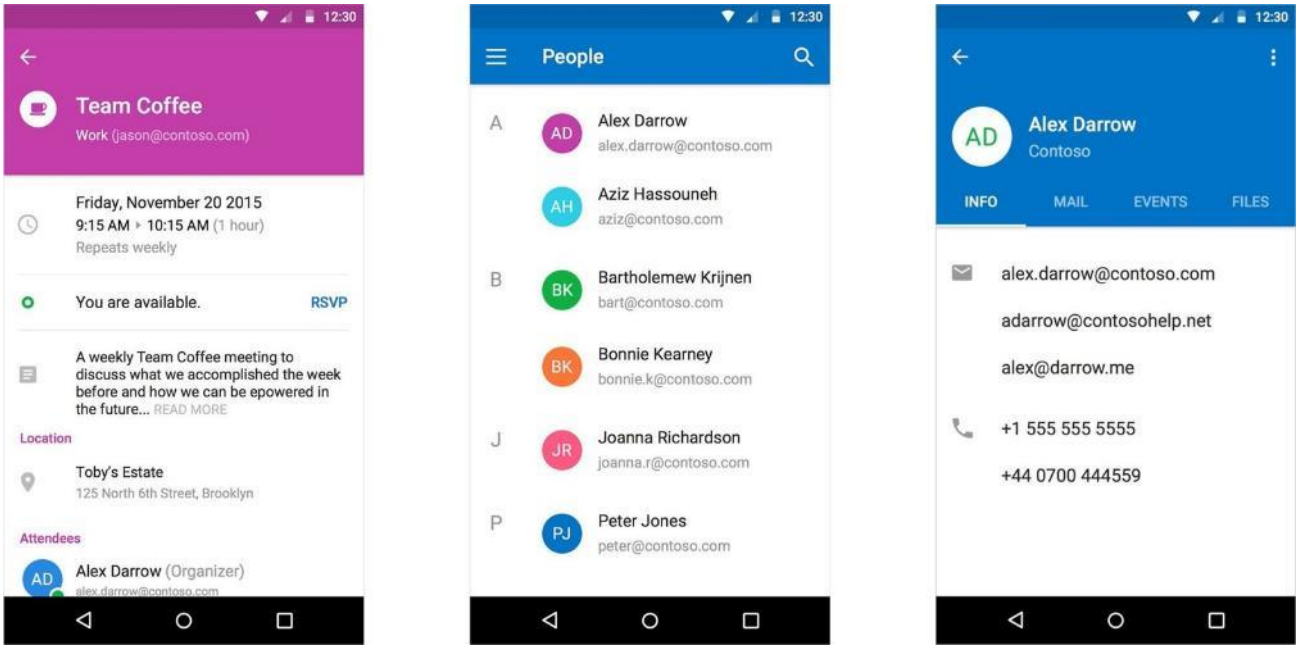

Рисунок 2.3 – Интерфейс шаблона *Material Design*

Шаблон содержит много различных виджетов, панелей, элементов интерфейса, и позволяет в краткие сроки создать удобное приложение.

Material Design представляет собой комплексную концепцию создания визуальных, движущихся и интерактивных элементов для различных платформ и устройств. Теперь *Android* включает в себя поддержку приложений с элементами *Material Design*. Чтобы использовать элементы *Material Design* в своих приложениях под *Android*, руководствуйтесь инструкциями в спецификации *Material Design*, а также воспользуйтесь новыми компонентами и функциями, доступными в *Android* 5.0 (уровень *API* 21) и выше.

Android предоставляет следующие элементы для построения приложений в соответствии с концепцией *Material Design*:

– новую тему;

– новые виджеты для сложных представлений;

– новые *API*-интерфейсы для нестандартных теней и анимаций [21].

Создание и настройка внешнего вида пользовательского интерфейса приложения происходит в *Android Studio*, файлы с разметкой интерфейса сохраняются в формате *XML*.

Преимущество объявления пользовательского интерфейса в файле *XML* заключается в том, что таким образом вы можете более эффективно отделить представление своего приложения от кода, который управляет его поведением.

Описания пользовательского интерфейса находятся за пределами кода вашего приложения. Это означает, что вы можете изменять или адаптировать интерфейс без необходимости вносить правки в исходный код и повторно компилировать его. Например, можно создать разные файлы *XML* макета для экранов разных размеров и разных ориентаций экрана, а также для различных языков. Кроме того, объявление макета в *XML* упрощает визуализацию структуры пользовательского интерфейса, благодаря чему отладка проблем также становится проще.

В данной статье мы научим вас объявлять макет в *XML*. Если вы предпочитаете создавать экземпляры объектов *View* во время выполнения, обратитесь к справочной документации для классов *ViewGroup* и *View* [22].

Интерфейс программного средства должен удовлетворять следующим требованиям:

– интерфейс должен быть интуитивно понятен и не вызывать замешательства;

– в интерфейсе должен присутствовать доступ к наиболее необходимым для пользователя частям программного средства;

– интерфейс должен располагать цветовой гаммой, не раздражающей глаз и не вызывающей чувство отвращения, цвета должны быть спокойных тонов;

– интерфейс должен максимально использовать свободное пространство экрана мобильного устройства, но не должно быть нагромождения;

– использование программного средства должно быть понятно пользователю без специальной подготовки и не должно требовать прохождения предварительного обучения в использовании программы;

– расположение элементов, стиль иконок и поведение интерфейса должно следовать устоявшимся стандартам разработки программ для системы *Android*, для того чтобы программное средство не выбивалось из визуального стиля системы.

Для создания пользовательского интерфейса в среде разработки *Android Studio* существует несколько вариантов. Некоторые из них могут включать в себя использование специальных программ для прототипирования интерфейсов или используют инструменты позволяющие создавать программу под разные системы. Ниже мы рассмотрим основные варианты *Android*-разработчика.

Первый вариант – это заполнение файла макета через ручное написание кода, рисунок 2.4.

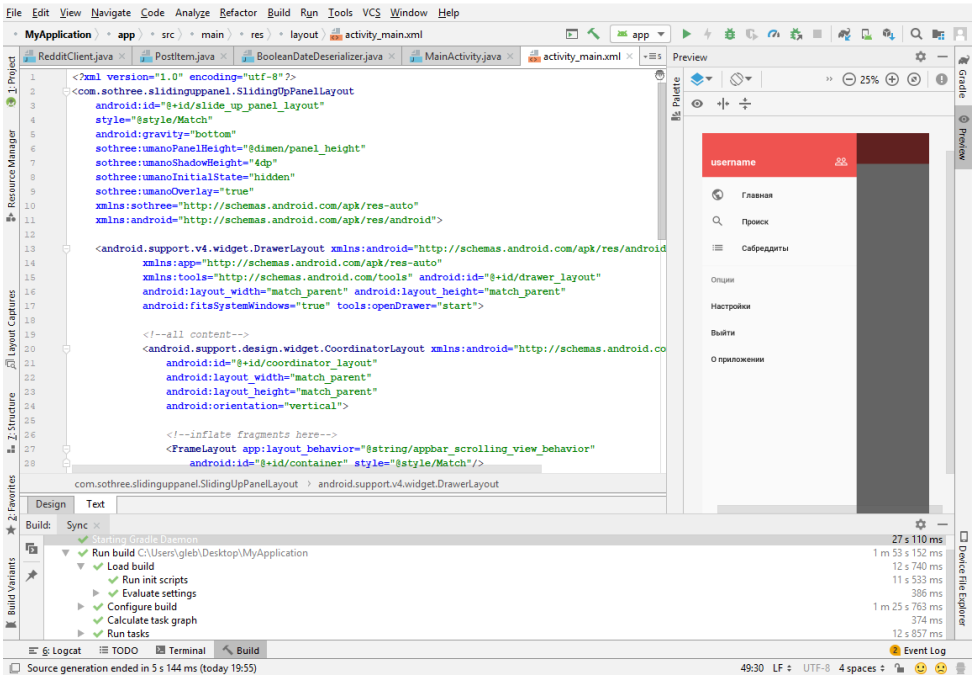

Рисунок 2.4 – Ручное написание кода в файл макета в среде разработки *Android Studio*

Второй это возможность работать с *UI* компонентами при помощи функции *Drag-and-Drop*, рисунок 2.5.

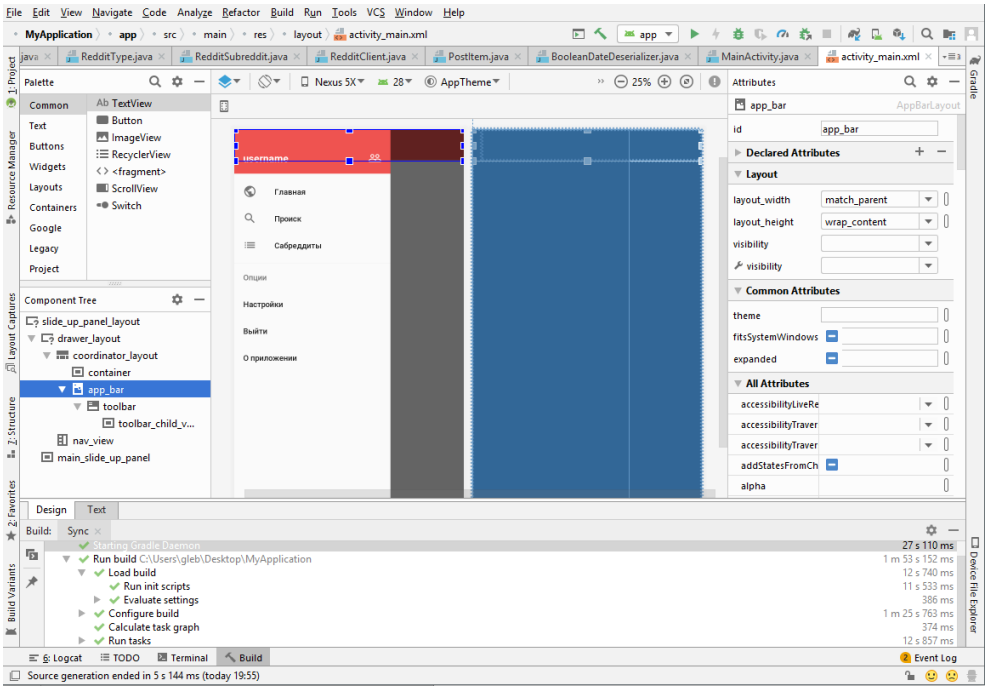

Рисунок 2.5 – Работа с *UI* компонентами при помощи функции *Drag-and-Drop* в среде разработки *Android Studio*

Оба варианта создания интерфейса имеют свои преимущества, способ с ручным изменением *XML*-файла даёт полный контроль на настройкой интерфейса, но не даёт наглядной картины о внешнем виде и может оказаться затруднительным для неопытных разработчиков. Способ создания интерфейса с помощью перетаскивания элементов из панели инструментов в область экрана является наиболее наглядным и позволяет быстро получить представление о внешнем виде будущего интерфейса, однако реализация сложных функций и переходов данным способом может быть невозможна, так как он даёт ограниченный контроль над настройкой.

При разработке пользовательского интерфейса использовались оба варианта, так как это позволяет воспользоваться преимуществами одного и избежать недостатков другого способа. Первоначальная настройка интерфейса происходила с помощь функции *Drag-and-Drop*, а более глубокая настройка происходила с помощью ручного редактирования файла макета. На рисунке 2.6 представлен интерфейс главного экрана приложения.

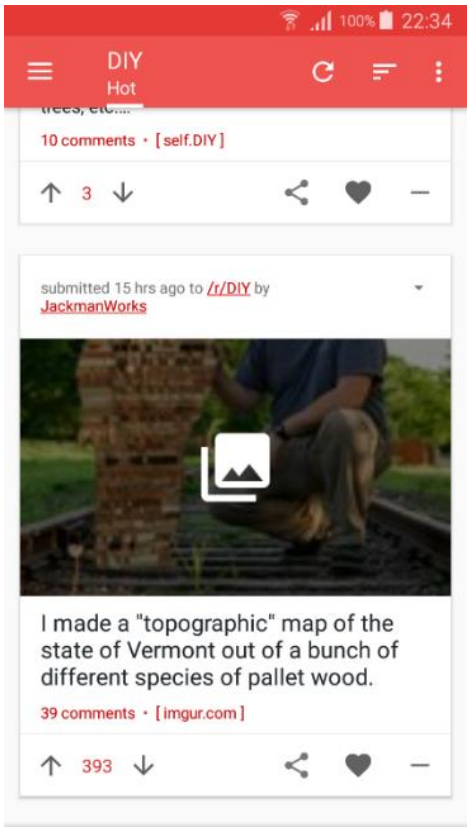

Рисунок 2.6 – Пользовательский интерфейс главного экрана

На главном экране пользователь может видеть список публикаций, загруженных по выбранному типу сортировки.

Также были разработаны файлы макета для части программного средства где отображается информация о публикации и комментарии к ней (рисунок 2.7).

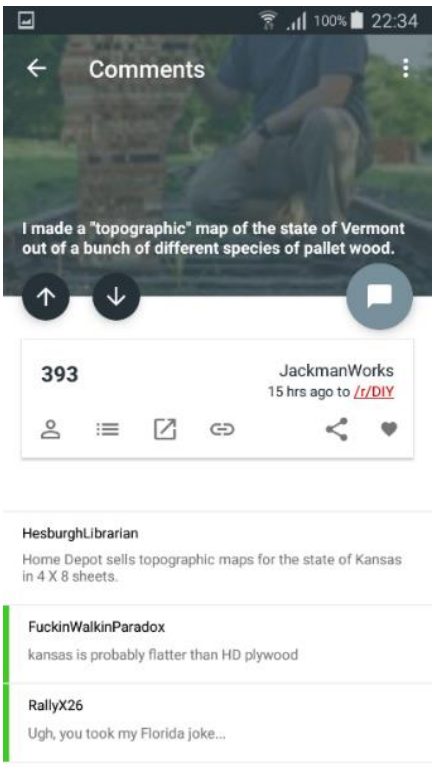

Рисунок 2.7 – Интерфейс экрана информации о публикации

На данном экране можно увидел изображение публикации, список комментарием и кнопки для взаимодействия с публикацией.

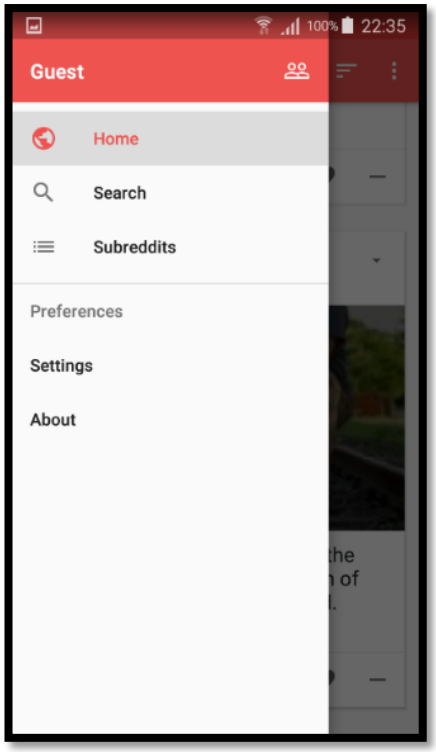

Рисунок 2.8 – Пользовательский интерфейс экрана с открытым боковым меню

При переходе в боковое меню можно получить быстрый доступ к основным функциям программного средства, в них входит: переход к настройкам, выбор другого потока публикаций, выбор профиля пользователя.

Доступ к данному меню можно получить из любой части программного интерфейса, это даёт возможность для быстрой настройки программного средства и переходам к другим разделам веб-сервиса.

Для быстрого перемещения между публикациями была реализована функция смахивания. На рисунке 2.9 показан момент перехода от текущей публикации к списку всех публикаций.

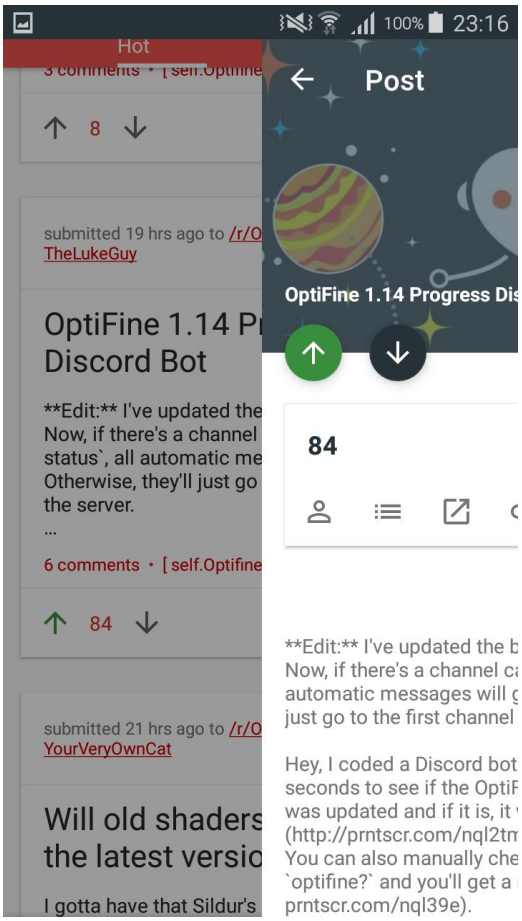

Рисунок 2.9 – Изображение функции смахивания

Данная функция упрощает взаимодействие с интерфейсом путём исключения необходимости нажатия кнопки выхода, которая находится в углу экрана. Такой подход следует принципу естественности интерфейса.

Естественный интерфейс — такой, который не вынуждает пользователя существенно изменять привычные для него способы решения задачи. Это, в частности, означает, что сообщения и результаты, выдаваемые приложением, не должны требовать дополнительных пояснений. Целесообразно также сохранить систему обозначений и терминологию, используемые в данной предметной области.

Использование знакомых пользователю понятий и образов (метафор) обеспечивает интуитивно понятный интерфейс при выполнении его заданий. Метафоры являются своего рода «мостиком», связывающим образы реального мира с теми действиями и объектами, которыми приходится манипулировать пользователю при его работе на компьютере; они обеспечивают «узнавание», а не «вспоминание». Пользователи запоминают действие, связанное со знакомым объектом, более легко, чем они запомнили бы имя команды, связанной с эти действием [23].

В верхней части пользовательского интерфейса всегда видна строка ввода для поискового запроса, рисунок 2.10.

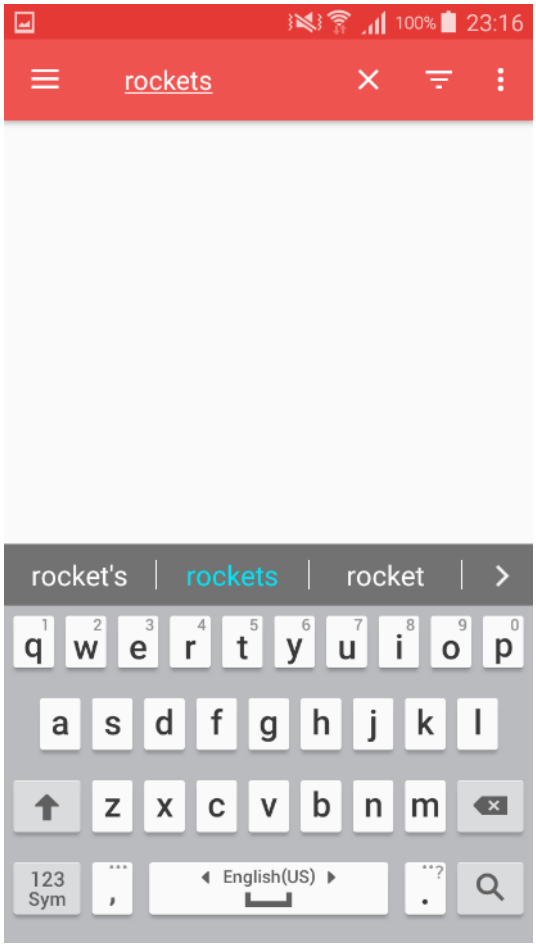

Рисунок 2.10 – Пользовательский интерфейс экрана в момента использования поисковой строки

Такое расположение данного элемента интерфейса позволяет пользователю не задумываться о местонахождении поисковой строки, так как она всегда находится в поле зрения и в любой момент готова к использованию. Такой подход реализует принцип согласованности. Согласованность важна для всех аспектов интерфейса, включая имена команд, визуальное представление информации и поведение интерактивных элементов. Для реализации свойства

согласованности в создаваемом программном обеспечении, необходимо учитывать его различные аспекты [24].

При нажатии на заголовок раздела можно открыть всплывающее окно со списком самых популярных разделов, рисунок 2.11.

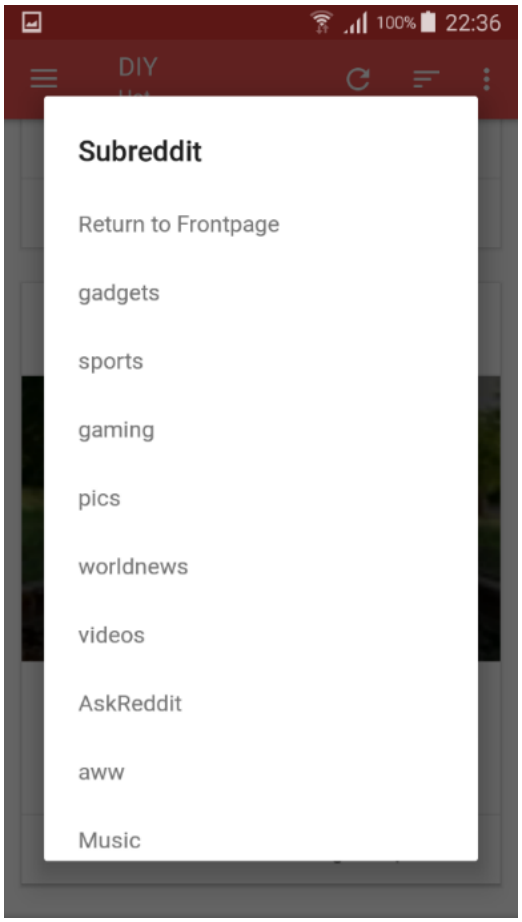

Рисунок 2.11 – Пользовательский интерфейс экрана с открытым всплывающим окном для выбора раздела

Данная функция позволяет быстро и удобно перемещаться по различным разделам веб-сервиса.

Цвета темы, используемые в приложении хранятся в файле *colors.xml*.

Листинг кода файла *colors.xml*:

```
      <?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Также при помощи языка *XML*, были созданы кнопки. В файле *button_for_dialog.xml* написана кнопка, используемая в диалогах и в файле *button_shape.xml*, используемая в основных экранах.

Листинг кода файла *button_for_dialog.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" an-
droid:shape="rectangle" >
    <corners
        android:topLeftRadius="0dp"
        android:topRightRadius="0dp"
        android:bottomLeftRadius="0dp"
        android:bottomRightRadius="0dp"
        />

    <gradient
        android:angle="45"
        android:centerX="35%"
        android:centerColor="#229FA8"
        android:startColor="#3AE8DC"
        android:endColor="#4A68FF"
        android:type="linear"
        />
    <size
        android:width="200dp"
        android:height="50dp"
        />
    <stroke
        android:width="3dp"
        android:color="#878787"
        />
</shape>
```

Листинг кода файла *button_shape.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" an-
droid:shape="rectangle" >
    <corners
        android:radius="14dp"
        />
    <gradient
        android:angle="45"
        android:centerX="35%"
        android:centerColor="#47A891"
        android:startColor="#6F5DE8"
        android:endColor="#392BFF"
        android:type="linear"
        />
    <padding
        android:left="0dp"
        android:top="0dp"
        android:right="0dp"
        android:bottom="0dp"
        />
    <size
        android:width="120dp"
        android:height="30dp"
        />
    <stroke
```

```
        android:width="3dp"
        android:color="#878787"
        />
</shape>
```

Интерфейс проектировался таким образом, чтобы на экране устройства отображалась самая необходимая для пользователя информация и функции. Также получились интуитивно понятное меню и навигация в приложении. Интерфейс должен быть простым. При этом имеется в виду не упрощенчество, а обеспечение легкости в его изучении и в использовании. Кроме того, он должен предоставлять доступ ко всему перечню функциональных возможностей, предусмотренных данным приложением. Реализация доступа к широким функциональным возможностям и обеспечение простоты работы противоречат друг другу. Разработка эффективного интерфейса призвана сбалансировать эти цели[25].

Таким образом был разработан пользовательский интерфейс, отвечающий требованиям, описанным в начале подраздела.

2.6 Проектирование интерфейса взаимодействия с серверным API

Для взаимодействия с веб-сервисом *reddit.com* необходимо придерживаться *API* команд представленные в документации для разработчиков на специальной странице сервиса. Для работы с этими командами будет использоваться библиотека *Retrofit*, которая позволяет автоматизировать создания *JSON* запросов и последующего преобразования полученных данных. Чтобы получить доступ ко всем функциям программного средства, необходимо авторизироваться и сохранить ключ сессии, для этого будет использоваться сущность *User*, в которой будет сохранена ссылка на сущность *Token* для хранения данных о ключе. Интерфейс *RedditAuthProvider* позволяет получать ключ доступа и обновлять его после истечения срока действия.

Данное веб-сервис поддерживает авторизацию с помощью *OAuth*.

OAuth – популярный протокол, который позволяет социальным сервисам интегрироваться между собой и дает безопасный способ обмена персональной информацией [26].

Для доступа ко всем функциям сервиса необходимо пройти авторизацию через *OAuth* для этого необходимо пройти ряд шагов:

– клиент посредством протокола *HTTPS* отправляет серверу запрос, который содержит идентификатор клиента, метку времени, адрес обратного вызова по которому нужно вернуть токен;

– сервер подтверждает запрос и отвечает клиенту токеном доступа (*Access Token*) и частью разделённого секрета;

– клиент передает токен владельцу ресурсов (пользователю) и перенаправляет его на сервер для прохождения авторизации;

– сервер, получив от пользователя токен, запрашивает у него его логин и пароль, и в случае успешной аутентификации просит пользователя подтвердить доступ клиента к ресурсам (авторизация), после чего пользователь перенаправляется сервером к клиенту;

– клиент передает серверу токен посредством протокола *TLS* и запрашивает доступ к ресурсам;

– сервер подтверждает запрос и отвечает клиенту новым токеном доступа;

– используя новый токен, клиент обращается к серверу за ресурсами;

– сервер подтверждает запрос и предоставляет ресурсы [27].

Листинг файла *RedditAuthProvider*:

```
public interface RedditAuthProvider {
    //oauth
    String NAMESPACE = "api/v1";
    String ACCESS_TOKEN = "/access_token";
    String REVOKE_TOKEN = "/revoke_token";
    String REFRESH_TOKEN = "/refresh_token";


    @FormUrlEncoded
    @POST(NAMESPACE+ACCESS_TOKEN)
    Observable<AccessToken> obtainAccessToken(
            @Field("grant_type") String grantType,
            @Field("device_id") String deviceId
    );

    @FormUrlEncoded
    @POST(NAMESPACE+ACCESS_TOKEN)
    Call<AccessToken> obtainAccessTokenSync(
            @Field("grant_type") String grantType,
            @Field("device_id") String deviceId
    );

    @FormUrlEncoded
    @POST(NAMESPACE+ACCESS_TOKEN)
    Call<AccessToken> refreshAccessToken(
            @Header("redditapp-refresh-header") String msg,
            @Field("grant_type") String grantType,
            @Field("refresh_token") String refreshToken);
```

В документации для разработчиков описано огромное количество различных команд для взаимодействия с веб-сервисом, однако многие из них могут требовать особых прав доступа или вовсе не поддерживаются текущей версией сервиса. Ниже мы рассмотрим основные команды, которые использовались в данном приложении:

– *Reddit.getAccessToken* используется для получения ключа для текущей сессии. В этом запросе передаётся уникальный номер приложения *appClientId* и *appClientSecret*, но в нашем случает второй параметр не используется, так же передаётся *redirectUri*, которые необходим для браузерных приложений, поэтому он также не используется;

– *Reddit.refreshAccessToken* используется для обновления ключа сессии, если у него истёк срок действия. В этом запросе передаются *appClientId* и *appClientSecret*, а в ответ пользователь получает *refreshToken*;

– *Reddit.getMyPrefs* используется для получения настроек профиля пользователя. Запрос отправляет *appClientId* и *accessToken*, а в ответ получает информацию в формате *JSON* с сохраненными в ней настройками;

– *Reddit.upVote* вызывается, когда пользователь хочет повысить рейтинг публикации. Запрос отправляет *appClientId*, *accessToken*, уникальный номер публикации и численное значение, на которое пользователь хочет повысить рейтинг, но в силу ограничений прав доступа, рейтинг можно повысить только на одну единицу, независимо от переданного значения;

– *Reddit.downVote* работает аналогично запросу *Reddit.upVote*;

– *Reddit.getComments* используется, когда пользователь открывает выбранную публикацию и начинается автоматическая загрузка комментариев. Запрос передаёт данные ключа пользователя и номер публикации, в ответ будет получен *JSON*-список данных, в котором будут храниться данные о заголовке публикации, текст комментария, количество ответов на этот комментарий, тип сортировки.

Reddit.getHot вызывается, когда пользователь открывает главную страницу или какой-либо подраздел, после чего начинается загрузку публикаций в определённом порядке. Этот запрос будет сортировать публикации по популярности в данный момент времени. Запрос передаёт данные о пользователе, название подраздела – *subreddit*, количество загружаемых публикаций – *limit*, с ограничением до 100 штук, в ответ приходит *JSON*-список публикаций. Аналогичные запросы существуют и для других типов сортировки: *Reddit.getControversial*, *Reddit.getNew*, *Reddit.getRandom*, *Reddit.getTop*, *Reddit.getRising*.

Рассмотрев основные шаги разработки программного средства, можно переходить к этапу инженерных расчётов и оценки производительности данной программы.

3 ИНЖЕНЕРНЫЕ РАСЧЁТЫ, ИСПОЛЬЗУЕМЫЕ В ДИПЛОМНОМ ПРОЕКТЕ

3.1 Оценка временных характеристик программного средства при работе с сервером

При эксплуатации мобильного приложение крайне важно, чтобы оно стабильно работало, быстро запускалось и не зависало при переходах между экранами. Тестирование приложение будет происходить на реальном *Android* устройстве *Samsung galaxy S5 mini*. Для точности опытов будет сделано по пять экспериментов и считаться среднее время операций по формуле:

$$T_{\text{с}} = \frac{\sum_{i=1}^{n}(t_{\text{н}i} - t_{\text{к}i})}{n}, \tag{3.1}$$

где $T_{\text{с}}$ – среднее время;

$t_{\text{н}i}$ – начальный момент времени (пользователь выбрал публикацию);

$t_{\text{к}i}$ – конечный момент времени (был получен результат от веб-сервиса).

Пользователи постоянно будут выбирать различные публикации представленных в списке на главной странице. Рассчитаем время открытия экрана публикации. Время нажатие кнопки и метод создания жизненного цикла активити будем выводить в логи (рисунок 3.1).

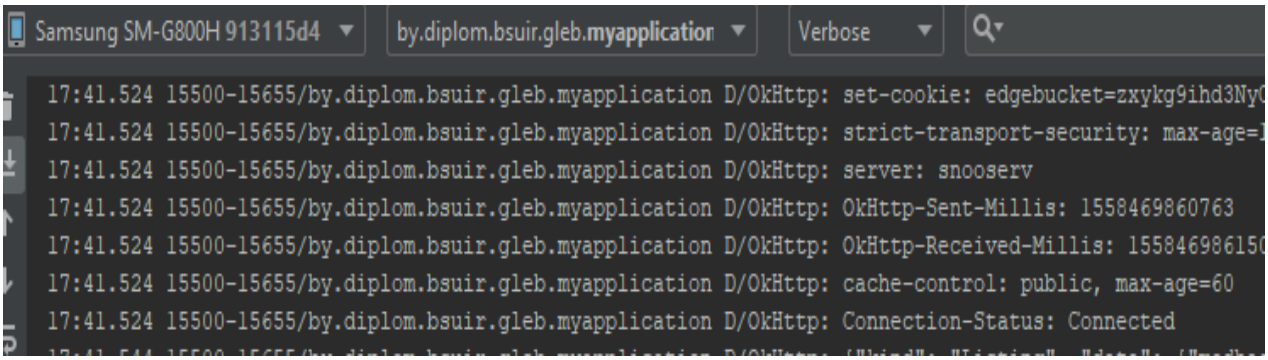

Рисунок 3.1 – Пример вывода логов в консоль *AndoidStudio*

Таблица 3.1 – Результаты опытов открытия экрана публикации

Номер опыта	Момент времени начала $t_{\text{н}i}$, *c*	Момент времени результата t_{ci}, *c*
1	17,652	17,762
2	22,982	23,052
3	27,892	27,992
4	55,862	55,952
5	58,952	59,022

Таким образом на основе полученных результатов по формуле 3.1 рассчитаем $T_с$:

$$T_с = \frac{0{,}11 + 0{,}07 + 0{,}1 + 0{,}09 + 0{,}07}{5} = 0{,}07 \text{ с.}$$

Аналогичным образом рассчитаем время открытия списка разделов.

Таблица 3.2 – Результаты опытов открытия экрана списка разделов

Номер опыта	Момент времени действий $t_{нi}$, c	Момент времени результата t_{ci}, c
1	13,329	14,339
2	20,569	21,169
3	25,319	25,699
4	28,699	29,019
5	34,149	34,449

Таким образом на основе полученных результатов по формуле 3.1 рассчитаем $T_с$:

$$T_с = \frac{1{,}01 + 0{,}6 + 0{,}1 + 0{,}38 + 0{,}3}{5} = 0{,}478 \text{ с.}$$

Из опытов видно, что задержка между тем, когда пользователь открывает разные экраны приложение очень мала. Можно сделать вывод о том, что приложение будет приятным в использовании и не будет раздражать пользователя неприятными задержками при работе.

3.2 Оценка эффективности функционирования программного средства

Как было рассмотрено в предыдущих разделах, программное средство выполняет все операции с веб-сервисом. Это составляет основу функциональных возможностей программного средства. Для комфортного использования приложения скорость добавления, чтения, изменения данных должна быть велика.

Рассчитаем время, которое потребуется программному средству, чтобы открыть их базы данных данные сохранённого пользователя для входа в учётную запись веб-сервиса.

Для точности проведем пять опытов, в каждом из которых будем засекать моменты времени. Сначала засечем момент времени, когда пользователь нажимает на экран устройства для входа в учётную запись. Затем будем засекать момент времени, когда данные прочитались из базы данных.

Разница между этими двумя моментами времени и даст время чтения и отображения. Отслеживать время будем при помощи логов в консоли *AndoidStudio* (рисунок 3.2).

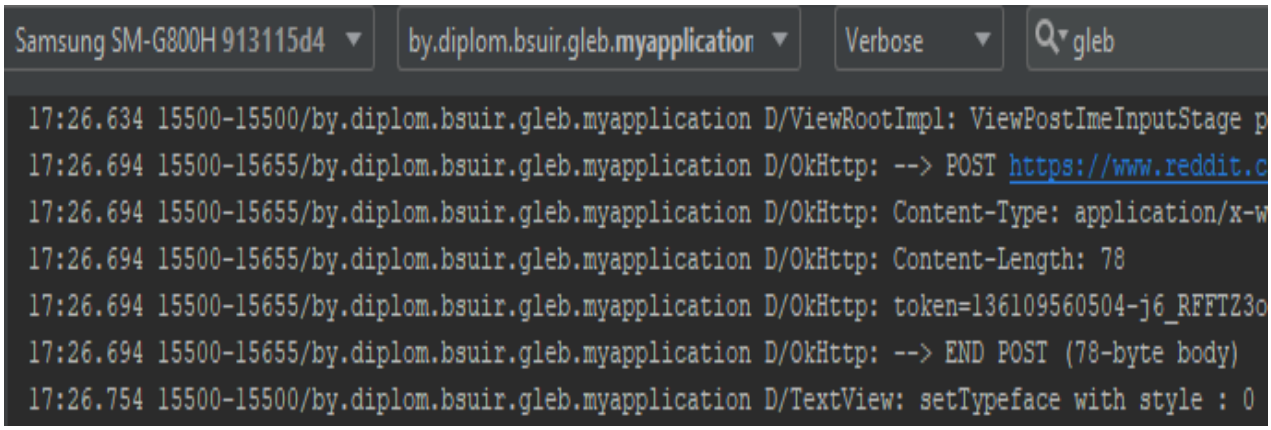

Рисунок 3.2 – Пример вывода результатов в консоль

В данном примере пользователь выполняет отправку запроса на веб-сервис используя данные учётной записи и сессионный ключ, которые сохранены в базе данный программного средства.

Рассчитывать время выполнения запроса чтения данных о учётных данных пользователя из базы данных будем по формуле 3.1.

Таблица 3.3 – Результаты опытов запросов загрузки данных

Номер опыта	Момент времени начала выполнения запроса $t_{нi}$, c	Момент времени после выполнения запроса t_{ci}, c
1	46,230	46,260
2	52,170	52,190
3	56,360	56,370
4	00,230	00,240
5	06,520	06,540

Таким образом на основе полученных результатов по формуле 3.1 рассчитаем T_c:

$$T_c = \frac{0{,}030 + 0{,}020 + 0{,}010 + 0{,}010 + 0{,}020}{5} = 0{,}018\,\text{с}.$$

В результате теста было выявлено, что программное средство выполняет задачу за 0,018с.

Данный результат является удовлетворительным и подтверждает информацию о том, что база данных *Realm* позволяет практически моментально считывать необходимые данные.

Аналогичным образом рассчитаем время добавление нового пользователя в базу данных. Для этого выведем в лог время начала транзакции и её закрытие. Отслеживать время будем при помощи логов в консоли *AndoidStudio* (рисунок 3.3).

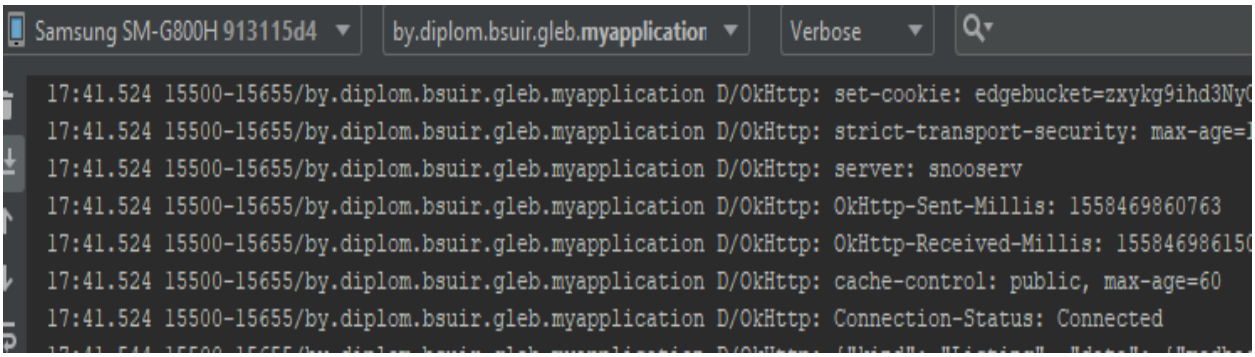

Рисунок 3.3 – Пример вывода результатов в консоль

Внесём полученные результаты для в таблицу для последующих расчётов.

Таблица 3.4 – Результаты опытов запросов создания пользователя

Номер опыта	Момент времени начала выполнения запроса $t_{нi}$, *c*	Момент времени после выполнения запроса t_{ci}, *c*
1	50,188	50,258
2	49,828	50,128
3	1,968	2,018
4	21,948	22,058
5	33,338	33,678

Таким образом, на основе полученных результатов, рассчитаем среднее время по формуле 3.1:

$$T_c = \frac{0{,}07 + 0{,}29 + 0{,}049 + 0{,}11 + 0{,}34}{5} = 0{,}171\text{с}.$$

Из опытов видно, что задержка между тем, когда пользователь делает какие-либо манипуляции с базой данных, время, между отправлением запроса и получением ответа очень мало. Этот эксперимент показывает, насколько быстрая нереляционная база данных *Realm*. Можно сделать вывод о том, что при использование *Realm* в конечном результате мы получаем быстродействие и эффективное программный продукт.

4 РУКОВОДСТВО ПО ЭКСПЛУАТАЦИИ ПРОГРАММНОГО СРЕДСТВА

4.1 Ввод в эксплуатацию программного средства

Основным способом распространения *Android* приложений является публикация в таких магазинах как:

– *Google Play*;
– *Amazon Appstore*;
– *Yandex market*;
– *Galaxy Market*;
– *Badoo Shop*;
– *Opera Mobile Store.*

Разработчик выкладывает программное средство в общий доступ, после чего потенциальный пользователь может бесплатно или за некоторую плату скачать приложения. На сегодняшний день лидирующую позицию занимает магазин *Google Play* и *Amazon Appstore*.

Amazon Appstore — альтернативный магазин приложений для Android. В нем гораздо меньше контента, чем в маркете Google Play, но зато есть большое количество эксклюзивов. Кроме того, Amazon очень часто проводит промо-акции и раздает десятки платных приложений и игр совершенно бесплатно.

Изначально Amazon попыталась выложить свой магазин в Google Play, но встретила непонимание со стороны Google. Приложение было удалено из официального маркета Android, но его можно установить с сайта Amazon [28].

Google Play (предыдущее название – *Android Market*) – магазин приложений, игр, книг, музыки и фильмов компании *Google* и других компаний, позволяющий владельцам устройств с операционной системой *Android* устанавливать и приобретать различные приложения (владельцам *Android*-устройств из Соединённых Штатов и России также доступно приобретение на *Google Play* книжных изданий, музыки, фильмов и периодики). Учётная запись разработчика, которая даёт возможность публиковать приложения, стоит $25. Платные приложения могут публиковать разработчики не из всех стран [29].

Для получения возможности размещать приложения в *Google Play*, необходим профиль разработчика.

С помощью промостраницы разработчика вы можете продвигать свой бренд и приложения в Google Play.

Если вы опубликовали в Google Play хотя бы одно приложение, то можете создать промостраницу разработчика. Пользователи будут заходить на нее в Play Маркете или по специальному URL, которым вы поделитесь с ними. На промостранице будет размещена информация о вашем бренде и приложениях, которые вы опубликовали в Google Play[30].

По умолчанию, при запуске приложения мы генерируем отладочный *APK*, который не может быть принят в *Google Play*. В *Google Play* каждое приложение должно быть подписано сертификатом, который содержит сведения о разработчике и секретный ключ, с помощью которого *Google Play* узнает, что *APK* загружается из правильного источника [31].

В *Android Studio* есть встроенный мастер для создания подписанного *APK*. Для того, чтобы подписать *APK* необходимо:

1 Открыть окно мастера, показано на рисунке 4.1.

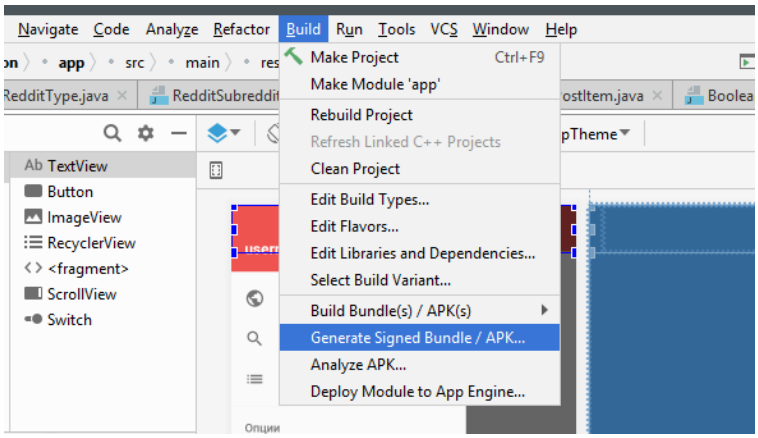

Рисунок 4.1 – Открытие окна матера создания *APK*

2 Создать *Key Store*, как показано на рисунке 4.2.

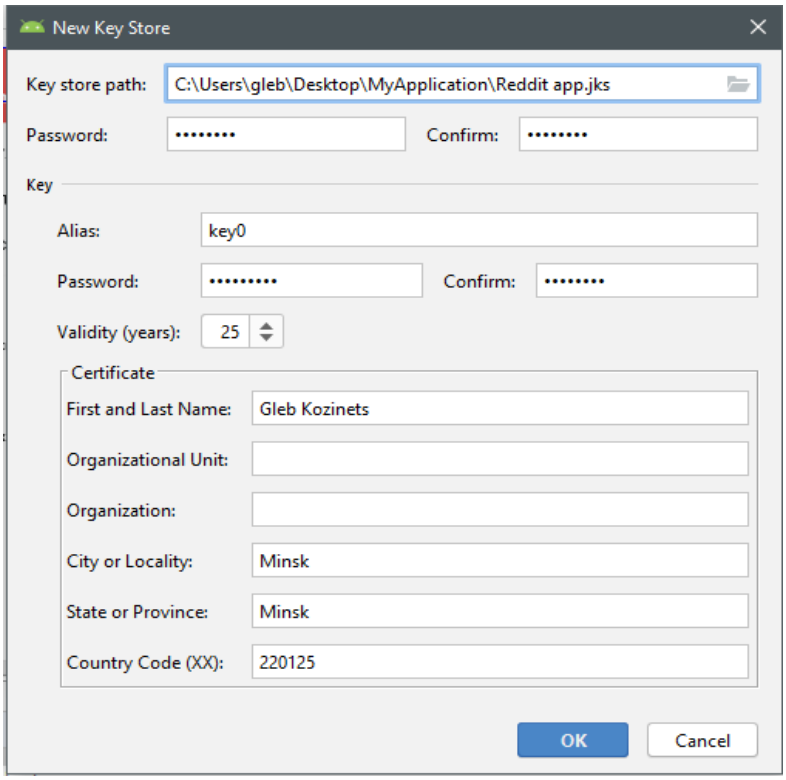

Рисунок 4.2 – Создание *Key Store*

3 Ввести мастер-пароль, как показано на рисунке 4.3.

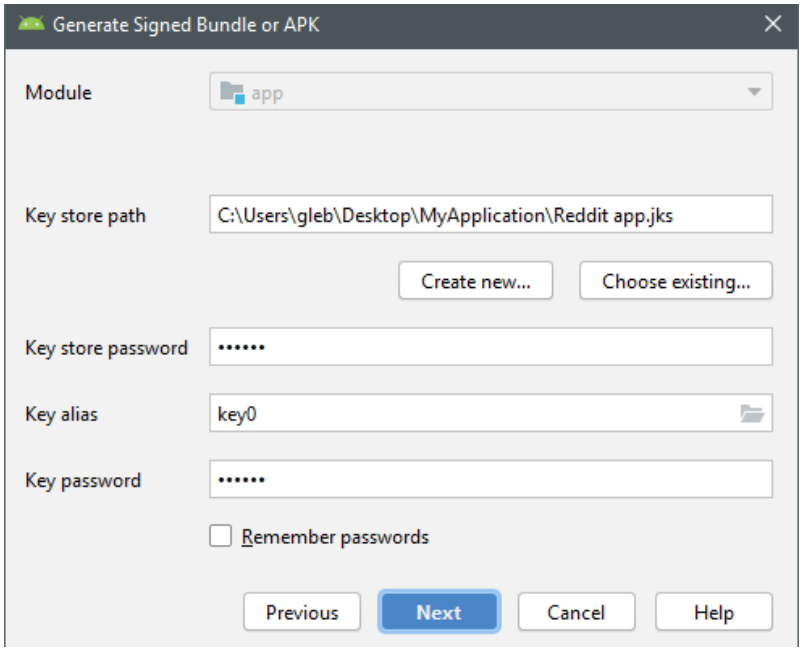

Рисунок 4.3 – Окно ввода пароля

4 Дождаться завершения создания *APK*.

После проделанных действий можно опубликовать приложение, следуя инструкциям на сайте *Google*. Когда приложение будет опубликовано, пользователи со всего мира смогут установить данное программное средство на свои мобильные устройства.

4.2 Руководство к пользованию разработанным программным средством

В руководстве пользователя будет объяснено, как пользоваться программным средством и какие функции оно имеет. Перед началом использования программы пользователю необходимо установить её на свой смартфон, который работает под системой *Android* версии не ниже 6.0.

При первом запуске в приложении не будет сохранено никаких данных о пользователе, поэтому будет использоваться гостевой профиль с настройками по умолчанию. Пользователь может продолжать пользоваться приложение даже, если у него нету учётной записи веб-сервиса *reddit.com*, но тогда будет закрыт функционал связанный с оценкой публикаций, просмотра профиля и отправки комментариев.

Для того, чтобы получить полный доступ к функциям приложения, пользователю необходимо пройти авторизацию. Для этого необходимо:

– открыть боковое меню;

– выбрать иконку профиля в верхней части экрана.

В открывшемся списке нужно нажать *Add an account* (рисунок 4.4).

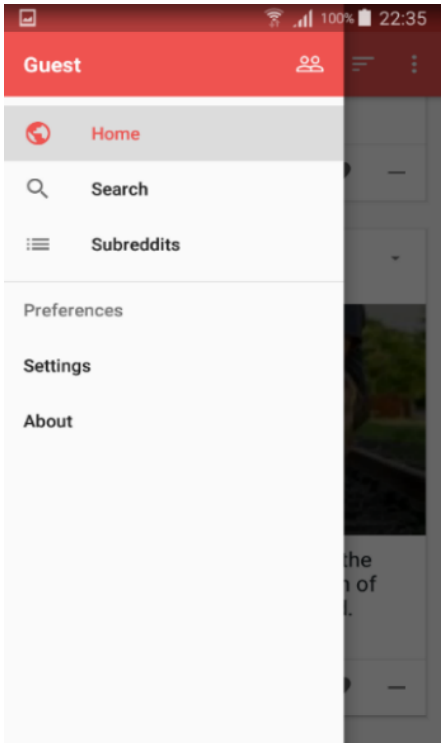

Рисунок 4.4 – Боковое меню

После нажатия на эту кнопку пользователь переходит на экран для ввода логина и пароля, когда данные введены, нужно нажать на кнопку *Login*, рисунок 4.5.

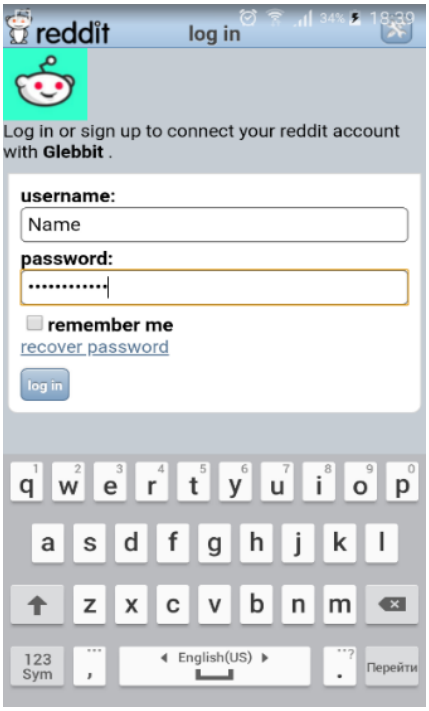

Рисунок 4.5 – Окно авторизации

На этом экране пользователю необходимо ввести свои логин и пароль для дальнейшего входа в свою учётную запись.

На главном экране пользователь может просматривать публикации выбранного раздела, которые отсортированы по популярности, рисунок 4.6.

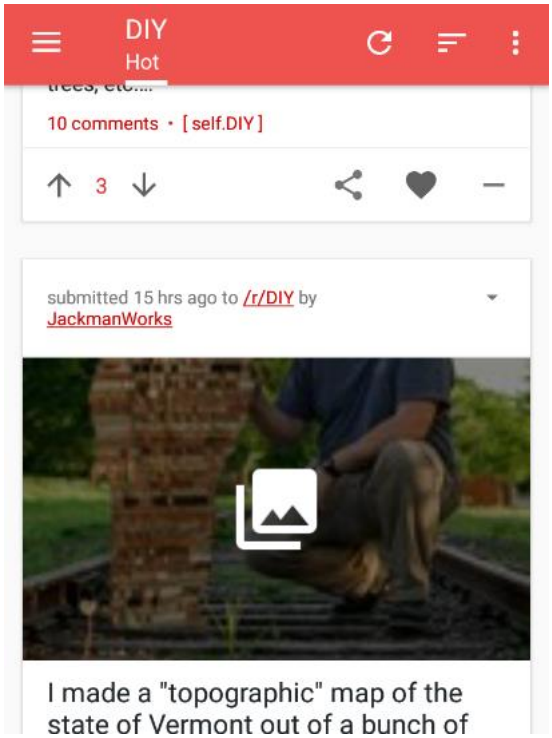

Рисунок 4.6 – Главный экран

Здесь можно увидель список загруженых публикаций, название текущего раздела и порядок сортировки, по которому выводится информация.

Для того чтобы выбрать другой тип сортировки пользователю необходимо нажать на иконку сортировки в верхнем правом углу экрана.

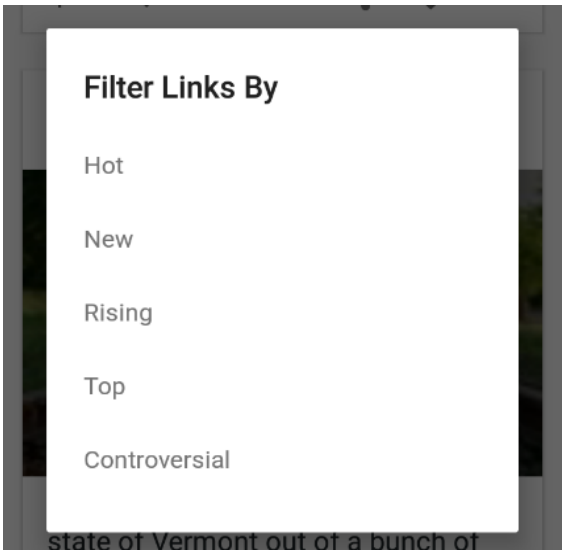

Рисунок 4.7 – Окно выбора типа сортировки

На открывшемся всплывающем окне нужно выбрать желаемый тип сортировки, после чего программа автоматически обновит список публикаций в выбранном порядке.

Если пользователь хочет вручную обновить список публикаций, то он может воспользоваться специальной кнопкой в верхней части экрана, слева от названия раздела, рисунок 4.8.

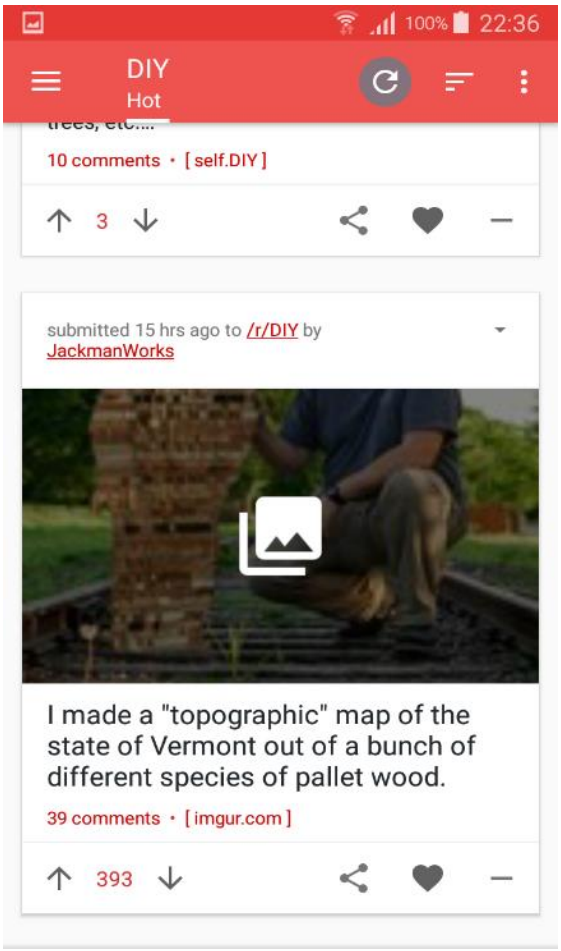

Рисунок 4.8 – Окно с кнопкой для обновления

При нажатии на эту кнопку программа сбрасывает данные о загруженных публикациях, после чего начинает повторную загрузку информации.

Для выбора других подразделов веб-сервиса пользователь может нажать на название текущего подраздела. На появившемся всплывающем окне пользователь может выбрать желаемый подраздел из представленного списка.

Если пользователь хочет узнать подробную информацию о существующих подразделах, пользователь может сделать это через дополнительный список.

Для просмотра иллюстраций прикреплённых к публикациям пользователь может нажать на желаемый рисунок прямо в списке публикаций. После нажатия выбранное изображение откроется на весь экран, что позволит лучше его рассмотреть

Для этого необходимо открыть боковое меню и выбрать пункт *Subreddits*, рисунок 4.9.

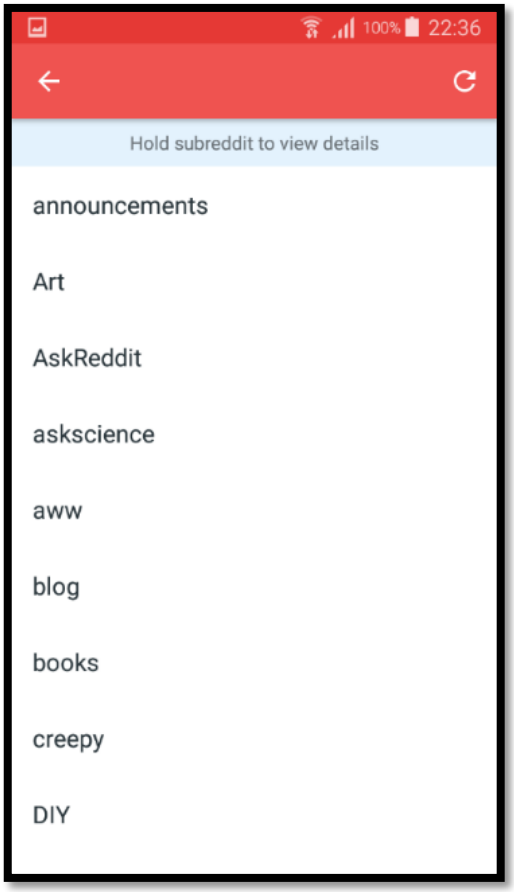

Рисунок 4.9 – Дополнительный список подразделов

В открывшемся окне можно увидеть список подразделов. Для просмотра дополнительной информации необходимо выполнить долгое нажатие на выбранный пункт списка. После долгого нажатия откроется экран с подробной информацией о разделе, в которой будет указано описание раздела, количество подписчиков, дата создания, количество публикаций, правила общения в данном раздели.

Для возвращения на предыдущее окно из текущего, пользователь может просто нажать кнопку возврата, которая находится под экраном смартфона или воспользоваться кнопкой перехода, которая находится в верхнем левом углу экрана.

Выйти из приложения можно, как и из всех программ в *Android*, кнопкой возврата, находящейся под экраном.

5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ *ANDROID*-КЛИЕНТА ДЛЯ ВЕБ-СЕРВИСА *REDDIT.COM*

5.1 Характеристика программного средства

Данное *Android*-приложение будет использоваться пользователями зарегистрированными в веб-сервисе *reddit.com* и использующие в качестве основного мобильного устройства *Android*-смартфон. Приложение позволяет получать доступ к веб-сервису способом, который является более удобным и быстрым, чем аналогичный вариант доступа к сервису через встроенный интернет браузер. Используя это приложение пользователи смогу взаимодействовать с основным функционалом веб-сервиса, получая информацию, которая была адаптирована под сенсорный экран смартфона.

Основными функциями данного программного средства являются:

– загрузка публикаций из выбранного раздела;

– чтение комментариев в выбранное публикации;

– просмотр рейтинга публикаций;

– отправка собственных комментариев к публикациям.

Программное средство было разработано для определённого веб-сервиса, благодаря этому приложение имеет только необходимый набор функций и использует меньше ресурсов смартфона, чем способ взаимодействия с веб-сервис, через встроенный интернет браузер.

Приложение будет размещаться в официальном магазине приложений для *Android*-смартфонов: *Google Play*. Базовая версия программного средства будет распространяться на бесплатной основе и имеет встроенные рекламные блоки, для получения дохода. Пользователи, для которых присутствие рекламы в приложении является неприемлемым, имеют возможность приобретения версии приложения без рекламы.

Обоснование экономической целесообразности инвестиций в разработку и использование данного программного средства осуществляется на основе расчёта и оценки следующих показателей:

− чистая дисконтированная стоимость (ЧДД);

− внутренняя норма доходности (ВНД);

− срок окупаемости инвестиций (ТОК);

− индекс рентабельности инвестиций ($Р_и$).

Разрабатываемое программное средство подлежит свободной реализации на рынке информационных технологий. Основное его предназначение – свободная продажа пользователям.

5.2 Расчёт затрат и отпускной цены программного средства

Для расчёта основной заработной платы исполнителей данного проекта применяется следующая формула:

$$З_о = К_{пр} \sum_{i=1}^{n} ЗП_{днi} \cdot t_{фi}, \quad (5.1)$$

где $К_{пр}$ – коэффициент премий, установленный за выполнение, перевыполнение плановых показателей ($К_{пр}$ = 1,3);

n – количество различных исполнителей, выполняющих соответствующую работу;

$ЗП_{днi}$ – дневная заработная плата i-го исполнителя, р.;

$t_{фi}$ – время фактической работы i-го рабочего по данному объекту, дней.

Для выполнения полного объема работ по разработке программного средства потребуется команда исполнителей, представленная в таблице 5.1.

Месячная заработная плата каждого из исполнителей взята в соответствии со средней зарплатой соответствующего специалиста в Республике Беларусь.

Таблица 5.1 – Исполнители разрабатываемого проекта

Исполнитель	Трудоёмкость работ, ч	Месячная заработная плата, р.
Дизайнер	1008	1200
Программист	1408	2100
Тестировщик	1008	1000

Дневная тарифная ставка рассчитывается путем деления месячной тарифной заработной платы каждого исполнителя на количество дней в месяце (22).

$$Т_{дн\ дизайер} = \frac{1200}{22} = 54{,}54 \text{ р./день},$$

$$Т_{дн\ прогр} = \frac{2100}{22} = 95{,}46 \text{ р./день},$$

$$Т_{дн\ Тест} = \frac{1000}{22} = 45{,}46 \text{ р./день}.$$

Таким образом, по формуле 5.1 высчитываем основную заработную плату исполнителей:

$$З_о = 1{,}3 \cdot (126 \cdot 54{,}54 + 176 \cdot 95{,}46 + 126 \cdot 45{,}46) = 40040{,}00 \text{ р.} \qquad (5.2)$$

5.3 Расчёт дополнительной заработной платы исполнителей

В данном разделе рассматриваются выплаты, предусмотренные законодательством Республики Беларусь о труде за неотработанное время. К ним относятся: оплата отпусков, льготных часов подростков и кормящих матерей, выполнение государственных обязанностей и прочие выплаты.

Указанные выплаты распределяются на себестоимость отдельных объектов по нормативу (в процентах к основной заработной плате производственных рабочих) и рассчитываются по следующей формуле:

$$З_д = \frac{З_о \cdot Н_д}{100} = \frac{40040{,}00 \cdot 20}{100} = 8008{,}00 \text{ р.} \qquad (5.3)$$

где $Н_д$ – норматив дополнительной заработной платы рабочих, % ($Н_д$ = 20%).

5.4 Расчёт отчислений на социальные нужды

В данном разделе рассматриваются отчисления на социальные нужды от фонда оплаты труда, предусмотренные законодательством Республики Беларусь.

Расчёт осуществляется по следующей формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100} = \frac{(40040{,}00 + 8008{,}00) \cdot 34}{100} = 16624{,}61 \text{р.} \qquad (5.4)$$

где $Н_{соц}$ – ставка отчислений в фонд социальной защиты, % ($Н_{соц}$ = 34%)

5.5 Расчёт прочих затрат

Расчёт накладных расходов осуществляется по следующей формуле:

$$Р_{пз} = \frac{З_о \cdot Н_{нак}}{100} = \frac{40040{,}00 \cdot 100}{100} = 40040{,}00 \text{р.} \qquad (5.5)$$

где $Н_{нак}$ – норматив накладных расходов, % ($Н_{нак}$ = 100%).

5.6 Расчёт производственной себестоимости

В расчёт себестоимости продукции (работ, услуг) входит стоимостная оценку используемых в процессе производства продукции (работ, услуг) природных ресурсов, сырья, материалов, топлива, энергии, основных средств, нематериальных активов, трудовых ресурсов, а также других затрат на ее производство и реализацию.

Производственная себестоимость проекта рассчитывается по следующей формуле:

$$С_{пр} = З_{о} + З_{д} + Р_{соц} + Р_{пр}, \quad (5.6)$$

$$С_{пр} = 40040{,}00 + 8008{,}00 + 16\,624{,}61 + 40040{,}00 = 104762{,}61\,\text{р.}$$

5.7 Расчет экономической эффективности реализации на рынке

Программное средство планируется разместить в магазине мобильных приложений, пользователи смогу загрузить бесплатную и платную версии приложений.

В среднем в год прогнозируется, что будет приобретаться 7000 платных версий программного средства по цене 5 руб., а доход от рекламы, размещаемой в бесплатной версии будет составлять 120000 руб.

Таким образом, разработчик получит доход в размере 35000 руб. на продаже платных версий, а также около 120000 за размещение рекламы в бесплатной версии приложения.

Следует также учесть затраты в год на модернизацию программного продукта.

Вычисляются они по формуле:

$$З_{м} = З_{с} \cdot В \cdot Н_{СЗ}, \quad (5.7)$$

где $З_{с}$ – зарплата сотрудника, руб.;

В – время работы, часы.

Налогооблагаемая прибыль, полученная разработчиком от реализации ПО на рынке, рассчитывается по формуле

$$П_{Н} = Ц \cdot N + Д - НДС - З_{В} - З_{П}, \quad (5.8)$$

где Д – доход от размещения рекламы на мобильном приложении, реализуемом бесплатно, руб.;

НДС – налог на добавленную стоимость, (20%);

$З_П$ – затраты на поддержание функционарования, руб.;

$З_В$ – затраты на вычислительные ресурсы, руб.;

Ц – цена на платную версию приложения, 5 руб.

Налог на добавленную стоимость рассчитывается по формуле:

$$НДС = Ц \cdot N + Д \cdot 0{,}2, \quad (5.9)$$

Налог на прибыль рассчитывается по формуле:

$$Н_П = П_Н \cdot 0{,}18, \quad (5.10)$$

Чистая прибыль рассчитывается по формуле:

$$П = П_Н - Н_П, \quad (5.11)$$

Таблица 5.2 – Расчет годовой чисто прибыли

Показатели	Значения
Выручка от реализации ПО	155000,00
НДС	-31000,00
Затраты на поддержания функционирования	-3028,50
Затраты на вычислительные ресурсы	-40000,00
Налогооблагаемая прибыль	446 585,14
Налог на прибыль	80971,50
Чистая прибыль	66396,63

Чистый поток наличности рассчитывается по формуле:

$$П_Н = П - З_М, \quad (5.12)$$

Для оценки эффективности проекта требуется вычислить чистый поток наличности с нарастающим итогом. Он вычисляется по формуле:

$$П_{НИ} = П_{Н1} - П_{Н2}, \quad (5.13)$$

где $П_{Н1}$ – чистый поток наличности за прошедший период, руб;

$П_{Н2}$ – чистый поток наличности за нынешний период, руб.

Так как приходится сравнивать разновременные результаты (экономический эффект) и затраты (инвестиции в разработку программного продукта), необходимо привести их к единому моменту времени – началу расчетного периода, что обеспечивает их сопоставимость.

Для этого необходимо использовать дисконтирование путем умножения соответствующих результатов и затрат на коэффициент дисконтирования соответствующего года *t*, который определяется по формуле:

$$\alpha_t = \frac{1}{(1+E_Н)^t}, \quad (5.14)$$

где $Е_Н$ – норма дисконта (в долях единиц), равная или больше средней процентной ставки по банковским депозитам действующей на момент осуществления расчетов: 15%;

t – порядковый номер года периода реализации инвестиционного проекта (предполагаемый период использования разрабатываемого ПО пользователем и время на разработку).

Чистый дисконтированный доход рассчитывается по формуле:

$$ЧДД = \sum_{t=1}^{n} (П_{Нt} \cdot \alpha_t - З_t \cdot \alpha_t), \quad (5.15)$$

где *n* – расчетный период, лет;

$П_{Нt}$ – результат (экономический эффект – прибыль или чистая прибыль), полученный в году *t*, руб.;

$З_t$ – затраты (инвестиции) (затраты на разработку (модернизацию) или на приобретение и внедрение ПО) в году *t*, руб.

Далее рассчитывается величина внутренней нормы доходности (ВНД), которая представляет собой норму (ставку, базу) дисконта (дисконтирования), при которой ЧДД становится равным нулю.

$$ВНД = \alpha_1 - ЧДД_1 \cdot \frac{\alpha_2 - \alpha_1}{ЧДД_2 - ЧДД_1}, \quad (5.16)$$

где α_1 – ставка дисконта, при которой ЧДД остается положительным;

α_2 – ставка дисконта, при которой ЧДД становится отрицательным;

$ЧДД_1$ – чистый дисконтированный доход при ставке дисконта r_1;

$ЧДД_2$ – чистый дисконтированный доход при ставке дисконта r_2.

Срок окупаемости проекта - момент, когда суммарный дисконтированный результат (эффект) станет равным (превысит) дисконтированную сумму инвестиций. Т.е. определяется через какой период времени инвестиционный проект начнет приносить инвестору прибыль.

Рентабельность инвестиций ($Р_u$) рассчитывается как отношение суммы дисконтированных результатов (эффектов) к осуществленным инвестициям:

$$Р_u = \frac{\sum_{t=1}^{n} П_{Нt} \cdot \alpha_t}{\sum_{t=1}^{n} З_t \cdot \alpha_t}, \quad (5.17)$$

Таблица 5.3 – Расчет эффективности инвестиционного проекта по разработке программного обеспечения

Показатели	Шаги расчета			
	0	1	2	3
Чистая прибыль		66396,63	66396,63	66396,63
Чистый поток наличности	-104 762,61	48 796,63	48 796,63	48 796,63
Чистый поток наличности с нарастающим итогом		-55 965,98	-7 169,35	41 627,28
Ставка дисконта	15,00%			
Коэффициент дисконтирования	1,00	0,87	0,76	0,66

Продолжение таблицы 5.3

Показатели	Шаги расчета			
	0	1	2	3
Дисконтированный чистый поток наличности	-104762,608	42431,85	36897,26	32084,58
Чистый дисконтированный доход	6651,08			
Внутренняя норма доходности	18,79%			
Рентабельность инвестиций	1,06			
Срок окупаемости	2 года 2 месяца			

Таким образом, все затраты на разработку программного средства окупятся через два года и два месяца от реализации программного продукта при распространении через интернет-магазин приложений.

Следовательно, реализация программного средства для размещения и анализа услуг спортивных клубов на рынке является экономически эффективной и его разработку целесообразно осуществлять.

ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования было решено улучшить взаимодействие между пользователями и веб-сервисом *reddit.com* путём разработки мобильного приложения для операционной системы *Android*. В процессе исследования были рассмотрены существующие виды корпоративных социальных порталов и социальных сетей, перечень необходимых и самых популярных функций, что помогло разобраться в том, какой именно функционал является наиболее важным для подобного рода системы и на чём стоит сконцентрироваться при разработке в первую очередь.

Также были исследованы основные принципы и технологии, применяемые при разработке высоконагруженных облачных систем, были выбраны самые актуальные компоненты и библиотеки для разработки продукта.

Данное программное средство написано на языке *Java* для операционной системы *Android* и имеет ряд следующих функций:

– просмотр публикаций;
– оценка публикаций;
– чтение комментариев;
– авторизация с помощью *OAuth*;
– изменение настроек профиля.

В результате разработки цель была достигнута – было спроектировано и разработано приложения для доступа к веб-сервису *reddit.com*, позволяющее многократно повысить эффективность взаимодействия пользователей. Процесс разработки и сама готовая система были подробно описаны с использованием различных диаграмм, а также было создано удобное руководство пользователя, которое позволит любому разобраться в системе без особых проблем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] *Meet Android Studio* [Электронный ресурс]. – Режим доступа: *https://developer.android.com/studio/intro*.

[2] Интерфейс среды разработки [Электронный ресурс]. – Режим доступа: *https://www.fandroid.info/urok-2-kotlin-sozdanie-proekta-v-android-studio-i-zapusk-prilozheniya-na-android-ustrojstve/*.

[3] *Android Studio release notes* [Электронный ресурс]. – Режим доступа: *https://developer.android.com/studio/releases/index.html*.

[4] Блинов И. Н. *Java*. Промышленное программирование : практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск : УниверсалПресс, 2007. – 704 с.

[5] Интерфейс *LinkedIn* [Электронный ресурс]. – Режим доступа: *https://downdetector.com/status/linkedin*.

[6] Что такое *LinkedIn* [Электронный ресурс]. – Режим доступа: *https://semantica.in/blog/chto-takoe-linkedin.html.*

[7] Что такое *Twitter* [Электронный ресурс]. – Режим доступа: https://www.kasper.by/blog/ twitter-i-dlya-chego-on-nuzhen-kompanii/.

[8] Сервис распространения [Электронный ресурс]. – Режим доступа: *https://store.google.com/*.

[9] Реалистичный *Realm*. 1 год опыта [Электронный ресурс]. – Режим доступа: *https://habr.com/ru/post/*328418/.

[10] Оптимизация кода с помощью *Butter Knife* [Электронный ресурс]. – Режим доступа: *https://ru.smedialink.com/razrabotka/optimizatsiya-koda-s-pomoshhyu-butter-knife/*.

[11] Пример использования *Retrofit* 2 в приложениях *Android* [Электронный ресурс]. – Режим доступа:*http://javaway.info/ispolzovanie-retrofit-2-v-prilozheniyah-android/*.

[12] Особенности *Retrofit* 2 в приложениях *Android* [Электронный ресурс]. – Режим доступа: *https://habr.com/ru/post/351890/*.

[13] Что такое такое *rest api* [Электронный ресурс]. – Режим доступа: *https://toster.ru/q/*136265.

[14] Модуль записи тестов *Espresso* [Электронный ресурс]. – Режим доступа: *http://developer.alexanderklimov.ru/android/debug/espresso.php*.

[15] *Android Studio*: качественные приложения *Android* [Электронный ресурс]. – Режим доступа: *https://developer.android.com/distribute/best-practices/develop/build-with-android-studio?hl=ru*.

[16] Что такое *Activity* [Электронный ресурс]. – Режим доступа: *https://developer.android.com/guide/components/activities/?hl=ru*.

[17] Жизненный цикл на *Android* [Электронный ресурс]. – Режим доступа: *http://developer.alexanderklimov.ru/android/theory/lifecycle.php*

[18] *Material Design* для *Android* [Электронный ресурс]. – Режим доступа: *https://developer.android.com/design/material?hl=ru*.

[19] Основы *Gradle* [Электронный ресурс]. – Режим доступа: *https://javarush.ru/groups/posts/2126-kratkoe-znakomstvo-s-gradle*.

[20]Сравнение работы баз данных Электронный ресурс]. – Режим доступа: *https://jetruby.com/ru/blog/realm-mobilnaya-baza-dannyh/*.

[21]Файл *AndroidManifest.xml* [Электронный ресурс]. – Режим доступа: *http://developer.alexanderklimov.ru/android/theory/AndroidManifestXML.php*.

[22] Макеты [Электронный ресурс]. – Режим доступа: *https://webhamster.ru/mytetrashare/index/mtb*189/14906146443*cuohwb6fa*.

[23]Принципы разработки пользовательского интерфейса [Электронный ресурс]. – Режим доступа: *https://studfiles.net/preview/3654276/page:3/*.

[24]Основы разработки интерфейса [Электронный ресурс]. – Режим доступа: *https://studfiles.net/preview/3654276/page:5/*.

[25]Простота интерфейса [Электронный ресурс]. – Режим доступа: *https://pandia.ru/text/78/164/51745.php*.

[26] *OAuth* понятным языком [Электронный ресурс]. – Режим доступа: *https://habr.com/ru/company/mailru/blog/*115163/.

[27] Что такое *OAuth* [Электронный ресурс]. – Режим доступа: *https://dic.academic.ru/dic.nsf/ruwiki/*515170.

[28]Для кого *Amazon Store* [Электронный ресурс]. – Режим доступа: *https://www.iguides.ru/main/gadgets/google/how_to_install_amazon_appstore/*.

[29] Начало работы с *Google Play* [Электронный ресурс]. – Режим доступа: *https://support.google.com/googleplay/answer/topic=*3364260.

[30]страница разработчика [Электронный ресурс]. – Режим доступа: *https://support.google.com/googleplay/androiddeveloper/answer/6226441*.

[31] Подписанный *APK* в *Android* [Электронный ресурс]. – Режим доступа: *http://java-help.ru/android-studio-create-signed-apk/*.

ПРИЛОЖЕНИЕ А
(обязательное)
Отчёт по анализу пояснительной записки на предмет заимствования материала

Анализ пояснительной записки дипломного проекта на предмет заимствований был выполнен на сайте *antiplagiat.ru*. Результаты анализа приведены на рисунке А.1.

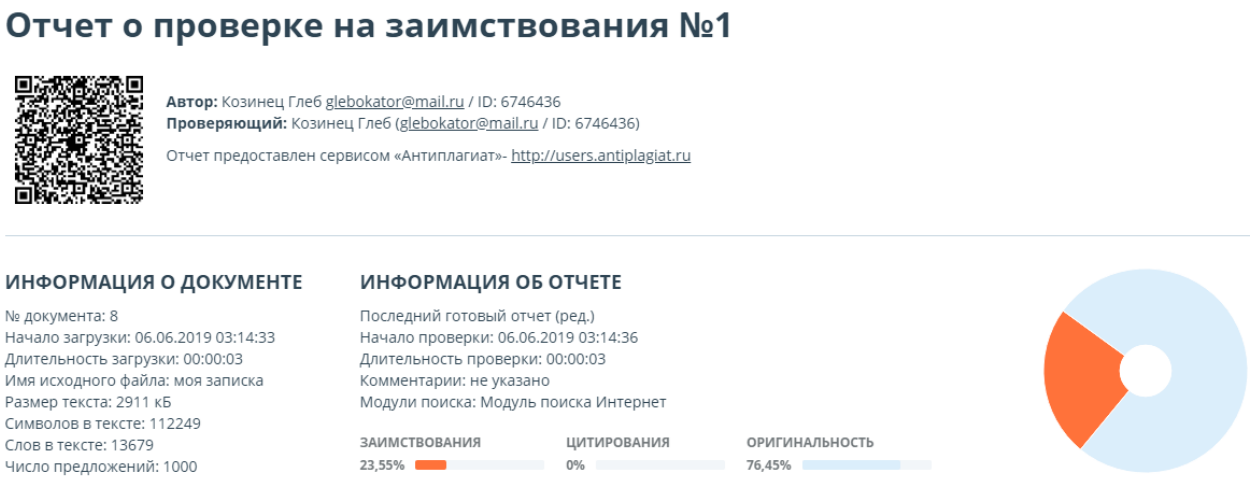

Рисунок А.1 – Отчёт о проверке на заимствования пояснительной записки

Анализ показал, что пояснительная записка дипломного проекта удовлетворяет требованиям к уникальности материала.

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинги программ

```
package by.diplom.bsuir.gleb.myapplication.ui.common.auth;

import android.content.Context;
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.webkit.CookieManager;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.FrameLayout;
import android.widget.Toast;

import com.afollestad.materialdialogs.MaterialDialog;
import by.diplom.bsuir.gleb.myapplication.R;
import by.diplom.bsuir.gleb.myapplication.data.managers.presenters.AuthPre-
senterImpl;
import by.diplom.bsuir.gleb.myapplication.data.managers.storage.db.Database-
Helper;
import by.diplom.bsuir.gleb.myapplication.data.managers.storage.db.Data-
baseManager;
import by.diplom.bsuir.gleb.myapplication.data.models.api.red-
dit.auth.AuthPrefs;
import by.diplom.bsuir.gleb.myapplication.data.models.api.red-
dit.auth.AuthWrapper;
import by.diplom.bsuir.gleb.myapplication.data.models.db.Prefs;
import by.diplom.bsuir.gleb.myapplication.data.models.db.User;
import by.diplom.bsuir.gleb.myapplication.data.network.api.reddit.base.Red-
ditServiceBase;
import by.diplom.bsuir.gleb.myapplication.ui.App;
import by.diplom.bsuir.gleb.myapplication.ui.AppComponent;
import by.diplom.bsuir.gleb.myapplication.ui.common.auth.views.AuthView;
import by.diplom.bsuir.gleb.myapplication.ui.common.base.BaseActivity;
import by.diplom.bsuir.gleb.myapplication.ui.common.main.MainActivity;
import by.diplom.bsuir.gleb.myapplication.ui.common.utils.widgets.DialogUtil;
import by.diplom.bsuir.gleb.myapplication.ui.common.views.BaseContextView;

import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;

import butterknife.ButterKnife;
import butterknife.InjectView;
import io.realm.RealmChangeListener;
import io.realm.RealmResults;

public class AuthActivity extends BaseActivity implements AuthView {

    public static final String RESULT_USER_NAME = "result_user_name";
    public static final String RESULT_USER_ID = "result_user_id";
    public static final String REQUEST_NEW_ACCOUNT = "request_new_account";
```

```
    public static final String REQUEST_APP_OAUTH = "request_app_oauth";

    @InjectView(R.id.web_view)
    WebView mContentView;
    @InjectView(R.id.loading_view)
    FrameLayout loadingLayout;

    private AuthComponent authComponent;
    private RealmChangeListener realmChangeListener;
    private RealmResults<User> users;

    private boolean resultIncluded;
    private boolean requestAppOAuth;
    private String userId;
    private String userName;

    @Inject
    App app;
    @Inject
    DatabaseManager databaseManager;
    @Inject
    AuthPresenterImpl authPresenter;
    @Inject
    DialogUtil dialogUtil;



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ButterKnife.inject(this);
        checkIntent(getIntent());
        setUpRealm();
        if(!requestAppOAuth){
            setUpWebView();
        }else{
            authPresenter.getAccessToken();
        }
    }

    private void checkIntent(Intent intent) {
        //check whether result expected or not
        resultIncluded = intent.getBooleanExtra(REQUEST_NEW_ACCOUNT, false);
        //check if it's requesting an app oauth
        requestAppOAuth = intent.getBooleanExtra(REQUEST_APP_OAUTH, false);
    }

    private void setUpRealm() {
        realmChangeListener = () -> {
            //changes made in this context are related to the session being
set
            transitionToMainActivity(resultIncluded, true);
        };
        getRealm().addChangeListener(realmChangeListener);
    }

    private void setResult(boolean isSuccess) {
        Intent resultIntent = new Intent();
```

```
    if(userId != null && userName != null){
        resultIntent.putExtra(RESULT_USER_NAME, userName);
        resultIntent.putExtra(RESULT_USER_ID, userId);
    }
    setResult(isSuccess ? RESULT_OK : RESULT_CANCELED, resultIntent);
}

private void setUpWebView() {
    clearCookies(CookieManager.getInstance());
    mContentView.getSettings().setBuiltInZoomControls(true);
    mContentView.getSettings().setDisplayZoomControls(false);
    mContentView.getSettings().setJavaScriptEnabled(true);
    mContentView.loadUrl(getAuthUrl());
    mContentView.setWebViewClient(new WebViewCustomClient());
}

private void clearCookies(CookieManager manager){
    if(manager.getCookie(getAuthUrl()) != null){
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            manager.removeSessionCookies(value -> {});
        }else{
            manager.removeSessionCookie();
        }
    }
}

@Override
protected void setupComponent(AppComponent appComponent) {
    authComponent = DaggerAuthComponent.builder()
                    .appComponent(appComponent)
                    .authModule(new AuthModule(this))
                    .build();
    authComponent.inject(this);
}

@Override
public AppComponent component() {
    return authComponent;
}

@Override
public DialogUtil getDialogUtil() {
    return null;
}

@Override
protected int getLayoutId() {
    return R.layout.activity_auth;
}

@Override
protected int getContainerId() {
    return 0;
}

@Override
protected RealmChangeListener getRealmSessionChangeListener() {
    return null;
```

```
    }

    @Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);
    }

    @Override
    protected void onDestroy() {
        authPresenter.unregisterForEvents();
        getRealm().removeAllChangeListeners();
        ButterKnife.reset(this);
        super.onDestroy();
    }

    @Override
    public void onResume() {
        super.onResume();
        authPresenter.registerForEvents();
        getRealm().addChangeListener(realmChangeListener);
    }

    @Override
    public void showLoading() {
        loadingLayout.setVisibility(View.VISIBLE);
        mContentView.setVisibility(View.GONE);
    }

    @Override
    public void hideLoading() {
        mContentView.setVisibility(View.VISIBLE);
        loadingLayout.setVisibility(View.GONE);
    }

    @Override
    public void showInfoMessage() {
        mContentView.setVisibility(View.VISIBLE);
    }

    @Override
    public void showErrorMessage(String error) {
        mContentView.setVisibility(View.VISIBLE);
        if(resultIncluded){
            transitionToMainActivity(resultIncluded, false);
        }else{
            app.getToastHandler().showToast(error, Toast.LENGTH_LONG);
        }
    }

    @Override
    public BaseContextView getContentContext() {
        return getBaseActivity();
    }

    @Override
    public void transitionToMainActivity(boolean resultIncluded, boolean is-
Success) {
        if(resultIncluded){
```

```
                            authPrefs.setShowTrending(prefs.isShowTrend-
ing());
                            authPrefs.setStoreVisits(prefs.isStoreVisits());
                            authPrefs.setMedia(prefs.getMedia());
                            authPrefs.setHighlightControversial(prefs.isHigh-
lightControversial());
                            authPrefs.setIgnoreSuggestedSort(prefs.isI-
gnoreSuggestedSort());

                            wrapper.setAuthPrefs(authPrefs);
                            break;
                        }
                    }
                    authPresenter.updateSession(wrapper);
                })
                .negativeText("No, thanks")
                .onNegative((dialog, which) -> showPrefer-
encesSyncDialog(wrapper))
                .cancelable(false)
                .canceledOnTouchOutside(false)
                .show();
    }

    private void showPreferencesSyncDialog(AuthWrapper wrapper) {
        dialogUtil.build()
                .title("Use preferences from your Reddit account?")
                .positiveText("Yes")
                .negativeText("No")
                .onPositive((dialog, which) -> authPresenter.updateSes-
sion(wrapper))
                .onNegative((dialog, which) -> {
                    wrapper.getAuthPrefs().setToDefault();
                    authPresenter.updateSession(wrapper);
                })
                .cancelable(false)
                .canceledOnTouchOutside(false)
                .show();
    }


    public static Intent intent(Context context, boolean isLauncher){
        Intent intent = new Intent(context, AuthActivity.class);
        //common flag
        intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
        //only if true
        if(isLauncher){
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_AC-
TIVITY_CLEAR_TASK);
            intent.putExtra(REQUEST_APP_OAUTH, true);
        }else{
            intent.putExtra(REQUEST_NEW_ACCOUNT, true);
        }
        return intent;
    }
}
```

ПРИЛОЖЕНИЕ В
(обязательное)
Ведомость дипломного проекта