Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инновационного непрерывного образования

Кафедра электронных вычислительных машин

Отчет по преддипломной практике

Студент	В.Ю. Буряк
Руководитель	В.В. Захаренко
Консультант от кафедры ЭВМ	В.В. Марченко
Нормоконтролер	А.С. Сидорович

МИНСК 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развлечения в жизни каждого человека играют очень важную роль. Ведь без отдыха и позитивных эмоций очень трудно сосредоточиться на действительно серьёзных вещах, таких как, например, семья или работа. Всем время от времени нужно отдыхать, расслабляться и получать позитивные эмоции, проецируя их в дальнейшем на все остальные сферы жизни.

Современный мир предлагает огромное множество вариантов для развлечений, включающее в себя как активный, так и пассивный отдых. Однако современный человек – занятой человек. У него есть много дел, которые нельзя откладывать в долгий ящик. Именно поэтому многие выбирают для себя просмотр фильмов, сериалов, мультфильмов и прочего медиаконтента в качестве основного способа отдохнуть, расслабиться и отвлечься.

Однако бывает очень трудно выбрать для себя фильм или сериал из того огромного множества, которое на сегодняшний момент существует в мире кинематографа. Люди тратят на это много времени, но и это не гарантирует того, что выбранный контент не разочарует вас и вы не потратите время в пустую, вместо того, чтобы полностью насладиться просмотром и отдохнуть. Как раз эту проблему и должно решить приложение, которое разрабатывается в рамках данного дипломного проекта. Подразумевается разработка именно мобильного приложения, так как на сегодняшний момент трудно встретить человека, у которого бы его не было. Мобильные устройства стали незаменимыми помощниками человека во всех сферах жизни.

Целью данного дипломного проекта является разработка и реализация программного средства для мобильной платформы Android, которое позволит производить поиск медиаконтента, учитывая предпочтения зрителя, основываясь на его истории просмотров и поставленных оценках. Целевой аудиторией этого приложения являются люди, любящие кинематограф, однако его может использовать абсолютно любой человек, которому понадобилось найти для себя что-то в этом огромном мире кино.

В соответствии с поставленной целью были определены следующие задачи:

- выбор версий мобильной платформы Android;
- выбор поставщика информации;
- реализация схемы получения и доставки информации от поставщика к пользовательскому приложению, а также протокола передачи этой информации;
- разработка серверной части приложения в виде отдельного модуля;
- разработка базы данных для серверной части приложения;
- разработка и реализация схемы доставки информации от серверной части приложения к пользовательскому, а также протокола обмена информацией между этими частями;

– разработка модуля формирования предпочтений зрителя;
– разработка пользовательского приложения.

Программное средство будет состоять из двух частей: мобильного приложения и веб-сервера, которые будут предоставлять следующие функции:

– регистрация пользователя;
– авторизация пользователя;
– поиск медиаконтента;
– добавление медиаконтента в список просмотренных;
– оценка медиаконтента;
– аудит истории просмотров и оценок;
– получение рекомендаций к просмотру.

Поддерживая все вышеперечисленные функции, программное средство сможет обеспечивать пользователя всей необходимой информацией о мире кинематографа, а также выступать в качестве надёжного личного консультанта по миру кинематографа.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор существующих аналогов.

На этапе проектирования системы были тщательно изучены существующие аналоги. Одним из наиболее похожих примеров является мобильное приложение «Кинопоиск» (рисунок 1.1):

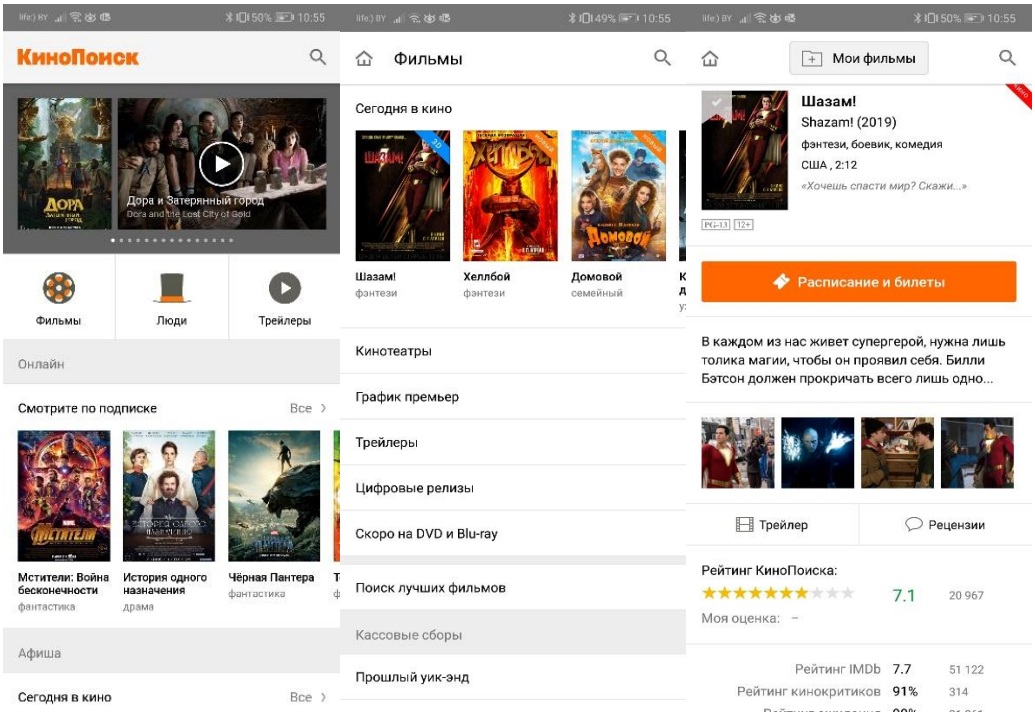

Рисунок 1.1 – Мобильное приложение «Кинопоиск»

Данное мобильное приложение обладает большим функционалом и позволяет производить поиск фильмов по различным категориям. Плюсом приложения можно выделить возможность просмотреть трейлер к фильму или сериалу. Личный профиль зрителя также присутствует. Присутствует и возможность отмечать фильмы как просмотренные, а также ставить им оценки.

Однако приложение не позволяет формировать личные предпочтения и получать рекомендации к просмотру, можно лишь просмотреть список похожих фильмов, выбрав его в библиотеке. Пользовательский интерфейс слишком загружен информацией, которую невозможно скрыть из отображения. Приложение в основном ориентировано на русскоязычную аудиторию. Также существует и веб-версия этого ресурса, практически аналогичная по своему содержанию и функционалу. Стоит также отметить, что в веб-версии этого приложения присутствует возможность просмотреть медиаконтент целиком, а не только трейлер к нему.

Еще одно похожее приложение - «IMDb Movies & TV» (рисунок 1.2). Это мобильное приложение очень похоже на «Кинопоиск». Приложение ориентировано на охват большей целевой аудитории, поддерживая множество

мировых языков, однако в нём нет возможности установки русскоязычного интерфейса. Также в данном приложении присутствует возможность получать личные рекомендации к просмотру. Существует и веб-версия приложения, аналогичная по своему функционалу.

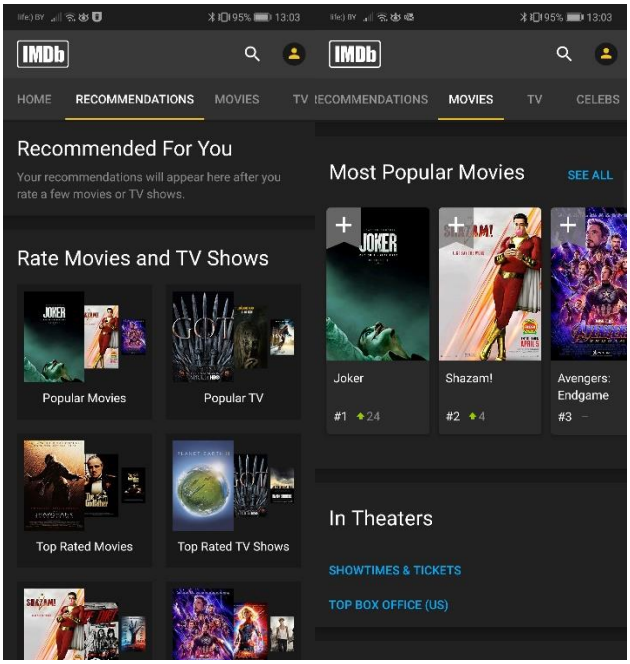

Рисунок 1.2 – Мобильное приложение «IMDb Movies & TV»

«MyShows» - в отличие от сервисов, которые описаны ранее, «MyShows» посвящён сугубо сериалам (рисунок 1.3).

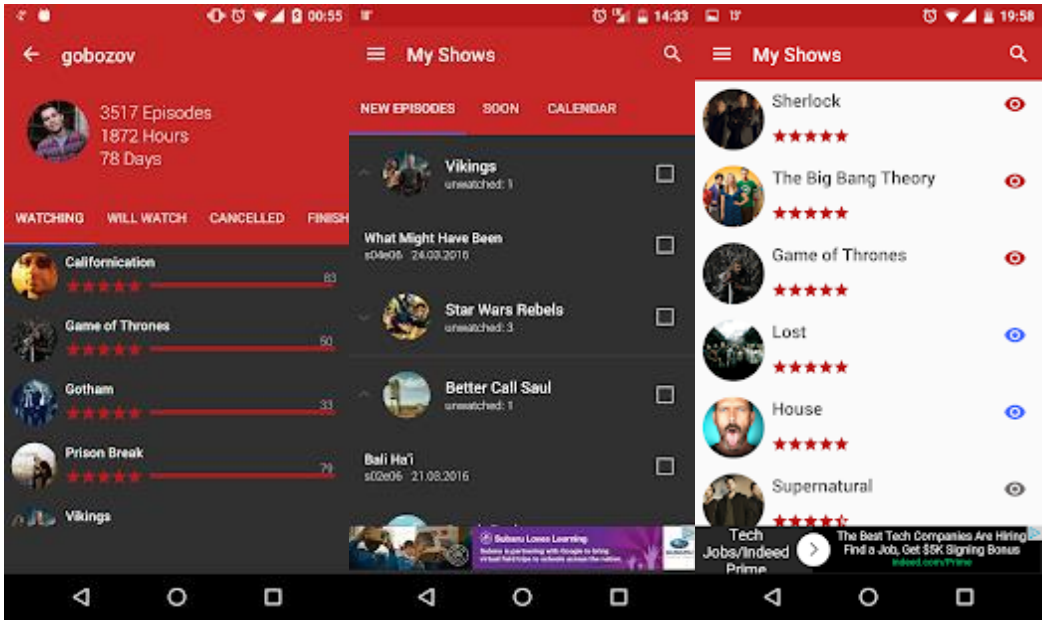

Рисунок 1.3 – Мобильное приложение «MyShows»

Это приложение не источник информации о фильмах и другом контенте, а простой и удобный трекер, который позволяет отслеживать зрительский прогресс. Из информации в приложении можно найти только описания сериалов и названия серий на русском языке. Есть также рейтинг всех шоу на основе популярности среди участников MyShows. Каждый пользователь может отмечать, сколько эпизодов каких сериалах он уже посмотрел, а также телепроекты на будущее. В результате приложение показывает количество оставшихся и просмотренных эпизодов в каждом шоу вместе с потраченным на просмотр временем. Функционала, позволяющего получать список рекомендаций к просмотру в данном сервисе нет.

1.2 Аналитический обзор.

Мобильное приложение – программное обеспечение, предназначенное для работы на смартфонах, планшетах и других мобильных устройствах. Многие мобильные приложения предустановлены на самом устройстве или могут быть загружены на него из онлайн-магазинов приложений, таких как App Store, Google Play и другие.

Для разработки мобильных приложений под операционную систему Android используется язык программирования Java, однако он не является единственным возможным вариантом. Тем не менее это традиционный выбор для таких приложений.

Java является одним из самых популярных языков программирования. Язык имеет большую историю - первый релиз Java состоялся ещё в 1995 году - поэтому не удивительно, что существует огромное количество учебников и специализированных сайтов, посвящённых Java-разработке. В качестве ярких примеров можно привести наиболее распространённые [1] и [2].

Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируется в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры. Виртуальная машина Java (Java Virtual Machine, далее JVM)) - основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE). Виртуальная машина Java интерпретирует байт-код, предварительно созданный из исходного текста программы компилятором.

Одним из главных свойств Java, обеспечившим ему огромную популярность, является переносимость. Философия Java провозглашает принцип «Write once - run everywhere», указывающий на то, что, в теории, Java-код может быть исполнен на любой JVM, работающей на любом устройстве. В реальной жизни, конечно, данный принцип работает с большими оговорками, однако это не мешает успешной работоспособности JVM более чем на трёх миллиардах устройств. Скорее всего, благодаря заманчивой перспективе переносимости, именно Java был выбран в качестве языка программирования для операционной системы (ОС) Android. Философия

платформы включает в себя варианты использования ОС на любом устройстве с любой микропроцессорной архитектурой, будь то ARM, x86 или любая другая

Одно из главных преимуществ платформы Android — ее открытость. Операционная система Android построена на основе открытого исходного кода и находится в свободном распространении. Это позволяет разработчикам получить доступ к исходному коду Android и понять, каким образом реализованы свойства и функции приложений.

Любое Android-приложение, содержит в себе хотя бы один из так называемых App Components - компонентов приложения. Существуют четыре вида основных компонентов приложения: Activity, Service, Content Provider и Broadcast Receiver.

В связи с большой популярностью мобильных разработок, существует большое количество источников информации по Android-разработке. Основным источником, в первую очередь, является официальная документация [3]. Помимо технической документации комплекта средств разработки (Software Development Kit, SDK), официальный сайт содержит множество примеров, руководства по реализации стандартных компонентов, пошаговые рекомендации по разработке приложений с нуля, а также требования к дизайну.

Одной из наиболее популярных книг является [4]. Данная книга представляет собой практический курс по написанию программного обеспечения на базе второй версии Android SDK. Несомненно, книга сильно устарела на данный момент, однако всё ещё является хорошим структурированным руководством по изучению платформы. Поскольку множество приложений создается с условием совместимости с более старыми версиями ОС Android, многие примеры из данной книги не потеряли актуальности.

Активити представляет собой один экран с пользовательским интерфейсом. Все пользовательские активити являются классами, унаследованными от стандартного класса Activity. Каждая активити обладает определённым жизненным циклом (рисунок 1.1), который является хорошей демонстрацией специфики мобильных приложений. Поскольку ресурсы мобильных устройств весьма ограничены, достоверно можно сказать лишь то, что в памяти существуют лишь те объекты, которые так или иначе относятся к отображаемому на экране содержимому. Как только активити скрывается с экрана, например, если пользователь сворачивает приложение кнопкой «Домой», активити переходит в состояние «остановлено» (stopped). Начиная с этого момента, ОС может уничтожить все объекты в оперативной памяти, относящиеся к данной активити, включая саму активити. Исходя из этого очень важно вовремя сохранять состояние приложения. Второй особенностью активити является то, что каждый раз при смене ориентации устройства с ландшафтной на портретную или наоборот, активити уничтожается и создаётся заново. Сложно сказать, чем продиктовано подобное решение,

скорее всего это сделано для того, чтобы обеспечить поддержку различных вариантов пользовательского интерфейса для различной ориентации устройства. Данная особенность порождает вторую важную проблему - проблему сохранения состояния при изменении конфигурации (например, при повороте экрана). Подробнее о работе с активностью можно прочитать в специализированном разделе документации [3].

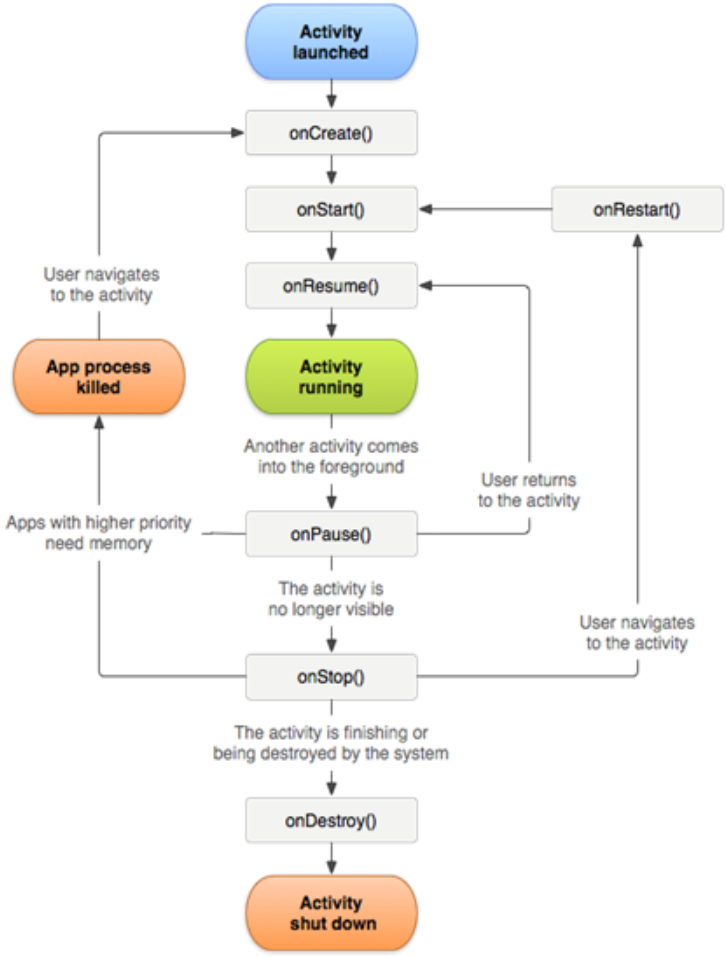

Рисунок 1.1 – Жизненный цикл активити.

Сервис - это компонент, который работает в фоне и используется для длительных и ресурсоёмких операций. Отличительной особенностью сервиса является возможность запуска в привилегированном режиме (foreground mode), благодаря которому ОС не завершает приложение в фоне максимально долгое время. Несмотря на такую возможность, разработчикам крайне не рекомендуется злоупотреблять ею и наоборот, завершать сервисы как можно раньше. Привилегированный режим обычно используется для записи или воспроизведения потокового содержимого, а также для скачивания или загрузки файлов. Помимо упоминавшихся ранее источников, хорошим руководством по работе с сервисами является [5].

Ещё одним часто используемым компонентом является Broadcast Receiver (в русскоязычной литературе встречается перевод «широковещательный приемник»). Broadcast receiver - это компонент, служащий для обработки широковещательных сообщений. ОС Android использует специальные объекты для сообщения между компонентами - интенты (intent). Иногда в литературе встречается прямой перевод - «намерения». Интент - это объект класса Intent представляющий собой некоторое сообщение. С каждым интентом может ассоциироваться некий код ошибки, действие, URI (Uniform Resource Identifier), а также произвольный набор пересылаемых объектов. Объекты, так называемые extras, хранятся в ассоциативном контейнере с уникальными строковыми ключами. Поддерживаются все примитивные типы, строки, а также объекты, реализующие интерфейсы Serializable или Parcelable. Интенты могут быть узконаправленными, например, для запуска сервиса или активити, или же широковещательными, обычно используемые в качестве нотификации о каком-либо событии. Интенты широко используются в ОС Android: благодаря широковещательным интентам, с помощью широковещательных приемников можно подписываться на различные системные события, от срабатывания будильника и до перехода по ссылке в браузере.

Последним нерассмотренным компонентом является поставщик контента - content provider. Это, пожалуй, наименее часто используемый компонент, служащий для предоставления доступа к некоторому хранилищу данных. Существуют системные поставщики контента для доступа к телефонным контактам, получения информации об установленных приложениях и многого другого. Доступ осуществляется с помощью URI по аналогии с доступом к веб-страницам. Низкая популярность данного компонента связана с его специфичностью: далеко не каждое приложение имеет необходимость в предоставлении доступа к своим данным.

Широковещательные приёмники и поставщики контента хорошо описаны во многих источниках, например, на популярном русскоязычном ресурсе, посвящённом изучению программирования для ОС Android [6].

Помимо изучения работы с основными компонентами, каждый разработчик должен ознакомится с основными шаблонами проектирования ОС Android. Наиболее часто используемыми стандартными шаблонами проектирования являются наблюдатель (observer) и слушатель (listener). Также широко используются такие принципы как обмен сообщений и позднее связывание. Помимо этого, Android диктует свои правила при реализации некоторых стандартных процессов, тем самым заставляя разработчикам следовать определённым специфическим шаблонам проектирования. Ярким примеров этого могут служить загрузчики (loaders). Загрузчики являются потомками класса Loader и служат для асинхронной загрузки данных. Главным преимуществом загрузчиков является то, что их жизненный цикл не зависит от других компонентов. Для доступа к загрузчикам используется специальный компонент Loader Manager, который позволяется работать с

загрузчиками любым классам, реализующим интерфейс Loader Manager Loader Callbacks. Несмотря на некоторые встроенные во фреймворк реализации, разработчикам довольно часто приходится реализовывать свои специфические загрузчики, например, для работы с локальной базой данных или получения массивов информации по сети. Хорошую статью о практической реализации загрузчиков можно найти по адресу [7]. Там же можно отыскать и другие полезные статьи по специфическим шаблонам проектирования.

Несмотря на довольно динамичное развитие SDK и внедрение новых шаблонов проектирования, в последнее время издавалось не много литературы с упором на актуальные версии средств разработки. В качестве одного из примеров, можно привести [8]. Книга во многом схожа с [2], однако базируется на современной версии платформы и отдельно затрагивает тему обеспечения совместимости с использованием стандартных компонентов.

На данный момент весьма перспективно выглядит IDE от Google - Android Studio. Она позиционируется как специализированное средство для Android-разработки от компании разработчика самой ОС. Android Studio основывается на InelliJ IDEA и обладает всеми её достоинствами. В добавок к этому Android Studio содержит специализированные средства разработки и анализа кода, специфичные для Android-разработки. Пожалуй, единственным недостатком Android Studio является то, что она официально всё ещё находится в стадии бета-тестирования. При разработке данного проекта использовалась именно эта IDE. Подробнее ознакомится с данным продуктом можно с помощью [9]. Там же можно найти подробные описания процесса миграции проектов на новою IDE с других популярных решений.

Отличительной особенностью Android Studio является использование постепенно набирающей популярность системы сборки Gradle. Её описание можно найти на официальном сайте [10] или же на ресурсе [11], применительно конкретно к Android Studio. К особенностям Gradle, выгодно отличающей её от других систем сборки, стоит отнести сплав концепции «build by convention», используемой во всех популярных аналогах, например, Maven, с использование скриптов на языке Groovy. Подобный подход позволяет легко создавать многомодульные сборки, что весьма актуально для Android-проектов, использующих пользовательские библиотеки. Ещё одной возможностью Gradle, активно используемой в Android Studio является возможность создавать основанные на Gradle проекты, при этом фактически всё, что необходимо - это пара стандартных скриптов для построения и скачиваемый из общедоступного репозитория модуль для их обработки. Подобная лаконичная структура весьма эффектно смотрится при использовании системы контроля версий: фактически все файлы проекта являются функционально необходимыми и не захламляют репозиторий.

Стоит отметить, что все выше перечисленные среды разработки либо являются полностью бесплатными, либо имеют бесплатные версии.

В заключении следует упомянуть некоторые источники, специфичные для данного дипломного проекта. В проекте активно используется взаимодействие с базой данных (БД). На ОС Android в качестве БД использует SQLite. SQLite является простой, легковесной реляционной базой данных, используемой на всех популярных мобильных платформах (iOS, Windows Phone). Помимо официальной документации [3], описывающей шаблон проектирования для работы с БД, полезным является официальная документация БД [12]. SQLite обладает определёнными особенностями, ознакомление с которыми необходимо даже при наличии опыта работы с другими реляционными БД. SQLite содержит всего лишь четыре типа данных и не поддерживает хранимые процедуры. Вдобавок к этим ограничениям, добавляется и то, что входящие в Android SDK средства для работы БД не способствуют созданию даже самой простой структуры БД с зависимостями между таблицами, всячески подталкивая к хранению данных в нескольких больших, несвязанных таблицах.

Следует отметить, что отличительной чертой Android-разработки, с точки зрения обзора литературы, является большое число интернет-ресурсов. Это обусловлено, во-первых, популярностью платформы, а, во-вторых, особенностями мобильной разработки. Несмотря на наличие официальной документации и нескольких основных книг по разработке, на практике существует большое число нюансов, которые обычно не рассматриваются в официальных источниках.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Проанализировав необходимый к реализации функционал, были выработаны требования к проектируемой системе. Для реализации системы было решено разбить архитектуру приложения на несколько составных модулей. Это позволит вносить изменения в существующий код приложения, которые не будут затрагивать остальные модули системы, а воздействовать лишь на свой блок. Тем самым обеспечим архитектуру не только модульностью, но и масштабируемостью.

Масштабируемость – это способность системы расширяться, наращивать функционал и увеличивать свою производительность.

Мобильное приложение разрабатывается под операционную систему Android. Перед началом разработки очень важно выбрать версию операционной системы, под которую разрабатывается приложение. Операционная система Android является системой с обратной совместимостью. Это означает, что приложения, которые были написаны используя старую версию SDK (Software Development Kit), гарантированно будут поддерживаться на последней версии системы, а также на версиях системы, которые будут выходить в будущем. Ознакомившись с данными, иллюстрирующими распространенность версий ОС мобильных устройств на данный момент, выбор был сделан в пользу версии Android 5.0, которая используются на 80.2% устройств (рисунок 2.1).

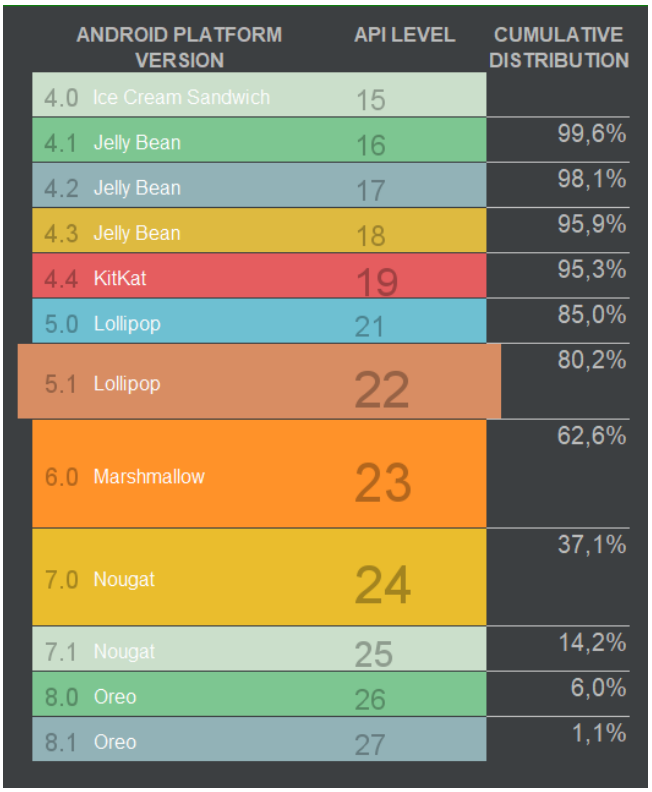

Рисунок 2.1 – Распространенность версий ОС Android.

В разрабатываемом мобильном приложении можно выделить несколько составных модулей(блоков):

- модуль ресурсов;
- модуль представлений;
- модуль данных;
- модуль обработки событий;
- модуль взаимодействия с API.

Модуль ресурсов представляет собой файлы в формате XML, которые использует операционная система Android для хранения следующих видов данных:

- описание анимаций приложения;
- данные приложения (цвета приложения, стили приложения, строковые данные, размеры и отступы);
- иконки и изображения;
- описания меню;
- разметка экранов пользовательского интерфейса.

Этот модуль обеспечивает приложению гибкость настройки. Так, можно настроить поддержку нескольких языков в приложении, создав для каждого языка свой строковый ресурс или быстро заменить описанные выше ресурсы, не вмешиваясь в исходный код приложения.

Модуль представлений – это модуль, содержащий в себе всё необходимое, для отображения пользовательского интерфейса и управления им. Ключевым компонентом пользовательского интерфейса в Android является activity (активность). Одна активность обычно представляет собой один экран приложения, на котором пользователь может взаимодействовать с ним. Активности описываются в виде Java-класса, унаследованного от системного класса android.app.Activity, переопределяя его методы. Также этот модуль контролирует состояние этих активностей с помощью наблюдения за их жизненным циклом. Для инициализации компонентов активности, модуль представлений обращается к ресурсам, описанным в модуле ресурсов.

Модуль данных – является модулем, который хранит в себе модели данных в виде Java-объектов. Этот модуль предоставляет другим модулям структурированные данные о некотором объекте и методы для управления этими данными. Модели данных не зависят от пользовательского интерфейса, а лишь предоставляют для него данные к отображению. В некоторых моделях данных описаны части бизнес-логики приложения, которая затрагивает только поведение той модели, в которой она описана.

Модуль обработки событий взаимодействует с модулем представлений, модулем данных и модулем взаимодействия с API, отлавливая события, которые необходимо в дальнейшем обработать определённым образом. Такое поведение обеспечивается благодаря методам обратного вызова, описанных в других модулях. Эти методы вызываются при возникновении некоторого

события (нажатие кнопки пользователем, ответ от сервера на посланный запрос, поворот экрана, закрытие приложения).

Модуль взаимодействия с API: представляет собой интерфейс, который обращается по протоколу HTTP к поставщику контента за необходимой информацией и получает её в формате JSON. Затем, через взаимодействие с модулем обработки событий обрабатывает полученные данные и преобразует ее в вид POJO (Plain Old Java Objects) для модели данных. После того как обновятся модели данных, происходит взаимодействие с модулем представлений через модуль обработки событий. Модуль представлений в свою очередь обновляет содержимое пользовательского интерфейса. Для реализации этого модуля используется сторонняя библиотека Retrofit, которая позволяет описать лишь интерфейс взаимодействия с API, без необходимости реализовывать обмен данными по протоколу HTTP стандартными средствами Java.

Разрабатываемое программное средство подразумевает под собой не только мобильное приложение, но также и серверную часть, обслуживающую его. Приложения под Android традиционно разрабатываются на языке программирования Java. Для удобства и совместимости, серверная часть также будет разработана на этом языке программирования.

Серверная часть приложения отвечает за хранение и предоставление пользовательской информации, такой как имя пользователя, его пароль, история просмотров и оценок пользователя. Также посредством обращения пользовательского мобильного приложения к веб-серверу, происходит авторизация пользователя в системе.

Серверная часть реализуется с использованием Spring Framework. Spring – универсальный фреймворк с открытым исходным кодом для Java-платформы. Spring можно рассматривать как целый набор мини-фреймворков, большинство из которых могут работать независимо друг от друга.

Использование Spring и его мини-фреймворков при реализации серверной части программного средства позволит реализовать все модули, необходимые для работы приложения:

– Spring Security для авторизации пользователя;
– Spring Boot для простого и быстрого разворачивания приложения на сервере;
– Spring Data для работы с базой данных приложения.

Серверная часть содержит в себе следующие модули:

– веб-сервер приложения;
– модуль обработки запросов;
– модуль бизнес-логики;
– модуль работы с БД;
– база данных;

Веб-сервер приложения представляет собой контейнер сервлетов с открытым исходным кодом Apache Tomcat. Выбор именно этого веб-сервера

обусловлен тем, что он наиболее совместим с Java, и считается стандартом в разработке веб-приложений на этом языке программирования. Веб-сервер получает запросы по протоколу HTTP и передаёт эти запросы модулю обработки запросов.

Модуль обработки запросов получает запрос от веб-сервера и проверяет возможность обработки этого запроса. Если же запрос валиден и обработать его можно, то модуль обработки запросов извлекает из него всю необходимую информацию (например параметры) и передает дальнейшее управление обработкой запроса модулю бизнес-логики. Этот модуль также отвечает за отдачу информации обратно клиенту (если требуется). После обработки запроса Spring MVC автоматически формирует ответ клиенту в сконфигурированном заранее на сервере формате, например в формате JSON, который отлично подходит под задачи передачи объектов приложения между двумя модулями. Этот модуль является RESTful сервисом. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Модуль бизнес-логики предназначен для выполнения конкретных операций над данными приложения, взаимодействуя при этом с модулем базы данных. Этот модуль представляет собой слой, который находится по середине между слоем сервиса (модуль обработки запросов) и слоем данных (модуль базы данных). Поэтому в нем находится реализация всех функций, которые должны присутствовать в серверной части программного средства, разрабатываемого в рамках дипломного проекта. Здесь описаны процессы авторизации пользователя в приложении, получение информации о просмотрах и оценках, а также формируются индивидуальные рекомендации к просмотру для пользователя.

Модуль работы с базой данных взаимодействует с базой данных приложения, модифицируя или извлекая данные для других слоев. Предоставляет другим модулям интерфейс, через который они могут общаться с базой данных, не зная ничего о её структуре и наполнении. Такая архитектура позволяет изменять структуру базы данных, не изменяя бизнес-логику и архитектуру других модулей и не нарушая целостности системы. При изменении структуры данных необходимо лишь изменить интерфейс, который предоставляется данным модулем другим слоям для взаимодействия с данными из базы. В этом модуле используется Spring Data, которая позволяет еще больше упростить процесс работы с базой данных, реализуя доступ к сложным SQL-запросам через простое описание в названии метода, который используется для вызова этого запроса.

База данных - набор сведений, хранящихся некоторым упорядоченным способом. В серверной части приложения было решено использовать MySQL в качестве СУБД. Плюсы использования именно этой СУБД:

- полнофункциональная версия является полностью бесплатной;
- может запускаться на любой ОС;
- MySQL поддерживает большинство функционала SQL;

– большое количество функций, обеспечивающих безопасность, которые поддерживается по умолчанию;

– MySQL легко работает с большими объемами данных и легко масштабируется;

– упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

База данных приложения содержит в себе несколько таблиц, связанных между собой и описывающих информацию о пользователях, а также данные их личных профилей, такие как историю просмотров и оценок. Пароли пользователей хранятся в зашифрованном виде, чтобы обеспечить конфиденциальность информации, а также защитить пользователей от утечки личных данных.

Обе части архитектуры приложения (пользовательское мобильное приложение и серверная часть) реализованы в рамках архитектуры MVC (Model-View-Controller). Этот шаблон проектирования предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Под компонентом следует понимать некую отдельную часть кода, каждая из которых играет одну из ролей: Контроллера, Модели или Представления, где Модель служит для извлечения и манипуляций данными приложения, Представление отвечает за видимое пользователю отображение этих данных, а Контроллер управляет всем процессом взаимодействия компонентов между собой, обрабатывая события. Такая архитектура позволяет модифицировать каждый компонент системы независимо друг от друга.

Поставщик информации - также является частью структурной модели программного средства, однако представляет собой сторонний ресурс, предоставляющий доступ по ключу к данным о медиаконтенте (API), который можно получить в формате JSON. При необходимости, модуль взаимодействия с API обращается на этот ресурс, отсылая ему запрос определенного формата по протоколу HTTP. Поставщик информации обрабатывает этот запрос и отсылает клиенту ответ, содержащий запрашиваемую информацию о медиаконтенте.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.304 С1.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробно функционирование мобильного приложения. Для этого проведем анализ основных модулей программы и рассмотрим их зависимости. Также проанализируем все модули, которые входят в состав программного кода и рассмотрим назначение всех переменных, полей и методов этих модулей.

В состав мобильного приложения входят следующие модули:

- модуль взаимодействия с API;
- модуль представлений;
- модуль данных;
- модуль ресурсов

Большинство приложений под мобильную операционную систему Android содержат в себе один или несколько экранов, с которыми взаимодействует пользователь. В мобильном приложении, которое реализовывается в рамках данного дипломного проекта, также будет содержаться несколько таких экранов. Каждый экран приложения описывается классом, унаследованным от системного класса `Activity`.

У каждой `Activity` есть свой жизненный цикл, которым управляет операционная система. Это означает, что конкретный экран приложения может находиться в разное время в разных состояниях, в зависимости от того, как пользователь взаимодействует с этим приложением. Каждое такое состояние соответствует определенному этапу жизненного цикла. Класс `Activity` предоставляет возможность сообщать экрану приложения о том, что его состояние изменилось посредством методов обратного вызова, которые вызываются при соответствующих переходах на другой этап жизненного цикла `Activity`. Переопределяя эти методы в классах, которые описывают экран приложения, разработчик получает возможность контролировать переходы между различными этапами жизненного цикла, обрабатывая эти события необходимым ему образом.

Модуль представлений является модулем, содержащим в себе все `Activity` приложения, а также логику его поведения при переходах между различными этапами своего жизненного цикла.

3.1 Класс MainActivity. Основные компоненты.

При запуске мобильного приложения, первым делом пользователь попадает на главный экран приложения, представленный классом `MainActivity`. На этом экране у пользователя есть возможность воспользоваться следующим функционалом приложения:

- поиск медиаконтента;
- просмотр списка рекомендаций к просмотру;
- просмотр списка популярного на текущий момент медиаконтента;
- просмотр информации о личном профиле;

– переход к экрану аудита истории просмотров и оценок;
– переход к экрану изменение информации личного профиля;
– выход из личного профиля.

Класс `MainActivity` является наследником класса `AppCompatActivity`, который, в свою очередь, через другие иерархические связи, в конечном итоге наследуются от класса `Activity`. Благодаря такому наследованию, появляется возможность переопределить методы класса `Activity`, которые вызываются при переходах между этапами жизненного цикла главного экрана приложения.

Перед отображением экрана пользователю, вызывается метод `onCreate()`, в котором необходимо определить и проинициализировать элементы пользовательского интерфейса. Все элементы пользовательского интерфейса описаны в специальном модуле ресурсов в виде файлов в формате XML. Так, в модуле ресурсов находятся несколько пакетов, включая пакет layout, в котором хранятся ресурсы разметки пользовательского интерфейса для каждого экрана приложения.

В методе `onCreate()`, который переопределён в классе `MainActivity`, вызывается специальный метод супер-класса `setContentView()`, в который передается идентификатор ресурса, содержащего разметку необходимого экрана приложения. Для экрана `MainActivity` этот ресурс определяется идентификатором `R.layout.activity_main`. Передав его в качества аргумента методу `setContentView()`, в качестве пользовательского интерфейса будет установлена разметка, содержащаяся в переданном ресурсе.

При запуске приложения на данном этапе, графический интерфейс успешно отобразится на экране мобильного устройства, однако для взаимодействия с компонентами этого интерфейса внутри программного кода, необходимо проинициализировать все его элементы в классе `MainActivity`. Для описания этих элементов используются специальные объекты, унаследованные от стандартного класса `View`. Эти элементы также инициализируются в методе `onCreate()`, который будет вызван при создании пользовательского экрана, до его отображения конечному пользователю. Для их инициализации необходимо объявить каждый компонент интерфейса, содержащийся в разметке `R.layout.activity_main`, отдельным полем в глобальной области видимости класса.

Для `MainActivity` такими полями являются:
– private TabLayout tabLayout;
– private ViewPager viewPager;
– private ViewPagerAdapter adapter;
– private Toolbar toolbar;
– public SearchView searchView.

Поля класса ассоциируются с компонентами интерфейса из разметки с помощью метода Activity `findViewById()`. В качестве аргумента этот метод принимает уникальный идентификатор компонента, определенный в модуле ресурсов. Таким образом связываются все компоненты пользовательского интерфейса во всех классах приложения, описывающих Activity.

Одним из компонентов MainActivity является элемент `SearchView`. Этот элемент используется для организации удобного поиска медиаконтента в приложении. `SearchView` инициализируется в отдельном этапе жизненного цикла приложения, который вызывает метод обратного вызова `onCreateOptionsMenu(Menu menu)`. Обусловлено это тем, что этот элемент является специальным компонентом, называемым меню. Такие меню описываются специальным одноименным пакетом в модуле ресурсов, также в виде разметки XML.

Этапы инициализации компонента `SearchView`:

– вызов специального метода `Activity getMenuInflater()`, который возвращает объект типа `MenuInflater`, использующийся для инициализации ресурса меню;

– вызов метода `inflate()` объекта `MenuInflater`, на вход которому подается файл ресурса, в котором описывается меню;

– инициализация объекта `MenuItem`, описывающего меню;

– инициализация `SearchView` путем приведения типа объекта `MenuItem` к типу `SearchView`.

– установка обработчика события на компонент.

В методе `onCreate()` также инициализируются и другие компоненты, использующиеся в реализации всей необходимой функциональности главного экрана, в том числе и фрагменты, о которых будет подробно сказано в разделе 3.2.

3.2 Класс MainActivity. Фрагменты главного экрана приложения.

В `MainActivity` реализована большая часть пользовательского функционала. Все элементы для его использования очень трудно разместить на одном экране, так как на это просто не хватит пространства экрана смартфона. Для решения этой проблемы было решено использовать так называемые Фрагменты (Fragments). Фрагменты, как и `Activity`, представляют собой контейнеры, в которых можно разместить элементы пользовательского интерфейса. Однако использование фрагментов позволяет не отображать их на экране до тех пор, пока это не потребуется пользователю. В нужный момент, который определяет разработчик специальной логикой, фрагменты вставляются в определенные места `Activity`, отображаясь на экране для пользователя. Места, в которые вставляются фрагменты – это специальные контейнеры, одним из которых является `ViewPager`, который и используется в `MainActivity`.

Для комфортной навигации между фрагментами через элемент ViewPager, также был добавлен контейнер TabLayout, объявленный в классе MainActivity. Этот контейнер представляет собой удобное для пользователя меню навигации между фрагментами, отображая на специальной панели имя текущего отображаемого фрагмента (рисунок 3.1).

ViewPager отвечает за показ фрагментов и их прокрутку, используя специальный адаптер ViewPagerAdapter. Этот адаптер предоставляет элементу ViewPager информацию о том, какой именно фрагмент в текущий момент времени должен быть отображен в нем, анализируя поведение пользователя (пользователь взаимодействует с этой областью экрана, проводя пальцем или стилусом вправо, или влево по нему).

Фрагменты, которые используются на экране MainActivity приложения:

– MovieSearchFragment;
– UserProfileFragment;
– RecommendationsFragment.

Фрагменты, как и Activity, должны наследоваться от специального класса. Для них таким классом является класс Fragment. При обнаружении класса, который наследуется от Fragment, операционная система понимает, какое именно поведение задать этому компоненту. Фрагменты очень похожи на Activity:

– инициализация проводится также, как и для Activity;
– графические интерфейс также описывается в модуле ресурсов;
– имеют свой личный жизненный цикл.

После инициализации всех фрагментов, можно проводить инициализацию специального адаптера ViewPagerAdapter и отображать фрагменты на главном экране приложения с помощью элемента ViewPager. Этапы инициализации фрагментов и отображения их в ViewPager:

– объявление и инициализация объектов типа Fragment;

– объявление и инициализация объекта ViewPager, с помощью вызова метода Activity findViewById(), в который передаётся уникальный идентификатор компонента ViewPager;

– объявление и инициализация объекта ViewPagerAdapter при помощи передачи в его конструктор объекта FragmentManager, который можно получить при помощи вызова метода Activity getSupportFragmentManager();

– добавление фрагментов в адаптер ViewPagerAdapter при помощи последовательных вызовов его метода addFragment(), в качестве параметров которому передаются два аргумента: один из фрагментов, необходимый к отображению в ViewPager, а также строка, содержащая заголовок (название) этого фрагмента, который будет отображаться пользователю;

– установка адаптера в `ViewPager` при помощи вызова его метода `setAdapter()`, в который передается `ViewPagerAdapter`.

– инициализация `TabLayout` при помощи вызова его метода `setupWithViewPager()`, в который передается инициализированный элемент `ViewPager`;

– вызов метода `setSupportActionBar()`, в который передается объект `ToolBar` пользовательского интерфейса, являющегося «шапкой» приложения, и содержащего заголовок (название) приложения, а также элементы меню.

После инициализации `MainActivity` и его фрагментов, главный экран приложения готов к отображению конечному пользователю (рисунок 3.2.1).

Рисунок 3.1 – Главный экран приложения после инициализации компонентов.

3.3 Фрагмент MovieSearchFragment.

Как было сказано ранее, фрагменты обладают жизненным циклом также, как и `Activity`. Инициализацию фрагмента и связку его компонентов графического интерфейса с его файлом ресурсов, нужно проводить на самом раннем этапе жизненного цикла фрагмента, который определяется вызовом метода `onCreateView()`.

Полями класса `MovieSearchFragment` являются:

– protected static RecyclerView recyclerView;
– SwipeRefreshLayout swipeRefreshLayout;
– public static List<Movie> moviesList;
– protected static List<Movie> filteredMovieList;
– static MovieApi apiService;
– static MovieRecyclerViewAdapter recyclerViewAdapter;
– static boolean loading;
– ProgressBar progressBar;

Объект RecyclerView является компонентом, который отвечает за отображения списка фильмов в фрагменте. Этот элемент обладает крайне гибкой настройкой, о которой будет сказано ниже. Каждый элемент такого списка – это отдельный компонент View, файлом ресурса которого является файл `res/layout/item_movie.xml`. Внешний вид проинициализированного элемента представлен на рисунке 3.2.

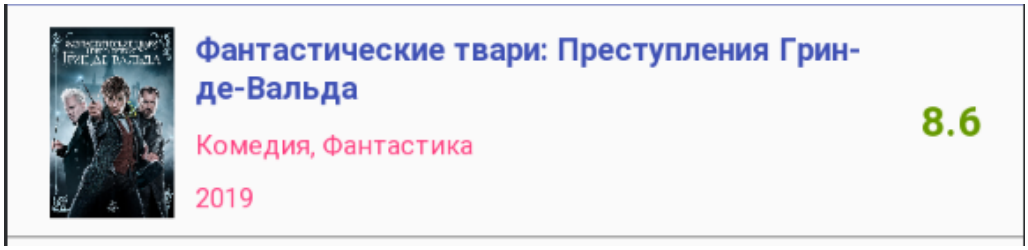

Рисунок 3.2 – Элемент списка res/layout/item_movie.xml.

Таких элементов в списке может быть много, и за загрузку элементов в список `RecyclerView` отвечает специальный адаптер, который представлен отдельным классом `MovieRecyclerViewAdapter`, являющийся наследником класса `RecyclerView.Adapter<>`.

Класс `RecyclerView.Adapter<>`, как и все классы, для которых он является базовым, предоставляет возможность динамически заполнять контейнер `RecyclerView` элементами списка (рисунок 3.3.1), инициализируя их элементы графического интерфейса определенным образом, используя специальный класс `ViewHolder`.

`MovieRecyclerViewAdapter` располагается в пакете приложения bsuir.study.movier.adapter и работает в паре с `MovieHolder`, который располагается в пакете bsuir.study.movier.adapter.holder. `MovieHolder` является наследником класса `ViewHolder`.

Одним из полей класса `MovieRecyclerViewAdapter` является коллекция `List<Movie> moviesList`, которая хранит данные о медиаконтенте, которыми будут заполняться элементы списка в `RecyclerView`.

Наследуясь от RecyclerView.Adapter<>, класс MovieRecyclerViewAdapter должен переопределить все методы своего базового класса. Есть несколько методов, обязательных для переопределения:

– Public MovieHolder onCreateViewHolder(ViewGroup parent, int viewType);

– public int getItemViewType(int position);

– public void onBindViewHolder(MovieHolder movieHolder, final int position);

– public int getItemCount().

Метод onCreateViewHolder() отвечает за инициализацию компонента View, являющегося элементом списка, и определяет его разметку из модуля ресурсов (рисунок 3.3.1). Этот метод возвращает новый объект MovieHolder, содержащий внутри себя инициализированные компоненты графического интерфейса, готовые к дальнейшему заполнению.

Метод getItemViewType() возвращает уникальный идентификатор ресурса R.layout.item_movie, указывающего на файл с разметкой одиночного элемента списка.

Метод getItemCount() возвращает кол-во элементов, которые в конечном итоге должны быть загружены в RecyclerView. Это количество определяется количеством элементов в коллекции данных List<Movie> moviesList.

Самым важным методом в классе MovieRecyclerViewAdapter, является метод onBindViewHolder(MovieHolder movieHolder, final int position). Именно этот метод отвечает за заполнение элемента списка данными. Он принимает два параметра в качестве аргументов вызова:

– объект MovieHolder, инициализированный методом onCreateViewHolder();

– позиция элемента в коллекции данных List<Movie>, по которому следует обращаться к коллекции для получения информации о медиаконтенте.

В теле этого метода происходит заполнение компонентов пользовательского интерфейса элемента списка MovieHolder данными, такими как заголовок, жанр, год, а также постер. Постер фильма в элементе списка представляет собой View-компонент ImageView, в который помещается изображение.

Хранить все изображения на устройстве не представляется возможным, ввиду его огромного количества и постоянного обновления. Эта проблема решается с помощью специальной библиотеки Picasso, которая позволяет загрузить изображение из сети Интернет по его URL. Такое заполнение происходит посредством последовательных вызовов статических методов библиотеки:

– with(), в который передается текущий контекст приложения;

– load(), в который передается URL изображения для загрузки;

– `placeholder()`, в который передается идентификатор из модуля ресурсов, указывающий на изображение, которое будет отображаться по умолчанию в элементе `ImageView` до того, как произойдет полная загрузка изображения из сети Интернет;

– `into()`, в который передается идентификатор элемента `ImageView` из модуля ресурсов, в который необходимо произвести загрузку изображения.

Также в этом методе происходит регистрация обработчика события касания для каждого элемента списка, посредством вызова метода `movieHolder.itemView.setOnClickListener(onClickListener)`, принимающий в качестве единственного аргумента объект типа `View.OnClickListener`, который необходимо предварительно инициализировать. Инициализация такого объекта происходит благодаря использованию анонимного класса. Анонимный (безымянный) класс объявляется без задания имени класса и переменных данного безымянного типа – задаётся только конструктор класса вместе с его реализацией. У анонимного класса может быть только один экземпляр, причём он создаётся сразу при объявлении класса. Поэтому перед объявлением анонимного класса следует ставить оператор `new`.

Переопределяя метод этого класса `onClick()`, можно определить поведение системы при возникновении события нажатия на элемент списка. Дальнейшим поведением системы после такого события является запуск экрана приложения, содержащего детальную информацию о медиаконтенте, по элементу списка которого это событие произошло (рисунок 3.3).

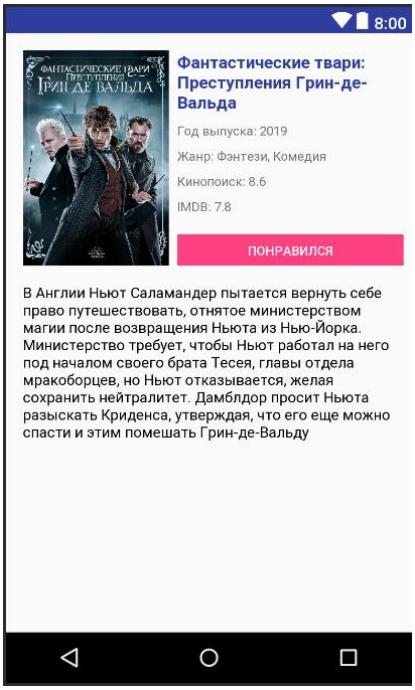

Рисунок 3.3 – Экран приложения для просмотра детальной информации о медиаконтенте.

3.4 Данные для MovieSearchFragment. Класс Movie.

Одним из полей класса `MovieSeacrhFragment` является коллекция `List<Movie> moviesList`. Данная коллекция, как было описано выше, является хранилищем данных о медиаконтенте, который необходимо отобразить пользователю в виде списка `RecyclerView`. Эта коллекция содержит в себе произвольное количество объектов `Movie`, которые представляют модуль данных в структурной схеме приложения,

Класс `Movie` расположен в пакете bsuir.study.movier.model и содержит следующие поля, помогающие максимально широко определить модель данных:

– `private int id` – содержит уникальный идентификатор медиаконтента в базе постащика контента;

– `private String title` – содержит название медиаконтента (фильма, сериала и др.)

– `private String rating` – содержит рейтинг медиаконтента;

– `private String overview` – содержит краткое описание для объекта медиаконтента;

– `private String genre` – содержит жанр, который представляет объект медиаконтента;

– `private String kinopoiskRating` – содержит рейтинг по версии ресурса «Кинопоиск»;

– `private String imdbRating` – содержит рейтинг по версии ресурса “iMDB”;

– `private String posterImgUrl` – содержит URL изображения постера для медиаконтента;

– `private int year` – содержит год выпуска объекта медиаконтента.

Все вышеперечисленные поля можно инициализировать с помощью конструктора класса, который принимает значения для этих полей в качестве своих аргументов. Все поля класса объявлены с использованием модификатора доступа `private`, что означает что доступ к этим полям для изменения возможет только изнутри этого объекта. Ни один другой компонент приложения не имеет доступ к модификации данных, содержащихся в этих объектах. Для их изменения используются специальные методы с префиксами «`get`» и «`set`», которые определены внутри класса `Movie`. Так как методы определены внутри класса, они имеют доступ для изменения значений полей класса.

Для получения информации о медиаконтенте, используется специальный модуль взаимодействия с API, который вызывается из других модулей приложения по необходимости. Данные о медиаконтенте приходят в мобильное приложение в формате JSON (рисунок 3.4). JSON основан на двух структурах данных:

– коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, запись, структура, словарь, хэш, именованный список или ассоциативный массив;

– упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

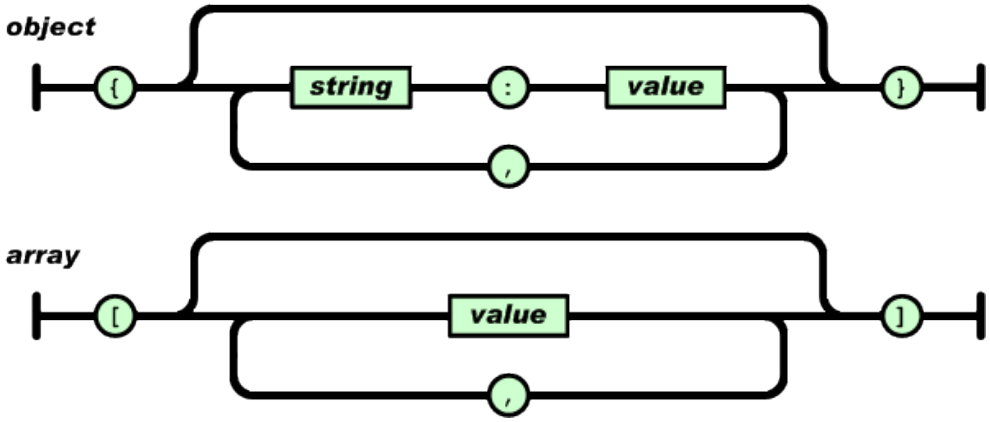

Рисунок 3.4 – Схематичное изображение формата JSON.

Для обработки данных, которые пришли в модуль взаимодействия с API от поставщика контента, используется библиотека Gson, которая помогает обрабатывать строку в формате JSON, конвертируя ее в необходимые Java-объекты и наоборот. Такая конвертация называется «сериализация» и «десериализация» соответственно.

Как при сериализации, так и при десериализации, Gson использует ряд аннотаций, которыми помечаются поля класса Java-объекта. В классе `Movie` поля помечены аннотациями `@SerializedName` и `@Expose`.

Аннотация `@ SerializedName` позволяет указать библиотеке Gson имя ключа, которое она будет искать в JSON-строке при десериализации объекта. Если не указывать эту аннотацию, то библиотека по умолчанию будет искать ключ, соответствующий названию переменной поля класса, в который эта строка будет десериализовываться.

Аннотация `@Expose` работает только в паре с методом `GsonBuilder.excludeFieldsWithoutExposeAnnotation()`, который исключает из обработки все поля, не имеющие аннотации `@Expose`. Если не использовать такой подход, то по умолчанию Gson будет производить попытки десериализовать JSON-строку для всех полей класса, что не всегда удобно, так как некоторые поля класса могут быть служебными, ожидать которых в пришедшей JSON-строке не приходится.

Указав и сконфигурировав все необходимые аннотации для полей класса, можно производить запросы через модуль взаимодействия с API к поставщику контента и, используя библиотеку Gson, создавать объекты `Movie`, помещая их в последствии в коллекцию `List<Movie>`

moviesList, которая используется в качестве исходных данных для создания и отображения списка медиаконтента конечному пользователю.

3.5 Взаимодействие с API. Класс APIClient и интерфейс MovieApi.

API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface) - описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

Модуль взаимодействия с API отвечает за отправку HTTP запросов к серверу поставщика контента, на котором находится стороннее API, предоставляющее разностороннюю информацию и медиаконтенте.

Все запросы формируются и отправляются в соответствии с протоколом HTTP. При таком способе общения с сервером, все запросы отправляются по определенному URL, каждый из которых соответствует определенному методу API.

Класс APIClient отвечает за отправку HTTP запросов к поставщику контента. Для этого используется библиотека Retrofit, позволяющая организовать работу с API, не прибегая при этом к системным вызовам функций Android. Такой подход очень удобен, так как при этом разработчику не приходится заботиться о правильной структуре HTTP запроса, а лишь описать интерфейс общения с API. Такой интерфейс в дипломном проекте описывается java-интерфейсом MovieApi.

Класс APIClient содержит в себе два поля:

– private static final String BASE_API_URL = "https://api.themoviedb.org/";

– private static Retrofit retrofit = null.

У всех URL которые используются для запросов к API, можно выделить общую часть, с которой начинаются все URL, в независимости от метода, к которому происходит обращение. Для выбранного API такую общую часть также можно выделить. Эта общая часть URL указана в поле класса BASE_API_URL. Это поле объявлено как статическое и константное, что означает невозможность его изменения программными вызовами. Это гарантирует его однозначность во всех модулях системы.

Вторым полем класса APIClient является переменная типа Retrofit. Это класс одноименной библиотеки, описанной выше. Эта переменная инициализируется в методе public static Retrofit getClient(), вызов которой возвращает проинициализированное значение переменной этого типа, подготавливая тем самым библиотеку Retrofit к работе.

Для корректной работы с Retrofit понадобятся три класса:

– POJO (Plain Old Java Object) – этим классом является класс Movie;

– Retrofit - класс для обработки результатов. Этим классом является описанный выше класс APIClient. Ему нужно указать базовый адрес в методе `baseUrl()`, которым является поле `BASE_API_URL` класса `APIClient`;

– Interface - интерфейс для управления адресом, используя команды GET и POST. Этим интерфейсом в приложении является интерфейс MovieAPI.

В интерфейсе `MovieAPI` необходимо описать методы для взаимодействия Retrofit с поставщиком контента по определенным правилам. Так, используются следующие аннотации, которые характеризуют метод запроса к серверу API поставщика контента (таблица 3.1).

Таблица 3.1 – Описание аннотаций Retrofit.

Аннотация	Описание
@GET()	GET-запрос для базового адреса. Также можно указать параметры в скобках.
@POST()	POST-запрос для базового адреса. Также можно указать параметры в скобках.
@Path	Переменная для замещения конечной точки, например, username подставится в {username} в адресе конечной точки.
@Query	Задаёт имя ключа запроса со значением параметра.
@Body	Используется в POST-вызовах (из Java-объекта в JSON-строку).

Для аннотации `@GET` в скобках указывается дополнение к базовому адресу URL. Аналогичное дополнение указывается и для аннотации `@POST()`.

Аннотация `@Query` используется перед атрибутами методов `MovieApi`, если необходимо подставить значение атрибута в URL, составленный аннотацией `@GET()`. Используя эти правила, был описан метод `MovieApi`, позволяющий обратиться к поставщику контента за информацией о контенте, фильтруя его специальным параметром «`query`», в который указывается искомое название контента или его описание. Этот метод используется для асинхронного обращения к API, не блокируя при этом главный поток приложения, обрабатывающий и отрисовывающий пользовательский интерфейс.

Диаграмма классов, иллюстрирующая взаимосвязи между всеми перечисленными выше компонентами программного средства приведена на чертеже ГУИР.400201.304 РР.1.

ЗАКЛЮЧЕНИЕ

За время преддипломной практики была продела большая работа по изучению материала и литературы, непосредственно касающихся темы дипломного проекта. Были систематизированы накопленные знания, получение в университете за всё время обучения.

Проанализировав эти знания, была составлена глава «1 ОБЗОР ЛИТЕРАТУРЫ», в которой описывается обоснование выбора операционной системы для разрабатываемого продукта, а также его целевая аудитория. Проведён обзор аналогов.

Логически представив взаимодействие программных модулей, была составлена глава «2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ», в которой описывается взаимосвязь компонентов системы, а также модели их поведения.

Также была составлена глава «3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ», из которой становится понятным каким именно образом модули из главы 2 взаимодействуют друг с другом.

Были составлены структурная и функциональная схемы разрабатываемого в рамках дипломного проектирования программного средства.

Итогом практики стал отчет, в котором содержится несколько глав дипломного проекта, а также реализовано большинство модулей программного средства, разрабатываемого в рамках дипломного проектирования.

Знания, полученные при разработке, а также составления отчета – бесценны.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Eckel, B. Thinking in Java (4th Edition) / Bruce Eckel – Prentice Hall Ptr, 2006. – 1079 с.

[2] Java™ Platform, Standard Edition 7 API Specification [Электронный ресурс]. – Электронные данные. – Режим доступа: http://docs.oracle.com/javase/7/docs/api/ (дата доступа 01.04.2019).

[3] Android Developers [Электронный ресурс]. – Электронные данные. – Режим доступа: http://developer.android.com/develop/index.html (дата доступа 01.04.2019).

[4] Meier, R. Professional Android 2 Application Development / Reto Meier – WROX, 2010. – 576 с.

[5] Android Service - Tutorial [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.vogella.com/tutorials/AndroidServices/article.html (дата доступа 03.04.2019).

[6] Учебник о Android. Уроки для начинающих Tutorial [Электронный ресурс]. – Электронные данные. – Режим доступа: http://startandroid.ru/ru/uroki/vse-uroki-spiskom.html (дата доступа 05.04.2019).

[7] Android Design Patterns [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.androiddesignpatterns.com/2012/08/implementing-loaders.html (дата доступа 10.04.2019).

[8] Харди, Б. Программирование под Android / Б. Харди, Б. Филлипс – СпБ: Питер, 2014. – 592 с.

[9] Welcome to Android Studio [Электронный ресурс]. – Электронные данные. – Режим доступа: http://tools.android.com/welcome-to-android-studio (дата доступа 10.04.2019).

[10] Gradle [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.gradle.org/ (дата доступа 11.04.2019).

[11] Gradle Plugin User Guide [Электронный ресурс]. – Электронные данные. – Режим доступа: http://tools.android.com/tech-docs/new-build-system/user-guide (дата доступа 01.04.2019).

[12] SQLite Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: http://sqlite.org/docs.html (дата доступа 12.04.2019).

ПРИЛОЖЕНИЕ А
(обязательное)
Вводный плакат

Сам плакат на А4
(рамка на обратной стороне)

ПРИЛОЖЕНИЕ Б
(обязательное)
Схема структурная

Сама схема на А4 с рамкой

ПРИЛОЖЕНИЕ В
(обязательное)
Схема функциональная

Сама схема на А4 с рамкой