МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Бронирование билетов в кинотеатр»

Выполнил студент Ярмолик Николай Сергеевич

(Ф.И.О.)

Руководитель проекта ассистент Колмаков М.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: ассистент Колмаков М.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: ассистент Колмаков М.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

Реферат

Лит.

Листов

1

*БГТУ 74217078, 2020*

Пояснительная записка курсового проекта содержит 49 страниц пояснительной записки, 8 таблиц, 26 иллюстраций, 8 источников литературы.

C#, java, ASP.NET core, MICROSOFT SQL SERVER, ENTITY FRAMEWORK.

Целью курсового проекта является разработка мобильного приложения по бронированию мест в кинотеатре.

В первой главе проводится анализ конкурентов приложения с их общим описанием, а также выделением положительных и отрицательных моментов по каждому приложению.

Вторая глава посвящена процессу проектирования системы.

В третьей главе описывается процесс разработки, принципы функционирования и назначение созданных компонент проекта.

В четвертой главе описывается проведение негативное тестирования приложения

В пятой главе описано руководство пользователя, позволяющее подробно понять интерфейс программного средства.

В заключении приведены результаты проделанной работы.

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

Содержание

Лит.

Листов

1

*БГТУ 74217078, 2020*

**Содержание**

[Введение 4](#_Toc42220264)

[1. Обзор аналогов приложения 5](#_Toc42220265)

[1.1 Обзор приложений 5](#_Toc42220266)

[1.2 Выводы по разделу 7](#_Toc42220267)

[2. Проектирование 8](#_Toc42220268)

[2.1 Диаграмма использования приложения 8](#_Toc42220269)

[2.2 Проектирование базы данных 9](#_Toc42220270)

[2.3 Проектирование мобильного приложения 12](#_Toc42220271)

[2.4 Проектирование сервера 12](#_Toc42220272)

[2.5 Выводы по разделу 12](#_Toc42220273)

[3. Программная реализация 13](#_Toc42220274)

[3.1 Постановка задачи 13](#_Toc42220275)

[3.2 Технические средства для реализации проекта 13](#_Toc42220276)

[3.3 Разработка базы данных 14](#_Toc42220277)

[3.4 Разработка мобильного приложения 14](#_Toc42220278)

[3.5 Разработка сервера 24](#_Toc42220279)

[3.6 Выводы по разделу 27](#_Toc42220280)

[4. Тестирование программного продукта 28](#_Toc42220281)

[4.1 Негативное тестирование 28](#_Toc42220282)

[4.2 Выводы по разделу 29](#_Toc42220283)

[5. Инструкция пользователя 30](#_Toc42220284)

[5.1 Выводы по разделу 37](#_Toc42220285)

[Заключение 38](#_Toc42220286)

[Список использованных источников 39](#_Toc42220287)

[ПРИЛОЖЕНИЕ А 40](#_Toc42220288)

**Введение**

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

Введение

Лит.

Листов

1

*БГТУ 74217078, 2020*

В современном мире сети кинотеатров обычно предоставляют информацию о сеансах специальным сайтам и приложениях, благодаря которым пользователи могут покупать и бронировать билеты на различные мероприятия, либо используют приложения, разработанные специально под конкретную сеть кинотеатров.

Целью моего курсового проекта стала разработка приложения, позволяющего автоматизировать бронирование мест в кинотеатрах. Помимо этого, данное приложение позволяет просматривать актуальные фильмы доступные на текущий момент. При бронировании билета приложение предоставит полный список кинотеатров и сеансов доступных для данного фильма. Также позволит просматривать забронированные билеты в удобной форме. Для этого были определены следующие задачи:

* Изучить аналогичные программные средства и литературные источники по тему курсового проекта
* Спроектировать приложение для бронирования билетов в кинотеатре
* Реализовать приложение для бронирования билетов в кинотеатре
* Протестировать приложение
* Написать инструкцию пользователя к приложению

Актуальность данной работы заключается в том, что информатизация затрагивает почти все сферы нашей жизни без исключения, включая и киносферу. Использование приложения не просто поможет экономить время, но также сохранять все купленные билеты в одном месте, не боясь, что забронированный билет может потеряться. Данное приложение позволит бронировать билеты в любом месте, достаточно лишь иметь смартфон с доступом в интернет, что намного удобнее, чем традиционный способ покупки билетов.

1. Обзор аналогов приложения

ФИО

Подпись

Дата

Лист

1

КП 01.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

1 Обзор аналогов приложения

Лит.

Листов

3

*БГТУ 74217078, 2020*

1.1 Обзор приложений

Приложение «Мираж Синема»

На рисунке 1.1 приведен общий интерфейс приложения. В приложении вы найдете расписание сеансов в кинотеатрах. Можно выбрать удобный сеанс и кинозал, покупка билета в один клик, участие в бонусной программе и просмотр интересного кино. Смотри свое кино в приложении, узнай афишу кино в любимом кинотеатре, выбери удобный сеанс и категорию кинозала для просмотра.

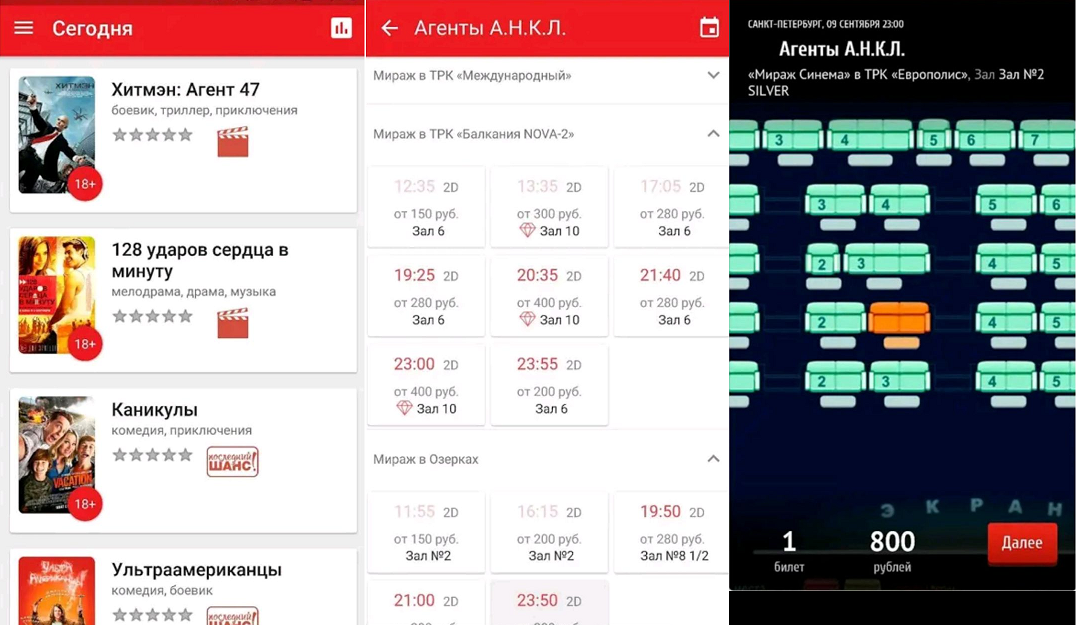


Рисунок 1.1 – Приложение «Мираж Синема»

Приложение предоставляет возможность просмотра купленных билетов, также подробное расписание кинотеатров и удобный выбор сеансов, имеющихся у данной сети кинотеатров.

Приложение «PIONEER»

Удобное приложение основной функцией которого является возможность покупки билетов в сети кинотеатров Линия Кино.

На рисунке 1.2 приведено общее представление о приложении

Данное приложение позволяет выбирать фильмы, кинотеатры и лучшие места, оплатить банковской картой с телефона и, минуя кассу. Просмотр киноафиши и графика премьер.

Также предоставляет информацию о всех кинотеатрах сети

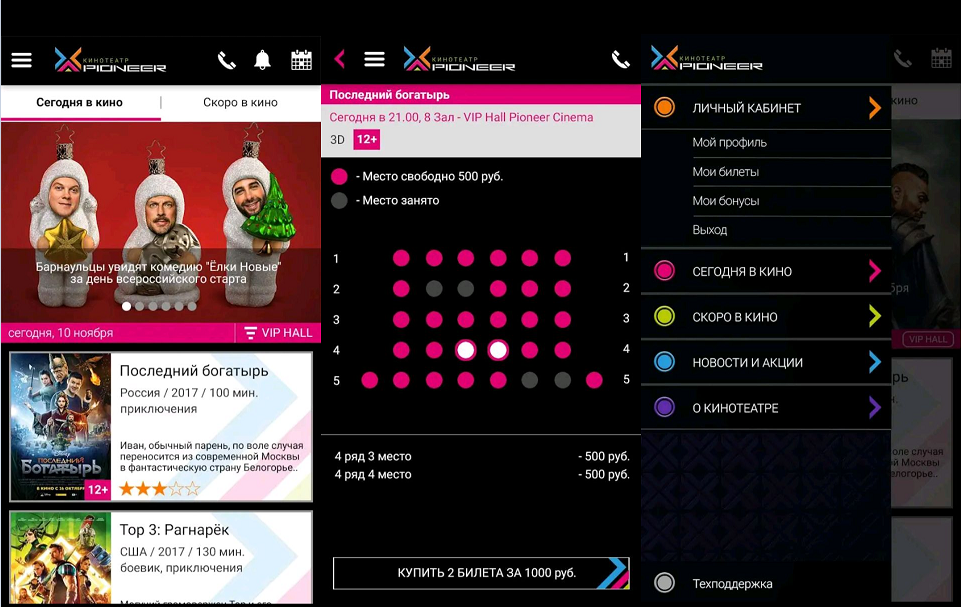


Рисунок 1.2 –Приложение «PIONEER»

Приложение «KASSA.RU»

Данное приложение предназначено для покупки билетов в кинотеатры KASSA.RU во многих городах России. Основная функция приложения – покупка билетов в кинотеатре.

На рисунке 1.3 приведено общее представление о приложении

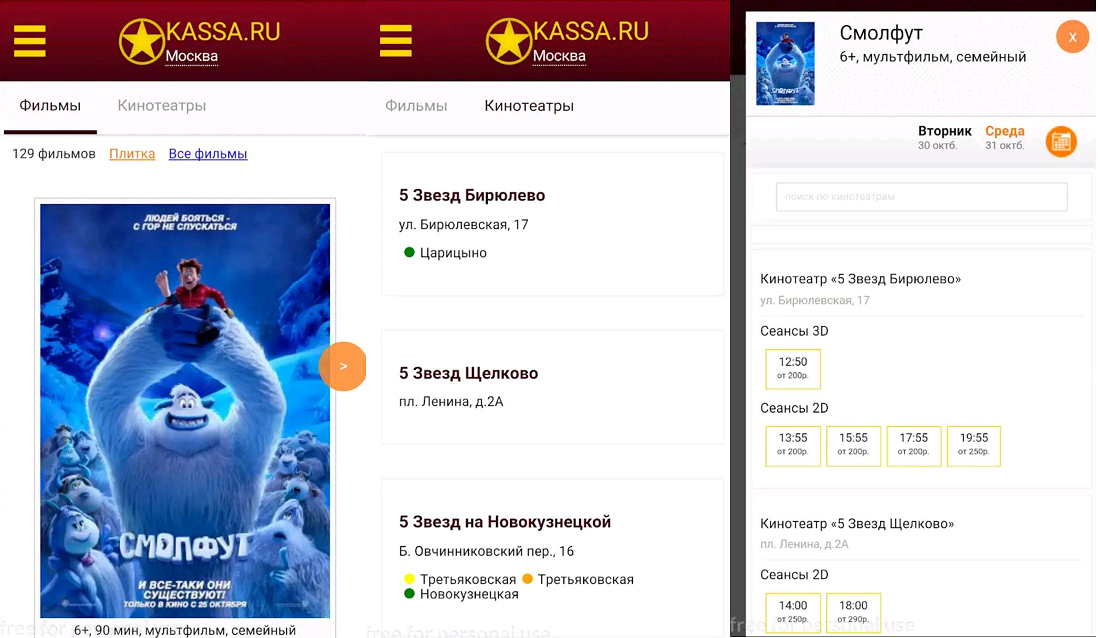


Рисунок 1.3 – Приложение «KASSA.RU»

Приложение позволяет покупать билеты в кинотеатры онлайн и без комиссий, отслеживать историю походов в личном кабинете, оплачивать покупки при помощи банковской карты, а также покупать билеты по самым выгодным условиям.

1.2 Выводы по разделу

Были рассмотрены мобильные приложения выполняющие функции бронирования, покупки и просмотра купленных билетов. В результате по каждому приложению были выделены положительные моменты, а также недостатки.

Первое приложение называется «Мираж Синема». Из положительных моментов следует выделить:

* приятный, достаточно современный дизайн
* простой и интуитивно понятный пользовательский интерфейс.
* широкая сеть кинотеатров с удобной покупкой билетов
* актуальная информация по сеансам с подробным описанием фильмов
* компонент отвечающий за отображение купленных билетов имеет уникальный и очень привлекательный дизайн

Из недостатков данного приложения можно выделить:

* некоторые компоненты не соответствуют стилистике приложения
* отсутствует полноценная регистрация
* при сворачивании и повторном разворачивании приложения перезагружается как при первом запуске
* отсутствует возможность оставить комментарии к отзывам

Следующее из рассмотренных приложений называется «PIONEER». Из положительных моментов можно выделить:

* хороший функционал приложения с полноценной регистрацией
* достаточно удобная навигация по приложению с некоторыми оговорками
* актуальная информация по сеансам с подробным описанием фильмов

Из недостатков этого приложения можно выделить:

* устаревший дизайн и сильная перегруженность интерфейса
* отсутствие работы офлайн, а также обновлений приложения
* периодически не подгружаются постеры к фильмам
* отсутствие возможности купить билет без регистрации

Последнее рассмотренное приложение называется «KASSA.RU». Из положительных моментов следует отметить:

* простой и достаточно понятный интерфейс с удобной навигацией
* довольно приятный дизайн некоторых компонентов
* актуальная информация по фильмам и сеансам

Из недостатков данного приложения можно отметить следующее:

* устаревший дизайн основной части приложения
* отсутствие работа офлайн, а также обновлений приложения
* большие отступы между элементами не соответствующие их размеру
* отсутствует возможность оставить отзыв
* неудобная форма покупки билета и выбора кинотеатра

Все положительные моменты, а также недостатки были учтены в рамках приложения «Бронирование мест в кинотеатр

**2. Проектирование**

ФИО

Подпись

Дата

Лист

1

КП 02.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

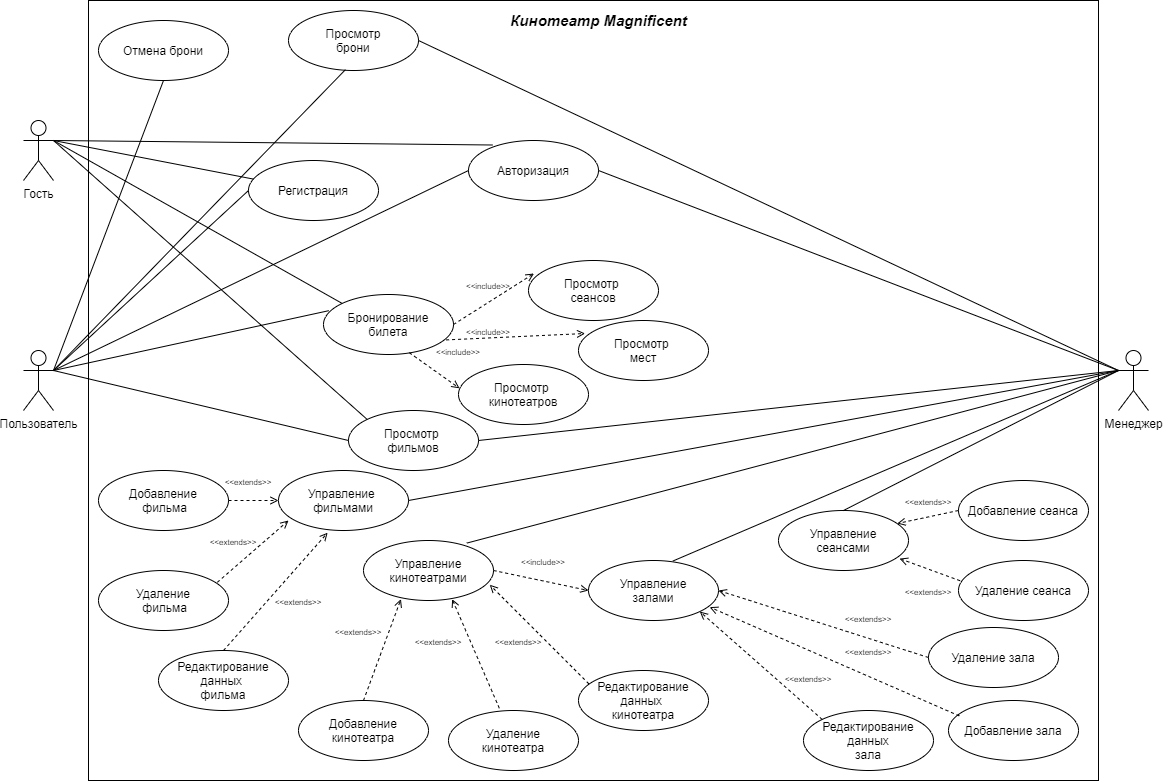
2. Проектирование приложения

Лит.

Листов

5

*БГТУ 74217078, 2020*

2.1 Диаграмма использования приложения

ДиаграммаUML–это графическое представление набора элементов,изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). На рисунке 2.1 можно ознакомиться с UML-диаграммой, а именно диаграммой вариантов использования.

Рисунок 2.1 – Диаграмма вариантов использования

При использовании приложения авторизованным пользователем доступна функциональность по просмотру фильмов с подробной информацией о них, включая отзывы и рейтинг, кинотеатров, удобный выбор сеансов, а также мест на них. При бронировании билетов они будут отображаться во вкладке билетов и по окончании сеанса есть возможность оставить отзыв. При авторизации под гостевым профилем доступны все функции кроме просмотра билетов. При авторизации под менеджером присутствует возможностей добавления, удаления, редактирования кинотеатров, залов и фильмов, а также функционал по добавлению сеансов.

2.2 Проектирование базы данных

В базе данных есть 8 таблиц, использующихся для следующих целей:

В таблице «user\_data» содержится информация о данных пользователей. Таблица состоит из 5 столбцов:

* id, тип данных uniqueidentifier, ограничения целостности: primary key, NOT NULL, DEFAULT newid()
* login, тип данных nvarchar(50), ограничения целостности: NOT NULL
* email, тип данных nvarchar(max), ограничения целостности NOT NULL
* password, тип данных nvarchar(max), ограничения целостности NOT NULL
* role\_id, тип данных int, ограничения целостности foreign key к таблице roles(id), NOT NULL

Таблица «roles» содержит номера ролей пользователей и состоит из 2 столбцов:

* id, тип данных int ограничения целостности: primary key, NOT NULL
* role, тип данных int ограничения целостности: NOT NULL

Таблица cinema\_info содержит информацию о кинотеатрах и состоит из 3 столбцов:

* id, тип данных uniqueidentifier, ограничения целостности: primary key, NOT NULL
* name, тип данных nvarchar(50), ограничения целостности: NOT NULL
* adress, тип данных nvarchar(max), ограничения целостности NOT NULL

Таблица «hall\_info» содержит информацию о залах, имеющихся у кинотеатров из таблицы cinema\_info. Данная таблица состоит из 4 столбцов:

* id, тип данных uniqueidentifier, ограничения целостности: primary key, NOT NULL
* cinema\_id, тип данных uniqueidentifier ограничения целостности foreign key к таблице cinema\_info(id), NOT NULL
* name, тип данных nvarchar(50), ограничения целостности: NOT NULL
* places, тип данных int ограничения целостности: NOT NULL

Таблица «film\_info» содержит информацию о фильме и состоит из 8 столбцов:

* id, тип данных uniqueidentifier, ограничения целостности: primary key, NOT NULL
* name, тип данных nvarchar(max), ограничения целостности: NOT NULL
* poster, тип данных nvarchar(max), ограничения целостности: NOT NULL
* country, тип данных nvarchar(max), ограничения целостности: NOT NULL
* genre, тип данных nvarchar(max), ограничения целостности: NOT NULL
* year, тип данных int ограничения целостности: NOT NULL
* duration, тип данных int ограничения целостности: NOT NULL
* description, тип данных text

Таблица séance содержит информацию о сеансе и состоит из 6 столбцов:

* id, тип данных uniqueidentifier, ограничения целостности: primary key, NOT NULL
* start\_time, тип данных smalldatetime, ограничения целосности: NOT NULL
* end\_time, тип данных smalldatetime, ограничения целосности: NOT NULL
* ticket\_price, тип данных float, ограничения целостности: NOT NULL
* hall\_id, тип данных uniqueidentifier ограничения целостности foreign key к таблице hall\_info(id), NOT NULL
* film\_id, тип данных uniqueidentifier ограничения целостности foreign key к таблице film\_info(id), NOT NULL

Следующая таблица называется «tickets» она содержит всю информацию о билетах, купленных пользователями и состоит из 4 столбцов:

* id, тип данных uniqueidentifier, ограничения целостности: primary key, NOT NULL
* place, тип данных int ограничения целостности: NOT NULL
* user\_id, тип данных uniqueidentifier ограничения целостности foreign key к таблице user\_data(id), NOT NULL
* seance\_id, тип данных uniqueidentifier ограничения целостности foreign key к таблице seance(id), NOT NULL

Последняя таблица называется «rating», она содержит отзыв и рейтинг к фильмам и состоит из 5 столбцов:

* id, тип данных uniqueidentifier, ограничения целостности: primary key, NOT NULL
* film\_id, тип данных uniqueidentifier ограничения целостности foreign key к таблице film\_info(id), NOT NULL
* user\_id, тип данных uniqueidentifier ограничения целостности foreign key к таблице user\_data(id), NOT NULL
* rating, тип данных float, ограничения целостности: NOT NULL
* review, тип данных nvarchar(max), ограничения целостности: NOT NULL

Далее будет описана структура локальной базы данных SQLite в рамках мобильного приложения. Она состоит из 3 таблиц, а именно: «user\_data», «seance», «tickets». Данные таблицы будут очень похожи на аналогичные таблицы в удаленной базе данных.

Первая таблица «user\_data» описывает информацию о пользователе и состоит из 4 столбцов:

* id, тип данных BLOB, ограничения целостности: primary key, UNIQUE
* login, тип данных TEXT
* email, тип данных TEXT
* password, тип данных TEXT

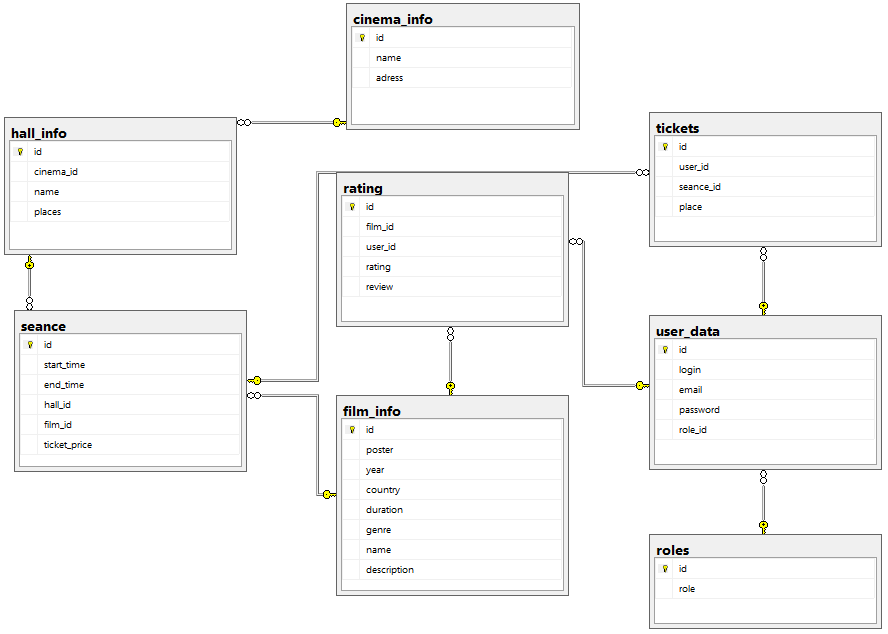
Вторая таблица называется «seance» и состоит из 7 столбцов, а именно:

* id, тип данных BLOB, ограничения целостности: primary key, UNIQUE
* cinema\_info, тип данных TEXT
* hall\_info, тип данных TEXT
* film\_info, тип данных TEXT
* date, тип данных TEXT
* start\_time, тип данных TEXT
* end\_time, тип данных TEXT

Последняя таблица называется «tickets» и содержит информацию о билетах, купленных пользователями, состоит из 4 столбцов, а именно:

* id, тип данных BLOB, ограничения целостности: primary key, UNIQUE
* séance\_id, тип данных BLOB, ограничения целостности foreign key к таблице seance(id)
* user\_id, тип данных BLOB, ограничения целостности foreign key к таблице user\_data(id)
* place, тип данных TEXT

На рисунке 2.2 приведена диаграмма основной базы данных состоящая из 8 таблиц.

Рисунок 2.2 – Диаграмма базы данных

При проектировании базы данных также учитывалась возможность каскадного удаления данных. Каскадное удаление данных будет применяться к таблицам «hall\_info», «rating», «seance», «tickets». В случае удаления кинотеатра соответственно будут удалены все залы, сеансы, билеты доуступные в данном кинотеатре.

2.3 Проектирование мобильного приложения

В рамках курсового проекта необходимо разработать мобильное приложение, позволяющее пользователю просматривать, выбирать и бронировать билеты в кинотеатр, а также получать полную информацию обо всех фильмах, а также сеансов по ним.

Приложение должно использовать архитектуру REST для обмена данных с сервером. Также приложение использовать SOAP сообщения для получения информации о текущей дате, городе и регионе. Приложение должно обеспечивать работу как в онлайн режиме, так и в офлайн режиме.

2.4 Проектирование сервера

Для синхронизации двух и более клиентов необходимо серверное приложение. Серверная часть приложения должна предоставлять API для доступа к каждой сущности в базе данных. Передаваемые или получаемые данные от сервера должны передаваться в формате JSON.

Серверное приложение должно быть построено на основе архитектурного стиля REST и будет использоваться для взаимодействия компонентов распределённого приложения в сети.

При обращении к серверу по заданному маршруту и определенному HTTP методу сервер будет определять с какой сущностью базы данных необходимо взаимодействовать и какие действия над данными необходимо будет произвести.

При помощи HTTP метода GET сервер будет возвращать данные клиенту, ссылаясь на аргументы в запросе, как на условие для выборки данных. Метод POST в свою очередь позволяет передать данные на сервер для последующей обработки. В рамках проекта также используются методы PUT для обновления записей в базе данных и метод DELETE для удаления записей из базы данных.

Дополнительно используется сервис, использующий SOAP, для получения и передачи xml сообщений.

2.5 Выводы по разделу

В данной главе было рассмотрено проектирование каждого отдельного компонента приложения.

По части мобильного приложения был определен основной функционал приложения, также были разработаны диаграмма вариантов использования приложения.

Была также спроектирована база данных, где были определены необходимые таблицы, поля для каждой из них, определены типы данных и ограничения целостности. Также были установлены связи между таблицами, которые описаны на схеме базы данных.

При проектировании серверного приложения был определен архитектурный стиль взаимодействия между клиентом и сервером, формат передаваемых данных, обозначили HTTP методы которые будут использованы в работе с серверной частью.

**3. Программная реализация**

ФИО

Подпись

Дата

Лист

1

КП 03.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

3 Программная реализация

Лит.

Листов

12

*БГТУ 74217078, 2020*

3.1 Постановка задачи

Основной целью курсового проекта является разработка мобильного приложения, позволяющего бронировать места в кинотеатрах. При выполнении поставленной цели в курсовом проекте необходимо решить следующие задачи:

* разработать простой и достаточно понятный интерфейс с современным дизайном
* обеспечить возможность выбора кинотеатра и времени сеанса
* реализовать удобный выбор мест на сеанс
* реализовать удобная форму для просмотра купленных билетов
* обеспечить возможность работы офлайн
* реализовать получение письма на почту с информацией о билете
* реализовать полноценную регистрация с авторизацией
* обеспечить возможность оставить отзыв
* реализовать подробное описание фильмов и их рейтинг
* обеспечить возможность просмотра отзыва к фильмам

3.2 Технические средства для реализации проекта

Для разработки приложений курсового проекта были использованы следующие технические средства:

Android Studio – интегрированная среда разработки для работы с платформой Android. Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств.

Java – строго типизированный объектно-ориентированный язык программирования. C# — объектно-ориентированный язык программирования.

.NET Core – это модульная платформа для разработки программного обеспечения с открытым исходным кодом. Entity Framework – специальная объектно-ориентированная технология на базе фреймворка .NET для работы с данными.

OkHttp – библиотека предоставляющая собой простой и удобный клиент для работы с HTTP запросами. Newtonsoft.Json – библиотека позволяющую преобразовывать объекты в JSON формат с большим количеством настроек.

REST – это стиль архитектуры программного обеспечения для распределенный систем, который, как правило используется для построения веб-служб. SOAP – протокол обмена структурированными сообщениями в распределенной вычислительной среде.

3.3 Разработка базы данных

Для хранения данных будет использована СУБД SQL Server.

Код создания структуры серверной базы данных в Приложении A.

Структурные объекты базы данных приведены в разделе проектирвания.

Также со стороны серверного приложения необходимо было добавить представление «View\_FilmInfo», которое предназначено для вывода общей информации о фильмах при переходе пользователя на главный экран мобильного приложения

3.4 Разработка мобильного приложения

Графическая структура проекта представлена на Рисунке 3.1. Мобильное приложение было реализовано в среде разработки – Android Studio 3.0. Для написания приложения был использован язык программирования Java.

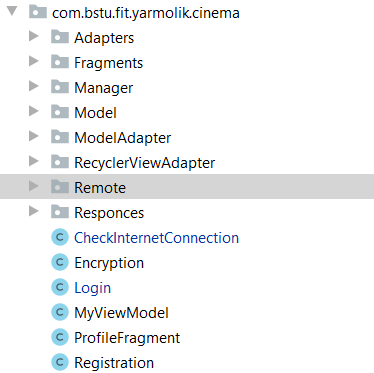


Рисунок 3.1 – Графическая структура мобильного проекта

В таблице 3.1 приведена информация о компонентах приложения: имя пакета и его описание.

Таблица 3.1 – Описание компонентов мобильного приложения

|  |  |
| --- | --- |
| Пакет | Описание |
| Adapters | Содержит классы используемые при работе с элементами CardView, RecyclerView, GridView |
| Fragments | Содержит классы, описывающие работу пользовательской части приложения |
| Manager | Содержит классы, описывающие работу части приложения для менеджера |

Окончание таблицы 3.1

|  |  |
| --- | --- |
|  |  |
| Model | Содержит классы описывающие get; set; для свойств позволяющие управлять доступом к полям в рамках приложения |
| ModelAdapter | Содержит классы описывающие get; set; для свойств позволяющие управлять доступом к полям используемым в адаптерах |
| Remote | Содержит классы описывающие get; set; для свойств позволяющие управлять доступом к полям используемым при отправке запроса на сервер |
| Responces | Содержит классы описывающие get; set; для свойств позволяющие управлять доступом к полям при получении ответа с сервера |

Пакет Adapters содержит описывающих логику работы таких элементов как RecyclerView, CardView, GridView.

Пакет состоит из классов FilmAdapter, MyAdapter, SeanceAdapter, SlidingImage\_Adapter, TicketAdapter, TicketStaticAdapter. Данные классы используются для заполнения данными элементов RecyclerView и CardView и содержат в себе 3 основных метода, а именно: onBindViewHolder, onCreateViewHolder, getItemCount.

Пример кода метода onBindViewHolder приведен в листинге 3.1

Листинг 3.1 – Метод onBindViewHolder

|  |
| --- |
| @Override **public void** onBindViewHolder(**final** CinemaViewHolder holder, **int** position) {  YoYo.*with*(Techniques.***FadeInRight***).playOn(holder.**cardView**);  holder.**cardName**.setText(**list**.get(position).getCardName());  holder.**imageResourceId**.setImageResource(**list**.get(position).getImageResourceId());  holder.**address**.setText(**list**.get(position).getAddress()); } |

Данный метод заполняет данными карточку и принимает в качестве параметра позицию текущего элемента, а также визуальный интерфейс holder. Дополнительно была использована библиотека YoYo, позволяющая делать различные анимации появления элементов. Далее указываются поля, в которые будут добавляться данные, которые получаются путем передачи из класса – родителя необходимых данных описываемых соответствующим классом, содержащим свойства для данного элемента в ArrayList и при помощи соответствующих свойств данные, извлекаются по конкретной позиции и добавляются в соответствующую карточку.

Следующим методом является onCreateViewHolder, код метода приведён в листинге 3.2.

Листинг 3.2 – Метод CreateViewHolder

|  |
| --- |
| @NonNull @Override **public** CinemaViewHolder onCreateViewHolder(@NonNull ViewGroup parent, **int** viewType) {  View view= LayoutInflater.*from*(parent.getContext()).inflate(R.layout.***recycler\_items***,parent,**false**);  **return new** CinemaViewHolder(view); } |

В данном методе мы указываем разметку, которую хотим использовать для отображения наших элементов. В качестве возвращаемого значения используется визуальный интерфейс CinemaViewHolder.

Последним методом является getItemCount, код метода приведен в листинге 3.3.

Листинг 3.3 – Метод getItemCount

|  |
| --- |
| @Override public int getItemCount() {return list.size();} |

Данный метод возвращает количество элементов нашего списка.

Данные 3 метода являются обязательными и используются во всех адаптерах, реализующих Card View. Отдельно отметим класс GridArrayAdapter, в проекте он используется для разделения части экрана на 10 колонок для отображения мест в кинозале. Данный класс содержит 1 основной метод getView использующийся как для построения визуальной части интерфейса, так и для описания функций данного компонента.

Далее приведен список классов, содержащихся в пакете Fragments, с кратким описанием функциональности по каждому классу.

* AddInfoTicket: класс используемый для выбора сеанса
* CinemaFragment: класс используемый для отображения всех имеющихся кинотеатров у данной сети кинотеатров.
* FilmFragment: класс используемый для отображения всех имеющихся фильмов у данной сети кинотеатров
* FragmentMore: класс используемый для отображения информации о сети кинотеатров, а также функционал для выхода из приложения
* InfoFilmFragment: класс используемый для отображения подробной информации о фильме, а также рейтинга фильма
* ListReview: класс описывающий get; set; для свойств позволяющие управлять доступом к полям при получении информации об отзыве
* ReviewActivity: класс используемый в Dialog для отображения отзыва о фильме
* OnFragmentBookListener: интерфейс содержащий один метод для реализации в классе
* SelectTickets: класс используемый для выбора мест на сеанс
* SliderFragment: класс используемый слайдером PageViewer для отображения фильмов
* TicketsFragment: класс используемый для отображения забронированных билетов
* TicketInfo: класс используемый для отображения общей информации о билете перед бронированием
* UserTicketActivity: класс используемый при выводе окна Dialog после нажатия кнопки на билете

В каждом классе в данном пакете используется метод init, предназначенный для инициализации элементов класса, таких как кнопки, поля для ввода, различные списки и т.д.

Далее будет приведен код основных методов в данном пакете.

Первый метод LoadCinema, код метода приведен в листинге 3.4

Листинг 3.3 – Метод LoadCinema

|  |
| --- |
| Call<List<CinemaResponce>> call=**iMyApi**.getCinema(); call.enqueue(**new** Callback<List<CinemaResponce>>() {  @Override  **public void** onResponse(Call<List<CinemaResponce>> call, Response<List<CinemaResponce>> response) {  **cinema**=response.body();  **for**(CinemaResponce post : **cinema**){  **nameList**.add(post.getName());  **idList**.add(post.getId());  **adressList**.add(post.getAdress());  } **for**(**int** i=0;i<**nameList**.size();i++){  **try**{  **listItems**.add(**new** CinemaModel(**images**[0], **nameList**.get(i),**adressList**.get(i)));  } **catch** (Exception ex){  Toast.*makeText*(getContext(), ex.getMessage(), Toast.***LENGTH\_LONG***).show(); }}  **carAdapter** = **new** MyAdapter(**listItems**);  **recyclerView**.setAdapter(**carAdapter**);  } @Override  **public void** onFailure(Call<List<CinemaResponce>> call, Throwable t) { Toast.*makeText*(getContext(), t.getMessage(), Toast.***LENGTH\_LONG***).show();  }}); |

Для начала создается объект Call, при этом используется интерфейс iMyApi в котором описываются все HTTP методы для обращения к серверу. Далее используется метод enqueue представляющий метод обратного вызова для результата, полученного с сервера. После чего из тела ответа при помощи класса CinemaResponce из папки Responces извлекается информация и помещается в соответствующие ArrayLists. Код некоторых методов приведен в Приложении А. Дальнейшие методы GET используемые в приложении будут иметь похожую структуру запроса. В рамках пакета Fragments это будут методы:

* loadFilmInfo: метод для загрузки информации о фильмах
* loadSeanceInfo: метод для загрузки информации о сеансах
* loadUserTicketsInfo: метод загрузки дополнительной информации о билете
* loadBookTickets: метод загрузки билетов пользователя
* loadReview: метод загрузки отзывов к фильмам

Также следует рассмотреть метод используемый для выхода из аккаунта, а также выхода из приложения finishApplicationWithCleanSharedPreferencesFile, код метода приведен в листинге 3.5

Листинг 3.4 – Метод finishApplicationWithCleanSharedPreferencesFile

|  |
| --- |
| **private void** finishApplicationWithCleanSharedPreferencesFile(){  File file= **new** File(**"/data/data/com.bstu.fit.yarmolik.cinema/shared\_prefs/user.xml"**);  file.delete();  Intent homeIntent = **new** Intent(Intent.***ACTION\_MAIN***);  homeIntent.addCategory( Intent.***CATEGORY\_HOME*** );  homeIntent.setFlags(Intent.***FLAG\_ACTIVITY\_CLEAR\_TOP***);  startActivity(homeIntent);  System.*exit*(0); } |

Данный метод вызывается при нажатии кнопки закрытия во фрагменте с информацией о сети кинотеатров. Для начала удаляется файл SharedPreferences хранящий логин и пароль пользователя, далее создается обьект интент, к которому устанавливаются флаг и категория, означающая переход на главный экран. После чего вызывается StartActivity и system.exit для полного выхода из приложения.

Далее следует рассмотреть пакет Manager, он содержит классы для работы функционала Менеджера. Далее перечислены классы в данном пакете:

* AddCinemaFragment: класс используемый для добавления кинотеатра
* AddHallFragment: класс используемый для добавления зала
* AddPlacesSeanceFragment: класс используемый дл добавления времени и цены билета
* AddSeanceFragment: класс для выбора основных компонентов сеанса
* DeleteCinemaFragment: класс используемый для удаления кинотеатра
* DeleteFilmFragement: класс используемый для удаления фильма
* DeleteHallFragment: класс используемый для удаления зала
* ManagerFilmFragment: класс используемый для добавления фильма
* onFragment1DataChangeListener: интерфейс используемый для перехода между фрагментами
* UpdateCinemaFragement: класс используемый для обновления данных кинотеатра
* UpdateFilmFragement: класс используемый для обновления данных фильма

В каждом классе в данном пакете используется метод init, предназначенный для инициализации элементов класса, таких как кнопки, поля для ввода, различные списки и т.д.

В рамках пакета Manager наиболее часто используемыми методами будут методы:

* loadFilmInfo: метод для загрузки информации о фильмах
* loadHallInfo: метод для загрузки информации о залах
* loadCinema: метод для загрузки информации о кинотеатрах

Код показывающий работу с 4 основными методами HTTP приведен в Приложении А.

Далее следует рассмотреть пакет Model содержащий классы описывающие get; set; позволяющие управлять доступом к полям в рамках каждого класса используемых при отправке данных на сервер. В таблице 3.2 приведена информация о данном пакете: имя класса и назначение:

Таблица 3.2 – Пакет Model

|  |  |
| --- | --- |
| Класс | Назначение |
| BoughtTicket | Покупка билета |
| CheckSeance | Проверка существования сеанса |
| CinemaInfo | Добавление в базу данных информации о кинотеатре |
| FilmInfo | Добавления в базу данных информации о фильме |
| HallInfo | Добавления в базу данных информацию о зале |
| LoginUser | Проверки логина и пароля пользователя |
| Rating | Добавления в базу данных рейтинга по фильму |
| Seance | Добавления в базу данных информацию о сеансе |
| SeanceDate | Получения информации о сеансе |
| FilmInfo | Добавление в базу данных информации о фильме |
| UserData | Добавления в базу данных информации о пользователе |

Далее следует рассмотреть пакет ModelAdapter содержащий классы описывающие get; set; позволяющие управлять доступом к полям в рамках работы с адаптерами. В таблице 3.3 приведена информация о данном пакете: имя класса и назначение:

Таблица 3.3 – Пакет ModelsAdapter

|  |  |
| --- | --- |
| Класс | Назначение |
| CinemaModel | Доступ к данным кинотеатра в CardView |
| FilmModel | Использования данных фильма в CardView |
| ReservationModel | Использования данных о забронированных билетах в CardView |
| SeanceModel | Использования данных сеанса в CardView |
| TicketModel | Использования данных билета в CardView |
| TicketStaticModel | Использования данных билета в CardView в автономном режиме |

Пакет Remote является наиболее важным в рамках данного проекта так как содержит в iMyApi все HTTP методы взаимодействия, а в RetrofitClient определяется обьект Retrofit для взаимодействия с сервером по архитектуре REST

Пример кода 4 основных методов HTTP при использовании REST приведен в листинге 3.5

Листинг 3.5 – Код основных методов HTTP в REST

|  |
| --- |
| @POST(**"api/films"**) Observable<String> filmInfo(@Body FilmInfo film); @GET(**"api/films"**) Call<List<FilmResponse>> getFilms(); @DELETE(**"api/films/{id}"**) Call<Void> deletePost(@Path(**"id"**) String id); @GET(**"api/films/{id}"**) Call<FilmResponse> getFilm(@Path(**"id"**) String id); @PUT(**"api/films/{id}"**) Call<FilmInfo> updateFilm(@Path(**"id"**) String id, @Body FilmInfo film); |

Первый метод POST предназначен для добавление записи о фильме в таблицу с фильмами. Далее приведен метод GET предназначенный для получениях всех фильмов. Метод DELETE используется для удаления фильма по его id. Метод PUT используется для обновления данных о фильме. Принципы написания методов используются и для других таблиц в рамках данной курсовой работы.

Далее следует рассмотреть метод getInstance класса RetrofitClient. Код метода приведен в листинге 3.6

Листинг 3.6 – Метод getInstance

|  |
| --- |
| **public static** Retrofit getInstance() {  HttpLoggingInterceptor logging = **new** HttpLoggingInterceptor();  logging.setLevel(HttpLoggingInterceptor.Level.***BODY***); OkHttpClient client = **new** OkHttpClient.Builder()  .addInterceptor(logging)  .build();  **if**(*instance*==**null**)  *instance*=**new** Retrofit.Builder()  .baseUrl(**"http://10.0.2.2:5000/"**)  *//.baseUrl("http://0e18e9b4.ngrok.io/")* .addConverterFactory(ScalarsConverterFactory.*create*())  .addConverterFactory(GsonConverterFactory.*create*())  .addCallAdapterFactory(RxJava2CallAdapterFactory.*create*())  .client(client)  .build();  **return** *instance*; } |

Для удобного отслеживания запросов здесь используется HTTPLoggingInterceptor, который отображает всю необходимую информацию о запросах/ответах такую как адрес, заголовки, тело запроса/ответа. Далее мы создаем нашего OkHttp клиента, непосредственно работающего с нашими запросами. Указываем в параметре baseUrl путь к нашему серверу, далее тип передаваемых данных будет в формате json, а также указываем что мы хотим использовать RxJavaFactory.

Следует также рассмотреть пакет Responces, содержащий классы, описывающие get; set; позволяющие управлять доступом к полям в рамках каждого класса используемых при получении данных с сервера. В таблице 3.4 приведена информация о данном пакете: имя класса и назначение:

Таблица 3.4 – Содержание пакета Responces

|  |  |
| --- | --- |
| Класс | Назначение |
| CinemaResponce | Получение информации о кинотеатрах |
| FilmResponce | Получение информации о фильмах |
| GuestResponce | Получение информации о гостевом профиле |
| HallResponce | Получение информации о залах, получаемых с сервера |
| PlacesNumber | Получение информации о количестве мест на сеанс |
| ReviewResponce | Получение информации об отзывах к фильму |
| SeanceDataResponce | Получение информации о дате сеанса |
| SeanceInfo | Получение информации о сеансе |
| TicketInfoData | Используется для информации о билете |
| TicketResponce | Получение информации о билетах |
| TicketStaticResponce | Получение информации о билетах |
| UserResponce | Получение информации о пользователе |
| UserTicket | Получение информации о конкретном билете |

Также одним из заданий курсового проекта требовалось реализовать SOAP, для этого в пакете SOAP определен класс CallSoap, для отправки SOAP сообщения. Пример кода для работы с SOAP на android приведен в Приложении А.

Далее в таблице приведены классы, находящиеся в корневой папке, а также класс MainActivity.

Таблица 3.5 – Классы в корневом пакете

|  |  |
| --- | --- |
| Класс CheckInternetConnection – Класс для проверки наличия интернет соединения | |
| isOnline | Метод для проверки наличия интернет соединения. |
| Класс Login – Класс для авторизации пользователей | |
| Skip | Метод для захода в приложение под гостевым профилем |
| SignUp | Метод перехода в окно регистрации |
| initViews | Метод инициализации для элементов класса |
| startAnimation | Метод для начала анимации |
| onStop | Метод вызываемый при закрытии activity |
| getGuestInfo | Загружает информацию о гостевом профиле |
| clear | Метод для очистки значений списков |

Окончание таблицы 3.5

|  |  |
| --- | --- |
| checkLocalUser | Метод проверки существования пользователя в локальной базе данных |
| loadLocalWithSharedPreferences | Метод вызываемый в случае выбором пользователя «Не выходить» значения checkbox |
| Класс MyViewModel – описывает модель под реализацию загрузки фильмов | |
| Класс Registration – Класс используемый для регистрации пользователя | |
| init | Метод инициализации для элементов класса |
| Md5 | Метод хеширования пароля алгоритмом md5 |

Ниже приведены 2 основных класса для Пользователя и Менеджера

Таблица 3.6 – Основные классы

|  |  |
| --- | --- |
| Класс MainActivity – Класс главной страница приложения | |
| init | Метод инициализации для элементов класса |
| loadUserTicketsInfo | Метод загрузки билетов в локальную базу данных |
| loadPlaces | Метод загрузки мест в локальную базу данных |
| onOptionsItemsSelected | Метод отслеживающий выбранные элементы навигации |
| onCreateOptionsMenu | Метод создания меню |
| Класс ManagerActivity – Класс главной страница Менеджера | |
| onBackPressed | Метод переопределяющий назначение кнопки назад |
| onOptionsItemsSelected | Метод отслеживающий выбранные элементы навигации |
| onCreateOptionsMenu | Метод создания меню |

3.5 Разработка сервера

Графическая структура сервера представлена на рисунке 3.2. Сервер был реализован в среде разработки – Microsoft Visual Studio 2019. Для написания приложения был использован язык программирования C#.

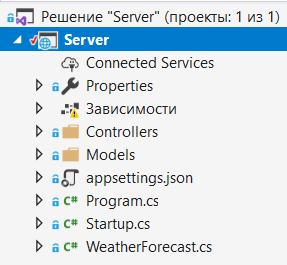


Рисунок 3.2 – Графическая структура сервера

На рисунке 3.3 представлена папка Controllers которая хранит в себе классы контроллеров обрабатывающие поступающие запросы к серверу.

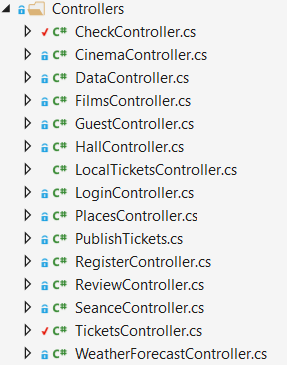


Рисунок 3.3 – Содержание папки Controllers

Код основных методов папки Controllers приведен в Приложении А

Таблица 3.7 – Содержание папки Controllers

|  |  |
| --- | --- |
| Класс CheckController – контроллер проверки сеанса | |
| /check (POST) | Метод проверяет, существует ли сеанс по текущему фильму с датой позже текущего времени |
| Класс CinemaController – контроллер для работы с таблицей Cinema | |
| /cinema (GET) | Метод получает данные по кинотеатрам |
| /cinema/{id} (GET) | Метод получает данные по кинотеатру по его id |
| /cinema (POST) | Метод добавляет кинотеатр в базу данных |
| /cinema/{id} (PUT) | Обновляет данные кинотеатра по id |
| /cinema/{id} (DELETE) | Удаляет данные кинотеатра по id |
| Класс DataController – контроллер для получения информации по конкретному сеансу | |
| /data (POST) | Метод проверяет, существует ли сеанс по переданным значениям и возвращает основную информацию по нему |

Окончание таблицы 3.7

|  |  |
| --- | --- |
| Класс FilmsController – контроллер для работы с таблицей Films | |
| /films (GET) | Метод получает данные по фильмам |
| /films/{id} (GET) | Метод получает данные по фильму по его id |
| /films (POST) | Метод добавляет фильм в базу данных |
| /films/{id} (PUT) | Обновляет данные фильма по id |
| /films/{id} (DELETE) | Удаляет данные фильма по id |
| Класс GuestController – контроллер возвращающий данные по пользователю с ролью гостя | |
| /guest/{id} (GET) | Метод получает данные по гостевому пользователю по его role\_id |
| Класс HallController – контроллер для работы с таблицей Hall | |
| /hall (GET) | Метод получает данные по залам |
| /hall/{id} (GET) | Метод получает данные по залу по его id |
| /hall (POST) | Метод добавляет зал в базу данных |
| /hall/{id} (PUT) | Обновляет данные зала по id |
| /hall/{id} (DELETE) | Удаляет данные зала по id |
| Класс LoginController – контроллер используется для проверки логина и пароля пользователя | |
| /login (POST) | Проверяет полученные логин и пароль |

В таблице 3.8 приведена информация о контроллерах, непосредственно связанных с бронированием билета.

Таблица 3.8 – Контроллеры используемые при бронировании билета.

|  |  |
| --- | --- |
| Класс LocalTicketsController – контроллер возвращающий информацию о билете | |
| /localtickets/{id} (GET) | Метод получает данные по билета по его id |
| Класс PublishController – контроллер используется добавления и получения билетов | |
| /publish/{id} (GET) | Метод получает места на сеанс |
| /publish (POST) | Метод добавляет места в таблицу Tickets |
| SendEmailAsync | Метод отправляет на почту сообщение с информацией о билете |
| Класс RegisterController – контроллер для работы с таблицей UserData | |
| /register (POST) | Добавляет данные пользователя в таблицу UserData |
| Класс ReviewController – контроллер для работы с таблицей Rating | |
| /review/{id} (GET) | Метод получает отзыв по id фильма |
| /review (POST) | Метод добавляет отзыв на фильм по id фильма |
| Класс SeanceController – контроллер для работы с таблицей Seance | |
| /seance/{id} (GET) | Метод получает сеанс по его id |
| /seance (POST) | Метод добавляет сеанс в таблицу Seance |
| Класс TicketsController – контроллер для работы с таблицей Tickets | |
| /seance/{id} (GET) | Метод получает билеты по id пользователя |
| /séance/GetPlaces/ (GET) | Метод получает места по id пользователя |

Папка Models содержит в себе классы, автоматически сгенерированные при помощи Entity Framework, для реализации использовалась команда «Scaffold». Пример выполнения команды приведен в листинге 3.7.

Листинг 3.7 – Генерация модели базы данных

|  |
| --- |
| Scaffold-DbContext"Server=DESKTOP-15P21ID;  Initial Catalog=Cinema;  user=Nikkooly;  password=\*\*\*\*\*\*\*;”  Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models. |

Первым параметром мы указываем имя нашего SQL сервера, вторым параметром указываем имя нашей базы данных, с которой будет осуществляться взаимодействие, после указываем имя и пароль от нашего сервера и выходную папку в которую будет сохранено представление нашей базы данных для Entity.

В случае каких-либо изменений в базе данных использовалась следующая команда «Scaffold» для обновления модели бд на сервере. Пример выполнения команды приведен в листинге 3.8.

Листинг 3.8 – Регенерация модели базы данных

|  |
| --- |
| Scaffold-DbContext"Server=DESKTOP-15P21ID;  Initial Catalog=Cinema;  user=Nikkooly;  password=\*\*\*\*\*\*\*;”  Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Force |

Описание данного листинга аналогично описанию листинга 3.1, но в конце необходимо добавить –Force, это указывает Scaffold что необходимо пересобрать модель базы данных с учетом всех изменений.

При разработке курсового проекта была реализована служба SOAP возвращающая по координатам текущее время, регион и город устройства. Код сервиса можно просмотреть в Приложении А

3.6 Выводы по разделу

В данной главе было рассмотрена разработка каждого отдельного компонента приложения.

На первоначальном этапе были рассмотрены технологии, которые применены для разработки курсового проекта.

Вторым этапом была разработка структуры базы данных на стороне сервера и клиента.

Третий этапом стала разработка клиента на мобильное устройство, где был представлен весь потенциал приложения и рассмотрен алгоритм работы со стороны клиента. Функционал для добавления фото путем передачи на сервер массива байт реализовать не удалось, принятые подход оказался не верным.

Четвертый этап – разработка серверного приложения, где была представлена архитектура приложения, структура библиотеки классов, рассмотрены Api сервера, а также был рассмотрен, алгоритм работы сервера синхронизации.

**4. Тестирование программного продукта**

ФИО

Подпись

Дата

Лист

1

КП 04.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

4. Тестирование программного продукта

Лит.

Листов

2

*БГТУ 74217078, 2020*

4.1 Негативное тестирование

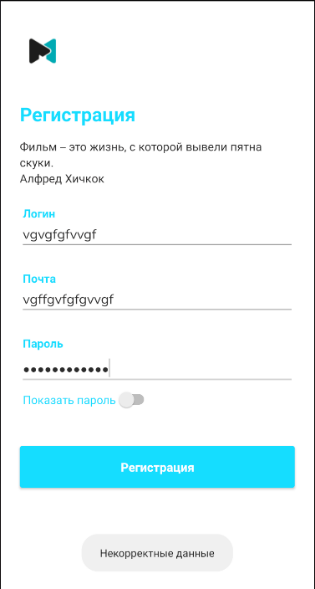
Данное программное средство тестировалось вручную, были выполнены все технические моменты, предусмотренные реализацией, а также были предприняты попытки нарушить работу приложения. В момент тестирования пользователи обращали внимание, что существуют различные условия при заполнении различных полей и ограничения на них. На рисунке 4.1 при вводе некорректных данных в поля логина почты и пароля будет выдаваться предупреждение «Некорректные данные».

Рисунок 4.1 – Проверка корректности ввода данных

На рисунке 4.2 приведена попытка регистрации пользователя под уже существуем email либо login будут выдаваться соответствующие предупреждения

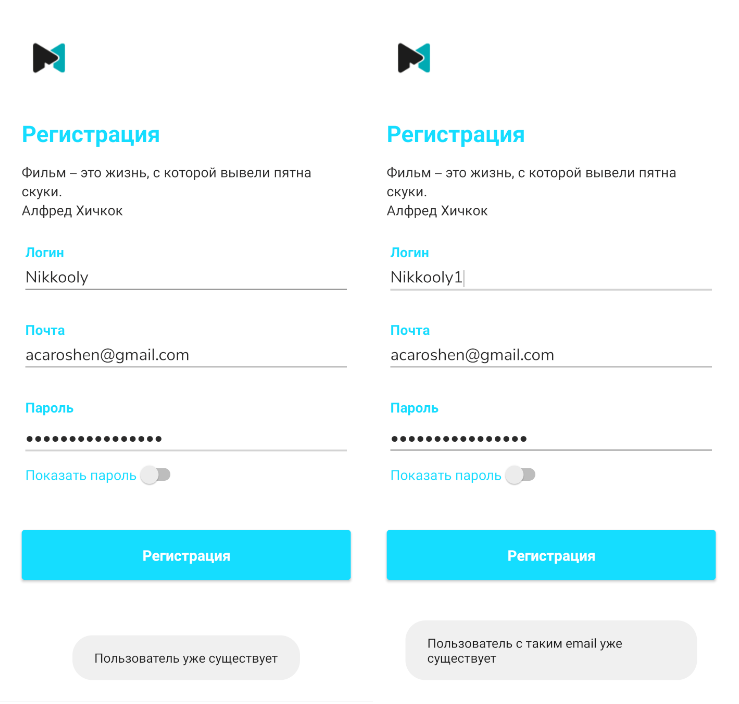


Рисунок 4.2 – Проверка на существование пользователей

При нажатии на кнопку «Скрепки» должна отображаться информация о дате, городе и регионе.

На рисунке 4.3 приведена форма в которой также присутствует проверка, и в случае, когда пользователь не выбирает ни одного места на сеанс либо попытается выбрать больше 7 мест одновременно будет показываться соответствующее предупреждение

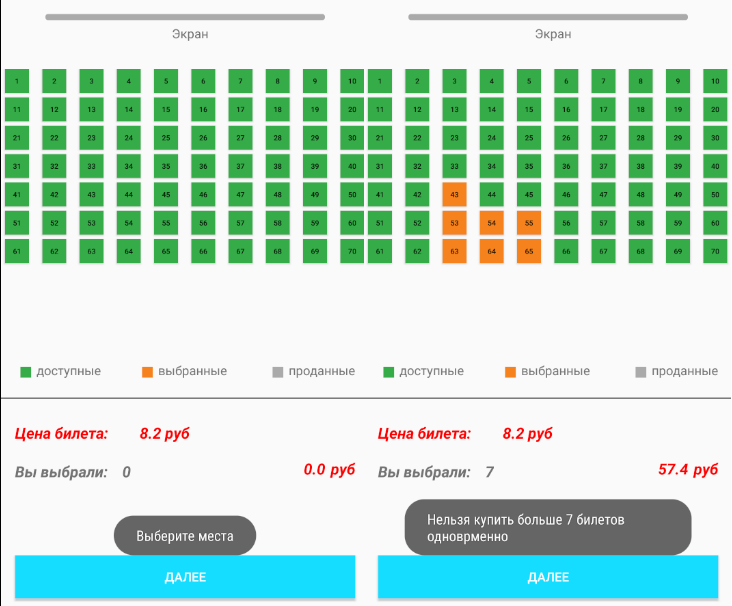


Рисунок 4.3 – Проверки корректности выбора мест

4.2 Выводы по разделу

В данной главе было рассмотрено негативное тестирование наиболее уязвимых компонентов мобильного приложения.

Была произведена проверка советующего предупреждений о том, что пользователь не сможет зарегистрироваться или авторизоваться, не заполнив соответствующие поля, а также не сможет зарегистрироваться если существует пользователь с соответствующим логином и почтой.

Также были проверены соответствующие ограничения на выбор количества места при бронировании билета на сеанс.

**5. Инструкция пользователя**

ФИО

Подпись

Дата

Лист

1

КП 05.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

3 Реализация программного средства

Лит.

Листов

8

*БГТУ 74217078, 2020*

Данное приложение позволяет пользователю бронировать билеты на сеанс, также просматривать забронированные билеты, предоставлять актуальную информацию по фильмам, а также сеансам по ним.

На рисунке 5.1 приведена форма авторизации. Вход в приложение осуществляется при помощи данной формы

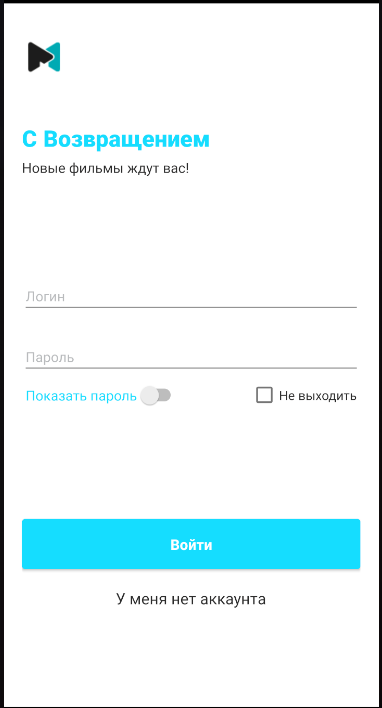


Рисунок 5.1 – Форма авторизации

Если вы хотите воспользоваться приложением без авторизации нажать на иконку приложения, вы будете перенаправлены на главный экран, однако вы не сможете увидеть купленные вами билеты, также в форме покупки билета вам придется вручную вводить данные электронной почты, на которую будет отправляться билет.

Если вы не зарегистрированы, можете нажать кнопку «Войти» и попасть на экран регистрации, приведенный на рисунке 5.2, где потребуется ввести предпочитаемый логин, пароль и вашу почту. Если все данные введены корректно и не существует пользователя с таким же логином или паролем вы будете зарегистрированы и откроется форма авторизации. Изображение с формой регистрации приведено на рисунке.

На рисунке 5.3 приведена форма авторизации. В форме авторизации вам необходимо ввести свой логин и пароль, в случае корректного ввода данных, а также подтверждения авторизации вы попадете на главное окно приложения.

При авторизации вы также можете пометить «галочкой» пункт не выходить, в таком случае при последующем использовании приложения вам не придется проходить авторизацию.

Рисунок, иллюстрирующий форму регистрации.

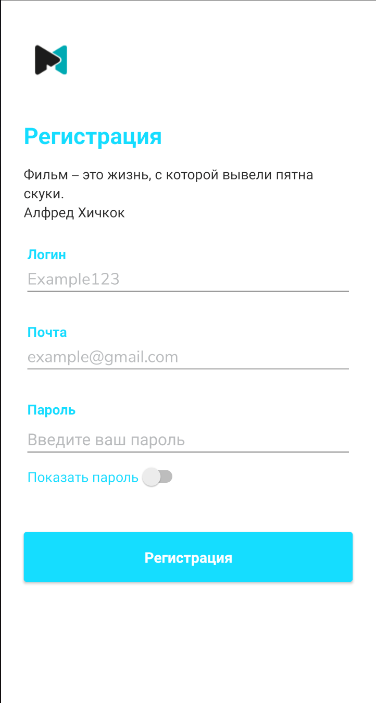


Рисунок 5.2 – Форма регистрации

Рисунок, иллюстрирующий главное окно приложения.

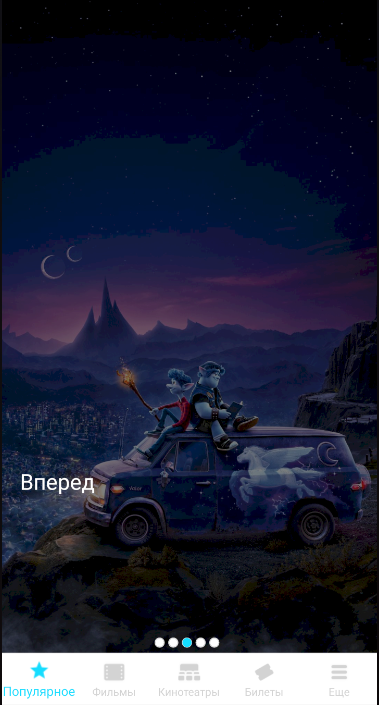


Рисунок 5.3 – Главное окно приложения

В главном окне приложения вы можете увидеть фильмы доступные в кинотеатре, также навигационное меню при помощи которого осуществляется навигация по приложению.

Навигационное меню состоит из 5 пунктов, в первом из них под названием «Популярное» отображается популярные фильмы, далее представлен пункт «Фильмы» в котором представлены все фильмы, именно через эту форму будет начинать осуществляться бронирование, она будет рассмотрена позже. Далее на рисунке 5.4 приведена форма «Кинотеатры», в ней перечислены все кинотеатры имеющиеся у данной сети

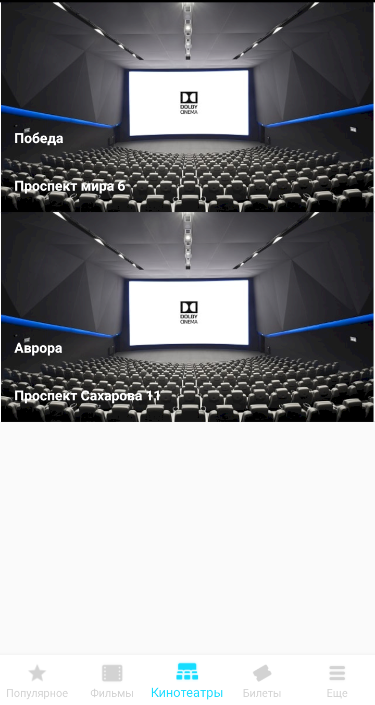


Рисунок 5.4 – Кинотеатры сети

Далее на рисунке 5.5 представлен пункт «Еще», в нем вы можете узнать всю информацию о приложении, а также выйти из приложения, либо из аккаунта если вы авторизовались

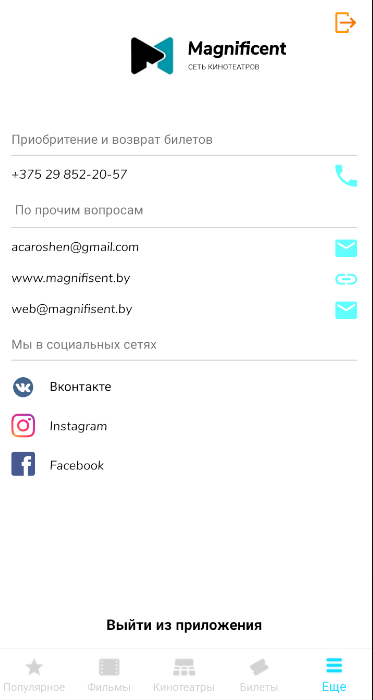


Рисунок 5.5 – Форма информации о приложении

Далее приведен пункт «Билеты» на рисунке 5.6. В данном компоненте будут отображаться купленные вами билеты. Если вы не производили авторизацию, то данный компонент будет пустым.

Справа компонент авторизованного пользователя, слева компонент в случае если не было произведено бронирование билетов, либо не произведена авторизация с учетом всех требований.

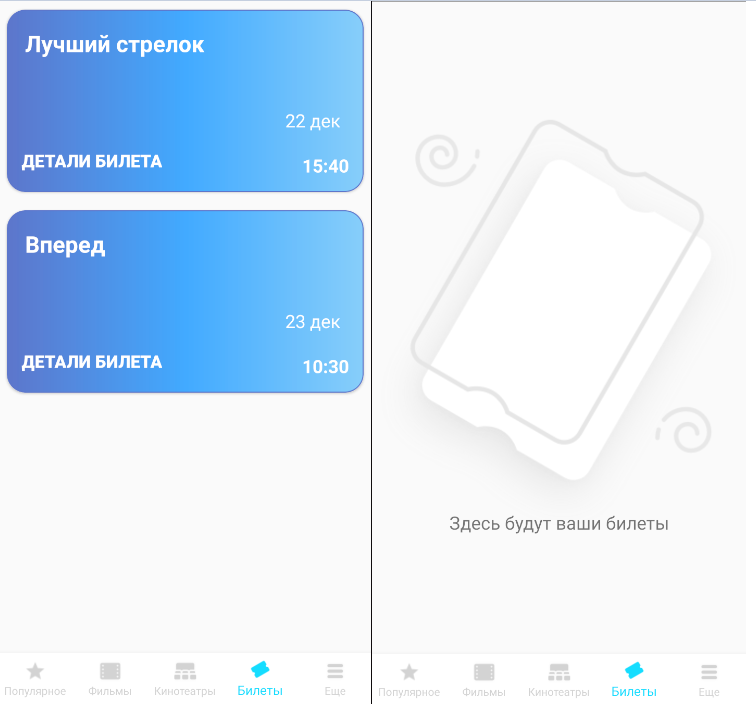


Рисунок 5.6 – Форма отображения купленных билетов

Данный компонент наряду с компонентом «Еще» будут работать в автономном режиме, в случае если вы прошли авторизацию, остальные компоненты не доступны в автономном режиме.

Если сеанс закончился появляется возможность оставить отзыв путем нажатия на кнопку отзыва, в результате будет отображаться форма на рисунке 5.7 для добавления отзыва по фильму, если отзыв был добавлен будет отображаться предупреждение об уже добавленном отзыве.

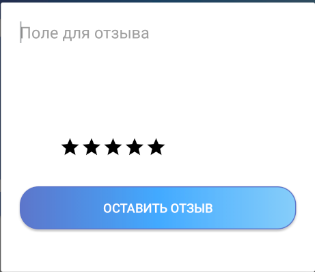


Рисунок 5.7 – Форма отзыва

На рисунке 5.8 приведен пункт «Фильмы», в данном компоненты отображаются текущие фильмы, а также рейтинг данных фильмов, по мнению пользователей с постером к фильму, именем фильма и годом выхода

Рейтинг формируется путем вычисления средней оценки пользователей по данному фильму. Все доступные в данной сети кинотеатров фильмы будут отображаться на данной форме.

При выборе фильма пользователь имеет возможно проанализировать целесообразность бронирования исходя из популярности того или иного фильма.

Простой визуальный интерфейс поможет быстро выбрать нужный фильм исходя из требуемых критериев.

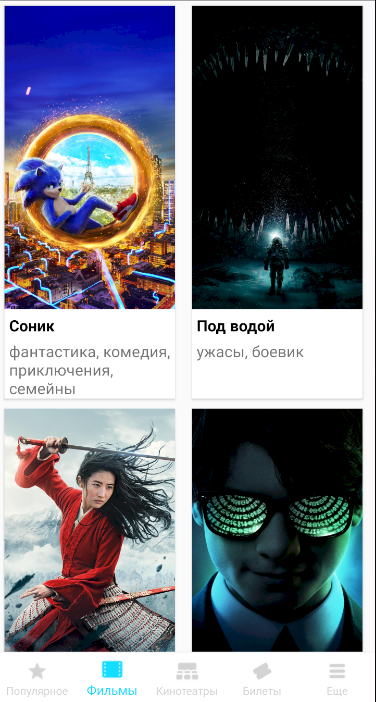


Рисунок 5.8 – Форма для отображения фильмов

При нажатии на фильм будет открываться форма с подробной информацией о фильме, а также при нажатии кнопки отзывов, можно прочитать отзывы по данному фильму от пользователей уже посмотревших его. На рисунке 5.11 приведено диалоговое окно, которое будет отображаться, если сеансы на данный фильм отсутствуют.

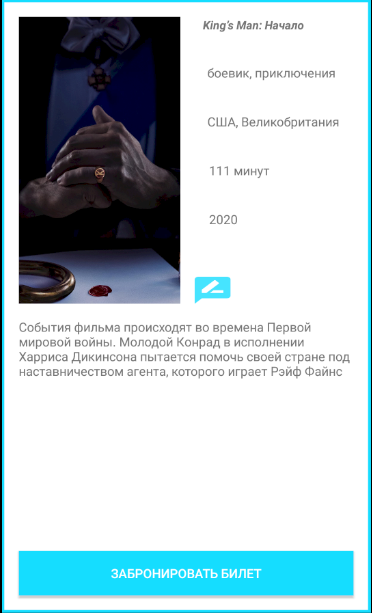


Рисунок 5.9 – Форма информации о фильме

Форма на рисунке 5.10 появляется при нажатии кнопки «отзыв» и отображает окно с отзывами пользователей.



Рисунок 5.10 – Форма отзыва пользователя

Форма на рисунке 5.11 будет отображаться в случае отсутствия сеансов на фильм. При нажатии кнопки «OK» будет осуществлен переход к списку фильмов, если нажать «Cancel» вы останетесь на форме с информации о фильме.



Рисунок 5.11 – Форма отсутствия сеансов

При нажатии кнопки «Забронировать билет» будет осуществлен переход к форме выбора сеанса. Необходимо выбрать кинотеатр из выпадающего списка, далее выбрать удобную дату просмотра при помощи календаря и время.

На рисунке 5.12 отображаются залы в которых будут проходить сеансы

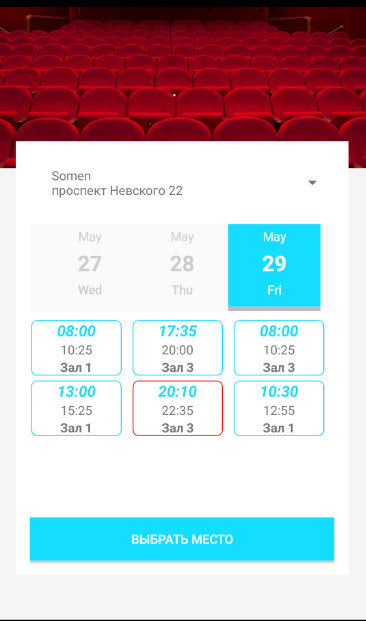


Рисунок 5.12 – Форма выбора сеанса

После выбора удобного времени сеанса и кинотеатра будет осуществлен переход к форме выбора места на сеансе изображенной на рисунке 5.13. Если не выбрать какой-либо пункт будет отображаться предупреждение.



Рисунок 5.13 – Форма выбора места

Выбранное место помечается оранжевым цветом, доступные для выбора места помечены зеленым цветом, забронированные места помечены серым. Одновременно можно забронировать не более 7 мест. Также на данной форме отображается количество выбранных мест и их общая стоимость.

После выбора необходимых мест и нажатия кнопки «Далее» будет осуществлен переход к финальной форме изображенной на рисунке 5.14, на которой можно просмотреть всю информацию по билету, в случае если пользователь не был авторизован требуется ввести в поле электронной почты свою электронную почту, в случае если была произведена авторизация, данное поле будет заполнено. После нажатия кнопки «Забронировать места» будет осуществлена бронь места, а также отправка письма на указанную почту со всеми необходимыми деталями билета.

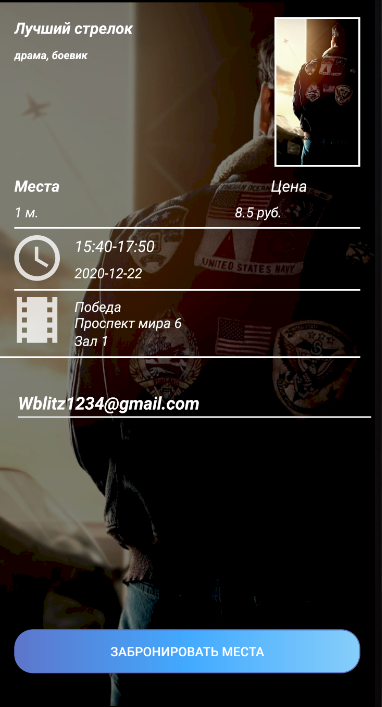


Рисунок 5.14 – Форма подтверждения бронирования

После чего будет осуществлен переход на главную страницу приложения.

5.1 Выводы по разделу

В данной главе было описано руководство пользователя.

Данное руководство позволяет пользователю узнать требования для установки приложения на телефон, а также как использовать все функции приложения. Были предоставлены скриншоты самого приложения для большей наглядности использования.

Заключение

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Ярмолик Н.С.

Провер.

Колмаков М.В.

Н. контр.

Колмаков М.В.

Утверд.

Колмаков М.В.

Заключение

Лит.

Листов

1

*БГТУ 74217078, 2020*

В курсовой работе описано и реализовано Android приложение, написанное на языке Java. При разработке хранилища данных использовалась удаленная база данных MS SQL. При разработке серверной части проекта использовалась технология ASP.NET Core Web API, для взаимодействия с базой данных со стороны сервера использовался Entity Framework.

В ходе выполнения курсовой работы было создано приложение для трех типов пользователей: гостя, пользователя и менеджера, которое выполняет следующие функции:

* выбора кинотеатра и даты сеанса
* удобный выбор мест на сеанс
* удобная форма для просмотра купленных билетов
* получение письма на почту с информацией о билете
* полноценная регистрация с авторизацией
* возможность оставить отзыв
* подробное описание фильмов и их рейтинг
* просмотр отзывов к фильмам
* добавление, удаление и редактирование фильмов
* добавление, удаление и редактирование кинотеатров
* добавление, удаление и редактирование залов
* добавление сеансов

Были решены следующие поставленные задачи:

* обзор аналогов;

В результате было рассмотрено 3 приложения и были выявлены их положительные и отрицательные стороны. Эта информация была использована для улучшения функционала приложения и пользовательского интерфейса.

* проектирование приложения;

Был спроектирован каждый отдельный компонент приложения. В мобильном приложении был определен основной функционал, также были разработаны необходимые диаграммы. При проектировании базы данных, были определены необходимые таблицы и разработана схема базы данных. В серверном приложении был определен архитектурный стиль взаимодействия между клиентом и сервером, формат передаваемых данных и HTTP методы для работы с серверной частью.

* реализация приложения;

Были реализованы базы данных на стороне сервера и клиента.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список использованных источников

ФИО

Подпись

Дата

Лист

1

КП 00.00.ПЗ

Разраб.

Ярмолик Н.С..

Провер.

Колмаков М.В..

Н. контр.

Колмаков М.В..

Утверд.

Колмаков М.В.

Список использованных источников

Лит.

Листов

1

*БГТУ 74217078, 2020*

1. Программирование под ОС Андроид [Электронный ресурс] / Metanit.com. – Режим доступа: https://metanit.com/java/android. Дата доступа: 06.04.2018
2. Руководство по ASP.NET Core 3 [Электронный ресурс] / Metanit.com. – Режим доступа: https://metanit.com/sharp/aspnet5. Дата доступа: 23.01.2020
3. Руководство по работе с датой и временем c# [Электронный ресурс] /blog.rc21net.ru – Режим доступа: https://blog.rc21net.ru/csharp-date-and-time/
4. Руководство по работе с OkHttp [Электронный ресурс] Режим доступа https://square.github.io/okhttp. Дата доступа 22.04.2019
5. Руководство по работе с SOAP на ASP.NET, Android [Электронный ресурс] Режим доступа: [https://www.codeproject.com/Articles/304302/Calling-Asp-Net-Webservice-ASMX-From-an-Android-Ap. Дата доступа 15.08.2013](https://www.codeproject.com/Articles/304302/Calling-Asp-Net-Webservice-ASMX-From-an-Android-Ap.%20Дата%20доступа%2015.08.2013)
6. Дон Гриффитс. Head First. Программирование на Андроид / Гриффитс Дон, Гриффитс Дэвид – СПб Питер, 2016. –709 с.
7. Герберт Шилдт. Java 8. Руководство для начинающих / Герберт Шилдт – И.Д. Вильямс, 2015. –720 с.
8. Mark L. Murphy. Busy Coder’s Guide to Android Development/ Mark L. Murphy – CommonsWare, 2019. –4419 с.

**ПРИЛОЖЕНИЕ А**

Листинг создания таблиц базы данных

create table cinema\_info(

id uniqueidentifier primary key NOT NULL,

name nvarchar(50) NOT NULL,

adress nvarchar(MAX) NOT NULL)

create table roles(

id int primary key identity(1,1),

role int NOT NULL)

create table user\_data(

id uniqueidentifier primary key NOT NULL,

login nvarchar(50) NOT NULL,

email nvarchar(50) NOT NULL,

password nvarchar(50) NOT NULL,

role\_id int foreign key references roles(id))

create table hall\_info(

id uniqueidentifier primary key NOT NULL,

cinema\_id uniqueidentifier NOT NULL,

name nvarchar(50) NOT NULL ,

places int NOT NULL)

create table film\_info(

id uniqueidentifier primary key NOT NULL,

poster nvarchar(max) NOT NULL,

year int NOT NULL,

country nvarchar(MAX) NOT NULL,

duration int NOT NULL,

genre nvarchar(MAX) NOT NULL,

name nvarchar(MAX) NOT NULL,

description text)

create table rating(

id uniqueidentifier primary key,

film\_id uniqueidentifier NOT NULL,

user\_id uniqueidentifier NOT NULL,

rating float NOT NULL)

create table seance(

id uniqueidentifier primary key NOT NULL,

start\_time smalldatetime NOT NULL,

end\_time smalldatetime NOT NULL,

hall\_id uniqueidentifier NOT NULL ,

film\_id uniqueidentifier NOT NULL ,

ticket\_price float NOT NULL)

create table tickets(

id uniqueidentifier primary key NOT NULL,

user\_id uniqueidentifier NOT NULL,

seance\_id uniqueidentifier NOT NULL,

place int NOT NULL)

Листинг описывающий работу с 4 основными методами HTTP в контроллере

public class CinemaController : Controller

{

CinemaContext cinemaContext = new CinemaContext();

[HttpGet]

public string Get()

{

return JsonConvert.SerializeObject(cinemaContext.CinemaInfo.Select(x => new { x.Id, x.Name, x.Adress }));

}

[HttpGet("{id}")]

public string Get(string id)

{

if (cinemaContext.CinemaInfo.Any(x => x.Id.ToString().Equals(id)))

{

try

{

return JsonConvert.SerializeObject(

cinemaContext.CinemaInfo

.Select(x => new { x.Id, x.Name, x.Adress})

.Where(u => u.Id.ToString().Equals(id))

.First());

}

catch (Exception ex)

{

return JsonConvert.SerializeObject(ex.Message);

}

}

else

{

return JsonConvert.SerializeObject("Кинотеатр не найден");

}

}

// POST api/<controller>

[HttpPost]

public string Post([FromBody]CinemaInfo value)

{

if (!cinemaContext.CinemaInfo.Any(cinema => cinema.Name.Equals(value.Name)))

{

CinemaInfo cinema = new CinemaInfo();

cinema.Name = value.Name;

cinema.Adress = value.Adress;

cinema.Id = Guid.NewGuid();

try

{

cinemaContext.Add(cinema);

cinemaContext.SaveChanges();

return JsonConvert.SerializeObject("Кинотеатр успешно добавлен");

}

catch (Exception ex)

{

return JsonConvert.SerializeObject(ex.Message);

}

}

else

{

return JsonConvert.SerializeObject("Такой кинотеатр уже существует!");

}

}

[HttpPut("{id}")]

public string Put(string id, [FromBody]CinemaInfo value)

{

CinemaInfo cinema = cinemaContext.CinemaInfo.Where(u => u.Id.ToString().Equals(id)).First();

cinema.Name = value.Name;

cinema.Adress = value.Adress;

try

{

cinemaContext.Update(cinema);

cinemaContext.SaveChanges();

return JsonConvert.SerializeObject("Данные успешно изменены");

}

catch (Exception ex)

{

return JsonConvert.SerializeObject(ex.Message);

}

}

[HttpDelete("{id}")]

public string Delete(string id)

{

CinemaInfo cinema = cinemaContext.CinemaInfo.FirstOrDefault(x => x.Id.ToString().Equals(id));

if (cinema == null)

{

return JsonConvert.SerializeObject("Кинотеатр не найден!");

}

cinemaContext.CinemaInfo.Remove(cinema);

cinemaContext.SaveChangesAsync();

return JsonConvert.SerializeObject("Успешно удален!");

}

}

Листинг описывающий Веб-сервис, принимающий в качестве параметров широту и долготу, и возвращающий текущее время, город и область

[WebMethod]

public string SoapService(string x,string y)

{

string result = "";

string apiKey = "Z6NTGOCFXWWOBMWYLNB6PZLCHZ08R3KX";

string format = "json";

WebRequest request = WebRequest.Create("https://api.geodatasource.com/city?key=" + apiKey + "&format=" + System.Web.HttpUtility.UrlEncode(format) + "&lat=" + System.Web.HttpUtility.UrlEncode(x) + "&lng=" + System.Web.HttpUtility.UrlEncode(y));

WebResponse response = request.GetResponse();

using (Stream stream = response.GetResponseStream())

{

using (StreamReader reader = new StreamReader(stream))

{

string line = "";

while ((line = reader.ReadLine()) != null)

{

result = line;

}

}

}

response.Close();

dynamic jsonObj = JsonConvert.DeserializeObject(result);

string city = jsonObj.city;

string region=jsonObj.region;

return "Ваш регион: "+region +"\n"+ "Ваш город: "+city+"\n"+"Текущее время:"+ DateTime.Now.ToString("dd MMM yyy HH:mm"); }}

Метод используемый для отправки сообщения на электронную почту на стороне сервера

private static async Task SendEmailAsync(string toUser, string cinema,string hall,string film, string date,string startTime,string EndTime, string places)

{

MailAddress from = new MailAddress("MagnifisentCinema@yandex.ru", "Magnifisent cinema");

MailAddress to = new MailAddress(toUser);

MailMessage m = new MailMessage(from, to);

m.Subject = "Покупка билета";

m.Body = $"Спасибо за покупку билетов в нашем кинотеатре. Информация по билету:<br> Кинотеатр: {cinema} <br> Зал: {hall} <br> Фильм: {film} <br> Дата: {date} <br> Начало сеанса: {startTime} <br> Конец сеанса: {EndTime} <br> Места: {places} <br> QR код находится во вложении письма";

m.IsBodyHtml = true;

m.Attachments.Add(new Attachment("F:/kursovoi/images/qr-code.png"));

SmtpClient smtp = new SmtpClient("smtp.yandex.by", 587);

smtp.Credentials = new NetworkCredential("MagnifisentCinema@yandex.ru", "123456789Nick");

smtp.EnableSsl = true;

await smtp.SendMailAsync(m);

}

Использования метода GET в мобильном приложении

**public void** loadFilmsInfo(){  
  
 Call<List<FilmResponse>> call=**iMyApi**.getFilms();  
 call.enqueue(**new** Callback<List<FilmResponse>>() {  
 @Override  
 **public void** onResponse(Call<List<FilmResponse>> call, Response<List<FilmResponse>> response) {  
 **if**(!response.isSuccessful()){  
 Toast.*makeText*(getContext(), response.code(), Toast.***LENGTH\_LONG***).show();  
 }  
 **posts**=response.body();  
 **for**(FilmResponse post : **posts**){  
 **nameList**.add(post.getName());  
 **idList**.add(post.getId());  
 **countryList**.add(post.getCountry());  
 **genreList**.add((post.getGenre()));  
 **descriptionList**.add(post.getDescription());  
 **durationList**.add(post.getDuration());  
 **posterList**.add(post.getPoster());  
 **yearList**.add(post.getYear());  
 **ratingList**.add(post.getRating());  
 }  
 **for**(**int** i=0;i<**posterList**.size();i++){  
 **try** {  
 **film**.add(**new** FilmModel(**nameList**.get(i), **yearList**.get(i).toString(), **ratingList**.get(i),**posterList**.get(i),**idList**.get(i)));*//getBitmapFromURL(posterList.get(i))));* }  
 **catch** (Exception ex){  
 Toast.*makeText*(getContext(),ex.getMessage()+**" Exception"**,Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
 **filmAdapter**=**new** FilmAdapter(**film**);  
 **recyclerView**.setAdapter(**filmAdapter**);  
 }  
 @Override  
 **public void** onFailure(Call<List<FilmResponse>> call, Throwable t) {  
 Toast.*makeText*(getContext(), t.getMessage(), Toast.***LENGTH\_LONG***).show();  
 }  
 });  
}

Использование метода GET с параметром id в мобильном приложении

**private void** loadFilm(String id){  
 AlertDialog alertDialog = **new** SpotsDialog.Builder()  
 .setContext(getContext())  
 .build();  
 alertDialog.show();  
 Call<FilmResponse> call=**iMyApi**.getFilm(id);  
 call.enqueue(**new** Callback<FilmResponse>() {  
 @Override  
 **public void** onResponse(Call<FilmResponse> call, Response<FilmResponse> response) {  
 **yearFilm**.setText(response.body().getYear().toString());  
 **genreFilm**.setText(response.body().getGenre());  
 **nameFilm**.setText(response.body().getName());  
 **descriptionFilm**.setText(response.body().getDescription());  
 **countryFilm**.setText(response.body().getCountry());  
 **durationFilm**.setText(response.body().getDuration().toString()+**" минут"**);  
 **posterFilm**=response.body().getPoster();  
 Picasso.*get*().load(response.body().getPoster()).into(**imageView**);  
 alertDialog.dismiss();  
 }  
  
 @Override  
 **public void** onFailure(Call<FilmResponse> call, Throwable t) {  
 Toast.*makeText*(getContext(), t.getMessage(), Toast.***LENGTH\_LONG***).show();  
 }  
 });  
}

Использование POST в мобильном приложении

**if**(!**choose**.equals(**""**) && !nameOfHall.equals(**""**) && !**chooseCount**.equals(**""**)){  
 String cinemaId=**idList**.get(position);  
 AlertDialog alertDialog = **new** SpotsDialog.Builder()  
 .setContext(getContext())  
 .build();  
 alertDialog.show();  
 HallInfo hall=**new** HallInfo(cinemaId,nameOfHall, Integer.*parseInt*(**chooseCount**));  
 **compositeDisposable**.add(**iMyApi**.addHall(hall)  
 .subscribeOn(Schedulers.*io*())  
 .observeOn(AndroidSchedulers.*mainThread*())  
 .subscribe(**new** Consumer<String>() {  
 @Override  
 **public void** accept(String s) **throws** Exception {  
 Toast.*makeText*(getContext(), s, Toast.***LENGTH\_SHORT***).show();  
 alertDialog.dismiss();  
 }  
 }, **new** Consumer<Throwable>() {  
 @Override  
 **public void** accept(Throwable throwable) **throws** Exception {  
 Toast.*makeText*(getContext(), throwable.getMessage(), Toast.***LENGTH\_SHORT***).show();  
 }  
 }));  
}  
**else**{  
 Toast.*makeText*(getContext(), **"Заполните поля"**, Toast.***LENGTH\_SHORT***).show();  
}

Использование PUT в мобильном приложении

**private void** updatePut(String id,String name){  
 AlertDialog alertDialog = **new** SpotsDialog.Builder()  
 .setContext(getContext())  
 .build();  
 alertDialog.show();  
 FilmInfo film= **new** FilmInfo(  
 name,  
 Integer.*parseInt*(**year**.getText().toString()),  
 **country**.getText().toString(),  
 Integer.*parseInt*(**duration**.getText().toString()),  
 **genre**.getText().toString(),  
 **description**.getText().toString(),  
 **poster**.getText().toString()  
  
 );  
 Call<FilmInfo> call= **iMyApi**.updateFilm(id,film);  
 call.enqueue(**new** Callback<FilmInfo>() {  
 @Override  
 **public void** onResponse(Call<FilmInfo> call, Response<FilmInfo> response) {  
 *//Toast.makeText(getContext(), response.message(), Toast.LENGTH\_LONG).show();* alertDialog.dismiss();  
 }  
  
 @Override  
 **public void** onFailure(Call<FilmInfo> call, Throwable t) {  
 *// Toast.makeText(getContext(), t.getMessage(), Toast.LENGTH\_LONG).show();* alertDialog.dismiss();  
  
 }  
 });  
}

Использование метода POST

Call<Void> call = **iMyApi**.deletePost(**idList**.get(position));  
call.enqueue(**new** Callback<Void>() {  
 @Override  
 **public void** onResponse(Call<Void> call, Response<Void> response) {  
 Toast.*makeText*(getContext(), response.message(), Toast.***LENGTH\_LONG***).show();  
 loadFilms();  
 alertDialog.dismiss();  
 }  
  
 @Override  
 **public void** onFailure(Call<Void> call, Throwable t) {  
 Toast.*makeText*(getContext(), t.getMessage(), Toast.***LENGTH\_LONG***).show();  
 alertDialog.dismiss();  
 }  
});

Листинг метода хэширования MD5 в мобильном приложении

**public static** String md5(**final** String s) {  
 **final** String MD5 = **"MD5"**;  
 **try** {  
 *// Create MD5 Hash* MessageDigest digest = java.security.MessageDigest  
 .*getInstance*(MD5);  
 digest.update(s.getBytes());  
 **byte** messageDigest[] = digest.digest();  
  
 *// Create Hex String* StringBuilder hexString = **new** StringBuilder();  
 **for** (**byte** aMessageDigest : messageDigest) {  
 String h = Integer.*toHexString*(0xFF & aMessageDigest);  
 **while** (h.length() < 2)  
 h = **"0"** + h;  
 hexString.append(h);  
 }  
 **return** hexString.toString();  
  
 } **catch** (NoSuchAlgorithmException e) {  
 e.printStackTrace();  
 }  
 **return ""**;  
}

Код описывающий работу с SOAP в рамках android

**public final** String **SOAP\_ACTION** = **"http://tempuri.org/SoapService"**;  
  
**public final** String **OPERATION\_NAME** = **"SoapService"**;  
  
**public final** String **WSDL\_TARGET\_NAMESPACE** = **"http://tempuri.org/"**;  
  
**public final** String **SOAP\_ADDRESS** = **"http://10.0.2.2/WebServiceDemo/Service.asmx"**;  
**public** CallSoap()  
{  
}  
**public** String Call(String a,String b)  
{  
 SoapObject request = **new** SoapObject(**WSDL\_TARGET\_NAMESPACE**,**OPERATION\_NAME**);  
 PropertyInfo pi=**new** PropertyInfo();  
 pi.setName(**"x"**);  
 pi.setValue(a);  
 pi.setType(String.**class**);  
 request.addProperty(pi);  
 pi=**new** PropertyInfo();  
 pi.setName(**"y"**);  
 pi.setValue(b);  
 pi.setType(String.**class**);  
 request.addProperty(pi);  
  
 SoapSerializationEnvelope envelope = **new** SoapSerializationEnvelope(  
 SoapEnvelope.***VER11***);  
 envelope.**dotNet** = **true**;  
  
 envelope.setOutputSoapObject(request);  
  
 HttpTransportSE httpTransport = **new** HttpTransportSE(**SOAP\_ADDRESS**);  
 Object response=**null**;  
 **try** {  
 httpTransport.call(**SOAP\_ACTION**, envelope);  
 response = envelope.getResponse();  
 }  
 **catch** (Exception exception)  
 {  
 response=exception.toString();  
 }  
 **return** response.toString();  
}

Код для отрисовки элемента

@Override  
**public void** onBindViewHolder(@NonNull TicketViewHolder holder, **int** position) {  
 YoYo.*with*(Techniques.***FadeInRight***).playOn(holder.**cardView**);  
 holder.**nameFilm**.setText(**list**.get(position).getFilmName());  
 holder.**dateSeance**.setText(**list**.get(position).getDate());  
 holder.**timeSeance**.setText(**list**.get(position).getTime());  
 holder.**button**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 Call<List<PlacesResponse>> call=**iMyApi**.getPlaces(Login.*userId*,**list**.get(position).getSeanceId());  
 ArrayList<Integer> loadPlace=**new** ArrayList<>();  
 StringBuilder stringBuilder=**new** StringBuilder();  
 call.enqueue(**new** Callback<List<PlacesResponse>>() {  
 @Override  
 **public void** onResponse(Call<List<PlacesResponse>> call, Response<List<PlacesResponse>> response) {  
 **for** (PlacesResponse placesResponse : response.body()) {  
 loadPlace.add((placesResponse.getPlace()));  
 }  
 **for** (**int** i = 0; i < loadPlace.size(); i++) {  
 stringBuilder.append(loadPlace.get(i)+**","**);  
 }  
 String places=stringBuilder.toString();  
 String placesFinal=places.substring(0,places.length()-1);  
 holder.**reservedSeats**.setText(placesFinal);  
 holder.**cinema**.setText(**list**.get(position).getCinema());  
 holder.**hall**.setText(**list**.get(position).getHall());  
 holder.openPopup();  
 }  
 @Override  
 **public void** onFailure(Call<List<PlacesResponse>> call, Throwable t) {  
  
 }  
 });  
  
 }  
 });  
 **try** {  
 Date endDate= **simpleDateFormat**.parse(**list**.get(position).getEndTime());  
 String s=**simpleDateFormat**.format(**date**);  
 Date currentDate=**simpleDateFormat**.parse(s);  
 **if**(currentDate.after(endDate)){  
 holder.**reviewButton**.setVisibility(View.***VISIBLE***);  
 holder.**reviewButton**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 holder.openReview(**list**.get(position).getFilmId());  
 }  
 });  
 }  
 **else**{  
 holder.**reviewButton**.setVisibility(View.***GONE***);  
 }  
 } **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
  
}