Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК5 «Системы обработки информации»***

Р А С Ч Е Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовой работе на тему:

Веб-приложение для автоматизации бронирования билетов в кинотеатре

по дисциплине ***Архитектура автоматизированных систем обработки информации и управления***

Студент: гр. ИУК5-51Б _____ (Поляков Р. А.)
(подпись) (Ф.И.О.)

Руководитель: _____ (Галущенко А. Н.)
(подпись) (Ф.И.О.)

Оценка руководителя _____ баллов _____
30-50 (дата)

Оценка защиты _____ баллов _____
30-50 (дата)

Оценка проекта _____ баллов _____
(оценка по пятибалльной шкале)

Комиссия: _____ (_____)
(подпись) (Ф.И.О.)

_____ (_____)
(подпись) (Ф.И.О.)

_____ (_____)
(подпись) (Ф.И.О.)

Калуга, 2020

Калужский филиал
федерального государственного бюджетного образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой **ИУК5-КФ**

_____(_____)

«____»_____2020 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине ***Архитектура автоматизированных систем обработки информации и управления***

Студент *Поляков Р. А., ИУК5-51Б*
(фамилия, инициалы, индекс группы)

Руководитель *Галущенко А. Н.*
(фамилия, инициалы)

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 10 нед., 100% к 14 нед.

1. Тема курсовой работы

Веб-приложение для автоматизации бронирования билетов в кинотеатре.

2. Техническое задание

Разработать веб-приложение для автоматизации бронирования билетов в кинотеатре и интегрировать его со спроектированной базой данных.

3. Оформление курсовой работы

3.1. Расчетно-пояснительная записка на_____ листах формата А4.

3.2. Перечень графического материала КП (плакаты, схемы, чертежи и т.п.)_____

_____

_____

_____

_____

Дата выдачи задания «_____»_____2020 г.

Руководитель курсовой работы _____/_____/
(подпись) (Ф.И.О.)

Задание получил / / « » 2020 г.
(подпись) (Ф.И.О.)

Примечание:
Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

ОГЛАВЛЕНИЕ

1. Техническое задание 5
1.1 Наименование системы 5
1.2 Основания для разработки 5
1.3 Исполнитель 5
1.4 Назначение и цель разработки системы 5
1.5 Характеристика объектов автоматизации 5
1.6 Требования к системе 6
1.6.1. Требования к задачам, подлежащим решению 6
1.6.2. Требования к архитектуре АСОИ 6
1.6.3. Требования к составу программных компонентов 6
1.6.4. Требования к входным/выходным данным 6
1.6.5. Требования к временным характеристикам 7
1.6.6. Требования к составу технических средств 7
1.6.7. Требования к основным функциям системы 7
1.6.8. Состав и содержание работ по созданию системы 7
1.7 Этапы работ 7
1.8 Порядок контроля и приемки автоматизированной системы 8
1.9 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы действия 9
1.10 Требования к документированию 9
1.11 Дополнительные условия 9
1.12 Источники разработки 10
2. Исследовательская часть 11
2.1. Постановка задачи 11
2.2. Перечень задач, подлежащих решению в процессе разработки 11
2.3. Анализ существующих аналогов 12

2.4. Предлагаемые функции приложения для реализации 18
2.5. Обоснование выбора языка разработки 19
2.6. Обоснование выбора СУБД 19
2.7. Библиотека React 20
2.8. Фреймворк Express 20
2.9. NodeJS 20
2.10. Node-postgres 21
2.11. Язык дизайна Material-UI 21
2.12. Выводы 21
3. Проектно - конструкторская часть 23
3.1 Общие сведения 23
3.2 Архитектура веб-приложения 23
3.3 Концептуальная модель базы данных «Кинотеатр» 24
3.4 Логическая модель базы данных «Кинотеатр» 25
3.5 Физическое проектирование базы данных «Кинотеатр» 27
3.6 Серверная часть веб-приложения 32
3.7 Клиентская часть веб-приложения 40
3.8 Взаимодействие частей веб-приложения 43
3.9 UML диаграмма размещения 46
3.10 UML диаграмма деятельности 46
4. Проектно – технологическая часть 48
4.1 Требования к аппаратной платформе 48
4.2 Тестирование и отладка рабочей программы 48
4.3 Руководство администратора 48
4.4 Руководство пользователя 49
Заключение 53
Список использованных источников 54

1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1 Наименование системы

Настоящее Техническое задание определяет требования и порядок создания веб-приложения для автоматизации бронирования билетов в кино.

1.2 Основания для разработки

Основанием для разработки веб-приложения для автоматизации бронирования билетов в кино является задание в соответствии с программой дисциплины «Архитектура автоматизированных систем обработки информации и управления»

1.3 Исполнитель

Исполнителем проекта является студент Калужского филиала МГТУ им. Н. Э. Баумана, факультета ИУК «Информатики и управления», кафедры ИУК5 «Системы обработки информации», группы ИУК5-51Б, Поляков Роман Андреевич.

1.4 Назначение и цель разработки системы

Разрабатываемое приложение предназначено для демонстрации современных подходов в веб-разработке.

Целью создания системы является закрепление полученных навыков работы с СУБД PostgreSQL, языком программирования JavaScript и изучение принципов взаимодействия БД PostgreSQL с технологиями веб-программирования.

1.5 Характеристика объектов автоматизации

Объектом автоматизации является процесс покупки билетов в кинотеатре.

1.6 Требования к системе

1.6.1. Требования к задачам, подлежащим решению

В ходе реализации программного продукта необходимо предусмотреть интерфейс веб-приложения, разработать модель данных и установить взаимодействие между frontend и backend частями.

1.6.2. Требования к архитектуре АСОИ

Разрабатываемая система должно быть построено с применением клиент-серверной архитектуры.

В качестве основного стека технологий использовать «PERN» (PostgreSQL, Express, React and Node.js), представленный на рисунке 1.

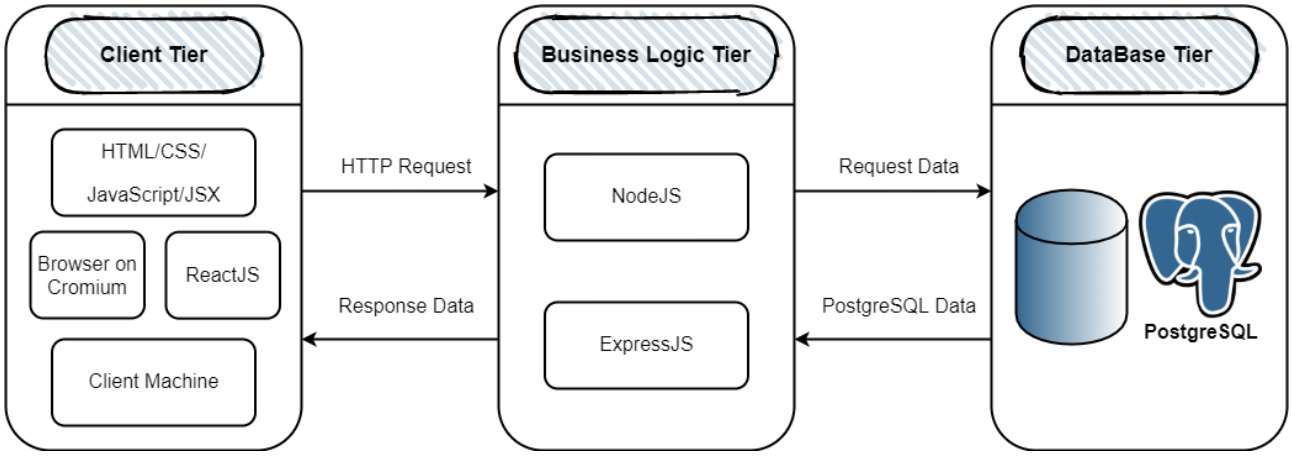

Рисунок 1 — Стек технологий «PERN»

1.6.3. Требования к составу программных компонентов

Программный продукт должен представлять собой протестированное веб-приложение, размещенное на удаленном сервере.

1.6.4. Требования к входным/выходным данным

Входные данные должны быть логичными и отображать то, что пользователь хочет получить в результате работы программы.

Выходные данные должны максимально правильно и полностью соответствовать действиям конечных пользователей.

1.6.5. Требования к временным характеристикам

Требования к временным характеристикам программы не предъявляются.

1.6.6. Требования к составу технических средств

Для построения программного продукта необходимы следующие аппаратно-технические и программные средства:

- ОС: Windows 10;
- Visual Studio Code;
- веб-браузер на основе Chromium;
- база данных PostgreSQL;
- СУБД pgAdmin.

1.6.7. Требования к основным функциям системы

Система должна содержать следующие функции:

- авторизация;
- регистрация;
- бронирование билетов;
- просмотр забронированных билетов.

1.6.8. Состав и содержание работ по созданию системы

Работы по созданию системы выполняются в четыре этапа:

25% к 4 нед., 50% к 7 нед., 75% к 10 нед., 100% к 14 нед.

1.7 Этапы работ

Разработка будет происходить в соответствие со следующим планом:

- 2 неделя (12 сентября). Разработка, согласование и утверждение темы, задания на разработку, технического задания (альфа);

- 4 неделя (26 сентября). Оформление ТЗ. Описание предметной области и требований к системе. Обзор существующих аналогов программному продукту, выбор программного обеспечения, библиотек для создания продукта. Концептуальная схема базы данных. Прототип и скетчи интерфейса;
- 7 неделя (17 октября). Оформление введения и исследовательской части. Обоснование выбора БД, логическая схема БД, физическая схема данных. Проектирование архитектуры системы, состава и методов взаимодействия компонентов. Демонстрация работы макета системы с БД;
- 10 неделя (7 ноября). Оформление проектно-конструкторской части. Демонстрация работающего приложения. Презентация (альфа). Тестирование и отладка приложения. Разработка эксплуатационной документации.
- 14 неделя (1 декабря). Демонстрация проекта. Защита.

1.8 Порядок контроля и приемки автоматизированной системы

Сопровождать разработку автоматизированной системой контролем и приемкой результатов работы на каждом этапе в соответствии с разделом 1.7.

В системе управление версий (Git) необходимы две ветки – dev и master. В первой будет вестись основная разработка системы, во второй будет находится отлаженная часть проекта, а в конечном результате – сам проект.

Сдача-приёмка работ производится поэтапно, в соответствии с учебным планом. Основанием для сдачи-приёмки работ служит «Отчёт о завершении работ по стадии», представляемый Исполнителем. Для сдачи-приемки представляется также документация, перечисленная в пункте 1.10 настоящего ТЗ.

1.9 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы действия

Для подготовки объекта автоматизации к вводу системы в действие должны быть проведены следующие мероприятия:

- приведение поступающей в систему информации (в соответствии с требованиями к информационному обеспечению) к виду, пригодному для обработки;
- изменения, которые необходимо осуществить в объекте автоматизации;
- создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям, содержащимся в ТЗ;

1.10 Требования к документированию

Должны быть разработаны следующие программные документы:

1. Расчетно-пояснительная записка:
   - техническое задание;
   - научно-исследовательская часть;
   - проектирований компонентов программного продукта;
   - проектно-технологическая часть.
2. Графическая часть.

1.11 Дополнительные условия

Клиентская часть представляет собой сайт, на котором размещена карусель фильмов, идущих в кино. Список фильмов возвращать из базы данных по прописанным SQL-запросам. Для фильма в карусели отображается миниатюра фото, заголовок, краткое описание. Реализовать возможность просмотреть описания конкретного фильма подробно. Реализовать функцию покупки билета.

Реализовать возможность авторизации для пользователя и просмотра купленных им билетов.

Серверная часть должна принимать данные от клиента и заполнять базу данных в соответствии с прописанными SQL запросы. Реализовать методы серверной части в соответствии с архитектурным стилем REST.

Клиентская часть приложения должна быть реализована с использованием следующих библиотек: React, React-Router, Lodash, Material-UI. Для сборки проекта использовать Webpack, Babel. Для линтинга проекта и соблюдения код-стайла использовать: ESLint, Eslint-config-airbnb.

Для приложения использовать монорепозиторий на GitHub.

1.12 Источники разработки

При разработке технического задания использованы следующие источники и литература:

- ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы»;
- ГОСТ 34.601-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Стадии создания»;
- Статья «Разработка Технического задания по ГОСТ 34 легко и просто» - https://habr.com/ru/post/432852/ ;
- Статья «Шаблон технического задания на АС по ГОСТ 34» - http://technicaldocs.ru/гост34/шаблоны/техническое_зада

2. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

2.1. Постановка задачи

Киноиндустрия, в общем и целом, является одним из главных развлечений для человека. Миллионы людей каждый день смотрят фильмы как в обычных кинотеатрах, так и онлайн-кинотеатрах, горячо обсуждают текущие кинопроекты в соцсетях, с нетерпением ждут выхода новых.

Обычный человек, когда идет в кинотеатр в первую очередь хочет получить удовольствие. Чтобы обеспечить комфорт зрителям владельцы кинотеатров используют системы кондиционирования, комфортные кресла, большие и качественные экраны, мощные аудиосистемы, но зачастую, по разным причинам не обращают внимание на такой важный процесс как покупка билета на киносеанс. До сих пор во многих кинотеатрах мы можем наблюдать очереди у касс перед началом сеанса, неудобные киноафиши и невозможность купить билет заранее через интернет. С решением этих проблем может справится веб-приложение, автоматизирующее примитивные процессы. Оно позволит зрителю посмотреть афишу выбрать необходимый фильм, место в зале и купить билет, а главное сделать это за несколько минут с помощью интуитивно понятного интерфейса.

Главной задачей курсовой работы является проектирование и создание веб-приложения с использованием базы данных для реализации возможностей просмотра киноафиши и покупки билета в кинотеатр.

2.2. Перечень задач, подлежащих решению в процессе разработки

Для выполнения поставленной задачи необходимо определить оптимальную архитектуру для построения веб-приложения и средства реализации программного продукта, соответствующего данной архитектуре. Также необходимо исследовать существующие аналоги приложений для кинотеатров, разработать концептуальные и физические модели проектирования базы данных для приложения, разработать базу данных для хранения

необходимой информации, спроектировать и разработать визуальную часть приложения и провести её тестирование.

2.3. Анализ существующих аналогов

Одним из вариантов веб-приложения для автоматизации работы кинотеатра является продукт компании «Платформа 24». Кинотеатр адаптирует готовый шаблон под свой бренд как конструктор. Это прекрасное решение для небольших кинотеатров, в котором реализованы все необходимые функции: показ афиши, покупка и просмотр уже купленных билетов, просмотр информации о кинотеатре. Из минусов я бы отметил отсутствие идентичности у сайта и не эргономичное размещение сетки фильмов.

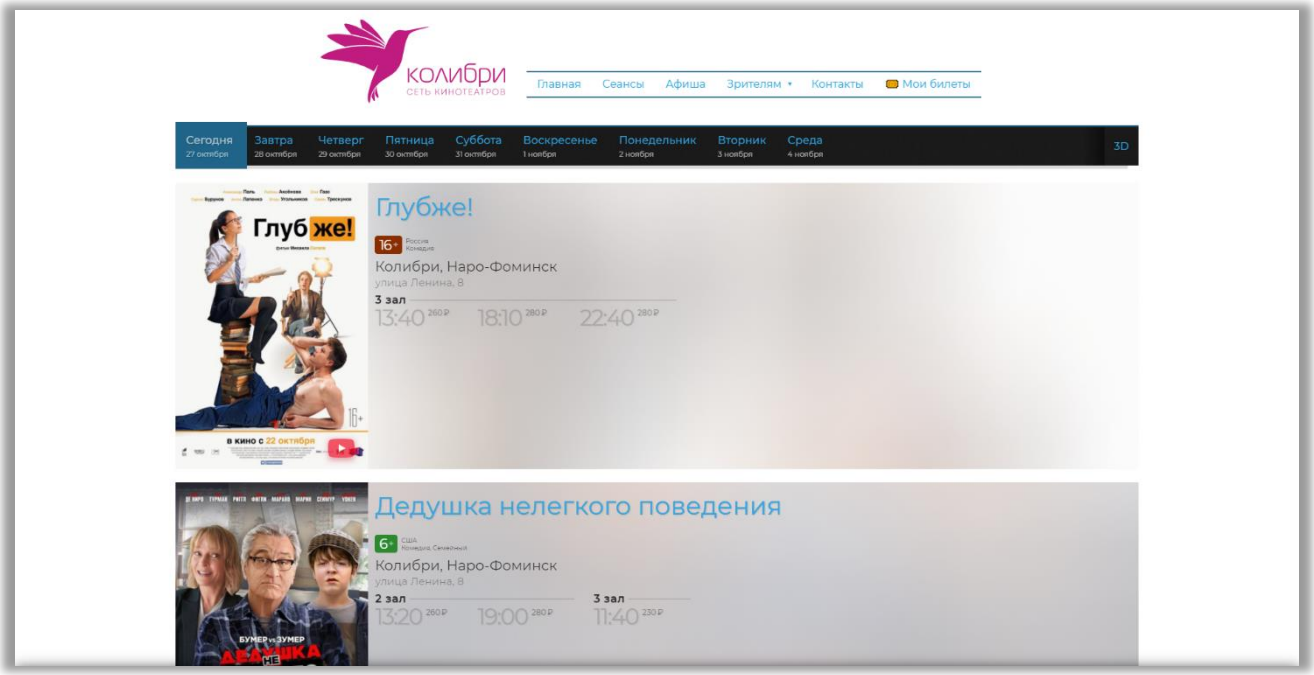

Рисунок 2 — Сеансы кинотеатра

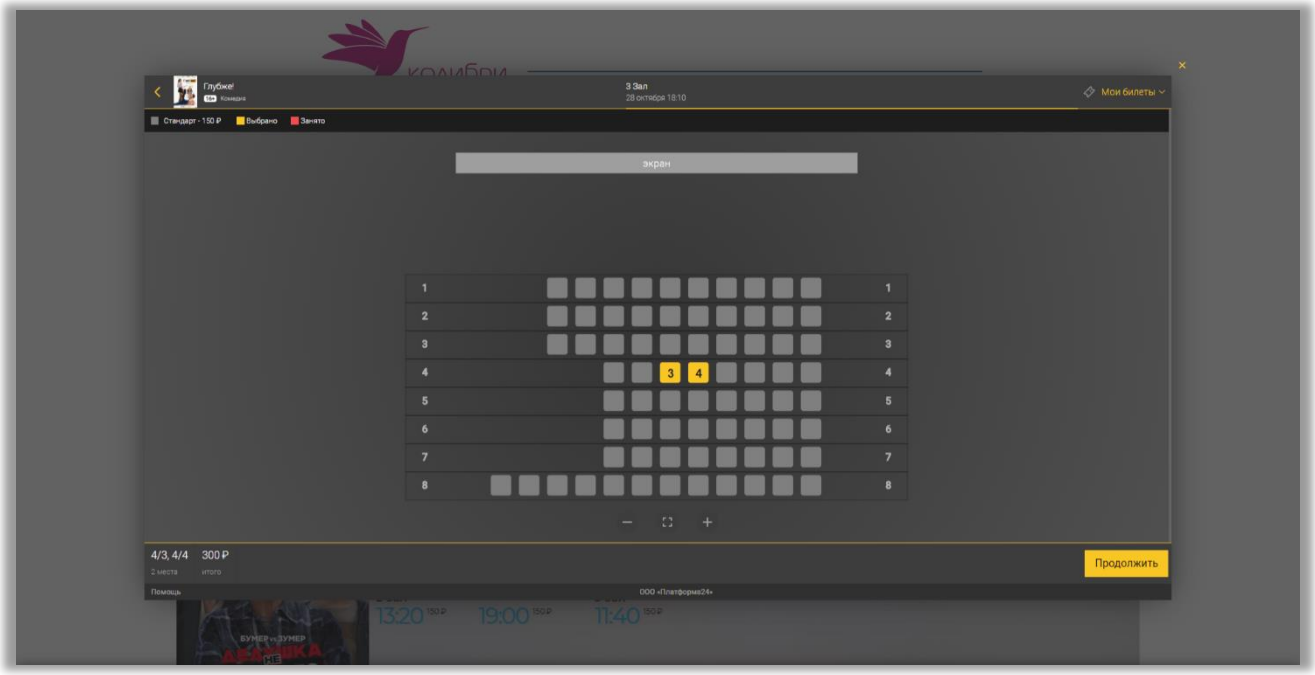

Рисунок 3 — Выбор места и ряда

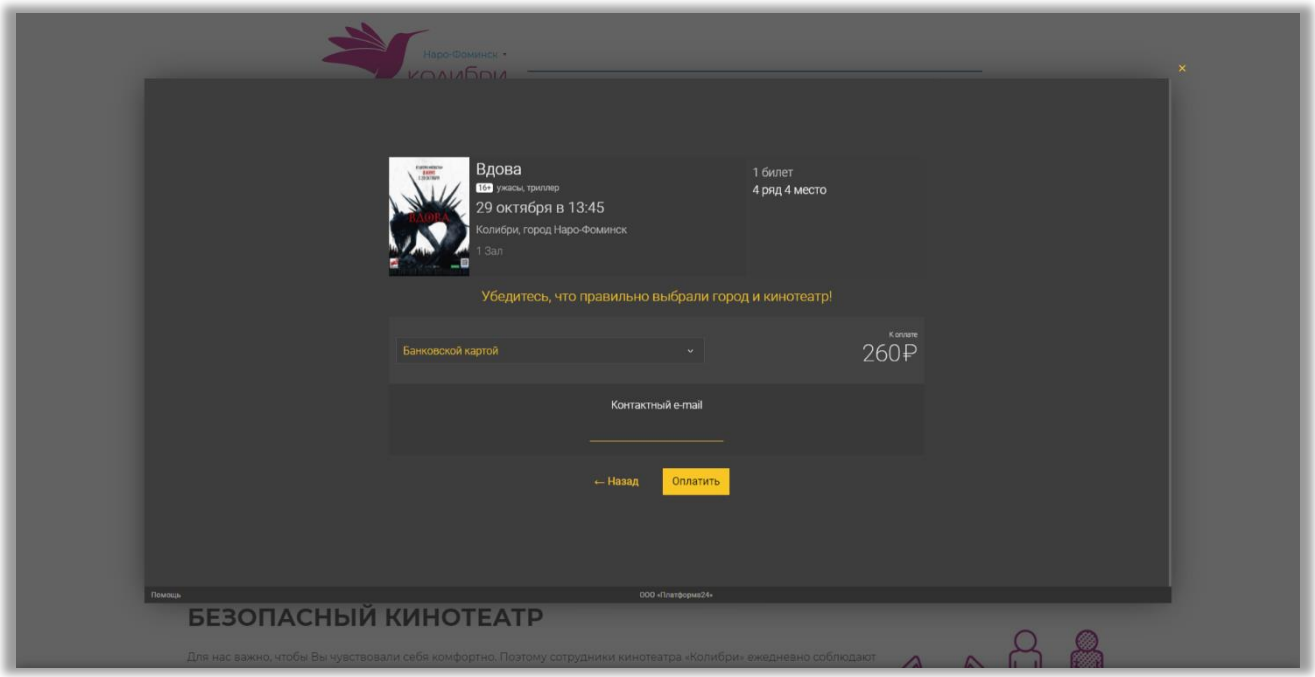

Рисунок 4 — Оплата билета

По-иному ситуация обстоит с крупными сетями кинотеатров, они могут себе позволить собственный сайт со всеми вытекающими преимуществами. Как правило, прослеживается фирменный стиль оформления, присутствует

собственная реализация виджетов для оформления билетов либо интеграция сторонних виджетов, например, от компании Rambler.

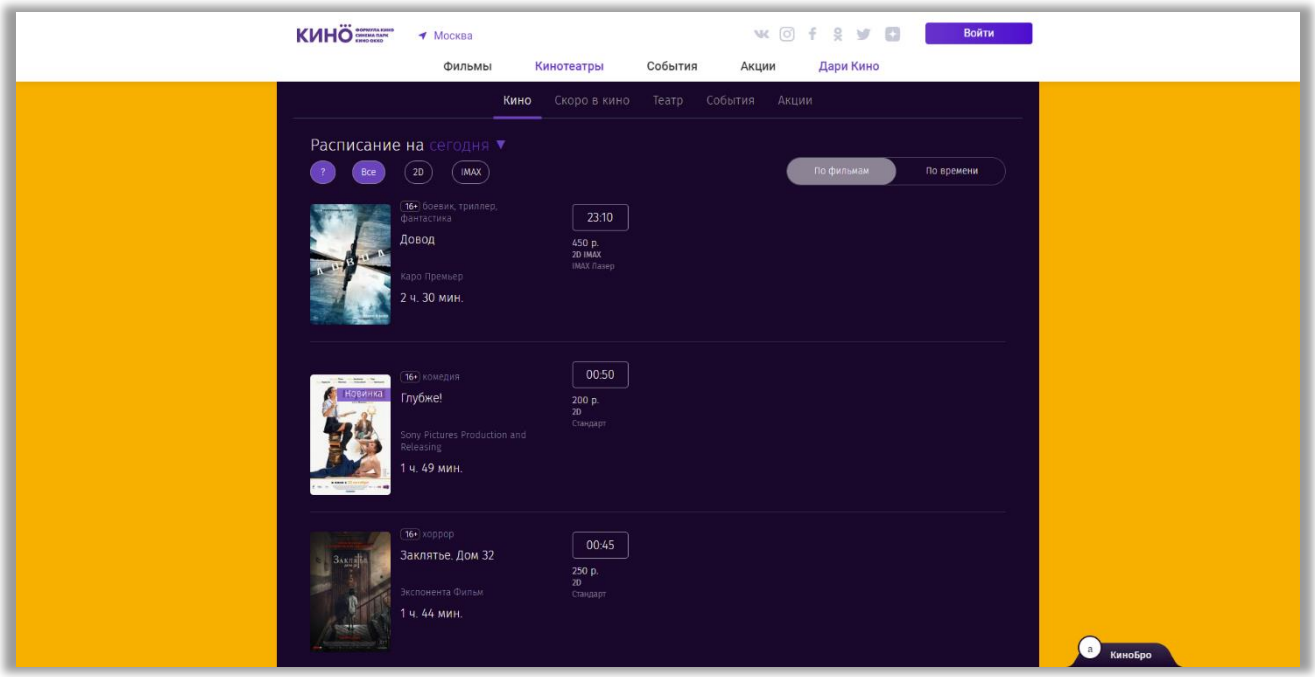

Рисунок 5 — Сеансы кинотеатра «Формула Кино»

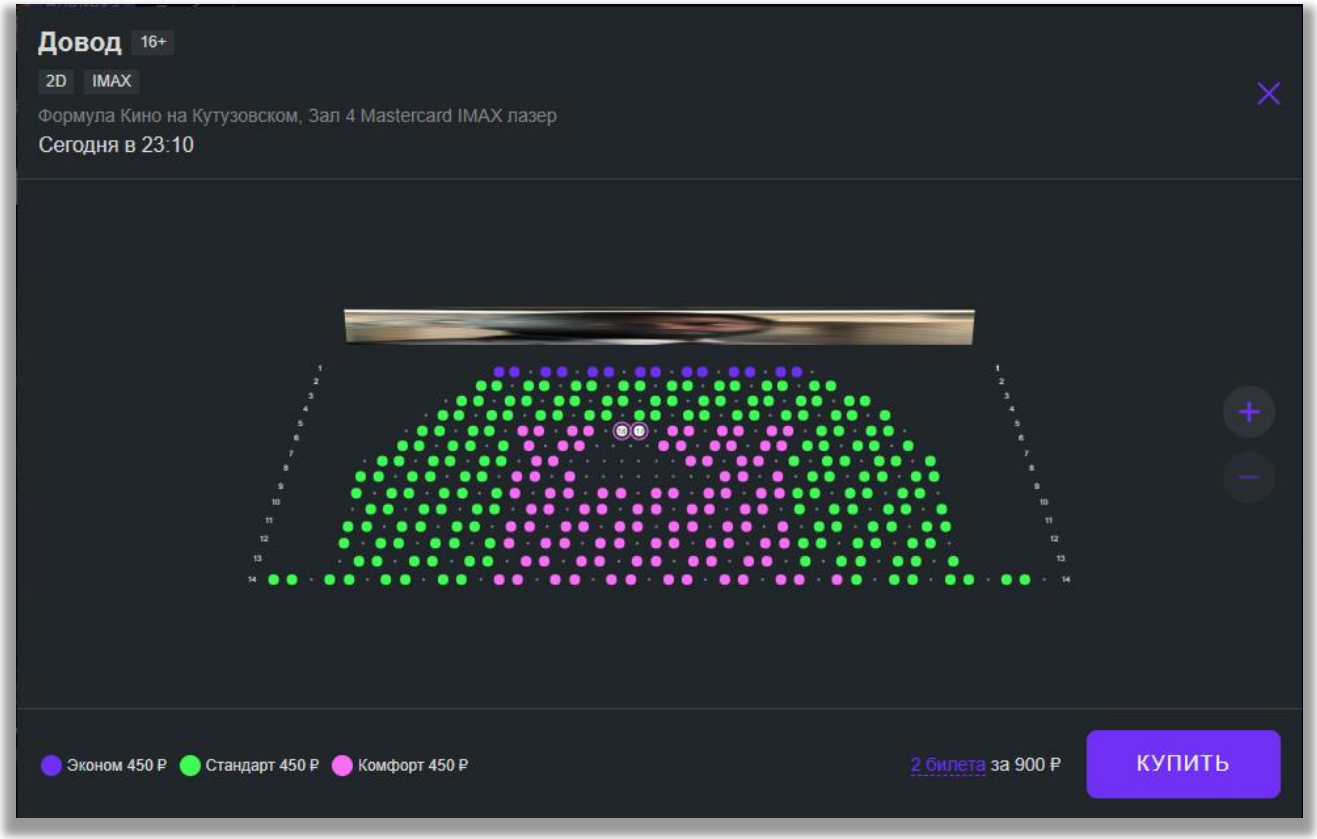

Рисунок 6 — Виджет оформления билета

Стоит также рассмотреть одну из крупнейших сетей кинотеатров в США — «CineMark»

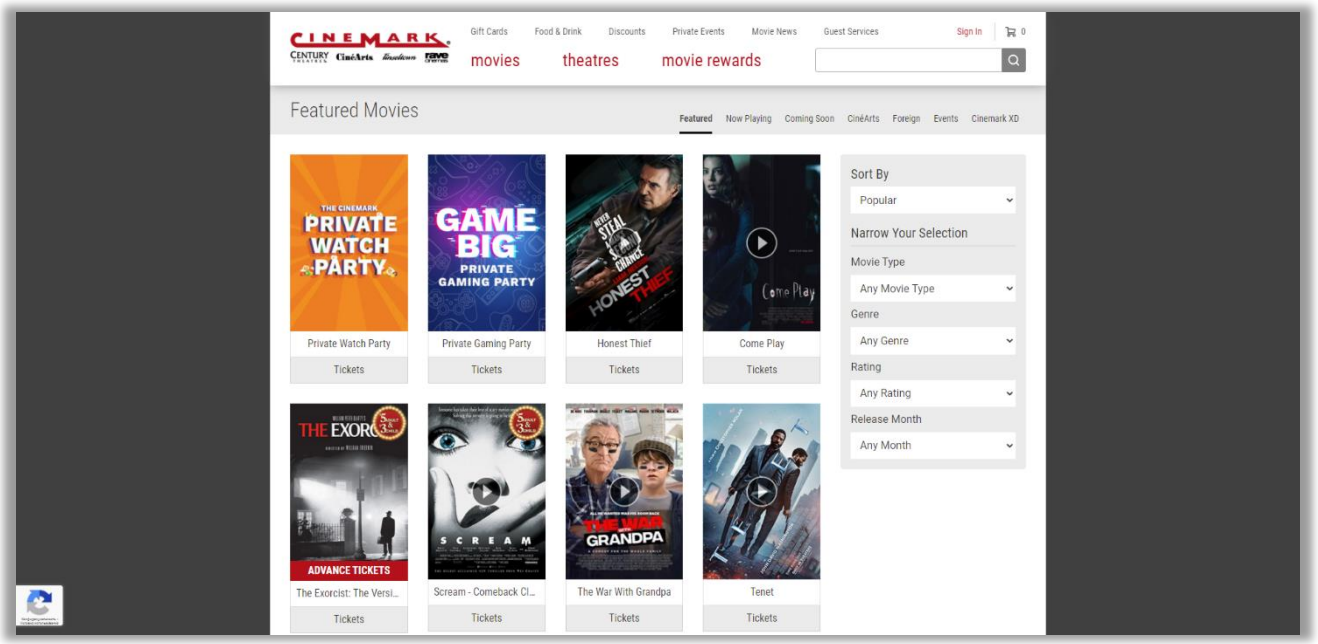

Рисунок 7 — Сетка фильмов в прокате

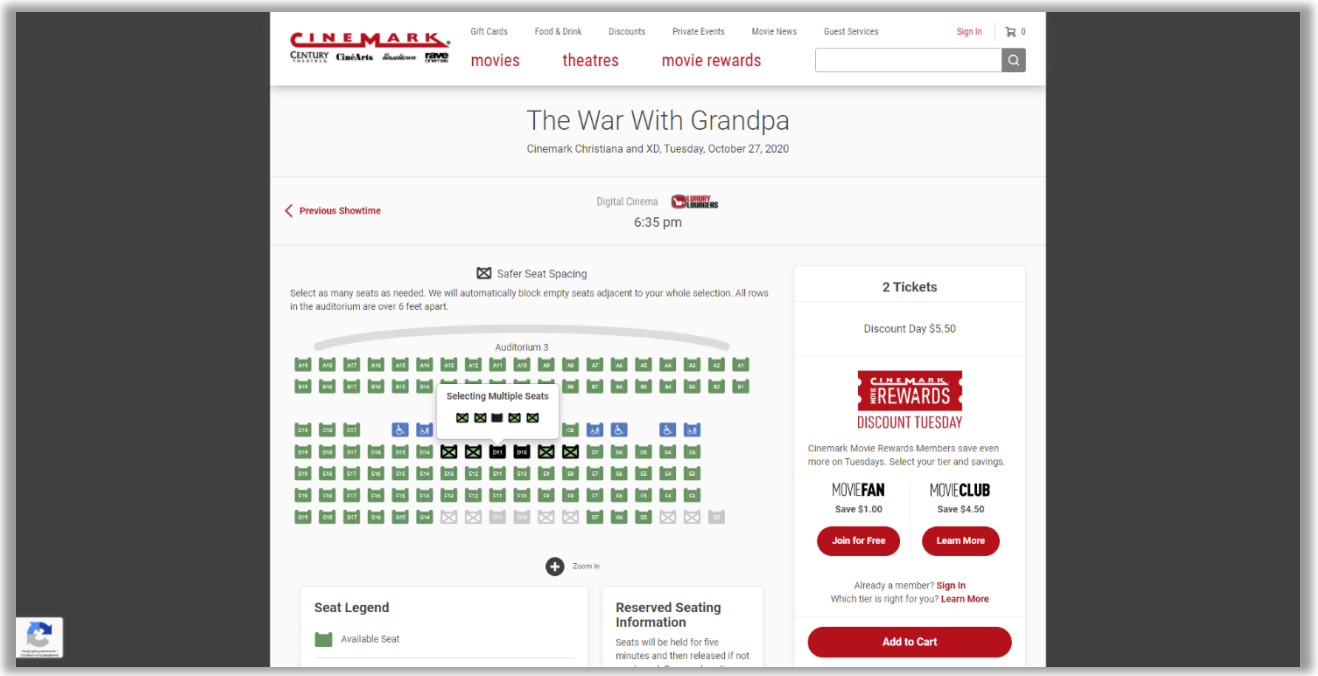

Рисунок 8 — Выбор места

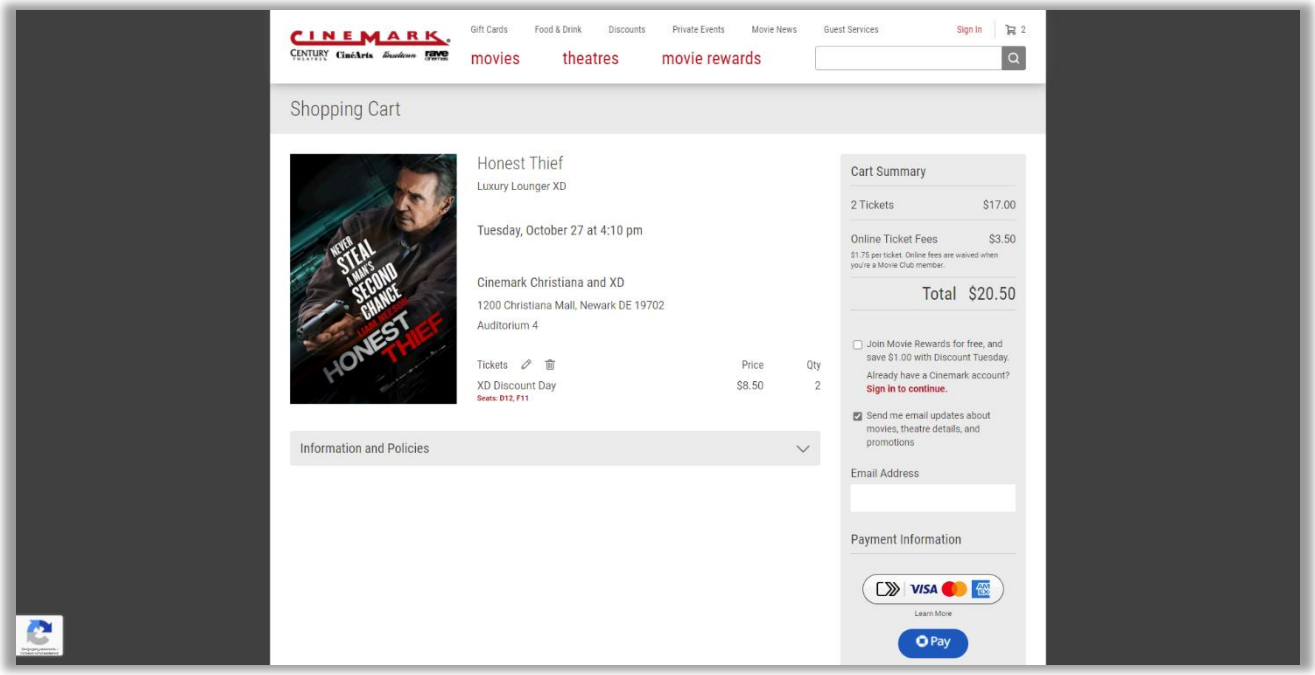

Рисунок 9 — Завершение оформления билета

Как видно, подход к реализации другой: оформление реализовано поэтапно без лишних всплывающих окон. Это несомненно является достоинством данного веб-приложения. Однако устаревший перегруженный дизайн оставляет желать лучшего.

При этом сайт другой сети «AMC» выглядит современно и в общем очень удобен.

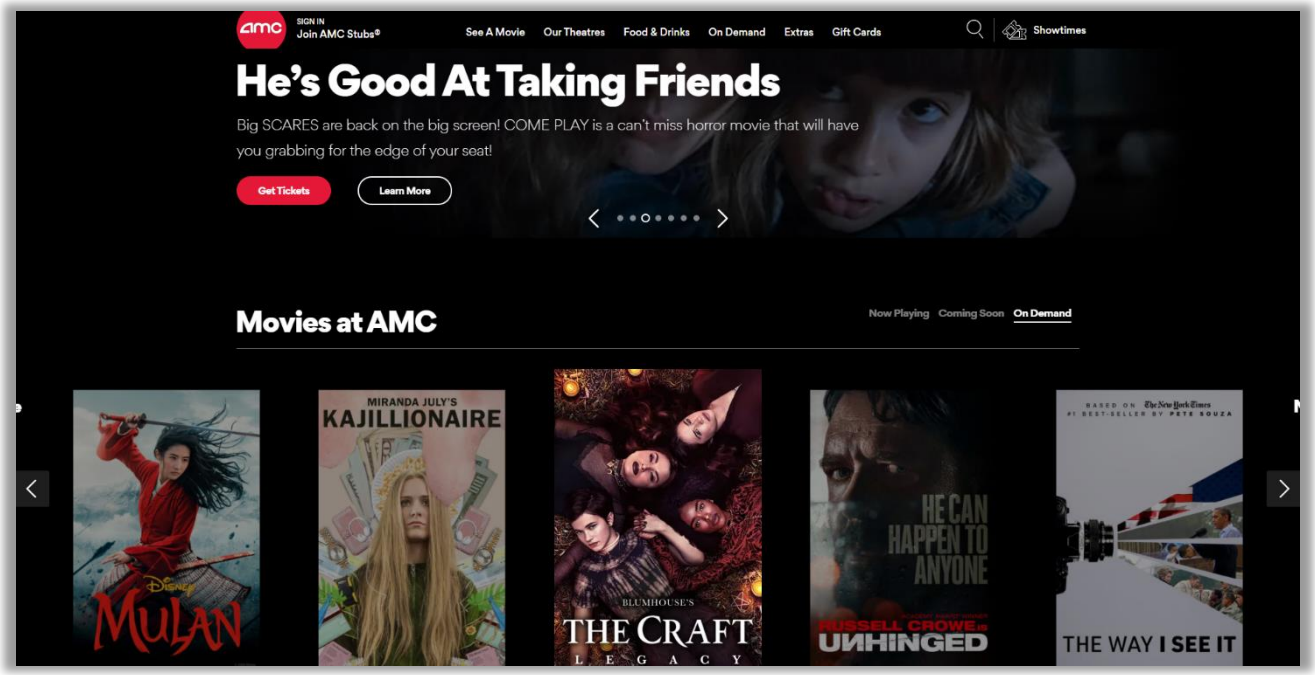

Рисунок 10 — Главная страница веб-приложения

Процесс оформления билета также пошаговый.

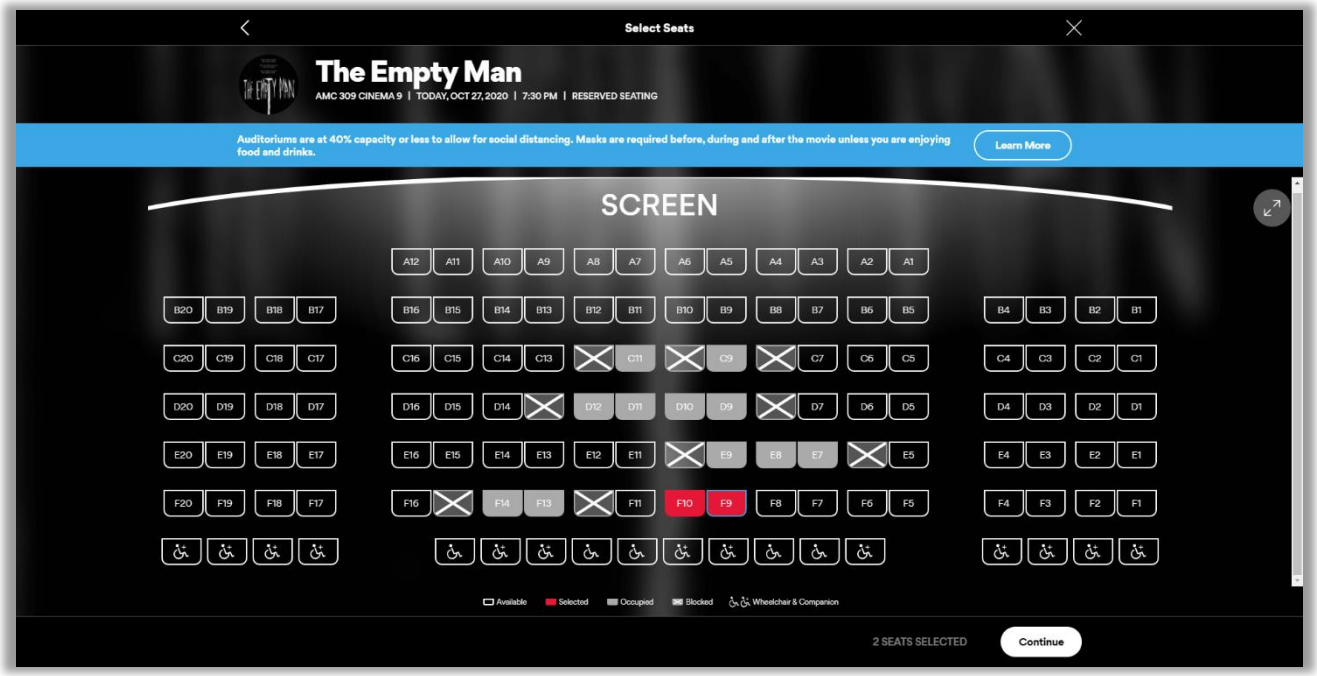

Рисунок 11 — Выбор места и ряда

2.4. Предлагаемые функции приложения для реализации

Прежде чем подходить к этапу реализации программных функций, необходимо понять, нужны ли они конечному потребителю, если речь идет о клиентских фишках, или работоспособен ли программный продукт без них, если мы говорим о серверных и других «скрытых» методах.

На основании разобранных выше аналогов разрабатываемой автоматизированной системы было выделено два основных подхода реализации функций веб-приложения: с помощью встраиваемых виджетов и использование перерисовки компонентов страницы. Использование второго подхода более целесообразно по следующим причинам: библиотека React позволяет не обновлять страницу веб-приложения целиком, а только необходимые элементы и с точки зрения предполагаемых действий конечного пользователя наличие бесшовного пошагового процесса работы с сайтом считаю приоритетным над реализацией всплывающих виджетов.

Для реализации MVP:

Со стороны клиента предлагаются следующие функции:

- регистрация с помощью почты и пароля;
- авторизация с помощью почты и пароля;
- просмотр купленных билетов;
- просмотр карусели фильмов;
- открытие карточки фильма с возможностью покупки билетов.

Со стороны серверной части предлагаются следующие функции:

- реализация регистрации и авторизации пользователя (создание соответствующих записей в БД и проверка наличия пользователя в БД);
- функция занесения информации о купленных билетах в БД;

- получение информации о фильмах и привязанных к ним сеансам из БД.

Основная задача — разработка веб-приложения. Среди современных инструментов для веб-разработки можно рассмотреть следующие IDE:

- Brackets;
- Atom;
- Microsoft Visual Studio Code;
- Eclipse;
- VIM.

Идеальным вариантом для решения поставленных мною задач является легкий редактор, обладающий функциями IDE — Microsoft Visual Studio Code по следующим причинам:

- удобная и простая кастомизация интерфейса;
- расширяемая библиотека плагинов;
- хорошие инструменты отладки и рефакторинга;
- поддержка множества языков;
- встроенные инструменты интеграции с GitHub, GIT.

2.5. Обоснование выбора языка разработки

В качестве основного языка разработки был выбран JavaScript. Это гибкий язык с огромным количеством различных библиотек и фреймворков. Он работает со всеми распространёнными веб-браузерами, позволяет писать код серверной части приложения, используя Node.js, и разрабатывать одностраничные и многостраничные приложения с помощью Express.js.

2.6. Обоснование выбора СУБД

Так как разработка веб-приложения будет производится с использованием стека технологий PERN, системой управления базами данных была выбрана

PostgreSQL — самая продвинутая свободная объектно-реляционная система управления базами данных, обладающая большим рядом преимуществ.

2.7. Библиотека React

React — это библиотека слоев представления. Никакой маршрутизации, никаких контроллеров, только пользовательский интерфейс. React — это декларативная, эффективная, основанная на компонентах и гибкая библиотека JS. Все компоненты, созданные с помощью React используют JS, а не языка вёрстки типа html, jade или hbs. Каждый элемент создан только с использованием JS.

Использование React ускоряет работу веб-приложения, так как мы можем разделять один большой компонент на малые компоненты, создавая так называемы одностраничные приложения. При выполнении сложных запросов клиента нет необходимости в перерисовки всего интерфейса. Достаточно обновить только некоторые компоненты.

Безусловно есть и минус - React не работает со старыми браузерами.

2.8.Фреймворк Express

Express — это самый популярный фреймворк Node.JS для обработки нескольких различных HTTP-запросов по определенному URL-адресу. Кроме того, он минимален, имеет открытый исходный код и гибкий, что помогает разработчику вкладывать меньше усилий и времени в разработку еще более совершенных веб-сайтов и приложений.

2.9. NodeJS

Node.js — это среда выполнения JavaScript. И JavaScript браузера, и Node.js работают на движке времени выполнения JavaScript V8. Этот движок берет код JavaScript и преобразует его в более быстрый машинный код. Машинный код — это низкоуровневый код, который компьютер может запускать без необходимости его предварительной интерпретации. Node.js использует

управляемую событиями неблокирующую модель ввода-вывода, что делает его легким и эффективным.

Экосистема пакетов Node.js, npm, является крупнейшей экосистемой библиотек с открытым исходным кодом в мире. В npm (менеджер пакетов Node) есть пакеты, которые можно использовать приложениях, чтобы ускорить и повысить эффективность разработки.

2.10. Node-postgres

Node-postgres — это набор модулей Node для взаимодействия с базой данных PostgreSQL. Он поддерживает обратные вызовы, промисы, async / await, пул соединений, подготовленные операторы, курсоры, результаты потоковой передачи, привязки C / C ++, синтаксический анализ типов и многое другое. Как и у самой PostgreSQL, у него много функций.

2.11. Язык дизайна Material-UI

Material-UI — это язык дизайна, разработанный Google в 2014 году и очень популярный для веб-приложений и мобильных приложений. Материальный дизайн вдохновлен физическим миром и его текстурами, в том числе тем, как они отражают свет и отбрасывают тени. Материальные поверхности меняют представление о бумаге и чернилах. С помощью компонентов из библиотеки Material-UI очень легко использовать элементы Material Design в разрабатываемом веб-приложении на React.

2.12. Выводы

Таким образом, исходя из требований к реализуемой системе, рассмотрения возможностей наиболее подходящих инструментов, вариантов разработки и последующего их сравнения было решено использовать следующие решения:

Основной язык программирования — JavaScript. Для разработки веб-приложения была выбрана интегрированная среда разработки Microsoft Visual

Studio Code. В качестве базы данных — PostgreSQL, СУБД — PgAdmin (psql). Для отладки функций серверной части — Postman.

3. ПРОЕКТНО - КОНСТРУКТОРСКАЯ ЧАСТЬ

3.1 Общие сведения

Данная курсовая работа представляет собой интеграцию базы данных и веб-приложения вместе предназначенные для решения вопроса автоматизации покупки билета в кинотеатр.

3.2 Архитектура веб-приложения

Разрабатываемая система представляет собой одностраничное веб-приложение, построенное на архитектуре «клиент-сервер», то есть серверная часть приложения принимает запросы от клиентской части, обрабатывает их и возвращает результат, написанное на языке JavaScript с использованием библиотек необходимых для реализации предполагаемых функций.

Преимущества данной архитектуры, следующие:

- мощный сервер дешевле множества мощных клиентских машин — если мы хотим, чтобы приложение не тормозило, нужна хорошая машина. Она у вас будет одна. Или несколько, если нагрузка большая, но явно меньше, чем количество клиентов.
- нет дублирования кода — основной код хранится на сервере, клиент отвечает только за отрисовку интерфейсов и проверку данных.
- персональные данные в безопасности — простой пользователь не видит лишнего. Он не знает секретные ключи, данные других пользователей.

К минусам архитектуры «клиент-сервер» относятся:

- если упал сервер или отвалилась база, то есть испортилось 1 звено — всё. Сотни, тысячи, да хоть миллионы клиентов если есть — никто не может работать.
- высокая стоимость оборудования — к железу для серверов совсем

иные требования к надежности.

- необходим человек, которые будет администрировать сервер и базу данных.

3.3 Концептуальная модель базы данных «Кинотеатр»

Концептуальная модель является входными данными для процесса проектирования логической и физической, поэтому необходимо на начальном этапе определить сущности базы данных, их атрибуты и отношения между сущностями.

Для будущей базы данных сущностями будут выступать:

- сеанс;
- фильм;
- зритель (пользователь);
- билет;
- зал.

Каждая сущность должна обладать уникальным идентификатором, который будет обеспечиваться генератором случайных идентификаторов. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Также сущность должна обладать следующими свойствами:

- к одному и тому же имени должна всегда применяться одна и та же интерпретация;
- иметь один или несколько атрибутов, которые либо принадлежат сущности, либо наследуются через связь;
- иметь один или несколько атрибутов, которые однозначно идентифицируют каждый экземпляр сущности.

Каждая сущность может обладать любым количеством связей с другими сущностями модели.

На основе описания предметной области смоделируем данные с помощью методологии Питера Чена на рисунке 12.

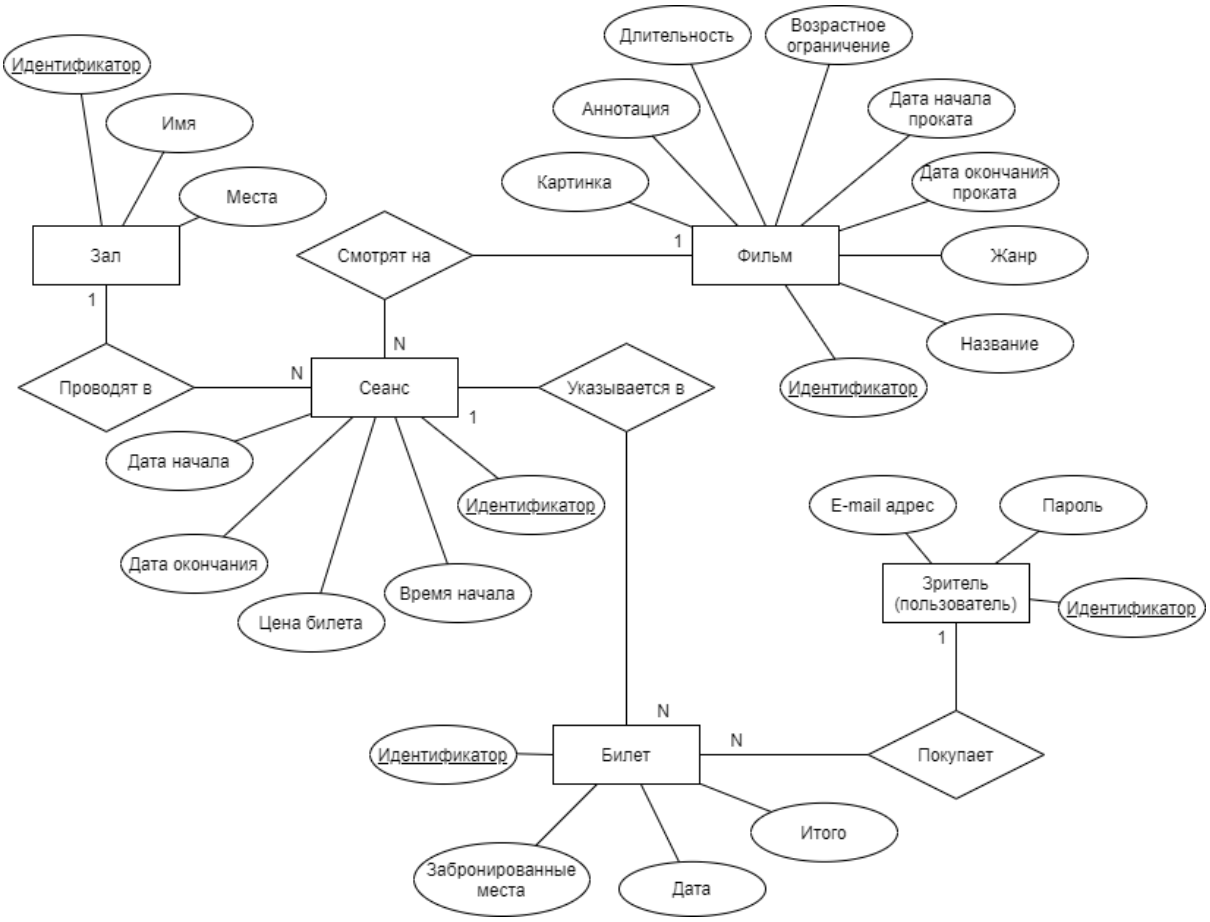

Рисунок 12 — Концептуальная схема проектируемой БД

3.4 Логическая модель базы данных «Кинотеатр»

При организации отношений на логическом уровне необходимо помнить о нормализации. Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение. Среди них:

- адекватность базы данных предметной области;

- легкость разработки и сопровождения базы данных;
- скорость выполнения операций обновления данных;
- скорость выполнения операций выборки данных.

Общее назначение процесса нормализации заключается в следующем:

- исключение некоторых типов избыточности;
- устранение некоторых аномалий обновления;
- разработка проекта базы данных, который является достаточно «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения;
- упрощение процедуры применения необходимых ограничений целостности.

Учитывая всевозможные особенности реализации интерфейса в веб-приложении предлагается логическая схема, указанная на рисунке 13.

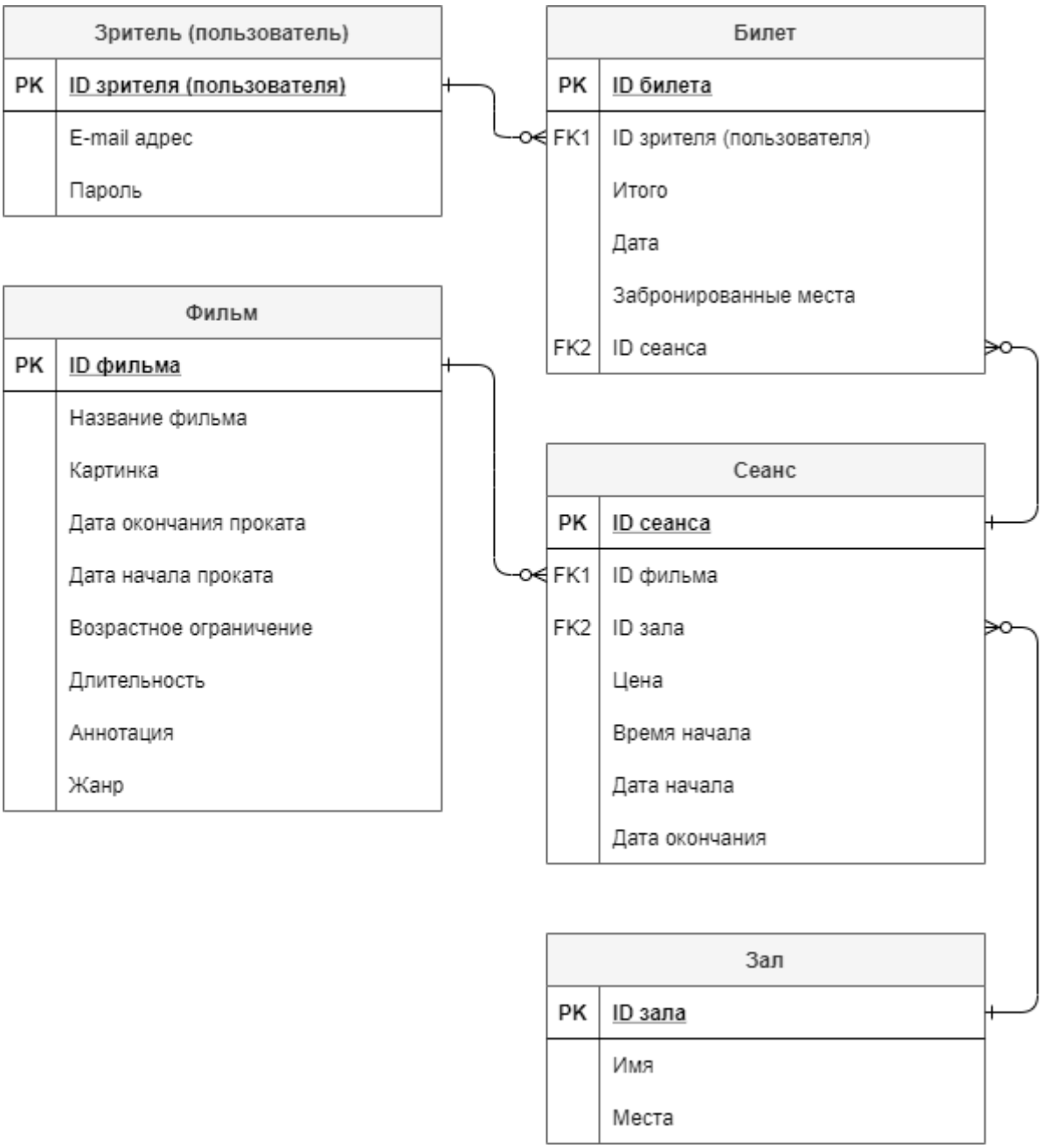

Рисунок 13 — Логическая схема проектируемой БД

3.5 Физическое проектирование базы данных «Кинотеатр»

Физическая модель данных зависит от конкретной СУБД, в ней содержится информация обо всех объектах БД. В физической модели важно описать всю информацию о конкретных физических объектах — таблицах, доменах, индексах, процедурах и т.д.

Проектируемая база данных должна удовлетворять следующим эксплуатационным требованиям:

- Оптимальная производительность;
- Расширяемость при реорганизации, гибкость при изменении, отказоустойчивость при удалении связей, атрибутов и сущностей;
- Загруженные в базу данных данные должны оставаться корректными;
- Загружаемые данные должны проверятся на корректность.

Ниже определены атрибуты сущностей, их формат и свойства:

Таблица 1 описывает сущность «Зритель (пользователь)», в ней хранится идентификатор, по которому мы будем определять, кто пользуется системой, email и пароль пользователя.

Таблица 1

Сущность «Зритель (пользователь)»

Атрибут	Ключ	Формат	Размер	Свойства
user_id	PRIMARY KEY	uuid	-	DEFAULT uuid_generate_v4()
user_email	-	VARCHAR	255	NOT NULL
user_password	-	VARCHAR	255	NOT NULL

Таблица 2 описывает модель «Билет», в ней хранится информация о купленных пользователем билеты.

Таблица 2

Сущность «Билет»

Атрибут	Ключ	Формат	Размер	Свойства

reservation_id	PRIMARY KEY	uuid	-	DEFAULT uuid_generate_v4()
user_id	FOREIGN KEY	uuid	-	NOT NULL REFERENCES users(user_id) ON DELETE CASCADE
showtime_id	FOREIGN KEY	uuid	-	NOT NULL REFERENCES showtimes(showtime_id) ON DELETE CASCADE
booked_seats	-	INTEGER[][]	-	NOT NULL
start_date	-	DATE	-	NOT NULL
total	-	INTEGER	-	NOT NULL

Таблица 3 описывает сущность «Сеанс», в ней хранится информация о сеансе к определенному фильму, а также схема зала, которая будет изменятся по мере оформления билетов пользователями системы.

Таблица 3

Сущность «Сеанс»

Атрибут	Ключ	Формат	Размер	Свойства
showtime_id	PRIMARY KEY	uuid	-	DEFAULT uuid_generate_v4()
ticket_price	-	INTEGER	-	NOT NULL
start_at	-	VARCHAR	255	NOT NULL

start_date	-	TIMESTAMP	-	NOT NULL
end_date	-	TIMESTAMP	-	NOT NULL
movie_id	FOREIGN KEY	uuid	-	NOT NULL REFERENCES movies(movie_id) ON DELETE CASCADE
hallscheme_id	FOREIGN KEY	uuid	-	NOT NULL REFERENCES hallschemes(hallscheme_id) ON DELETE CASCADE

Таблица 4 описывает сущность «Фильм», она необходима для отрисовки афиши, а также для предоставления пользователю подробной информации о фильме, билет на который он пожелает купить.

Таблица 4

Сущность «Фильм»

Атрибут	Ключ	Формат	Размер	Свойства
movie_id	PRIMARY KEY	uuid	-	DEFAULT uuid_generate_v4()
movie_title	-	VARCHAR	255	NOT NULL
movie_description		VARCHAR	255	NOT NULL
image_url	-	VARCHAR	255	NOT NULL

back_image_url	-	VARCHAR	255	NOT NULL
movie_director	-	VARCHAR	100	NOT NULL
movie_duration	-	VARCHAR	255	NOT NULL
movie_genre	-	TEXT[]	-	NOT NUL
release_date	-	DATE	-	NOT NULL
end_date	-	DATE	-	NOT NULL

Таблица 5 описывает сушность «Зал», она хранит в себе имя зала, для отображения пользователю, в каком именно зале будет производится показ фильма, а также двумерный массив мест, заполненный значениями от 1 до 4. Каждое значение отвечает за определенное состояние места:

- 1 – место занято
- 2 – место выбрано пользователем
- 3 – место свободно
- 4 – место недоступно (можно использовать в случае ремонта мест в зале или, например, для соблюдений социальной дистанции)

Таблица 5

Сущность «Зал»

Атрибут	Ключ	Формат	Размер	Свойства

hallscheme_id	PRIMARY KEY	uuid	-	DEFAULT uuid_generate_v4()
hall_name	-	VARCHAR	100	NOT NULL
seats	-	INTEGER[][]	-	NOT NULL

Именно информацию этих сущностей обрабатывает серверная часть разрабатываемого веб-приложения.

3.6 Серверная часть веб-приложения

Разработка серверной части выполнялась с использованием NodeJS, ExpressJS, и node-postgres.

Изначально в файле app.js создается родительский машрут и потр на котором будет прослушиваться сервер.

Далее расписываются все маршруты веб-приложения.

```
const express = require("express");
const app = express();
const cors = require("cors");

//middleware
app.use(cors());
app.use(express.json()); //req.body

//Routes//
app.use("/auth", require("./routes/jwtAuth"));

app.use("/dashboard", require("./routes/dashboard"));

app.use("/movies", require("./routes/movies"));

app.use("/booking", require("./routes/booking"));

app.use("/reservations", require("./routes/reservations"));

const port = process.env.PORT || 5000;

app.listen(port, () => {
    console.log(`Server has started on port ${port}`);
```

```
});
```

В папке server\routes описываются дочерние маршруты веб-приложения. Все что связано со входом пользователя в систему и регистрацией описано в файле jwtAuth.js. Авторизация пользователя происходит с помощью JSONWebToken, который после создания передается в local.Storage.

```
const router = require("express").Router();
const pool = require("../db");
const bcrypt = require("bcrypt");
const jwtGenerator = require("../utils/jwtGenerator");
const validinfo = require("../middleware/validinfo");
const authorization = require("../middleware/authorization");

// Регистрация
router.post("/register", validinfo, async (req, res) => {
    try {
        const { email, password } = req.body;
        const user = await pool.query("SELECT * FROM users WHERE user_email = $1",
[
            email,
        ]);

        if (user.rows.length !== 0) {
            return res.status(401).json("Пользователь уже зарегистрирован!");
        }
        // Шифрование пароля
        const saltRounds = 10;
        const salt = await bcrypt.genSalt(saltRounds);
        const bcryptPassword = await bcrypt.hash(password, salt);

        // Новый пользователь
        const newUser = await pool.query(
            "INSERT INTO users (user_email, user_password) VALUES ($1, $2) RETURNIN
G *",
            [email, bcryptPassword]
        );
        // Генерирование токена
        const token = jwtGenerator(newUser.rows[0].user_id);

        res.json({ token });
    } catch (err) {
        console.error(err.message);
        res.starus(500).send("Server Error");
    }
```

```
});

// Логин
router.post("/login", validinfo, async (req, res) => {
    try {
        const { email, password } = req.body;

        const user = await pool.query("SELECT * FROM users WHERE user_email = $1",
[
            email,
        ]);
        if (user.rows.length === 0) {
            return res.status(401).json("Неверный email или пароль!");
        }

        const validPassword = await bcrypt.compare(
            password,
            user.rows[0].user_password
        );
        if (!validPassword) {
            return res.status(401).json("Неверный email или пароль!");
        }
        const token = jwtGenerator(user.rows[0].user_id);

        res.json({ token });
    } catch (err) {
        console.error(err.message);
        res.starus(500).send("Server Error");
    }
});

// Проверка входа
router.get("/is-verify", authorization, async (req, res) => {
    try {
        res.json(true);
    } catch (err) {
        console.error(err.message);
        res.starus(500).send("Server Error");
    }
});

module.exports = router;
```

Методы регистрации и входа практически идентичны. Метод регистрации сначала проверяет, зарегистрирован ли пользователь в системе, и при успешной проверке создает нового пользователя, передавая в БД email и зашифрованный с

помощью библиотеки bcrypt. Метод входа сначала проверяет введенные данные, и при успешной проверке возвращает пользователю токен для авторизации. Для проверки введенных значения используется middleware функция validinfo.js. Она проверяет данные, возвращает статус 401 при ошибке.

```
module.exports = function (req, res, next) {
    const { email, password } = req.body;

    function validEmail(userEmail) {
        return /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(userEmail);
    }

    if (req.path === "/register") {
        if (![email, password].every(Boolean)) {
            return res.status(401).json("Заполните все поля!");
        } else if (!validEmail(email)) {
            return res.status(401).json("Неправильный email");
        }
    } else if (req.path === "/login") {
        if (![email, password].every(Boolean)) {
            return res.status(401).json("Заполните все поля!");
        } else if (!validEmail(email)) {
            return res.status(401).json("Неправильный email");
        }
    }

    next();
};
```

В методе проверки входе используется middleware стрелочная функция authorization.js проверки наличия JWT токена в заголовке. Стоит отметить что все методы в маршрутах используют эту функцию, чтобы неавторизированные пользователи не могли получить данные из базы данных.

```
const jwt = require("jsonwebtoken");
require("dotenv").config();

module.exports = async (req, res, next) => {
    try {
        const jwtToken = req.header("token");

        if (!jwtToken) {
            return res.status(403).send("Not Aithorize");
```

```
        }

        const payload = jwt.verify(jwtToken, process.env.jwtSecret);

        req.user = payload.user;

        next();
    } catch (err) {
        console.error(err.message);
        res.status(403).send("Not Aithorize");
    }
};
```

Функция генерации токена описана в файле server\utils\jwtGenerator.js. Она принимает идентификатор пользователя и с помощью прописанного ключа шифрования в файле server\.env возвращает токен.

```
const jwt = require("jsonwebtoken");
require("dotenv").config();

function jwtGenerator(user_id) {
    const payload = {
        user: {
            id: user_id,
        },
    };

    return jwt.sign(payload, process.env.jwtSecret, { expiresIn: "1hr" });
}

module.exports = jwtGenerator;
```

Маршрут главной страницы описан в файле routes\dashboard.js. Здесь вызывается метод получения информации о пользователе по параметру полученному из клиента.

```
const router = require("express").Router();
const authorization = require("../middleware/authorization");
const pool = require("../db");

// Получение идентификатора пользователя
router.get("/", authorization, async (req, res) => {
    try {
```

```
        const user = await pool.query(
            "SELECT user_id, user_email FROM users WHERE user_id = $1",
            [req.user.id]
        );
        res.json(user.rows[0]);
    } catch (err) {
        console.error(err.message);
        res.status(500).send("Server Error");
    }
});

module.exports = router;
```

В файле routes\movies.js описаны методы получения всех фильмов и фильма по его идентификатору из БД.

```
const router = require("express").Router();
const authorization = require("../middleware/authorization");
const pool = require("../db");

// Полечение фильмов
router.get("/", authorization, async (req, res) => {
    try {
        const movies = await pool.query("SELECT * FROM movies");
        res.json(movies.rows);
    } catch (err) {
        console.error(err.message);
        res.status(500).send("Server Error");
    }
});

// Получение фильма по идентификатору
router.get("/:id", authorization, async (req, res) => {
    const { id } = req.params;
    try {
        const movie = await pool.query("SELECT * FROM movies WHERE movie_id = $1",
[
            id,
        ]);
        res.json(movie.rows);
    } catch (err) {
        console.error(err.message);
        res.status(500).send("Server Error");
    }
});

module.exports = router;
```

В файле routes\booking.js описаны методы получения объединенной информации по идентификатору фильма, отдельно получение сеансов и схем залов.

```
const router = require("express").Router();
const authorization = require("../middleware/authorization");
const pool = require("../db");

// Получение схем и сенасов для фильма
router.get("/:id", authorization, async (req, res) => {
    const { id } = req.params;
    try {
        const response = await pool.query(
            "SELECT * FROM hallschemes AS h JOIN showtimes AS s ON s.hallscheme_id=
h.hallscheme_id JOIN movies AS m ON s.movie_id=m.movie_id WHERE m.movie_id=$1",
            [id]
        );
        res.json(response.rows);
    } catch (err) {
        console.error(err.message);
        res.status(500).send("Server Error");
    }
});

// Получение сенасов для фильма
router.get("/showtimes/:id", authorization, async (req, res) => {
    const { id } = req.params;
    try {
        const response = await pool.query(
            "SELECT * FROM showtimes WHERE movie_id=$1 ",
            [id]
        );
        res.json(response.rows);
    } catch (err) {
        console.error(err.message);
        res.status(500).send("Server Error");
    }
});

// Обнволение схемы зала
router.put("/hallschemes/:id", authorization, async (req, res) => {
    try {
        const hallscheme_id = req.params.id;
        const seats = req.body.newSeats;
        const updateHallscheme = await pool.query(
```

```
            "UPDATE hallschemes SET seats = $1 WHERE hallscheme_id = $2 ",
            [seats, hallscheme_id]
        );

        res.json(updateHallscheme.rows[0]);
    } catch (err) {
        console.error(err.message);
    }
});
module.exports = router;
```

И, наконец, в файле routes\reservations.js описано два метода: присваивание билетов пользователю по его идентификатору и получение всех билетов пользователя.

```
const router = require("express").Router();
const authorization = require("../middleware/authorization");
const pool = require("../db");

// Присваивание билетов пользователю
router.post("/:id", authorization, async (req, res) => {
    try {
        const user_id = req.params.id;
        const showtime_id = req.body.shId;
        const booked_seats = req.body.bookedSeats;
        const start_date = req.body.selectedDate;
        const total = req.body.total;

        const newReservation = await pool.query(
            "INSERT INTO reservations (user_id, showtime_id, booked_seats, start_da
te, total) VALUES ($1, $2,$3,$4,$5) RETURNING *",
            [user_id, showtime_id, booked_seats, start_date, total]
        );

        res.json(newReservation.rows[0]);
    } catch (err) {
        console.error(err.message);
        res.starus(500).send("Server Error");
    }
});

// Получение всех билетов по идентификатору пользователя
router.get("/user/:id", authorization, async (req, res) => {
    const { id } = req.params;
    try {
        const response = await pool.query(
```

```
            "SELECT *, reservations.start_date AS RSD FROM reservations JOIN showti
mes  ON reservations.showtime_id=showtimes.showtime_id JOIN movies ON movies.movie_
id=showtimes.movie_id WHERE user_id=$1",
            [id]
        );
        res.json(response.rows);
    } catch (err) {
        console.error(err.message);
        res.status(500).send("Server Error");
    }
});

module.exports = router;
```

Обработка сервером запросов к базе данных происходим через пул соединения.

```
const pool = require("../db");
```

В файле server\db.js указаны параметры для обращения к базе данных.

```
const Pool = require("pg").Pool;
const pool = new Pool({
    user: "postgres",
    password: " password",
    host: "localhost",
    port: 5432,
    database: "cinemaapp",
});

module.exports = pool;
```

3.7 Клиентская часть веб-приложения

Клиентская часть приложения была инициализирована с помощью библиотеки React. Все компоненты были вынесены в папку client\components. Все экспорты описаны в файле client\components\index.js.

```
// Components
export { default as Dashboard } from './Dashboard';
export { default as SignIn } from './SignIn';
```

```
export { default as SignUp } from './SignUp';
...
```

```
// MuiCore Items
export { default as Slide } from '@material-ui/core/Slide';
export { default as Menu } from '@material-ui/core/Menu';
export { default as MenuItem } from '@material-ui/core/MenuItem';
...
```

```
// Icons
export { default as IconButton } from '@material-ui/core/IconButton';
export { default as AccountCircle } from '@material-ui/icons/AccountCircle';
...
```

Исполняемый файл клиентской части App.js использует маршрутизацию для отрисовки соответствующих маршруту компонентов.

```
  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <BrowserRouter>
        <div className="container" />
        <Switch>
          <Route exact path="/">
            <Redirect to="/dashboard" />
          </Route>

          <Route
            exact
            path="/login"
            render={(props) =>
              !isAuthenticated ? (
                <SignIn {...props} setAuth={setAuth} />
              ) : (
                <Redirect to="/dashboard" />
              )
            }
          />
          <Route
            exact
            path="/register"
            render={(props) =>
              !isAuthenticated ? (
                <SignUp {...props} setAuth={setAuth} />
              ) : (
```

```
              <Redirect to="/login" />
            )
          }
        />
        <Route
          exact
          path="/dashboard"
          render={(props) =>
            isAuthenticated ? (
              <Dashboard {...props} setAuth={setAuth} />
            ) : (
              <Redirect to="/login" />
            )
          }
        />
        <Route path="/movies/:id" component={Movie} />
        <Route path="/booking/:id" component={BookingPage} />
        <Route path="/dashboard/myTickets" component={MyTickets} />
        <Route path="*" component={() => 'ERROR 404 NOT FOUND'} />
      </Switch>
    </BrowserRouter>
  </ThemeProvider>
 );
}
```

Тут же используются хуки состояния и хуи эффектов для проверки верифицирован ли пользователь и установки аутентификации.

```
const [isAuthenticated, setIsAuthenticated] = useState(false);
  const setAuth = (boolean) => {
    setIsAuthenticated(boolean);
  };

  async function isAuth() {
    try {
      const response = await fetch('http://localhost:5000/auth/is-verify', {
        method: 'GET',
        headers: { token: localStorage.token },
      });

      const parseRes = await response.json();

      if (
        parseRes === true ? setIsAuthenticated(true) : setIsAuthenticated(false)
      );
    } catch (err) {
      console.log(err.message);
    }
```

```
  }

  useEffect(() => {
    isAuth();
  });
```

3.8 Взаимодействие частей веб-приложения

База данных защищена от атак с помощью SQL-инъекций, так как все запросы являются параметризированными.

Данные между клиентом и сервером передаются в формате JSON. Рассмотрим пример обращения клиентской части к серверу на примере отображения карусели фильмов.

Для хранения массива данных используется хук состояния

```
const [data, setData] = useState('');
```

Сам метод представляет собой HTTP-запрос GET с использованием конструкции try..catch. В заголовке передается токен пользователя для доступа данных. В ответ мы принимает данные, которые возвращает нам сервер из базы данных. Далее мы переносим данные в хук состояния. В случае возникновения ошибки, выводим сообщение в консоль.

```
const getData = async () => {
  try {
    const res = await fetch('http://localhost:5000/movies', {
      method: 'GET',
      headers: { token: localStorage.token },
    });

    const parseData = await res.json();
    setData(parseData);
  } catch (err) {
    console.error(err.message);
  }
};
```

Вызов запроса:

```
useEffect(() => {
```

```
    getData();
  }, []);
```

В результате вызова запроса, клиент получает для отрисовки следующий ответ от сервера, который представляет из себя массив из 6 элементов.

```
1.  (6) [{…}, {…}, {…}, {…}, {…}, {…}]
      1.  0: {movie_id: "b19db187-03a7-4f81-8521-
          52daaacdefaf", movie_title: "Джентльмены", movie_description: "Один ушлый американец ещё
          со студенческих лет прит… чернокожих спортсменов и даже русского
          олигарха.", image_url: "https://image.tmdb.org/t/p/original/hClLP88yMIuhslwSVxtYrvWmxfp.jp
          g", back_image_url: "https://image.tmdb.org/t/p/original/9Qfawg9WT3cSbBXQgDRuWbYS9lj.jpg",
           …}
      2.  1: {movie_id: "7998ee39-9038-4e25-bca3-
          a436a4d44001", movie_title: "Паразиты", movie_description: "Обычное корейское семейство
          Кимов жизнь не балует.…риходит необычный план по трудоустройству
          сестры.", image_url: "https://image.tmdb.org/t/p/original/TU9NIjwzjoKPwQHoHshkFcQUCG.jpg",
           back_image_url: "https://image.tmdb.org/t/p/original/hAtRKAi24kw4wyGB2IyvmFdVGHC.jpg", …}
      3.  2: {movie_id: "197dc38d-9f33-4391-875c-8569086094ee", movie_title: "Достать
          ножи", movie_description: "На следующее утро после празднования 85-летия изве…тра
          покойного, которая физически не выносит
          ложь.", image_url: "https://image.tmdb.org/t/p/original/4HWAQu28e2yaWrtupFPGFkdNU7V.jpg",
          back_image_url: "https://image.tmdb.org/t/p/original/g1eZwt1uXQupUY1WpU8iQXkcaK6.jpg", …}
      4.  3: {movie_id: "3c88a0f8-55b2-487e-b78d-2ed2c1d1ca5b", movie_title: "Ford против
          Ferrari", movie_description: "В начале 1960-х Генри Форд II принимает решение ул…тку
          впоследствии знаменитого спорткара Ford
          GT40.", image_url: "https://image.tmdb.org/t/p/original/5ZQjqTi72MifO4G79nJ5niv0ioN.jpg",
          back_image_url: "https://image.tmdb.org/t/p/original/qfY58nletpMcVNOrCQg95T3kUwh.jpg", …}
      5.  4: {movie_id: "a6069137-2e98-4c68-9f64-4e17efbf3bb9", movie_title: "Кролик
          Джоджо", movie_description: "Гитлеровская Германия. У немного стеснительного 10…ясняет,
          что мама прячет в доме еврейскую
          девушку.", image_url: "https://image.tmdb.org/t/p/original/agoBZfL1q5G79SD0npArSlJn8BH.jpg
          ", back_image_url: "https://image.tmdb.org/t/p/original/8CX2kCBV9x5ASTQ3ipLToz6TtCa.jpg",
          …}
      6.  5: {movie_id: "f8facf7c-ec1f-4c59-b818-
          f547d3830e9c", movie_title: "Джокер", movie_description: "Готэм, начало 1980-х годов.
          Комик Артур Флек живет… него не добрую улыбку, а ухмылку злодея
          Джокера.", image_url: "https://image.tmdb.org/t/p/original/n6bUvigpRFqSwmPp1m2YADdbRBc.jpg
          ", back_image_url: "https://image.tmdb.org/t/p/original/rlay2M5QYvi6igbGcFjq8jxeusY.jpg",
          …}
      7.  length: 6
```

Каждый элемент массива — это объект, который содержит в себе информацию об одном фильме. Ключ — это есть ничто иное как название атрибута сущности (конкретно «movies») в базе данных, а в значении хранится соответствующий атрибуту набор данных в формате VARCHAR, INTEGER или каком-либо другом, описанном в физической модели базы данных выше.

```
1.  0:
        1.  back_image_url: "https://image.tmdb.org/t/p/original/9Qfawg9WT3cSbBXQgDRuWbYS9lj.j
            pg"
        2.  end_date: "2020-12-05T16:10:25.000Z"
        3.  image_url: "https://image.tmdb.org/t/p/original/hClLP88yMIuhslwSVxtYrvWmxfp.jpg"
        4.  movie_description: "Один ушлый американец ещё со студенческих лет приторговывал
            наркотиками, а теперь придумал схему нелегального обогащения с использованием
```

```
    поместий обедневшей английской аристократии и очень неплохо на этом разбогател.
    Другой пронырливый журналист приходит к Рэю, правой руке американца, и предлагает
    тому купить киносценарий, в котором подробно описаны преступления его босса при
    участии других представителей лондонского криминального мира - партнёра-еврея,
    китайской диаспоры, чернокожих спортсменов и даже русского олигарха."
5.  movie_director: "Гай Ричи"
6.  movie_duration: "113"
7.  movie_genre: (3) ["Криминал", "Комедия", "Боевик"]
8.  movie_id: "b19db187-03a7-4f81-8521-52daaacdefaf"
9.  movie_title: "Джентльмены"
10. release_date: "2020-12-01T16:10:25.000Z"
11. __proto__: Object
```

Полученные массив данных с помощью функции map() разбивается на отдельные элементы, тем самым обрисовывая все 6 фильмов.

```
{data.length !== 0 &&
          data[0].movie_id !== null &&
          data.map((movies) => (
            <Link
              href={`movies/${movies.movie_id}`}
              style={{ textDecoration: 'none' }}
            >
              <Card className={classes.card}>
                <CardActionArea className={classes.action}>
                  <CardMedia
                    className={classes.cardMedia}
                    image={movies.image_url}
                    title={movies.movie_title}
                  >
                    <CardContent className={classes.cardContent}>
                      <Typography gutterBottom variant="h4" component="h4">
                        {movies.movie_title}
                      </Typography>
                      <Typography>
                        {textTruncate(movies.movie_description, 200)}
                      </Typography>
                    </CardContent>
                  </CardMedia>
                </CardActionArea>
              </Card>
            </Link>
          ))}
```

Аналогичным образом описаны остальные запросы клиента к серверу.

3.9 UML диаграмма размещения

Для общего представления функционального назначения системы используем диаграмму использования (use case diagram).

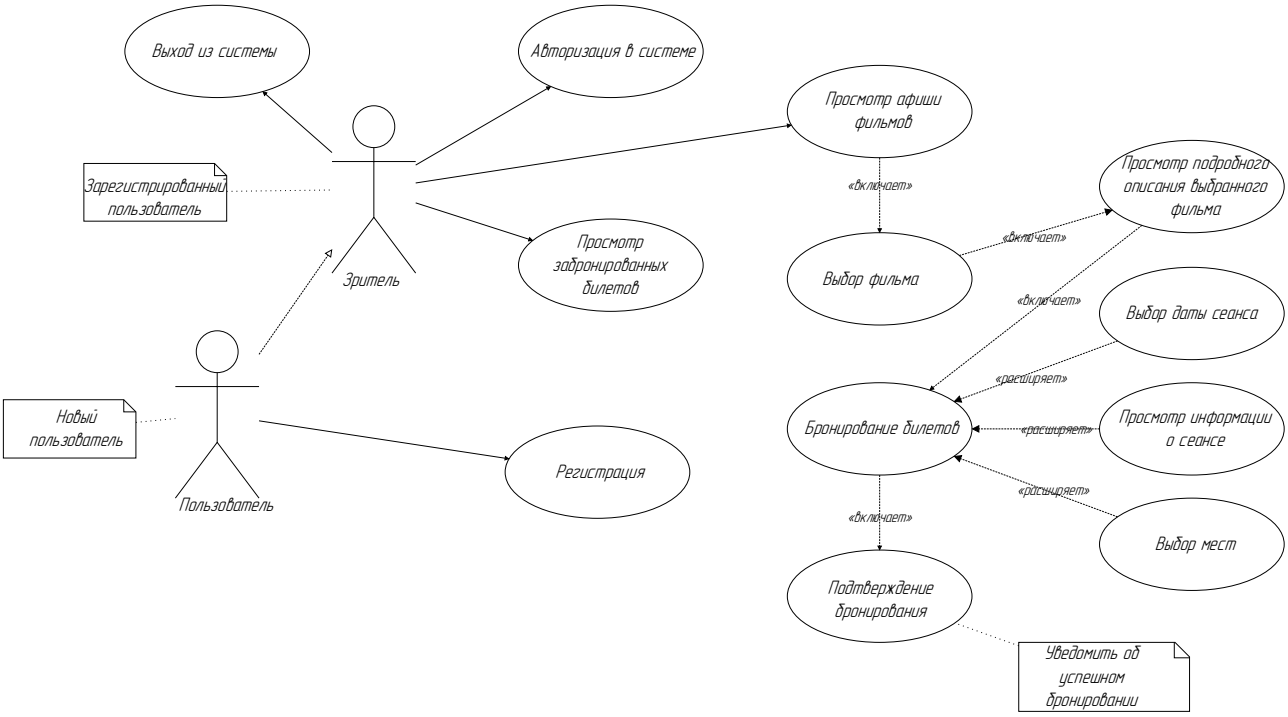

Рисунок 14 — UML диаграмма использования

3.10 UML диаграмма деятельности

Для описания поведения пользователя на основе указания потоков управления и потоков данных используем диаграмму деятельности (activity diagram).

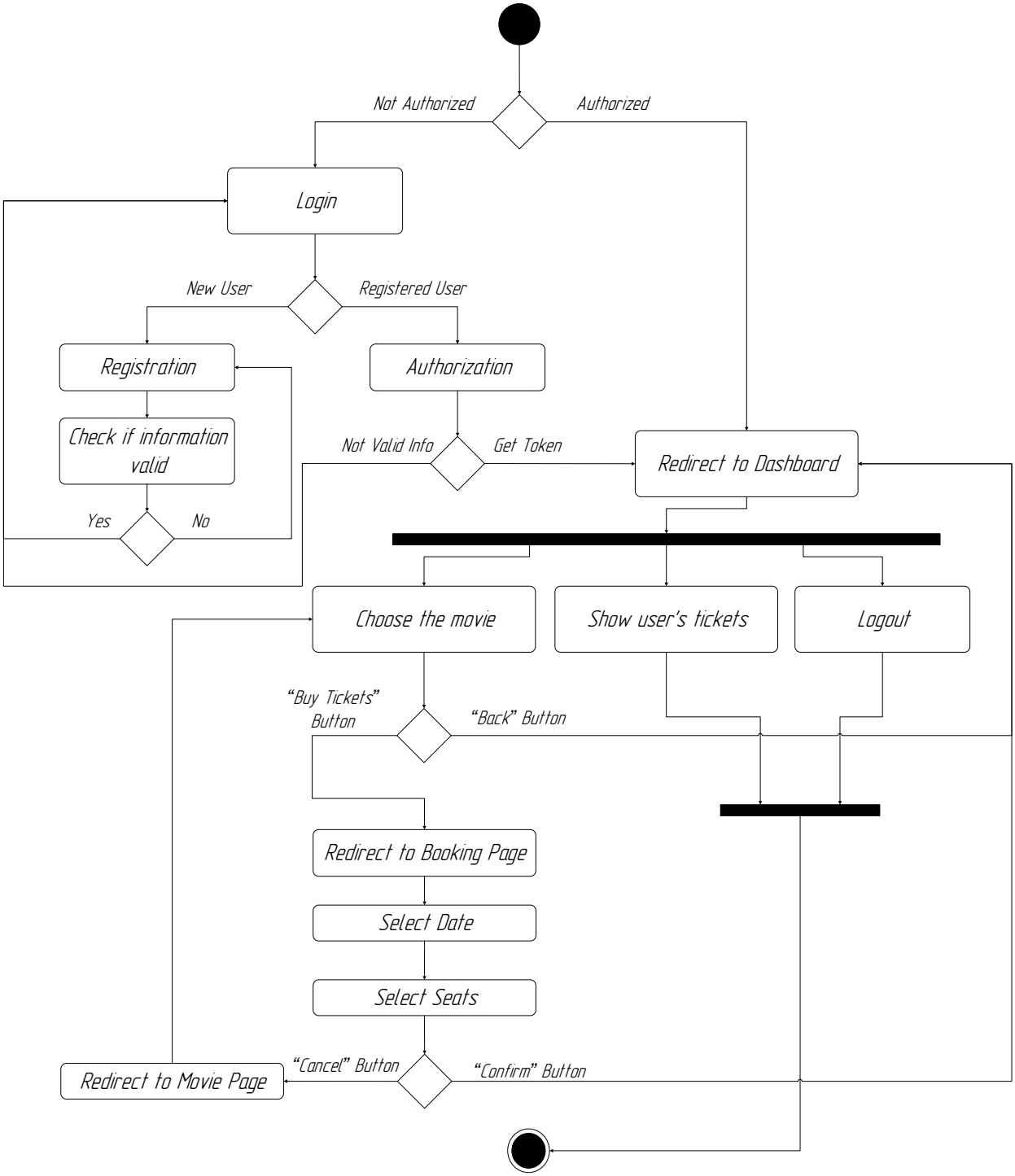

Рисунок 15 — UML диаграмма деятельности

4. ПРОЕКТНО – ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

4.1 Требования к аппаратной платформе

Исходя из размера разработанного веб-приложения и ресурсов, требуемых для его исправной работы необходимо, чтобы компьютер обладал следующими минимальными характеристиками:

- интернет-соединение на основе протокола TCP/IP с пропускной способностью 50/100 Мбит/с;
- двухъядерный процессор с тактовой частотой не менее 1.2 ГГц;
- 4Gb оперативной памяти;
- ОС Windows 8/macOS 10.12 Sierra.

Для взаимодействия с приложением необходимо наличие, компьютерной мыши и монитора.

Также возможно использование веб-приложения на портативных устройствах: ноутбуках, планшетных компьютерах, смартфонах.

4.2 Тестирование и отладка рабочей программы

Тестирование приложения производилось с использованием персонального компьютера с установленным браузером на основе Chromium. Тестирование включало в себя проверку всех функций проверку всех API функций приложения для работы с БД, корректного отображения запрашиваемой информации в браузере и отрисовки пользовательского интерфейса.

4.3 Руководство администратора

Чтобы управлять веб-приложением администратору необходимо произвести следующие действия:

- установить IDE MS Visual Studio Code или его аналог;
- клонировать проект с GitHub github.com/xreyaf/cinema-app в

локальный репозиторий;

- инициализировать проект;
- установить PostgreSQL, создать заполнить базу данных, выполнив запросы из файла server/database.sql;
- подключится к БД;
- собрать серверную часть приложения;
- собрать клиентскую часть приложения;
- развернуть приложение и БД на специализированных сервисах.

4.4 Руководство пользователя

При первом запуске веб-приложения, пользователю необходимо войти в систему используя email и пароль (рисунок 16).

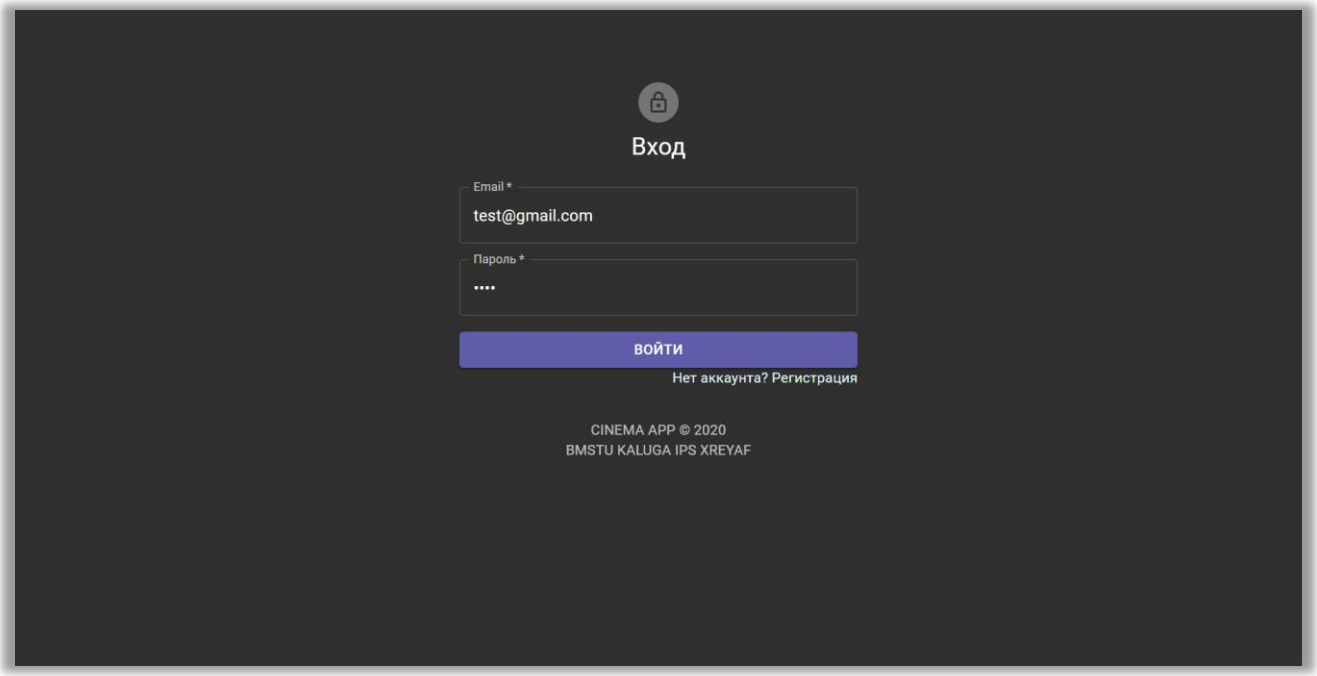

Рисунок 16 — Окно входа

Если пользователь – новый, ему следует зарегистрироваться, нажав на ссылку регистрации и заполнив форму регистрации (рисунок 17).

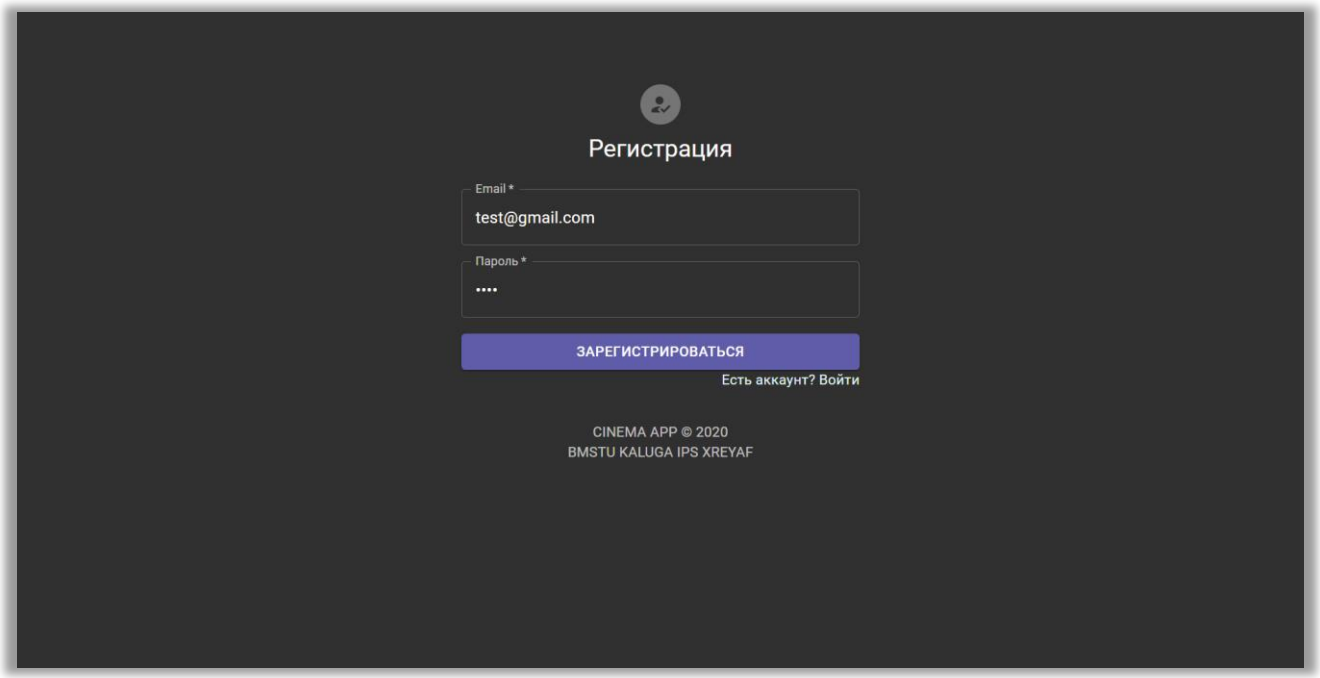

Рисунок 17 — Окно регистрации

При успешном входе откроется главное окно с каруселью фильмов (рисунок 18). Нажав на фильм, откроется окно с полным описанием (рисунок 19).

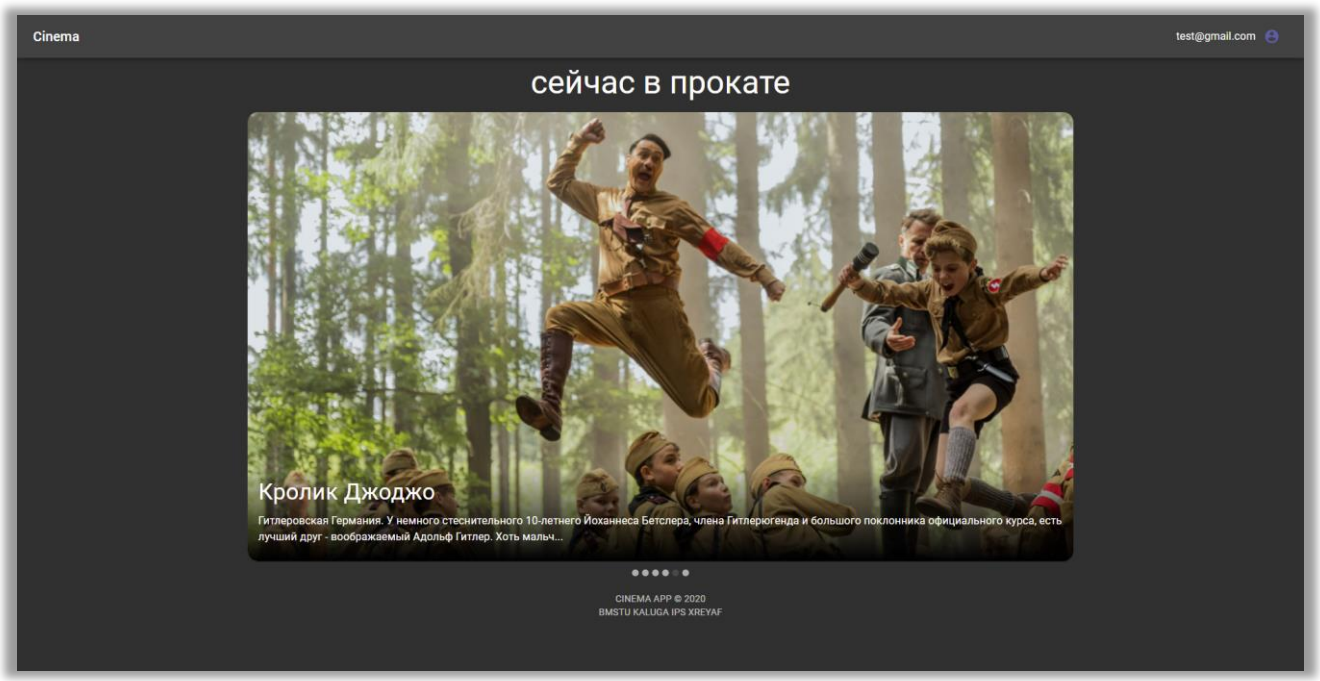

Рисунок 18 — Главное окно

Нажав на кнопку «назад», можно вернутся на главное окно, либо перейти к выбору мест, нажав на кнопку «купить билеты».

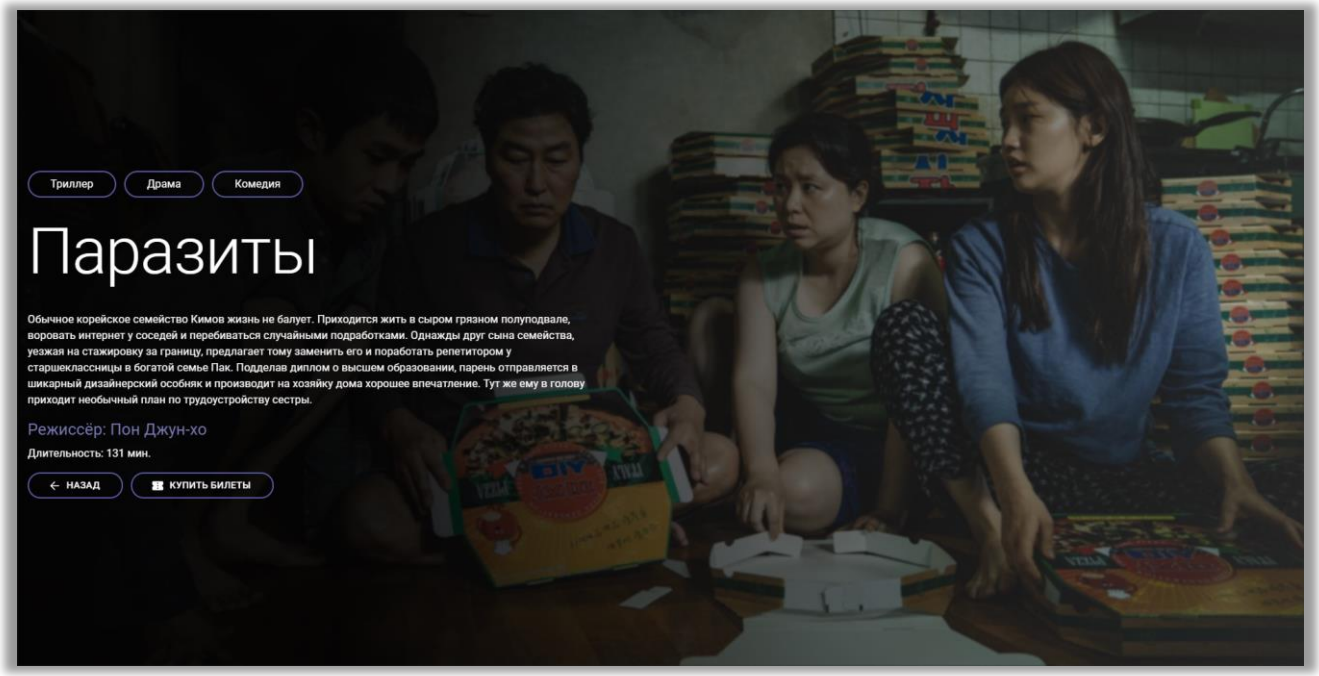

Рисунок 19 — Окно с детализацией фильма

На странице бронирования билетов, можно посмотреть информацию о сеансе, выбрать места (рисунок 20).

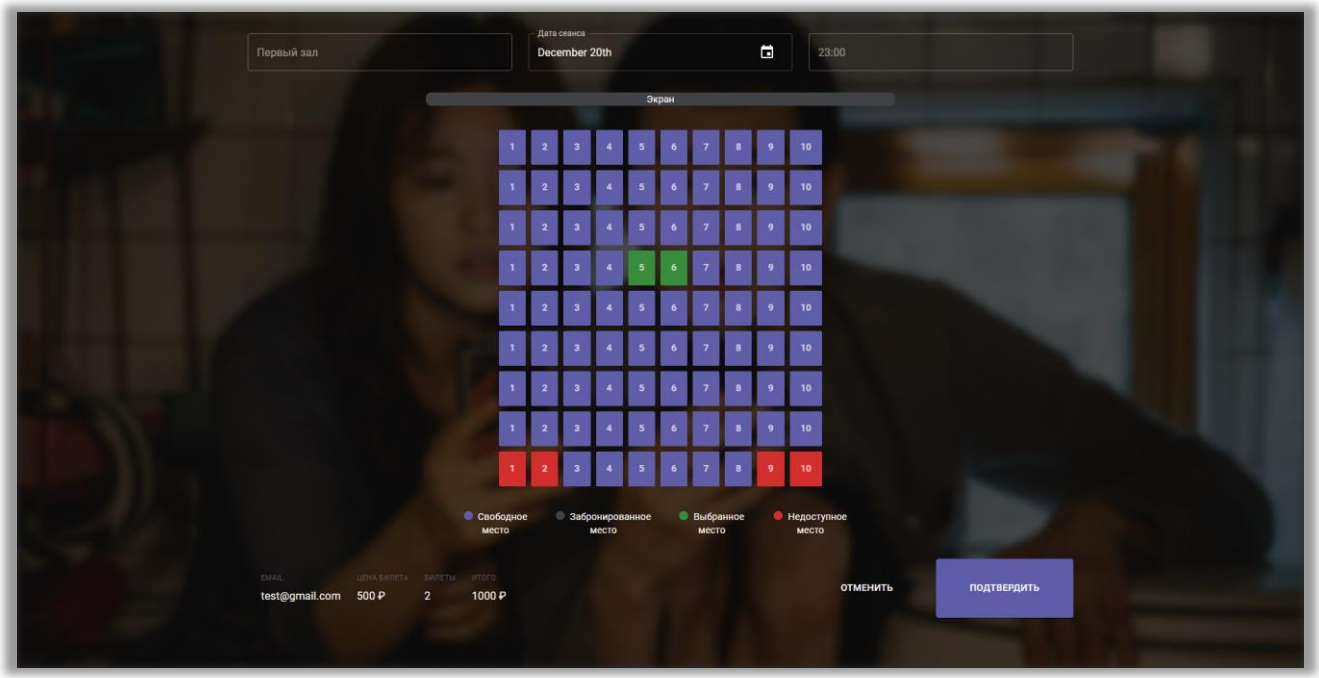

Рисунок 20 — Окно бронирования билетов

На данном этапе можно отменить действие, либо подтвердить бронь нажатием соответствующих кнопок.

Посмотреть приобретённые билеты можно путём нажатия на иконку рядом с email в правом верхнем углу и выбора соответствующего пункта меню (рисунок 21). Откроется окно с информацией о купленных билетах (рисунок 22).

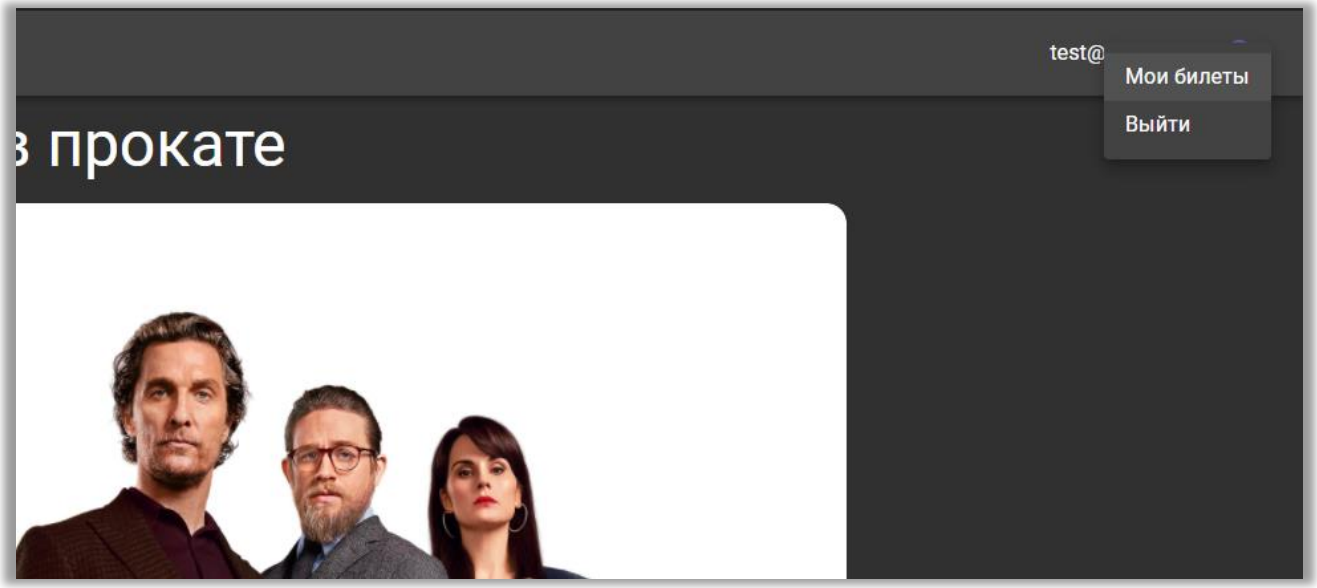

Рисунок 21 — Меню управления аккаунтом

Рисунок 22 — Билеты пользователя

Выход из системы происходит при нажатии пункта «Выйти», показанного в меню на рисунке 21.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было создано веб-приложение для автоматизации бронирования билетов в кинотеатр на языке JavaScript с интегрированной базой данных.

Веб-приложение позволяет покупать билеты на фильм в кинотеатре. Пользователь может задавать их координаты, заливать цветом и менять толщину кисти. Предусмотрена функция очистки области рисования.

Поставленные перед выполнением данной курсовой работы задачи решены, но, несмотря на это, существуют пути дальнейшего развития автоматизированной системы, так как предметная область курсовой работы предоставляет разработчику широкие возможности реализации функций и веб-интерфейсов.

Для достижения поставленной цели был решён ряд задач:

- исследование предметной области;
- проектирование концептуальной, логической и физической моделей базы данных;
- создание и заполнение базы данных;
- проектирование и реализация пользовательского интерфейса;
- интеграция базы данных и серверной части приложения;
- реализация взаимодействия клиентской и серверной частей приложения;
- написание расчётно-пояснительной записки и создание графических материалов

Веб-приложение, созданное в результате выполнения курсовой работы, отвечает поставленным в начале работы требованиям и готово к эксплуатации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кудрявцев, К. Я. Создание баз данных: учебное пособие / К. Я. Кудрявцев. — М.: НИЯУ МИФИ, 2010. — 155 с. — URL: https://e.lanbook.com/book/75822;

2. Новиков Б. А. Основы технологий баз данных: учебное пособие / Б. А. Новиков, Е. А. Горшкова, Н. Г. Графеева; под ред. Е. В. Рогова. — 2-е изд. — М.: ДМК Пресс, 2020. — 582 с — URL: https://edu.postgrespro.ru/dbtech.pdf;

3. Тарасов, С. В. СУБД для программиста. Базы данных изнутри / С. В. Тарасов. — М.: СОЛОН-Пресс, 2015. — 320 с. — URL: https://e.lanbook.com/book/64959;

4. Шёниг, Г. -. PostgreSQL 11. Мастерство разработки / Г. -. Шёниг ; перевод с английского А. А. Слинкина. — М.: ДМК Пресс, 2020. — 352 с. — URL: https://e.lanbook.com/book/131714;

5. What is PERN stack? [Электронный ресурс] //GeeksforGeeks — URL: https://www.geeksforgeeks.org/what-is-pern-stack;

6. Современный учебник JavaScript [Электронный ресурс] // learn.javascript.ru — URL: https://learn.javascript.ru/;

7. React. JavaScript-библиотека для создания пользовательских интерфейсов [Электронный ресурс]// Reactjs.org — URL: https://ru.reactjs.org;

8. Документация к PostgreSQL [Электронный ресурс]// Postgrespro.ru — URL: https://postgrespro.ru/docs/postgresql/12/index;

9. Node-postgres. Набор модулей node.js для взаимодействия с базой данных PostgreSQL [Электронный ресурс]// Node-postgres.com — URL: https://node-postgres.com/;

10. Express. Быстрый, гибкий, минималистичный веб-фреймворк для приложений Node.js [Электронный ресурс]// Expressjs.com — URL:

https://expressjs.com/ru/;

11. MATERIAL-UI React компоненты для быстрой и легкой веб-разработки. [Электронный ресурс]// Material-ui.com — URL: https://next.material-ui.com/ru/;

12. ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы»;

13. ГОСТ 34.601-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Стадии создания»;

14. Разработка Технического задания по ГОСТ 34 легко и просто [Электронный ресурс]// Habr — URL: https://habr.com/ru/post/432852.