МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра игровой индустрии (ИИ)

КУРСОВАЯ РАБОТА

по дисциплине: Бэкенд-разработка
по профилю: Программная инженерия (09.03.04)
направления профессиональной подготовки: Разработка программных продуктов и проектирование информационных систем

Тема: «Веб-сервис резервирования билетов»

Студент: Шумахер Марк Евгеньевич
Группа: ИКБО-20-22
Работа представлена к защите _____/ _____/*Шумахер М.Е.*
(подпись и ф.и.о. студента)

Руководитель: ст.преп., Рачков Андрей Владимирович

Работа допущена к защите _____(дата) _____/*Рачков А.В.*
(подпись и ф.и.о.рук-ля)

Оценка по итогам защиты: _____
_____ / *Рачков Андрей Владимирович, ст.преп* /
_____ / _____/
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2025 г.

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Бэкенд-разработка
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Шумахер Марк Евгеньевич
Группа: ИКБО-20-22
Срок представления к защите: 12.05.2025
Руководитель: Рачков Андрей Владимирович, ст. преподаватель

Тема: Веб-сервис резервирования билетов

Исходные данные: используемые технологии: Python, Fast API, PostgreSQL, VS Code, наличие: внешнего вида страниц, соответствующего современным стандартам веб-разработки, использование паттерна проектирования MVC. Нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор технологий разработки веб-приложения. 3. Разработать архитектуру веб-приложения на основе выбранного паттерна проектирования. 4. Реализовать слой серверной логики веб-приложения с применением выбранной технологии. 5. Реализовать слой логики базы данных. 6. Разработать слой клиентского представления веб-приложения 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____/Р. Г. Болбаков/, «27» февраля 2025 г.

Задание на КР выдал: _____/А.В. Рачков/, «27» февраля 2025 г.

Задание на КР получил: _____/М.Е. Шумахер/, «27» февраля 2025 г.

АННОТАЦИЯ

Отчет 29 с., 19 рис., 14 источн., 4 табл.

PYTHON, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, ПРИЛОЖЕНИЕ, FAST API, ИНТЕРНЕТ-РЕСУРС, АРХИТЕКТУРА, ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Объект исследования – клиент-серверное приложение «TicketBooking».

Цель работы – анализ, проектирование и реализация веб-сервиса по бронированию билетов.

В ходе работы был проведен анализ предметной области и обзор веб-приложений с аналогичной тематикой, а также проанализированы архитектурные стили и современные подходы к проектированию.

Результатом работы является веб-приложение «TicketBooking», учитывая постоянную потребность в посещении разных событий и концертов, предполагается, что в перспективе число пользователей веб-приложения будет расти.

ANNOTATION

Report 29 p., 19 fig., 14 sources, 4 tabl.

JAVA, SPRING FRAMEWORK, ARCHITECTURE, DESIGN PATTERNS, CLIENT-SERVER APPLICATION, INTERNET RESOURCE, EVENT

The object of the study is the client–server application "TicketBooking".

The purpose of the work is the analysis, design and implementation of a web–based ticket booking service.

In the course of the work, an analysis of the subject area and a review of web applications with a similar theme were carried out, as well as architectural styles and modern design approaches were analyzed.

The result of the work is the TicketBooking web application, given the constant need to attend various events and concerts, it is assumed that the number of users of the web application will grow in the future.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящем отчете применяют следующие термины с соответствующими определениями:

Клиент – программное обеспечение или устройство, запрашивающее ресурсы или услуги у сервера в сети

Микросервис – независимый модуль системы, выполняющий определённую бизнес-логику

Монолит – архитектура, где вся система построена как единое приложение с общей кодовой базой

Фреймворк – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта

Хостинг – услуга по предоставлению ресурсов для размещения информации на сервере, постоянно имеющем доступ к сети

Dockerfile – файл для предварительной работы, набор инструкций, который нужен для записи образа

Render – бесплатный хостинг для размещения приложений

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете применяются следующие сокращения и обозначения:

СУБД	–	система управления базами данных
JS	–	JavaScript, язык программирования
SQL	–	Structured Query Language
UML	–	Unified Modeling Language

ВВЕДЕНИЕ

Цель работы – проанализировать существующие решения в области онлайн-бронирования, изучить их функциональные возможности и особенности взаимодействия с пользователем, а также разработать собственный веб-сервис для резервирования билетов. В качестве технологической основы будут использованы язык программирования Python [1], фреймворк FastAPI [2] для создания API и PostgreSQL [3] в качестве системы управления базами данных. Для достижения поставленной цели были сформулированы следующие задачи:

– провести анализ предметной области, связанной с разрабатываемым приложением,

– выбрать инструменты для реализации серверной части,

– изучить популярные паттерны проектирования и обосновать выбор технологий для реализации выбранной архитектуры,

– разработать архитектуру и приложение с применением выбранных технологий,

– протестировать разработанное приложение.

Для выполнения этих задач применяются методы анализа и сравнительного исследования. Информационной базой для этой работы являются знания, полученные в ходе практических занятий курса «Бэкенд-разработка», а также данные из интернет-ресурсов.

В разделе с описанием предметной области расположена основная информация о проведенном анализе предметной области, функциональные и нефункциональные требования к системе, а также ограничения системы.

В разделе обоснования выбора технологий описано используемое программное обеспечение и технологии.

В разделе разработки приложения описана структура приложения, подключение к базе данных и реализация клиентской и серверной частей, а также тестирование и развертывание.

В разделе заключения подводятся итоги курсовой работы.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Анализ предметной области

Анализ предметной области проводился среди веб-приложений на тематику «Бронирование билетов на мероприятия». Были выбраны три веб-приложения: Яндекс афиша [4], Ticketland [5], MTC live [6] представленные на рисунках 1.1-1.3.

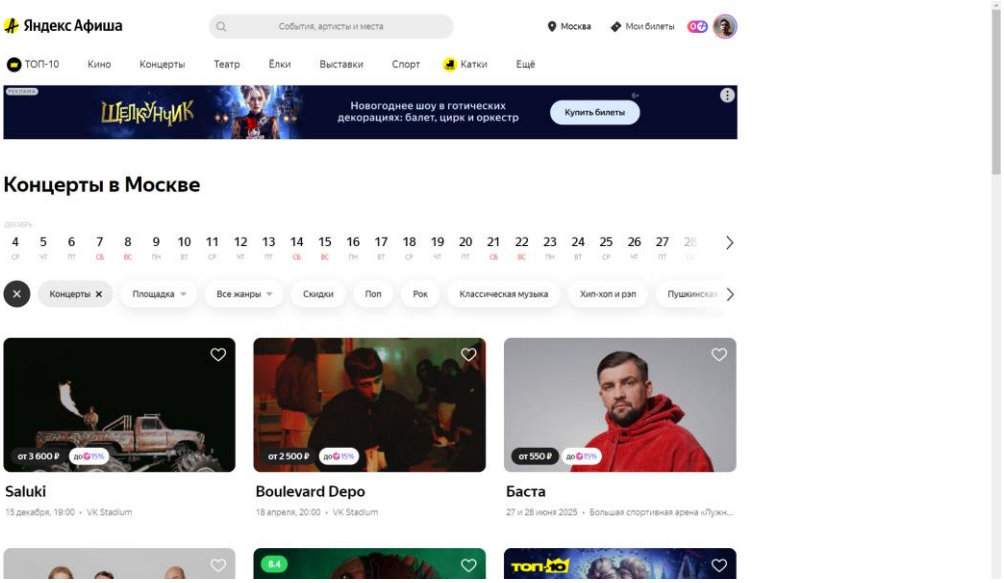

Рисунок 1.1 – Клиентская часть сайта Яндекс афиша

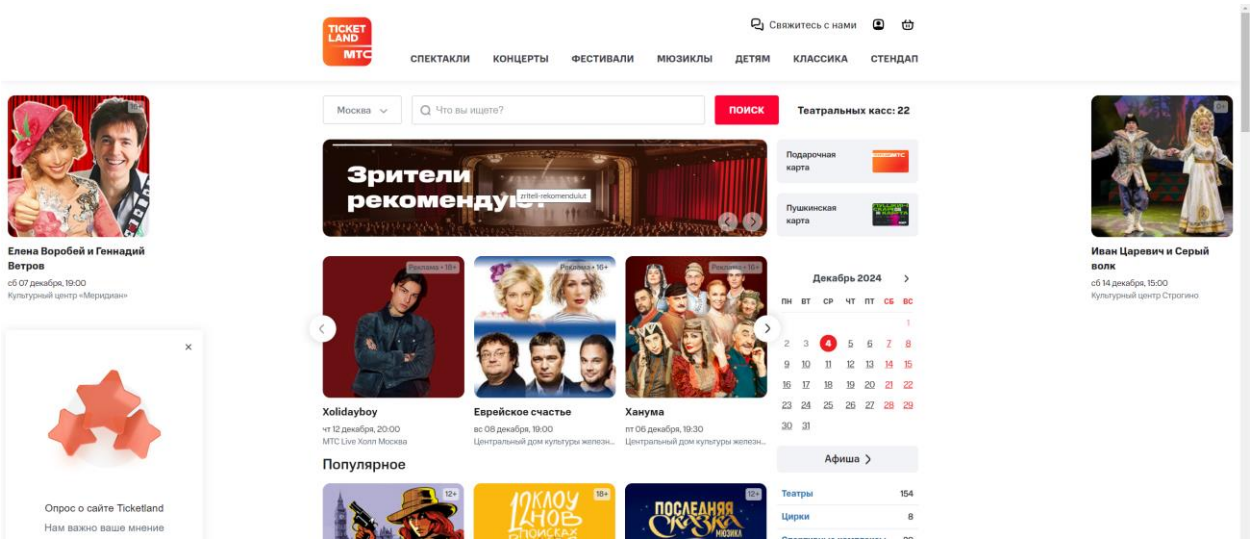

Рисунок 1.2 – Клиентская часть сайта Ticketland

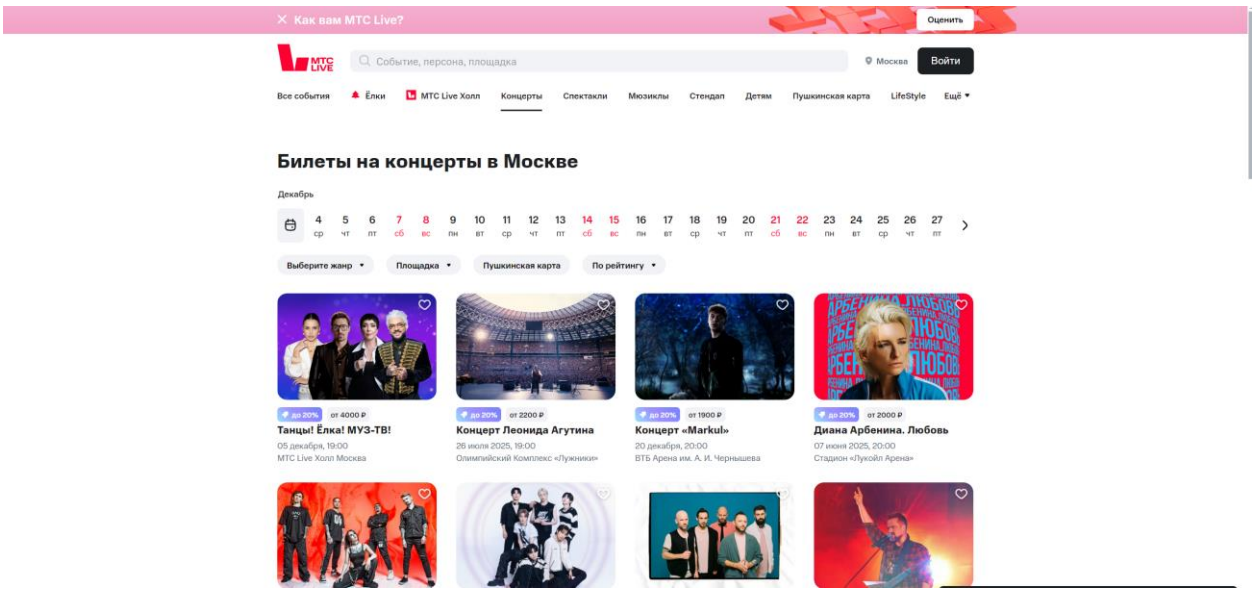

Рисунок 1.3 – Клиентская часть сайта MTC live

Все рассмотренные веб-приложения предоставляют пользователям возможность регистрации и входа в аккаунт, управления личными данными, записи на мероприятия, а также просмотра информации о доступных событиях.

Из анализа данных веб-приложений следует выделить ключевые возможности, которыми должен обладать разрабатываемый веб-ресурс данной тематики:

– регистрация на выбранное мероприятие,

– просмотр списка забронированных билетов,

– просмотр количества оставшихся билетов на мероприятия,

– просмотр списка доступных мероприятий.

Приложение должно быть разработано с учетом требований современного рынка веб-приложений в данной тематике, обеспечивая конкурентоспособность. Для этого необходимо внедрить актуальные функции, такие как предоставление информации о мероприятиях, а также возможность просмотра и бронирования доступных билетов. Кроме того, важно обеспечить удобство интерфейса, высокую производительность и адаптацию под различные устройства для привлечения широкой аудитории.

Уникальность приложения заключается в его простоте и удобстве для пользователей. Интуитивно понятный интерфейс позволяет легко разобраться в функционале, а отсутствие сложной системы авторизации делает процесс еще быстрее и проще. Обычно люди проверяют билеты уже непосредственно перед мероприятием, и возможность мгновенно увидеть свои билеты, не тратя время на авторизацию, является значительным преимуществом. Это не только экономит время, но и создает положительное впечатление от использования приложения, что способствует его привлекательности для пользователей.

1.2 Описание функционала приложения

В разрабатываемом приложении для регистрации на мероприятия необходимо реализовать следующие функции: пользователь должен иметь возможность зарегистрироваться через email; перед регистрацией система должна проверять доступность мест на мероприятие; пользователь должен видеть список всех доступных мероприятий, включая их названия и количество оставшихся мест. Для удобства пользователь должен иметь возможность просматривать список мероприятий, на которые он уже зарегистрировался, с указанием их названий и уникальных номеров билетов. Система должна обрабатывать ошибки, возникающие в процессе регистрации или при работе с данными, и информировать пользователя о таких ситуациях.

Среди нефункциональных требований ключевое внимание уделяется удобству и производительности интерфейса, который должен быть интуитивно понятным и обеспечивать быстрый доступ к основным функциям. Создание и просмотр записей должны происходить без задержек. Взаимодействие клиента и сервера должно быть реализовано оптимальным образом для обеспечения стабильной работы приложения даже при большом количестве одновременных пользователей. Система должна быть легко масштабируемой, чтобы поддерживать рост нагрузки и объема данных. Это, например, включает в себя увеличение количества одновременных активных пользователей с 100 до 1000, а также расширение общей пользовательской базы с 500 до 10 000 зарегистрированных пользователей, которые когда-либо

взаимодействовали с сервисом. Кроме того, архитектура должна эффективно обрабатывать рост объема данных в базе данных от 100 мегабайт до 10 гигабайт, обеспечивая стабильность, производительность и отказоустойчивость на всех этапах масштабирования. Особое внимание необходимо уделить безопасности данных пользователей, включая шифрование конфиденциальной информации, такой как email.

Проект имеет ряд ограничений, которые могут повлиять на его работу. Система будет функционировать в условиях ограниченных серверных ресурсов, а именно – оперативная память от 2 ГБ, а процессорное время от 300 секунд CPU в сутки. Это потребует эффективной оптимизации алгоритмов и процессов для снижения нагрузки на серверы, поддержания стабильности работы и обеспечения высокой производительности при увеличении нагрузки. Возможны задержки при обработке запросов, если в базе данных хранится большое количество данных, например, при значительном числе зарегистрированных мероприятий или пользователей. Увеличение нагрузки на сервер может привести к замедлению работы приложения, особенно при большом количестве пользователей, одновременно выполняющих действия, такие как регистрация или просмотр мероприятий.

Разработка и тестирование ключевых компонентов системы должны быть завершены в срок не более 1-2 месяцев, что обеспечит своевременный запуск проекта и готовность к увеличению нагрузки.

1.3 Описание функционала приложения в нотации UML

Для создания UML-диаграмм [7] использован сервис draw.io [8].

Согласно функциональным требованиям создана диаграмма Use-case, показанная на рисунке 1.4.

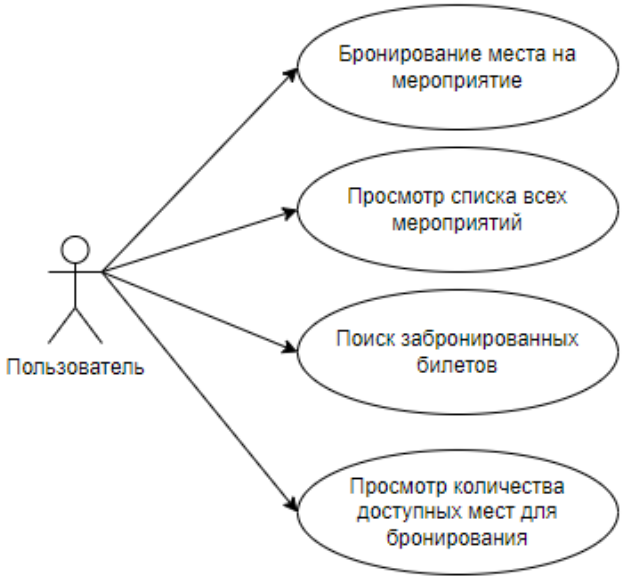

Рисунок 1.4 – Диаграмма Use-case

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

2.1 Используемое прикладное программное обеспечение

Для того, чтобы определить наиболее подходящую среду разработки проведем сравнительный анализ сред разработки VS Code [9] и PyCharm [10], как показано в таблице 2.1.

Таблица 2.1 – Характеристики сред разработки

Характеристика	PyCharm	VS Code
Поддержка Python	Глубокая интеграция, продвинутый автодополнение, рефакторинг	Требует расширений (например, Pylance)
Производительность	Ресурсоемкий (требует больше ОЗУ)	Легковесный, быстрый запуск
Интеграция с FastAPI	Встроенная поддержка через плагины	Требует ручной настройки (расширения)
Работа с базами данных	Встроенные инструменты для PostgreSQL (в Pro-версии)	Требует расширений (например, SQLTools)
Отладка и тестирование	Продвинутые возможности отладки и профилирования	Базовая отладка + расширения (напр., Python Debugger)
Интеграция с Git	Полноценная встроенная поддержка	Требует расширения (GitLens и др.)

Для разработки приложения выбран VS Code. Это окружение обладает широкими возможностями кастомизации, имеет низкие системные требования при сохранении высокой функциональности, поддерживает все необходимые технологии через систему расширений, обеспечивает кроссплатформенную совместимость. Данный выбор оптимально соответствует требованиям проекта по производительности, гибкости и удобству разработки.

Для проверки работы клиент-серверного приложения был выбран браузер Google Chrome, поскольку он широко используется, обладает высокой производительностью и поддерживает современные веб-стандарты.

2.2 Используемые технологии

2.2.1 Языки программирования

Для выбора языка программирования для разработки клиент-серверного приложения был проведён сравнительный анализ нескольких популярных языков, учитывающий такие ключевые аспекты, как платформенная независимость, производительность, библиотеки и фреймворки, поддержка многозадачности, простота и удобство разработки, безопасность. В рамках

анализа были рассмотрены Java, Python и C++, каждый из которых обладает своими сильными сторонами и ограничениями, зависящими от требований проекта. Основные характеристики этих языков приведены в таблице 2.2.

Таблица 2.2 – Характеристики языков программирования

Характеристики	Python (FastAPI)	Java (Spring Boot)	C++ (Qt Framework)
Платформенная независимость	Высокая (кроссплатформенный интерпретатор)	Высокая (JVM)	Ограниченная (требует пересборки)
Производительность	Достаточная для веб-сервисов	Высокая	Максимальная
Библиотеки и инструменты	Богатая экосистема (Pydantic, SQLAlchemy)	Spring Ecosystem (Hibernate, Security)	Qt Libraries (GUI, сетевые модули)
Поддержка многозадачности	Asyncio для асинхронных операций	Многопоточность и Reactive Streams	Низкоуровневые потоки
Скорость разработки	Очень высокая (минимум boilerplate-кода)	Средняя (требует конфигурации)	Низкая (ручное управление ресурсами)
Безопасность	Встроенные механизмы валидации	Enterprise-уровень (Spring Security)	Ответственность на разработчике

Для серверной разработки Python обладает рядом конкурентных преимуществ, включая высокую производительность, отказоустойчивость и низкий порог вхождения. Эти характеристики делают его особенно подходящим для backend-разработки. Использование фреймворка FastAPI дополнительно оптимизирует процесс. FastAPI предлагает встроенную поддержку асинхронности, автоматическую генерацию документации и простую интеграцию с базами данных, что критически важно для подобных систем.

2.2.2 Базы данных

Для выбора подходящей базы данных для разрабатываемого клиент-серверного приложения был проведен сравнительный анализ нескольких популярных баз данных. В рамках анализа были рассмотрены такие варианты, как PostgreSQL, MongoDB [11] и MySQL [12]. Основные характеристики этих баз данных, а также их сильные и слабые стороны, приведены в таблице 2.3.

Таблица 2.3 – Характеристики баз данных

Характеристики	PostgreSQL	MongoDB	MySQL
Тип базы данных	Реляционная	Документо-ориентированная	Реляционная
Масштабируемость	Хорошая вертикальная и горизонтальная масштабируемость	Отличная горизонтальная масштабируемость (sharding)	Хорошая вертикальная масштабируемость
Производительность	Высокая производительность для сложных запросов	Высокая производительность для больших объемов неструктурированных данных	Хорошая производительность для стандартных SQL-запросов
Интеграция с Python	Отличная (psycopg2, SQLAlchemy)	Хорошая (PyMongo, Motor для async)	Хорошая (mysql-connector, SQLAlchemy)
Гибкость схемы	Строгая схема (нужна схема таблицы)	Очень гибкая (без схемы)	Строгая схема (нужна схема таблицы)

Выбор PostgreSQL в качестве базы данных был обусловлен рядом факторов, которые делают её идеальным выбором для разрабатываемого клиент-серверного приложения. PostgreSQL обладает мощными возможностями для работы с реляционными данными, включая полную поддержку SQL, сложные запросы, транзакции и интеграцию с различными типами данных. Это обеспечило высокую гибкость в хранении и управлении данными приложения, что важно для обеспечения надежности и эффективности работы.

2.2.3 Инструменты DevOps

Для хостинга клиент-серверного приложения был выбран облачный сервис Render [13], который предоставляет автоматизированные инструменты для развертывания, масштабирования и управления инфраструктурой. Это решение позволяет разработчикам сосредоточиться на разработке функционала приложения, избавляя от необходимости вручную настраивать серверы и управлять ресурсами. Render поддерживает широкий спектр технологий, таких как Docker, Node.js и Java, что делает его гибким и универсальным решением для хостинга различных типов приложений. Интеграция с GitHub позволяет автоматизировать обновление приложения

при каждом изменении в репозитории. Также, благодаря функции автоматического масштабирования, Render эффективно управляет ресурсами в зависимости от текущей нагрузки, обеспечивая высокую доступность и стабильную производительность системы. На рисунке 2.1 представлена диаграмма компонентов.

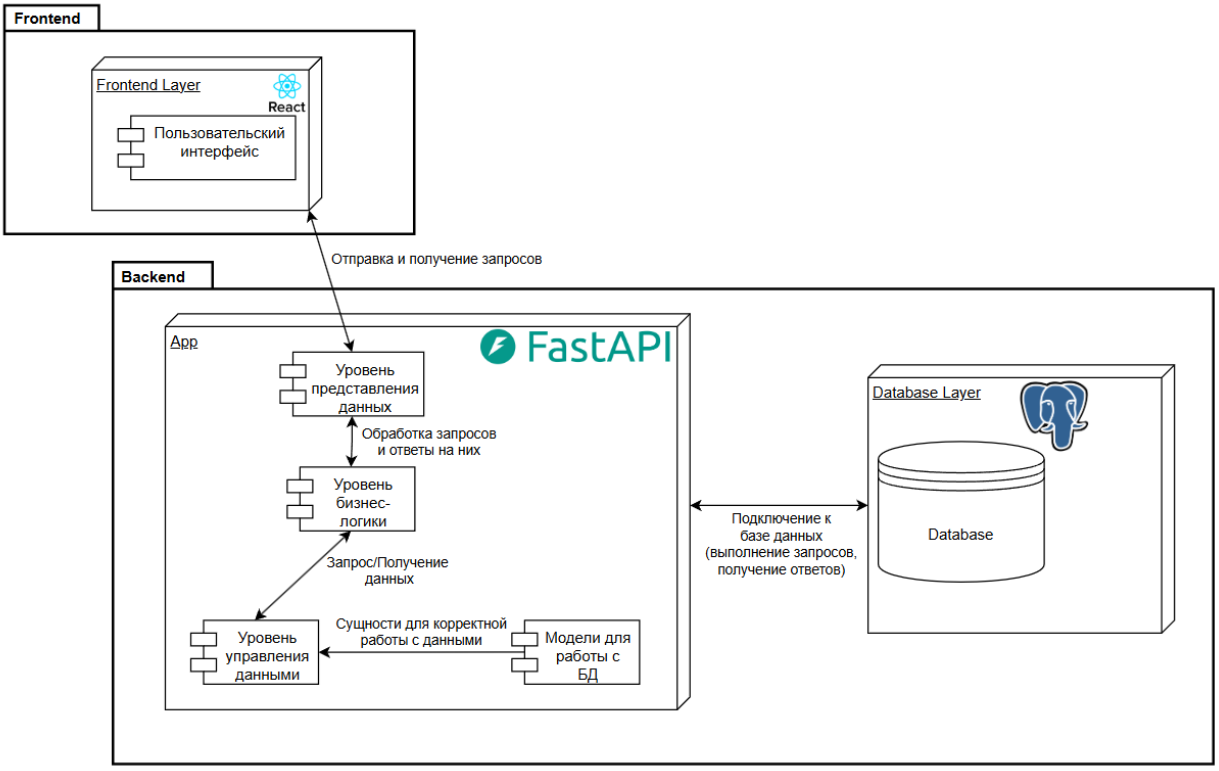

Рисунок 2.1 – Диаграмма компонентов системы

2.3 Анализ архитектурных стилей и паттернов проектирования

Для выбора архитектурного стиля разрабатываемого приложения необходимо провести анализ наиболее популярных архитектур.

Одним из наиболее распространенных является монолитная архитектура. Монолитная архитектура – это подход, при котором все компоненты и функции приложения объединены в единую систему. Она выделяется своей простотой в разработке и развертывании: все элементы находятся в одном месте, что уменьшает затраты времени и ресурсов. Кроме того, тестировать монолитное приложение проще, так как вся система проверяется как целое. Однако масштабируемость в монолите представляет собой вызов: для увеличения производительности необходимо

масштабировать всю систему, а не отдельные её части. Это усложняет обновления, поскольку изменения одного компонента могут повлиять на другие части, увеличивая вероятность ошибок и сбоев. Кроме того, использование разных технологий внутри одной системы ограничено.

Клиент-серверная архитектура, другой популярный стиль, основывается на разделении приложения на две части: клиент, запрашивающий услуги, и сервер, предоставляющий их. Основное преимущество такого подхода – чёткое разделение ролей, что упрощает поддержку и развитие системы. Однако этот стиль имеет и слабые стороны, например, зависимость от сети. Если клиент генерирует слишком много запросов, сервер может испытывать перегрузку, что негативно влияет на производительность системы.

Сервисно-ориентированная архитектура (SOA) строится на идее создания приложения в виде набора взаимосвязанных сервисов, каждый из которых выполняет конкретную задачу. Такой подход позволяет повторно использовать компоненты, снижая затраты на разработку, и упрощает масштабирование отдельных частей системы. Однако взаимодействие между сервисами часто требует сложного управления. Для обеспечения совместимости и стабильной работы приложения требуется значительное внимание и усилия.

Микросервисная архитектура – это современный подход, развивающий идеи сервисно-ориентированной архитектуры (SOA). Она предполагает деление приложения на множество небольших и автономных сервисов, каждый из которых отвечает за выполнение своей задачи. Такой стиль дает высокую гибкость: для каждого микросервиса можно использовать свои технологии и языки программирования, что облегчает адаптацию системы к разным требованиям. Кроме того, микросервисы упрощают процесс развертывания и обновления: отдельные сервисы можно обновлять независимо, без необходимости вмешательства в работу всего приложения.

Для разрабатываемого приложения по регистрации на мероприятия был выбран монолитный архитектурный стиль. Это решение обусловлено рядом факторов.

Во-первых, приложение имеет относительно простую структуру. Его основные функции – регистрация на мероприятие, просмотр списка доступных событий и бронирование тесно связаны между собой и не требуют сложной декомпозиции. Во-вторых, монолит упрощает и ускоряет разработку. Все модули объединены в одном проекте, что снижает затраты на организацию взаимодействия между компонентами. Это особенно важно при ограниченных сроках и ресурсах.

Также упрощается тестирование и развёртывание. Приложение можно проверять и внедрять как единое целое, без необходимости настройки и согласования независимых сервисов. Монолит лучше подходит для небольших систем с предсказуемой нагрузкой. Даже при росте количества пользователей, нагрузка остаётся управляемой за счёт масштабирования сервера.

Наконец, внесение изменений происходит быстрее, поскольку не требует координации между командами, как в микросервисной архитектуре. Это важно при возможных изменениях требований в ходе разработки. Таким образом, выбор монолита для данного проекта является обоснованным и оптимальным.

Для выбора архитектурного паттерна разрабатываемого приложения необходимо разобрать наиболее популярные виды. Сравнительный анализ характеристик архитектурных паттернов приведён в таблице 2.4.

Таблица 2.4 – Характеристики архитектурных паттернов

Характеристика	MVC (Model-View-Controller)	MVP (Model-View-Presenter)	MVVM (Model-View-ViewModel)
Сложность реализации	Низкая. Простая реализация на начальных этапах.	Средняя. Требует больше усилий для реализации из-за добавления презентера.	Высокая. Реализация сложнее всего из-за необходимости создания ViewModel.
Использование слоев	Да	Да	Да

Продолжение таблицы 2.4

Удобство тестирования	Хорошая, но тестирование контроллера сложно.	Очень высокая, презентер легко тестировать.	Отличная, ViewModel полностью изолируется.
Гибкость	Средняя, сложно добавлять новые функции.	Высокая, благодаря слабой зависимости слоев.	Очень высокая, легко адаптируется под изменения.
Тестируемость	Высокая	Очень высокая	Высокая
Поддержка изменений	Затруднена из-за сильных зависимостей.	Удобная благодаря изоляции логики в презентере.	Лучшая, изменения легко изолируются.
Масштабируемость	Умеренная, часто требует переписывания кода.	Высокая, легко добавлять новые функции.	Отличная, минимальное вмешательство в код.
Поддержка изменений	Затруднена из-за сильных зависимостей.	Удобная благодаря изоляции логики в презентере.	Лучшая, изменения легко изолируются.

Для разработки приложения наиболее подходящим является архитектурный паттерн MVC (Model-View-Controller). Он разделяет логику на модель, представление и контроллер, что делает процесс разработки и поддержки более удобным. Данный паттерн идеально подходит для приложений с простым интерфейсом, не требующих сложных взаимодействий, характерных для MVP или MVVM. MVC обеспечивает эффективное управление данными и их отображением при сохранении низкой сложности реализации.

3 РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Разработка серверной части приложения

После анализа предметной области, выбора архитектурного стиля, описания компонентов системы и сценариев их взаимодействия, а также определения необходимых технологий, начинается этап непосредственной реализации системы.

Разработка началась с серверной части, в рамках которой были реализованы ключевые точки взаимодействия между клиентом и сервером.

На рисунках 3.1-3.4 представлена основная логика работы серверной части приложения.

```python
@app.get("/events", response_model=list[EventOut])
def list_events(db: Session = Depends(get_db)):
    return db.query(Event).all()

@app.post("/book", response_model=BookingOut)
def book_ticket(data: BookingCreate, db: Session = Depends(get_db)):
    event = db.query(Event).filter(Event.id == data.event_id).first()
    if not event:
        raise HTTPException(status_code=404, detail="Event not found")
    if event.available_tickets <= 0:
        raise HTTPException(status_code=400, detail="No tickets available")


    user_bookings = db.query(Booking).filter(Booking.email == data.email, Booking.event_id == data.event_id).all()
    total_booked = len(user_bookings)

    if total_booked + data.ticket_count > 5:
        raise HTTPException(status_code=400, detail=f"Вы уже забронировали {total_booked}, максимальное количество билетов для бронирования - 5.")


    ticket_numbers = []
    for _ in range(data.ticket_count):
        ticket_number = str(uuid.uuid4())
        booking = Booking(
            email=data.email,
            event_id=data.event_id,
            ticket_number=ticket_number
        )
        db.add(booking)
        ticket_numbers.append(ticket_number)

    event.available_tickets -= data.ticket_count
    db.commit()

    return BookingOut(
        ticket_number=", ".join(ticket_numbers),
        email=data.email,
        event_name=event.name
    )

@app.get("/bookings/{email}", response_model=list[BookingOut])
def get_bookings_by_email(email: str, db: Session = Depends
    bookings = db.query(Booking).join(Event).filter(Booking.email == email).all()
    return [
        BookingOut(
            ticket_number=b.ticket_number,
            email=b.email,
            event_name=b.event.name
        ) for b in bookings
    ]
```

Загрузка...

Рисунок 3.1 – Конечные точки приложения

```python
from pydantic import BaseModel, EmailStr

class EventOut(BaseModel):
    id: int
    name: str
    available_tickets: int

    class Config:
        from_attributes = True

class BookingCreate(BaseModel):
    email: EmailStr
    event_id: int
    ticket_count: int

class BookingOut(BaseModel):
    ticket_number: str
    email: EmailStr
    event_name: str
```

Рисунок 3.2 – Схемы данных для работы Pydantic

```python
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from database import engine
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Event(Base):
    __tablename__ = "events"
    id = Column(Integer, primary_key=True)
    name = Column(String, unique=True, nullable=False)
    available_tickets = Column(Integer, default=0)
    bookings = relationship("Booking", back_populates="event")

class Booking(Base):
    __tablename__ = "bookings"
    id = Column(Integer, primary_key=True)
    email = Column(String, index=True)
    ticket_number = Column(String, unique=True, index=True)
    event_id = Column(Integer, ForeignKey("events.id"))
    event = relationship("Event", back_populates="bookings")
```

Рисунок 3.3 – Модели базы данных

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import os

DATABASE_URL = os.getenv("DATABASE_URL", "postgresql://postgres:password@localhost:5432/postgres")
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

Рисунок 3.4 – Подключение к базе данных

3.2 Клиентская часть приложения

Для разработки клиентской части использовался фреймворк React [14]. Клиентская часть приложения состоит из страницы, которая предоставляет пользователю возможность взаимодействовать с сервисом для бронирования билетов на различные события, рисунок 3.5.

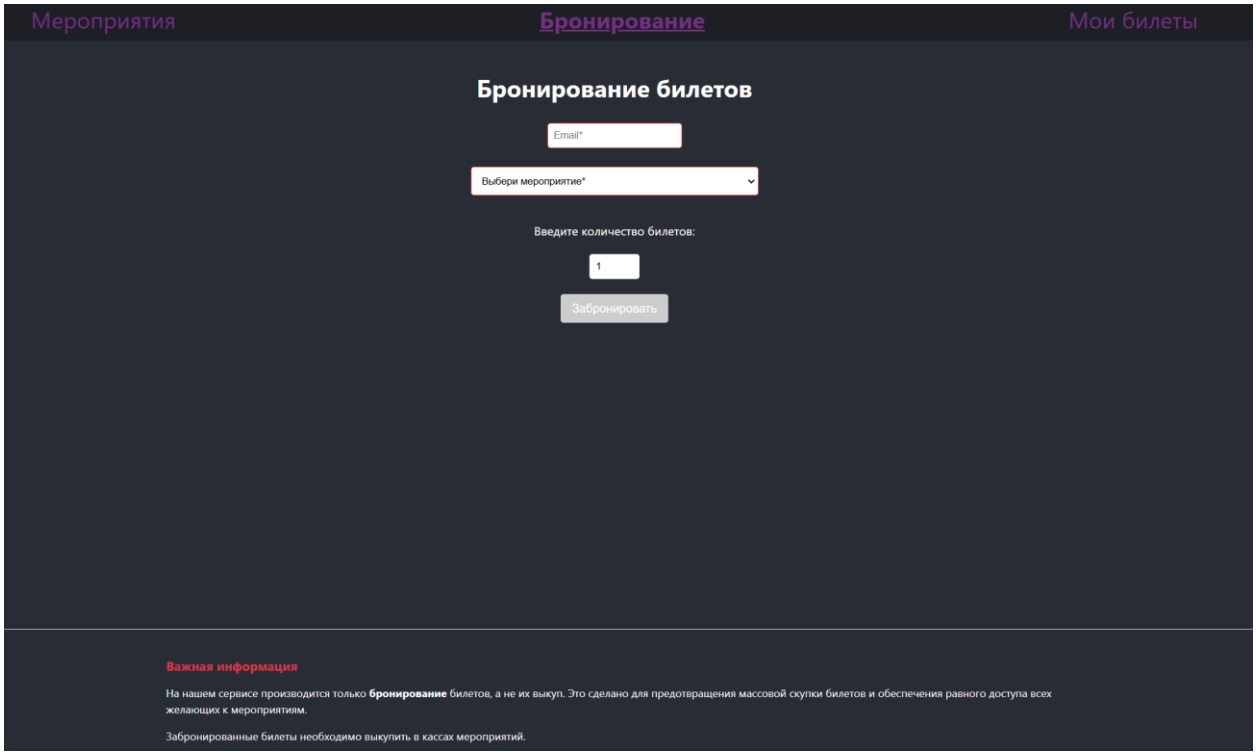

Рисунок 3.5 – Главная страница

Заполнив форму на бронь, появится сообщение об успешной регистрации на мероприятие, рисунок 3.6.

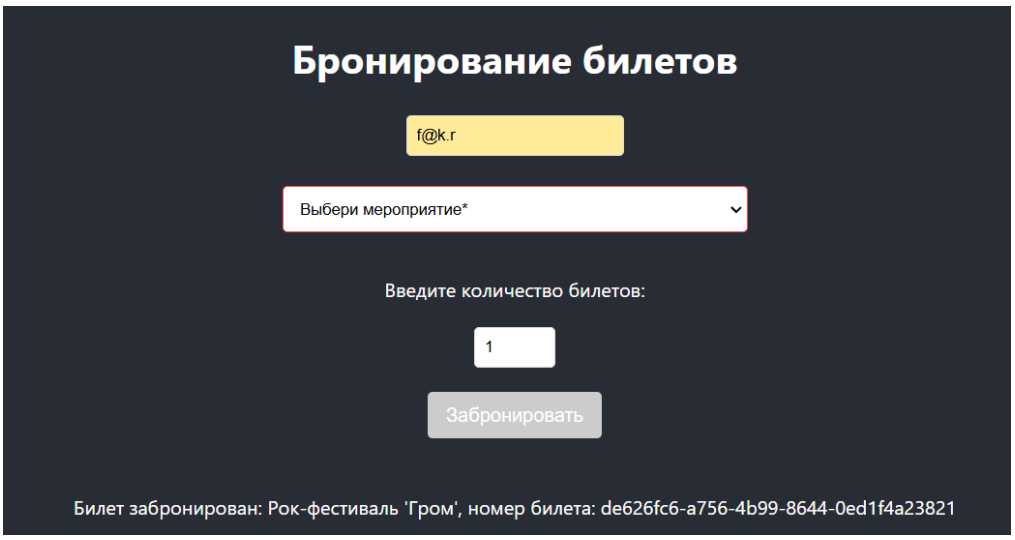

Рисунок 3.6 – Успешная регистрация

Также есть страница для поиска билетов через email, введя в форму почту, выйдет список мероприятий, на которые зарегистрировался пользователь, рисунок 3.7.

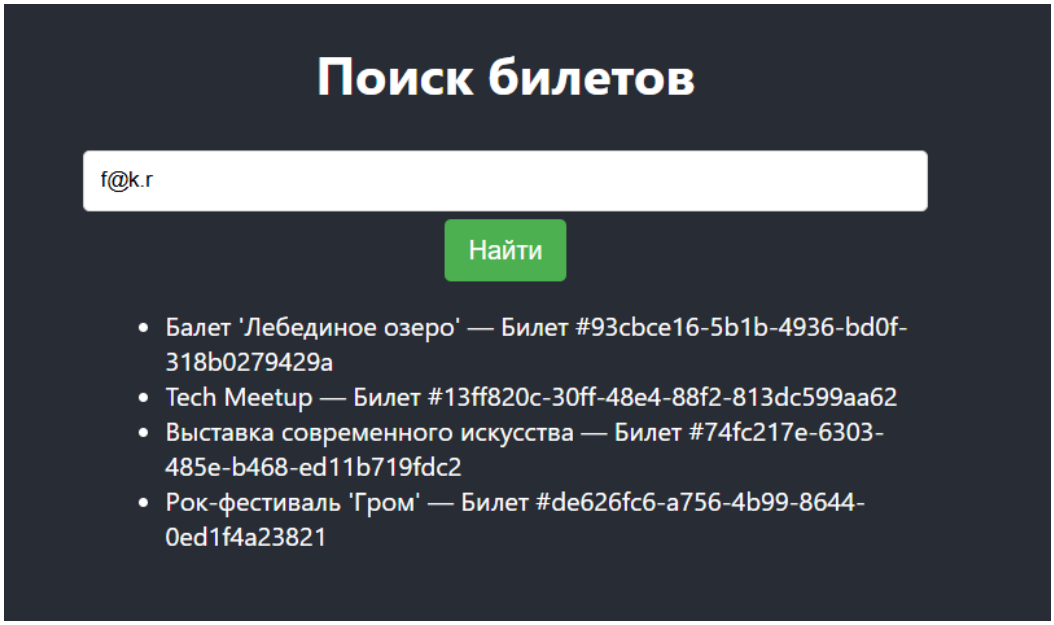

Рисунок 3.7 – Поиск билетов

3.3 Тестирование приложения

Используя программу Postman и браузер, были протестированы несколько конечных точек приложения. Для начала получим список всех мероприятий, рисунок 3.8.

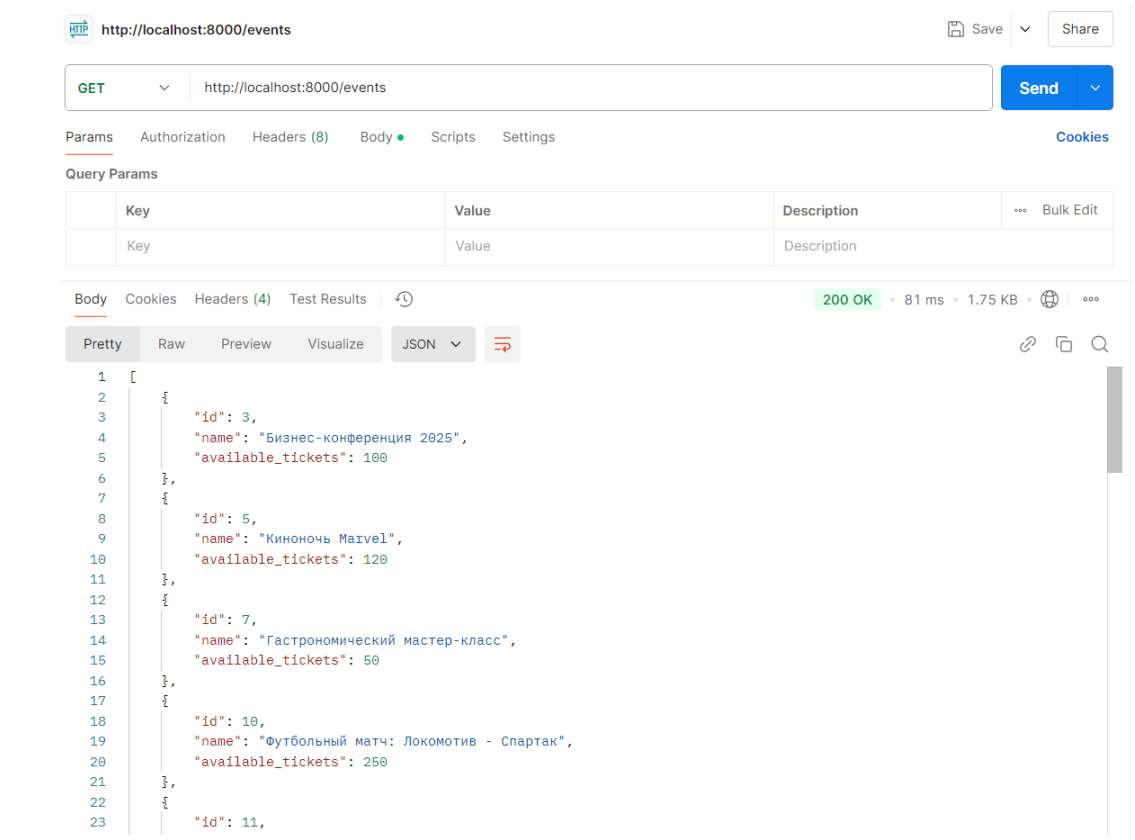

Рисунок 3.8 – GET запрос на получение всех мероприятий

Протестируем регистрацию на мероприятие, рисунок 3.9.

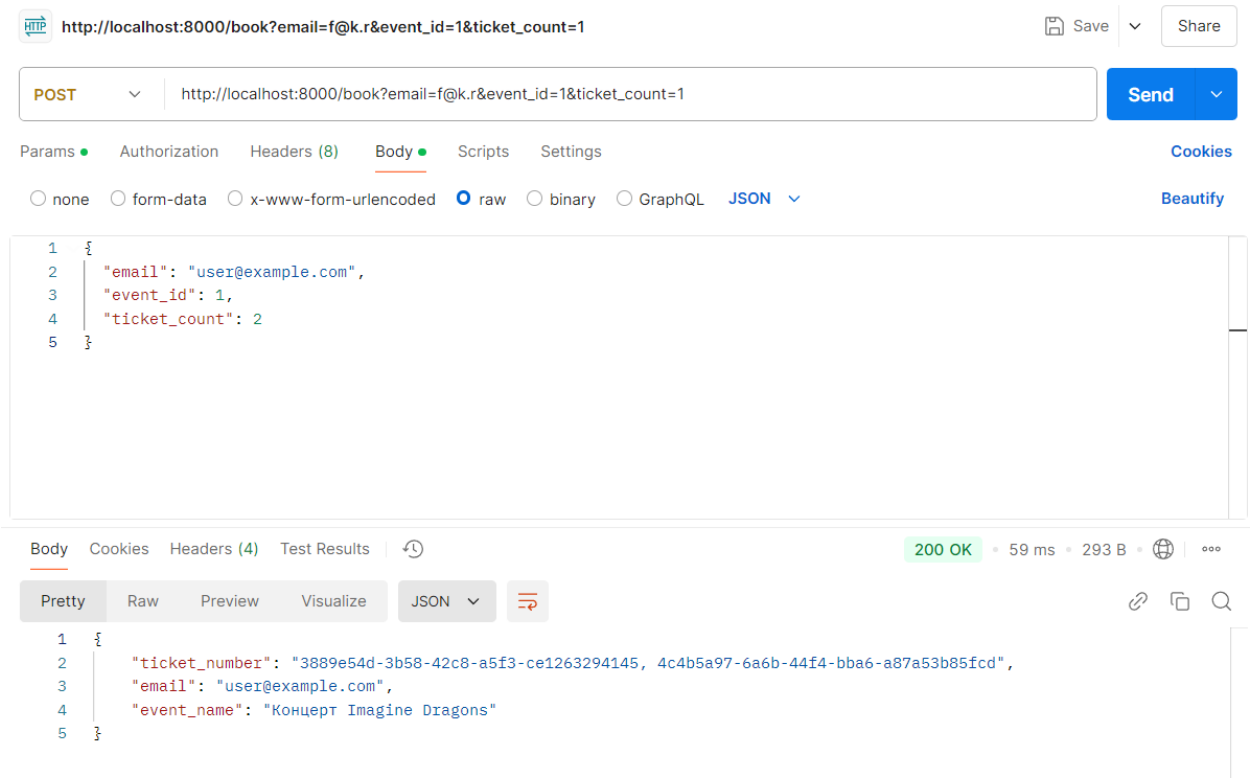

Рисунок 3.9 – POST запрос регистрации на мероприятие

Далее выполним тоже самое ничего не меняя, чтобы отловить ошибку о том, что на 1 почту нельзя купить более 5 билетов на одно и то же мероприятие, рисунок 3.10.

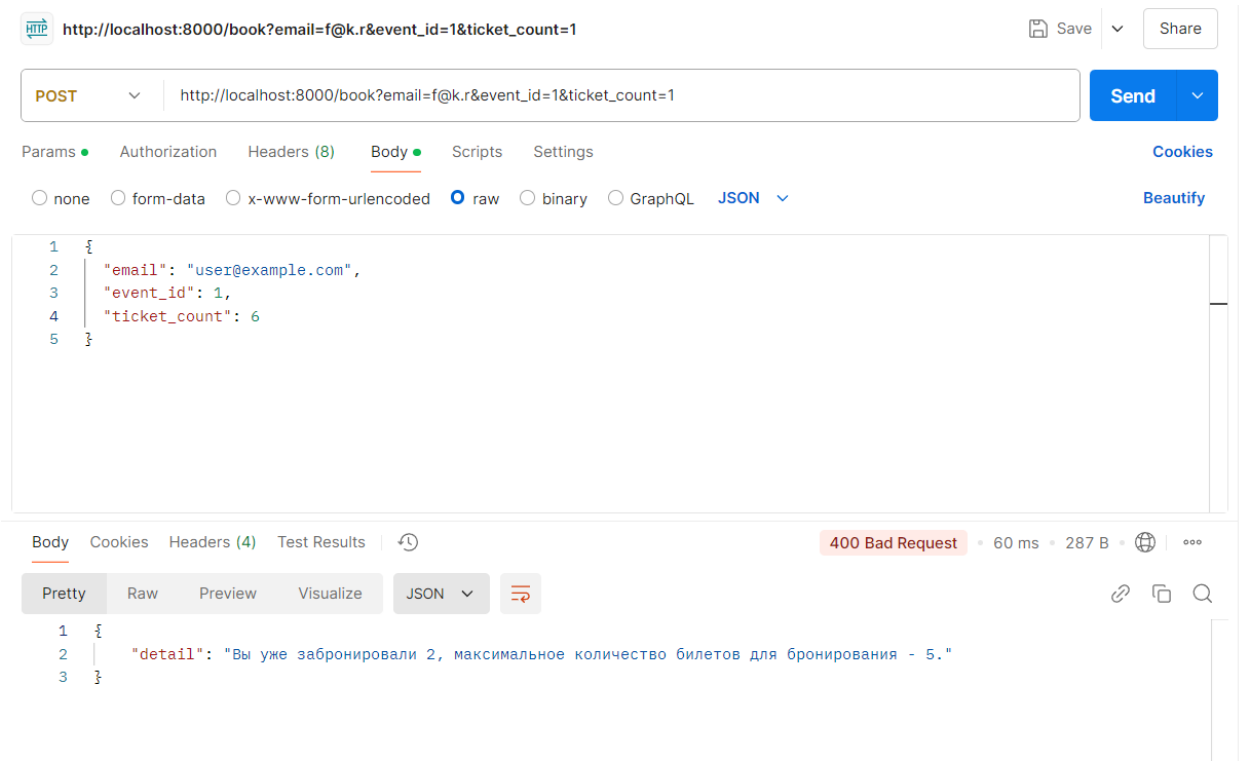

Рисунок 3.10 – POST запрос повторной регистрации на мероприятие

Как мы видим, мы не смогли повторно туда зарегистрироваться, значит все работает отлично. Протестируем функцию поиска билетов по email, рисунок 3.11.

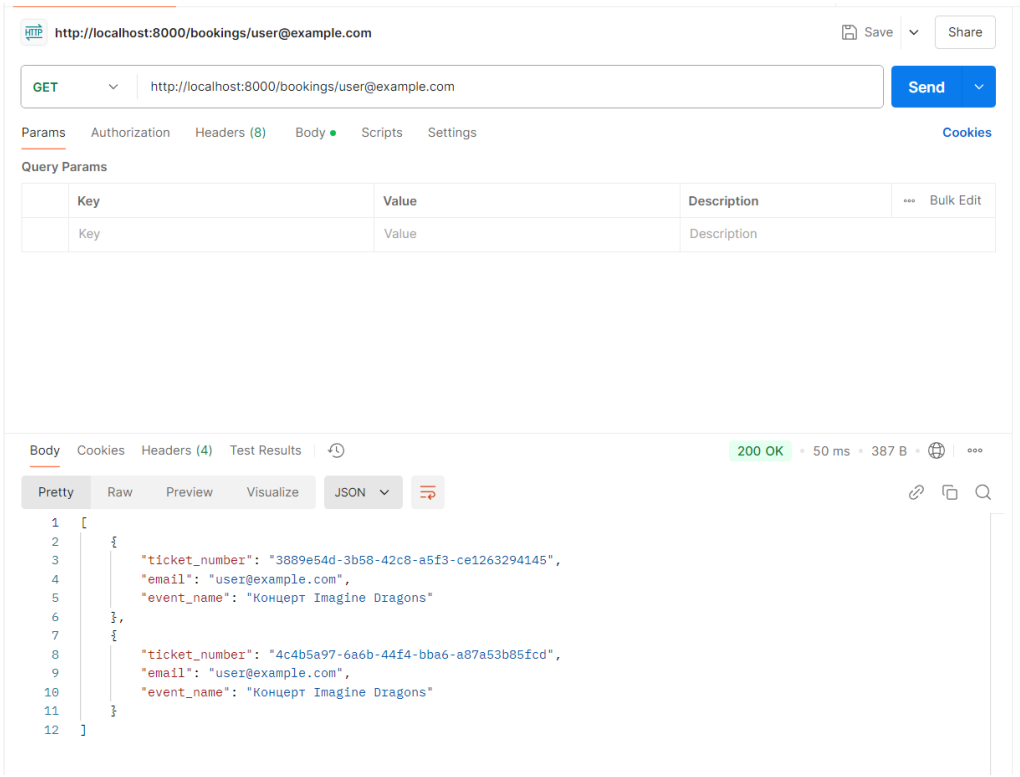

Рисунок 3.11– POST запрос регистрации на мероприятие без билетов

Все конечные точки протестированы и успешно пройдены.

3.4 Контейнеризация

Чтобы развернуть разработанную систему необходимо прописать 2 файла Dockerfile и docker-compose, представленные на рисунках 3.12-3.14.

```
FROM python:3.11-slim

WORKDIR /app

COPY backend/requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Рисунок 3.12 – Файл Dockerfile для baceknd

```dockerfile
FROM node:20-alpine

WORKDIR /app

COPY frontend/package.json frontend/package-lock.json ./

RUN npm install

COPY . .

RUN npm run build

RUN npm install -g serve

EXPOSE 3000

CMD ["serve", "-s", "build", "-l", "3000"]
```

Рисунок 3.13 – Файл Dockerfile для frontend

```yaml
version: '3.8'

Run All Services
services:
  Run Service
  backend:
    build:
      context: .
      dockerfile: backend/Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - ./backend:/app
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://postgres:postgres@db:5432/postgres

  Run Service
  frontend:
    build:
      context: .
      dockerfile: frontend/Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
    depends_on:
      - backend

  Run Service
  db:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

volumes:
  postgres_data:
```

Рисунок 3.14 – Файл docker-compose

ЗАКЛЮЧЕНИЕ

В процессе разработки клиент-серверного приложения для записи пользователей на мероприятия был проведен анализ существующих решений, архитектурных стилей и паттернов проектирования, а также детальное изучение предметной области. На основе полученных данных было разработано приложение с монолитной архитектурой, что позволило ускорить реализацию и упростить разработку. Данный подход оказался наиболее эффективным для приложения с ограниченным функционалом и предполагаемой нагрузкой, обеспечив гибкость и минимизацию сроков разработки.

В результате работы были достигнуты все поставленные цели и задачи: определены ключевые требования и ограничения системы, проанализированы инструменты и средства разработки. Итогом стало разработанное и протестированное приложение, которое демонстрирует стабильную работу, обладает интуитивно понятным интерфейсом и предоставляет пользователям необходимый функционал для записи на мероприятия.

Для ознакомления с разработанным приложением можно обратиться к исходному коду, который расположен на сайте GitHub: https://github.com/Shumila71/ticketBooking.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Python 3.13.3 documentation [Электронный ресурс]. – URL: https://docs.python.org/3 (дата обращения 01.05.2025).

2. Fast API documentation [Электронный ресурс]. – URL: https://fastapi.tiangolo.com/ (дата обращения 01.05.2025).

3. PostgreSQL [Электронный ресурс]. – URL: https://www.postgresql.org/ (дата обращения 14.10.2024).

4. Яндекс афиша [Электронный ресурс]. – URL: https://afisha.yandex.ru/moscow/ (дата обращения 01.05.2025).

5. Ticketland [Электронный ресурс]. – URL: https://www.ticketland.ru/ (дата обращения 01.05.2025).

6. MTC live [Электронный ресурс]. – URL: https://live.mts.ru/moscow/collections/concerts (дата обращения 01.05.2025).

7. UML-диаграммы [Электронный ресурс]. – URL: https://www.uml-diagrams.org/ (дата обращения 01.05.2025).

8. Draw.io [Электронный ресурс]. – URL: https://app.diagrams.net/ (дата обращения 01.05.2025).

9. VS Code [Электронный ресурс]. – URL: https://code.visualstudio.com/ (дата обращения 01.05.2025).

10. The only Python IDE you need [Электронный ресурс]. – URL: https://www.jetbrains.com/pycharm (дата обращения 01.05.2025).

11. MongoDB [Электронный ресурс]. – URL: https://www.mongodbmanager.com/ (дата обращения 01.05.2025).

12. MySQL [Электронный ресурс]. – URL: https://www.mysql.com/ (дата обращения 01.05.2025).

13. Render Quickstarts [Электронный ресурс]. – URL: https://docs.render.com/ (дата обращения 03.05.2025).

14. React [Электронный ресурс]. – URL: https://ru.react.js.org/ (дата обращения 02.05.2025).