МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Приложение для подбора и продажи билетов на развлекательные мероприятия

Студент _____ Евсеев Александр Владимирович
(подпись, дата) (ФИО полностью)

Руководитель _____ Кольчугина Е.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«20» декабря 2024 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент _Евсеев Александр Владимирович _____гр.21ВИ1 факультета __ФВТ____
<Фамилия, имя, отчество полностью>

2.Тема работы __Приложение для подбора и продажи билетов на развлекательные ___ мероприятия_____

Тема утверждена приказом ПГУ №_____от «_____»_____20_____

3. Руководитель работы _д.т.н., профессор кафедры «МО и ПЭВМ» Кольчугина Е.А._

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать веб-приложение для подбора и продажи билетов на развлекательные мероприятия.
2) Среда функционирования: web-браузер под управлением ОС Windows 10 и больше.
3) Среда разработки: Windows 10.
Языковые средства разработки: C#, JavaScript, SQL, UML.
Инструментальные средства: Visual Studio 2022, Visual Studio Code, Microsoft SQL Server, Visual Paradigm, npm, Swagger, React, Microsoft Entity Framework Core, Rabbit MQ, BCrypt.NET, Xunit, HTML, CSS, Bootstrap, Visual Paradigm 17.2, Visual Paradigm Online.
Технологии разработки: микросервисная архитектура, ООП.
4) Основные функции приложения:
а) регистрация и авторизация;
б) покупка и бронь билетов;
в) отмена покупки или брони билета;
г) поиск мероприятий;
д) добавление и редактирование мероприятий;
е) изменение имени пользователя и пароля.
5) Особенности: функция добавления мест на мероприятие с использованием блоков, названия которых можно ввести; функция массового редактирования мест.

6)Требования к СУБД и модели данных: для каждого микросервиса должна быть реализована своя независимая база данных, если она нужна исходя из его логики.
7)Провести тестирование с использованием браузера Google Chrome.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи;
2) Проектирование программных средств;
3) Реализация программных средств;
4) Тестирование программных средств;
5) Оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 199 листов, 39 рисунков, 22 таблицы и 20 использованных источников.

ПОДБОР И ПРОДАЖА БИЛЕТОВ, WEB-ПРИЛОЖЕНИЕ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, C#, JS, REACT, БРОКЕР СООБЩЕНИЙ, API, UML, RABBIT MQ, SQL SERVER, БАЗА ДАННЫХ, SQL.

Объект разработки – веб-приложение.

Предмет разработки – приложение для подбора и продажи билетов на развлекательные мероприятия.

Цель выпускной квалификационной работы – спроектировать и реализовать приложение для подбора и продажи билетов на развлекательные мероприятия.

Технологии разработки – разработка базы данных проводилась в СУБД Microsoft SQL Server. Серверная часть приложения разрабатывалась на объектно-ориентированном языке C#. Среда разработки серверной части – платформа Visual Studio 2022. Клиентская часть разрабатывалась на скриптовом языке JavaScript с использованием библиотеки React. Среда разработки – редактор кода Visual Studio Code.

Среда исполнения – веб-браузеры настольных и мобильных устройств.

Результаты работы – в ходе выполнения выпускной квалификационной работы были спроектированы и разработаны серверная часть, клиентская часть и база данных приложения для подбора и продажи билетов на развлекательные мероприятия.

Содержание

Введение 7

1 Постановка задачи и анализ требований. Методы и средства решения задачи 8

1.1 Анализ требований к поставленной задаче и предметной области 8

1.2 Анализ аналогов 9

1.3 Выбор методов и средств разработки 12

2 Проектирование приложения для подбора и продажи билетов на развлекательные мероприятия 16

2.1 Архитектура приложения 16

2.2 Проектирование структур данных 17

2.2.1 Концептуальная модель базы данных 17

2.2.2 Логическая модель базы данных 18

2.2.3 Проектирование классов 20

2.3 Проектирование алгоритмов 22

2.4 Проектирование интерфейсов 24

2.4.1 Проектирование API 24

2.4.2 Проектирование пользовательского интерфейса 26

3 Программная реализация приложения для подбора и продажи билетов на развлекательные мероприятия 36

3.1 Программная реализация баз данных 36

3.2 Программная реализация серверной части приложения 40

3.3 Программная реализация клиентской части приложения 43

4 Отладка и тестирование программного обеспечения 45

4.1 Обоснование выбора режима тестирования 45

4.2 Модульное тестирование 46

4.3 Тестирование черным ящиком 48
5 Контроль качества программного обеспечения 50
5.1 Обоснование выбора метрик программного кода 50
5.2 Метрики качества программного кода 50
6 Планирование разработки и оценка бюджета 53
Заключение 56
Список использованных источников 57
Приложение А (рекомендуемое) Графическая часть 59
Приложение Б (обязательное) Листинг серверной части приложения 71
Приложение В (обязательное) Листинг клиентской части приложения 120
Приложение Г (обязательное) Скрипт создания базы данных 186
Приложение Д (рекомендуемое) Листинг модульных тестов 189
Приложение Е (рекомендуемое) Результаты тестирования черным ящиком .. 196

Введение

В современном мире заинтересованность людей в развлекательных мероприятиях с каждым годом увеличивается все больше. Вместо пассивного отдыха люди ставят в приоритет интересное времяпровождение вне дома. Наиважнейшей частью культурной жизни многих людей становятся концерты, фестивали и выставки. Но зачастую поиск интересного мероприятия и информации о доступных местах, покупка или бронирование билета занимает слишком много времени и сил. Чтобы решить эту проблему, разрабатываются системы, предоставляющие возможность искать и покупать билеты на различные мероприятия в сети, не выходя из дома. Пользователи могут искать мероприятия по разнообразным категориям, ориентируясь на свои вкусы, моментально получать информацию о количестве доступных мест на мероприятии, в котором заинтересован пользователь, месте, дате и времени проведения события, очень быстро оплачивать или бронировать билеты, а также отменять покупки, снимать бронь и просматривать список всех своих билетов в одном месте.

Целью выпускной квалификационной работы является разработка приложения для подбора и продажи билетов на развлекательные мероприятия. Разработка включает в себя несколько этапов:

– постановка задачи и анализ требований;
– проектирование приложения;
– программная реализация приложения;
– тестирование и откладка приложения;
– контроль качества приложения.

Каждый из представленных этапов является важной частью разработки приложения. Успешное выполнение всех этапов предоставляет возможность создать приложение для подбора и продажи билетов на развлекательные мероприятия с большим функционалом, а также удобным, понятным и доступным пользовательским интерфейсом.

1 Постановка задачи и анализ требований. Методы и средства решения задачи

1.1 Анализ требований к поставленной задаче и предметной области

В рамках выпускной квалификационной работы необходимо разработать приложение, предоставляющее пользователям возможность удобно и эффективно искать, покупать и бронировать билеты на развлекательные мероприятия. Помимо покупки и бронирования билетов приложение должно давать возможность пользователям просматривать приобретенные билеты и взаимодействовать с ними, просматривать и редактировать личную информацию, а также включать в себя возможность добавления и управления мероприятиями для администраторов.

Для хранения информации о пользователях, мероприятиях, свободных, занятых местах и билетах необходимо использовать реляционные СУБД. Это позволит сохранять в структурированной форме всю информацию о мероприятиях, их состояниях и покупках пользователей.

Для реализации серверной части предлагается использовать микросервисную архитектуру. Данный тип архитектуры позволит разделить серверное приложение на абстрагированные сервисы, взаимодействующие друг с другом. Это поможет повысить значения отказоустойчивости и доступности приложения [1]. Серверная часть должна предоставлять возможность авторизации и аутентификации пользователя для разграничения функционала администратора и пользователя.

Для удобства использования клиентского приложения его интерфейс должен быть разбит на несколько основных компонентов:

– главная страница;
– поиск билетов;
– личный кабинет.

На главной странице необходимо отобразить новые мероприятия, разделенные по категориям. Личный кабинет должен отображать кабинет пользователя или панель администратора. Администратор может просматривать все основные страницы приложения, но не должен иметь возможность покупать или бронировать билеты.

1.2 Анализ аналогов

В процессе исследования рынка были выявлены основные системы аналогичного назначения: «Пенза афиша», «Афиша Тюмень», «RedKassa».

«Пенза афиша» – информационный портал города Пенза, позволяющий искать и приобретать билеты на различные развлекательные мероприятия. Интерфейс приложения представлен на рисунке 1.

Рисунок 1 – Пользовательский интерфейс приложения «Пенза афиша»

Основным преимуществом данного приложения является широкий функционал и большое количество фильтров для подбора мероприятий.

Основные недостатки данного приложения:

– интерфейс нагружен большим количеством компонентов;

– отсутствие личного кабинета;

– интерфейс плохо адаптирован для мобильных устройств.

Приложение «Афиша Тюмень», интерфейс которого представлен на рисунке 2, является аналогичным предыдущему решению информационным порталом для подбора и продажи билетов.

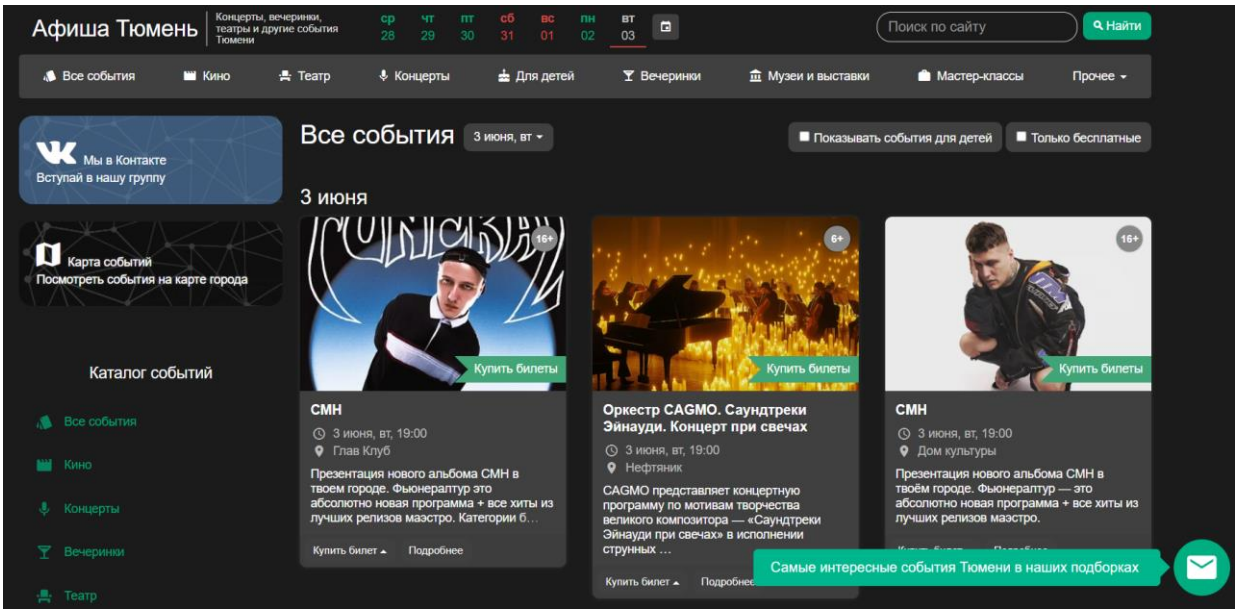

Рисунок 2 – Интерфейс приложения «Афиша Тюмень»

Выявленные преимущества данного приложения:

– удобный интерфейс;

– большое количество категорий мероприятий.

Основные недостатки приложения «Афиша Тюмень»:

– отсутствие личного кабинета;

– небольшое количество фильтров и сортировок результатов поиска (допустим, отсутствует возможность отсортировать представленные мероприятия по цене).

«RedKassa» – крупное веб–приложение, позволяющее пользователям из разных городов подбирать и покупать билеты на различные развлекательные мероприятия. Интерфейс данного решения представлен на рисунке 3.

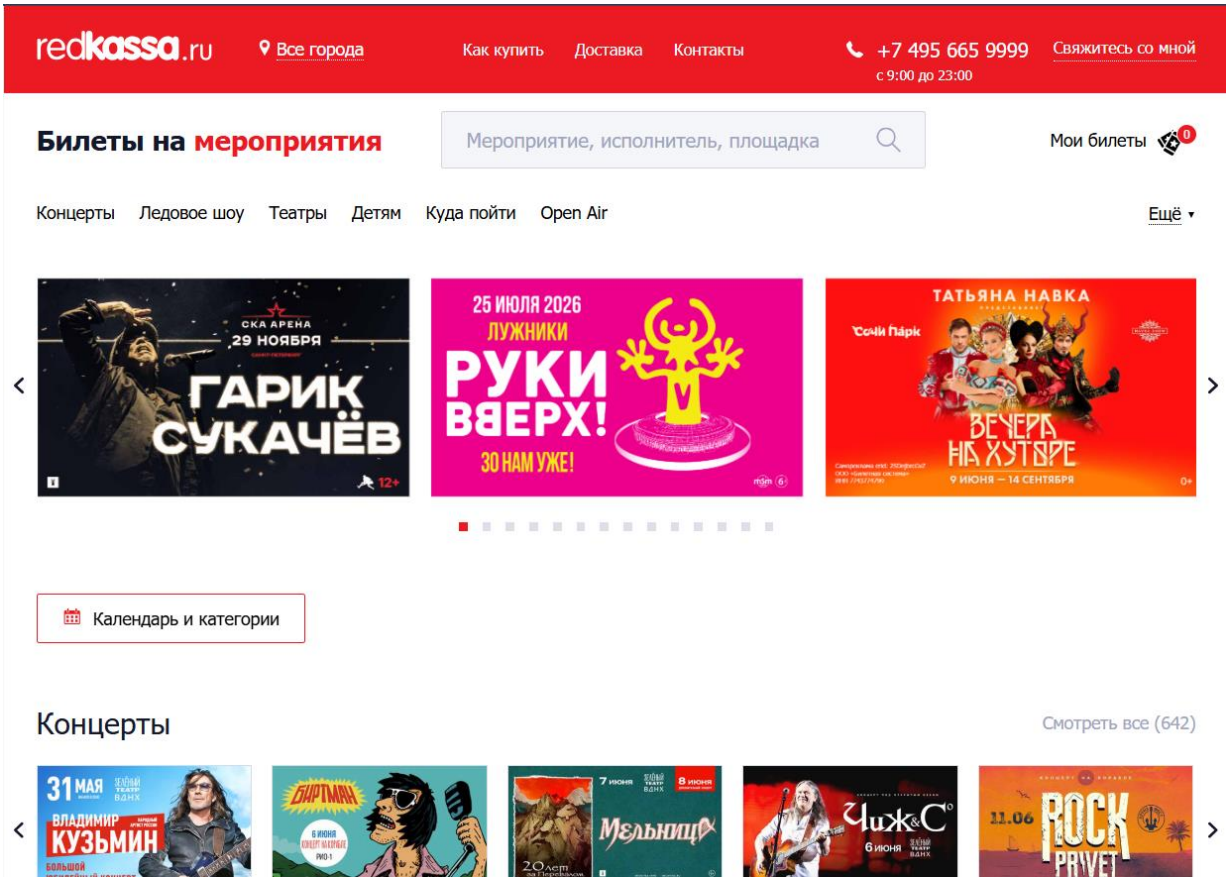

Рисунок 3 – Интерфейс приложения «RedKassa»

Преимущества «RedKassa»:

– присутствие личного кабинета и возможности управления билетами;

– быстрые отклики;

– приложение хорошо адаптировано для мобильных устройств.

Выявленными недостатками являются отсутствие сортировки мероприятий и высокая стоимость разработки, так как приложение довольно большое.

Для проведения анализа представленных систем аналогичного назначения было сформулировано пять критериев, по которым будет проводиться сравнение. Данные критерии будут оцениваться следующим образом:

0 баллов – критерий не реализован;

1 балл – критерий реализован частично;

2 балла – критерий полностью реализован.

Результаты сравнения представлены в таблице 1.

Таблица 1 – Результаты сравнения аналогов

	Удобство пользовательского интерфейса	Адаптивность пользовательского интерфейса	Доступность функций	Производительность системы	Низкая стоимость разработки и сопровождения	Итого
Разрабатываемое приложение	2	1	2	2	2	9
Пенза афиша	1	1	2	2	1	7
Афиша Тюмень	2	2	1	2	1	8
RedKassa	2	2	2	2	0	8

Таким образом, можно сделать вывод, что разрабатываемое приложение соответствует всем критериям. Основным его преимуществом является низкая стоимость разработки и сопровождения.

1.3 Выбор методов и средств разработки

Основываясь на проанализированных требованиях были сформулированы основные варианты использования, описывающие взаимодействие приложения и пользователей.

Ниже предоставлены спецификации наиболее важных вариантов использования.

На рисунке 4 представлена спецификация варианта использования «Войти в систему».

Название	Войти в систему
ID	1
Описание	Пользователь входит в систему
Акторы	Неавторизированный/незарегистрированный пользователь
Предусловие	1. Пользователь вошел в приложение 2. Пользователь не авторизирован
Основной поток	1. Пользователь переходит на страницу личного кабинета или нажимает на “Купить билет” / “Забронировать билет” 2. Система открывает форму для входа 3. Пользователь вводит логин 4. Пользователь вводит пароль 5. Пользователь входит в систему
Альтернативный поток	3а. Пользователь переходит на форму для регистрации 4а. Пользователь заполняет форму регистрации (имя, номер телефона, email, пароль) 5а. Система отображает сообщение об успешной регистрации 6а. Пользователь попадает на форму входа 7а. Пользователь вводит логин 8а. Пользователь вводит пароль 9а. Пользователь входит в систему
Постусловие	1. Пользователь получает доступ к личному кабинету 2. Пользователь получает возможность управления билетами

Рисунок 4 – Спецификация варианта использования «Войти в систему»

На рисунке 5 представлена спецификация варианта использования «Добавить мероприятие».

Название	Добавить мероприятие
ID	13
Описание	Администратор добавляет мероприятия
Акторы	Администратор
Предусловие	3. Администратор вошел в приложение 4. Администратор зашел на страницу личного кабинета
Основной поток	1. Администратор переходит на вкладку “Добавить мероприятие” 2. Система открывает форму для добавления мероприятия 3. Администратор заполняет форму для добавления мероприятия 4. Администратор нажимает на кнопку “Создать”
Постусловие	1. Система выводит уведомление об успешном добавлении мероприятия

Рисунок 5 – Спецификация варианта использования «Добавить мероприятие»

На рисунке 6 представлена спецификация варианта использования «Купить билет на мероприятие».

Название	Купить билет на мероприятие
ID	19
Описание	Пользователь покупает билет на мероприятие
Акторы	Авторизированный пользователь
Предусловие	1. Пользователь находится на странице выбор места на мероприятие 2. Пользователь выбрал свободное место
Основной поток	1. Пользователь нажимает на кнопку “Купить билет” 2. Система отображает модальное окно с сообщением о том, что платежный сервис еще не интегрирован 3. Пользователь закрывает окно
Постусловие	1. Система выводит уведомление об успешной покупке билета

Рисунок 6 – Спецификация варианта использования «Купить билет на мероприятие»

Для наглядной демонстрации всех вариантов использования, описывающих функциональные требования приложения, и групп пользователей, к которым они относятся, была разработана диаграмма вариантов использования. На данной диаграмме отображены основные типы пользователей системы, а также приведены относящиеся к ним варианты использования [2]. Данная диаграмма представлена на рисунке 7.

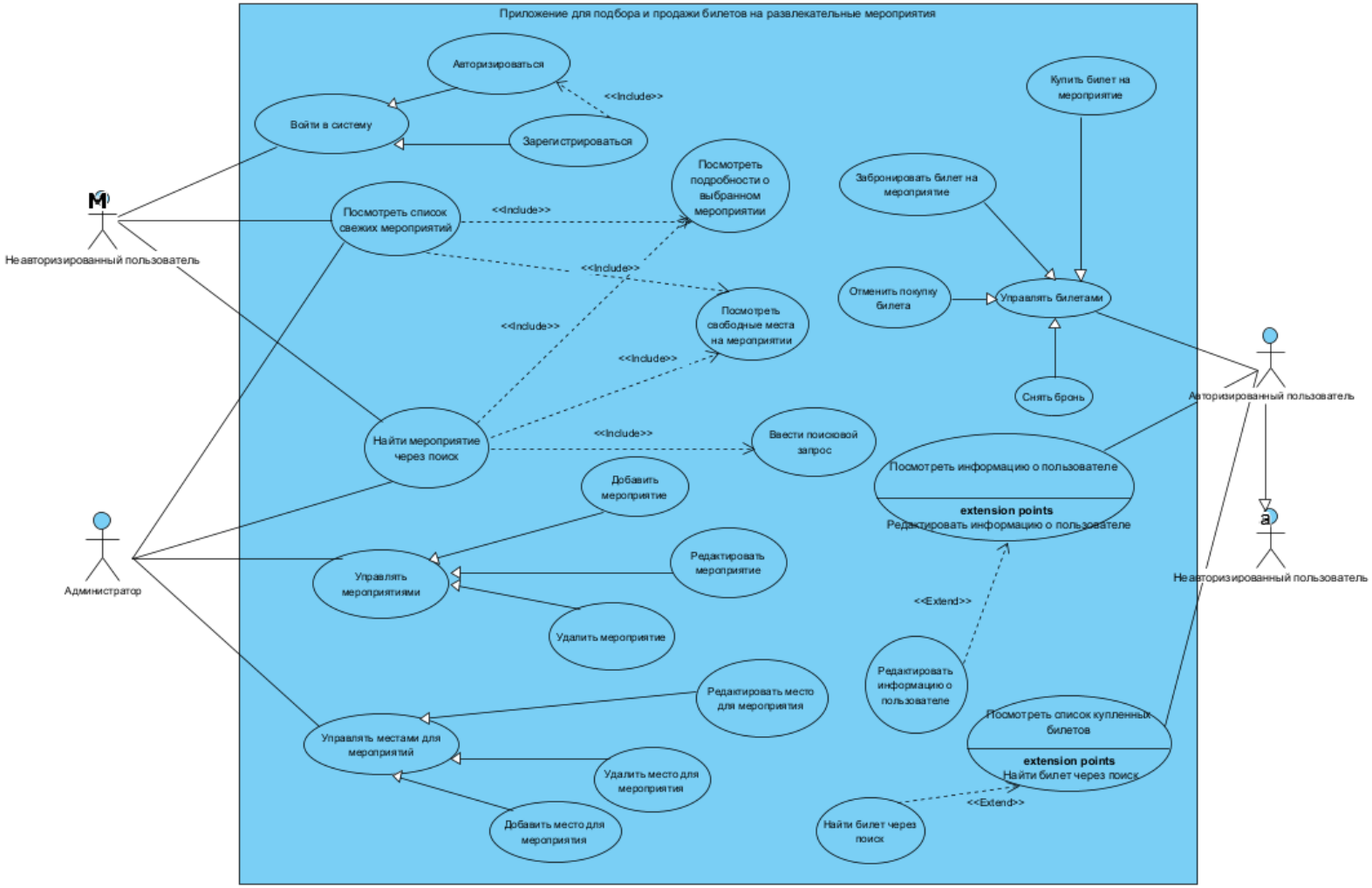

Рисунок 7 – Диаграмма вариантов использования

Для реализации серверной части используется платформа ASP.NET Core и язык C#. Данные технологии обеспечивают высокую производительность и позволяют использовать объектно-ориентированное программирование, что может значимо упростить написание кода. Так же платформа ASP.NET Core удобна для взаимодействия с клиентской частью через REST API [3].

Серверная часть разделена на три микросервиса: «Пользователь», «Мероприятие», «Билет». Базы данных необходимы для каждого сервиса. Для их создания используется язык работы с реляционными базами данных SQL и реляционная СУБД Microsoft SQL server. Данная СУБД обладает высокими показателями производительности и масштабируемости, а также удобным интерфейсом SQL Server Management Studio [4].

Для взаимодействия микросервисов используется брокер сообщений Rabbit MQ. Чтобы отправить сообщение, микросервис добавляет его в обменник, который сам распределяет сообщения по очередям. Другие микросервисы подписываются на очереди и при получении сообщения отправляют подтверждение. Данный брокер сообщений позволяет управлять очередями, способствует увеличению производительности за счет асинхронного обмена сообщениями и обладает понятным пользовательским интерфейсом [5].

Для взаимодействия серверной части и СУБД используется Microsoft Entity Framework Core. Данный фреймворк упрощает взаимодействие с базой данных, написание запросов и управление таблицами [6].

Для разработки клиентской части используется язык JavaScript и библиотека React, позволяющая разделить интерфейс на компоненты и сделать его удобным и динамичным [7]. Для разработки пользовательского интерфейса используется фреймворк Bootstrap, содержащий в себя множество CSS и HTML компонентов.

Таким образом, в результате анализа требований была реализована диаграмма вариантов использования, были описаны основные варианты использования, определяющие взаимодействие пользователя и приложения, и основные технологии, использующиеся в нем.

2 Проектирование приложения для подбора и продажи билетов на развлекательные мероприятия

2.1 Архитектура приложения

Согласно представленным требованиям, была спроектирована диаграмма компонентов, отображающая архитектуру приложения для подбора и продажи билетов на развлекательные мероприятия и взаимодействие его компонентов во время процесса входа пользователя в систему. Данная диаграмма представлена на рисунке 8.

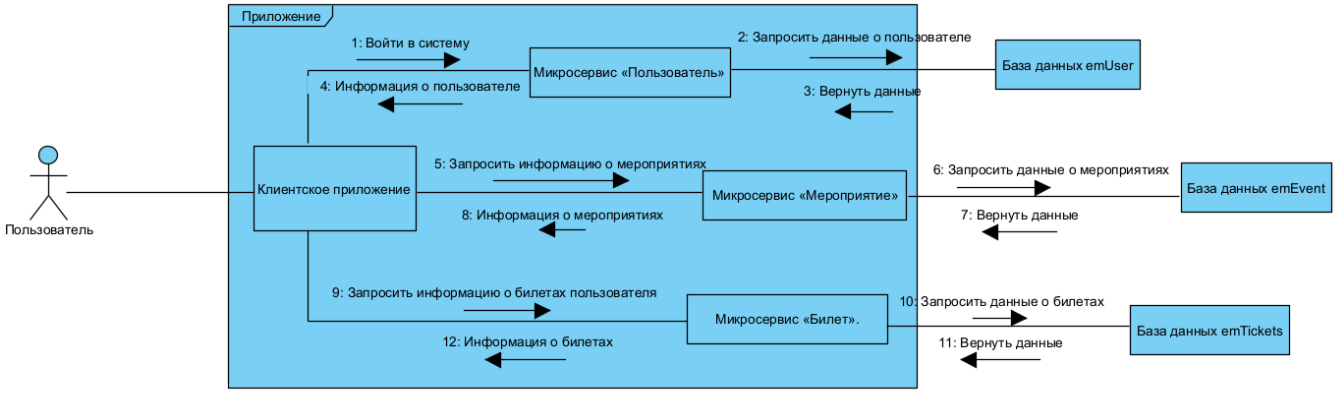

Рисунок 8 – Диаграмма коммуникаций

Клиентское приложение взаимодействует с серверным через HTTP–запросы.

Серверное приложение разделено на три автономных микросервиса. Клиентское приложение взаимодействует напрямую с каждым микросервисом. Внутри каждого микросервиса реализованы модели для представления данных, а также контроллеры, отвечающие за обработку сообщений с клиентской части. Во время процессов покупки и бронирования билетов микросервисы взаимодействуют между собой с помощью брокера сообщений Rabbit MQ. Это нужно для совместного управления данными и отсутствия разногласий в базах данных.

У каждого микросервиса есть своя база данных. Для взаимодействия микросервисов с базами данных используется фреймворк Entity Framework Core.

Таким образом, в соответствии с требованиями, была спроектирована архитектура приложения для подбора и продажи билетов на развлекательные мероприятия.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Исходя из представленных требований, были выявлены основные сущности предметной области разрабатываемого приложения. Для отображения взаимосвязей между ними, их атрибутов и структуры данных была реализована концептуальная модель базы данных [8]. Данная диаграмма отображена на рисунке 9.

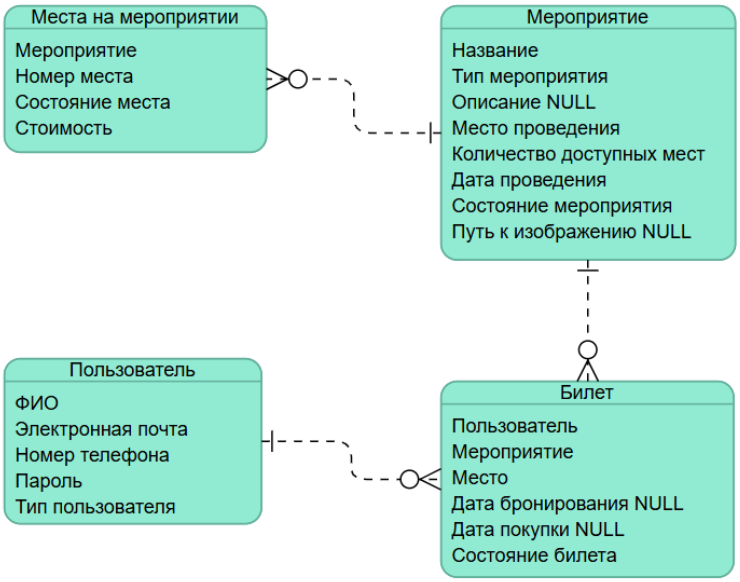

Рисунок 9 – Концептуальная модель базы данных

В таблице 2 содержится описание представленных сущностей.

Таблица 2 – Описание сущностей

Наименование сущности	Идентификатор сущности	Описание сущности
Мероприятие	Event	Содержит информацию о мероприятии, его описание, характеристики и состояние

Продолжение таблицы 2

Наименование сущности	Идентификатор сущности	Описание сущности
Места на мероприятии	EventSeats	Содержит информацию о местах для посетителей мероприятия, их стоимости и состоянии
Пользователь	User	Содержит информацию о пользователе
Билет	Ticket	Содержит информацию о оплаченных или забронированных пользователями мест на мероприятия, даты покупки, бронирования и состояние билета

Таким образом, данная модель в полной мере отображает выявленные сущности, взаимосвязи между ними, их атрибуты и структуру.

2.2.2 Логическая модель базы данных

На основе концептуальной модели базы данных была спроектирована логическая модель, которая уточняет связи и ограничения данных.

В процессе проектирования было принято решение о разделении модели данных на три отдельные модели: «User», «Event», «Tickets». Такое разделение было проведено, так как этого требует выбранная архитектура: каждый микросервис должен быть независимым компонентом, из этого следует, что у каждого должна быть своя база данных.

На рисунке 10 представлена логическая модель базы данных «emUser».

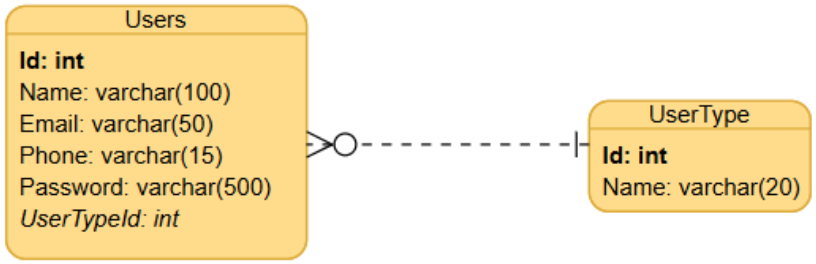

Рисунок 10 – Логическая модель базы данных «emUser»

Данная модель описывает структуру данных микросервиса «Пользователь».

На рисунке 11 представлена логическая модель базы данных «emEvent».

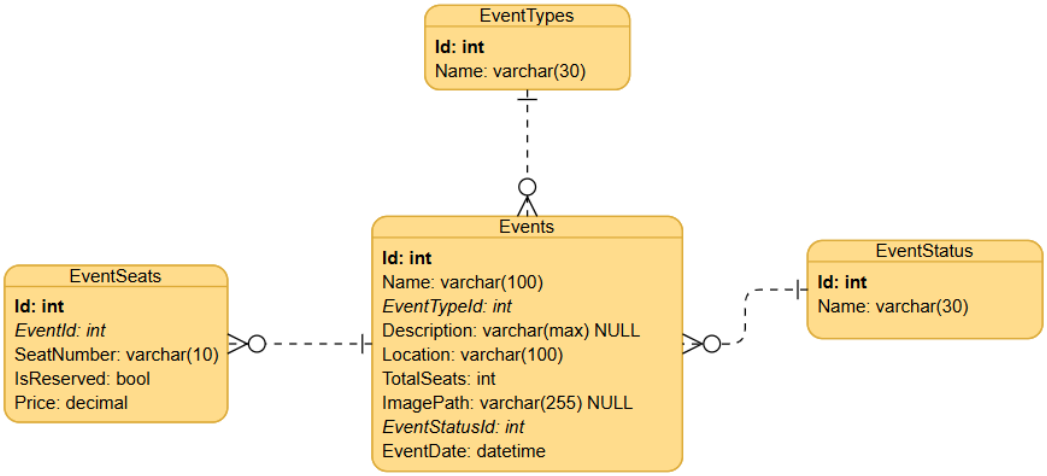

Рисунок 11 – Логическая модель базы данных «emEvent»

Данная модель описывает структуру данных микросервиса «Мероприятие».

На рисунке 12 представлена логическая модель базы данных «emTickets».

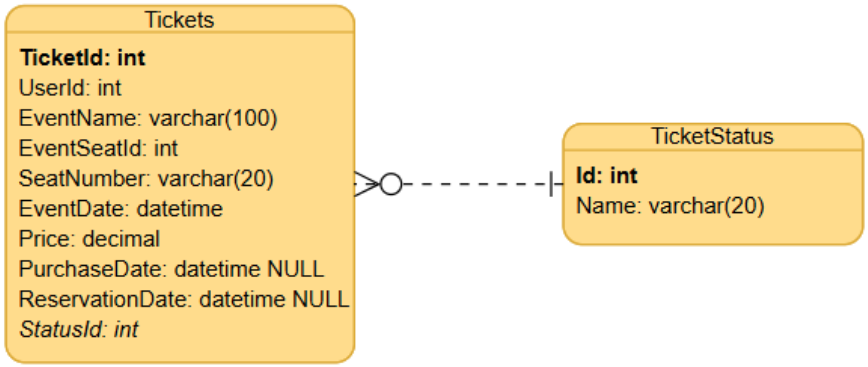

Рисунок 12 – Логическая модель базы данных «emTickets»

Данная модель описывает структуру данных микросервиса «Билет».

Представленные логические модели нормализованы до 3 нормальной формы и в полной мере отображают структуру данных приложения.

2.2.3 Проектирование классов

Для каждого из выделенных микросервисов была разработана диаграмма классов в нотации UML, описывающая структуру серверной части приложения: интерфейсы, классы, их атрибуты и связи между ними, типы данных, а также методы и их аргументы [9].

Описание спроектированных классов микросервиса «Мероприятие», в котором реализована логика добавления, удаления и управления мероприятиями, приведено в таблице 3.

Таблица 3 – Описание спроектированных классов

Название	Описание
EventsDbContext	Класс, определяющий сущности, связанные с базой данных, а также управляющий запросами и подключением к базе данных.
IEventRepository	Интерфейс, определяющий, какие методы должны быть реализованы для работы с базой данных.
EventRepository	Класс, реализующий интерфейс «IEventRepository». Содержит в себе методы, отвечающие за доступ и работу с базой данных.
IEventService	Интерфейс, определяющий, какие методы должны быть реализованы для выполнения бизнес–логики
EventService	Класс, реализующий интерфейс «IEventService». Содержит в себе методы, необходимые для выполнения бизнес–логики.
RabbitMQConsumerService	Класс, предназначенный для получения сообщений из очереди брокера «Rabbit MQ», передачи их в обработку и отправке ответов.
EventsController	Класс, предназначенный для обработки HTTP запросов.

На рисунке 13 представлена диаграмма классов микросервиса «Мероприятие»

Диаграммы классов микросервисов Билет и Пользователь представлены в приложении А на рисунках А.1 и А.2 соответственно.

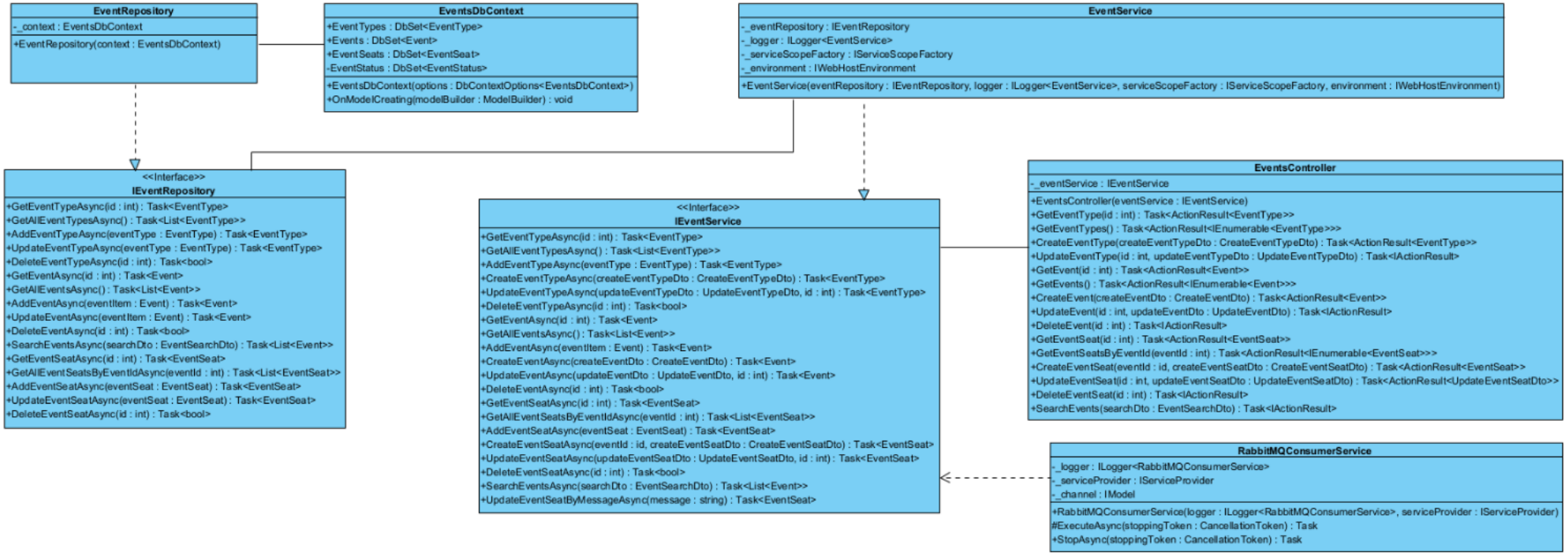

Рисунок 13 – Диаграмма классов микросервиса «Мероприятие»

Таким образом, спроектированные классы реализуют необходимую бизнес–логику, взаимодействие с базой данных и с серверным приложением.

2.3 Проектирование алгоритмов

В процессе проектирования алгоритмов была разработана диаграмма деятельности в нотации UML, отображающая общую логику покупки билета и алгоритм работы приложения во время выполнения данного процесса [10]. Диаграмма представлена на рисунке 14.

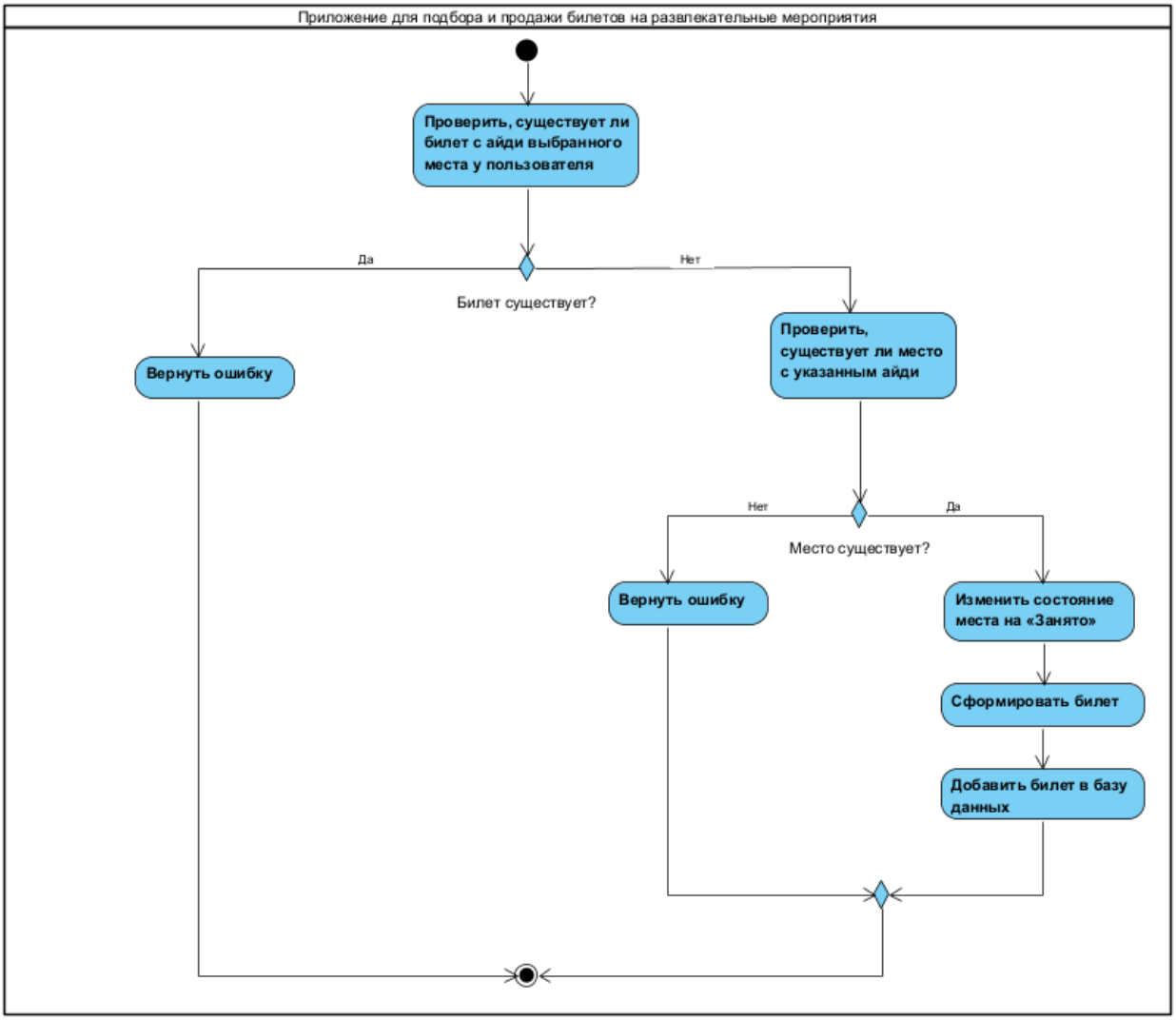

Рисунок 14 – Диаграмма деятельности алгоритма покупки билета

На диаграмме изображены удачный и неудачный исход процесса покупки билета. В результате неудачных исходов система должна вернуть ошибку. В случае удачи формируется билет и добавляется в базу данных.

Для более детального отображения алгоритма была спроектирована диаграмма последовательности, визуализирующая взаимодействие компонентов приложения, их жизненный цикл и порядок сообщений, которыми они обмениваются друг с другом [11]. Диаграмма представлена на рисунке 15.

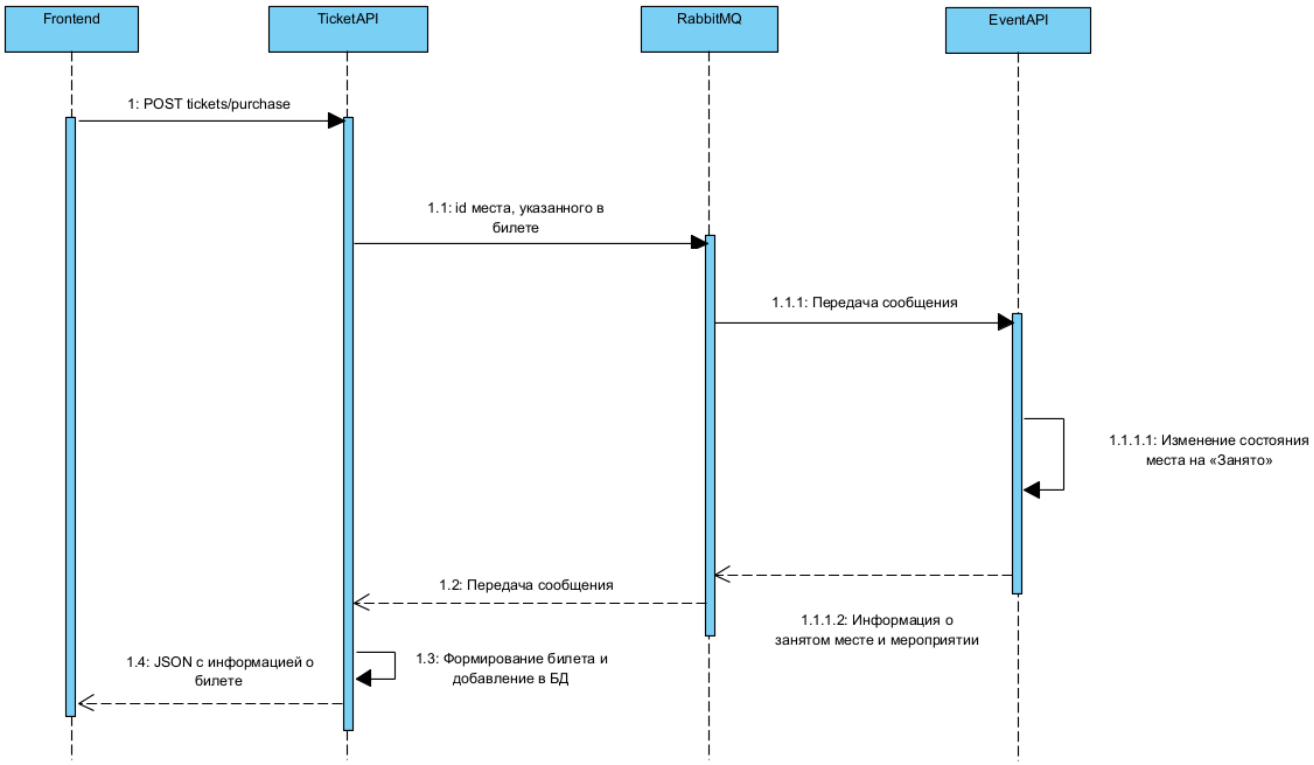

Рисунок 15 – Диаграмма последовательности

На диаграмме описывается взаимодействие клиентской и серверной частей: отправка запроса с клиента, его обработка в микросервисе «Билет», а также взаимодействие микросервисов через брокер сообщений RabbitMQ.

С клиентской части подается HTTP-запрос. Его обрабатывает микросервис «Билет», затем он отправляет идентификатор места, занимаемого пользователем, в микросервис «Мероприятие». В нем меняется состояние данного места, собирается информация о мероприятии и самом месте и отправляется обратно. На основе этой информации формируется и добавляется в базу данных билет. Затем информация о нем отправляется в клиентское приложение.

Таким образом, в соответствии с выявленными требованиями, был спроектирован один из ключевых алгоритмов работы приложения.

2.4 Проектирование интерфейсов

2.4.1 Проектирование API

Серверное приложение разбито на три микросервиса: «Пользователь», «Мероприятие», «Билет». Взаимодействие клиентской и серверной частей осуществляется через HTTP–запросы.

Микросервис «Пользователь» отвечает за регистрацию и авторизацию пользователя. В таблице 4 представлена спецификация HTTP–запросов этого сервиса.

Таблица 4 – Спецификация HTTP–запросов микросервиса «Пользователь»

URL запроса	HTTP метод	Описание	Query параметры запроса
/api/users/register	POST	Регистрация пользователя	–
/api/users/login	POST	Авторизация пользователя	–
/api/users/{id}	GET	Получение информации о пользователе	ID пользователя
/api/users/{id}/name	PUT	Изменение имени пользователя	ID пользователя
/api/users/{id}/password	PUT	Изменение пароля пользователя	ID пользователя

Микросервис «Мероприятие», спецификация HTTP запросов которого представлена в таблице 5, отвечает за добавление, редактирование и удаление мероприятий и связанной с ними информации.

Таблица 5 – Спецификация HTTP–запросов микросервиса «Мероприятие»

URL запроса	HTTP метод	Описание	Query параметры запроса
/api/eventtype	GET	Получение информации о типах мероприятий	–
/api/event/{id}	GET	Получение информации о мероприятии	ID мероприятия
/api/event/{id}	PUT	Изменение информации о мероприятии	ID мероприятия

Продолжение таблицы 5

URL запроса	HTTP метод	Описание	Query параметры запроса
/api/event/{id}	DELETE	Удаление мероприятия	ID мероприятия
/api/event	GET	Получение информации о всех мероприятиях	–
/api/event	POST	Добавление нового мероприятия	–
/api/eventseat/{id}	GET	Получение информации о посетительском месте на мероприятии	ID места
/api/eventseat/{id}	PUT	Изменение информации о посетительском месте на мероприятии	ID места
/api/eventseat/{id}	DELETE	Удаление посетительского места на мероприятии	ID места
/api/{eventId}/eventseat	GET	Получение информации о всех посетительских местах на мероприятии	ID мероприятия
/api/{eventId}/eventseat	POST	Добавление посетительского места на мероприятии	ID мероприятия
/api/search	GET	Получение информации о мероприятиях через поиск	–

Микросервис «Билет» отвечает за функционал покупки и бронирования билетов. Его спецификация отображена в таблице 6.

Таблица 6 – Спецификация HTTP–запросов микросервиса «Билет»

URL запроса	HTTP метод	Описание	Query параметры запроса
/api/tickets/purchase	POST	Покупка билета	–
/api/tickets/reservation	POST	Бронирование билета	–

Продолжение таблицы 6

URL запроса	HTTP метод	Описание	Query параметры запроса
/api/tickets/cancel/purchase	POST	Отмена покупки	–
/api/tickets/cancel/reservation	POST	Отмена бронирования	–
/api/tickets/{userId}	GET	Получение информации о билетах пользователя	ID пользователя
/api/tickets/purchaseBooked	POST	Покупка забронированного билета	–

Представленные HTTP запросы определяют интерфейс, необходимый для взаимодействия клиентского и серверного приложений.

2.4.2 Проектирование пользовательского интерфейса

В результате анализа требований был спроектирован пользовательский интерфейс. Интерфейс является графическим, разделен на несколько страниц, включающих в себя разные функции приложения. Это обеспечивает его удобность и понятность.

В начале работы с приложением открывается главная страница, представленная на рисунке 16.

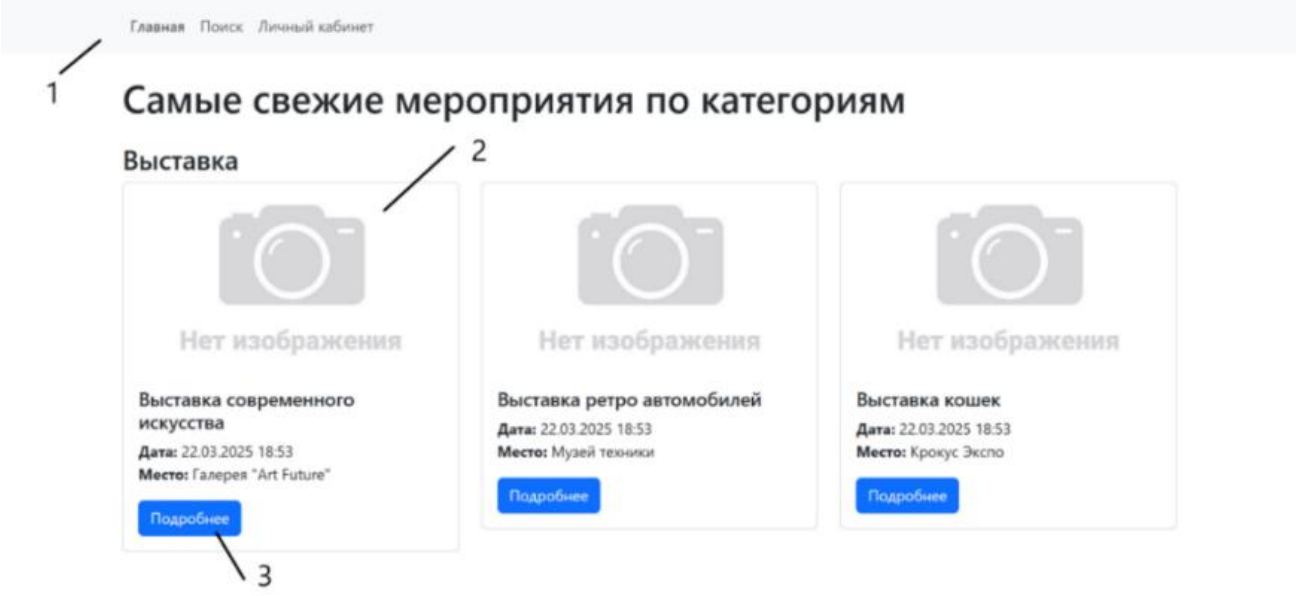

Рисунок 16 – Главная страница приложения

Основные элементы главной страницы:

1 – панель для перехода между страницами «Главная», «Поиск» и «Личный кабинет»;

2 – карточки самых новых по дате мероприятий, отображающиеся в своей категории;

3 – кнопка для перехода на страницу, содержащую информацию о мероприятии.

С главной страницы можно перейти на страницу личного кабинета. Если пользователь не авторизирован, то вместо личного кабинета откроется страница входа в систему, представленная на рисунке 17.

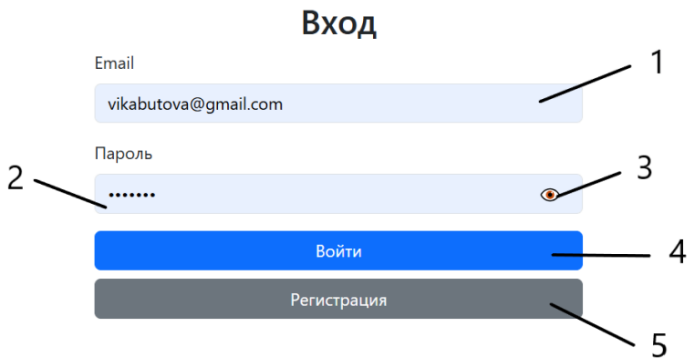

Рисунок 17 – Страница авторизации

Основные элементы данной страницы:

1 – поле для ввода электронной почты;

2 – поле для ввода пароля;

3 – кнопка для отображения или скрывания пароля;

4 – кнопка для входа в систему;

5 – кнопка для перехода на страницу регистрации.

При нажатии на кнопку «Регистрация» перед пользователем откроется страница регистрации, представленная на рисунке 18.

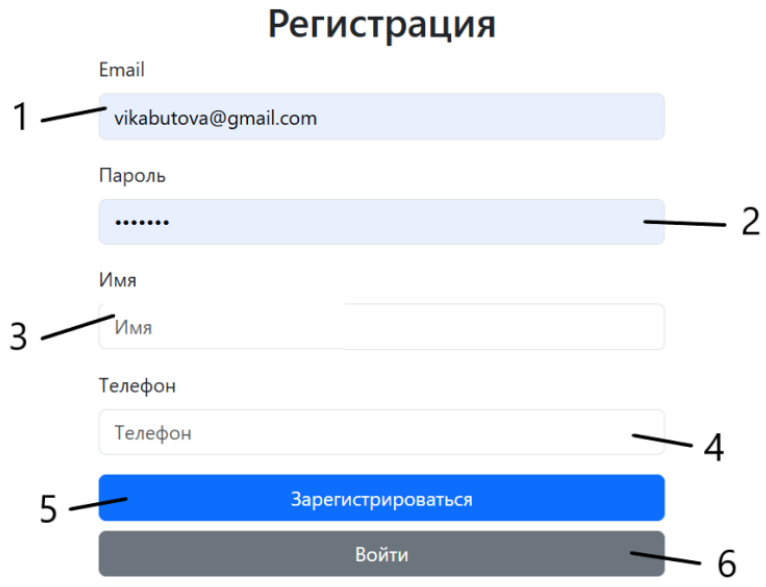

Рисунок 18 – Страница регистрации

Основные элементы страницы регистрации:

1 – поле для ввода электронной почты;

2 – поле для ввода пароля;

3 – поле для ввода имени;

4 – поле для ввода номера телефона;

5 – кнопка для отправки формы регистрации;

6 – кнопка для перехода на страницу «Вход».

После удачной авторизации пользователя открывается личный кабинет. Страница личного кабинета представлена на рисунке 19.

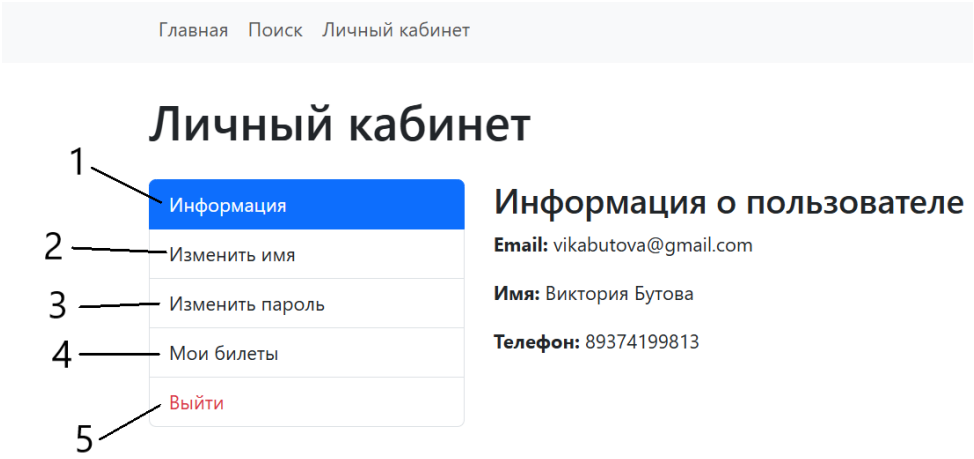

Рисунок 19 – Страница личного кабинета

Основные элементы данной страницы:

1 – кнопка для отображения личной информации;

2 – кнопка для отображения поля изменения имени;

3 – кнопка для отображения полей изменения пароля;

4 – кнопка для отображения купленных и забронированных билетов пользователя;

5 – кнопка для выхода из аккаунта пользователя;

Со страницы «Личный кабинет» через верхнюю панель можно перейти на страницы «Поиск» и «Главная». Страница поиска представлена на рисунке 20.

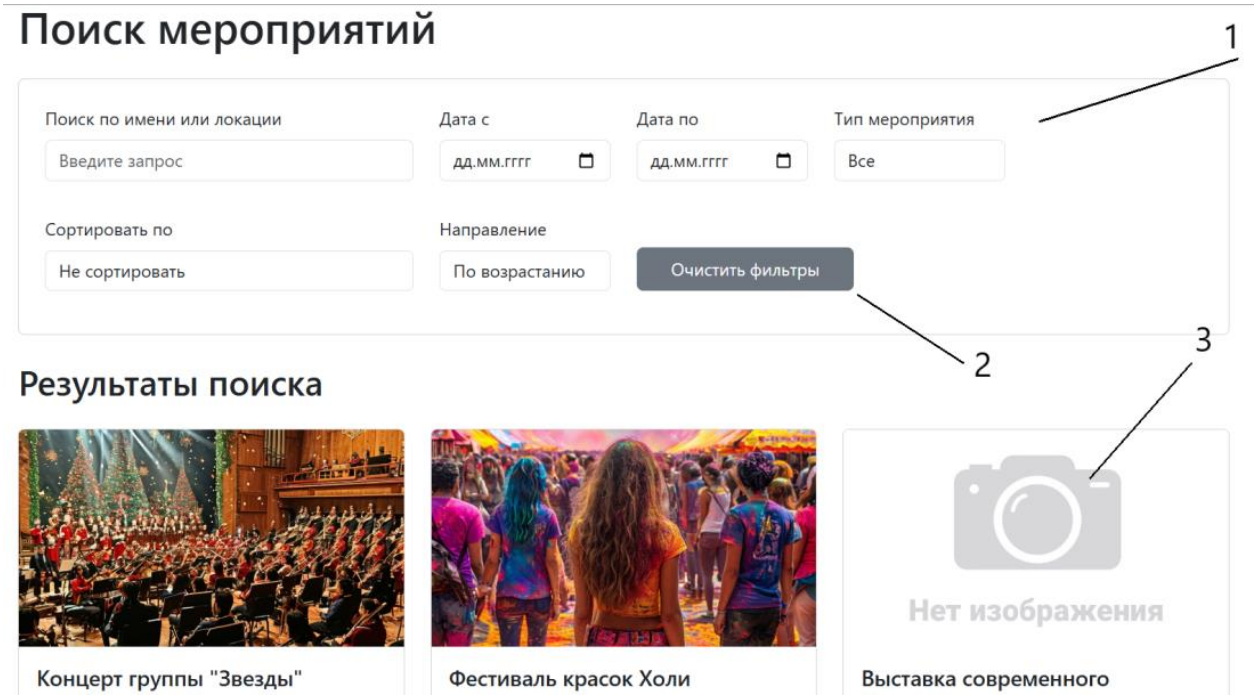

Рисунок 20 – Страница «Поиск»

Основные элементы страницы «Поиск»:

1 – фильтры и поиск мероприятий;

2 – кнопка для очистки фильтров;

3 – карточки найденных мероприятий.

При выборе мероприятия через поиск или через главную страницу открывается страница с подробностями мероприятия. Данная страница представлена на рисунке 21.

Рисунок 21 – Страница информации о мероприятии

Основные элементы страницы информации о мероприятии:

1 – информация о мероприятии;

2 – кнопка покупки билета на мероприятие (доступна, если мероприятие активно);

3 – кнопка возврата к предыдущей странице;

При нажатии на кнопку «Купить билет» откроется представленная на рисунке 22 страница покупки билета.

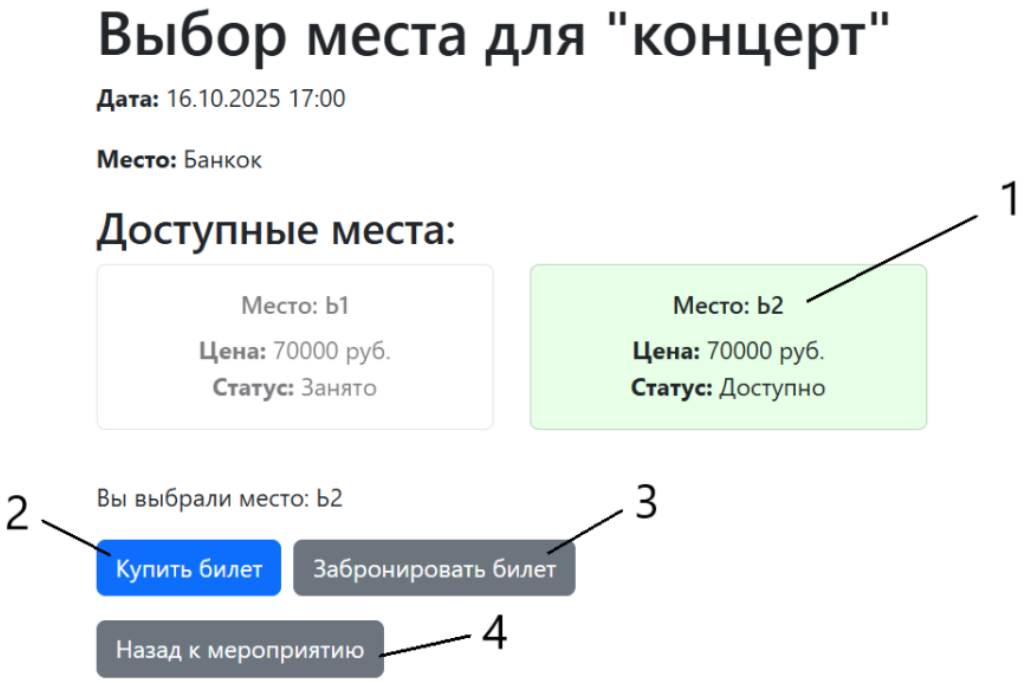

Рисунок 22 – Страница покупки билета

Основные элементы данной страницы:

1 – карточки посетительских;

2 – кнопка купить билет (появляется после выбора места);

3 – кнопка забронировать билет (появляется после выбора места);

5 – кнопка для возврата на страницу мероприятия.

После покупки или бронирования в личном кабинете пользователя должна появиться карточка мероприятия.

Также был спроектирован кабинет администратора, представленный на рисунке 23.

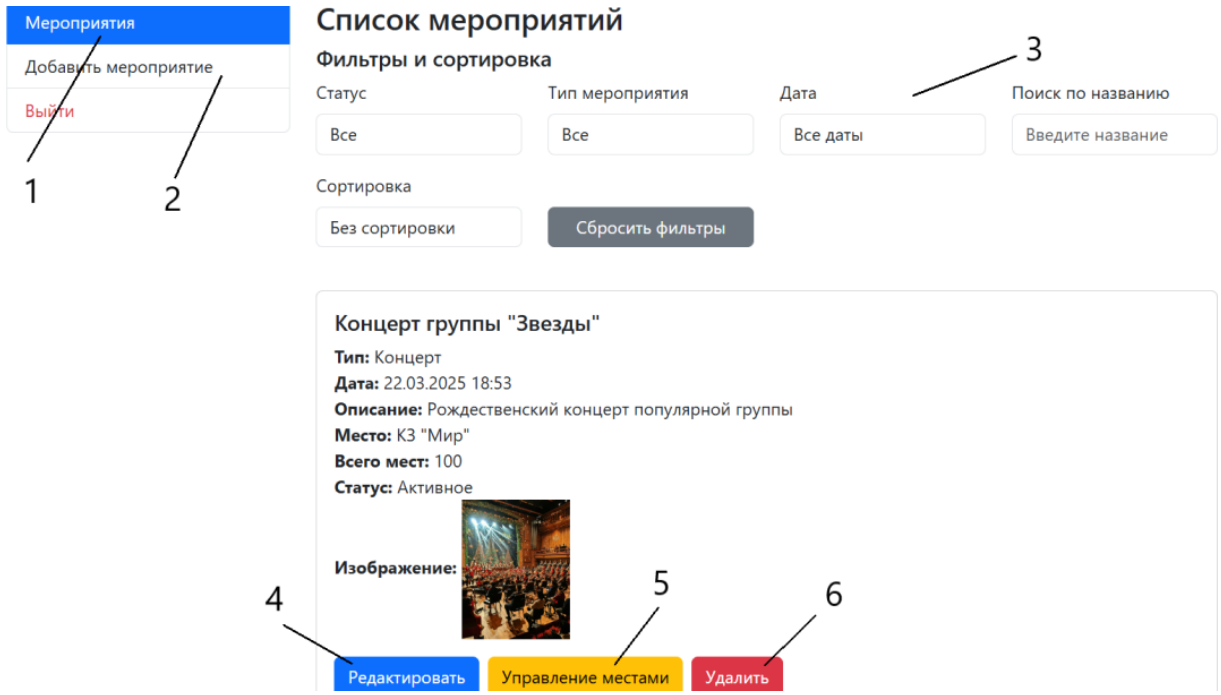

Рисунок 23 – Страница «Панель администратора»

Основные элементы страницы кабинета администратора:

1 – кнопка для переключения на страницу со списком мероприятий;

2 – кнопка для переключения на форму добавления мероприятия;

3 – фильтры и поиск мероприятий;

4 – кнопка для перехода на страницу редактирования мероприятия;

5 – кнопка для перехода на страницу управления местами;

6 – кнопка для удаления мероприятия.

При нажатии на кнопку «Управление местами» открывается страница редактирования и добавления мест. Данная страница отображена на рисунке 24.

Рисунок 24 – Страница управления местами

Основные элементы представленной страницы:

1 – чек-поинт для использования блоков (блок А с местами А1, А2 и так далее) для добавления мест;

2 – кнопка для отображения формы добавления мест;

3 – кнопка для сохранения заполненной формы добавления мест;

4 – кнопка для отмены и возврата;

5 – кнопка для отображения существующих мест.

Чек-поинт позволяет использовать блоки для обозначения мест. Пользователь может добавить блок, ввести его название и добавить в него места. Также форму для создания блока можно скрыть. Можно добавить места без блоков. Есть возможность комбинировать добавление мест разными способами, то есть совместить добавления блоками и без.

При нажатии на кнопку «Показать существующие места» отобразится дополнительная часть страницы с управлением местами. Данная часть представлена на рисунке 25.

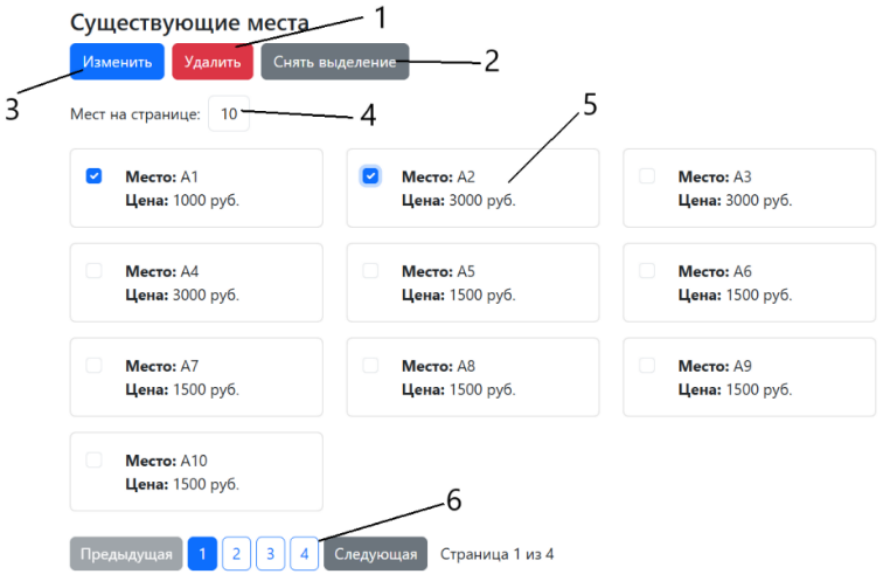

Рисунок 25 – Страница редактирования мест

Основные элементы страницы редактирования мест:

1 – кнопка для удаления выделенных мест;

2 – кнопка для снятия выделения;

3 – кнопка для изменения выделенных мест;

4 – поле для ввода количество отображаемых карточек на странице;

5 – карточки с информацией о местах;

6 – пагинация.

При нажатии на кнопку «Изменить» откроется форма массового редактирования мест. Данная форма приведена на рисунке 26.

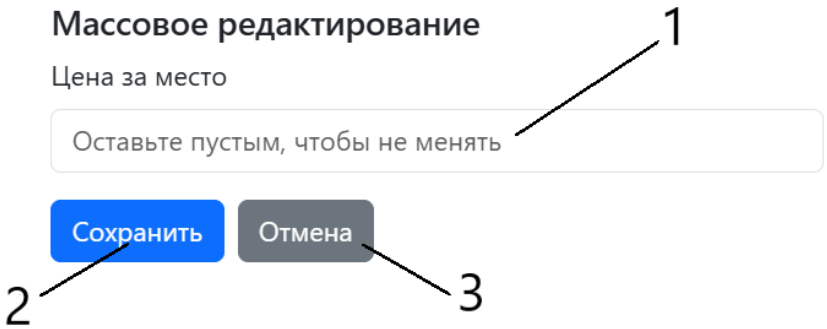

Рисунок 26 – Форма массового редактирования мест

Основные элементы данной формы:

1 – поле для ввода новой цены для выбранных мест;

2 – кнопка сохранения изменений;

3 – кнопка отмены.

Однако если нажать на отдельную карточку, откроется форма редактирования конкретного места, представленная на рисунке 27.

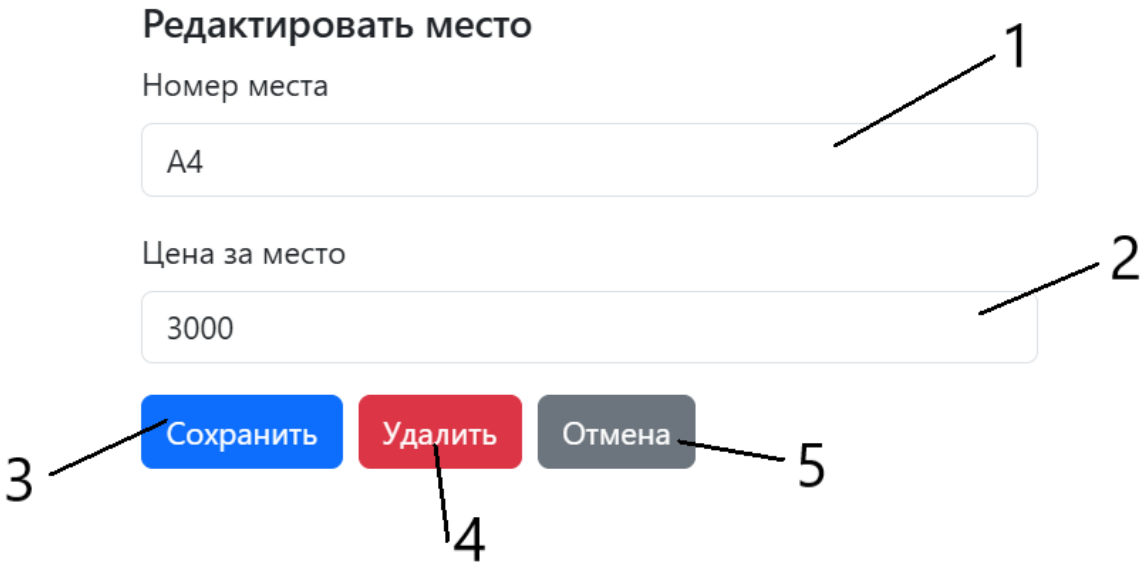

Рисунок 27 – Форма редактирования одного выбранного места

Основные элементы формы редактирования места:

1 – поле для изменения номера места;

2 – поле для изменения цены за место;

3 – кнопка сохранения изменений;

4 – кнопка удаления места;

5 – кнопка отмены изменений.

При нажатии на кнопку удалить удаляются не изменения, а место в целом.

При переходе с вкладки «Мероприятия» на вкладку «Добавить мероприятие» открывается форма добавления мероприятия, отображенная на рисунке 28.

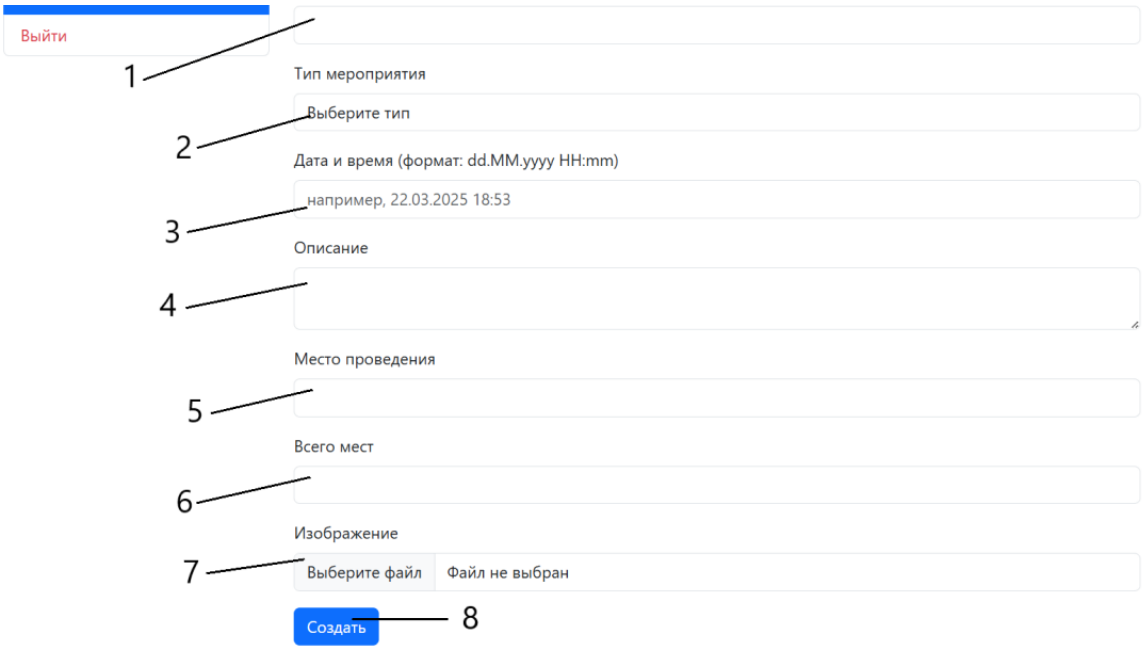

Рисунок 28 – Форма для добавления мероприятия

Основные элементы формы для добавления мероприятия

1 – поле для ввода названия мероприятия;

2 – выпадающий список с типами мероприятия;

3 – поле для ввода даты и времени мероприятия;

4 – поле для ввода описания мероприятия;

5 – поле для ввода места проведения мероприятия;

6 – поле для ввода количества посетительских мест на мероприятии;

7 – кнопка для добавления изображения;

8 – кнопка для отправки формы и создания мероприятия.

При нажатии на кнопку «Редактировать мероприятие» на вкладке «Мероприятия» откроется форма, аналогичная форме добавления мероприятия.

Таким образом, спроектированный графический интерфейс предоставляет пользователю функционал приложения в удобном и понятном виде.

3 Программная реализация приложения для подбора и продажи билетов на развлекательные мероприятия

3.1 Программная реализация баз данных

Основываясь на концептуальной и логических моделях баз данных была спроектирована физическая модель. Данная модель отражает, как именно данные будут храниться в конкретной СУБД, какие у них будут типы, связи и ограничения. Для программной реализации базы данных используется реляционная СУБД Microsoft SQL server.

В соответствии с логической моделью, физическая тоже разделена на три модели. На рисунке 29 представлена модель базы данных «emEvent».

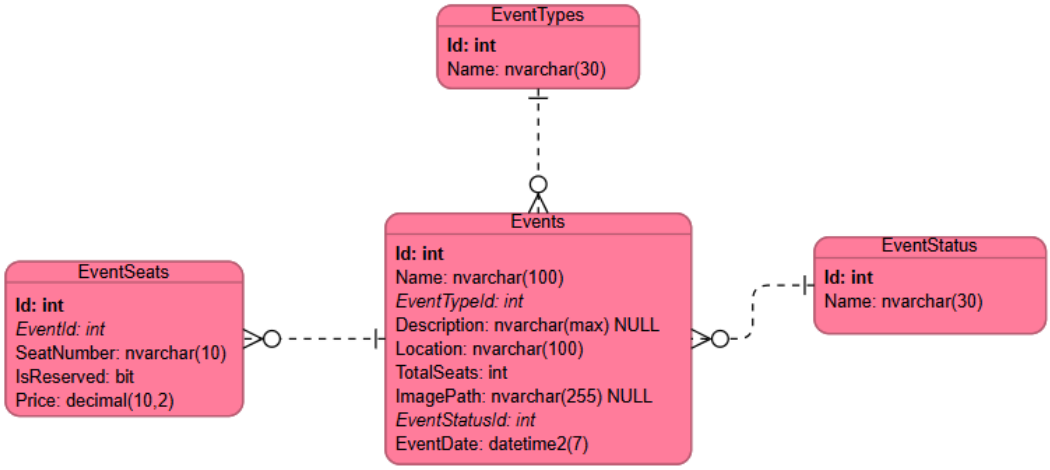

Рисунок 29 – Физическая модель базы данных «emEvent»

На основе физической модели в выбранной СУБД была реализована база данных emEvent. Скрипт создания базы данных представлен в приложении Г на листинге Г.2. В таблице 7 представлена спецификация отношения «Events».

Таблица 7 – Спецификация отношения «Events»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
Id	Идентификатор	int	Нет	>0	PK	1
Name	Название мероприятия	nvarchar(100)	Нет	Нет	–	Концерт

Продолжение таблицы 7

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
EventTypeId	Id типа мероприятия	int	Нет	>0	FK	1
Description	Описание мероприятия	nvarchar(max)	Да	Нет	–	Концерт популярной группы
Location	Место проведения	nvarchar(100)	Нет	Нет	–	Пенза
TotalSeats	Количество мест	int	Нет	>0	–	100
ImagePath	Постер	nvarchar(255)	Да	Нет	–	Img/cnrt.png
EventStatusId	Id статуса мероприятия	int	Нет	>0	FK	1
EventDate	Дата проведения	datetime2(7)	Нет	Нет	–	2025–01–01 13:10

В таблице 8 представлена спецификация отношения «EventSeats», содержащего данные о всех пользовательских местах на мероприятии.

Таблица 8 – Спецификация отношения «EventSeats»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
Id	Идентификатор	int	Нет	>0	PK	1
EventId	Id мероприятия	int	Нет	>0	FK	1
SeatNumber	Номер места	nvarchar(10)	Нет	Нет	–	A1
IsReserved	Состояние места	bit	Нет	[1;2]	–	1
Price	Цена за место	decimal(10,2)	Нет	>0	–	1000.00

Спецификация отношения «EventTypes», содержащего данные о типах мероприятий, представлена в таблице 9.

Таблица 9 – Спецификация отношения «EventTypes»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
Id	Идентификатор	int	Нет	>0	PK	1
Name	Наименование	nvarchar(30)	Нет	Нет	–	Концерт

Спецификация отношения «EventStatus» представлена в таблице 10.

Таблица 10 – Спецификация отношения «EventStatus»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
Id	Идентификатор	int	Нет	>0	PK	1
Name	Наименование	nvarchar(30)	Нет	Нет	–	Активное

На рисунке 30. отображена физическая модель базы данных «emTickets». Скрипт создания данной базы представлен в приложении Г на листинге Г.3.

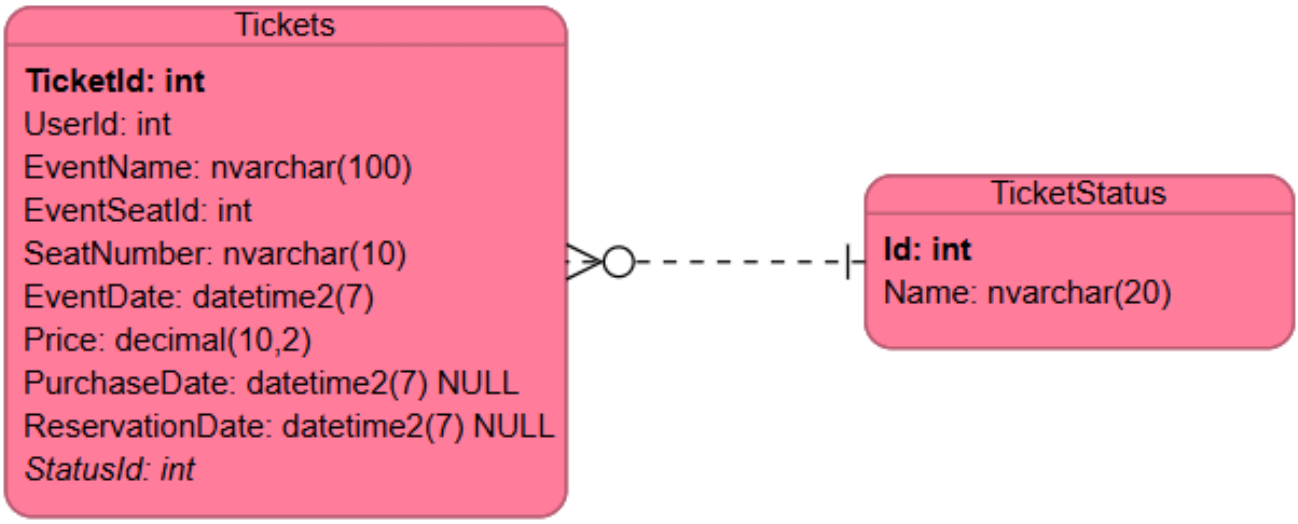

Рисунок 30 – Физическая модель базы данных «emTickets»

В таблице 11 представлена спецификация отношения «Tickets» реализованной в СУБД Microsoft SQL server базы данных «emTickets».

Таблица 11 – Спецификация отношения «Tickets»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
TicketId	Идентификатор	int	Нет	>0	PK	1
UserId	Id пользователя, владеющего билетом	int	Нет	>0	–	133
EventName	Название мероприятия	nvarchar(100)	Нет	Нет	–	Концерт группы
EventSeatId	Id посетительского места	int	Нет	>0	–	45
SeatNumber	Номер места	nvarchar(10)	Нет	Нет	–	A10

Продолжение таблицы 11

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
EventDate	Дата проведения мероприятия	datetime2(7)	Нет	Нет	–	2026–02–01 12:20
Price	Цена за место	decimal(10,2)	Нет	>0	–	1100.00
PurchaseDate	Дата покупки	datetime2(7)	Да	Нет	–	2026–02–01 12:20
ReservationDate	Дата бронирования	datetime2(7)	Да	Нет	–	2026–02–01 12:20
StatusId	Id статуса билета	id	Нет	>0	FK	3

Спецификация отношения «TicketStatus» представлена в таблице 12.

Таблица 12 – Спецификация отношения «TicketStatus»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
Id	Идентификатор	int	Нет	>0	PK	1
Name	Наименование	nvarchar(20)	Нет	Нет	–	Куплен

Физическая модель базы данных «emUser» представлена на рисунке 31. Скрипт создания данной базы данных представлен в приложении Г на листинге Г.1.

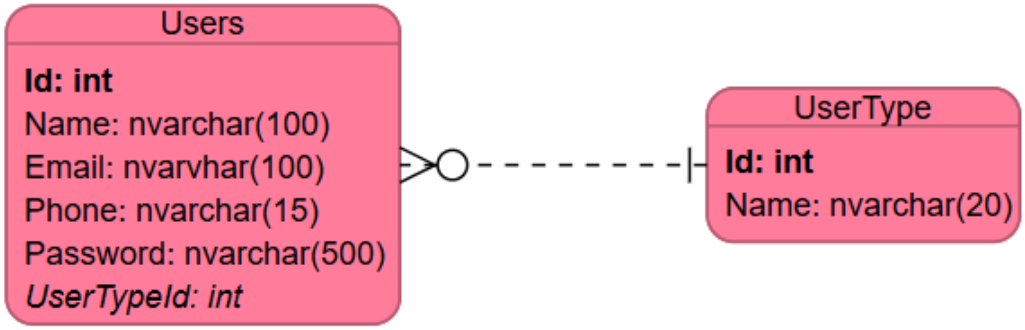

Рисунок 31 – Физическая модель базы данных «emUser»

Спецификация отношения «Users» представлена в таблице 13.

Таблица 13 – Спецификация отношения «Users»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
Id	Идентификатор	int	Нет	>0	PK	1
Name	Имя пользователя	nvarchar(100)	Нет	Нет	–	Вика Бутова
Email	Почта пользователя	nvarchar(100)	Нет	Нет	–	mji@gl.com
Phone	Номер телефона пользователя	nvarchar(15)	Нет	Нет	–	8800544
Password	Пароль	nvarchar(500)	Нет	Нет	–	12345qe
UserTypeId	Id типа пользователя	int	Нет	>0	–	2

В таблице 14 отображена спецификация отношения «UserType».

Таблица 14 – Спецификация отношения «UserType»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Роль	Пример
Id	Идентификатор	int	Нет	>0	PK	1
UserType	Тип пользователя	nvarchar(20)	Нет	Нет	–	Admin

Таким образом, разработанные базы данных позволяют хранить всю необходимую информацию приложения для подбора и продажи билетов на развлекательные мероприятия.

3.2 Программная реализация серверной части приложения

Разработка серверной части проводилась в интегрированной среде разработки Visual Studio 2022. Приложение было реализовано на языке C# с помощью технологии для создания веб–приложений ASP.NET Core на платформе .NET. Диаграмма компонентов серверного приложения представлена на рисунке 32. Код серверной части приведен в приложении Б.

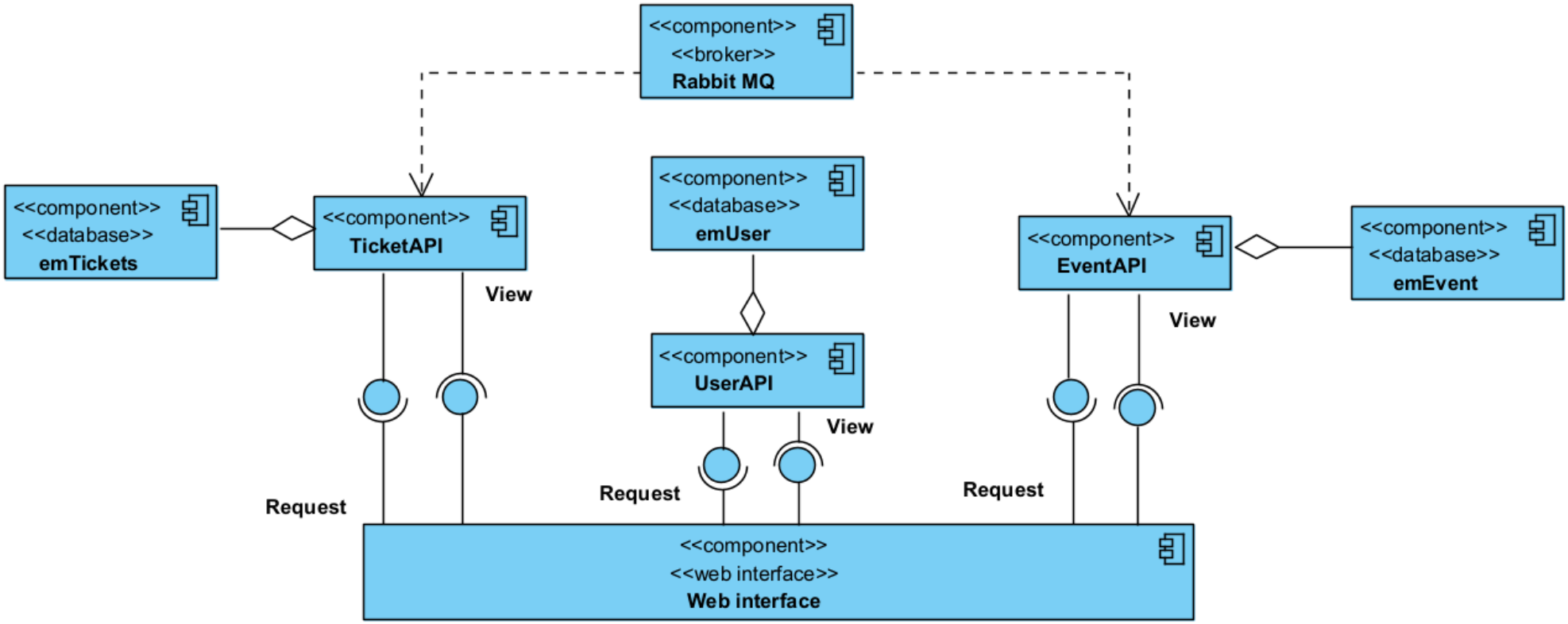

Рисунок 32 – Диаграмма компонентов серверного приложения

В таблице 15 представлено описание компонентов серверной части.

Таблица 15 – Описание компонентов серверной части

Компонент	Описание
UserAPI	Микросервис «Пользователь», отвечающий за авторизацию, регистрацию и управление данными пользователя.
emUser	База данных микросервиса «Пользователь».
EventAPI	Микросервис «Мероприятие», отвечающий за управление информацией о мероприятиях.
emEvent	База данных микросервиса «Мероприятие».
TicketAPI	Микросервис «Билет», отвечающий за управление данными о билетах.
emTickets	База данных микросервиса «Билет».
RabbitMQ	Брокер сообщений, отвечающий за взаимодействие микросервисов «Мероприятие» и «Билет».

Серверное приложение разделено на три микросервиса: «TicketAPI» («Билет»), «EventAPI» («Мероприятие») и «UserAPI» («Пользователь»). Для обмена сообщениями между микросервисами «TicketAPI» и «EventAPI» используется брокер сообщений «RabbitMQ». Из «TicketAPI» сообщения поступают в обменник «Exchange», который затем перенаправляет их в очередь «seat–updates–queue». Из очереди сообщения поступают в микросервис «EventAPI».

Для реализации аутентификации были использованы JWT–токены. JWT токены – это безопасный способ обмена информацией между клиентом и сервером с помощью шифрования. После авторизации в системе через клиентское приложение, сервер выдает уникальный JWT–токен. Для подтверждения авторизации и аутентификации пользователя токен передается на сервер в последующих HTTP–запросах с клиента [12].

Для повышения безопасности данных пользователя было реализовано хеширование паролей. Хеш – результат работы хеш–функции, то есть математического алгоритма, отображающего данные произвольного размера в битовый массив фиксированного размера. Хранение в базе данных хешей

паролей приводит к уменьшению риска нарушения безопасности, так как из хеша невозможно получить пароль обратно. [13].

Для хеширования паролей в серверной части приложения используется алгоритм BCrypt. Данный алгоритм заключается в том, что каждому пользователю при регистрации выдается строка с уникальным значением, которая смешивается с его паролем. Затем получившийся результат проходит через алгоритм хеширования. Таким образом создается уникальный хеш. Для реализации данного алгоритма используется библиотека BCrypt.NET [14].

Для отображения развертывания компонентов системы в каждом ее узле была спроектирована диаграмма развертывания, представленная на рисунке 33.

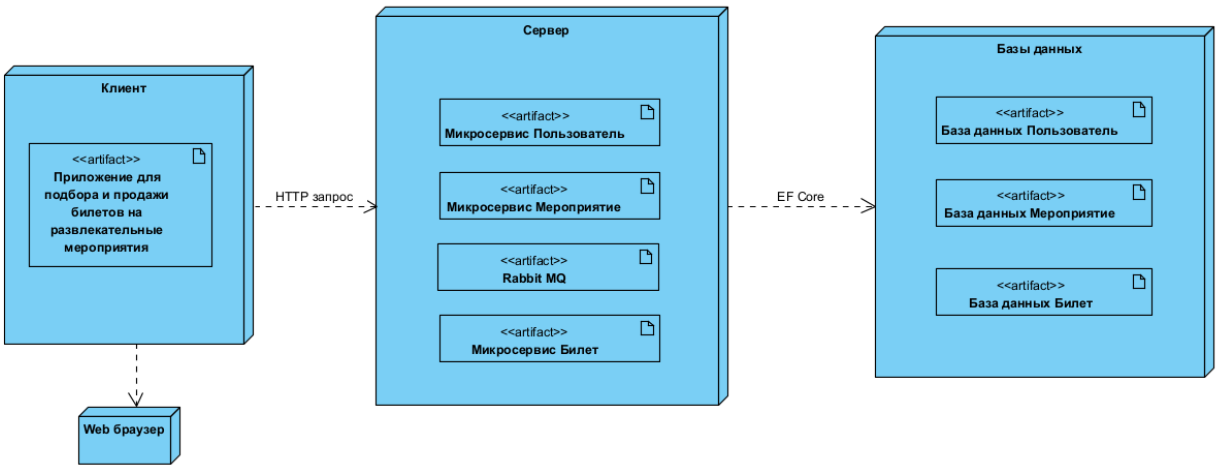

Рисунок 33 – Диаграмма развертывания

Приложения для подбора и продажи билетов на развлекательные мероприятия может быть развернуто на устройствах с операционной системой Windows 10 и больше. Клиентская и серверная части могут быть развернуты отдельно друг от друга.

Таким образом, была реализована серверная часть приложения для подбора и покупки билетов на развлекательные мероприятия.

3.3 Программная реализация клиентской части приложения

Клиентская часть приложения была разработана в редакторе исходного кода Visual Studio Code. Приложение было разработано на языке JavaScript с

использованием библиотеки React, разделяющей интерфейс на компоненты. Диаграмма компонентов клиентской части приложения представлена на рисунке 33. Код клиентской части приведен в Приложении В.

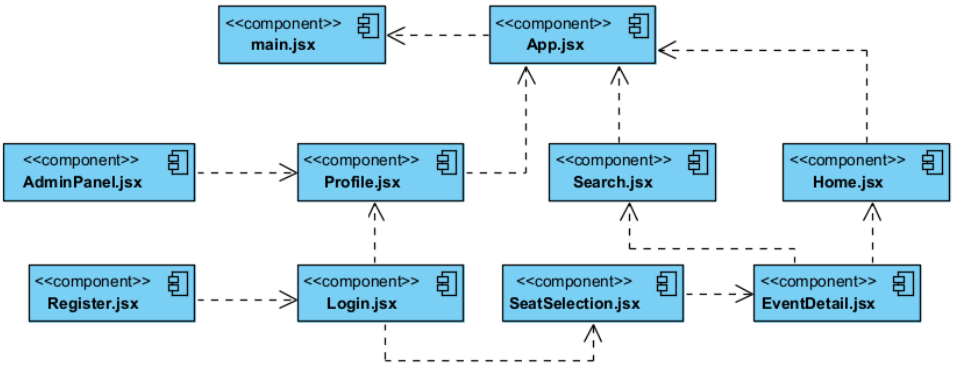

Рисунок 34 – Диаграмма компонентов клиентской части

В таблице 16 представлено описание компонентов клиентской части.

Таблица 16 – Описание компонентов клиентской части

Компонент	Описание
main.jsx	Компонент, являющийся точкой входа и запускающий компонент App.jsx.
App.jsx	Главный компонент, управляющий навигацией.
Profile.jsx	Компонент, реализующий функционал страницы «Личный кабинет».
Search.jsx	Компонент, реализующий функционал страницы «Поиск».
Home.jsx	Компонент, реализующий функционал страницы «Главная».
AdminPanel.jsx	Компонент, реализующий функционал страницы «Панель администратора».
EventDetail.jsx	Компонент, реализующий функционал страницы информации о мероприятии.
SeatSelection.jsx	Компонент, реализующий функционал страницы выбора места для покупки или бронирования.
Login.jsx	Компонент, реализующий функционал страницы «Вход».
Register.jsx	Компонент, реализующий функционал страницы «Регистрация».

Таким образом, была реализована клиентская часть приложения для подбора и покупки билетов на развлекательные мероприятия.

4 Отладка и тестирование программного обеспечения

4.1 Обоснование выбора режима тестирования

Тестирование – проверка реализуемого программного обеспечения на соответствие заданным требованиям, необходимая для подтверждения отсутствия дефектов [15]. Существует довольно много видов тестирования, одними из самых используемых являются:

– модульное тестирование;
– тестирование черным ящиком;
– интеграционное тестирование.

Модульное тестирование или Unit–тестирование – это тестирование, нацеленное на проверку корректного выполнения отдельных модулей или методов программного кода. Модульное тестирование позволяет обнаруживать и затем устранять ошибки на ранних стадиях разработки. Это делает приложение более качественным и эффективным, поэтому этот режим тестирования был выбран для автоматического тестирования приложения [16].

Тестирование черным ящиком – тестирование, суть которого заключается в том, что тестируемая система воспринимается как черный ящик, то есть ее внутренняя структура, архитектура и реализация неизвестны. Этот вид тестирования позволяет понять поведение системы при ожидаемых и неожиданных действиях пользователя. Данный способ позволяет узнать, работает ли приложение корректно и соответствует ли оно требованиям, не вникая в его реализацию. Это и является основным преимуществом тестирования черным ящиком и является обоснованием для выбора данного режима для ручного тестирования приложения [17].

Интеграционное тестирование – это тип тестирования, при котором логически объединяются уже прошедшие юнит–тестирование модули и тестируются как группа. Цель интеграционного тестирования – найти проблемы во взаимодействиях выбранных модулей. Преимущество этого тестирования

заключается в обеспечении надежности системы за счет проверки корректного взаимодействия ее компонентов [18].

4.2 Модульное тестирование

Для выполнения модульного тестирования на примере микросервиса «Пользователь» были разработаны Unit–тесты в среде разработки Visual Studio 2022 на языке C#. Код Unit–тестов приведен в приложении Д. В процессе разработки тестов был составлен план модульного тестирования, приведенный в таблице 17.

Таблица 17 – План модульного тестирования

Тест	Описание	Ожидаемый результат	Актуальный результат
RegisterAsync_NewUser_ShouldCreateUser	Проверка правильной работы метода, отвечающего за регистрацию пользователя	Пользователь зарегистрирован	Пользователь зарегистрирован
RegisterAsync_ExistingUser_ReturnsNull	Проверка правильной работы метода, отвечающего за регистрацию пользователя, при вводе данных существующего пользователя	Возвращен null объект	Возвращен null объект
VerifyPasswordAsync_ValidCredentials_ReturnsTrue	Проверка правильной работы метода, отвечающего за вход пользователя в систему	Возвращено true	Возвращено true
VerifyPasswordAsync_InvalidEmail_ReturnsFalse	Проверка правильной работы метода, отвечающего за вход пользователя в систему, при вводе некорректной почты	Возвращено false	Возвращено false
VerifyPasswordAsync_InvalidPassword_ReturnsFalse	Проверка правильной работы метода, отвечающего за вход пользователя в систему, при вводе некорректного пароля	Возвращено false	Возвращено false
GetUserByEmailAsync_ExistingUser_ReturnsUser	Проверка правильной работы метода, возвращающего информацию о пользователе по его id	Возвращен объект User	Возвращен объект User

Продолжение таблицы 17

Тест	Описание	Ожидаемый результат	Актуальный результат
UpdateUserNameAsync_ValidId_UpdatesName	Проверка правильной работы метода, изменяющего имя пользователя, при вводе корректного имени пользователя	Имя изменено	Имя изменено
UpdateUserNameAsync_InvalidId_ReturnsNull	Проверка правильной работы метода, изменяющего имя пользователя, при вводе некорректного имени пользователя	Возвращен null объект	Возвращен null объект
UpdateUserPasswordAsync_ValidCredentials_ReturnsTrue	Проверка правильной работы метода, изменяющего пароль пользователя, при вводе корректного id пользователя и корректного старого пароля	Возвращено true	Возвращено true
UpdateUserPasswordAsync_InvalidOldPassword_ReturnsFalse	Проверка правильной работы метода, изменяющего пароль пользователя, при вводе некорректного старого пароля	Возвращено false	Возвращено false
UpdateUserPasswordAsync_InvalidId_ReturnsFalse	Проверка правильной работы метода, изменяющего пароль пользователя, при вводе некорректного id пользователя	Возвращено false	Возвращено false

На рисунке 35 представлены результаты модульного тестирования.

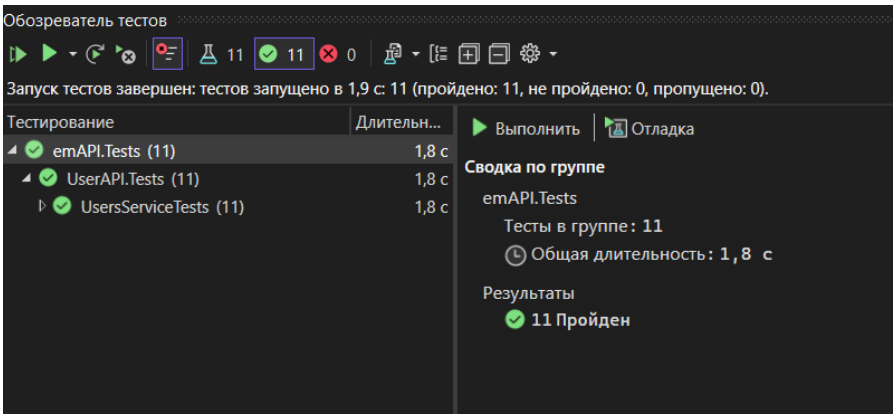

Рисунок 35 – Результаты модульного тестирования микросервиса «Пользователь»

Основываясь на результатах тестирования, можно сделать вывод, что проблем не было выявлено и модули приложения работают корректно.

4.3 Тестирование черным ящиком

В качестве функционального тестирования было выбрано тестирование черным ящиком. Для проведения тестирования на примере поля для ввода наименования мероприятия были составлены классы эквивалентности, представленные в таблице 18.

Таблица 18 – Классы эквивалентности

Требования	Правильные КЭ	Неправильные КЭ
Длина наименования	1–100 (1)	0 (2) 101 – ∞ (3)
Структура	Начинается с буквы (4)	Начинается не с буквы (цифра, символ) (5)
Содержание	Буквы (кириллица/латиница) (6) Цифры (7) Пробелы (8) Дефисы (–) (9) Точки (.) (10)	Остальное (11)

В соответствии с представленными классами эквивалентности были разработаны тест–кейсы, отображенные в таблице 19. Результаты тестирования представлены в приложении Е.

Таблица 19 – Тест–кейсы

№	КЭ	Входные данные	Выходные данные
1	1, 4, 6, 7, 8, 9	Концерт группы The1975	Успешно (рисунок Е.1)
2	2	‘’	Ошибка: поле выделяется, отображается сообщение «Заполните это поле» (рисунок Е.2)
3	3, 4, 6, 8, 10	Аааааааааааааааааааааааааааа Аааааааааааааааааааааааааааа Аааааааааааааааааааааааааааа Аааааааааааааааааааааааааааа Аааааааааааааааааааааааааааа	Ошибка: длина наименования больше 100 символов (рисунок Е.3)

Продолжение таблицы 19

№	КЭ	Входные данные	Выходные данные
4	1, 5, 6, 8, 10	.Ольга	Ошибка: наименование начинается не с буквы (рисунок Е.4)
5	1, 4, 6, 11	Альбина@@@@@ва	Ошибка: наименование содержит недопустимые символы (рисунок Е.5)
6	1, 5, 7	2010	Ошибка: наименование начинается не с буквы (рисунок Е.6)
7	1, 5, 9	———	Ошибка: наименование начинается не с буквы (рисунок Е.7)
8	1, 5, 10	…	Ошибка: наименование начинается не с буквы (рисунок Е.8)
9	1, 5, 7, 9, 10	30–41	Ошибка: наименование начинается не с буквы (рисунок Е.9)
10	1, 4, 6, 7, 8	Новогодний утренник 2005	Успешно (рисунок Е.10)
11	1, 4, 6, 8	Вечеринка без подробностей	Успешно (рисунок Е.11)
12	1,4,6,9,10	ВыставкаИмениИсагиИзБлича–2025	Успешно (Рисунок Е.12)

Исходя из представленных тест–кейсов можно сделать вывод, что результаты работы приложения во время тестирования не отличаются от ожидаемых, что означает корректную работу приложения.

5 Контроль качества программного обеспечения

5.1 Обоснование выбора метрик программного кода

Метрики программного кода – это показатели, по которым оценивается качество и характеристики программного кода. С помощью метрик можно анализировать код, повышать его качество, улучшать структуру, выявлять слабые места, оценивать его сложность, читаемость, поддерживаемость и прочите характеристики. Одни из часто используемых метрик:

– метрика Холстеда;
– метрика Чепина.

Метрика Холстеда основана на анализе кода программы через подсчет операторов и операндов. С помощью данной метрики можно оценить такие характеристики, как длина программы, объем, сложность и другие [19].

Метрика Чепина основана на анализе текста программы и выявлении следующих групп переменных: переменные для расчетов и обеспечения вывода, модифицируемые переменные, управляющие переменные и не используемые переменные. Для каждой группы рассчитываются весовые коэффициенты, на основе которых делается вывод о влиянии этой группы на код программы [20].

Для оценки качества программного кода была выбрана метрика Холстеда.

5.2 Метрики качества программного кода

Для подсчета метрики Холстеда нужно вычислить 4 показателя кода приложения:

– число уникальных операторов программы (n1);
– число уникальных операндов программы (n2);
– общее число операторов в программе (N1);
– общее число операндов в программе (N2);
– теоретическое число уникальных операторов (n1');
– теоретическое число уникальных операндов (n2').

Вычисление характеристик будет проводиться для метода покупки билета PurchaseTicketAsync класса TicketService микросервиса TicketAPI. Расчеты показателей приведены в таблице 20.

Таблица 20 – Расчеты показателей кода

№	Оператор	Число операторов	Операнд	Число операндов
1	=	10	userId	2
2	.	7	eventSeatId	2
3	await	3	eventSeat	5
4	return	3	null	3
5	{	3	ticket	2
6	new	2	message	2
7	var	2	_ticketRepository	2
8	public	1	GetTicketByUserAndEventSeatAsync	1
9	async	1	EventSeatId	2
10	Task <Ticket>	1	IsReserved	1
11	if	1	true	1
12	!=	1	_messageBroker	1
13	try	1	SendMessageAsync	1
14	catch	1	«event–seat–exchange»	1
15	??	1	«seat.reserved»	1
16	;		TimeoutException	1
17			ex	2
18			_logger	1
19			LogWarning	1
20			«Не удалось зарезервировать место: {Message}»	1
21			Message	1
22			Ticket	1
23			EventName	1
24			Event	2
25			Name	1
26			SeatNumber	2
27			EventDate	2
28			DateTime	2
29			MinValue	1
30			Price	2
31			StatusId	1
32			1	1
33			PurchaseDate	1
34			Now	1
35			ReservationDate	1
36			AddTicketAsync	1

Результаты подсчета операторов и операндов отображены в таблице 21.

Таблица 21 – Результаты расчета показателей кода

Показатель	Значение
Число уникальных операторов программы (n1)	16
Число уникальных операндов программы (n2)	36
Общее число операторов в программе (N1)	44
Общее число операндов в программе (N2)	52
Теоретическое число уникальных операторов (n1')	12
Теоретическое число уникальных операндов (n2')	31

Основываясь на рассчитанных показателях, можно вычислить основные характеристики метрик Холстеда. Результаты вычислений приведены в таблице 22.

Таблица 22 – Характеристики метрик Холстеда

Характеристика	Значение
Словарь программы (n)	52
Длина программы (N)	96
Объем программы (V)	547.39
Вычислительная сложность программы (D)	11.56
Информационное содержание (I)	47.34

Все итоговые значения характеристик метрик Холстеда находятся в пределах нормы. Значения длины программы, словаря и объема указывают на компактность анализируемого метода. Из умеренного значения вычислительной сложности и информационного содержания можно сделать вывод о средней сложности понимания метода и его ясной структуре.

6 Планирование разработки и оценка бюджета

Для оценки затраченного времени на разработку приложения для подбора и продажи билетов на развлекательные мероприятия и наглядного планирования данного процесса были выделены основные этапы проекта:

1) постановка задачи и анализ требований;
2) проектирование;
3) программная реализация;
4) контроль качества программного обеспечения.

Каждый этап включает в себя набор задач, описание которых отображено на рисунке 36.

Название	Продолжит...	Начало	Окончание
Начало проекта	0 дней	08.04.2025, 08:00	08.04.2025, 08:00
Постановка задачи и анализ тр	5 дней	**08.04.2025, 08:00**	**14.04.2025, 17:00**
Постановка задачи	1 день	08.04.2025, 08:00	08.04.2025, 17:00
Анализ систем аналогичного назн	1 день	09.04.2025, 08:00	09.04.2025, 17:00
Формулировка требований	1 день	11.04.2025, 08:00	11.04.2025, 17:00
Планирование разработки и оцен	1 день	14.04.2025, 08:00	14.04.2025, 17:00
Проектирование	9 дней	**15.04.2025, 08:00**	**25.04.2025, 17:00**
Проектирование структур данных	1 день	15.04.2025, 08:00	15.04.2025, 17:00
Проектирование классов	4 дней	16.04.2025, 08:00	21.04.2025, 17:00
Проектирование алгоритмов	1 день	22.04.2025, 08:00	22.04.2025, 17:00
Проектирование интерфейсов	3 дней	23.04.2025, 08:00	25.04.2025, 17:00
Программная реализация	11 дней	**28.04.2025, 08:00**	**12.05.2025, 17:00**
Программная реализация баз дан	3 дней	28.04.2025, 08:00	30.04.2025, 17:00
Программная реализация серверн	5 дней	01.05.2025, 08:00	07.05.2025, 17:00
Программная реализация клиентс	3 дней	08.05.2025, 08:00	12.05.2025, 17:00
Контроль качества программно	8 дней	**13.05.2025, 08:00**	**22.05.2025, 17:00**
Выбор режима тестирования	1 день	13.05.2025, 08:00	13.05.2025, 17:00
Тестирование программных средс	5 дней	14.05.2025, 08:00	20.05.2025, 17:00
Рассчет метрик качества програм	2 дней	21.05.2025, 08:00	22.05.2025, 17:00
Оформление отчета	8 дней	**23.05.2025, 08:00**	**03.06.2025, 17:00**
Оформление пояснительной запи	8 дней	23.05.2025, 08:00	03.06.2025, 17:00
Конец проекта	0 дней	03.06.2025, 17:00	03.06.2025, 17:00

Рисунок 36 – Описание задач каждого этапа

Для визуального отображения процесса разработки используя программное обеспечение для управления проектами Project Libre была спроектирована диаграмма Ганта, отображающая все этапы процесса. Диаграмма представлена на рисунке 37.

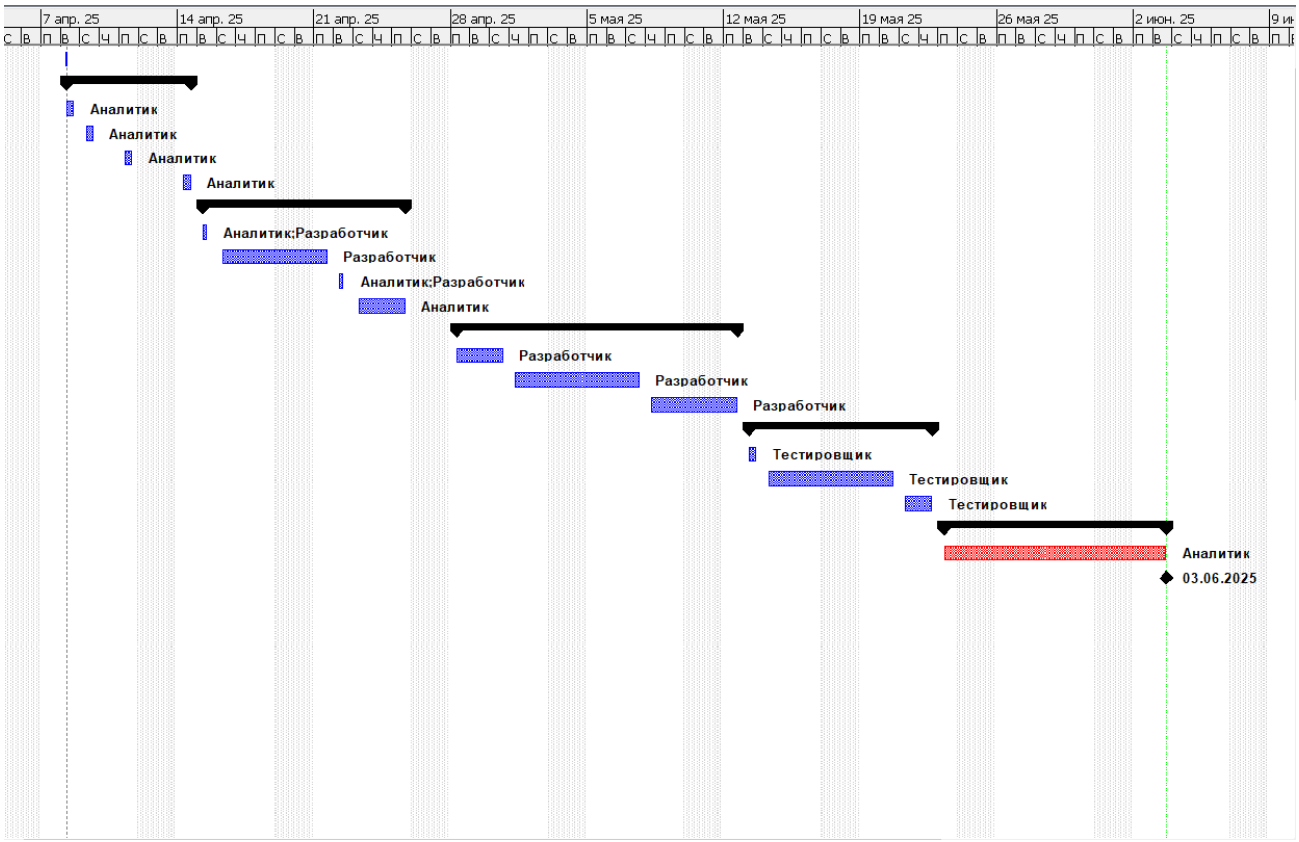

Рисунок 37 – Диаграмма Ганта

Из диаграммы Ганта можно сделать вывод, что в начале разработки ведется активная аналитическая деятельность: проводится анализ требований и предметной области. На этапе проектирования описывается архитектура приложения, интерфейсы и модели данных. Основные участники этапа – разработчик и аналитик. Разработкой занимаются только разработчики. На этапе тестирования основным участником является тестировщик, но также участвует аналитик.

Для оценки финансовых затрат были назначены цены ресурсов. Информация о ценах отображена на рисунке 38.

Название	Стандартная ставка
Аналитик	150,00 ₽/час
Разработчик	120,00 ₽/час
Тестировщик	100,00 ₽/час

Рисунок 38 – Цены за ресурсы

Оценка стоимости ресурсов для каждого этапа проекта представлена на рисунке 39.

Название	Оставшаяся стоимость
Начало проекта	0,00 ₽
Постановка задачи и ана	**4800,00 ₽**
Постановка задачи	1200,00 ₽
Анализ систем аналогичнои	1200,00 ₽
Формулировка требований	1200,00 ₽
Планирование разработки	1200,00 ₽
Проектирование	**9600,00 ₽**
Проектирование структур д	1080,00 ₽
Проектирование классов	3840,00 ₽
Проектирование алгоритмс	1080,00 ₽
Проектирование интерфей	3600,00 ₽
Программная реализация	**10560,00 ₽**
Программная реализация б	2880,00 ₽
Программная реализация с	4800,00 ₽
Программная реализация к	2880,00 ₽
Контроль качества прогр	**6400,00 ₽**
Выбор режима тестирован	800,00 ₽
Тестирование программны	4000,00 ₽
Рассчет метрик качества п	1600,00 ₽
Оформление отчета	**9600,00 ₽**
Оформление пояснительнс	9600,00 ₽
Конец проекта	0,00 ₽

Рисунок 39 – Стоимость ресурсов на каждом этапе

Исходя из оценки стоимости ресурсов и затраченного времени, можно сделать вывод, что на проектирование, разработку и тестирование приложения для подбора и продажи билетов на развлекательные мероприятия потребуется 40960 рублей. Все этапы займут 56 дней.

Заключение

В рамках выпускной квалификационной работы было спроектировано и разработано приложение для подбора и продажи билетов на развлекательные мероприятия.

На этапе анализа требований была поставлена задача, проведен анализ требований к разработке и систем аналогичного назначения, а также выбраны и обоснованы технологии.

На этапе проектирования приложения для подбора и продажи билетов на развлекательные мероприятия были спроектированы концептуальная модель базы данных и логические модели баз данных, спроектированы и описаны диаграммы классов для каждого микросервиса, а также было проведено проектирование интерфейса взаимодействия клиентской и серверной части и проектирование пользовательского интерфейса.

Этап программной реализации включал в себя реализацию базы данных в СУБД SQL Server, разработку трех микросервисов в интегрированной среде разработки Visual Studio на языке c# с помощью технологии для создания веб–приложений ASP.NET Core, а также реализацию серверного приложения в редакторе исходного кода Visual Studio Code на языке JavaScript с использованием библиотеки React. Разработанное приложение было протестировано модульными тестами и функциональным тестированием черным ящиком, все тесты были успешно пройдены. Также был проведен подсчет метрик, в результате которого был сделан вывод об оптимальности разработанного приложения.

Для проекта были проведены планирование разработки и оценка бюджета. В результате было выявлено, что для выполнения всех этапов разработки необходимо 56 дней и 40960 рублей.

Таким образом, в результате выполнения выпускной квалификационной работы поставленные задачи по разработке приложения для подбора и продажи билетов на развлекательные мероприятия были выполнены в полном объеме.

Список использованных источников

1. Памятка РП. Микросервисная архитектура // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/795297/ (дата обращения: 9.04.2025).

2. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/566218/ (дата обращения: 9.04.2025).

3. Руководство по ASP.NET Core 9 // Metanit [Электронный ресурс]. URL: https://metanit.com/sharp/aspnet6/ (дата обращения: 10.04.2025).

4. Когда использовать базу данных MS SQL Server? // ДБ сервис [Электронный ресурс]. URL: https://www.dbserv.ru/blog/kogda-ispolzovat-bazu-dannyh-ms-sql-server (дата обращения: 10.04.2025).

5. Когда и зачем нужен RabbitMQ // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/slurm/articles/684412/ (дата обращения: 10.04.2025).

6. Entity Framework Core // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/otus/articles/500012/ (дата обращения: 10.04.2025).

7. Основы React: всё, что нужно знать для начала работы // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/ruvds/articles/343022/ (дата обращения: 10.04.2025).

8. Моделирование данных: обзор работы // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/556790/ (дата обращения: 16.04.2025).

9. Отношения классов – от UML к коду // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/150041/ (дата обращения: 18.04.2025).

10. Диаграмма активностей (Activity diagram) // Flexberry [Электронный ресурс] URL: https://flexberry.github.io/ru/fd_activity-diagram.html (дата обращения: 20.04.2025).

11. Диаграмма последовательности (sequence-диаграмма) // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/814769/ (дата обращения: 21.04.2025).

12. JWT для начинающих: что такое JSON Web Tokens и зачем они нужны // Tproger [Электронный ресурс]. URL: https://tproger.ru/articles/jwt-dlya-nachinayushhih-chto-takoe-json-web-tokens-i-zachem-oni-nuzhny (дата обращения: 29.04.2025).

13. Хеш-функция, что это такое? // Хабр [Электронный ресурс] URL: https://habr.com/ru/articles/534596/ (дата обращения: 29.04.2025).

14. Как библиотека bcrypt автоматически справляется с добавлением соли и хешированием паролей? // Skypro [Электронный ресурс] URL: https://sky.pro/wiki/java/sokhranenie-sluchaynoy-soli-pri-kheshirovanii-paroley-bcrypt/ (дата обращения: 29.04.2025).

15. Что такое тестирование программного обеспечения? Определение, основы и типы // Logrocon [Электронный ресурс] URL: https://logrocon.ru/news/testing_is (дата обращения: 16.05.2025).

16. Лучшие практики модульного тестирования // Хабр [Электронный ресурс] URL: https://habr.com/ru/articles/795105/ (дата обращения: 16.05.2024).

17. Тестирование методом «чёрного ящика» // Точка качества [Электронный ресурс] URL: https://tquality.ru/blog/testirovanie-metodom-chernogo-yashchika/ (дата обращения: 17.05.2024).

18. Интеграционное тестирование: что это? Виды, примеры. // Logrocon [Электронный ресурс] URL: https://logrocon.ru/news/intgration_testing (дата обращения: 17.05.2024).

19. Метрики кода программного обеспечения // PVS-studio [Электронный ресурс] URL: https://pvs-studio.ru/ru/blog/posts/a0045/ (дата обращения: 18.05.2024).

20. Лексический анализ. Метрики Чепина // StudFiles [Электронный ресурс] URL: https://studfile.net/preview/6196490/page:8/ (дата обращения: 18.05.2024).

Приложение А

(рекомендуемое)

Графическая часть

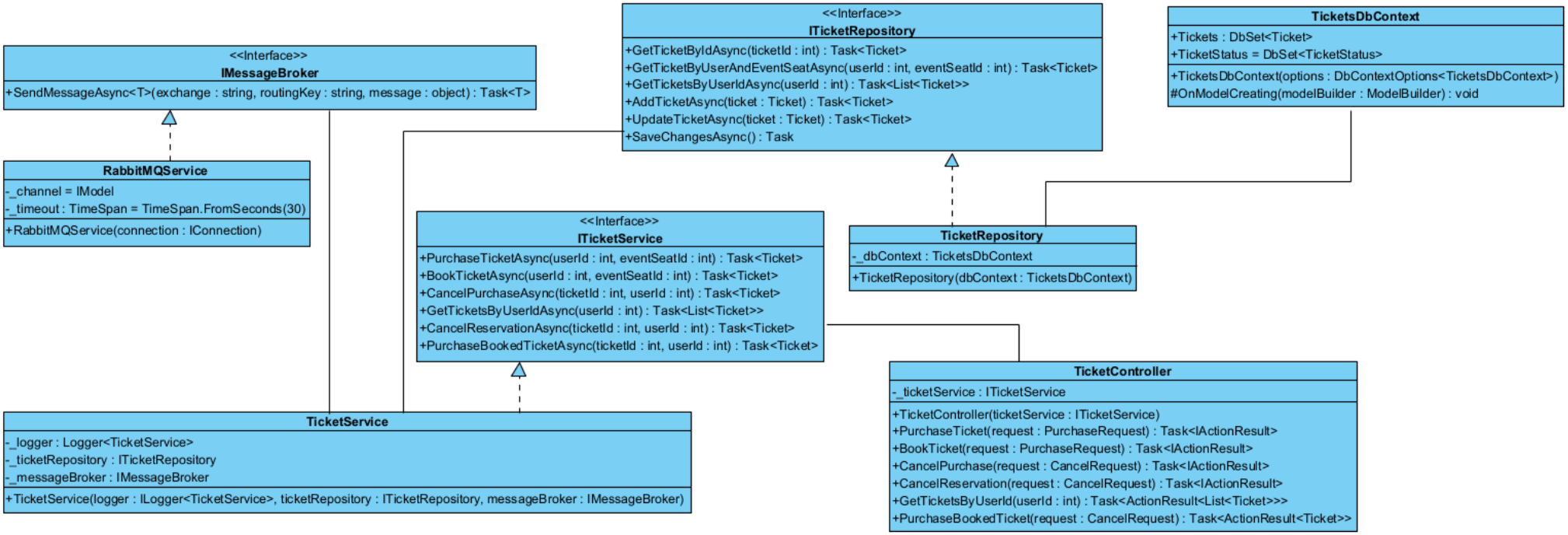

Рисунок А.1 – Диаграмма классов микросервиса Билет

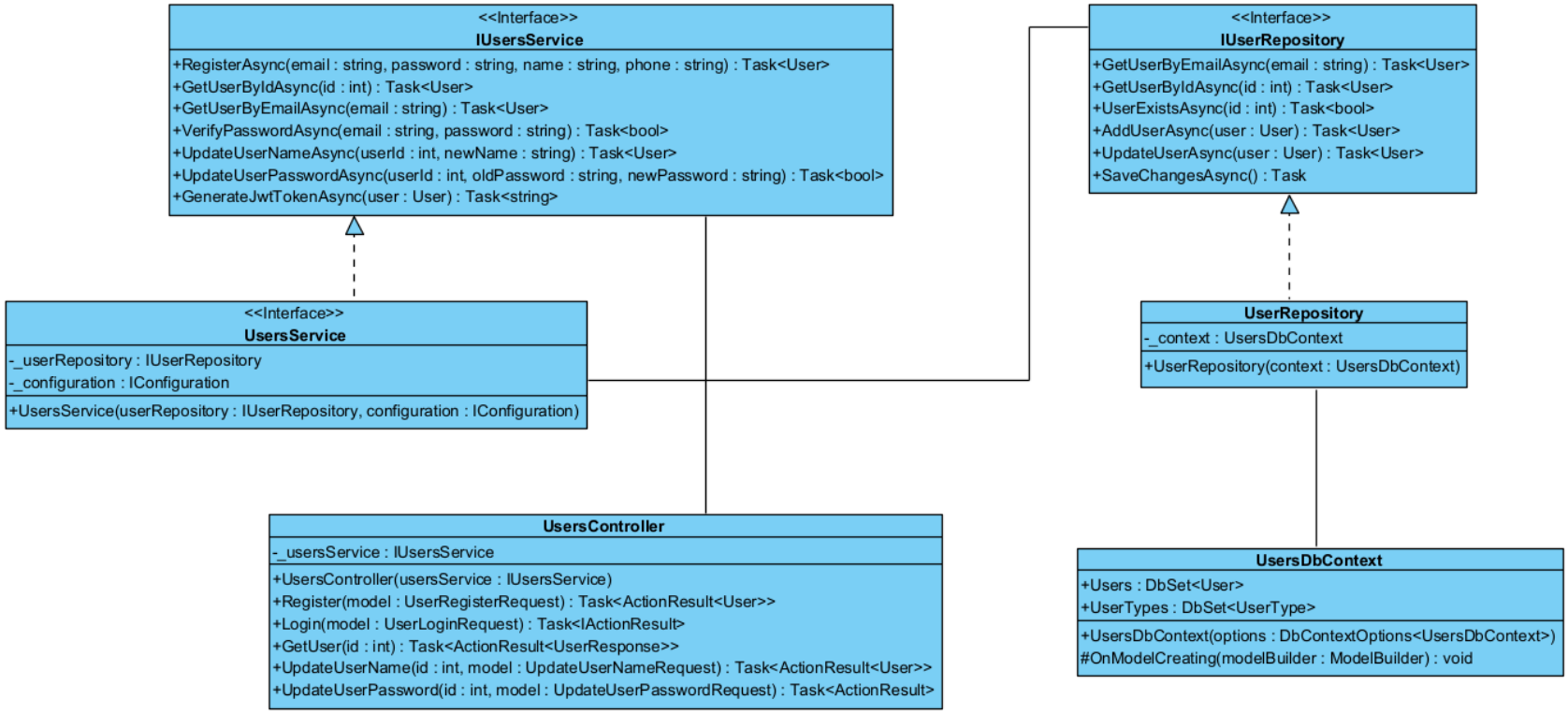

Рисунок А.2 – Диаграмма классов микросервиса Пользователь

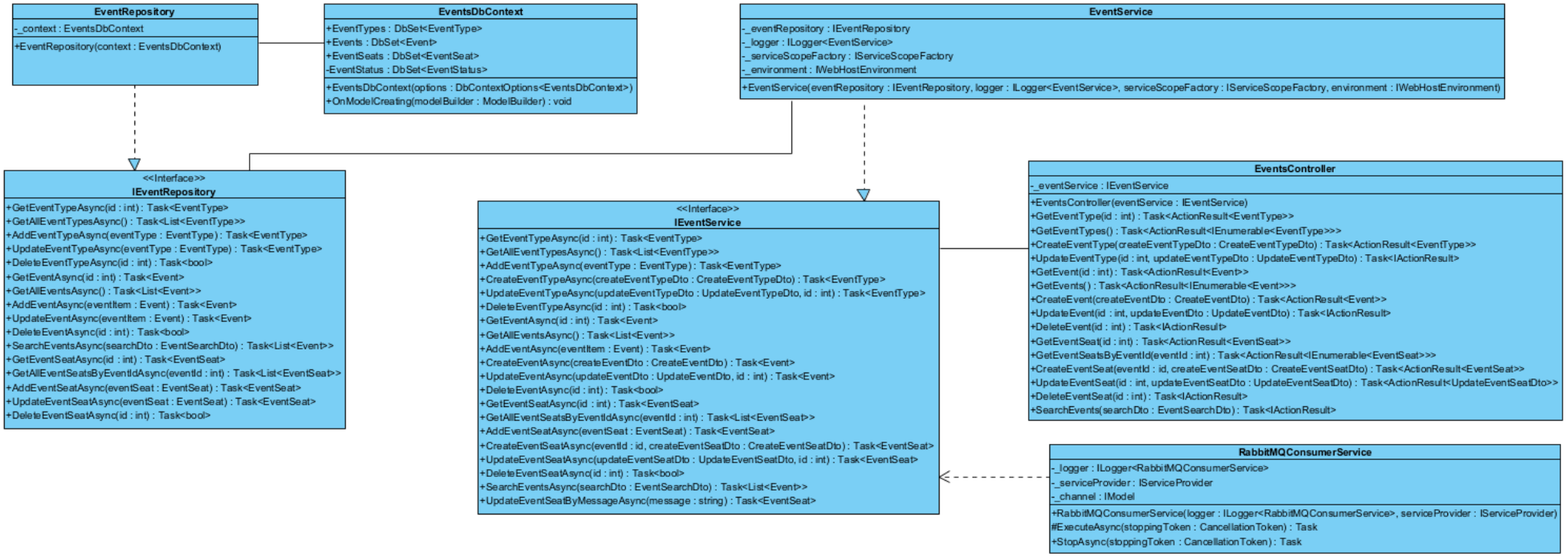

Рисунок А.3 – Диаграмма классов микросервиса Мероприятие

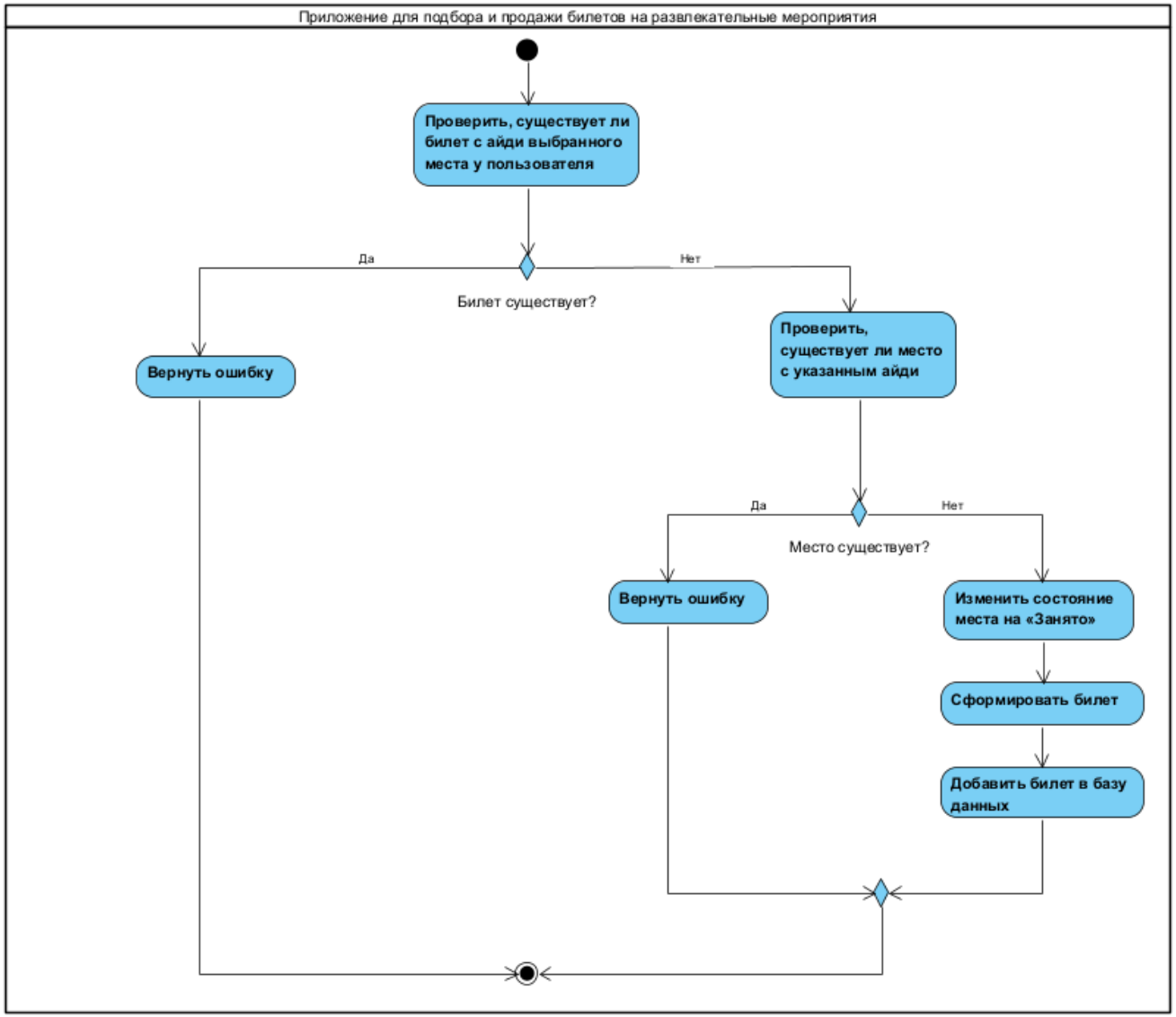

Рисунок А.4 – Диаграмма деятельности

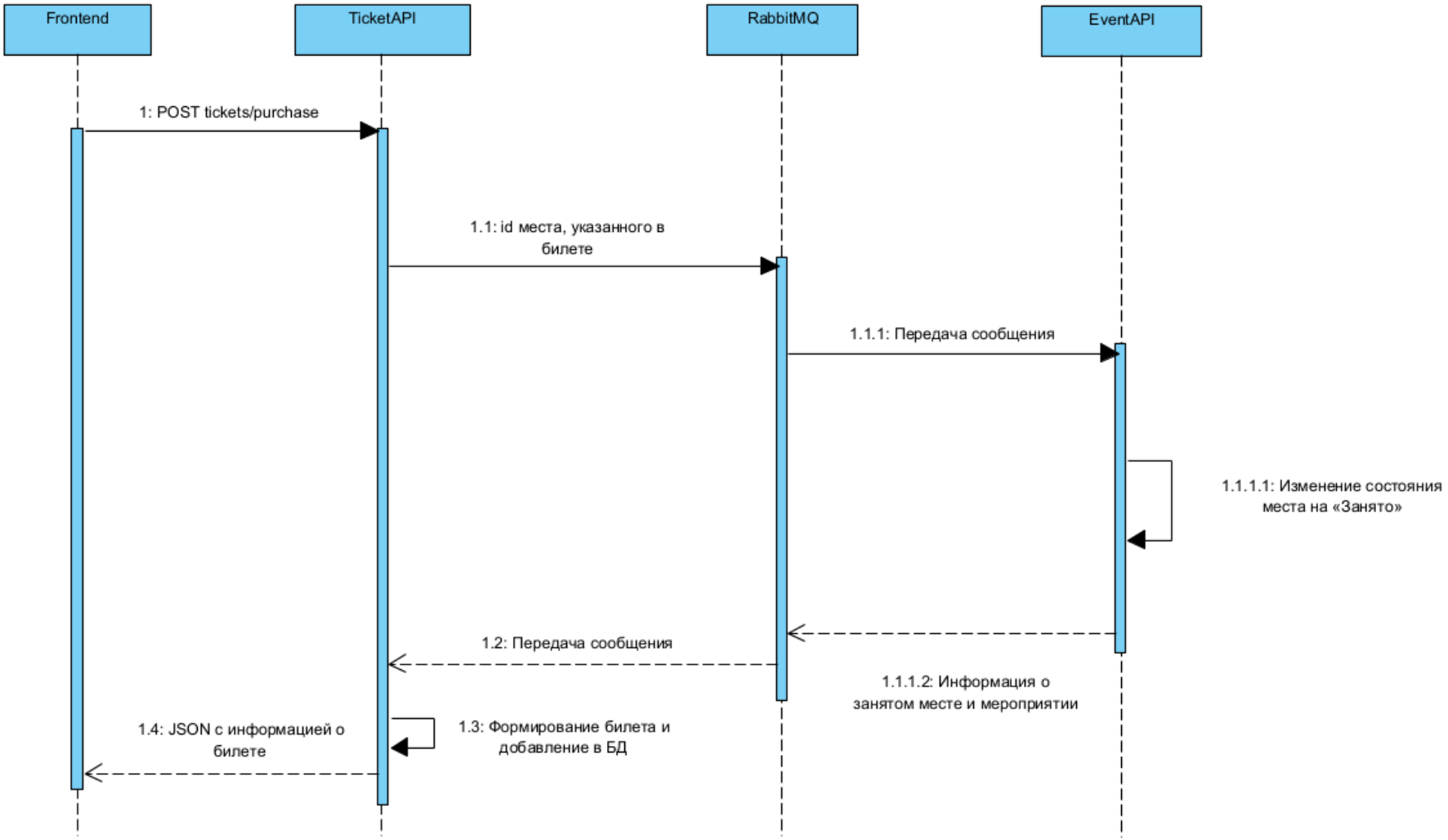

Рисунок А.5 – Диаграмма последовательности

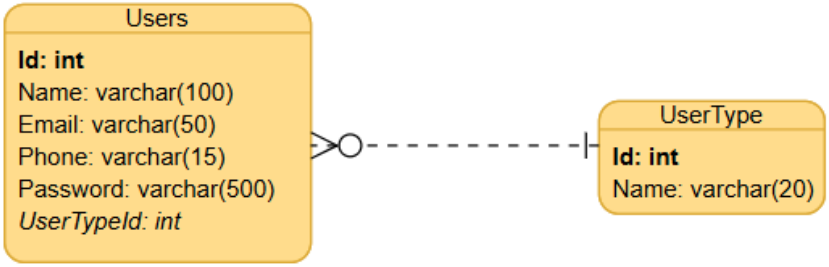

Рисунок А.6 – Логическая модель базы данных «emUser»

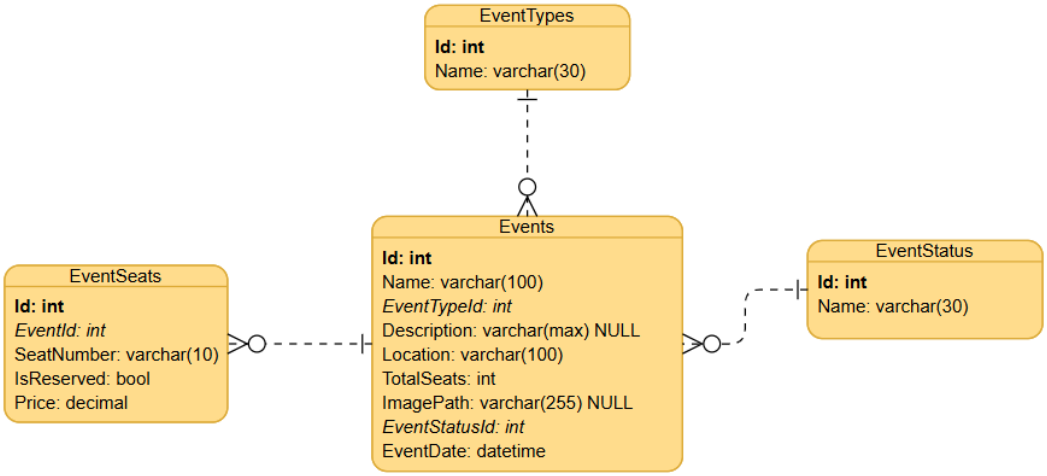

Рисунок А.7 – Логическая модель базы данных «emEvent»

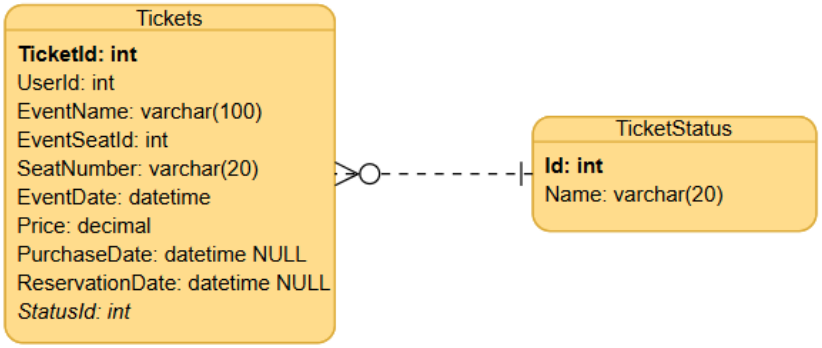

Рисунок А.8 – Логическая модель базы данных «emTickets»

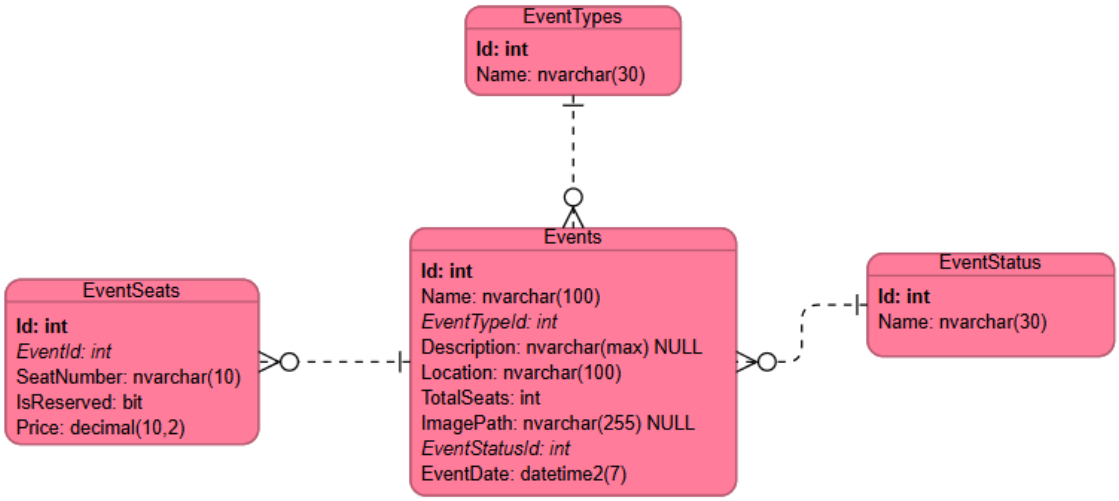

Рисунок А.9 – Физическая модель базы данных emEvent

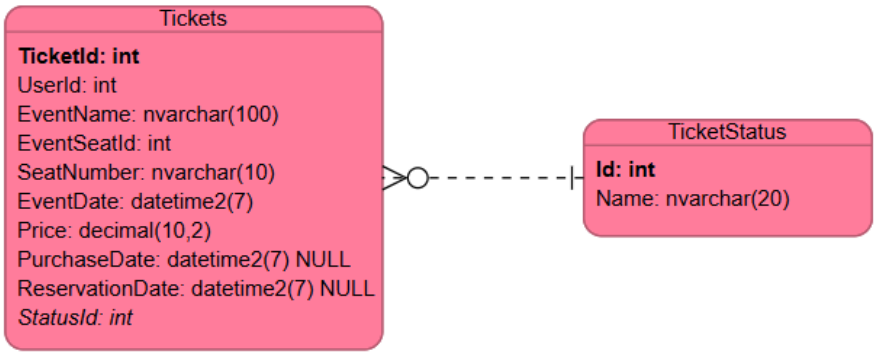

Рисунок А.10 – Физическая модель базы данных emTickets

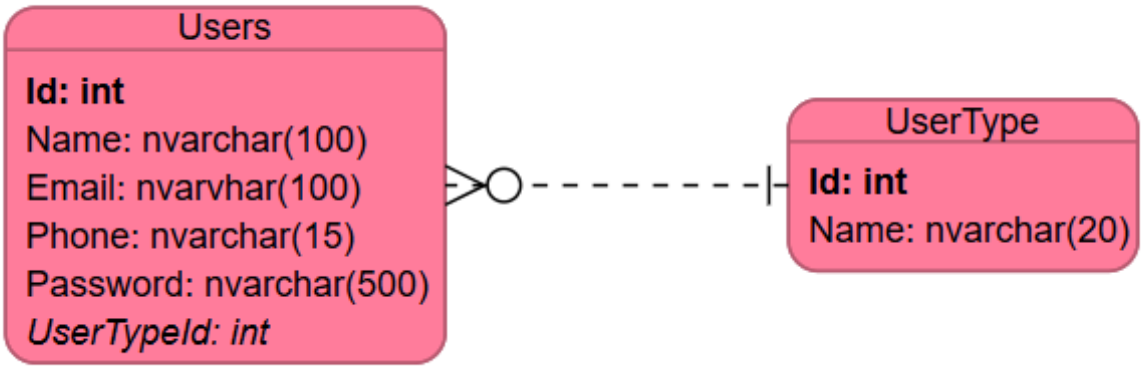

Рисунок А.11 – Физическая модель базы данных emUser

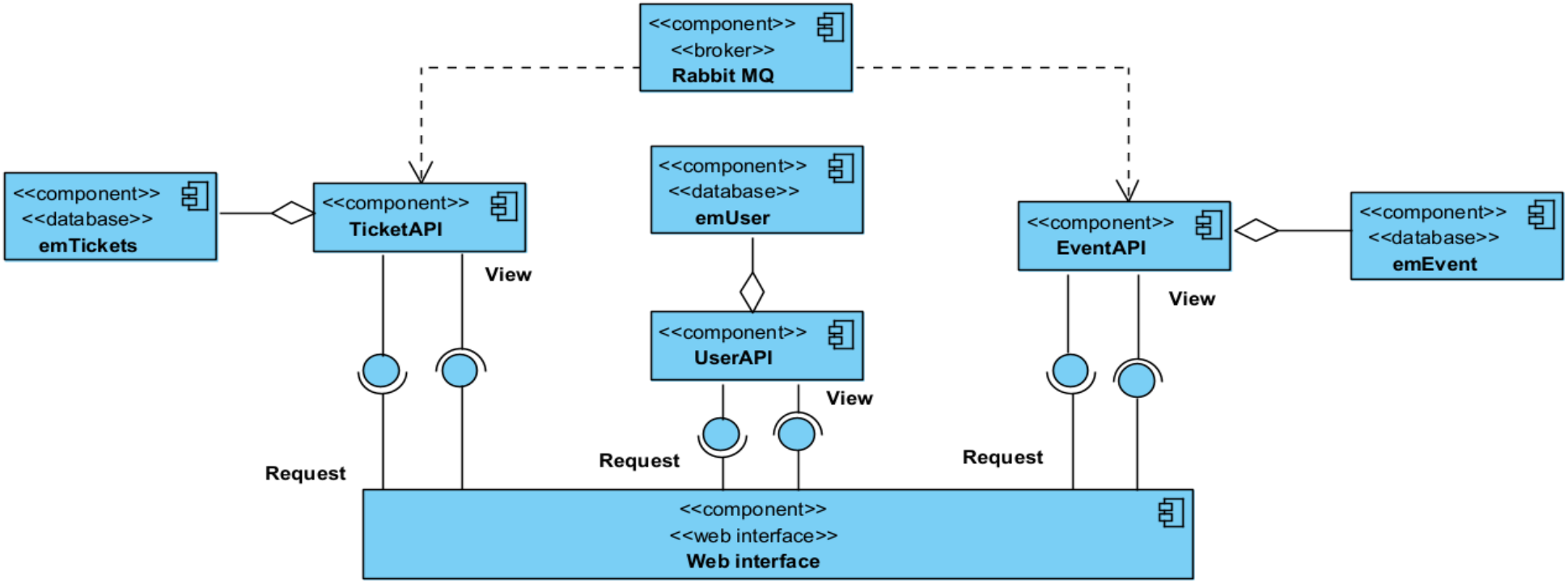

Рисунок А.12 – Диаграмма компонентов серверного приложения

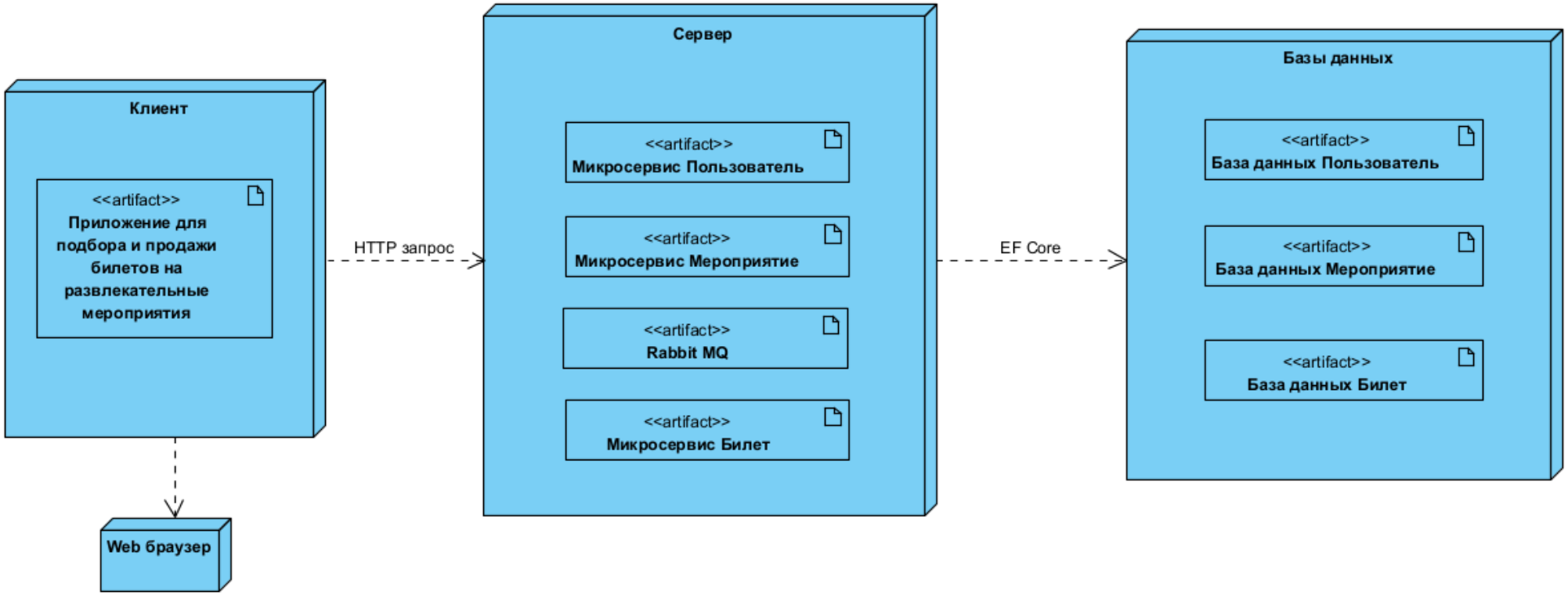

Рисунок А.13 – Диаграмма развертывания

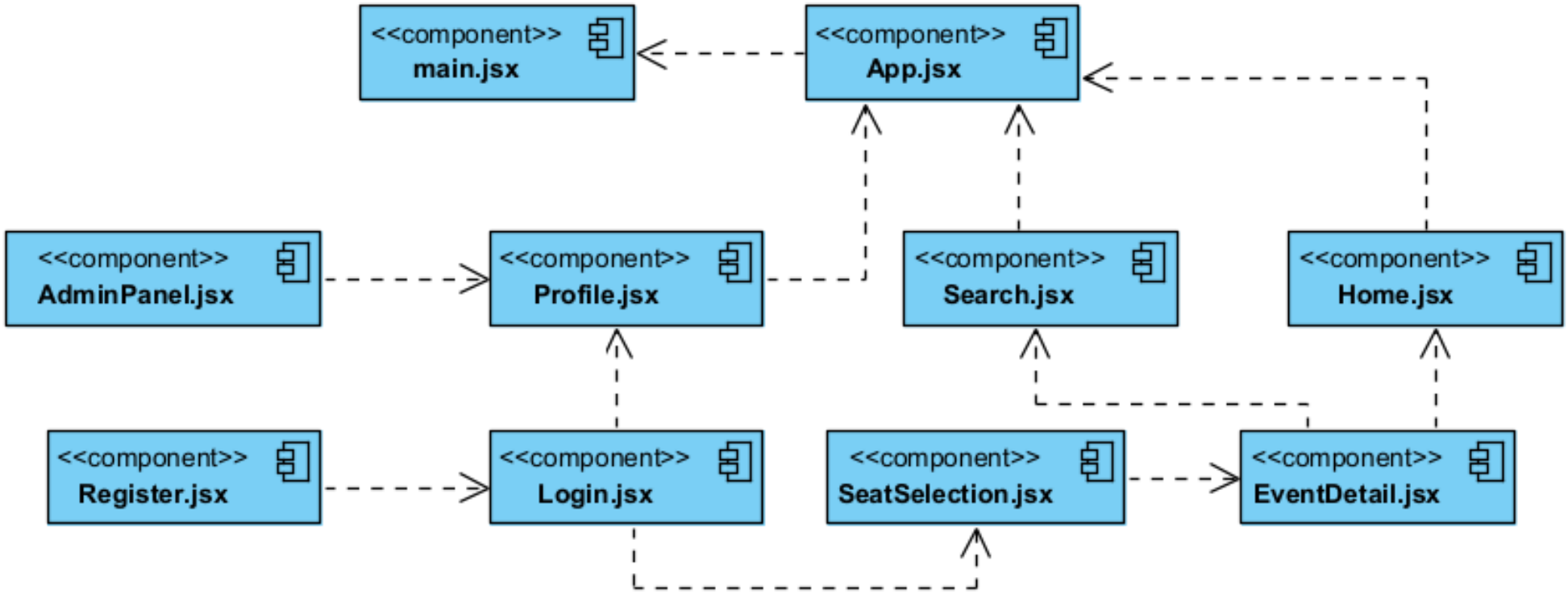

Рисунок А.14 – Диаграмма компонентов клиентской части

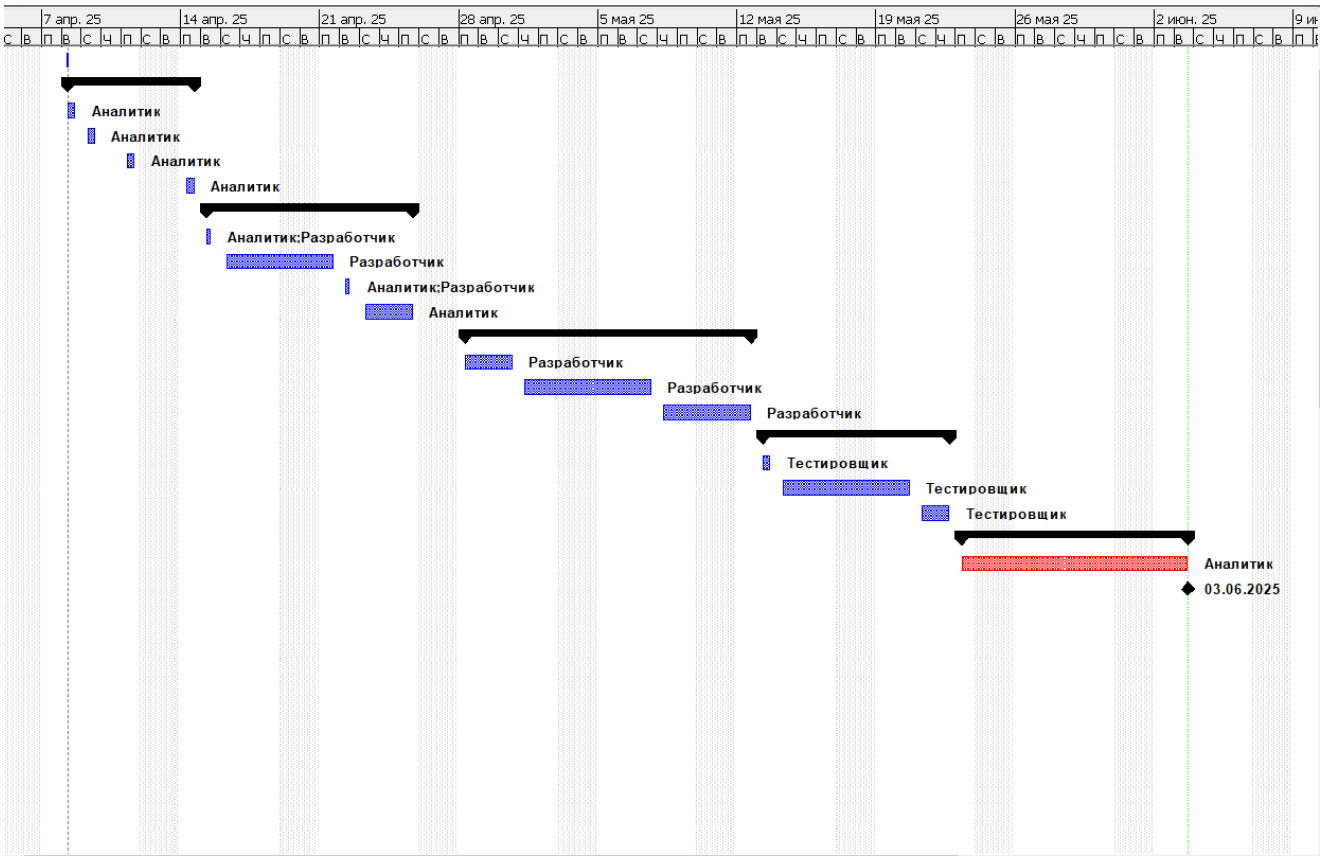

Рисунок А.15 – Диаграмма Ганта

Приложение Б

(обязательное)

Листинг серверной части приложения

Листинг Б.1 – Файл EventRepository.cs

```
using Microsoft.EntityFrameworkCore;
using EventApi.Models;
using EventAPI.Data;
using EventAPI.Models;

namespace EventApi.Data
{
    public class EventRepository : IEventRepository
    {
        private readonly EventsDbContext _context;

        public EventRepository(EventsDbContext context)
        {
            _context = context;
        }

        // EventTypes
        public async Task<EventType> GetEventTypeAsync(int id)
        {
            return await _context.EventTypes.FindAsync(id);
        }

        public async Task<List<EventType>> GetAllEventTypesAsync()
        {
            return await _context.EventTypes.ToListAsync();
        }

        public async Task<EventType> AddEventTypeAsync(EventType
eventType)
        {
            _context.EventTypes.Add(eventType);
            await _context.SaveChangesAsync();
            return eventType;
        }

        public async Task<EventType> UpdateEventTypeAsync(EventType
eventType)
        {
            _context.EventTypes.Update(eventType);
            await _context.SaveChangesAsync();
            return eventType;
        }

        public async Task<bool> DeleteEventTypeAsync(int id)
        {
            var eventType = await _context.EventTypes.FindAsync(id);
            if (eventType == null)
            {
                return false;
            }
            _context.EventTypes.Remove(eventType);
            await _context.SaveChangesAsync();
            return true;
        }
```

```
// Events
public async Task<Event> GetEventAsync(int id)
{
    return await _context.Events
        .Include(e => e.EventStatus)
        .Include(e => e.EventType)
        .FirstOrDefaultAsync(e => e.Id == id);
}

public async Task<List<Event>> GetAllEventsAsync()
{
    return await _context.Events
        .Include(e => e.EventStatus)
        .Include(e => e.EventType)
        .ToListAsync();
}

public async Task<Event> AddEventAsync(Event eventItem)
{
    _context.Events.Add(eventItem);
    await _context.SaveChangesAsync();
    return eventItem;
}

public async Task<Event> UpdateEventAsync(Event eventItem)
{
    _context.Events.Update(eventItem);
    await _context.SaveChangesAsync();
    return eventItem;
}

public async Task<bool> DeleteEventAsync(int id)
{
    var eventItem = await _context.Events.FindAsync(id);
    if (eventItem == null)
    {
        return false;
    }

    var eventSeats = await _context.EventSeats
        .Where(es => es.EventId == id)
        .ToListAsync();
    if (eventSeats.Any())
    {
        _context.EventSeats.RemoveRange(eventSeats);
    }

    _context.Events.Remove(eventItem);
    await _context.SaveChangesAsync();
    return true;
}

// EventSeats
public async Task<EventSeat> GetEventSeatAsync(int id)
{
    return await _context.EventSeats
        .Include(e => e.Event)
```

```
                .ThenInclude(e => e.EventStatus)
                .FirstOrDefaultAsync(e => e.Id == id);
        }

        public async Task<List<EventSeat>>
GetAllEventSeatsByEventIdAsync(int eventId)
        {
            return await _context.EventSeats
                .Where(es => es.EventId == eventId)
                .ToListAsync();
        }

        public async Task<EventSeat> AddEventSeatAsync(EventSeat
eventSeat)
        {
            _context.EventSeats.Add(eventSeat);
            await _context.SaveChangesAsync();
            return eventSeat;
        }

        public async Task<EventSeat> UpdateEventSeatAsync(EventSeat
eventSeat)
        {
            _context.EventSeats.Update(eventSeat);
            await _context.SaveChangesAsync();
            return eventSeat;
        }

        public async Task<bool> DeleteEventSeatAsync(int id)
        {
            var eventSeat = await _context.EventSeats.FindAsync(id);
            if (eventSeat == null)
            {
                return false;
            }
            _context.EventSeats.Remove(eventSeat);
            await _context.SaveChangesAsync();
            return true;
        }

        public async Task<List<Event>> SearchEventsAsync(EventSearchDto
searchDto)
        {
            var query = _context.Events
                .Include(e => e.EventStatus)
                .AsQueryable();


            if (!string.IsNullOrEmpty(searchDto.SearchString))
            {
                var searchLower = searchDto.SearchString.ToLower();
                query = query.Where(e =>
EF.Functions.Like(e.Name.ToLower(), $»%{searchLower}%») ||

EF.Functions.Like(e.Location.ToLower(), $»%{searchLower}%»));
            }
```

```
            if (searchDto.DateFrom.HasValue)
            {
                query = query.Where(e => e.EventDate >=
searchDto.DateFrom.Value);
            }


            if (searchDto.DateTo.HasValue)
            {
                query = query.Where(e => e.EventDate <=
searchDto.DateTo.Value);
            }


            if (searchDto.EventTypeId.HasValue)
            {
                query = query.Where(e => e.EventTypeId ==
searchDto.EventTypeId.Value);
            }


            if (searchDto.EventStatusId.HasValue)
            {
                query = query.Where(e => e.EventStatusId ==
searchDto.EventStatusId.Value);
            }


            if (!string.IsNullOrEmpty(searchDto.SortBy))
            {
                switch (searchDto.SortBy.ToLower())
                {
                    case «date»:
                        query = searchDto.SortDescending
                            ? query.OrderByDescending(e => e.EventDate)
                            : query.OrderBy(e => e.EventDate);
                        break;
                    case «price»:
                        query = searchDto.SortDescending
                            ? query.OrderByDescending(e =>
e.EventSeats.Min(es => es.Price))
                            : query.OrderBy(e => e.EventSeats.Min(es =>
es.Price));
                        break;
                    default:
                        query = query.OrderBy(e => e.EventDate);
                        break;
                }
            }
            else
            {
                query = query.OrderBy(e => e.EventDate);
            }

            return await query.ToListAsync();
        }
```

```
    }
}
```

Листинг Б.2 – Файл EventDbContext.cs

```
using EventApi.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

namespace EventApi.Data
{
    public class EventsDbContext: DbContext
    {
        public EventsDbContext(DbContextOptions<EventsDbContext>
options)
    : base(options)
        {
        }
        public DbSet<EventType> EventTypes { get; set; }
        public DbSet<Event> Events { get; set; }
        public DbSet<EventSeat> EventSeats { get; set; }
        public DbSet<EventStatus> EventStatus { get; set; }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {

            )
            modelBuilder.Entity<Event>()
                .HasOne(e => e.EventType)
                .WithMany() // Или .WithMany(et => et.Events),
если нужно обратное свойство в EventType
                .HasForeignKey(e => e.EventTypeId);

            modelBuilder.Entity<Event>()
                .HasOne(e => e.EventStatus)
                .WithMany()
                .HasForeignKey(e => e.EventStatusId);

            modelBuilder.Entity<EventSeat>()
                .HasOne(es => es.Event)
                .WithMany(e => e.EventSeats)
                .HasForeignKey(es => es.EventId);

            // это чтобы была уникальность
            modelBuilder.Entity<EventSeat>()
                .HasIndex(es => new { es.EventId, es.SeatNumber })
                .IsUnique();
        }
    }
}
```

Листинг Б.3 – Файл IEventRepository.cs

```
using EventApi.Models;
using EventAPI.Models;

namespace EventAPI.Data
{
    public interface IEventRepository
    {
        Task<EventType> GetEventTypeAsync(int id);
        Task<List<EventType>> GetAllEventTypesAsync();
        Task<EventType> AddEventTypeAsync(EventType eventType);
        Task<EventType> UpdateEventTypeAsync(EventType eventType);
        Task<bool> DeleteEventTypeAsync(int id);

        // Events
        Task<Event> GetEventAsync(int id);
        Task<List<Event>> GetAllEventsAsync();
        Task<Event> AddEventAsync(Event eventItem);
        Task<Event> UpdateEventAsync(Event eventItem);
        Task<bool> DeleteEventAsync(int id);
        Task<List<Event>> SearchEventsAsync(EventSearchDto searchDto);

        // EventSeats
        Task<EventSeat> GetEventSeatAsync(int id);
        Task<List<EventSeat>> GetAllEventSeatsByEventIdAsync(int
eventId);
        Task<EventSeat> AddEventSeatAsync(EventSeat eventSeat);
        Task<EventSeat> UpdateEventSeatAsync(EventSeat eventSeat);
        Task<bool> DeleteEventSeatAsync(int id);
    }
}
```

Листинг Б.4 – Файл IEventService.cs

```
using EventApi.Models;
using EventAPI.Models;

namespace EventApi.Services
{
    public interface IEventService
    {
        // EventTypes
        Task<EventType> GetEventTypeAsync(int id);
        Task<List<EventType>> GetAllEventTypesAsync();
        Task<EventType> CreateEventTypeAsync(CreateEventTypeDto
createEventTypeDto);
        Task<EventType> UpdateEventTypeAsync(int id,
UpdateEventTypeDto updateEventTypeDto);
        Task<bool> DeleteEventTypeAsync(int id);

        // Events
        Task<Event> GetEventAsync(int id);
        Task<List<Event>> GetAllEventsAsync();
```

```
        Task<Event> CreateEventAsync(CreateEventDto
createEventDto);
        Task<Event> UpdateEventAsync(int id, UpdateEventDto
updateEventDto);
        Task<bool> DeleteEventAsync(int id);

        // EventSeats
        Task<EventSeat> GetEventSeatAsync(int id);
        Task<List<EventSeat>> GetAllEventSeatsByEventIdAsync(int
eventId);
        Task<EventSeat> CreateEventSeatAsync(int eventId,
CreateEventSeatDto createEventSeatDto);
        Task<EventSeat> UpdateEventSeatAsync(int id,
UpdateEventSeatDto updateEventSeatDto);
        Task<bool> DeleteEventSeatAsync(int id);

        Task<List<Event>> SearchEventsAsync(EventSearchDto
searchDto);

        // сообщения

        Task<EventSeat> UpdateEventSeatByMessageAsync(string
message);
    }
}
```

Листинг Б.5 – Файл EventService.cs

```
using EventApi.Data;
using EventApi.Models;
using EventAPI.Data;
using EventAPI.Models;
using Microsoft.Extensions.DependencyInjection;
using System.Text.Json;
using System.Threading.Tasks;

namespace EventApi.Services
{
    public class EventService : IEventService
    {
        private readonly IEventRepository _eventRepository;
        private readonly ILogger<EventService> _logger;
        private readonly IServiceScopeFactory
_serviceScopeFactory;
        private readonly IWebHostEnvironment _environment;

        public EventService(
            IEventRepository eventRepository,
            ILogger<EventService> logger,
            IServiceScopeFactory serviceScopeFactory,
            IWebHostEnvironment environment)
        {
            _eventRepository = eventRepository;
            _logger = logger;
```

```
            _serviceScopeFactory = serviceScopeFactory;
            _environment = environment;
        }

        // EventTypes
        public async Task<EventType> GetEventTypeAsync(int id)
        {
            return await _eventRepository.GetEventTypeAsync(id);
        }

        public async Task<List<EventType>> GetAllEventTypesAsync()
        {
            return await _eventRepository.GetAllEventTypesAsync();
        }

        public async Task<EventType>
CreateEventTypeAsync(CreateEventTypeDto createEventTypeDto)
        {
            var eventType = new EventType { Name =
createEventTypeDto.Name };
            return await
_eventRepository.AddEventTypeAsync(eventType);
        }

        public async Task<EventType> UpdateEventTypeAsync(int id,
UpdateEventTypeDto updateEventTypeDto)
        {
            var eventType = await
_eventRepository.GetEventTypeAsync(id);
            if (eventType == null)
            {
                return null;
            }

            eventType.Name = updateEventTypeDto.Name ??
eventType.Name;
            return await
_eventRepository.UpdateEventTypeAsync(eventType);
        }

        public async Task<bool> DeleteEventTypeAsync(int id)
        {
            return await
_eventRepository.DeleteEventTypeAsync(id);
        }

        // Events
        public async Task<Event> GetEventAsync(int id)
        {
            return await _eventRepository.GetEventAsync(id);
        }
```

```
        public async Task<List<Event>> GetAllEventsAsync()
        {
            return await _eventRepository.GetAllEventsAsync();
        }

        public async Task<Event> CreateEventAsync(CreateEventDto
createEventDto)
        {
            var eventDate = createEventDto.EventDate.Value;
            eventDate = eventDate.AddSeconds(-
eventDate.Second).AddTicks(-(eventDate.Ticks %
TimeSpan.TicksPerSecond));

            var imagePath = await
SaveImageAsync(createEventDto.ImageFile,
_environment.WebRootPath);
            var eventItem = new Event
            {
                Name = createEventDto.Name,
                EventTypeId = createEventDto.EventTypeId,
                EventDate = eventDate,
                Description = createEventDto.Description,
                Location = createEventDto.Location,
                TotalSeats = createEventDto.TotalSeats,
                ImagePath = imagePath,
                EventStatusId = createEventDto.EventStatusId
            };
            return await
_eventRepository.AddEventAsync(eventItem);
        }

        public async Task<Event> UpdateEventAsync(int id,
UpdateEventDto updateEventDto)
        {
            var eventItem = await
_eventRepository.GetEventAsync(id);
            if (eventItem == null)
            {
                return null;
            }

            eventItem.Name = updateEventDto.Name ??
eventItem.Name;
            eventItem.EventTypeId = updateEventDto.EventTypeId ??
eventItem.EventTypeId;
            eventItem.EventDate = updateEventDto.EventDate ??
eventItem.EventDate;
            eventItem.Description = updateEventDto.Description ??
eventItem.Description;
            eventItem.Location = updateEventDto.Location ??
eventItem.Location;
```

```
            eventItem.TotalSeats = updateEventDto.TotalSeats ??
eventItem.TotalSeats;
            eventItem.EventStatusId = updateEventDto.EventStatusId
?? eventItem.EventStatusId;

            if (updateEventDto.ImageFile != null)
            {
                if (!string.IsNullOrEmpty(eventItem.ImagePath))
                {
                    var oldImagePath =
Path.Combine(_environment.WebRootPath,
eventItem.ImagePath.TrimStart('/'));
                    if (File.Exists(oldImagePath))
                    {
                        File.Delete(oldImagePath);
                    }
                }
                eventItem.ImagePath = await
SaveImageAsync(updateEventDto.ImageFile,
_environment.WebRootPath);
            }

            return await
_eventRepository.UpdateEventAsync(eventItem);
        }

        public async Task<bool> DeleteEventAsync(int id)
        {
            return await _eventRepository.DeleteEventAsync(id);
        }

        // EventSeats
        public async Task<EventSeat> GetEventSeatAsync(int id)
        {
            return await _eventRepository.GetEventSeatAsync(id);
        }

        public async Task<List<EventSeat>>
GetAllEventSeatsByEventIdAsync(int eventId)
        {
            return await
_eventRepository.GetAllEventSeatsByEventIdAsync(eventId);
        }

        public async Task<EventSeat> CreateEventSeatAsync(int
eventId, CreateEventSeatDto createEventSeatDto)
        {
            var eventSeat = new EventSeat
            {
                EventId = eventId,
                SeatNumber = createEventSeatDto.SeatNumber,
                Price = createEventSeatDto.Price
```

```
            };
            return await
_eventRepository.AddEventSeatAsync(eventSeat);
        }

        public async Task<EventSeat> UpdateEventSeatAsync(int id,
UpdateEventSeatDto updateEventSeatDto)
        {
            var eventSeat = await
_eventRepository.GetEventSeatAsync(id);
            if (eventSeat == null)
            {
                return null;
            }

            eventSeat.SeatNumber = updateEventSeatDto.SeatNumber
?? eventSeat.SeatNumber;
            eventSeat.IsReserved = updateEventSeatDto.IsReserved
?? false;
            eventSeat.Price = updateEventSeatDto.Price ??
eventSeat.Price;

            return await
_eventRepository.UpdateEventSeatAsync(eventSeat);
        }

        public async Task<EventSeat>
UpdateEventSeatByMessageAsync(string message)
        {
            var jsonDoc = JsonDocument.Parse(message);
            if (!jsonDoc.RootElement.TryGetProperty(«EventSeatId»,
out var idElement))
            {
                _logger.LogWarning(«в сообщении нет
EventSeatId!»);
                return null;
            }

            int eventSeatId = idElement.GetInt32();
            var updateEventSeatDto =
JsonSerializer.Deserialize<UpdateEventSeatDto>(message);

            var eventSeat = await
_eventRepository.GetEventSeatAsync(eventSeatId);
            if (eventSeat == null)
            {
                _logger.LogWarning($»EventSeat with id
'{eventSeatId}' not found.»);
                return null;
            }
```

```
            eventSeat.IsReserved = updateEventSeatDto.IsReserved
?? false;
            await
_eventRepository.UpdateEventSeatAsync(eventSeat);

            return await
_eventRepository.GetEventSeatAsync(eventSeatId);
        }

        public async Task<bool> DeleteEventSeatAsync(int id)
        {
            return await
_eventRepository.DeleteEventSeatAsync(id);
        }

        public async Task<List<Event>>
SearchEventsAsync(EventSearchDto searchDto)
        {
            return await
_eventRepository.SearchEventsAsync(searchDto);
        }

        private async Task<string?> SaveImageAsync(IFormFile?
imageFile, string webRootPath)
        {
            if (imageFile == null || imageFile.Length == 0)
            {
                return null;
            }

            if (string.IsNullOrEmpty(webRootPath))
            {
                throw new InvalidOperationException();
            }

            var allowedExtensions = new[] { «.jpg», «.jpeg»,
«.png», «.gif» };
            var extension =
Path.GetExtension(imageFile.FileName).ToLower();
            if (!allowedExtensions.Contains(extension))
            {
                throw new ArgumentException(«Недопустимый формат
изображения»);
            }


            var fileName = $»{Guid.NewGuid()}{extension}»;
            var folderPath = Path.Combine(webRootPath, «images»);
            var filePath = Path.Combine(folderPath, fileName);


            if (!Directory.Exists(folderPath))
```

```
            {
                Directory.CreateDirectory(folderPath);
            }


            using (var stream = new FileStream(filePath,
FileMode.Create))
            {
                await imageFile.CopyToAsync(stream);
            }


            return $»/images/{fileName}»;
        }
    }
}
```

Листинг Б.6 – Файл RabbitMQConsumerService.cs

```
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Extensions.DependencyInjection;
using EventAPI.Services;
using EventApi.Services;
using System.Text.Json;
using EventApi.Models;
using Azure;

namespace EventAPI.Services
{
    public class RabbitMQConsumerService : BackgroundService
    {
        private readonly ILogger<RabbitMQConsumerService> _logger;
        private readonly IServiceProvider _serviceProvider;
        private IModel _channel; // Канал для каждого сервиса

        public
RabbitMQConsumerService(ILogger<RabbitMQConsumerService> logger,
IServiceProvider serviceProvider)
        {
            _logger = logger ?? throw new
ArgumentNullException(nameof(logger));
            _serviceProvider = serviceProvider ?? throw new
ArgumentNullException(nameof(serviceProvider));
        }
```

```
        protected override async Task
ExecuteAsync(CancellationToken stoppingToken)
        {
            stoppingToken.ThrowIfCancellationRequested();


            using (var scope = _serviceProvider.CreateScope())
            {
                try
                {
                    var connection =
scope.ServiceProvider.GetRequiredService<IConnection>();
                    _channel = connection.CreateModel();

                    _channel.QueueDeclare(queue: «seat-updates-
queue»,
                                          durable: true,
                                exclusive: false,
                                autoDelete: false,
                                arguments: null);
                    var consumer = new
EventingBasicConsumer(_channel);

                    consumer.Received += async (model, ea) =>
                    {
                        byte[] body = ea.Body.ToArray();
                        string message =
Encoding.UTF8.GetString(body);

                        _logger.LogInformation($»Сообщение:
{message}»);

                        // Использовать другой scope для
EventService, так как это другой поток
                        using (var nestedScope =
_serviceProvider.CreateScope())
                        {
                            try
                            {

                                var eventService =
nestedScope.ServiceProvider.GetRequiredService<IEventService>();

                                EventSeat result = await
eventService.UpdateEventSeatByMessageAsync(message);

_logger.LogInformation(«UpdateEventSeatByMessageAsync returned:
{Result}», result);

                                //ответ
                                var replyMessage = new { Status =
«OK», Result = result };
```

```
                                _logger.LogInformation(«Созданное
ответное сообщение: {ReplyMessage}», replyMessage);
                                var replyBody =
Encoding.UTF8.GetBytes(JsonSerializer.Serialize(replyMessage));
                                if (ea.BasicProperties?.ReplyTo !=
null)
                                {
                                    var replyProps =
_channel.CreateBasicProperties();
                                    replyProps.CorrelationId =
ea.BasicProperties.CorrelationId;

_channel.BasicPublish(exchange: ««,
                                       routingKey:
ea.BasicProperties.ReplyTo,
                                       basicProperties:
replyProps,
                                       body: replyBody);

_logger.LogInformation($»Отправлено ответное
сообщение{ea.BasicProperties.ReplyTo} « +
                                        $»with correlationId
{ea.BasicProperties.CorrelationId}»);
                                }

                                _channel.BasicAck(ea.DeliveryTag,
false);  //
                            }
                            catch (Exception ex)
                            {
                                _logger.LogError($»Ошибка:
{ex.Message}»);
                                _channel.BasicNack(ea.DeliveryTag,
false, false);
                            }
                        }
                    };

                    _channel.BasicConsume(queue: «seat-updates-
queue»,
                                          autoAck: false,
                                          consumer: consumer);

                    _logger.LogInformation(«Прослушивание
сообщений...»);

                    await Task.CompletedTask;
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, «Ошибка RabbitMQ»);
```

```
                }
            }


            await Task.Delay(Timeout.Infinite, stoppingToken); //
Keep the service running
        }

        public override Task StopAsync(CancellationToken
stoppingToken)
        {
            _logger.LogInformation(«RabbitMQ Consumer Service
stopping»);
            _channel?.Close();
            return base.StopAsync(stoppingToken);
        }


    }
}
```

Листинг Б.7 – Файл EventsController.cs

```
using EventApi.Models;
using EventApi.Services;
using EventAPI.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace EventApi.Controllers
{
    [Route(«api/»)]
    [ApiController]
    public class EventsController : Controller
    {
        private readonly IEventService _eventService;

        public EventsController(IEventService eventService)
        {
            _eventService = eventService;
        }

        // EventTypes
        [HttpGet(«eventtype/{id}»)]
        public async Task<ActionResult<EventType>>
GetEventType(int id)
        {
            var eventType = await
_eventService.GetEventTypeAsync(id);

            if (eventType == null)
            {
```

```
                return NotFound();
            }

            return eventType;
        }

        [HttpGet(«eventtype»)]
        public async Task<ActionResult<IEnumerable<EventType>>>
GetEventTypes()
        {
            var eventTypes = await
_eventService.GetAllEventTypesAsync();
            return Ok(eventTypes);
        }

        [HttpPost(«eventtype»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<ActionResult<EventType>>
CreateEventType(CreateEventTypeDto createEventTypeDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var eventType = await
_eventService.CreateEventTypeAsync(createEventTypeDto);

            return CreatedAtAction(nameof(GetEventType), new { id
= eventType.Id }, eventType);
        }

        [HttpPut(«eventtype/{id}»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<IActionResult> UpdateEventType(int id,
UpdateEventTypeDto updateEventTypeDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var eventType = await
_eventService.UpdateEventTypeAsync(id, updateEventTypeDto);

            if (eventType == null)
            {
                return NotFound();
            }

            return NoContent();
        }
```

```
        [HttpDelete(«eventtype/{id}»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<IActionResult> DeleteEventType(int id)
        {
            var result = await
_eventService.DeleteEventTypeAsync(id);

            if (!result)
            {
                return NotFound();
            }

            return NoContent();
        }

        // Events
        [HttpGet(«event/{id}»)]
        public async Task<ActionResult<Event>> GetEvent(int id)
        {
            var eventItem = await _eventService.GetEventAsync(id);

            if (eventItem == null)
            {
                return NotFound();
            }

            eventItem.EventDate =
DateTime.Parse(eventItem.EventDate.Value.ToString(«dd.MM.yyyy
HH:mm»));


            return eventItem;
        }

        [HttpGet(«event»)]
        public async Task<ActionResult<IEnumerable<Event>>>
GetEvents()
        {
            var events = await _eventService.GetAllEventsAsync();
            foreach (var e in events)
            {
                if (e.EventDate.HasValue)
                {
                    e.EventDate =
DateTime.Parse(e.EventDate.Value.ToString(«dd.MM.yyyy HH:mm»));
                }
            }
            return Ok(events);
        }

        [HttpPost(«event»)]
```

```
        [Authorize(Roles = «Administrator»)]
        public async Task<ActionResult<Event>>
CreateEvent([FromForm] CreateEventDto createEventDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var eventItem = await
_eventService.CreateEventAsync(createEventDto);

            return CreatedAtAction(nameof(GetEvent), new { id =
eventItem.Id }, eventItem);
        }

        [HttpPut(«event/{id}»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<IActionResult> UpdateEvent(int id,
[FromForm] UpdateEventDto updateEventDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var eventItem = await
_eventService.UpdateEventAsync(id, updateEventDto);

            if (eventItem == null)
            {
                return NotFound();
            }

            return Ok(eventItem);
        }

        [HttpDelete(«event/{id}»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<IActionResult> DeleteEvent(int id)
        {
            var result = await _eventService.DeleteEventAsync(id);

            if (!result)
            {
                return NotFound();
            }

            return NoContent();
        }

        // EventSeats
```

```
        [HttpGet(«eventseat/{id}»)]
        public async Task<ActionResult<EventSeat>>
GetEventSeat(int id)
        {
            var eventSeat = await
_eventService.GetEventSeatAsync(id);

            if (eventSeat == null)
            {
                return NotFound();
            }

            return eventSeat;
        }

        [HttpGet(«{eventId}/eventseat»)]
        public async Task<ActionResult<IEnumerable<EventSeat>>>
GetEventSeatsByEventId(int eventId)
        {
            var eventSeats = await
_eventService.GetAllEventSeatsByEventIdAsync(eventId);
            return Ok(eventSeats);
        }


        [HttpPost(«{eventId}/eventseat»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<ActionResult<EventSeat>>
CreateEventSeat(int eventId, [FromBody] CreateEventSeatDto
createEventSeatDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var eventSeat = await
_eventService.CreateEventSeatAsync(eventId, createEventSeatDto);

            return CreatedAtAction(nameof(GetEventSeat), new { id
= eventSeat.Id }, eventSeat);
        }

        [HttpPut(«eventseat/{id}»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<ActionResult<UpdateEventSeatDto>>
UpdateEventSeat(int id, UpdateEventSeatDto updateEventSeatDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
```

```
            var eventSeat = await
_eventService.UpdateEventSeatAsync(id, updateEventSeatDto);

            if (eventSeat == null)
            {
                return NotFound();
            }

            return updateEventSeatDto;
        }

        [HttpDelete(«eventseat/{id}»)]
        [Authorize(Roles = «Administrator»)]
        public async Task<IActionResult> DeleteEventSeat(int id)
        {
            var result = await
_eventService.DeleteEventSeatAsync(id);

            if (!result)
            {
                return NotFound();
            }

            return NoContent();
        }

        [HttpGet(«search»)]
        public async Task<IActionResult> SearchEvents([FromQuery]
EventSearchDto searchDto)
        {
            var events = await
_eventService.SearchEventsAsync(searchDto);
            return Ok(events);
        }
    }
}
```

Листинг Б.8 – Файл IUserRepository.cs

```
using UserAPI.Models;

namespace UserAPI.Data
{
    public interface IUserRepository
    {
        Task<User> GetUserByEmailAsync(string email);
        Task<User> GetUserByIdAsync(int id);
        Task<bool> UserExistsAsync(string email);
        Task<User> AddUserAsync(User user);
        Task<User> UpdateUserAsync(User user);
        Task SaveChangesAsync();
    }
```

```
}
```

Листинг Б.9 – Файл UserRepository.cs

```
using Microsoft.EntityFrameworkCore;
using UserAPI.Models;

namespace UserAPI.Data
{
    public class UserRepository: IUserRepository
    {
        private readonly UsersDbContext _context;

        public UserRepository(UsersDbContext context)
        {
            _context = context;
        }

        public async Task<User> GetUserByEmailAsync(string email)
        {
            return await _context.Users.Include(u => u.UserType)
                .FirstOrDefaultAsync(u => u.Email == email);
        }

        public async Task<User> GetUserByIdAsync(int id)
        {
            return await _context.Users.Include(u => u.UserType)
                .FirstOrDefaultAsync(u => u.Id == id);
        }

        public async Task<bool> UserExistsAsync(string email)
        {
            return await _context.Users.AnyAsync(u => u.Email ==
email);
        }

        public async Task<User> AddUserAsync(User user)
        {
            _context.Users.Add(user);
            await _context.SaveChangesAsync();
            return user;
        }

        public async Task<User> UpdateUserAsync(User user)
        {
            _context.Users.Update(user);
            await _context.SaveChangesAsync();
            return user;
        }

        public async Task SaveChangesAsync()
        {
            await _context.SaveChangesAsync();
```

```
        }
    }
}
```

Листинг Б.10 – Файл UsersDbContext.cs

```
using UserAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace UserAPI.Data
{
    public class UsersDbContext : DbContext
    {
        public UsersDbContext(DbContextOptions<UsersDbContext>
options) : base(options)
        {
        }

        public DbSet<User> Users { get; set; }
        public DbSet<UserType> UserTypes { get; set; }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.Entity<User>()
               .HasOne(u => u.UserType)
               .WithMany()
               .HasForeignKey(u => u.UserTypeId);
        }
    }
}
```

Листинг Б.11 – Файл IUsersService.cs

```
using UserAPI.Models;

namespace UserAPI.Services
{
    public interface IUsersService
    {
        Task<User> RegisterAsync(string email, string password,
string name, string phone);
        Task<User> GetUserByIdAsync(int id);
        Task<User> GetUserByEmailAsync(string email);
        Task<bool> VerifyPasswordAsync(string email, string
password);
        Task<User> UpdateUserNameAsync(int userId, string
newName);
        Task<bool> UpdateUserPasswordAsync(int userId, string
oldPassword, string newPassword);
        Task<string> GenerateJwtTokenAsync(User user);
    }
}
```

Листинг Б.12 – Файл UsersService.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using UserAPI.Data;
using BCrypt.Net;
using UserAPI.Models;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace UserAPI.Services
{
    public class UsersService : IUsersService
    {
        private readonly IUserRepository _userRepository;
        private readonly IConfiguration _configuration;
        public UsersService(IUserRepository userRepository,
IConfiguration configuration)
        {
            _userRepository = userRepository;
            _configuration = configuration;
        }
        public async Task<User> GetUserByEmailAsync(string email)
        {
            return await
_userRepository.GetUserByEmailAsync(email);
        }

        public async Task<User> GetUserByIdAsync(int id)
        {
            return await _userRepository.GetUserByIdAsync(id);
        }

        public async Task<User> RegisterAsync(string email, string
password, string name, string phone)
        {
            if (await _userRepository.UserExistsAsync(email))
            {
                return null;
            }

            var newUser = new User
            {
                Email = email,
                Name = name,
                Phone = phone,
                UserTypeId = 2,
                Password =
BCrypt.Net.BCrypt.HashPassword(password)
            };
```

```
            return await _userRepository.AddUserAsync(newUser);
        }

        public async Task<bool> VerifyPasswordAsync(string email,
string password)
        {
            var user = await
_userRepository.GetUserByEmailAsync(email);

            if (user == null)
            {
                return false;
            }

            return BCrypt.Net.BCrypt.Verify(password,
user.Password);
        }

        public async Task<User> UpdateUserNameAsync(int userId,
string newName)
        {
            var user = await
_userRepository.GetUserByIdAsync(userId);

            if (user == null)
            {
                return null;
            }

            user.Name = newName;
            return await _userRepository.UpdateUserAsync(user);
        }

        public async Task<bool> UpdateUserPasswordAsync(int
userId, string oldPassword, string newPassword)
        {
            var user = await
_userRepository.GetUserByIdAsync(userId);

            if (user == null ||
!BCrypt.Net.BCrypt.Verify(oldPassword, user.Password))
            {
                return false;
            }

            user.Password =
BCrypt.Net.BCrypt.HashPassword(newPassword);
            await _userRepository.UpdateUserAsync(user);
            return true;
        }

        public async Task<string> GenerateJwtTokenAsync(User user)
```

```
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key =
Encoding.UTF8.GetBytes(_configuration[«Jwt:Key»]);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                {
                new Claim(ClaimTypes.NameIdentifier,
user.Id.ToString()),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.Role, user.UserTypeId == 1 ?
«Administrator» : «User»)
            }),
                Expires = DateTime.UtcNow.AddHours(1),
                Issuer = _configuration[«Jwt:Issuer»],
                Audience = _configuration[«Jwt:Audience»],
                SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
    }
}
```

Листинг Б.13 – Файл UsersController.cs

```
using UserAPI.Models;
using UserAPI.Services;
using Microsoft.AspNetCore.Mvc;

namespace UserAPI.Controllers
{
    [Route(«api/[controller]»)]
    [ApiController]
    public class UsersController : Controller
    {
        private readonly IUsersService _usersService;
        public UsersController(IUsersService usersService)
        {
            _usersService = usersService;
        }

        [HttpPost(«register»)]
        public async Task<ActionResult<User>> Register([FromBody]
UserRegisterRequest model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
```

```
            var newUser = await
_usersService.RegisterAsync(model.Email, model.Password,
model.Name, model.Phone);

            if (newUser == null)
            {
                return Conflict($»User with email '{model.Email}'
already exists.»);
            }
            return CreatedAtAction(nameof(GetUser), new { id =
newUser.Id }, newUser);
        }

        [HttpPost(«login»)]
        public async Task<IActionResult> Login([FromBody]
UserLoginRequest model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var user = await
_usersService.GetUserByEmailAsync(model.Email);
            if (user == null)
            {
                return Unauthorized(«Неправильная почта или
пароль.»);
            }

            bool isPasswordValid = await
_usersService.VerifyPasswordAsync(model.Email, model.Password);
            if (!isPasswordValid)
            {
                return Unauthorized(«Неправильная почта или
пароль.»);
            }

            var token = await
_usersService.GenerateJwtTokenAsync(user);
            return Ok(new { Token = token, Id = user.Id, Email =
user.Email, Name = user.Name });
        }


        [HttpGet(«{id}»)]
        public async Task<ActionResult<UserResponse>> GetUser(int
id)
        {
            var user = await _usersService.GetUserByIdAsync(id);
```

```
            if (user == null)
            {
                return NotFound();
            }

            var response = new UserResponse
            {
                Id = user.Id,
                Email = user.Email,
                Phone = user.Phone,
                Name = user.Name,
                UserTypeId = user.UserTypeId,
                UserType = user.UserType,
            };

            return response;
        }

        [HttpPut(«{id}/name»)]
        public async Task<ActionResult<User>> UpdateUserName(int
id, [FromBody] UpdateUserNameRequest model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var updatedUser = await
_usersService.UpdateUserNameAsync(id, model.NewName);

            if (updatedUser == null)
            {
                return NotFound();
            }

            return Ok(new { Message = «Имя успешно изменено» });
        }

        [HttpPut(«{id}/password»)]
        public async Task<ActionResult> UpdateUserPassword(int id,
[FromBody] UpdateUserPasswordRequest model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            bool passwordUpdated = await
_usersService.UpdateUserPasswordAsync(id, model.OldPassword,
model.NewPassword);

            if (!passwordUpdated)
```

```
            {
                return Unauthorized(«Неверный старый пароль или
пользователь не найден.»);
            }

            return Ok(new { Message = «Пароль успешно изменен» });
        }
    }
}
```

Листинг Б.14 – Файл ITicketRepository.cs

```
using TicketAPI.Models;

namespace TicketAPI.Data
{
    public interface ITicketRepository
    {
        Task<Ticket> GetTicketByIdAsync(int ticketId);
        Task<Ticket> GetTicketByUserAndEventSeatAsync(int userId,
int eventSeatId);
        Task<List<Ticket>> GetTicketsByUserIdAsync(int userId);
        Task<Ticket> AddTicketAsync(Ticket ticket);
        Task<Ticket> UpdateTicketAsync(Ticket ticket);
        Task SaveChangesAsync();
    }
}
```

Листинг Б.15 – Файл TicketRepository.cs

```
using Microsoft.EntityFrameworkCore;
using TicketAPI.Models;

namespace TicketAPI.Data
{
    public class TicketRepository: ITicketRepository
    {
        private readonly TicketsDbContext _dbContext;

        public TicketRepository(TicketsDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public async Task<Ticket> GetTicketByIdAsync(int ticketId)
        {
            return await _dbContext.Tickets.FindAsync(ticketId);
        }

        public async Task<Ticket>
GetTicketByUserAndEventSeatAsync(int userId, int eventSeatId)
        {
            return await _dbContext.Tickets
```

```
                .FirstOrDefaultAsync(t => t.UserId == userId &&
t.EventSeatId == eventSeatId);
        }

        public async Task<List<Ticket>>
GetTicketsByUserIdAsync(int userId)
        {
            return await _dbContext.Tickets
                .Where(t => t.UserId == userId)
                .Include(t => t.Status)
                .ToListAsync();
        }

        public async Task<Ticket> AddTicketAsync(Ticket ticket)
        {
            _dbContext.Tickets.Add(ticket);
            await _dbContext.SaveChangesAsync();
            return ticket;
        }

        public async Task<Ticket> UpdateTicketAsync(Ticket ticket)
        {
            _dbContext.Tickets.Update(ticket);
            await _dbContext.SaveChangesAsync();
            return ticket;
        }

        public async Task SaveChangesAsync()
        {
            await _dbContext.SaveChangesAsync();
        }
    }
}
```

Листинг Б.16 – Файл TicketsDbContext.cs

```
using TicketAPI.Models;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Reflection.Emit;
namespace TicketAPI.Data
{
    public class TicketsDbContext : DbContext
    {
        public TicketsDbContext(DbContextOptions<TicketsDbContext>
options)
: base(options)
        {
        }

        public DbSet<Ticket> Tickets { get; set; }
        public DbSet<TicketStatus> TicketStatus { get; set; }
```

```
        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.Entity<Ticket>()
                .HasOne(t => t.Status)
                .WithMany()
                .HasForeignKey(t => t.StatusId);

            modelBuilder.Entity<TicketStatus>()
                .HasIndex(ts => ts.Name)
                .IsUnique();
        }
    }
}
```

Листинг Б.17 – Файл ITicketService.cs

```
using TicketAPI.Models;

namespace TicketAPI.Services
{
    public interface ITicketService
    {
        Task<Ticket> PurchaseTicketAsync(int userId, int
eventSeatId);
        Task<Ticket> BookTicketAsync(int userId, int eventSeatId);
        Task<Ticket> CancelPurchaseAsync(int ticketId, int
userId);
        Task<List<Ticket>> GetTicketsByUserIdAsync(int userId);
        Task<Ticket> CancelReservationAsync(int ticketId, int
userId);
        Task<Ticket> PurchaseBookedTicketAsync(int ticketId, int
userId);


    }
}
```

Листинг Б.18 – Файл RabbitMQService.cs

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;
using TicketAPI.Models;

namespace TicketAPI.Services
{
    public class RabbitMQService : IMessageBroker
    {
        private readonly IModel _channel;
```

```
        private readonly TimeSpan _timeout =
TimeSpan.FromSeconds(30); // Таймаут 30 секунд

        public RabbitMQService(IConnection connection)
        {
            _channel = connection.CreateModel();
        }

        public async Task<T> SendMessageAsync<T>(string exchange,
string routingKey, object message)
        {
            var updateQueue = _channel.QueueDeclare().QueueName;
            var correlationId = Guid.NewGuid().ToString();
            var props = _channel.CreateBasicProperties();
            props.CorrelationId = correlationId;
            props.ReplyTo = updateQueue;
            props.Persistent = true;

            // Отправка сообщения
            var body =
Encoding.UTF8.GetBytes(JsonSerializer.Serialize(message));
            _channel.BasicPublish(exchange, routingKey, false,
props, body);

            // Ожидание ответа с таймаутом
            var consumer = new EventingBasicConsumer(_channel);
            var tcs = new TaskCompletionSource<string>();
            consumer.Received += (model, ea) =>
            {
                if (ea.BasicProperties.CorrelationId ==
correlationId)
                {

tcs.TrySetResult(Encoding.UTF8.GetString(ea.Body.ToArray()));
                }
            };
            _channel.BasicConsume(consumer, updateQueue, true);

            // Задаём таймаут
            var timeoutTask = Task.Delay(_timeout);
            var completedTask = await Task.WhenAny(tcs.Task,
timeoutTask);

            if (completedTask == timeoutTask)
            {
                // Таймаут сработал — очищаем очередь и
выбрасываем исключение
                _channel.QueueDelete(updateQueue, false, false);
                throw new TimeoutException(«Сервис недоступен:
ответ не получен в течение 30 секунд.»);
            }
```

```
            // Получаем результат, если ответ пришёл вовремя
            var responseMessage = await tcs.Task;
            _channel.QueueDelete(updateQueue, false, false);

            var options = new JsonSerializerOptions {
PropertyNameCaseInsensitive = true };
            var response =
JsonSerializer.Deserialize<ResponseMessage>(responseMessage,
options);
            return
JsonSerializer.Deserialize<T>(JsonSerializer.Serialize(response.Re
sult), options);
        }
    }
}
```

Листинг Б.19 – Файл TicketService.cs

```
using Microsoft.Extensions.Logging;
using TicketAPI.Models;
using TicketAPI.Data;
using System.Threading.Tasks;

namespace TicketAPI.Services
{
    public class TicketService : ITicketService
    {
        private readonly ILogger<TicketService> _logger;
        private readonly ITicketRepository _ticketRepository;
        private readonly IMessageBroker _messageBroker;

        public TicketService(ILogger<TicketService> logger,
ITicketRepository ticketRepository, IMessageBroker messageBroker)
        {
            _logger = logger;
            _ticketRepository = ticketRepository;
            _messageBroker = messageBroker;
        }

        public async Task<Ticket> PurchaseTicketAsync(int userId,
int eventSeatId)
        {
            if (await
_ticketRepository.GetTicketByUserAndEventSeatAsync(userId,
eventSeatId) != null)
            {
                return null;
            }

            var message = new { EventSeatId = eventSeatId,
IsReserved = true };
            EventSeat eventSeat;
```

```
            try
            {
                eventSeat = await
_messageBroker.SendMessageAsync<EventSeat>(«event-seat-exchange»,
«seat.reserved», message);
            }
            catch (TimeoutException ex)
            {
                _logger.LogWarning(«Не удалось зарезервировать
место: {Message}», ex.Message);
                return null;
            }

            var ticket = new Ticket
            {
                UserId = userId,
                EventName = eventSeat.Event.Name,
                EventSeatId = eventSeatId,
                SeatNumber = eventSeat.SeatNumber,
                EventDate = eventSeat.Event.EventDate ??
DateTime.MinValue,
                Price = eventSeat.Price,
                StatusId = 1,
                PurchaseDate = DateTime.Now,
                ReservationDate = null
            };

            return await _ticketRepository.AddTicketAsync(ticket);
        }

        public async Task<Ticket> BookTicketAsync(int userId, int
eventSeatId)
        {
            if (await
_ticketRepository.GetTicketByUserAndEventSeatAsync(userId,
eventSeatId) != null)
            {
                return null;
            }

            var message = new { EventSeatId = eventSeatId,
IsReserved = true };
            EventSeat eventSeat;

            try
            {
                eventSeat = await
_messageBroker.SendMessageAsync<EventSeat>(«event-seat-exchange»,
«seat.reserved», message);
            }
            catch (TimeoutException ex)
            {
```

```
                _logger.LogWarning(«Не удалось зарезервировать
место: {Message}», ex.Message);
                return null;
            }

            var ticket = new Ticket
            {
                UserId = userId,
                EventName = eventSeat.Event.Name,
                EventSeatId = eventSeatId,
                SeatNumber = eventSeat.SeatNumber,
                EventDate = eventSeat.Event.EventDate ??
DateTime.MinValue,
                Price = eventSeat.Price,
                StatusId = 2,
                PurchaseDate = null,
                ReservationDate = DateTime.Now
            };

            return await _ticketRepository.AddTicketAsync(ticket);
        }

        public async Task<Ticket> CancelPurchaseAsync(int
ticketId, int userId)
        {
            var ticket = await
_ticketRepository.GetTicketByIdAsync(ticketId);
            if (ticket == null || ticket.UserId != userId ||
ticket.PurchaseDate == null)
            {
                return null;
            }

            ticket.StatusId = 4;
            await _ticketRepository.UpdateTicketAsync(ticket);

            var message = new { EventSeatId = ticket.EventSeatId,
IsReserved = false };
            EventSeat eventSeat;

            try
            {
                eventSeat = await
_messageBroker.SendMessageAsync<EventSeat>(«event-seat-exchange»,
«seat.reserved», message);
            }
            catch (TimeoutException ex)
            {
                _logger.LogWarning(«Не удалось отменить покупку:
{Message}», ex.Message);
                return null;
            }
```

```
            return ticket;
        }

        public async Task<Ticket> CancelReservationAsync(int
ticketId, int userId)
        {
            var ticket = await
_ticketRepository.GetTicketByIdAsync(ticketId);
            if (ticket == null || ticket.UserId != userId ||
ticket.PurchaseDate != null || ticket.ReservationDate == null)
            {
                return null;
            }

            ticket.StatusId = 3;
            await _ticketRepository.UpdateTicketAsync(ticket);

            var message = new { EventSeatId = ticket.EventSeatId,
IsReserved = false };
            EventSeat eventSeat;

            try
            {
                eventSeat = await
_messageBroker.SendMessageAsync<EventSeat>(«event-seat-exchange»,
«seat.reserved», message);
            }
            catch (TimeoutException ex)
            {
                _logger.LogWarning(«Не удалось отменить бронь:
{Message}», ex.Message);

            }
            return ticket;
        }

            public async Task<List<Ticket>>
GetTicketsByUserIdAsync(int userId)
            {
                return await
_ticketRepository.GetTicketsByUserIdAsync(userId);
            }

            public async Task<Ticket>
PurchaseBookedTicketAsync(int ticketId, int userId)
            {
                var ticket = await
_ticketRepository.GetTicketByIdAsync(ticketId);
                if (ticket == null || ticket.UserId != userId ||
ticket.PurchaseDate != null || ticket.ReservationDate == null)
                {
```

```
                    return null;
                }

                ticket.PurchaseDate = DateTime.Now;
                ticket.StatusId = 1;
                return await
_ticketRepository.UpdateTicketAsync(ticket);
            }
        }
    }
```

Листинг Б.20 – Файл IMessageBroker.cs

```
namespace TicketAPI.Services
{
    public interface IMessageBroker
    {
        Task<T> SendMessageAsync<T>(string exchange, string
routingKey, object message);
    }
}
```

Листинг Б.21 – Файл TicketController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TicketAPI.Models;
using TicketAPI.Services;

namespace TicketAPI.Controllers
{
    [ApiController]
    [Route(«api/tickets»)]
    public class TicketController : Controller
    {
        private readonly ITicketService _ticketService;
        public TicketController(ITicketService ticketService)
        {
            _ticketService = ticketService;
        }

        [HttpPost(«purchase»)]
        [Authorize(Roles = «Administrator,User»)]
        public async Task<IActionResult> PurchaseTicket([FromBody]
PurchaseRequest request)
        {
            try
            {
                var ticket = await
_ticketService.PurchaseTicketAsync(request.UserId,
request.EventSeatId);
                if (ticket == null)
                {
```

```
                    return BadRequest(«Не удалось купить билет:
место уже занято, недоступно или возникла ошибка в сервисе.»);
                }
                return Ok(ticket);
            }
            catch (TimeoutException)
            {
                return StatusCode(503, «Сервис недоступен:
попробуйте позже.»);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $»Произошла ошибка на
сервере: {ex.Message}»);
            }
        }

        [HttpPost(«reservation»)]
        [Authorize(Roles = «Administrator,User»)]
        public async Task<IActionResult> BookTicket([FromBody]
PurchaseRequest request)
        {
            try
            {
                var ticket = await
_ticketService.BookTicketAsync(request.UserId,
request.EventSeatId);
                if (ticket == null)
                {
                    return BadRequest(«Не удалось забронировать
билет: место уже занято или сервис недоступен.»);
                }
                return Ok(ticket);
            }
            catch (TimeoutException)
            {
                return StatusCode(503, «Сервис недоступен:
попробуйте позже.»);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $»Произошла ошибка на
сервере: {ex.Message}»);
            }
        }

        [HttpPost(«cancel/purchase»)]
        [Authorize(Roles = «Administrator,User»)]
        public async Task<IActionResult> CancelPurchase([FromBody]
CancelRequest request)
        {
            try
```

```
            {
                var ticket = await
_ticketService.CancelPurchaseAsync(request.TicketId,
request.UserId);
                if (ticket == null)
                {
                    return BadRequest(«Не удалось отменить
покупку: билет не найден или не был куплен.»);
                }
                return Ok(ticket);
            }
            catch (TimeoutException)
            {
                return StatusCode(503, «Сервис недоступен:
попробуйте позже.»);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $»Произошла ошибка на
сервере: {ex.Message}»);
            }
        }

        [HttpPost(«cancel/reservation»)]
        [Authorize(Roles = «Administrator,User»)]
        public async Task<IActionResult>
CancelReservation([FromBody] CancelRequest request)
        {
            try
            {
                var ticket = await
_ticketService.CancelReservationAsync(request.TicketId,
request.UserId);
                if (ticket == null)
                {
                    return BadRequest(«Не удалось отменить
бронирование: билет не найден или не был забронирован.»);
                }
                return Ok(ticket);
            }
            catch (TimeoutException)
            {
                return StatusCode(503, «Сервис недоступен:
попробуйте позже.»);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $»Произошла ошибка на
сервере: {ex.Message}»);
            }
        }
```

```
        [HttpGet(«{userId}»)]
        [Authorize(Roles = «Administrator,User»)]
        public async Task<ActionResult<List<Ticket>>>
GetTicketsByUserId(int userId)
        {
            var tickets = await
_ticketService.GetTicketsByUserIdAsync(userId);

            if (tickets == null)
            {
                return StatusCode(500);
            }

            if (tickets.Count == 0)
            {
                return NotFound(«Не приобретено ни одного
билета»);
            }

            return Ok(tickets);
        }

        [HttpPost(«purchaseBooked»)]
        [Authorize(Roles = «Administrator,User»)]
        public async Task<ActionResult<Ticket>>
PurchaseBookedTicket([FromBody] CancelRequest request)
        {
            try
            {
                var ticket = await
_ticketService.PurchaseBookedTicketAsync(request.TicketId,
request.UserId);
                if (ticket == null)
                {
                    return BadRequest(«Не удалось купить билет,
так как он уже куплен или не забронирован.»);
                }
                return Ok(ticket);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $»Произошла ошибка на
сервере: {ex.Message}»);
            }
        }
    }
}
```

Листинг Б.22 – Файл IUserRepository.cs

```
using UserAPI.Models;

namespace UserAPI.Data
```

```
{
    public interface IUserRepository
    {
        Task<User> GetUserByEmailAsync(string email);
        Task<User> GetUserByIdAsync(int id);
        Task<bool> UserExistsAsync(string email);
        Task<User> AddUserAsync(User user);
        Task<User> UpdateUserAsync(User user);
        Task SaveChangesAsync();
    }
}
```

Листинг Б.23 – Файл UserRepository.cs

```
using Microsoft.EntityFrameworkCore;
using UserAPI.Models;

namespace UserAPI.Data
{
    public class UserRepository: IUserRepository
    {
        private readonly UsersDbContext _context;

        public UserRepository(UsersDbContext context)
        {
            _context = context;
        }

        public async Task<User> GetUserByEmailAsync(string email)
        {
            return await _context.Users.Include(u => u.UserType)
                .FirstOrDefaultAsync(u => u.Email == email);
        }

        public async Task<User> GetUserByIdAsync(int id)
        {
            return await _context.Users.Include(u => u.UserType)
                .FirstOrDefaultAsync(u => u.Id == id);
        }

        public async Task<bool> UserExistsAsync(string email)
        {
            return await _context.Users.AnyAsync(u => u.Email ==
email);
        }

        public async Task<User> AddUserAsync(User user)
        {
            _context.Users.Add(user);
            await _context.SaveChangesAsync();
            return user;
        }
```

```
        public async Task<User> UpdateUserAsync(User user)
        {
            _context.Users.Update(user);
            await _context.SaveChangesAsync();
            return user;
        }

        public async Task SaveChangesAsync()
        {
            await _context.SaveChangesAsync();
        }
    }
}
```

Листинг Б.24 – Файл UsersDbContext.cs

```
using UserAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace UserAPI.Data
{
    public class UsersDbContext : DbContext
    {
        public UsersDbContext(DbContextOptions<UsersDbContext>
options) : base(options)
        {
        }

        public DbSet<User> Users { get; set; }
        public DbSet<UserType> UserTypes { get; set; }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.Entity<User>()
               .HasOne(u => u.UserType)
               .WithMany()
               .HasForeignKey(u => u.UserTypeId);
        }
    }
}
```

Листинг Б.25 – Файл IUsersService.cs

```
using UserAPI.Models;

namespace UserAPI.Services
{
    public interface IUsersService
    {
        Task<User> RegisterAsync(string email, string password,
string name, string phone);
        Task<User> GetUserByIdAsync(int id);
```

```
        Task<User> GetUserByEmailAsync(string email);
        Task<bool> VerifyPasswordAsync(string email, string
password);
        Task<User> UpdateUserNameAsync(int userId, string
newName);
        Task<bool> UpdateUserPasswordAsync(int userId, string
oldPassword, string newPassword);
        Task<string> GenerateJwtTokenAsync(User user);
    }
}
```

Листинг Б.26 – Файл .cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using UserAPI.Data;
using BCrypt.Net;
using UserAPI.Models;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace UserAPI.Services
{
    public class UsersService : IUsersService
    {
        private readonly IUserRepository _userRepository;
        private readonly IConfiguration _configuration;
        public UsersService(IUserRepository userRepository,
IConfiguration configuration)
        {
            _userRepository = userRepository;
            _configuration = configuration;
        }
        public async Task<User> GetUserByEmailAsync(string email)
        {
            return await
_userRepository.GetUserByEmailAsync(email);
        }

        public async Task<User> GetUserByIdAsync(int id)
        {
            return await _userRepository.GetUserByIdAsync(id);
        }

        public async Task<User> RegisterAsync(string email, string
password, string name, string phone)
        {
            if (await _userRepository.UserExistsAsync(email))
            {
                return null;
            }
```

```
            var newUser = new User
            {
                Email = email,
                Name = name,
                Phone = phone,
                UserTypeId = 2,
                Password =
BCrypt.Net.BCrypt.HashPassword(password)
            };

            return await _userRepository.AddUserAsync(newUser);
        }

        public async Task<bool> VerifyPasswordAsync(string email,
string password)
        {
            var user = await
_userRepository.GetUserByEmailAsync(email);

            if (user == null)
            {
                return false;
            }

            return BCrypt.Net.BCrypt.Verify(password,
user.Password);
        }

        public async Task<User> UpdateUserNameAsync(int userId,
string newName)
        {
            var user = await
_userRepository.GetUserByIdAsync(userId);

            if (user == null)
            {
                return null;
            }

            user.Name = newName;
            return await _userRepository.UpdateUserAsync(user);
        }

        public async Task<bool> UpdateUserPasswordAsync(int
userId, string oldPassword, string newPassword)
        {
            var user = await
_userRepository.GetUserByIdAsync(userId);

            if (user == null ||
!BCrypt.Net.BCrypt.Verify(oldPassword, user.Password))
```

```
            {
                return false;
            }

            user.Password =
BCrypt.Net.BCrypt.HashPassword(newPassword);
            await _userRepository.UpdateUserAsync(user);
            return true;
        }

        public async Task<string> GenerateJwtTokenAsync(User user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key =
Encoding.UTF8.GetBytes(_configuration[«Jwt:Key»]);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                {
                new Claim(ClaimTypes.NameIdentifier,
user.Id.ToString()),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.Role, user.UserTypeId == 1 ?
«Administrator» : «User»)
            }),
                Expires = DateTime.UtcNow.AddHours(1),
                Issuer = _configuration[«Jwt:Issuer»],
                Audience = _configuration[«Jwt:Audience»],
                SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
    }
}
```

Листинг Б.27 – Файл UsersController.cs

```
using UserAPI.Models;
using UserAPI.Services;
using Microsoft.AspNetCore.Mvc;

namespace UserAPI.Controllers
{
    [Route(«api/[controller]»)]
    [ApiController]
    public class UsersController : Controller
    {
        private readonly IUsersService _usersService;
        public UsersController(IUsersService usersService)
        {
```

```
            _usersService = usersService;
        }

        [HttpPost(«register»)]
        public async Task<ActionResult<User>> Register([FromBody]
UserRegisterRequest model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var newUser = await
_usersService.RegisterAsync(model.Email, model.Password,
model.Name, model.Phone);

            if (newUser == null)
            {
                return Conflict($»User with email '{model.Email}'
already exists.»);
            }
            return CreatedAtAction(nameof(GetUser), new { id =
newUser.Id }, newUser);
        }

        [HttpPost(«login»)]
        public async Task<IActionResult> Login([FromBody]
UserLoginRequest model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var user = await
_usersService.GetUserByEmailAsync(model.Email);
            if (user == null)
            {
                return Unauthorized(«Invalid email or password.»);
            }

            bool isPasswordValid = await
_usersService.VerifyPasswordAsync(model.Email, model.Password);
            if (!isPasswordValid)
            {
                return Unauthorized(«Invalid email or password.»);
            }

            var token = await
_usersService.GenerateJwtTokenAsync(user);
            return Ok(new { Token = token, Id = user.Id, Email =
user.Email, Name = user.Name });
```

```
        }


        [HttpGet(«{id}»)]
        public async Task<ActionResult<UserResponse>> GetUser(int
id)
        {
            var user = await _usersService.GetUserByIdAsync(id);

            if (user == null)
            {
                return NotFound();
            }

            var response = new UserResponse
            {
                Id = user.Id,
                Email = user.Email,
                Phone = user.Phone,
                Name = user.Name,
                UserTypeId = user.UserTypeId,
                UserType = user.UserType,
            };

            return response;
        }

        [HttpPut(«{id}/name»)]
        public async Task<ActionResult<User>> UpdateUserName(int
id, [FromBody] UpdateUserNameRequest model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var updatedUser = await
_usersService.UpdateUserNameAsync(id, model.NewName);

            if (updatedUser == null)
            {
                return NotFound();
            }

            return Ok(new { Message = «Имя успешно изменено» });
        }

        [HttpPut(«{id}/password»)]
        public async Task<ActionResult> UpdateUserPassword(int id,
[FromBody] UpdateUserPasswordRequest model)
        {
            if (!ModelState.IsValid)
```

```
            {
                return BadRequest(ModelState);
            }

            bool passwordUpdated = await
_usersService.UpdateUserPasswordAsync(id, model.OldPassword,
model.NewPassword);

            if (!passwordUpdated)
            {
                return Unauthorized(«Неверный старый пароль или
пользователь не найден.»);
            }

            return Ok(new { Message = «Пароль успешно изменен» });
        }
    }
}
```

Приложение В

(обязательное)

Листинг клиентской части приложения

Листинг В.1 – Файл main.jsx

```
// src/main.jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Листинг В.2 – Файл App.jsx

```
import { BrowserRouter, Routes, Route, Link, Navigate } from
'react-router-dom';
import Home from './components/Home.jsx';
import EventDetail from './components/EventDetail.jsx';
import SeatSelection from './components/SeatSelection';
import Login from './components/Login.jsx';
import Register from './components/Register.jsx';
import Profile from './components/Profile.jsx';
import Search from './components/Search.jsx';
import './App.css';

function App() {
  const isAuthenticated = !!localStorage.getItem('token');

  return (
    <BrowserRouter>
      <nav className=«navbar navbar-expand-lg navbar-light bg-
light»>
        <div className=«container»>
          <div className=«navbar-nav»>
            <Link className=«nav-link» to=«/»>Главная</Link>
            <Link className=«nav-link» to=«/search»>Поиск</Link>
            <Link
              className=«nav-link»
              to={isAuthenticated ? '/profile' : '/login'}
            >
              Личный кабинет
            </Link>
          </div>
        </div>
      </nav>
      <Routes>
        <Route path=«/» element={<Home />} />
        <Route path=«/event/:id» element={<EventDetail />} />
        <Route path=«/event/:id/seats» element={<SeatSelection />}
/>
        <Route
```

```
          path=«/profile»
          element={isAuthenticated ? <Profile /> : <Navigate
to=«/login» />}
        />
        <Route path=«/login» element={<Login />} />
        <Route path=«/register» element={<Register />} />
        <Route path=«/search» element={<Search />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

Листинг В.3 – Файл Home.jsx

```
import { useState, useEffect } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';

const Home = () => {
  const [events, setEvents] = useState([]);
  const [eventTypes, setEventTypes] = useState([]);

  useEffect(() => {
    fetchEvents();
    fetchEventTypes();
  }, []);

  const fetchEvents = async () => {
    try {
      const response = await
axios.get('https://localhost:7109/api/event');
      console.log('Полученные мероприятия:', response.data);
      setEvents(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке мероприятий:', err);
    }
  };

  const fetchEventTypes = async () => {
    try {
      const response = await
axios.get('https://localhost:7109/api/eventtype');
      setEventTypes(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке типов мероприятий:',
err);
    }
  };

  const getEventsByType = (typeId) => {
    const filteredEvents = events
```

```
      .filter(event => event.eventType?.id === typeId)
      .sort((a, b) => {
        const parseDate = (dateStr) => {
          const [datePart, timePart] = dateStr.split('  ');
          const [day, month, year] = datePart.split('.');
          const [hours, minutes] = timePart.split(':');
          return new Date(`${year}-${month}-
${day}T${hours}:${minutes}:00`);
        };
        const dateA = parseDate(a.eventDate);
        const dateB = parseDate(b.eventDate);
        console.log(`Сортировка: ${a.name} (${dateA}) vs ${b.name}
(${dateB})`);
        return dateB - dateA;
      })
      .slice(0, 3);
    console.log(`Мероприятия для типа ${typeId}:`,
filteredEvents);
    return filteredEvents;
  };

  return (
    <div className=«container mt-4»>
      <style>
        {`
          .event-card {
            transition: transform 0.3s ease, filter 0.3s ease;
          }
          .event-card:hover {
            transform: scale(1.05);
            filter: brightness(90%);
          }
          .event-image {
            height: 200px;
            object-fit: cover;
          }
        `}
      </style>
      <h1 className=«mb-4» >Самые свежие мероприятия по
категориям</h1>
      {/* <p>Выберите мероприятие из списка ниже:</p> */}
      {eventTypes.length === 0 ? (
        <p>Загрузка категорий...</p>
      ) : (
        eventTypes.map(type => (
          <div key={type.id} className=«mb-5»>
            <h3>{type.name}</h3>
            {getEventsByType(type.id).length === 0 ? (
              <p>Нет мероприятий в этой категории.</p>
            ) : (
              <div className=«row»>
                {getEventsByType(type.id).map(event => (
```

```
                  <div key={event.id} className=«col-md-4 mb-3»>
                    <div className=«card event-card»>
                      {event.imagePath && (
                        <img
src={`https://localhost:7109${event.imagePath}`}
                          alt={event.name}
                          className=«card-img-top event-image»
                        />
                      )}
                      <div className=«card-body»>
                        <h5 className=«card-
title»>{event.name}</h5>
                        <p className=«card-text»>
                          <strong>Дата:</strong>
{event.eventDate}<br />
                          <strong>Место:</strong> {event.location}
                        </p>
                        <Link
                          to={`/event/${event.id}`}
                          state={{ from: '/' }}
                          className=«btn btn-primary»
                        >
                          Подробнее
                        </Link>
                      </div>
                    </div>
                  </div>
                ))}
              </div>
            )}
          </div>
        ))
      )}
    </div>
  );
};

export default Home;
```

Листинг В.4 – Файл EventDetail.jsx

```
import { useState, useEffect } from 'react';
import { useParams, Link, useLocation, useNavigate } from 'react-
router-dom';
import axios from 'axios';

const EventDetail = () => {
  const { id } = useParams();
  const [event, setEvent] = useState(null);
  const [showNotification, setShowNotification] = useState(false);
  const location = useLocation();
  const navigate = useNavigate();
```

```
  useEffect(() => {
    fetchEvent();
  }, [id]);

  const fetchEvent = async () => {
    try {
      const response = await
axios.get(`https://localhost:7109/api/event/${id}`);
      setEvent(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке мероприятия:', err);
    }
  };

  if (!event) return <div className=«container mt-
5»>Загрузка...</div>;

  const parseEventDate = (dateStr) => {
    const [datePart, timePart] = dateStr.split('  ');
    const [day, month, year] = datePart.split('.');
    const [hours, minutes] = timePart.split(':');
    return new Date(`${year}-${month}-
${day}T${hours}:${minutes}:00`);
  };

  const isTypeRestricted = event.eventType?.id === 2 ||
event.eventType?.id === 3;
  const eventDate = parseEventDate(event.eventDate);
  const today = new Date();
  today.setHours(0, 0, 0, 0);
  const isDatePast = eventDate < today;

  const isBuyDisabled = isTypeRestricted || isDatePast;

  const prevPath = location.state?.from || '/';
  const isFromSearch = prevPath === '/search';

  const handleBuyClick = (e) => {
    e.preventDefault();
    if (isBuyDisabled) {
      setShowNotification(true);
      setTimeout(() => setShowNotification(false), 3000);
    } else {
      navigate(`/event/${id}/seats`);
    }
  };

  const handleBackClick = () => {
    if (isFromSearch) {
      navigate('/search', { state: { fromEventDetail: true } });
    } else {
```

```
      navigate(prevPath);
    }
  };

  return (
    <div className=«container mt-4»>
      {showNotification && (
        <div className=«alert alert-warning» role=«alert»>
          Покупка билетов на мероприятие недоступна
        </div>
      )}
      <h1 className=«mb-4»>{event.name}</h1>
      <div className=«row»>
        <div className=«col-md-6»>
          {event.imagePath ? (
            <img
              src={`https://localhost:7109${event.imagePath}`}
              alt={event.name}
              className=«img-fluid mb-3»
              style={{ maxHeight: '300px', objectFit: 'cover' }}
            />
          ) : (
            <p>Изображение отсутствует</p>
          )}
        </div>
        <div className=«col-md-6»>
          <p><strong>Тип мероприятия:</strong>
{event.eventType?.name || 'Не указан'}</p>
          <p><strong>Дата:</strong> {event.eventDate}</p>
          <p><strong>Описание:</strong> {event.description}</p>
          <p><strong>Место:</strong> {event.location}</p>
          {/* <p><strong>Всего мест:</strong>
{event.totalSeats}</p> */}
          <button
            className=«btn btn-success»
            onClick={handleBuyClick}
            disabled={isBuyDisabled}
          >
            Купить билет
          </button>
          <button
            className=«btn btn-secondary ms-2»
            onClick={handleBackClick}
          >
            Назад
          </button>
        </div>
      </div>
    </div>
  );
};
```

```
export default EventDetail;
```

Листинг В.5 – Файл SeatSelection.jsx

```
import { useState, useEffect } from 'react';
import { useParams, Link } from 'react-router-dom';
import axios from 'axios';

const SeatSelection = () => {
  const { id } = useParams();
  const [event, setEvent] = useState(null);
  const [seats, setSeats] = useState([]);
  const [selectedSeat, setSelectedSeat] = useState(null);
  const [showModal, setShowModal] = useState(false);
  const [modalAction, setModalAction] = useState('');
  const [notification, setNotification] = useState('');
  const [isAdmin, setIsAdmin] = useState(false);
  const [userId, setUserId] = useState(null);

  useEffect(() => {
    const user = JSON.parse(localStorage.getItem('user'));
    const storedUserId = localStorage.getItem('userId');
    const token = localStorage.getItem('token');

    if (user && storedUserId && token) {
      setUserId(parseInt(storedUserId));
      const userTypeId = user.userType?.id || 0;
      setIsAdmin(userTypeId === 1);
    } else {
      setUserId(null);
    }

    fetchEvent();
    fetchSeats();
  }, [id]);

  const fetchEvent = async () => {
    try {
      const response = await
axios.get(`https://localhost:7109/api/event/${id}`);
      setEvent(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке мероприятия:', err);
    }
  };

  const fetchSeats = async () => {
    try {
      const response = await
axios.get(`https://localhost:7109/api/${id}/eventseat`);
      setSeats(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке мест:', err);
```

```
    }
  };

  const handleSeatSelect = (seat) => {
    if (seat.isReserved) return;
    setSelectedSeat(seat.id === selectedSeat ? null : seat.id);
  };

  const handlePurchase = async () => {
    const token = localStorage.getItem('token');
    if (!userId || !token) {
      window.location.href = '/login';
      return;
    }
    if (!selectedSeat) {
      setNotification('Пожалуйста, выберите место.');
      setTimeout(() => setNotification(''), 3000);
      return;
    }
    console.log('Отправка запроса на покупку:', { userId,
eventSeatId: selectedSeat });
    try {
      const response = await axios.post(
        'https://localhost:7112/api/tickets/purchase',
        {
          userId,
          eventSeatId: selectedSeat,
        },
        {
          headers: {
            Authorization: `Bearer ${token}`,
            'Content-Type': 'application/json',
          },
        }
      );
      if (response.status === 200) {
        setModalAction('purchase');
        setShowModal(true);
      }
    } catch (err) {
      console.error('Ошибка при покупке билета:', err);
      const errorMessage =
        err.response?.status === 400
          ? err.response?.data || 'Не удалось купить билет: место
уже занято или недоступно.'
          : err.response?.status === 401
          ? (window.location.href = '/login')
          : err.response?.status === 503
          ? 'Сервис недоступен. Попробуйте позже.'
          : 'Ошибка при взаимодействии с сервером. Проверьте
соединение или попробуйте позже.';
      if (typeof errorMessage === 'string') {
```

```
        setNotification(errorMessage);
        setTimeout(() => setNotification(''), 3000);
      }
    }
  };

  const handleReservation = async () => {
    const token = localStorage.getItem('token');
    if (!userId || !token) {
      window.location.href = '/login';
      return;
    }
    if (!selectedSeat) {
      setNotification('Пожалуйста, выберите место.');
      setTimeout(() => setNotification(''), 3000);
      return;
    }
    console.log('Отправка запроса на бронирование:', { userId,
eventSeatId: selectedSeat });
    try {
      const response = await axios.post(
        'https://localhost:7112/api/tickets/reservation',
        {
          userId,
          eventSeatId: selectedSeat,
        },
        {
          headers: {
            Authorization: `Bearer ${token}`,
            'Content-Type': 'application/json',
          },
        }
      );
      if (response.status === 200) {
        setModalAction('reservation');
        setShowModal(true);
      }
    } catch (err) {
      console.error('Ошибка при бронировании билета:', err);
      const errorMessage =
        err.response?.status === 400
          ? err.response?.data || 'Не удалось забронировать билет:
место уже занято или недоступно.'
          : err.response?.status === 401
          ? (window.location.href = '/login')
          : err.response?.status === 503
          ? 'Сервис недоступен. Попробуйте позже.'
          : 'Ошибка при взаимодействии с сервером. Проверьте
соединение или попробуйте позже.';
      if (typeof errorMessage === 'string') {
        setNotification(errorMessage);
        setTimeout(() => setNotification(''), 3000);
```

```
      }
    }
  };

  const closeModal = () => {
    setShowModal(false);
    setNotification(modalAction === 'purchase' ? 'Билет успешно
куплен' : 'Билет успешно забронирован');
    setTimeout(() => setNotification(''), 3000);
    setSelectedSeat(null);
    fetchSeats();
  };

  if (!event) return <div className=«container mt-
5»>Загрузка...</div>;

  return (
    <div className=«container mt-5»>
      <style>
        {`
          .seat-card {
            transition: transform 0.3s ease, filter 0.3s ease,
background-color 0.3s ease;
          }
          .seat-card:hover {
            transform: scale(1.05);
            filter: brightness(90%);
          }
          .seat-card.selected {
            background-color: #e6ffe6;
          }
          .seat-card {
            cursor: pointer;
          }
          .seat-card.disabled {
            cursor: not-allowed;
            opacity: 0.6;
          }
          .seat-card.disabled:hover {
            transform: none;
            filter: none;
          }
          .modal-overlay {
            position: fixed;
            top: 0;
            left: 0;
            right: 0;
            bottom: 0;
            background: rgba(0, 0, 0, 0.5);
            display: flex;
            justify-content: center;
            align-items: center;
```

```
            z-index: 1000;
          }
          .modal-content {
            background: white;
            padding: 20px;
            border-radius: 8px;
            position: relative;
            width: 90%;
            max-width: 500px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
          }
          .modal-close {
            position: absolute;
            top: 10px;
            right: 10px;
            background: none;
            border: none;
            font-size: 24px;
            cursor: pointer;
          }
          .notification {
            position: fixed;
            bottom: 20px;
            right: 20px;
            background-color: #28a745;
            color: white;
            padding: 10px 20px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
            z-index: 1000;
            animation: fadeInOut 3s ease-in-out forwards;
          }
          @keyframes fadeInOut {
            0% { opacity: 0; transform: translateY(20px); }
            10% { opacity: 1; transform: translateY(0); }
            90% { opacity: 1; transform: translateY(0); }
            100% { opacity: 0; transform: translateY(20px); }
          }
        `}
      </style>

      {showModal && (
        <div className=«modal-overlay»>
          <div className=«modal-content»>
            <button className=«modal-close» onClick={closeModal}>
              ×
            </button>
            <h3>Интеграция с платежным сервисом</h3>
            <p>
              На данный момент функционал покупки и бронирования
билетов находится в разработке. Интеграция с платежным сервисом
будет добавлена в ближайшее время. Благодарим за понимание!
```

```
            </p>
          </div>
        </div>
      )}

      {notification && (
        <div className=«notification»>
          {notification}
        </div>
      )}

      <h1>Выбор места для «{event.name}»</h1>
      <p><strong>Дата:</strong> {event.eventDate}</p>
      <p><strong>Место:</strong> {event.location}</p>
      <h3>Доступные места:</h3>
      {seats.length === 0 ? (
        <p>Места отсутствуют.</p>
      ) : (
        <div className=«row»>
          {seats.map(seat => {
            const isAvailable = !seat.isReserved;
            const isSelected = seat.id === selectedSeat;
            return (
              <div key={seat.id} className=«col-md-3 mb-3»>
                <div
                  className={`card text-center ${isAvailable ?
'seat-card' : 'seat-card disabled'} ${isSelected ? 'selected' :
''}`}
                  onClick={() => handleSeatSelect(seat)}
                >
                  <div className=«card-body»>
                    <h6 className=«card-title»>Место:
{seat.seatNumber}</h6>
                    <p className=«card-text»>
                      <strong>Цена:</strong> {seat.price} руб.<br
/>
                      <strong>Статус:</strong> {isAvailable ?
'Доступно' : 'Занято'}
                    </p>
                  </div>
                </div>
              </div>
            );
          })}
        </div>
      )}
      {selectedSeat && !isAdmin ? (
        <div className=«mt-3»>
          <p>Вы выбрали место: {seats.find(s => s.id ===
selectedSeat)?.seatNumber}</p>
          <button className=«btn btn-primary me-2»
onClick={handlePurchase}>
```

```
            Купить билет
          </button>
          <button className=«btn btn-secondary»
onClick={handleReservation}>
            Забронировать билет
          </button>
        </div>
      ) : isAdmin ? (
        <div className=«mt-3»>
          <p className=«text-danger»>Администраторам недоступны
покупка и бронирование билетов.</p>
        </div>
      ) : null}
      <div className=«mt-3 mb-3»>
        <Link to={`/event/${id}`} className=«btn btn-secondary»>
          Назад к мероприятию
        </Link>
      </div>
    </div>
  );
};

export default SeatSelection;
```

Листинг В.6 – Файл Login.jsx

```
// src/components/Login.jsx
import { useState } from 'react';
import axios from 'axios';
import { Link, useNavigate } from 'react-router-dom';

const Login = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [showPassword, setShowPassword] = useState(false);
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await
axios.post('https://localhost:7047/api/users/login', {
        email,
        password,
      });
      const { token, id } = response.data;
      localStorage.setItem('token', token);
      localStorage.setItem('userId', id);
      const userResponse = await
axios.get(`https://localhost:7047/api/users/${id}`, {
        headers: { Authorization: `Bearer ${token}` },
      });
```

```
      localStorage.setItem('user',
JSON.stringify(userResponse.data));
      navigate('/profile');
    } catch (err) {
      if (err.response) {
        setError('Неверный email или пароль');
      } else if (err.request) {
        setError('Ошибка сети');
      } else {
        setError('Произошла ошибка: ' + err.message);
      }
    }
  };

  return (
    <div className=«d-flex justify-content-center align-items-
center vh-100»>
      <div className=«col-md-4»>
        <h2 className=«text-center»>Вход</h2>
        {error && <div className=«alert alert-
danger»>{error}</div>}
        <form onSubmit={handleLogin}>
          <div className=«mb-3»>
            <label className=«form-label»>Email</label>
            <input
              type=«email»
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              className=«form-control»
              placeholder=«Email»
            />
          </div>
          <div className=«mb-3 position-relative»>
            <label className=«form-label»>Пароль</label>
            <input
              type={showPassword ? 'text' : 'password'}
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              className=«form-control»
              placeholder=«Пароль»
            />
            <button
              type=«button»
              className=«btn btn-outline-secondary position-
absolute»
              style={{
                marginTop: '-18px',
                right: '0.5rem',
                transform: 'translateY(-50%)',
                border: 'none',
              }}
              onClick={() => setShowPassword(!showPassword)}
```

```
            >
              {showPassword ? '🙈' : '👁'}
            </button>
          </div>
          <button type=«submit» className=«btn btn-primary w-100
mb-2»>Войти</button>
          <Link to=«/register» className=«btn btn-secondary w-
100»>Регистрация</Link>
        </form>
      </div>
    </div>
  );
};

export default Login;
```

Листинг В.7 – Файл Register.jsx

```
import { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Register = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [name, setName] = useState('');
  const [phone, setPhone] = useState('');
  const [error, setError] = useState('');
  const [showModal, setShowModal] = useState(false);
  const navigate = useNavigate();

  const handleRegister = async (e) => {
    e.preventDefault();
    try {
      await
axios.post('https://localhost:7047/api/users/register', {
        email,
        password,
        name,
        phone,
      });
      setShowModal(true);
    } catch (err) {
      if (err.response) {
        setError('Ошибка регистрации');
      } else if (err.request) {
        setError('Ошибка сети');
      } else {
        setError('Произошла ошибка: ' + err.message);
      }
    }
  };
```

```
  const closeModal = () => {
    setShowModal(false);
    navigate('/login');
  };

  return (
    <div className=«d-flex justify-content-center align-items-
center vh-100»>
      <div className=«col-md-4»>
        <h2 className=«text-center»>Регистрация</h2>
        {error && <div className=«alert alert-
danger»>{error}</div>}
        <form onSubmit={handleRegister}>
          <div className=«mb-3»>
            <label className=«form-label»>Email</label>
            <input
              type=«email»
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              className=«form-control»
              placeholder=«Email»
              required
            />
          </div>
          <div className=«mb-3»>
            <label className=«form-label»>Пароль</label>
            <input
              type=«password»
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              className=«form-control»
              placeholder=«Пароль»
              required
            />
          </div>
          <div className=«mb-3»>
            <label className=«form-label»>Имя</label>
            <input
              type=«text»
              value={name}
              onChange={(e) => setName(e.target.value)}
              className=«form-control»
              placeholder=«Имя»
              required
            />
          </div>
          <div className=«mb-3»>
            <label className=«form-label»>Телефон</label>
            <input
              type=«tel»
              value={phone}
              onChange={(e) => setPhone(e.target.value)}
```

```
              className=«form-control»
              placeholder=«Телефон»
            />
          </div>
          <button type=«submit» className=«btn btn-primary w-
100»>Зарегистрироваться</button>
        </form>
        <button
          type=«button»
          className=«btn btn-secondary w-100 mt-2»
          onClick={() => navigate('/login')}
        >
          Войти
        </button>

        {}
        {showModal && (
          <div className=«modal d-block» style={{ backgroundColor:
'rgba(0,0,0,0.5)' }}>
            <div className=«modal-dialog»>
              <div className=«modal-content»>
                <div className=«modal-header»>
                  <h5 className=«modal-title»>Успех</h5>
                </div>
                <div className=«modal-body»>
                  Регистрация прошла успешно!
                </div>
                <div className=«modal-footer»>
                  <button type=«button» className=«btn btn-
primary» onClick={closeModal}>
                    OK
                  </button>
                </div>
              </div>
            </div>
          </div>
        )}
      </div>
    </div>
  );
};

export default Register;
```

Листинг В.8 – Файл Profile.jsx

```
import { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import AdminPanel from './AdminPanel';

const Profile = () => {
  const [user, setUser] = useState(null);
```

```
  const [activeSection, setActiveSection] = useState('info');
  const [newName, setNewName] = useState('');
  const [oldPassword, setOldPassword] = useState('');
  const [newPassword, setNewPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [showOldPassword, setShowOldPassword] = useState(false);
  const [showNewPassword, setShowNewPassword] = useState(false);
  const [showConfirmPassword, setShowConfirmPassword] =
useState(false);
  const [notifications, setNotifications] = useState([]);
  const [tickets, setTickets] = useState([]);
  const [expandedTickets, setExpandedTickets] = useState({});
  const [statusFilter, setStatusFilter] = useState('');
  const [dateFilter, setDateFilter] = useState('');
  const [searchQuery, setSearchQuery] = useState('');
  const navigate = useNavigate();

  const addNotification = (message, type = 'success') => {
    const id = Date.now();
    setNotifications(prev => [...prev, { id, message, type }]);
    setTimeout(() => {
      setNotifications(prev => prev.filter(notif => notif.id !==
id));
    }, 5000);
  };

  useEffect(() => {
    const storedUser = localStorage.getItem('user');
    if (storedUser) {
      const parsedUser = JSON.parse(storedUser);
      setUser(parsedUser);
      setNewName(parsedUser.name);
      if (parsedUser.userTypeId !== 1) {
        fetchTickets(parsedUser.id);
      }
    } else {
      navigate('/login');
    }
  }, [navigate]);

  const fetchTickets = async (userId) => {
    try {
      const token = localStorage.getItem('token');
      const response = await
axios.get(`https://localhost:7112/api/tickets/${userId}`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      const ticketsData = response.data;

      const today = new Date();
      for (const ticket of ticketsData) {
        const [datePart, timePart] = ticket.eventDate.split('  ');
```

```
        const [day, month, year] = datePart.split('.');
        const eventDate = new Date(`${year}-${month}-
${day}T${timePart}`);
        const isToday = eventDate.toDateString() ===
today.toDateString();
        if (isToday && ticket.statusId === 2) {
          await cancelReservation(ticket.ticketId, userId);
        }
      }

      const updatedResponse = await
axios.get(`https://localhost:7112/api/tickets/${userId}`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      setTickets(updatedResponse.data);
    } catch (err) {
      addNotification(err.response?.data || 'Ошибка при загрузке
билетов', 'error');
    }
  };

  const purchaseBookedTicket = async (ticketId, userId) => {
    try {
      const token = localStorage.getItem('token');
      const response = await axios.post(
        'https://localhost:7112/api/tickets/purchaseBooked',
        { ticketId, userId },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      addNotification(response.data?.Message || 'Билет успешно
оплачен', 'success');
      fetchTickets(userId);
    } catch (err) {
      addNotification(err.response?.data || 'Ошибка при оплате
билета', 'error');
    }
  };

  const cancelReservation = async (ticketId, userId) => {
    try {
      const token = localStorage.getItem('token');
      const response = await axios.post(
        'https://localhost:7112/api/tickets/cancel/reservation',
        { ticketId, userId },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      addNotification(response.data?.Message || 'Бронь успешно
снята', 'success');
      fetchTickets(userId);
    } catch (err) {
      addNotification(err.response?.data || 'Ошибка при снятии
брони', 'error');
```

```
    }
  };

  const cancelPurchase = async (ticketId, userId) => {
    try {
      const token = localStorage.getItem('token');
      const response = await axios.post(
        'https://localhost:7112/api/tickets/cancel/purchase',
        { ticketId, userId },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      addNotification(response.data?.Message || 'Покупка успешно
отменена', 'success');
      fetchTickets(userId);
    } catch (err) {
      addNotification(err.response?.data || 'Ошибка при отмене
покупки', 'error');
    }
  };

  const handleUpdateName = async (e) => {
    e.preventDefault();
    try {
      const token = localStorage.getItem('token');
      const userId = localStorage.getItem('userId');
      const response = await axios.put(
        `https://localhost:7047/api/users/${userId}/name`,
        { newName },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      const updatedUser = { ...user, name: newName };
      setUser(updatedUser);
      localStorage.setItem('user', JSON.stringify(updatedUser));
      addNotification(response.data.Message || 'Имя успешно
обновлено', 'success');
    } catch (err) {
      if (typeof err.response?.data === 'string') {
        addNotification(err.response.data, 'error');
      } else if (err.response?.data?.errors) {
        const errorMessages =
Object.values(err.response.data.errors).flat().join(' ');
        addNotification(errorMessages || 'Ошибка при обновлении
имени', 'error');
      } else {
        addNotification('Ошибка при обновлении имени', 'error');
      }
    }
  };

  const handleUpdatePassword = async (e) => {
    e.preventDefault();
    if (newPassword !== confirmPassword) {
```

```
      addNotification('Пароли не совпадают', 'error');
      return;
    }
    try {
      const token = localStorage.getItem('token');
      const userId = localStorage.getItem('userId');
      const response = await axios.put(
        `https://localhost:7047/api/users/${userId}/password`,
        { oldPassword, newPassword },
        { headers: { Authorization: `Bearer ${token}` } }
      );
      addNotification(response.data.Message || 'Пароль успешно
обновлён', 'success');
      setOldPassword('');
      setNewPassword('');
      setConfirmPassword('');
    } catch (err) {
      if (typeof err.response?.data === 'string') {
        addNotification(err.response.data, 'error');
      } else if (err.response?.data?.errors) {
        const errorMessages =
Object.values(err.response.data.errors).flat().join(' ');
        addNotification(errorMessages || 'Ошибка при обновлении
пароля', 'error');
      } else {
        addNotification('Ошибка при обновлении пароля', 'error');
      }
    }
  };

  const handleLogout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('userId');
    localStorage.removeItem('user');
    navigate('/login');
  };

  const toggleExpandTicket = (ticketId) => {
    setExpandedTickets(prev => ({
      ...prev,
      [ticketId]: !prev[ticketId],
    }));
  };

  if (!user) return null;

  if (user.userTypeId === 1) {
    return <AdminPanel user={user} handleLogout={handleLogout} />;
  }

  const today = new Date();
  const oneWeekFromNow = new Date(today);
```

```
  oneWeekFromNow.setDate(today.getDate() + 7);
  const oneWeekAgo = new Date(today);
  oneWeekAgo.setDate(today.getDate() - 7);

  const filterTickets = (ticketList) => {
    return ticketList.filter(ticket => {
      const [datePart, timePart] = ticket.eventDate.split('  ');
      const [day, month, year] = datePart.split('.');
      const eventDate = new Date(`${year}-${month}-
${day}T${timePart}`);

      if (statusFilter && ticket.statusId !==
parseInt(statusFilter)) {
        return false;
      }

      if (dateFilter) {
        if (dateFilter === 'today' && eventDate.toDateString() !==
today.toDateString()) {
          return false;
        }
        if (dateFilter === 'thisWeek' && (eventDate < today ||
eventDate > oneWeekFromNow)) {
          return false;
        }
        if (dateFilter === 'nextWeek') {
          const nextWeekStart = new Date(today);
          nextWeekStart.setDate(today.getDate() + 7);
          const nextWeekEnd = new Date(nextWeekStart);
          nextWeekEnd.setDate(nextWeekStart.getDate() + 7);
          if (eventDate < nextWeekStart || eventDate >=
nextWeekEnd) {
            return false;
          }
        }
        if (dateFilter === 'last7Days' && (eventDate > today ||
eventDate < oneWeekAgo)) {
          return false;
        }
      }

      if (searchQuery &&
!ticket.eventName.toLowerCase().includes(searchQuery.toLowerCase()
)) {
        return false;
      }

      return true;
    });
  };

  const activeTickets = tickets.filter(ticket => {
```

```
    const [datePart, timePart] = ticket.eventDate.split('  ');
    const [day, month, year] = datePart.split('.');
    const eventDate = new Date(`${year}-${month}-
${day}T${timePart}`);
    return eventDate >= today;
  });

  const pastTickets = tickets.filter(ticket => {
    const [datePart, timePart] = ticket.eventDate.split('  ');
    const [day, month, year] = datePart.split('.');
    const eventDate = new Date(`${year}-${month}-
${day}T${timePart}`);
    return eventDate < today;
  });

  const filteredActiveTickets = filterTickets(activeTickets);
  const filteredPastTickets = filterTickets(pastTickets);

  return (
    <div className=«container mt-4»>
      <style>
        {`
          .notifications {
            position: fixed;
            bottom: 20px;
            right: 20px;
            z-index: 1000;
          }
          .notification {
            position: relative;
            padding: 10px 20px;
            margin-top: 10px;
            border-radius: 5px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
            animation: fadeInOut 5s ease-in-out forwards;
          }
          .success {
            background-color: #28a745;
            color: white;
          }
          .error {
            background-color: #dc3545;
            color: white;
          }
          @keyframes fadeInOut {
            0% { opacity: 0; transform: translateY(20px); }
            10% { opacity: 1; transform: translateY(0); }
            90% { opacity: 1; transform: translateY(0); }
            100% { opacity: 0; transform: translateY(20px); }
          }
        `}
      </style>
```

```
      <h1 className=«mb-4»>Личный кабинет</h1>
      <div className=«row»>
        <div className=«col-md-3»>
          <div className=«list-group»>
            <button
              className={`list-group-item list-group-item-action
${activeSection === 'info' ? 'active' : ''}`}
              onClick={() => setActiveSection('info')}
            >
              Информация
            </button>
            <button
              className={`list-group-item list-group-item-action
${activeSection === 'updateName' ? 'active' : ''}`}
              onClick={() => setActiveSection('updateName')}
            >
              Изменить имя
            </button>
            <button
              className={`list-group-item list-group-item-action
${activeSection === 'updatePassword' ? 'active' : ''}`}
              onClick={() => setActiveSection('updatePassword')}
            >
              Изменить пароль
            </button>
            <button
              className={`list-group-item list-group-item-action
${activeSection === 'tickets' ? 'active' : ''}`}
              onClick={() => setActiveSection('tickets')}
            >
              Мои билеты
            </button>
            <button className=«list-group-item list-group-item-
action text-danger» onClick={handleLogout}>
              Выйти
            </button>
          </div>
        </div>
        <div className=«col-md-9»>
          {activeSection === 'info' && (
            <div>
              <h3>Информация о пользователе</h3>
              <p><strong>Email:</strong> {user.email}</p>
              <p><strong>Имя:</strong> {user.name}</p>
              <p><strong>Телефон:</strong> {user.phone || 'Не
указан'}</p>
              {/* <p><strong>Тип пользователя:</strong>
{user.userType.name}</p> */}
            </div>
          )}
          {activeSection === 'updateName' && (
            <div>
```

```
              <h3>Изменить имя</h3>
              <form onSubmit={handleUpdateName}>
                <div className=«mb-3»>
                  <label className=«form-label»>Новое имя</label>
                  <input
                    type=«text»
                    value={newName}
                    onChange={(e) => setNewName(e.target.value)}
                    className=«form-control»
                    placeholder=«Введите новое имя»
                    required
                  />
                </div>
                <button type=«submit» className=«btn btn-
primary»>Сохранить</button>
              </form>
            </div>
          )}
          {activeSection === 'updatePassword' && (
            <div>
              <h3>Изменить пароль</h3>
              <form onSubmit={handleUpdatePassword}>
                <div className=«mb-3 position-relative»>
                  <label className=«form-label»>Старый
пароль</label>
                  <input
                    type={showOldPassword ? 'text' : 'password'}
                    value={oldPassword}
                    onChange={(e) =>
setOldPassword(e.target.value)}
                    className=«form-control»
                    placeholder=«Старый пароль»
                    required
                  />
                  <button
                    type=«button»
                    className=«btn btn-outline-secondary position-
absolute»
                    style={{
                      marginTop: '-18px',
                      right: '0.5rem',
                      transform: 'translateY(-50%)',
                      border: 'none',
                    }}
                    onClick={() =>
setShowOldPassword(!showOldPassword)}
                  >
                    {showOldPassword ? '🙈' : '👁'}
                  </button>
                </div>
                <div className=«mb-3 position-relative»>
```

```
                    <label className=«form-label»>Новый
пароль</label>
                    <input
                      type={showNewPassword ? 'text' : 'password'}
                      value={newPassword}
                      onChange={(e) =>
setNewPassword(e.target.value)}
                      className=«form-control»
                      placeholder=«Новый пароль»
                      required
                    />
                    <button
                      type=«button»
                      className=«btn btn-outline-secondary position-
absolute»
                      style={{
                        marginTop: '-18px',
                        right: '0.5rem',
                        transform: 'translateY(-50%)',
                        border: 'none',
                      }}
                      onClick={() =>
setShowNewPassword(!showNewPassword)}
                    >
                      {showNewPassword ? '🙈' : '👁'}
                    </button>
                  </div>
                  <div className=«mb-3 position-relative»>
                    <label className=«form-label»>Подтвердите новый
пароль</label>
                    <input
                      type={showConfirmPassword ? 'text' :
'password'}
                      value={confirmPassword}
                      onChange={(e) =>
setConfirmPassword(e.target.value)}
                      className=«form-control»
                      placeholder=«Подтвердите новый пароль»
                      required
                    />
                    <button
                      type=«button»
                      className=«btn btn-outline-secondary position-
absolute»
                      style={{
                        marginTop: '-18px',
                        right: '0.5rem',
                        transform: 'translateY(-50%)',
                        border: 'none',
                      }}
                      onClick={() =>
setShowConfirmPassword(!showConfirmPassword)}
```

```
                  >
                    {showConfirmPassword ? '🙈' : '👁'}
                  </button>
                </div>
                <button type=«submit» className=«btn btn-
primary»>Сохранить</button>
              </form>
            </div>
          )}
          {activeSection === 'tickets' && (
            <div>
              <h3>Мои билеты</h3>
              <div className=«mb-4»>
                <h5>Фильтры</h5>
                <div className=«row»>
                  <div className=«col-md-4 mb-3»>
                    <label className=«form-label»>Статус</label>
                    <select
                      className=«form-control»
                      value={statusFilter}
                      onChange={(e) =>
setStatusFilter(e.target.value)}
                    >
                      <option value=««>Все</option>
                      <option value=«1»>Оплачен</option>
                      <option value=«2»>Забронирован</option>
                      <option value=«3»>Отменён (бронь)</option>
                      <option value=«4»>Отменён (покупка)</option>
                    </select>
                  </div>
                  <div className=«col-md-4 mb-3»>
                    <label className=«form-label»>Дата</label>
                    <select
                      className=«form-control»
                      value={dateFilter}
                      onChange={(e) =>
setDateFilter(e.target.value)}
                    >
                      <option value=««>Все даты</option>
                      <option value=«today»>Сегодня</option>
                      <option value=«thisWeek»>На этой
неделе</option>
                      <option value=«nextWeek»>На следующей
неделе</option>
                      <option value=«last7Days»>Прошлые 7
дней</option>
                    </select>
                  </div>
                  <div className=«col-md-4 mb-3»>
                    <label className=«form-label»>Поиск по
названию</label>
                    <input
```

```
                        type=«text»
                        className=«form-control»
                        placeholder=«Введите название мероприятия»
                        value={searchQuery}
                        onChange={(e) =>
setSearchQuery(e.target.value)}
                      />
                    </div>
                  </div>
                </div>

                <h4>Активные мероприятия</h4>
                {filteredActiveTickets.length === 0 ? (
                  <p>Нет активных билетов.</p>
                ) : (
                  filteredActiveTickets.map(ticket => {
                    const [datePart, timePart] =
ticket.eventDate.split('  ');
                    const [day, month, year] = datePart.split('.');
                    const eventDate = new Date(`${year}-${month}-
${day}T${timePart}`);
                    const isToday = eventDate.toDateString() ===
today.toDateString();
                    const isPurchased = ticket.statusId === 1;
                    const isBooked = ticket.statusId === 2;
                    const isCancelled = ticket.statusId === 3 ||
ticket.statusId === 4;
                    const isExpanded =
expandedTickets[ticket.ticketId] || false;

                    if (isCancelled && !statusFilter) return null;

                    return (
                      <div key={ticket.ticketId} className=«card mb-
2»>
                        <div
                          className=«card-body d-flex justify-
content-between align-items-center»
                          style={{ cursor: 'pointer' }}
                          onClick={() =>
toggleExpandTicket(ticket.ticketId)}
                        >
                          <div>
                            <h6 className=«card-title mb-
0»>{ticket.eventName}</h6>
                              <small className=«text-muted»>
                                {ticket.eventDate} | Место:
{ticket.seatNumber} | Статус: {ticket.status.name}
                              </small>
                          </div>
                          <button className=«btn btn-link p-0»>
```

```
                            {isExpanded ? 'Свернуть ▲' : 'Подробнее
▼'}
                          </button>
                        </div>
                        {isExpanded && (
                          <div className=«card-body border-top»>
                            <p className=«card-text»>
                              <strong>Цена:</strong> {ticket.price}
руб.<br />
                              <strong>Дата покупки:</strong>
{ticket.purchaseDate || 'Билет не оплачен'}<br />
                              {ticket.reservationDate &&
!isPurchased && (
                                <>
                                  <strong>Дата
бронирования:</strong> {ticket.reservationDate}<br />
                                </>
                              )}
                            </p>
                            {!isCancelled && (
                              <div className=«d-flex gap-2»>
                                {isBooked && (
                                  <button
                                    className=«btn btn-primary btn-
sm»
                                    onClick={() =>
purchaseBookedTicket(ticket.ticketId, user.id)}
                                    disabled={isToday}
                                  >
                                    Оплатить
                                  </button>
                                )}
                                {isBooked && (
                                  <button
                                    className=«btn btn-warning btn-
sm»
                                    onClick={() =>
cancelReservation(ticket.ticketId, user.id)}
                                    disabled={isToday}
                                  >
                                    Снять бронь
                                  </button>
                                )}
                                {isPurchased && (
                                  <button
                                    className=«btn btn-danger btn-
sm»
                                    onClick={() =>
cancelPurchase(ticket.ticketId, user.id)}
                                    disabled={isToday}
                                  >
                                    Отменить покупку
```

```
                                    </button>
                                  )}
                                </div>
                              )}
                            </div>
                          )}
                        </div>
                      );
                    })
                  )}

                  <h4 className=«mt-4»>Завершённые мероприятия</h4>
                  {filteredPastTickets.length === 0 ? (
                    <p>Нет завершённых мероприятий.</p>
                  ) : (
                    filteredPastTickets.map(ticket => {
                      const [datePart, timePart] =
ticket.eventDate.split('  ');
                      const [day, month, year] = datePart.split('.');
                      const eventDate = new Date(`${year}-${month}-
${day}T${timePart}`);
                      const isCancelled = ticket.statusId === 3 ||
ticket.statusId === 4;
                      const isExpanded =
expandedTickets[ticket.ticketId] || false;

                      if (isCancelled && !statusFilter) return null;

                      return (
                        <div key={ticket.ticketId} className=«card mb-
2»>
                          <div
                            className=«card-body d-flex justify-
content-between align-items-center»
                            style={{ cursor: 'pointer' }}
                            onClick={() =>
toggleExpandTicket(ticket.ticketId)}
                          >
                            <div>
                              <h6 className=«card-title mb-
0»>{ticket.eventName}</h6>
                              <small className=«text-muted»>
                                {ticket.eventDate} | Место:
{ticket.seatNumber} | Статус: {ticket.status.name}
                              </small>
                            </div>
                            <button className=«btn btn-link p-0»>
                              {isExpanded ? 'Свернуть ▲' : 'Подробнее
▼'}
                            </button>
                          </div>
                          {isExpanded && (
```

```
                        <div className=«card-body border-top»>
                          <p className=«card-text»>
                            <strong>Цена:</strong> {ticket.price}
руб.<br />
                            <strong>Дата покупки:</strong>
{ticket.purchaseDate || 'Билет не оплачен'}<br />
                            {ticket.reservationDate &&
ticket.statusId !== 1 && (
                              <>
                                <strong>Дата
бронирования:</strong> {ticket.reservationDate}<br />
                              </>
                            )}
                          </p>
                        </div>
                      )}
                    </div>
                  );
                })
              )}
            </div>
          )}
        </div>
      </div>
      <div className=«notifications»>
        {notifications.map(notif => (
          <div key={notif.id} className={`notification
${notif.type}`}>
            {notif.message}
          </div>
        ))}
      </div>
    </div>
  );
};

export default Profile;
```

Листинг В.9 – Файл Search.jsx

```
import { useState, useEffect } from 'react';
import axios from 'axios';
import { Link, useLocation, useNavigate } from 'react-router-dom';

const Search = () => {
  const [searchString, setSearchString] = useState('');
  const [dateFrom, setDateFrom] = useState('');
  const [dateTo, setDateTo] = useState('');
  const [eventTypeId, setEventTypeId] = useState('');
  const [sortBy, setSortBy] = useState('');
  const [sortDescending, setSortDescending] = useState(false);
  const [eventTypes, setEventTypes] = useState([]);
  const [results, setResults] = useState([]);
```

```
  const [currentPage, setCurrentPage] = useState(1);
  const eventsPerPage = 9;
  const location = useLocation();
  const navigate = useNavigate();
  const [minPrices, setMinPrices] = useState({});
  const [loadingPrices, setLoadingPrices] = useState({});

  useEffect(() => {
    fetchEventTypes();
    if (location.state?.fromEventDetail) {
      const savedState = localStorage.getItem('searchState');
      if (savedState) {
        const { params, results, page } = JSON.parse(savedState);
        setSearchString(params.searchString || '');
        setDateFrom(params.dateFrom || '');
        setDateTo(params.dateTo || '');
        setEventTypeId(params.eventTypeId || '');
        setSortBy(params.sortBy || '');
        setSortDescending(params.sortDescending || false);
        setResults(results || []);
        setCurrentPage(page || 1);
        results.forEach(event => fetchMinPrice(event.id));
      }
    } else {
      localStorage.removeItem('searchState');
    }
  }, [location]);

  useEffect(() => {
    handleSearch();
  }, [searchString, dateFrom, dateTo, eventTypeId, sortBy,
sortDescending]);

  const fetchEventTypes = async () => {
    try {
      const response = await
axios.get('https://localhost:7109/api/eventtype');
      setEventTypes(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке типов мероприятий:',
err);
    }
  };

  const fetchMinPrice = async (eventId) => {
    setLoadingPrices(prev => ({ ...prev, [eventId]: true }));
    try {
      const token = localStorage.getItem('token');
      const response = await
axios.get(`https://localhost:7109/api/${eventId}/eventseat`, {
        headers: { Authorization: `Bearer ${token}` },
      });
```

```
      const seats = response.data;
      const minPrice = seats.length > 0 ?
Math.min(...seats.map(seat => seat.price)) : null;
      setMinPrices(prev => ({ ...prev, [eventId]: minPrice }));
    } catch (err) {
      console.error(`Ошибка при загрузке мест для мероприятия
${eventId}:`, err);
      setMinPrices(prev => ({ ...prev, [eventId]: null }));
    } finally {
      setLoadingPrices(prev => ({ ...prev, [eventId]: false }));
    }
  };

  const handleSearch = async () => {
    const searchParams = {
      SearchString: searchString || undefined,
      DateFrom: dateFrom ? new Date(dateFrom).toISOString() :
undefined,
      DateTo: dateTo ? new Date(dateTo).toISOString() : undefined,
      EventTypeId: eventTypeId ? parseInt(eventTypeId) :
undefined,
      SortBy: sortBy || undefined,
      SortDescending: sortDescending,
    };

    const queryString = Object.entries(searchParams)
      .filter(([_, value]) => value !== undefined)
      .map(([key, value]) =>
`${key}=${encodeURIComponent(value)}`)
      .join('&');

    try {
      const response = await
axios.get(`https://localhost:7109/api/search?${queryString}`);
      const newResults = response.data;
      setResults(newResults);
      setCurrentPage(1);
      localStorage.setItem(
        'searchState',
        JSON.stringify({
          params: { searchString, dateFrom, dateTo, eventTypeId,
sortBy, sortDescending },
          results: newResults,
          page: 1,
        })
      );
      newResults.forEach(event => fetchMinPrice(event.id));
    } catch (err) {
      console.error('Ошибка при поиске:', err);
    }
  };
```

```
  const clearFilters = () => {
    setSearchString('');
    setDateFrom('');
    setDateTo('');
    setEventTypeId('');
    setSortBy('');
    setSortDescending(false);
    setCurrentPage(1);
    setResults([]);
    localStorage.removeItem('searchState');
  };

  const totalPages = Math.ceil(results.length / eventsPerPage);
  const startIndex = (currentPage - 1) * eventsPerPage;
  const currentEvents = results.slice(startIndex, startIndex +
eventsPerPage);

  const handlePageChange = (newPage) => {
    setCurrentPage(newPage);
    const savedState =
JSON.parse(localStorage.getItem('searchState'));
    if (savedState) {
      localStorage.setItem(
        'searchState',
        JSON.stringify({
          ...savedState,
          page: newPage,
        })
      );
    }
  };

  return (
    <div className=«container mt-4»>
      <style>
        {`
          .event-card {
            transition: transform 0.3s ease, filter 0.3s ease;
          }
          .event-card:hover {
            transform: scale(1.05);
            filter: brightness(90%);
          }
          .event-image {
            height: 200px;
            object-fit: cover;
          }
        `}
      </style>
      <h1 className=«mb-4»>Поиск мероприятий</h1>
      <div className=«row mb-4»>
        <div className=«col-md-12»>
```

```
          <div className=«card p-4»>
            <div className=«row»>
              <div className=«col-md-4 mb-3»>
                <label className=«form-label»>Поиск по имени или
локации</label>
                <input
                  type=«text»
                  className=«form-control»
                  value={searchString}
                  onChange={(e) =>
setSearchString(e.target.value)}
                  placeholder=«Введите запрос»
                />
              </div>
              <div className=«col-md-2 mb-3»>
                <label className=«form-label»>Дата с</label>
                <input
                  type=«date»
                  className=«form-control»
                  value={dateFrom}
                  onChange={(e) => setDateFrom(e.target.value)}
                />
              </div>
              <div className=«col-md-2 mb-3»>
                <label className=«form-label»>Дата по</label>
                <input
                  type=«date»
                  className=«form-control»
                  value={dateTo}
                  onChange={(e) => setDateTo(e.target.value)}
                />
              </div>
              <div className=«col-md-2 mb-3»>
                <label className=«form-label»>Тип
мероприятия</label>
                <select
                  className=«form-control»
                  value={eventTypeId}
                  onChange={(e) => setEventTypeId(e.target.value)}
                >
                  <option value=««>Все</option>
                  {eventTypes.map(type => (
                    <option key={type.id}
value={type.id}>{type.name}</option>
                  ))}
                </select>
              </div>
            </div>
            <div className=«row mt-3»>
              <div className=«col-md-4 mb-3»>
                <label className=«form-label»>Сортировать
по</label>
```

```
                <select
                  className=«form-control»
                  value={sortBy}
                  onChange={(e) => setSortBy(e.target.value)}
                >
                  <option value=««>Не сортировать</option>
                  <option value=«date»>Дате</option>
                  <option value=«price»>Цене</option>
                </select>
              </div>
              <div className=«col-md-2 mb-3»>
                <label className=«form-label»>Направление</label>
                <select
                  className=«form-control»
                  value={sortDescending}
                  onChange={(e) =>
setSortDescending(e.target.value === 'true')}
                >
                  <option value={false}>По возрастанию</option>
                  <option value={true}>По убыванию</option>
                </select>
              </div>
              <div className=«col-md-2 mb-3 align-self-end»>
              <button
                  className=«btn btn-secondary»
                  style={{ width: '200px', height: '40px' }}
                  onClick={clearFilters}
                >
                  Очистить фильтры
                </button>
              </div>
            </div>
          </div>
        </div>
      </div>
      <h2 className=«mb-4»>Результаты поиска</h2>
      {results.length === 0 ? (
        <p>Нет результатов.</p>
      ) : (
        <>
          {currentEvents.length === 0 ? (
            <p>Нет мероприятий на этой странице.</p>
          ) : (
            <div className=«row»>
              {currentEvents.map(event => (
                <div key={event.id} className=«col-md-4 mb-3»>
                  <div className=«card event-card»>
                    {event.imagePath && (
                      <img

src={`https://localhost:7109${event.imagePath}`}
                        alt={event.name}
```

```
                        className=«card-img-top event-image»
                      />
                    )}
                    <div className=«card-body»>
                      <h5 className=«card-title»>{event.name}</h5>
                      <p className=«card-text»>
                        <strong>Дата:</strong>
{event.eventDate}<br />
                        <strong>Место:</strong>
{event.location}<br />
                        <strong>Цена от:</strong>{' '}
                        {loadingPrices[event.id] ? 'Загрузка...' :
minPrices[event.id] !== null ? `${minPrices[event.id]} руб.` :
'Цена не указана'}
                      </p>
                      <Link
                        to={`/event/${event.id}`}
                        state={{ from: '/search' }}
                        className=«btn btn-primary»
                        onClick={() => {
                          localStorage.setItem(
                            'searchState',
                            JSON.stringify({
                              params: { searchString, dateFrom,
dateTo, eventTypeId, sortBy, sortDescending },
                              results,
                              page: currentPage,
                            })
                          );
                        }}
                      >
                        Подробнее
                      </Link>
                    </div>
                  </div>
                </div>
              ))}
            </div>
          )}
          {totalPages > 1 && (
            <nav aria-label=«Page navigation»>
              <ul className=«pagination justify-content-center»>
                <li className={`page-item ${currentPage === 1 ?
'disabled' : ''}`}>
                  <button className=«page-link» onClick={() =>
handlePageChange(1)}>
                    Первая
                  </button>
                </li>
                <li className={`page-item ${currentPage === 1 ?
'disabled' : ''}`}>
```

```
                  <button className=«page-link» onClick={() =>
handlePageChange(currentPage - 1)}>
                    Назад
                  </button>
                </li>
                {[...Array(totalPages)].map((_, i) => (
                  <li key={i + 1} className={`page-item
${currentPage === i + 1 ? 'active' : ''}`}>
                    <button className=«page-link» onClick={() =>
handlePageChange(i + 1)}>
                      {i + 1}
                    </button>
                  </li>
                ))}
                <li className={`page-item ${currentPage ===
totalPages ? 'disabled' : ''}`}>
                  <button className=«page-link» onClick={() =>
handlePageChange(currentPage + 1)}>
                    Вперед
                  </button>
                </li>
                <li className={`page-item ${currentPage ===
totalPages ? 'disabled' : ''}`}>
                  <button className=«page-link» onClick={() =>
handlePageChange(totalPages)}>
                    Последняя
                  </button>
                </li>
              </ul>
            </nav>
          )}
        </>
      )}
    </div>
  );
};

export default Search;
```

Листинг В.10 – Файл AdminPanel.jsx

```
import { useState, useEffect } from 'react';
import axios from 'axios';

const AdminPanel = ({ user, handleLogout }) => {
  const [activeSection, setActiveSection] = useState('events');
  const [notifications, setNotifications] = useState([]);
  const [events, setEvents] = useState([]);
  const [eventTypes, setEventTypes] = useState([]);
  const [newEvent, setNewEvent] = useState({
    name: '',
    eventTypeId: '',
    eventDate: '',
```

```
    description: '',
    location: '',
    totalSeats: '',
    image: null,
  });
  const [editEvent, setEditEvent] = useState(null);
  const [eventSeats, setEventSeats] = useState([]);
  const [existingSeats, setExistingSeats] = useState([]);
  const [showBlocks, setShowBlocks] = useState(false);
  const [showExistingSeats, setShowExistingSeats] =
useState(false);
  const [statusFilter, setStatusFilter] = useState('');
  const [typeFilter, setTypeFilter] = useState('');
  const [dateFilter, setDateFilter] = useState('');
  const [searchQuery, setSearchQuery] = useState('');
  const [sortOption, setSortOption] = useState('');
  const [selectedSeat, setSelectedSeat] = useState(null);
  const [selectedSeats, setSelectedSeats] = useState([]);
  const [showModal, setShowModal] = useState(false);
  const [modalMode, setModalMode] = useState('single');
  const [errors, setErrors] = useState({ seatNumber: '', price: ''
});
  const [currentPage, setCurrentPage] = useState(1);
  const [seatsPerPage, setSeatsPerPage] = useState(10);

  const addNotification = (message, type = 'success') => {
    const id = Date.now();
    setNotifications(prev => [...prev, { id, message, type }]);
    setTimeout(() => {
      setNotifications(prev => prev.filter(notif => notif.id !==
id));
    }, 10000);
  };

  useEffect(() => {
    fetchEvents();
    fetchEventTypes();
  }, []);

  const fetchEvents = async () => {
    try {
      const token = localStorage.getItem('token');
      if (!token) throw new Error('Токен отсутствует в
localStorage');
      const response = await
axios.get('https://localhost:7109/api/event', {
        headers: { Authorization: `Bearer ${token}` },
      });
      setEvents(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке мероприятий:', err);
```

```
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при загрузке мероприятий', 'error');
    }
  };

  const fetchEventTypes = async () => {
    try {
      const token = localStorage.getItem('token');
      if (!token) throw new Error('Токен отсутствует в
localStorage');
      const response = await
axios.get('https://localhost:7109/api/eventtype', {
        headers: { Authorization: `Bearer ${token}` },
      });
      setEventTypes(response.data);
    } catch (err) {
      console.error('Ошибка при загрузке типов мероприятий:',
err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при загрузке типов мероприятий', 'error');
    }
  };

  const fetchEventSeats = async (eventId) => {
    try {
      const token = localStorage.getItem('token');
      const response = await
axios.get(`https://localhost:7109/api/${eventId}/eventseat`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      setExistingSeats(response.data);
      setCurrentPage(1);
    } catch (err) {
      console.error('Ошибка при загрузке мест:', err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при загрузке мест', 'error');
    }
  };

  const validateEventName = (name) => {
    if (name.length === 0) {
      return { isValid: false, message: 'Ошибка: название
мероприятия не может быть пустым' };
    }
    if (name.length > 100) {
      return { isValid: false, message: 'Ошибка: длина
наименования больше 100 символов' };
    }

    const startsWithLetter = /^[A-Za-zA-Яа-яЁё]/.test(name);
    if (!startsWithLetter) {
```

```
      return { isValid: false, message: 'Ошибка: начинается не с
буквы' };
    }

    const validContent = /^[A-Za-zA-Яа-яЁё0-9\s\-
\.]*$/.test(name);
    if (!validContent) {
      return { isValid: false, message: 'Ошибка: содержит
недопустимые символы' };
    }

    return { isValid: true, message: 'Мероприятие успешно создано'
};
  };

  const handleCreateEvent = async (e) => {
    e.preventDefault();

    const validation = validateEventName(newEvent.name);
    if (!validation.isValid) {
      addNotification(validation.message, 'error');
      return;
    }

    try {
      const token = localStorage.getItem('token');
      const formData = new FormData();
      formData.append('name', newEvent.name);
      formData.append('eventTypeId', newEvent.eventTypeId);
      formData.append('eventDate', newEvent.eventDate);
      formData.append('description', newEvent.description);
      formData.append('location', newEvent.location);
      formData.append('totalSeats', newEvent.totalSeats);
      if (newEvent.image) formData.append('ImageFile',
newEvent.image);
      formData.append('eventStatusId', '1');

      const response = await
axios.post('https://localhost:7109/api/event', formData, {
        headers: {
          Authorization: `Bearer ${token}`,
          'Content-Type': 'multipart/form-data',
        },
      });
      addNotification('Мероприятие успешно создано', 'success');
      setNewEvent({
        name: '',
        eventTypeId: '',
        eventDate: '',
        description: '',
        location: '',
        totalSeats: '',
```

```
        image: null,
      });
      setActiveSection('events');
      fetchEvents();
    } catch (err) {
      console.error('Ошибка при создании мероприятия:', err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при создании мероприятия', 'error');
    }
  };

  const handleUpdateEvent = async (e) => {
    e.preventDefault();

    const validation = validateEventName(editEvent.name);
    if (!validation.isValid) {
      addNotification(validation.message, 'error');
      return;
    }

    try {
      const token = localStorage.getItem('token');
      const formData = new FormData();
      formData.append('name', editEvent.name);
      formData.append('eventTypeId', editEvent.eventTypeId);
      formData.append('eventDate', editEvent.eventDate);
      formData.append('description', editEvent.description);
      formData.append('location', editEvent.location);
      formData.append('totalSeats', editEvent.totalSeats);
      formData.append('eventStatusId', editEvent.eventStatusId);

      if (editEvent.image && typeof editEvent.image !== 'string')
{
        formData.append('ImageFile', editEvent.image);
      }

      const response = await
axios.put(`https://localhost:7109/api/event/${editEvent.id}`,
formData, {
        headers: {
          Authorization: `Bearer ${token}`,
          'Content-Type': 'multipart/form-data',
        },
      });
      addNotification('Мероприятие успешно обновлено', 'success');
      setEditEvent(null);
      setActiveSection('events');
      fetchEvents();
    } catch (err) {
      console.error('Ошибка при обновлении мероприятия:',
err.response?.data || err);
```

```
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при обновлении мероприятия', 'error');
    }
  };

  const handleDeleteEvent = async (eventId) => {
    try {
      const token = localStorage.getItem('token');
      await
axios.delete(`https://localhost:7109/api/event/${eventId}`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      addNotification('Мероприятие успешно удалено', 'success');
      fetchEvents();
    } catch (err) {
      console.error('Ошибка при удалении мероприятия:', err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при удалении мероприятия', 'error');
    }
  };

  const handleAddEventSeats = async (eventId) => {
    try {
      const token = localStorage.getItem('token');
      const event = events.find(e => e.id === eventId);
      const totalSeatsAdded = eventSeats.reduce((sum, block) =>
sum + parseInt(block.seatCount || 0), 0);

      if (totalSeatsAdded > event.totalSeats) {
        addNotification(`Общее количество мест
(${totalSeatsAdded}) превышает доступное количество
(${event.totalSeats})`, 'error');
        return;
      }

      let totalSeatsProcessed = 0;

      for (const block of eventSeats) {
        const seatCount = parseInt(block.seatCount);
        const price = parseFloat(block.price);
        const blockName = showBlocks ? block.blockName : '';

        for (let i = 1; i <= seatCount; i++) {
          const seatNumber = blockName ? `${blockName}${i}` :
i.toString();
          const seatData = {
            seatNumber: seatNumber,
            price: price,
          };
```

```
          await
axios.post(`https://localhost:7109/api/${eventId}/eventseat`,
seatData, {
            headers: {
              Authorization: `Bearer ${token}`,
              'Content-Type': 'application/json',
            },
          });

          totalSeatsProcessed++;
        }
      }

      addNotification(`Места успешно добавлены
(${totalSeatsProcessed} мест)`, 'success');
      setEventSeats([]);
      setShowBlocks(false);
      fetchEventSeats(eventId);
      setActiveSection('events');
      fetchEvents();
    } catch (err) {
      console.error('Ошибка при добавлении мест:',
err.response?.data || err);
      const errorMessage =
err.response?.data?.errors?.SeatNumber?.[0] ||
                           err.response?.data?.errors?.Price?.[0]
||
                           err.response?.data?.message ||
                           err.message ||
                           'Ошибка при добавлении мест';
      addNotification(errorMessage, 'error');
    }
  };

  const validateSeat = (seat, isBulk = false) => {
    let newErrors = { seatNumber: '', price: '' };
    let isValid = true;

    if (!isBulk) {
      if (!seat.seatNumber) {
        newErrors.seatNumber = 'Номер места обязателен';
        isValid = false;
      } else {
        const otherSeats = existingSeats.filter(s => s.id !==
(modalMode === 'single' ? seat.id : -1));
        if (otherSeats.some(s => s.seatNumber ===
seat.seatNumber)) {
          newErrors.seatNumber = 'Номер места должен быть
уникальным';
          isValid = false;
        }
      }
```

```
    }

    if (!isBulk || (isBulk && seat.price)) {
      if (!seat.price || seat.price <= 0) {
        newErrors.price = 'Цена должна быть больше 0';
        isValid = false;
      }
    }

    setErrors(newErrors);
    return isValid;
  };

  const handleUpdateEventSeat = async (seatId, updatedSeat) => {
    if (!validateSeat(updatedSeat)) return;

    try {
      const token = localStorage.getItem('token');
      await
axios.put(`https://localhost:7109/api/eventseat/${seatId}`,
updatedSeat, {
        headers: {
          Authorization: `Bearer ${token}`,
          'Content-Type': 'application/json',
        },
      });
      addNotification('Место успешно обновлено', 'success');
      fetchEventSeats(editEvent.id);
    } catch (err) {
      console.error('Ошибка при обновлении места:', err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при обновлении места', 'error');
    }
  };

  const handleDeleteEventSeat = async (seatId) => {
    try {
      const token = localStorage.getItem('token');
      await
axios.delete(`https://localhost:7109/api/eventseat/${seatId}`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      addNotification('Место успешно удалено', 'success');
      fetchEventSeats(editEvent.id);
    } catch (err) {
      console.error('Ошибка при удалении места:', err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при удалении места', 'error');
    }
  };

  const handleBulkUpdate = async (bulkData) => {
```

```
    if (!validateSeat(bulkData, true)) return;

    try {
      const token = localStorage.getItem('token');
      for (const seatId of selectedSeats) {
        const seatData = {
          price: bulkData.price || existingSeats.find(s => s.id
=== seatId).price,
        };
        await
axios.put(`https://localhost:7109/api/eventseat/${seatId}`,
seatData, {
          headers: {
            Authorization: `Bearer ${token}`,
            'Content-Type': 'application/json',
          },
        });
      }
      addNotification(`Выбранные места (${selectedSeats.length})
успешно обновлены`, 'success');
      setSelectedSeats([]);
      fetchEventSeats(editEvent.id);
      setShowModal(false);
    } catch (err) {
      console.error('Ошибка при массовом обновлении мест:', err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при массовом обновлении мест', 'error');
    }
  };

  const handleBulkDelete = async () => {
    try {
      const token = localStorage.getItem('token');
      for (const seatId of selectedSeats) {
        await
axios.delete(`https://localhost:7109/api/eventseat/${seatId}`, {
          headers: { Authorization: `Bearer ${token}` },
        });
      }
      addNotification(`Выбранные места (${selectedSeats.length})
успешно удалены`, 'success');
      setSelectedSeats([]);
      fetchEventSeats(editEvent.id);
      setShowModal(false);
    } catch (err) {
      console.error('Ошибка при массовом удалении мест:', err);
      addNotification(err.response?.data?.message || err.message
|| 'Ошибка при массовом удалении мест', 'error');
    }
  };

  const toggleSeatSelection = (seatId) => {
```

```
    setSelectedSeats(prev =>
      prev.includes(seatId) ? prev.filter(id => id !== seatId) :
[...prev, seatId]
    );
  };

  const clearSelection = () => {
    setSelectedSeats([]);
  };

  const filteredEvents = events.filter(event => {
    const [datePart, timePart] = event.eventDate.split(' ');
    const [day, month, year] = datePart.split('.');
    const eventDate = new Date(`${year}-${month}-
${day}T${timePart}`);

    const matchesStatus = !statusFilter || event.eventStatus?.id
=== parseInt(statusFilter);
    const matchesType = !typeFilter || event.eventType?.id ===
parseInt(typeFilter);
    const matchesSearch = !searchQuery ||
event.name.toLowerCase().includes(searchQuery.toLowerCase());
    let matchesDate = true;
    const today = new Date();
    const oneWeekFromNow = new Date(today);
    oneWeekFromNow.setDate(today.getDate() + 7);
    const oneWeekAgo = new Date(today);
    oneWeekAgo.setDate(today.getDate() - 7);
    if (dateFilter) {
      if (dateFilter === 'today' && eventDate.toDateString() !==
today.toDateString()) {
        matchesDate = false;
      }
      if (dateFilter === 'thisWeek' && (eventDate < today ||
eventDate > oneWeekFromNow)) {
        matchesDate = false;
      }
      if (dateFilter === 'last7Days' && (eventDate > today ||
eventDate < oneWeekAgo)) {
        matchesDate = false;
      }
    }

    return matchesStatus && matchesType && matchesSearch &&
matchesDate;
  });

  const sortedEvents = [...filteredEvents].sort((a, b) => {
    if (sortOption === 'nameAsc') return
a.name.localeCompare(b.name);
    if (sortOption === 'nameDesc') return
b.name.localeCompare(a.name);
```

```
    if (sortOption === 'dateAsc') {
      const [datePartA] = a.eventDate.split(' ');
      const [dayA, monthA, yearA] = datePartA.split('.');
      const dateA = new Date(`${yearA}-${monthA}-${dayA}`);

      const [datePartB] = b.eventDate.split(' ');
      const [dayB, monthB, yearB] = datePartB.split('.');
      const dateB = new Date(`${yearB}-${monthB}-${dayB}`);

      return dateA - dateB;
    }
    if (sortOption === 'dateDesc') {
      const [datePartA] = a.eventDate.split(' ');
      const [dayA, monthA, yearA] = datePartA.split('.');
      const dateA = new Date(`${yearA}-${monthA}-${dayA}`);

      const [datePartB] = b.eventDate.split(' ');
      const [dayB, monthB, yearB] = datePartB.split('.');
      const dateB = new Date(`${yearB}-${monthB}-${dayB}`);

      return dateB - dateA;
    }
    return 0;
  });

  const resetFilters = () => {
    setStatusFilter('');
    setTypeFilter('');
    setDateFilter('');
    setSearchQuery('');
    setSortOption('');
  };

  useEffect(() => {
    if (editEvent && activeSection === 'addSeats') {
      fetchEventSeats(editEvent.id);
    }
  }, [editEvent, activeSection]);

  const indexOfLastSeat = currentPage * seatsPerPage;
  const indexOfFirstSeat = indexOfLastSeat - seatsPerPage;
  const currentSeats = existingSeats.slice(indexOfFirstSeat,
indexOfLastSeat);
  const totalPages = Math.ceil(existingSeats.length /
seatsPerPage);

  const paginate = (pageNumber) => setCurrentPage(pageNumber);
  const nextPage = () => {
    if (currentPage < totalPages) setCurrentPage(currentPage + 1);
  };
  const prevPage = () => {
    if (currentPage > 1) setCurrentPage(currentPage - 1);
```

```
};

return (
  <div className=«container mt-4»>
    <style>
      {`
        .notifications {
          position: fixed;
          bottom: 20px;
          right: 20px;
          z-index: 1000;
        }
        .notification {
          position: relative;
          padding: 10px 20px;
          margin-top: 10px;
          border-radius: 5px;
          box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
          animation: fadeInOut 10s ease-in-out forwards;
        }
        .success {
          background-color: #28a745;
          color: white;
        }
        .error {
          background-color: #dc3545;
          color: white;
        }
        @keyframes fadeInOut {
          0% { opacity: 0; transform: translateY(20px); }
          10% { opacity: 1; transform: translateY(0); }
          90% { opacity: 1; transform: translateY(0); }
          100% { opacity: 0; transform: translateY(20px); }
        }
        .modal {
          position: fixed;
          top: 0;
          left: 0;
          width: 100%;
          height: 100%;
          background: rgba(0, 0, 0, 0.5);
          display: flex;
          justify-content: center;
          align-items: center;
          z-index: 1050;
        }
        .modal-content {
          background: white;
          padding: 20px;
          border-radius: 5px;
          width: 90%;
          max-width: 500px;
```

```
          }
          .seat-card {
            cursor: pointer;
            transition: transform 0.2s;
          }
          .seat-card:hover {
            transform: scale(1.02);
          }
          .seat-info {
            cursor: pointer;
          }
          .error-text {
            color: #dc3545;
            font-size: 0.875em;
            margin-top: 0.25rem;
          }
          .pagination {
            display: flex;
            gap: 5px;
            align-items: center;
            margin-top: 10px;
          }
          .pagination button {
            padding: 5px 10px;
          }
        `}
      </style>
      <h1 className=«mb-4»>Панель администратора</h1>
      <div className=«row»>
        <div className=«col-md-3»>
          <div className=«list-group»>
            <button
              className={`list-group-item list-group-item-action
${activeSection === 'events' ? 'active' : ''}`}
              onClick={() => setActiveSection('events')}
            >
              Мероприятия
            </button>
            <button
              className={`list-group-item list-group-item-action
${activeSection === 'addEvent' ? 'active' : ''}`}
              onClick={() => setActiveSection('addEvent')}
            >
              Добавить мероприятие
            </button>
            <button className=«list-group-item list-group-item-
action text-danger» onClick={handleLogout}>
              Выйти
            </button>
          </div>
        </div>
        <div className=«col-md-9»>
```

```
          {activeSection === 'events' && (
            <div>
              <h3>Список мероприятий</h3>
              <div className=«mb-4»>
                <h5>Фильтры и сортировка</h5>
                <div className=«row»>
                  <div className=«col-md-3 mb-3»>
                    <label className=«form-label»>Статус</label>
                    <select
                      className=«form-control»
                      value={statusFilter}
                      onChange={(e) =>
setStatusFilter(e.target.value)}
                    >
                      <option value=««>Все</option>
                      <option value=«1»>Активное</option>
                      <option value=«2»>Завершено</option>
                      <option value=«3»>Отменено</option>
                    </select>
                  </div>
                  <div className=«col-md-3 mb-3»>
                    <label className=«form-label»>Тип
мероприятия</label>
                    <select
                      className=«form-control»
                      value={typeFilter}
                      onChange={(e) =>
setTypeFilter(e.target.value)}
                    >
                      <option value=««>Все</option>
                      {eventTypes.map(type => (
                        <option key={type.id}
value={type.id}>{type.name}</option>
                      ))}
                    </select>
                  </div>
                  <div className=«col-md-3 mb-3»>
                    <label className=«form-label»>Дата</label>
                    <select
                      className=«form-control»
                      value={dateFilter}
                      onChange={(e) =>
setDateFilter(e.target.value)}
                    >
                      <option value=««>Все даты</option>
                      <option value=«today»>Сегодня</option>
                      <option value=«thisWeek»>На этой
неделе</option>
                      <option value=«last7Days»>Прошлые 7
дней</option>
                    </select>
                  </div>
```

```
                  <div className=«col-md-3 mb-3»>
                    <label className=«form-label»>Поиск по
названию</label>
                    <input
                      type=«text»
                      className=«form-control»
                      placeholder=«Введите название»
                      value={searchQuery}
                      onChange={(e) =>
setSearchQuery(e.target.value)}
                    />
                  </div>
                </div>
                <div className=«row»>
                  <div className=«col-md-3 mb-3»>
                    <label className=«form-
label»>Сортировка</label>
                    <select
                      className=«form-control»
                      value={sortOption}
                      onChange={(e) =>
setSortOption(e.target.value)}
                    >
                      <option value=««>Без сортировки</option>
                      <option value=«nameAsc»>По названию (А-
Я)</option>
                      <option value=«nameDesc»>По названию (Я-
А)</option>
                      <option value=«dateAsc»>По дате (сначала
старые)</option>
                      <option value=«dateDesc»>По дате (сначала
новые)</option>
                    </select>
                  </div>
                  <div className=«col-md-3 mb-3 d-flex align-
items-end»>
                    <button
                      className=«btn btn-secondary w-100»
                      onClick={resetFilters}
                    >
                      Сбросить фильтры
                    </button>
                  </div>
                </div>
              </div>
              {sortedEvents.length === 0 ? (
                <p>Нет мероприятий.</p>
              ) : (
                sortedEvents.map(event => (
                  <div key={event.id} className=«card mb-3»>
                    <div className=«card-body»>
                      <h5 className=«card-title»>{event.name}</h5>
```

```
                      <p className=«card-text»>
                        <strong>Тип:</strong>
{event.eventType?.name || 'Не указан'}<br />
                        <strong>Дата:</strong>
{event.eventDate}<br />
                        <strong>Описание:</strong>
{event.description}<br />
                        <strong>Место:</strong>
{event.location}<br />
                        <strong>Всего мест:</strong>
{event.totalSeats}<br />
                        <strong>Статус:</strong>
{event.eventStatus?.name || 'Не указан'}<br />
                        <strong>Изображение:</strong>{' '}
                        {event.imagePath ? (
                          <img
src={`https://localhost:7109${event.imagePath}`} alt={event.name}
style={{ maxWidth: '100px' }} />
                        ) : (
                          'Нет изображения'
                        )}
                      </p>
                      <div className=«d-flex gap-2»>
                        <button
                          className=«btn btn-primary»
                          onClick={() => {
                            setEditEvent({
                              ...event,
                              eventStatusId:
event.eventStatus?.id,
                            });
                            setActiveSection('editEvent');
                          }}
                        >
                          Редактировать
                        </button>
                        <button
                          className=«btn btn-warning»
                          onClick={() => {
                            setEditEvent(event);
                            setActiveSection('addSeats');
                          }}
                        >
                          Управление местами
                        </button>
                        <button
                          className=«btn btn-danger»
                          onClick={() =>
handleDeleteEvent(event.id)}
                        >
                          Удалить
                        </button>
```

```
                    </div>
                  </div>
                </div>
              ))
            )}
          </div>
        )}
        {activeSection === 'addEvent' && (
          <div>
            <h3>Добавить мероприятие</h3>
            <form onSubmit={handleCreateEvent}>
              <div className=«mb-3»>
                <label className=«form-label»>Название</label>
                <input
                  type=«text»
                  className=«form-control»
                  value={newEvent.name}
                  onChange={(e) => setNewEvent({ ...newEvent,
name: e.target.value })}
                  required
                />
              </div>
              <div className=«mb-3»>
                <label className=«form-label»>Тип
мероприятия</label>
                <select
                  className=«form-control»
                  value={newEvent.eventTypeId}
                  onChange={(e) => setNewEvent({ ...newEvent,
eventTypeId: e.target.value })}
                  required
                >
                  <option value=««>Выберите тип</option>
                  {eventTypes.map(type => (
                    <option key={type.id}
value={type.id}>{type.name}</option>
                  ))}
                </select>
              </div>
              <div className=«mb-3»>
                <label className=«form-label»>Дата и время
(формат: dd.MM.yyyy HH:mm)</label>
                <input
                  type=«text»
                  className=«form-control»
                  value={newEvent.eventDate}
                  onChange={(e) => setNewEvent({ ...newEvent,
eventDate: e.target.value })}
                  placeholder=«например, 22.03.2025 18:53»
                  required
                />
              </div>
```

```
                <div className=«mb-3»>
                  <label className=«form-label»>Описание</label>
                  <textarea
                    className=«form-control»
                    value={newEvent.description}
                    onChange={(e) => setNewEvent({ ...newEvent,
description: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Место
проведения</label>
                  <input
                    type=«text»
                    className=«form-control»
                    value={newEvent.location}
                    onChange={(e) => setNewEvent({ ...newEvent,
location: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Всего мест</label>
                  <input
                    type=«number»
                    className=«form-control»
                    value={newEvent.totalSeats}
                    onChange={(e) => setNewEvent({ ...newEvent,
totalSeats: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
                  <label className=«form-
label»>Изображение</label>
                  <input
                    type=«file»
                    className=«form-control»
                    onChange={(e) => setNewEvent({ ...newEvent,
image: e.target.files[0] })}
                  />
                </div>
                <button type=«submit» className=«btn btn-primary
mb-3»>Создать</button>
              </form>
            </div>
          )}
          {activeSection === 'editEvent' && editEvent && (
            <div>
              <h3>Редактировать мероприятие</h3>
              <form onSubmit={handleUpdateEvent}>
```

```
                <div className=«mb-3»>
                  <label className=«form-label»>Название</label>
                  <input
                    type=«text»
                    className=«form-control»
                    value={editEvent.name}
                    onChange={(e) => setEditEvent({ ...editEvent,
name: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Тип
мероприятия</label>
                  <select
                    className=«form-control»
                    value={editEvent.eventTypeId}
                    onChange={(e) => setEditEvent({ ...editEvent,
eventTypeId: e.target.value })}
                    required
                  >
                    <option value=««>Выберите тип</option>
                    {eventTypes.map(type => (
                      <option key={type.id}
value={type.id}>{type.name}</option>
                    ))}
                  </select>
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Дата и время
(формат: dd.MM.yyyy HH:mm)</label>
                  <input
                    type=«text»
                    className=«form-control»
                    value={editEvent.eventDate}
                    onChange={(e) => setEditEvent({ ...editEvent,
eventDate: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Описание</label>
                  <textarea
                    className=«form-control»
                    value={editEvent.description}
                    onChange={(e) => setEditEvent({ ...editEvent,
description: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
```

```
                  <label className=«form-label»>Место
проведения</label>
                  <input
                    type=«text»
                    className=«form-control»
                    value={editEvent.location}
                    onChange={(e) => setEditEvent({ ...editEvent,
location: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Всего мест</label>
                  <input
                    type=«number»
                    className=«form-control»
                    value={editEvent.totalSeats}
                    onChange={(e) => setEditEvent({ ...editEvent,
totalSeats: e.target.value })}
                    required
                  />
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Статус</label>
                  <select
                    className=«form-control»
                    value={editEvent.eventStatusId}
                    onChange={(e) => setEditEvent({ ...editEvent,
eventStatusId: e.target.value })}
                    required
                  >
                    <option value=«1»>Активное</option>
                    <option value=«2»>Завершено</option>
                    <option value=«3»>Отменено</option>
                  </select>
                </div>
                <div className=«mb-3»>
                  <label className=«form-label»>Новое
изображение</label>
                  <input
                    type=«file»
                    className=«form-control»
                    onChange={(e) => setEditEvent({ ...editEvent,
image: e.target.files[0] })}
                  />
                </div>
                <button type=«submit» className=«btn btn-primary
mb-3»>Сохранить</button>
                <button
                  type=«button»
                  className=«btn btn-secondary ms-2 mb-3»
                  onClick={() => setEditEvent(null)}
```

```
                >
                  Отмена
                </button>
              </form>
            </div>
          )}
          {activeSection === 'addSeats' && editEvent && (
            <div>
              <h3>Управление местами для мероприятия:
{editEvent.name}</h3>
              <div className=«mb-3»>
                <label className=«form-check-label»>
                  <input
                    type=«checkbox»
                    className=«form-check-input»
                    checked={showBlocks}
                    onChange={() => setShowBlocks(!showBlocks)}
                  />
                  Использовать блоки (например, А, Б, В)
                </label>
              </div>
              <h4>Добавление новых мест</h4>
              {eventSeats.map((block, index) => (
                <div key={index} className=«border p-3 mb-3»>
                  {showBlocks && (
                    <div className=«mb-3»>
                      <label className=«form-label»>Название
блока</label>
                      <input
                        type=«text»
                        className=«form-control»
                        value={block.blockName || ''}
                        onChange={(e) => {
                          const updatedSeats = [...eventSeats];
                          updatedSeats[index].blockName =
e.target.value;
                          setEventSeats(updatedSeats);
                        }}
                        required
                      />
                    </div>
                  )}
                  <div className=«mb-3»>
                    <label className=«form-label»>Количество
мест</label>
                    <input
                      type=«number»
                      className=«form-control»
                      value={block.seatCount || ''}
                      onChange={(e) => {
                        const updatedSeats = [...eventSeats];
```

```
                        updatedSeats[index].seatCount =
e.target.value;
                        setEventSeats(updatedSeats);
                      }}
                      required
                    />
                  </div>
                  <div className=«mb-3»>
                    <label className=«form-label»>Цена за
место</label>
                    <input
                      type=«number»
                      className=«form-control»
                      value={block.price || ''}
                      onChange={(e) => {
                        const updatedSeats = [...eventSeats];
                        updatedSeats[index].price =
e.target.value;
                        setEventSeats(updatedSeats);
                      }}
                      step=«0.01»
                      required
                    />
                  </div>
                  <button
                    type=«button»
                    className=«btn btn-danger»
                    onClick={() =>
setEventSeats(eventSeats.filter((_, i) => i !== index))}
                  >
                    Удалить
                  </button>
                </div>
              ))}
              <button
                type=«button»
                className=«btn btn-secondary mb-3»
                onClick={() => setEventSeats([...eventSeats, {
blockName: '', seatCount: '', price: '' }])}
              >
                Добавить {showBlocks ? 'блок' : 'места'}
              </button>
              <div className=«d-flex gap-2 mb-4»>
                <button
                  type=«button»
                  className=«btn btn-primary»
                  onClick={() =>
handleAddEventSeats(editEvent.id)}
                  disabled={eventSeats.length === 0}
                >
                  Сохранить новые места
                </button>
```

```
                  <button
                    type=«button»
                    className=«btn btn-secondary»
                    onClick={() => {
                      setEventSeats([]);
                      setShowBlocks(false);
                      setEditEvent(null);
                    }}
                  >
                    Отмена
                  </button>
                </div>
                <style>
                  {`
                    .btn-soft-lilac {
                      background-color: rgb(230, 138, 230);
                      border-color: #D8BFD8;
                      color: #333;
                    }
                    .btn-soft-lilac:hover {
                      background-color: #C1A3C1;
                      border-color: #C1A3C1;
                      color: #fff;
                    }
                  `}
                </style>
                <button
                  className=«btn btn-soft-lilac mb-3»
                  onClick={() =>
setShowExistingSeats(!showExistingSeats)}
                >
                  {showExistingSeats ? 'Скрыть существующие места' :
'Показать существующие места'}
                </button>
                {showExistingSeats && (
                  <div className=«mb-3»>
                    <h4>Существующие места</h4>
                    {existingSeats.length > 0 ? (
                      <>
                        {selectedSeats.length > 0 && (
                          <div className=«mb-3»>
                            <button
                              className=«btn btn-primary me-2»
                              onClick={() => {
                                setModalMode('bulk');
                                setSelectedSeat({ price: '' });
                                setShowModal(true);
                                setErrors({ seatNumber: '', price:
'' });

                              }}
                            >
                              Изменить
```

```
                            </button>
                            <button
                              className=«btn btn-danger me-2»
                              onClick={() => {
                                setModalMode('delete');
                                setShowModal(true);
                              }}
                            >
                              Удалить
                            </button>
                            <button
                              className=«btn btn-secondary»
                              onClick={clearSelection}
                            >
                              Снять выделение
                            </button>
                          </div>
                        )}
                        <div className=«mb-3»>
                          <label className=«form-label»>Мест на
странице:</label>
                          <select
                            className=«form-control w-auto d-inline-
block ms-2»
                            value={seatsPerPage}
                            onChange={(e) => {

setSeatsPerPage(parseInt(e.target.value));
                              setCurrentPage(1);
                            }}
                          >
                            <option value={5}>5</option>
                            <option value={10}>10</option>
                            <option value={20}>20</option>
                          </select>
                        </div>
                        <div className=«row»>
                          {currentSeats.map(seat => (
                            <div key={seat.id} className=«col-md-4
mb-3»>
                              <div className=«card seat-card»>
                                <div className=«card-body d-flex
align-items-start»>
                                  <div className=«form-check me-3»>
                                    <input
                                      type=«checkbox»
                                      className=«form-check-input»

checked={selectedSeats.includes(seat.id)}
                                      onChange={(e) => {
                                        e.stopPropagation();
```

```
toggleSeatSelection(seat.id);
                                    }}
                                  />
                                </div>
                                <div
                                  className=«seat-info flex-grow-
1»
                                  onClick={() => {
                                    setModalMode('single');
                                    setSelectedSeat(seat);
                                    setShowModal(true);
                                    setErrors({ seatNumber: '',
price: '' });
                                  }}
                                >
                                  <p className=«mb-
0»><strong>Место:</strong> {seat.seatNumber}</p>
                                  <p className=«mb-
0»><strong>Цена:</strong> {seat.price} руб.</p>
                                </div>
                              </div>
                            </div>
                          </div>
                        ))}
                      </div>
                      <div className=«pagination»>
                        <button
                          className=«btn btn-secondary»
                          onClick={prevPage}
                          disabled={currentPage === 1}
                        >
                          Предыдущая
                        </button>
                        {Array.from({ length: totalPages }, (_, i)
=> (
                          <button
                            key={i + 1}
                            className={`btn ${currentPage === i +
1 ? 'btn-primary' : 'btn-outline-primary'}`}
                            onClick={() => paginate(i + 1)}
                          >
                            {i + 1}
                          </button>
                        ))}
                        <button
                          className=«btn btn-secondary»
                          onClick={nextPage}
                          disabled={currentPage === totalPages}
                        >
                          Следующая
                        </button>
```

```
                        <span className=«ms-2»>
                          Страница {currentPage} из {totalPages}
                        </span>
                      </div>
                    </>
                  ) : (
                    <p>Нет существующих мест для этого
мероприятия.</p>
                  )}
                </div>
              )}
            </div>
          )}
          {showModal && (
            <div className=«modal»>
              <div className=«modal-content»>
                {modalMode === 'delete' ? (
                  <>
                    <h5>Подтверждение удаления</h5>
                    <p>Вы уверены, что хотите удалить
{selectedSeats.length} выделенных мест?</p>
                    <div className=«d-flex gap-2»>
                      <button
                        className=«btn btn-danger»
                        onClick={() => handleBulkDelete()}
                      >
                        Удалить
                      </button>
                      <button
                        className=«btn btn-secondary»
                        onClick={() => {
                          setShowModal(false);
                          setModalMode('single');
                        }}
                      >
                        Отмена
                      </button>
                    </div>
                  </>
                ) : (
                  <>
                    <h5>{modalMode === 'single' ? 'Редактировать
место' : 'Массовое редактирование'}</h5>
                    {modalMode === 'single' && (
                      <div className=«mb-3»>
                        <label className=«form-label»>Номер
места</label>
                        <input
                          type=«text»
                          className={`form-control
${errors.seatNumber ? 'is-invalid' : ''}`}
                          value={selectedSeat?.seatNumber || ''}
```

```
                          onChange={(e) => setSelectedSeat({
...selectedSeat, seatNumber: e.target.value })}
                        />
                        {errors.seatNumber && <div
className=«error-text»>{errors.seatNumber}</div>}
                      </div>
                    )}
                    <div className=«mb-3»>
                      <label className=«form-label»>Цена за
место</label>
                      <input
                        type=«number»
                        className={`form-control ${errors.price ?
'is-invalid' : ''}`}
                        value={selectedSeat?.price || ''}
                        onChange={(e) => setSelectedSeat({
...selectedSeat, price: e.target.value })}
                        step=«0.01»
                        placeholder={modalMode === 'bulk' ?
'Оставьте пустым, чтобы не менять' : ''}
                      />
                      {errors.price && <div className=«error-
text»>{errors.price}</div>}
                    </div>
                    <div className=«d-flex gap-2»>
                      <button
                        className=«btn btn-primary»
                        onClick={() => {
                          if (modalMode === 'single') {
                            handleUpdateEventSeat(selectedSeat.id,
{
                              seatNumber: selectedSeat.seatNumber,
                              price: selectedSeat.price,
                            });
                          } else {
                            handleBulkUpdate({
                              price: selectedSeat.price,
                            });
                          }
                        }}
                      >
                        Сохранить
                      </button>
                      {modalMode === 'single' && (
                        <button
                          className=«btn btn-danger»
                          onClick={() => {

handleDeleteEventSeat(selectedSeat.id);
                            setShowModal(false);
                          }}
                        >
```

```
                        Удалить
                      </button>
                    )}
                    <button
                      className=«btn btn-secondary»
                      onClick={() => {
                        setShowModal(false);
                        setSelectedSeat(null);
                        setErrors({ seatNumber: '', price: ''
});

                      }}
                    >
                      Отмена
                    </button>
                  </div>
                </>
              )}
            </div>
          </div>
        )}
      </div>
    </div>
    <div className=«notifications»>
      {notifications.map(notif => (
        <div key={notif.id} className={`notification
${notif.type}`}>
          {notif.message}
        </div>
      ))}
    </div>
  </div>
);
};

export default AdminPanel;
```

Приложение Г

(обязательное)

Скрипт создания базы данных

Листинг Г.1 – Скрипт создания базы данных emUser

```
CREATE DATABASE emUser;

CREATE TABLE UserType (
    id INT PRIMARY KEY IDENTITY,
    name NVARCHAR(20) UNIQUE
);

CREATE TABLE Users (
    id INT PRIMARY KEY IDENTITY,
    name NVARCHAR(100),
    email NVARCHAR(50) UNIQUE,
    phone NVARCHAR(15),
     password NVARCHAR(500),
     userTypeId int,
    FOREIGN KEY (userTypeId)  REFERENCES UserType(id)
);
```

Листинг Г.2 – Скрипт создания базы данных emEvent

```
CREATE DATABASE emEvent;

CREATE TABLE EventTypes (
    id INT PRIMARY KEY IDENTITY,
    name NVARCHAR(30) UNIQUE NOT NULL
);

CREATE TABLE Events (
    id INT PRIMARY KEY IDENTITY,
    name NVARCHAR(100),
    eventTypeId INT,
    event_date DATE,
     description NVARCHAR(MAX) NULL,
    location NVARCHAR(100),
    total_seats INT NOT NULL,
     image_path VARCHAR(255) NULL,
    FOREIGN KEY (eventTypeId) REFERENCES EventTypes(id)
);

CREATE TABLE EventSeats (
    id INT PRIMARY KEY IDENTITY,
    eventId INT NOT NULL,
    seat_number NVARCHAR(10) NOT NULL,
    is_reserved BIT DEFAULT 0,
     price DECIMAL(10, 2) NOT NULL DEFAULT 0.00,
    FOREIGN KEY (eventId) REFERENCES Events(id),
    CONSTRAINT UC_EventSeats UNIQUE (eventId, seat_number)
);

CREATE TABLE EventStatus (
     id INT PRIMARY KEY IDENTITY,
    name NVARCHAR(30) UNIQUE NOT NULL
```

```
);

ALTER TABLE Events
ADD eventStatusId INT NOT NULL DEFAULT 1;

ALTER TABLE Events
ADD CONSTRAINT FK_Events_EventStatus_EventStatusId
FOREIGN KEY (eventStatusId)
REFERENCES EventStatus (Id);
ALTER TABLE Events ALTER COLUMN eventDate DATE NOT NULL;

UPDATE Events
SET imagePath = ''
WHERE imagePath IS NULL;

ALTER TABLE Events
ADD EventDate_New DATETIME2
CONSTRAINT DF_Events_EventDate_New DEFAULT GETDATE() WITH VALUES;

ALTER TABLE Events
DROP COLUMN EventDate;

EXEC sp_rename 'Events.EventDate_New', 'EventDate', 'COLUMN';

ALTER TABLE Events
DROP CONSTRAINT DF_Events_EventDate_New;
```

Листинг Г.3 – Скрипт создания базы данных emTickets

```
CREATE DATABASE emTickets;

CREATE TABLE TicketStatus (
    id INT PRIMARY KEY IDENTITY,
    name NVARCHAR(20) UNIQUE
);

CREATE TABLE Tickets (
    ticketId INT PRIMARY KEY IDENTITY,
    userId INT NOT NULL,
     eventName NVARCHAR(100) NOT NULL,
    eventSeatId INT NOT NULL,
     seatNumber NVARCHAR(10),
     eventDate DATETIME2,
    price DECIMAL(8,2),
    statusId INT NOT NULL,
    purchaseDate DATETIME2 NULL,
    reservationDate DATETIME2 NULL,
    FOREIGN KEY (statusId) REFERENCES TicketStatus(id)
);
```

Приложение Д

(рекомендуемое)

Листинг модульных тестов

```
using Xunit;
using Moq;
using UserAPI.Services;
using UserAPI.Data;
using UserAPI.Models;
using Microsoft.Extensions.Configuration;
using System.Threading.Tasks;
using System.Security.Claims;

namespace UserAPI.Tests
{
    public class UsersServiceTests
    {
        private readonly Mock<IUserRepository>
_userRepositoryMock;
        private readonly Mock<IConfiguration> _configurationMock;
        private readonly UsersService _service;

        public UsersServiceTests()
        {
            _userRepositoryMock = new Mock<IUserRepository>();
            _configurationMock = new Mock<IConfiguration>();
            _configurationMock.Setup(c =>
c[«Jwt:Key»]).Returns(«your_super_secret_key_32_chars!!»);
            _configurationMock.Setup(c =>
c[«Jwt:Issuer»]).Returns(«UsersAPI»);
            _configurationMock.Setup(c =>
c[«Jwt:Audience»]).Returns(«UsersAPI»);
            _service = new
UsersService(_userRepositoryMock.Object,
_configurationMock.Object);
        }

        [Fact]
        public async Task RegisterAsync_NewUser_ShouldCreateUser()
        {
            var email = «test@example.com»;
            var password = «password123»;
            var name = «Test User»;
            var phone = «1234567890»;
            var hashedPassword =
BCrypt.Net.BCrypt.HashPassword(password);
            var expectedUser = new User { Id = 1, Email = email,
Name = name, Phone = phone, UserTypeId = 2, Password =
hashedPassword };
            _userRepositoryMock.Setup(r =>
r.UserExistsAsync(email)).ReturnsAsync(false);
            _userRepositoryMock.Setup(r =>
r.AddUserAsync(It.IsAny<User>())).ReturnsAsync(expectedUser);

            var result = await _service.RegisterAsync(email,
password, name, phone);
```

```
            Assert.NotNull(result);
            Assert.Equal(email, result.Email);
            Assert.Equal(name, result.Name);
            Assert.Equal(phone, result.Phone);
            Assert.True(BCrypt.Net.BCrypt.Verify(password,
result.Password));
            Assert.Equal(2, result.UserTypeId);
        }

        [Fact]
        public async Task RegisterAsync_ExistingUser_ReturnsNull()
        {
            var email = «test@example.com»;
            _userRepositoryMock.Setup(r =>
r.UserExistsAsync(email)).ReturnsAsync(true);

            var result = await _service.RegisterAsync(email,
«password123», «Test User», «1234567890»);

            Assert.Null(result);
        }

        [Fact]
        public async Task
VerifyPasswordAsync_ValidCredentials_ReturnsTrue()
        {
            var email = «test@example.com»;
            var password = «password123»;
            var hashedPassword =
BCrypt.Net.BCrypt.HashPassword(password);
            var user = new User { Id = 1, Email = email, Password
= hashedPassword, UserTypeId = 2 };
            _userRepositoryMock.Setup(r =>
r.GetUserByEmailAsync(email)).ReturnsAsync(user);

            var result = await _service.VerifyPasswordAsync(email,
password);

            Assert.True(result);
        }

        [Fact]
        public async Task
VerifyPasswordAsync_InvalidEmail_ReturnsFalse()
        {
            var email = «wrong@example.com»;
            _userRepositoryMock.Setup(r =>
r.GetUserByEmailAsync(email)).ReturnsAsync((User)null);

            var result = await _service.VerifyPasswordAsync(email,
«password123»);
```

```
            Assert.False(result);
        }

        [Fact]
        public async Task
VerifyPasswordAsync_InvalidPassword_ReturnsFalse()
        {
            var email = «test@example.com»;
            var password = «wrongpassword»;
            var hashedPassword =
BCrypt.Net.BCrypt.HashPassword(«password123»);
            var user = new User { Id = 1, Email = email, Password
= hashedPassword, UserTypeId = 2 };
            _userRepositoryMock.Setup(r =>
r.GetUserByEmailAsync(email)).ReturnsAsync(user);

            var result = await _service.VerifyPasswordAsync(email,
password);

            Assert.False(result);
        }

        [Fact]
        public async Task
GetUserByEmailAsync_ExistingUser_ReturnsUser()
        {

            var email = «test@example.com»;
            var expectedUser = new User { Id = 1, Email = email,
Name = «Test User», UserTypeId = 2 };
            _userRepositoryMock.Setup(r =>
r.GetUserByEmailAsync(email)).ReturnsAsync(expectedUser);


            var result = await
_service.GetUserByEmailAsync(email);


            Assert.NotNull(result);
            Assert.Equal(expectedUser.Id, result.Id);
            Assert.Equal(expectedUser.Email, result.Email);
        }


        [Fact]
        public async Task
UpdateUserNameAsync_ValidId_UpdatesName()
        {

            var userId = 1;
            var newName = «New Name»;
```

```
            var user = new User { Id = userId, Email =
«test@example.com», Name = «Old Name», UserTypeId = 2 };
            _userRepositoryMock.Setup(r =>
r.GetUserByIdAsync(userId)).ReturnsAsync(user);
            _userRepositoryMock.Setup(r =>
r.UpdateUserAsync(It.IsAny<User>())).ReturnsAsync((User u) => u);


            var result = await
_service.UpdateUserNameAsync(userId, newName);


            Assert.NotNull(result);
            Assert.Equal(newName, result.Name);
        }

        [Fact]
        public async Task
UpdateUserNameAsync_InvalidId_ReturnsNull()
        {

            var userId = 999;
            _userRepositoryMock.Setup(r =>
r.GetUserByIdAsync(userId)).ReturnsAsync((User)null);


            var result = await
_service.UpdateUserNameAsync(userId, «New Name»);


            Assert.Null(result);
        }

        [Fact]
        public async Task
UpdateUserPasswordAsync_ValidCredentials_ReturnsTrue()
        {

            var userId = 1;
            var oldPassword = «oldpassword»;
            var newPassword = «newpassword»;
            var hashedOldPassword =
BCrypt.Net.BCrypt.HashPassword(oldPassword);
            var user = new User { Id = userId, Email =
«test@example.com», Password = hashedOldPassword, UserTypeId = 2
};
            _userRepositoryMock.Setup(r =>
r.GetUserByIdAsync(userId)).ReturnsAsync(user);
            _userRepositoryMock.Setup(r =>
r.UpdateUserAsync(It.IsAny<User>())).ReturnsAsync((User u) => u);
```

```
            var result = await
_service.UpdateUserPasswordAsync(userId, oldPassword,
newPassword);


            Assert.True(result);
            Assert.True(BCrypt.Net.BCrypt.Verify(newPassword,
user.Password));
        }

        [Fact]
        public async Task
UpdateUserPasswordAsync_InvalidOldPassword_ReturnsFalse()
        {

            var userId = 1;
            var oldPassword = «wrongpassword»;
            var newPassword = «newpassword»;
            var hashedOldPassword =
BCrypt.Net.BCrypt.HashPassword(«correctpassword»);
            var user = new User { Id = userId, Email =
«test@example.com», Password = hashedOldPassword, UserTypeId = 2
};
            _userRepositoryMock.Setup(r =>
r.GetUserByIdAsync(userId)).ReturnsAsync(user);


            var result = await
_service.UpdateUserPasswordAsync(userId, oldPassword,
newPassword);


            Assert.False(result);
        }

        [Fact]
        public async Task
UpdateUserPasswordAsync_InvalidId_ReturnsFalse()
        {

            var userId = 999;
            _userRepositoryMock.Setup(r =>
r.GetUserByIdAsync(userId)).ReturnsAsync((User)null);


            var result = await
_service.UpdateUserPasswordAsync(userId, «oldpassword»,
«newpassword»);


            Assert.False(result);
        }
```

```
        }
    }
```

Приложение Е

(рекомендуемое)

Результаты тестирования черным ящиком

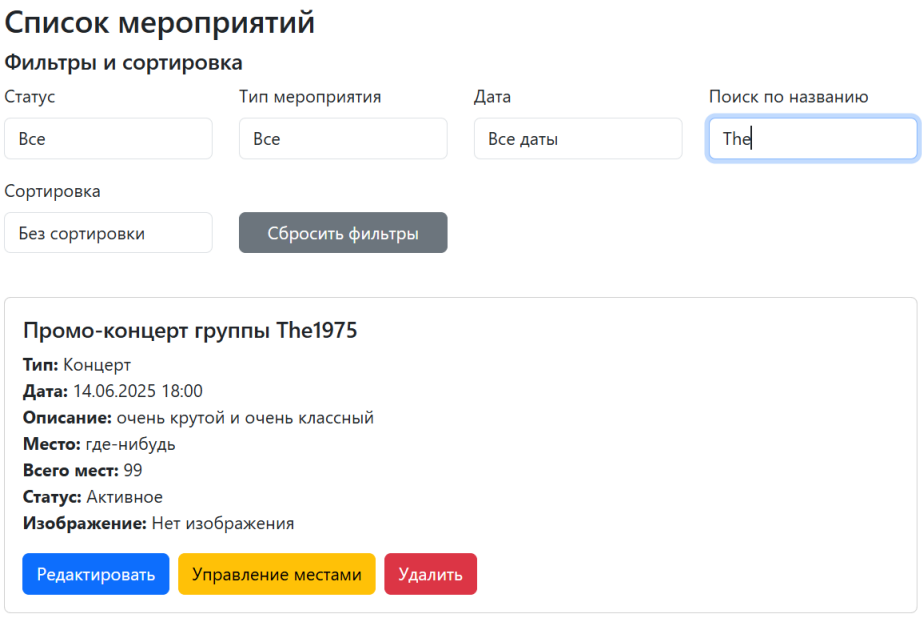

Рисунок Е.1 – Успешно добавленное мероприятие

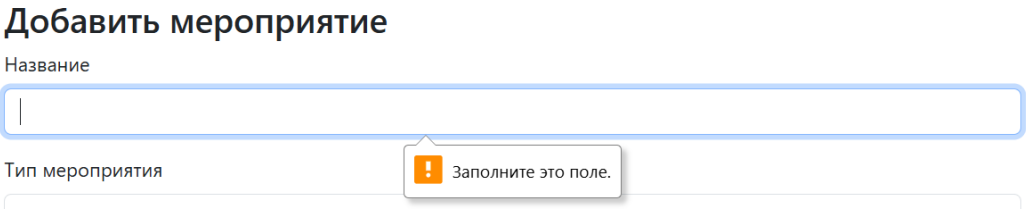

Рисунок Е.2 – Сообщение о пустом поле

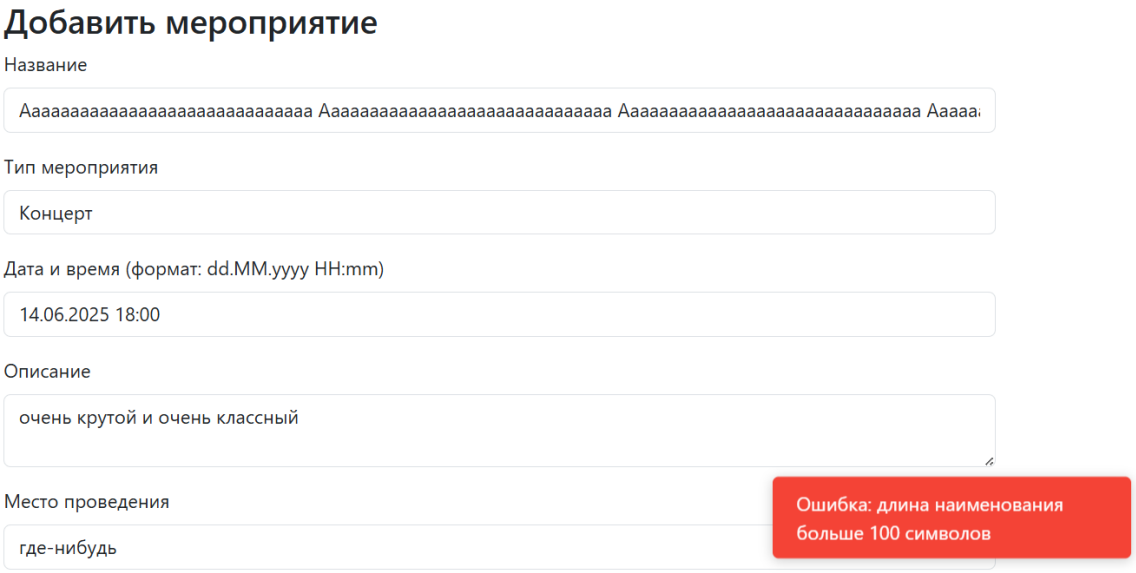

Рисунок Е.3 – Сообщение о слишком длинном названии

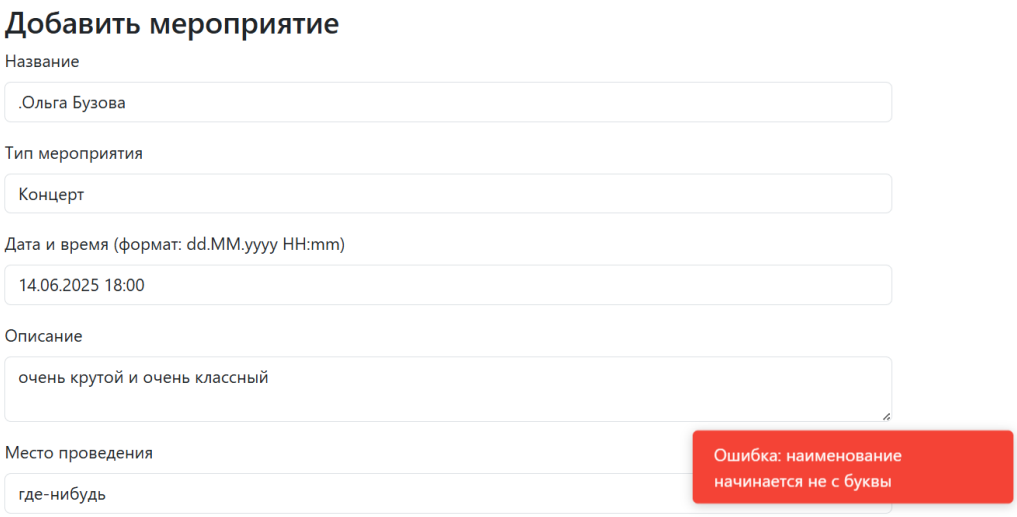

Рисунок Е.4 – Сообщение о некорректном вводе названия

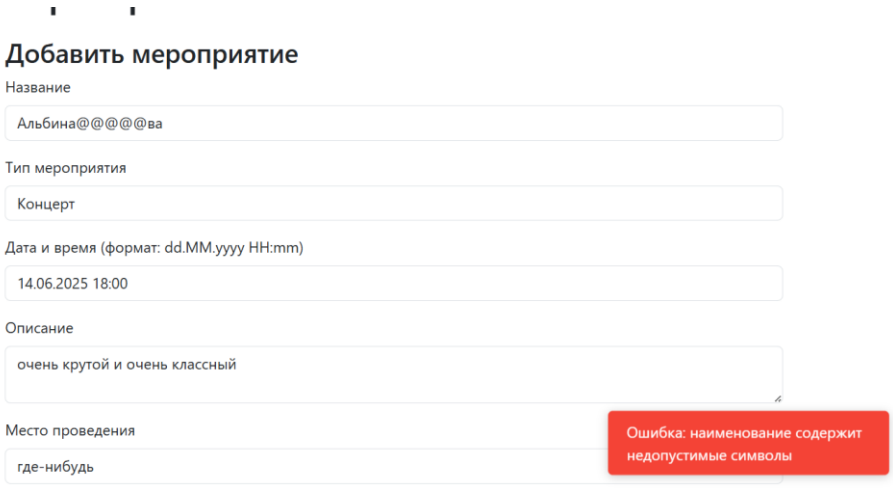

Рисунок Е.5 – Сообщение о некорректном вводе названия

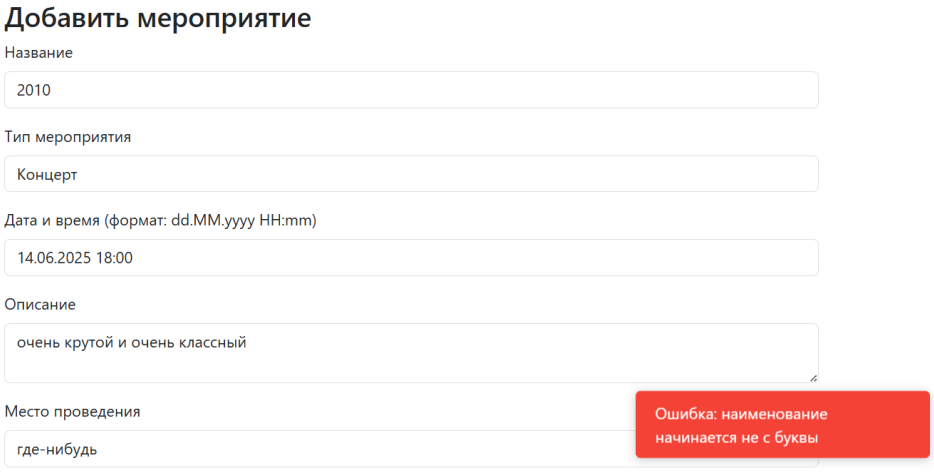

Рисунок Е.6 – Сообщение о некорректном вводе названия

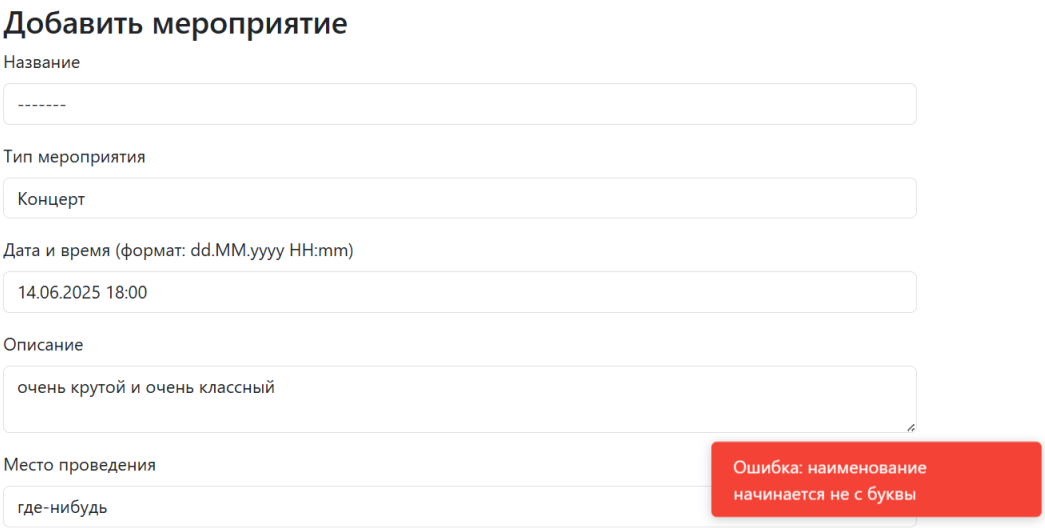

Рисунок Е.7 – Сообщение о некорректном вводе названия

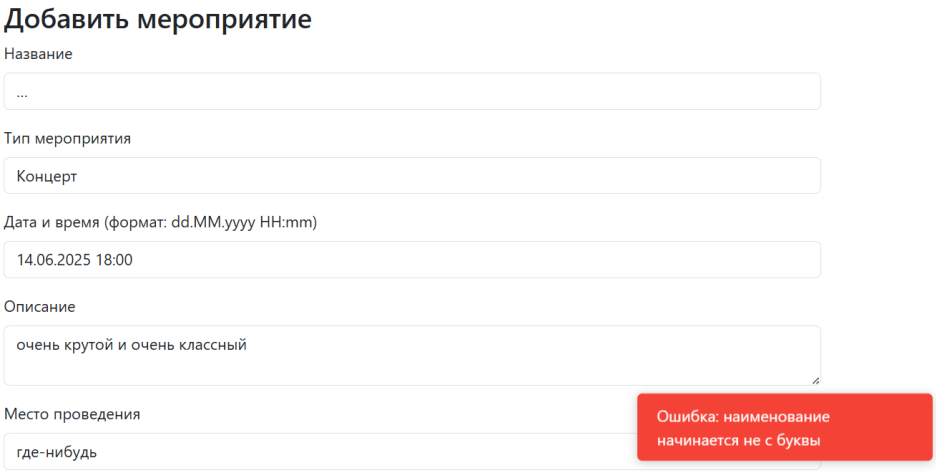

Рисунок Е.8 – Сообщение о некорректном вводе названия

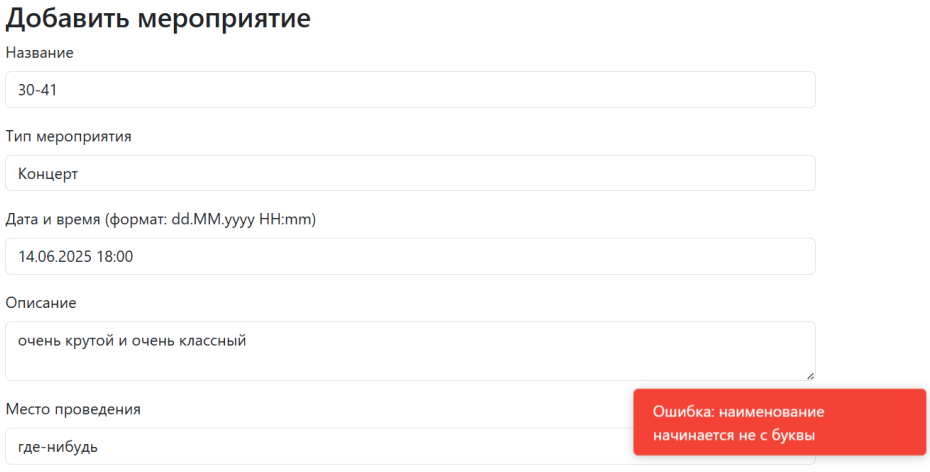

Рисунок Е.9 – Сообщение о некорректном вводе названия

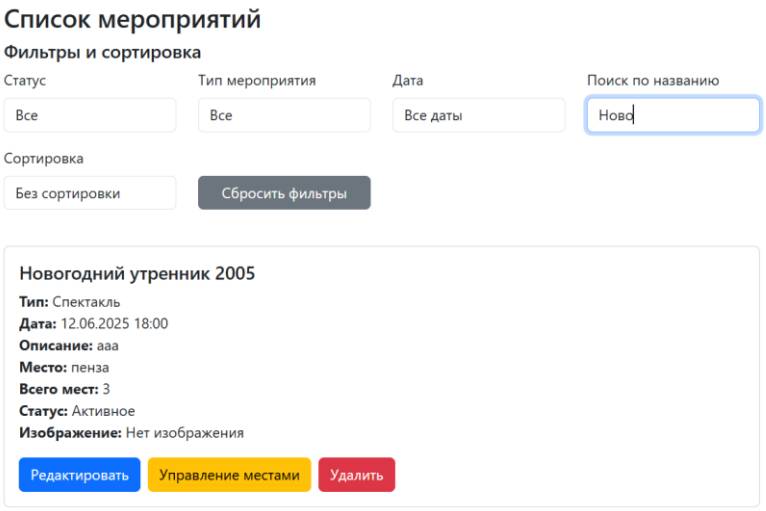

Рисунок Е.10 – Успешно добавленное мероприятие

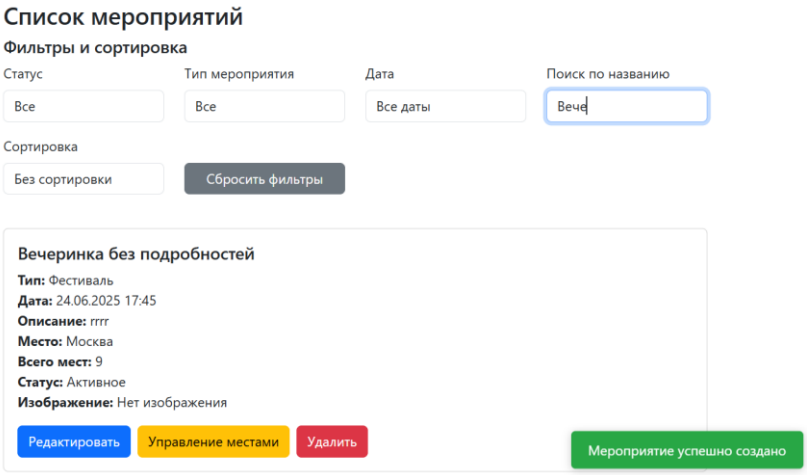

Рисунок Е.11 – Успешно добавленное мероприятие

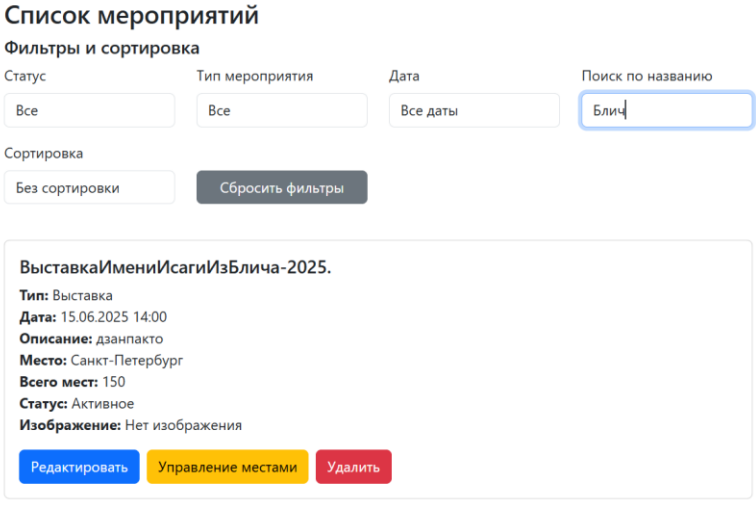

Рисунок Е.12 – Успешно добавленное мероприятие