МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

СТАРТАП-ПРОЕКТ. СИСТЕМА БРОНИРОВАНИЯ ГОСТИНИЧНЫХ НОМЕРОВ

Студент 27.05.24 Александров Владимир Сергеевич
(подпись, дата) (ФИО полностью)

Руководитель 27.05.24 М. П. Синёв
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 27.05.24 А. С. Бычков
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.24 г. № 11)

Заведующий кафедрой М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 04.06.2024 № 2)

Секретарь ГЭК Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2024

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____ М.А. Митрохин
«19» 04 2024 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Александров Владимир Сергеевич гр. 20ВВП1 факультета вычислительная техника

2.Тема работы «Стартап-проект. Система бронирования гостиничных номеров»

Тема утверждена приказом ПГУ № 405/о от "19" апреля 2024г.

3. Руководитель бакалаврской работы к.т.н. доцент кафедры ВТ ПГУ Синёв М. П.

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

Назначение разработки - Система бронирования гостиничных номеров.

Разработать автоматизированную информационную систему "Система бронирования гостиничных номеров", использующую React для веб-клиента, Spring Framework на стороне сервера и базу данных PostgreSQL. Составить бизнес-модель и финансовый план. Язык программирования Java, JavaScript, PostgreSQL, среда разработки IntelliJ IDEA, WebStorm, PgAdmin. Программа должна обладать интуитивно понятным интерфейсом

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать инструменты для разработки, реализовать приложение.

6. Календарный график выполнения работы

п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10 %	9.04.24 - 25.04.24	
2.	Выбор архитектуры системы и технологий, применяемых при проектировании и разработке	20 %	26.04.24 - 02.05.24	
3.	Разработка интерфейсов	20 %	03.05.24 - 07.05.24	
4.	Разработка логики системы	20 %	08.05.24 - 19.05.24	
5	Тестирование и отладка	10 %	20.05.24 - 22.05.24	
6.	Выполнение задания по БЖД	5 %	23.05.24 - 25.05.24	
7	Оформление выпускной работы	15 %	25.05.23 - 31.05.23	

Дата выдачи задания "19" апреля 2024 г.

Руководитель бакалаврской работы _____ 19.04.24 М.П. Синев
(подпись, дата)

Задание к исполнению принял студент _____ 19.04.24 В. С. Александров
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 31.05.2024 Л.Р. Фионова
(подпись, дата
(подпись, дата)

Реферат

Бакалаврская работа на тему «Стартап-проект. Система бронирования гостиничных номеров» содержит 147 листов, 34 рисунка, 7 использованных источников, 3 приложения

АВТОМАТИЗИРОВАННАЯ ИНФОРМАЦИОННАЯ СИСТЕМА, БРОНИРОВАНИЕ ГОСТНИЧНЫХ НОМЕРОВ, ВЕБ-КЛИЕНТ, БАЗА ДАННЫХ PostgreSQL

Цель работы – создание стартап-проекта, посвященного разработке системы бронирования гостиничных номеров.

Объект исследования - автоматизированная информационная система бронирования гостиничных номеров, использующую React для веб-клиента, Spring Framework на стороне сервера и базу данных PostgreSQL.

В процессе работы выполнен анализ предметной области, аналитический обзор аналогов разрабатываемой автоматизированной информационной системы бронирования гостиничных номеров «Hotellook», «Trivago», «Oktogo», «Ostrovok» и др. Разработана автоматизированная информационная система бронирования гостиничных номеров, которая обладает различными функциональными возможностями: добавление, редактирование и удаление данных из номерного фонда, а также управление бронированием и т.д. Реализована система аутентификации и авторизации пользователей для обеспечения безопасности данных. Разработан бизнес – план продвижения системы бронирования среди целевой аудитории.

Проект разработан с использованием таких технологий, как React.js для фронтенда, Spring Framework для бэкенда и база данных PostgreSQL для хранения данных.

					ПГУ 1.090301.01.001 ПЗ
		№ докум.	Подпись		
Разраб	Александров В.С.		27.05.24	Система бронирования гостиничных номеров	Лит. / Лист. 2 / Листов 147
Провер.	Синёв М.П.		27.05.24		
Н. Контр.	Бычков А.С.		27.05.24		
					ФВТ каф. ВТ
Утвер.	Митрохин М.А.		27.05.24		Гр. 20ВВП1

Содержание

Перечень сокращений и обозначений 6

Введение 6

1 Анализ предметной области 8

1.1 Описание предметной области 8

1.2 Актуальность предметной области 14

1.3 Анализ аналогов 13

1.3.2 Система «Hotellook» 13

1.3.2 Система «Trivago» 13

1.3.3 Система «Oktogo» 14

1.3.4 Система «Ostrovok» 15

1.3.5 Вывод по анализу аналогов 15

2 Бизнес-план 17

2.1 Целевая аудитория 17

2.2 План развития продукта 17

2.2 Финансовый план 19

3 Постановка задачи 22

3.1 Требования к системе 22

3.1.1 Технические требования к системе 22

3.1.2 Функциональные требования к системе 22

3.1.3 База данных PostgreSQL 23

3.1.4 Сервер на Spring Framework 23

3.1.6 Веб-клиент на React 24

3.1.7 Безопасность 24

3.2 Архитектура приложения 25

3.2.1 База данных (PostgreSQL) 26

3.2.2 Сервер приложений (Spring Framework) 29

3.2.3 Описание API системы 29
3.2.4 Веб-клиент (React) 30
4 Разработка системы 32
4.1 Настройка среды разработки 32
4.1.1 Настройка среды IntelliJ IDEA и Spring Boot 32
4.1.2 Настройка среды WebStorm и React 34
4.1.3 Выполнение настройки среды PgAdmin 36
4.2 Разработка сервера на Spring Boot 38
4.2.1 Реализация сервера 38
4.2.2 Реализация аутентификации в системе на Spring Boot 44
4.2.3 Реализация миграций в серверной части 47
4.3 Разработка клиентской части на React 50
4.3.1 Реализация окна авторизации 50
4.3.2 Реализация окна просмотра бронирования 52
4.3.3 Реализация главного окна 54
4.3.4 Реализация окна просмотра и бронирования номера 56
4.3.5 Реализация календаря 59
4.3.6 Реализация элемента плитки объявления 60
4.3.7 Реализация окна добавления объявления 61
4.3.8 Реализация карты 62
5. Руководство пользователя 65
6. Безопасность жизнедеятельности. 69
Заключение 72
Список использованных источников 77
Приложение А 74
Приложение Б 79
Приложение В 138

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ОС – операционная система

ПК – персональный компьютер

API (Application Programming Interface) – программный интерфейс приложения

IDE (Integrated Development Enviroment) – Интегрированная среда разработки

MVC (Model-View-Controller) – «Модель-Представление-Контроллер»

UML (Unified Modeling Language) – унифицированый язык моделирования

Введение

Проблема совершенствования систем бронирования гостиничных номеров открывает дискуссию вокруг одного из наиболее значимых аспектов современного туристического и гостиничного бизнеса. В эпоху цифровизации и ускоренного темпа жизни, потребности пользователей в удобных и быстрых сервисах для планирования поездок становятся всё более значимыми.

Президент Российской Федерации В.В. Путин во время оглашения послания Федеральному Собранию 29 февраля 2024 г. подчеркнул, что к 2030 году турпоток в России должен вырасти в два раза и достигнет 140 млн человек в год.

Гостиничный бизнес, стремясь удовлетворить эти потребности, активно внедряет информационные технологии, среди которых системы онлайн-бронирования играют ключевую роль.

Онлайн-бронирование гостиничных номеров не только облегчает процесс выбора и аренды жилья для путешественников, но и предоставляет отелям инструмент управления загрузкой, ценообразованием и клиентским сервисом. Система бронирования является связующим звеном между клиентами и поставщиками услуг, обеспечивая эффективное взаимодействие с максимальным удобством для обеих сторон.

В настоящее время рынок предлагает множество решений для бронирования гостиничных номеров, от международных платформ до индивидуальных систем отелей. Это создает как возможности, так и вызовы для разработчиков и управляющих гостиницами, стремящихся предложить уникальные услуги и преимущества для своих клиентов. В этом контексте, разработка собственной системы бронирования становится стратегическим решением, позволяющим выделиться на фоне конкурентов и удовлетворить специфические потребности аудитории, особенно после ухода некоторых иностранных систем.

Цель данного введения — обозначить актуальность и значимость создания системы бронирования гостиничных номеров, определить ключевые направления анализа и разработки, а также подчеркнуть роль информационных технологий в современном гостиничном бизнесе. Это работа направлена на создание эффективного, удобного для пользователя и технологически совершенного решения, которое станет надёжным инструментом в руках как путешественников, так и управляющих гостиницами.

1 Анализ предметной области

1.1 Описание предметной области

Предметная область бронирования гостиниц обширна и многогранна, охватывая всё, что связано с поиском, сравнением и бронированием различных видов жилья для временного проживания. Эта область включает в себя гостиницы и отели, хостелы, апартаменты, курортные зоны, предлагаемые для краткосрочной аренды.

1.1.1 Описание бизнес процесса

Регистрация профиля пользователя:

– регистрация. Пользователь вводит личные данные (электронная почта, логин) и пароль (рисунок 1). Система проверяет уникальность данных;

– авторизация: пользователь входит в систему, используя логин и пароль (рисунок 1);

– управление профилем: пользователи могут изменять свои контактные данные (рисунок 1).

Поиск и фильтрация номеров:

– ввод критериев: пользователи указывают даты заезда/выезда, количество гостей, предпочитаемый тип номера (рисунок 1);

– сортировка результатов по цене (рисунок 1).

Просмотр и выбор номера:

– детальная информация: описание номера, фотографии, условия бронирования и отмены (рисунок 1);

– выбор дополнительных опций: раннего заезда или позднего выезда.

Процесс бронирования:

– ввод данных: заполнение данных для бронирования (выбор номера, даты заезда и выезда, и других данных, (Рисунок. 1);

– подтверждение бронирования: проверка всех деталей и условий, согласие с политикой отмены бронирования (рисунок 1);

– интеграция с платёжными системами: выбор способа оплаты и проведение платежа через безопасный платежный шлюз (будет реализовано позже, Рисунок 1).

Оплата бронирований:

– обработка транзакции: система обеспечивает обработку платежей через внешние платежные системы (в данный системе реализовано не будет реализовано) (рисунок 1);

– подтверждение и чек: отправка электронного чека и подтверждения бронирования на электронную почту пользователя (рисунок 1).

Управление бронированием:

– просмотр текущих и прошлых бронирований: Пользователи могут просматривать свои бронирования, а также получать напоминания о предстоящих поездках (рисунок 1);

– изменение и отмена бронирования: возможность изменения или отмены бронирования согласно политике отеля (рисунок 1);

– управление номерным фондом администраторами;

– добавление и удаление номеров: администраторы могут добавлять новые или удалять существующие номера из системы, что позволяет гостинице гибко управлять предложением в зависимости от сезона или спроса(рисунок 1);

– обновление информации о номерах: администраторы могут изменять информацию о номерах, включая описание, фотографии, цены что способствует актуализации данных и улучшению привлекательности предложения(рисунок 1);

– управление ценовой политикой: администраторы имеют возможность настройки цен в зависимости от различных факторов, таких как загрузка отеля, предстоящие события или специальные предложения, обеспечивая тем самым оптимальное ценообразование (рисунок 1);

– мониторинг загрузки номеров: систематический анализ и отслеживание статистики загрузки помогает в определении трендов и планировании необходимых корректировок в управлении номерным фондом;

– взаимодействие с отделом обслуживания: Обеспечение своевременного обмена информацией между администраторами и службой уборки относительно статуса номеров готовы к заселению, требуется уборка и т.д., что способствует повышению качества обслуживания.

Служба поддержки.

Осуществление обратной связи необходимо для того, чтобы пользователи могли связаться с администраций. Такой бизнес-процесс описывает функциональную часть системы бронирования.

В предметной области целесообразно выделить следующие ключевые компоненты:

– платформы бронирования, к которым относятся электронные системы и веб-сайты. Они собирают разнообразные данные о вариантах размещения гостей, предлагают инструменты для сравнения цен, условий проживания и доступности номеров. Подобные платформы, помимо этого аккумулируют пользовательские отзывы и рейтинги, что помогает потребителям принять обоснованный выбор;

– поставщики услуг размещения, к которым относятся гостиницы, отели. Они используют эти платформы для привлечения клиентов и управления бронированием;

– пользователями системы выступают клиенты, ищущие подходящее размещение. Данная категория объединяет как индивидуальных путешественников, семьи, так и организованные группы;

– технологическая инфраструктура необходима для обеспечения функционирование систем бронирования. В нее входят серверы, базы данных, интерфейсы для пользователей (веб-сайты и мобильные приложения) в

интеграции с другими системами, в том числе, с платежными шлюзами и системами управления отелями.

Указанная предметная область активно развивается благодаря внедрению новых технологий, которые нацелены на осуществление процесса бронирования более эффективным, безопасным и удобным для всех участников способом.

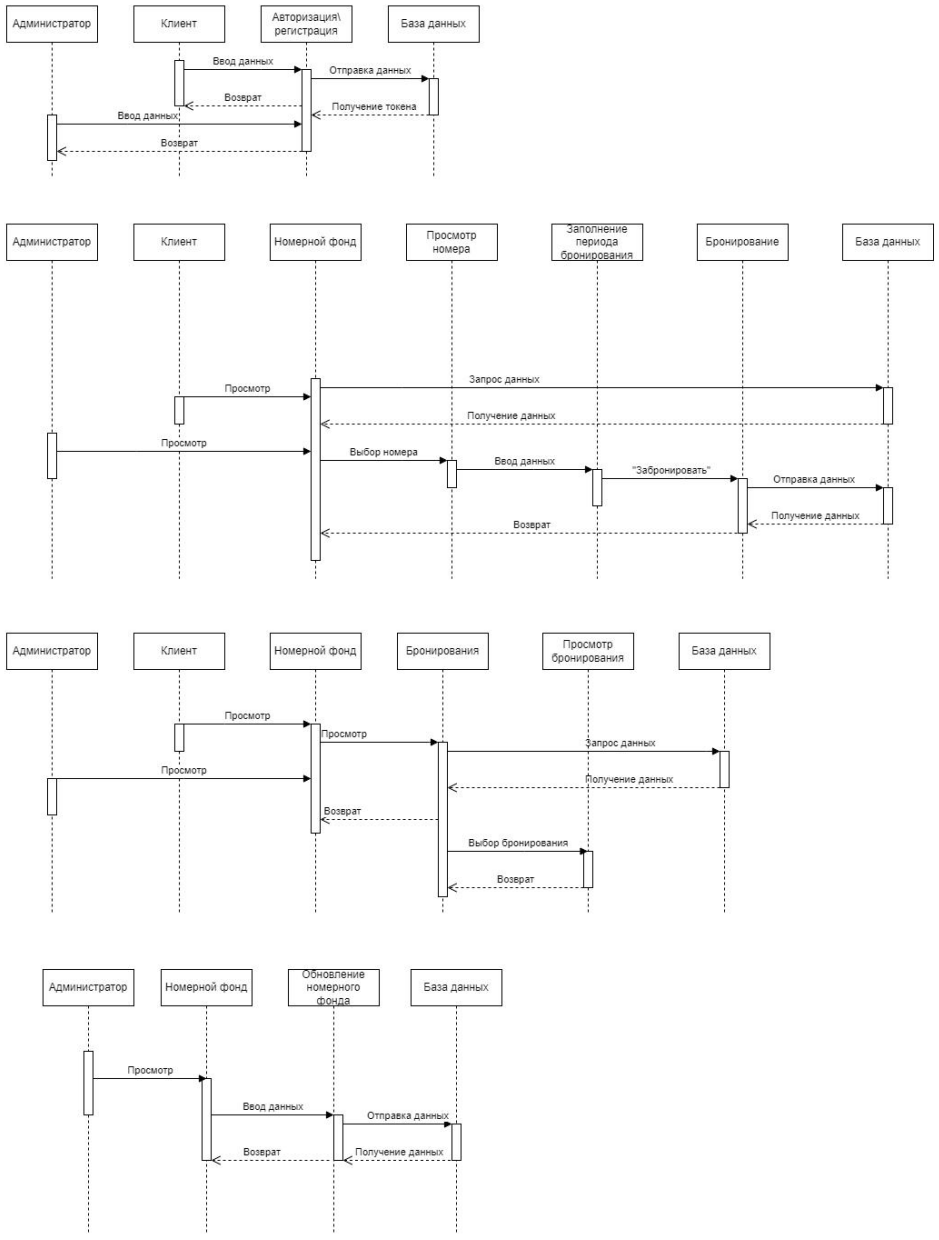


Рисунок 1 - UML-диаграмма бизнес процесса системы

1.2 Актуальность предметной области

Президент России Владимир Путин во время оглашения послания Федеральному Собранию 29 февраля 2024 г. подчеркнул, что к 2030 году турпоток в России должен вырасти в два раза и достигнет 140 млн человек в год.

Актуальность проблемы создания систем бронирования гостиниц связана и с тем, что подобные системы выполняют роль ключевого элемента для обеспечения эффективной работы гостиниц. Увеличение мобильности граждан, рост численности международных и внутренних путешествий способствуют росту спроса на удобные и эффективные инструменты для планирования поездок.

Системы бронирования дают возможность потребителям сравнивать различные варианты проживания, автоматизировать процесс бронирования, тем самым, съэкономить время и снизить затраты на путешествия.

Рассматриваемые системы бронирования помимо перечисленного выполняют важную роль в оптимизации управления ресурсами для отелей и других видов жилья. Они дают возможность гостиницам максимально нарастить загрузку жилого фонда, оптимизировать ценообразование и улучшить качество обслуживания, используя данные и аналитику для принятия обоснованных управленческих решений.

Следовательно, целесообразно сделать вывод о том, что развитие и улучшение систем бронирования гостиниц является важной задачей, обусловленной как текущими трендами потребления, так и потенциалом прогресса в области информационных технологий, управления и обработки данных.

1.3 Анализ аналогов

После аналитического обзора доступных систем бронирования, выполнен детальный анализ популярных платформ для резервирования гостиниц, что позволило выделить их ключевые преимущества и потенциальные недостатки:

1.3.2 Система «Hotellook»

Преимущества системы: быстрая загрузка и оперативная работа, наличие множества партнеров для сравнения цен (рисунок 2).

К основным недостаткам относится то, что переадресация на сайты партнеров может создать дополнительные действия для пользователей, что, безусловно, усложняет процесс бронирования.

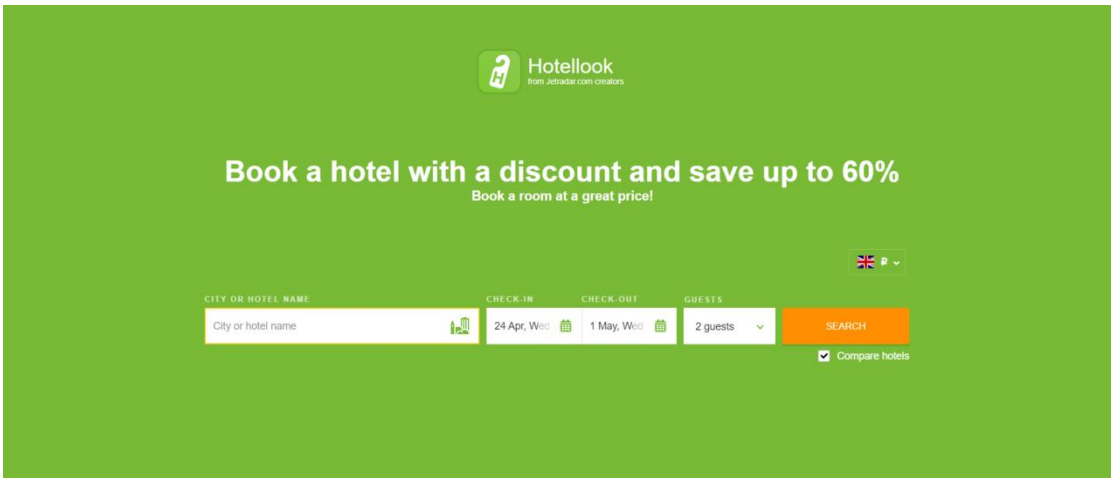


Рисунок 2 - Система «Hotellook»

1.3.2 Система «Trivago»

К преимуществам данной системы относится возможность осуществления быстрого поиска, быстрой загрузки страниц; точное сравнение цен, анализ предложений со скидками, отображение пользователю исключительно свободных номеров, что экономит время пользователя (рисунок 3).

Недостатки заключаются в следующем: недостаточно детальное описание предложений может оставить пользователей без существенной для

них информации о гостинице. Это создает трудность при принятии обоснованного решения без изучения дополнительных ресурсов.

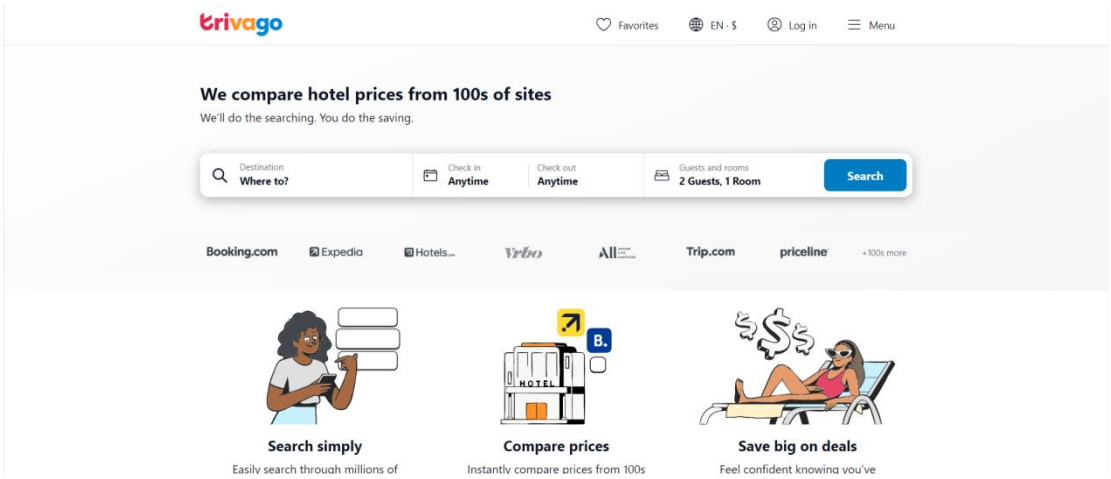


Рисунок 3 - Система «Trivago»

1.3.3 Система «Oktogo»

К преимуществам системы можно отнести следующее: сравнительно большой выбор, возможность расчета стоимости проживания при дополнительных условиях (рисунок 4).

Недостатки заключаются в следующем: не интуитивно понятный интерфейс может стать причиной возникновения барьера для новых пользователей, а это, в свою очередь, потребует временные затраты на адаптацию.

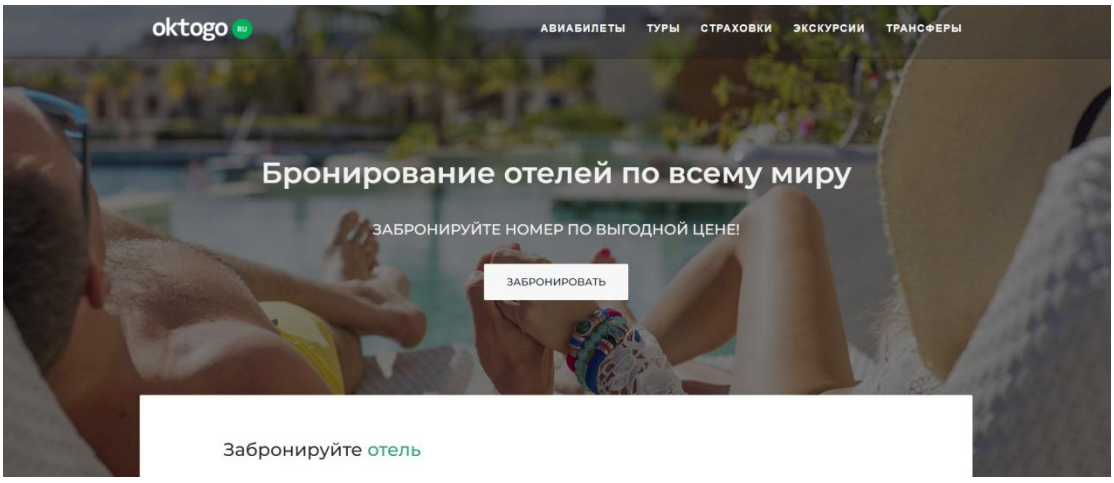


Рисунок 4 - Система «Oktogo»

1.3.4 Система «Ostrovok»

Преимущества данной системы: большое количество предложений жилья, возможность бронирования без предоплаты, контроль цен без завышения на популярные направления, персональные скидки, высокая безопасность онлайн-транзакций (рисунок 5).

Недостатки во многом аналогичны недостаткам системы Hotellook: переадресация на сайты партнеров для завершения бронирования может усложнить процесс. Вероятно, что ряд пользователей предпочли бы завершать все этапы бронирования в рамках одного сайта с целью упрощения процесса.

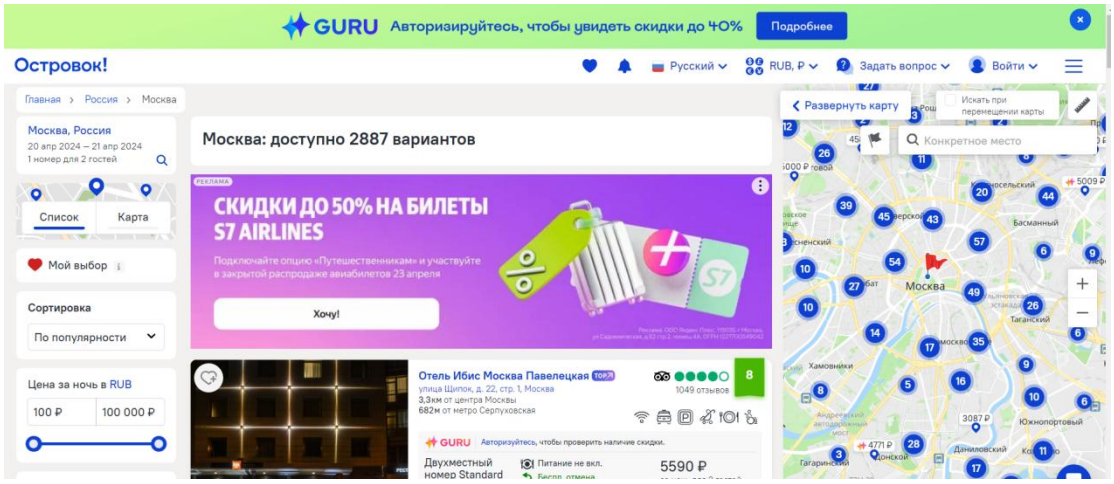


Рисунок 5 - Система «Ostrovok»

1.3.5 Вывод по анализу аналогов

В результате анализа популярных платформ бронирования гостиниц, таких как Hotellook, Trivago, Oktogo и Ostrovok, целесообразно сделать следующие выводы:

Сильные стороны рассмотренных платформ: все платформы обладают сильными сторонами, такими как быстрая загрузка и работа, точное сравнение цен и специализированные предложения, которые делают их вполне привлекательными для пользователей.

Ключевые недостатки рассмотренных систем: несмотря на ряд преимуществ, большинство платформ сталкивается с общими проблемами. К ним относятся, например, зависимость от переадресации на сайты партнеров и

не интуитивно понятные интерфейсы, что способно усложнить процесс бронирования для пользователей. Кроме того, недостаточно подробное описание отелей и услуг также может снизить удовлетворенность пользователей и повлиять на их решение о бронировании из-за невозможности бронирования без осуществления предоплаты.

Возможности улучшения систем: анализ существующих решений дает возможность идентифицировать ключевые области для улучшения в разрабатываемой системе бронирования. Устранение установленных недостатков и интеграция уникальных функций, в том числе, улучшенная пользовательская информация и упрощённый процесс бронирования, способны значительно повысить конкурентоспособность новой системы.

2 Бизнес-план

2.1 Целевая аудитория

К целевой аудитории создаваемой системы бронирования гостиничных номеров относятся:

– отели среднего и малого бизнеса, включая частные отели и небольшие гостиничные сети, которым необходимо улучшение процесса управления бронированиям и повышение уровня сервиса. Уровень посещаемости подобных отелей составляет примерно от 50 до 70% от общего числа гостей в регионе. Соотношение гостиниц со своими системами к гостиницам без своих систем бронирования составляет ориентировочно 30:70;

– хостелы и гостевые дома представляют собой небольшие заведения, которые хотят предложить своим клиентам удобный способ бронирования и повысить эффективность управления гостиничными номерами. Уровень посещаемости хостелов и гостевых домов составляет примерно 60 - 80% от общего числа гостей в регионе. Соотношение заведений со своими системами бронирования к заведениям, не имеющим собственных систем, составляет примерно 20:80.

Следовательно, ключевое внимание при разработке и продвижения системы бронирования необходимо уделять отелям среднего и малого бизнеса, хостелам и гостевым домам, поскольку они составляют значительную часть рынка и имеют высокий потенциал для внедрения современных технологий управления бронированиями.

2.2 План развития продукта

Исследование и анализ рынка:

– провести детальный анализ конкурентов на рынке, чтобы определить текущие тренды, потребности клиентов и потенциальные ниши (4 недели);

– изучить отзывы пользователей систем бронирования и выявить ключевые боли и пожелания для улучшения продукта.

Начальный запуск и маркетинг:

– разработать минимально жизнеспособный продукт (MVP) с основными функциями для тестирования на первых пользователях (12 недель);

– запустить рекламную кампанию в социальных сетях и через Google Ads для привлечения внимания к новому продукту;

– провести демонстрации продукта и вебинары для потенциальных клиентов, включая владельцев гостиниц и бизнес-путешественников.

Расширение функциональности:

– внедрить дополнительные функции, такие как интеграция с внешними платежными системами, улучшенные алгоритмы поиска и фильтрации номеров;

– разработать мобильное приложение для повышения удобства пользователей, предпочитающих бронирование через мобильные устройства (8 недель);

– обеспечить поддержку многоязычного интерфейса для привлечения международной аудитории.

Обратная связь и улучшения:

– активно собирать и анализировать отзывы пользователей, регулярно обновлять систему на основе этих данных (6 недель);

– тестировать новые функции и улучшения с группой пользователей для минимизации ошибок и недочетов.

Расширение на новые рынки:

– исследовать возможность выхода на международные рынки, начиная с ближайших зарубежных стран;

– заключать партнерские соглашения с туристическими агентствами и платформами для продвижения продукта.

Поддержка и обслуживание:

– обеспечить круглосуточную поддержку клиентов для оперативного решения возникающих вопросов и проблем;

– ввести в практику проведение регулярных обучающих мероприятий и вебинаров для администраторов отелей с целью максимального использования возможностей системы.

Монетизация заключается в разработке гибкой ценовой политики, обеспечивающей учет потребностей различных гостиничных заведений.

Политика постоянного улучшения и введения инноваций означает постоянное отслеживание новых технологических трендов и их интеграцию их в создаваемый программный продукт.

2.2 Финансовый план

Начальные инвестиции (на первые 6 месяцев) включают следующие расходы (рисунок 5):

– разработка электронной системы бронирования: 300 000 руб.;

– затраты на рекламу и маркетинг: 50 000 руб.;

– заработная плата сотрудников и разработчиков системы (с учетом начислений на зарплату): 200 000 руб.;

– затраты на содержание офиса и применяемого оборудования: 50 000 руб.;

– накладные расходы: 50 000 руб.

Итого: 650 000 рублей

– Операционные ежемесячные расходы на период следующих 6 месяцев(рисунок 5):

– заработная плата сотрудников (с учетом начислений на зарплату): 30 000 руб.;

– поддержка работы серверов и хостинг: 10 000 руб.;

– реклама и маркетинг: 10 000 руб.;

– затраты на обслуживание и поддержку клиентов: 5 000 руб.;

– иные операционные расходы: 5 000 руб.

Итого: 60 000 руб. в месяц.

Прогнозируемые ежемесячные доходы (рисунок 5):

– лицензия на использование системы (из расчета по 10 000 руб. на одну гостиницу, всего - 10 гостиниц): 100 000 руб.;

– подписка для гостиниц (10 гостиниц по 5 000 рублей): 50 000 руб.;

– дополнительные платные услуги / модули (5 гостиниц по 2 000 руб.): 10 000 руб.

Итого: 160 000 руб. в месяц.

Планирование роста прибыли на 1 год (рисунок 5):

Чистая прибыль за год рассчитывается следующим образом: 600 000 руб. (прибыль за первые 6 месяцев) плюс 600 000 руб. (прибыль за следующие 6 месяцев) минус 650 000 руб. (начальные инвестиции). В итоге получаем 550 000 руб.

Таким образом, получаем чистую прибыль за первый год 550 000 руб.

Количество гостиниц: 10.

Прогнозируемый детальный финансовый план представлен на графике (рисунок 5).

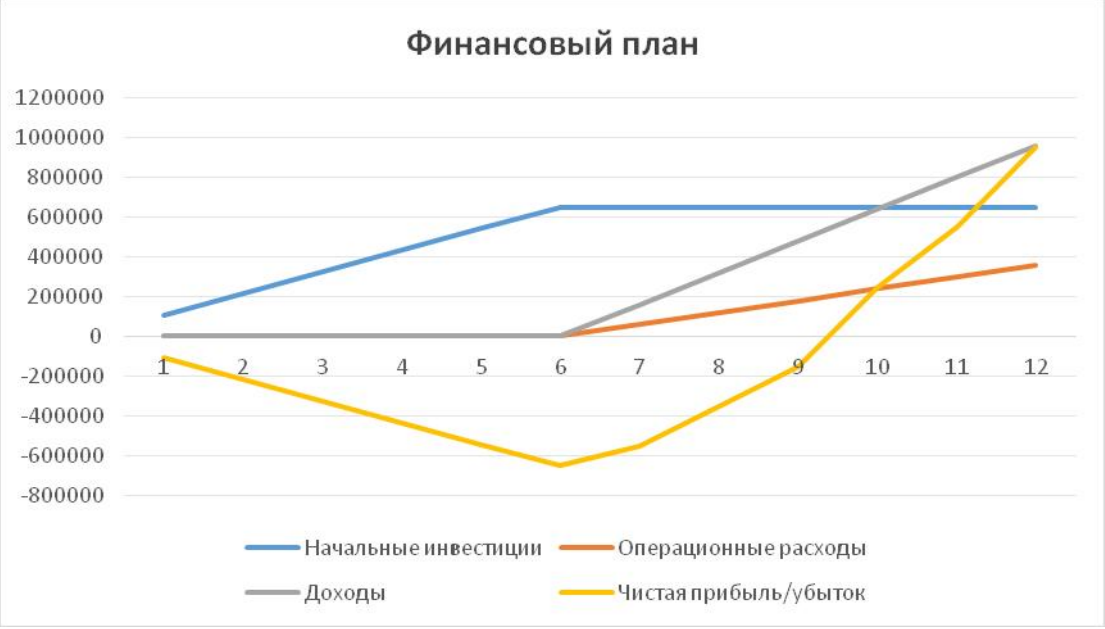


Рисунок 5 - Финансовый план

Предложенная в данном бизнес-плане система бронирования гостиничных номеров, характеризуется высоким потенциалом, что подтверждает целесообразность создания проекта для успешного развёртывания и развития на рынке туристических услуг. Предложенная система учитывает потребности ключевой аудитории, обеспечивает эффективную доходность, включает план развития и поддержки продукта на рынке. Это, в свою очередь, позволяет гибко адаптироваться к различным изменениям и новшествам в отрасли.

3 Постановка задачи

3.1 Требования к системе

3.1.1 Технические требования к системе

Клиентская часть должна быть доступна через десктопные браузеры, чтобы обеспечить стабильную работу в последних версиях браузеров, таких как Chrome, Firefox, Safari и Edge. Необходимо обеспечить совместимость с различными операционными системами, используемыми на компьютерах (Windows, Linux). Применяемый подход должен обеспечивать доступность и функциональность сервиса для пользователей независимо от используемых операционных систем.

Серверная часть включает размещение серверов в центрах обработки данных, расположенных на некотором расстоянии от клиентов. Это обеспечивает доступность и надежность системы. Важно убедиться, что система работает на стабильных версиях Linux, таких как Ubuntu или Windows 10. Эти операционные системы выбраны из-за их надежности, безопасности, а также всесторонней поддержки необходимых сетевых протоколов, приложений и др.

3.1.2 Функциональные требования к системе

Основными участники системы являются следующие субъекты:

– Клиенты (физические или юридические лица) могут просматривать объявления и бронировать номера;

– Администрация отеля управляет номерным фондом и бронированием;

– Системные администраторы отвечают за настройку и техническую поддержку системы бронирования.

Онлайн – бронирование означает возможность клиентов осуществлять бронирование через интернет.

Управление бронированиям означает возможность для администрации отеля осуществлять управление текущим и будущим бронированием.

Функция интеграции с платёжными системами означает в перспективе возможность добавления различных современных и актуальных платежных сервисов.

Управление номерным фондом включает в себя возможности по добавлению, удалению и изменению информации о номерах.

Перечисленные выше возможности системы отображены на UML-диаграмме вариантов использования (рисунок 6)

Разработаная система будет состоять из ряда ключевых компонентов, включающих базу данных, сервер и веб-клиент.

3.1.3 База данных PostgreSQL

Для хранения данных в системе выбрана база данных PostgreSQL. Этот выбор обоснован надёжностью системы, её высокой производительностью и обширной распространённостью, что важно для обработки бронирований и управления данными гостиниц.

3.1.4 Сервер на Spring Framework

Серверная часть системы будет разрабатывается с использованием Spring Framework, который является популярным фреймворком для создания высокопроизводительных и безопасных приложений на языке Java.

Spring Framework предоставляет множество возможностей для создания модульных и легко тестируемых приложений. Он поддерживает подключение к различным базам данных и упрощает реализацию таких функций, как например, авторизация и аутентификация пользователей. Разработка серверной части будет производиться в среде разработки IntelliJ IDEA, одной из ведущих сред для работы с Java и Spring, которая предоставляет все инструменты для написания кода, тестирования и отладки [1].

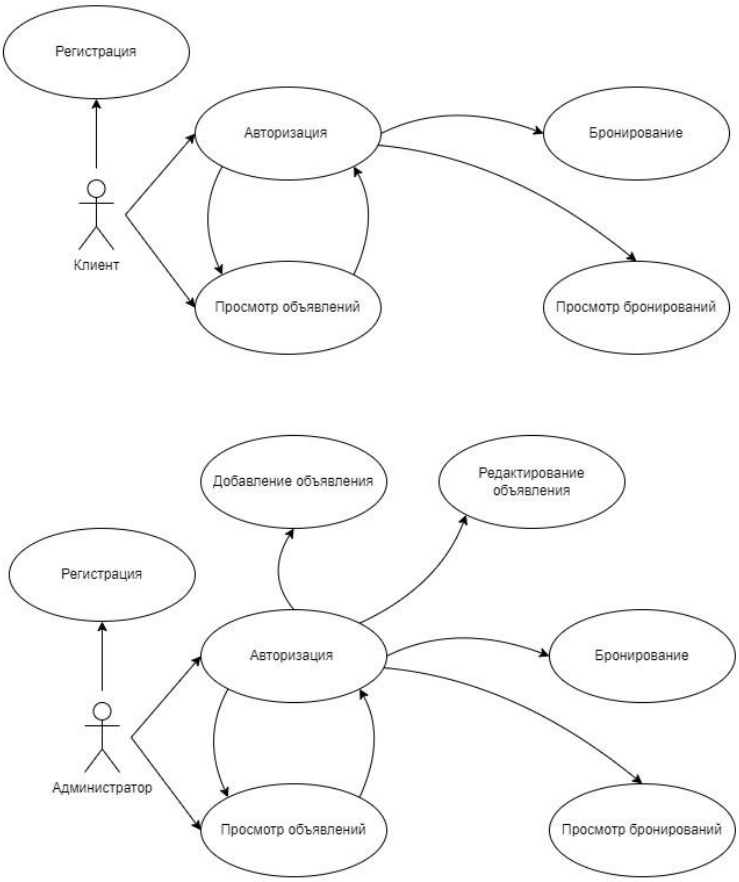


Рисунок 6 - UML-диаграмма вариантов использования

3.1.6 Веб-клиент на React

Пользовательский интерфейс системы будет разрабатывается с использованием React [2]. Он позволяет разрабатывать интерактивные, быстрые и адаптивные веб-приложения. Эта библиотека идеально подходит для создания сложных пользовательских веб - интерфейсов с динамичными элементами, что требуется для удобства пользователей. Разработка веб-клиента будет осуществляться в среде WebStorm, специализированной для языка JavaScript, которая предоставляет удобные инструменты для работы с React и другими современными технологиями веб-разработки [2].

3.1.7 Безопасность

При разработке системы особое внимание уделяется безопасности, авторизация пользователей в системе будет осуществляться с помощью

механизмов, предоставляемых Spring Security. Что обеспечит надежную защиту пользовательских данных и предотвратит доступ к системе со стороны недоброжелателей.

Такой подход к выбору технологий и инструментов разработки даст возможность создать надежную, масштабируемую и безопасную систему бронирования, что будет удовлетворять современным требованиям к веб - приложениям.

3.2 Архитектура приложения

При разработке архитектуры приложения для системы бронирования гостиничных номеров необходимо обеспечить ее надежность, эффективность и масштабируемость. В данной работе приложение будет разделено на три основных компонента:

– базу данных;

– сервер приложений;

– веб-клиент.

Такая структура позволяет четко разграничить функции каждого компонента и упростить их взаимосвязи (рисунки 7, 8).

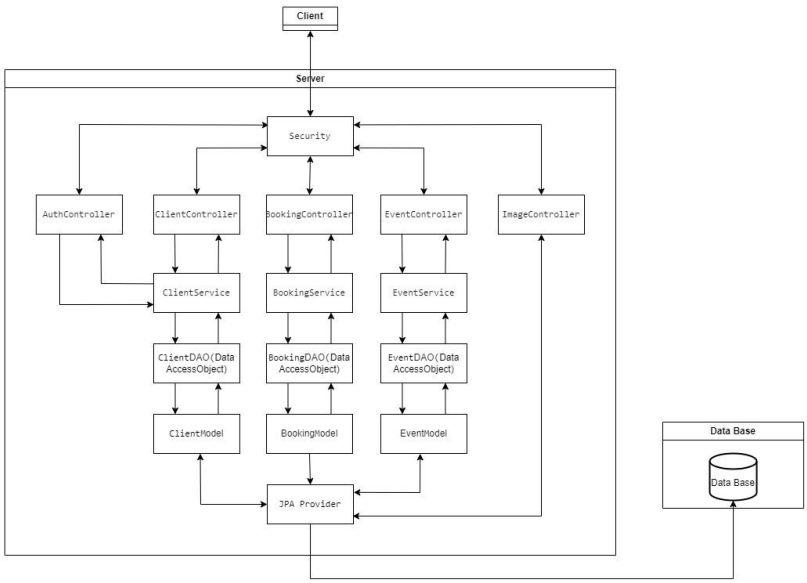


Рисунок 7 – Архитектура приложения

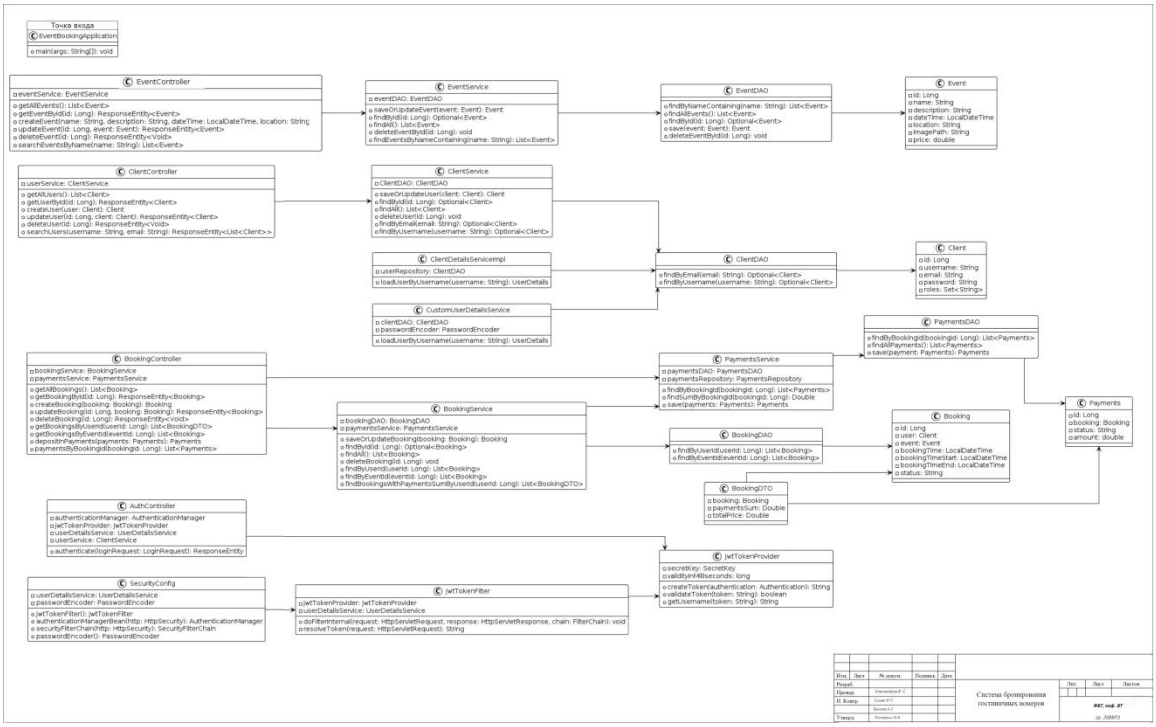


Рисунок 8 – UML-диаграмма классов

3.2.1 База данных (PostgreSQL)

PostgreSQL выбран в данной работе в качестве системы управления базой данных благодаря его ряду преимуществ, таким как, мощность, надежность и поддержка сложных запросов и транзакций, что представляется важным для системы бронирования. База данных (БД) предназначена для хранения всех данных, связанных с пользователями, бронированием, номерным фондом отелей, ценами и другими операционными параметрами. Базу данных необходимо оптимизировать для оперативного выполнения запросов и обеспечения высокого уровня транзакционной целостности.

Схема базы данных, которая представлена в виде ER-диаграммы (Entity-Relationship Diagram) (см. рисунок 9), иллюстрирует структуру базы данных и отношения между таблицами в системе бронирования гостиничных номеров. Представленная ER-диаграмма визуализирует данные и их связи, что важно на этапе проектирования баз данных.

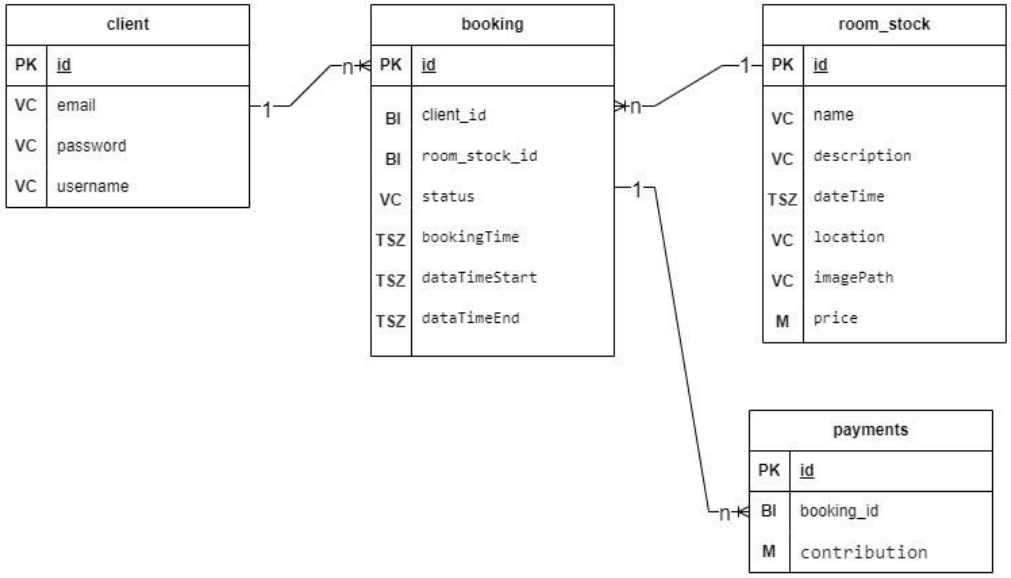


Рисунок 9 - ER-диаграмма базы данных (сущность — связь)

На диаграмме представлены четыре основные таблицы (блока):

– блок «client» содержит информацию о клиентах системы бронирования;

– блок «booking» отвечает за записи бронирования, устанавливает связи между клиентом и номерами, которые он бронирует;

– блок «room_stock» выполняет хранение информации о номерах, доступных к бронированию;

– блок «payments» отвечает за платежи, выполненные клиентами и связанные с конкретным бронированием.

Каждой таблице присущ первичный ключ (PK), обозначающий уникальный идентификатор записей в таблице. В таблице «booking» имеются внешние ключи (BI) client_id и room_stock_id, которые указывают соответственно на связь с таблицами client и room_stock. Таблица «payments» имеет внешний ключ booking_id, связывающий платежи с конкретным бронированием.

Отношения между таблицами обозначены линиями и кардинальностью:

– 1:n (один ко многим) между client и booking обозначает, что один клиент может осуществить несколько бронирований;

– 1:n между room_stock и booking обозначает, что один номер может быть забронирован более одного раза в разное время;

– 1:n между booking и payments свидетельствует о том, что одно бронирование может быть оплачено несколькими платежами.

Подобная схема представляется типичной для реляционных баз данных, используемых в системах бронирования для управления данными о клиентах, бронированиях и транзакциях.

Ниже рассмотрены основные нормальные формы:

Первая нормальная форма (1NF) означает, что в базе должны быть только атомарные значения, а каждое поле таблицы должны содержать уникальные значения. В данной схеме поля являются атомарными, то есть отсутствуют сгруппированные или повторяющиеся данные внутри одного поля, кроме того, каждая таблица имеет первичный ключ.

Второй нормальной форме (2NF) требуется выполнения всех условий 1NF, включая то, что каждый атрибут, который не является частью ключа, должен зависеть от всего первичного ключа, а не от его частей. Так как первичные ключи на диаграмме не составные, а внешние ключи используются для обеспечения связей, что соответствуют 2NF.

Третья нормальная форма (3NF) требует выполнение условия 2NF. То есть, все атрибуты, которые не являются ключами, должны быть взаимно независимыми от не ключевых атрибутов. На диаграмме нет транзитивных зависимостей, следовательно, схема также соответствует 3NF.

С учетом вышеизложенного, можно сделать вывод о том, что схема соответствует как минимум, первым трём нормальным формам (1NF, 2NF и 3NF).

3.2.2 Сервер приложений (Spring Framework)

Серверная часть приложения проектируется на базе Spring Framework, который поддерживает полный стек разработки веб-приложений и архитектуры [1]:

– spring Boot используется для упрощения настройки и развертывания приложений;

– spring MVC обеспечивает обработку запросов к серверу;

– spring Security необходим для аутентификации и авторизации пользователей;

– spring Data JPA служит для осуществления взаимодействия с базой данных посредством Java Persistence API.

Сервер должен обрабатывать все запросы от веб - клиента, осуществлять бизнес-логику (то есть, проверку доступности номеров, расчет цен, управление бронированием и т.д.), взаимодействовать с базой данных.

3.2.3 Описание API системы

Каждый контроллер выполняет функции управления различными данными и процессами: аутентификацией, бронированиям, клиентами, номерным фондом и изображениями (рисунок 8). Ниже приводится описание API системы:

Описание API AuthController Endpoint: /auth.

POST / login: Выполнение аутентификации пользователей посредством проверки их имени пользователя и пароля. В случае успешной аутентификации возвращается JWT (токен) и идентифицируется информация о пользователе.

Описание API BookingController Endpoint: /api/bookings:

– GET /: осуществляет возврат брони;

– GET /{id}: осуществляет возврат брони по указанному ID пользователя;

– POST /: создание нового бронирования;

– PUT /{id}: обновление существующей брони по ID пользователя;

– DELETE /{id}: удаление брони по ID;

– GET /user/{userId}: возврат брони для коекретного пользователя;

– GET /event/{eventId}: возврат брони рассматриваемого ID пользователя.

Описание API ClientController Endpoint: /api/clients:

– GET /: возврат всех клиентов;

– GET /{id}: возврат клиента по ID пользователя;

– POST /: создание нового клиента;

– PUT /{id}: обновление существующего клиента;

– DELETE /{id}: удаление клиента по ID пользователя;

– GET /search: выполнение поиска клиентов по имени пользователя или ID пользователя или по электронной почте;

Описание API EventController Endpoint: /api/events:

– GET /get_all: возврат всех номеров;

– GET /{id}: возврат мероприятия по ID;

– POST /create: создание нового мероприятия с возможностью добавить изображения;

– PUT /{id}: обновление существующего мероприятия;

– DELETE /{id}: удаление мероприятия;

– GET /search: выполнение поиска мероприятий по названию.

Описание API ImageController Endpoint: /api/images:

GET /{imageName}: возврат изображения по имени файла. Ответ должен содержать тип содержимого, а также изображение.

3.2.4 Веб-клиент (React)

Фронтенд представляет собой часть системы, которая реализуется на основе использования React - актуальной библиотеки, предназначенной для создания пользовательских интерфейсов. React позволяет разрабатывать

интерактивные UI с требуемой производительностью и следующими преимуществами [2]:

– использование компонентного подхода дает возможность легко создавать и повторно использовать UI-элементы;

– использование однонаправленного потока данных дает возможность упростить управление состоянием приложений;

– виртуальный DOM осуществляет минимизацию количества операций с реальным DOM, ускоряет отклик интерфейса;

– клиент выполняет обмен данными с сервером через REST API, получает данные в формате JSON, обновление интерфейса пользователя осуществляется в соответствии с полученной информацией.

С целью обеспечения безопасности и соблюдения целостности данных, между клиентом и сервером применены HTTPS и JSON Web Tokens (JWT), которые гарантируют безопасный обмен данными, а также аутентификацию пользователей.

Система содержит механизмы защиты от распространенных угроз, в том числе, SQL инъекций, XSS и CSRF атак. Следовательно, предложенная архитектура системы бронирования гостиничных номеров обеспечивает разделение ответственности между компонентами, что способствует упрощению разработки, тестирования и поддержки системы. Применение современных, проверенных технологий гарантирует высокую производительность, безопасность, превосходное пользовательское взаимодействие.

4 Разработка системы

4.1 Настройка среды разработки

4.1.1 Настройка среды IntelliJ IDEA и Spring Boot

Чтобы создать новый проект в IntelliJ IDEA на основе фреймворка Spring Boot [1], необходимо выполнить следующие действия (рисунок 10):

– запустите IntelliJ IDEA и выберите "Создать новый проект" на главном экране;

– в открывшемся окне выберите "Spring Initializr", чтобы создать проект Spring Boot (рисунок 10).

– Реализация параметров проекта (иллюстрирует Рисунок 11) подразумевает выполнение следующих шагов:

– ввод названия проекта и выбор моста сохранения;

– указание типа проекта Maven и языка программирования Java [1];

– выбор версии Spring Boot и других параметров проекта (рисунок 10).

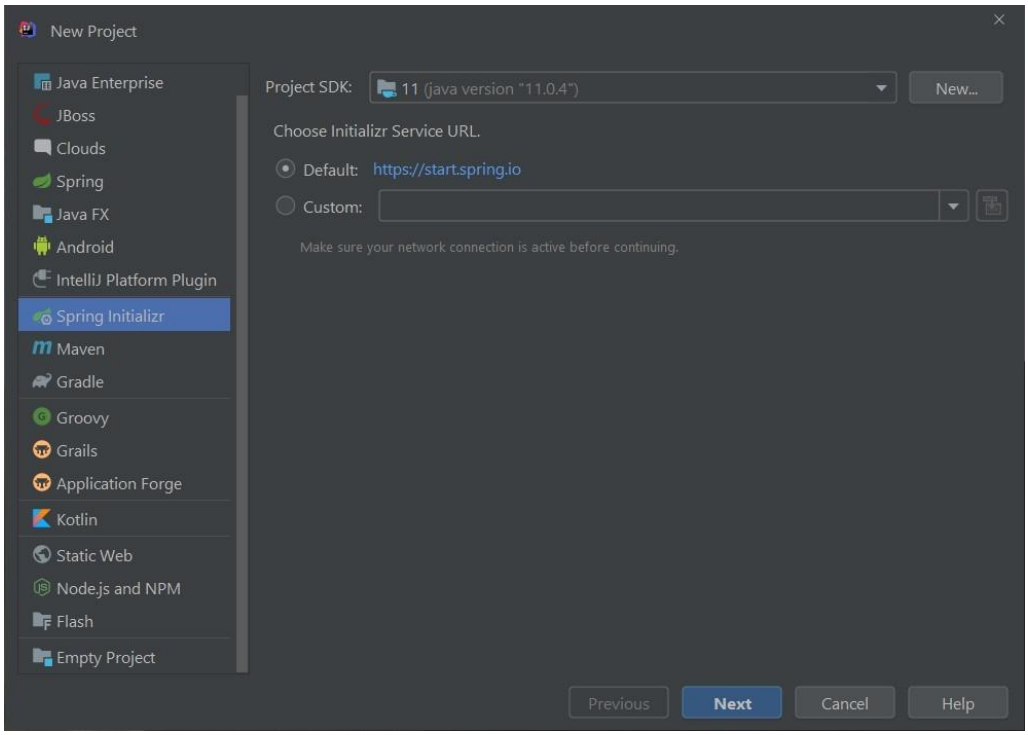


Рисунок 10 – Осуществление выбора типа проекта Spring Initializr

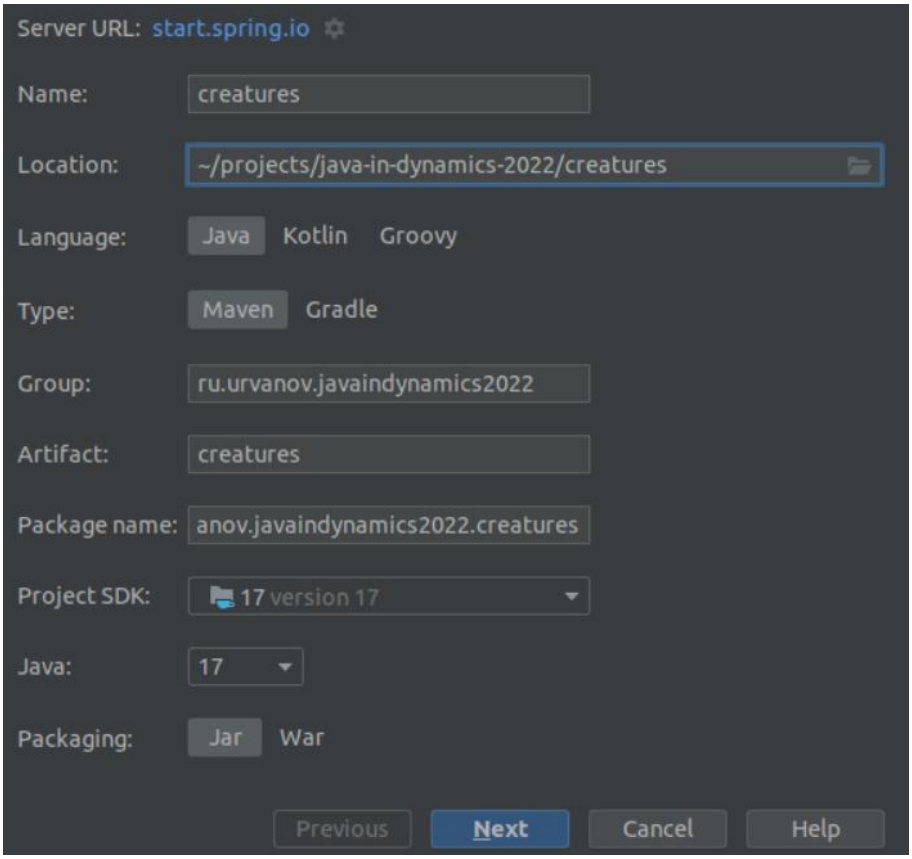


Рисунок 11 – Осуществление настроек нового проекта

После создания проекта в левой части окна IntelliJ IDEA будет отображено дерево проекта, в котором находятся все основные модули и файлы (рисунок 12).

Дерево файлов проекта включает в себя ряд модулей.

src/main/java: в этой папке содержатся исходные коды приложения, в том числе, классы и конфигурации;

src/main/resources: в данной папке расположены ресурсы приложения, в том числе, файлы конфигураций (application.yml) и дополнительные ресурсы;

src/test/java: папка предназначена для тестовых классов с целью выполнения тестирования приложения;

Maven files: это файлы конфигурации сборки проекта (pom.xml для Maven), в данной папке описаны зависимости проекта и плагины;

.mvn: папки содержат конфигурации сборочных инструментов Maven.

Перечисленные шаги необходимы для выполнения разработки приложений на Spring Boot с помощью IntelliJ IDEA.

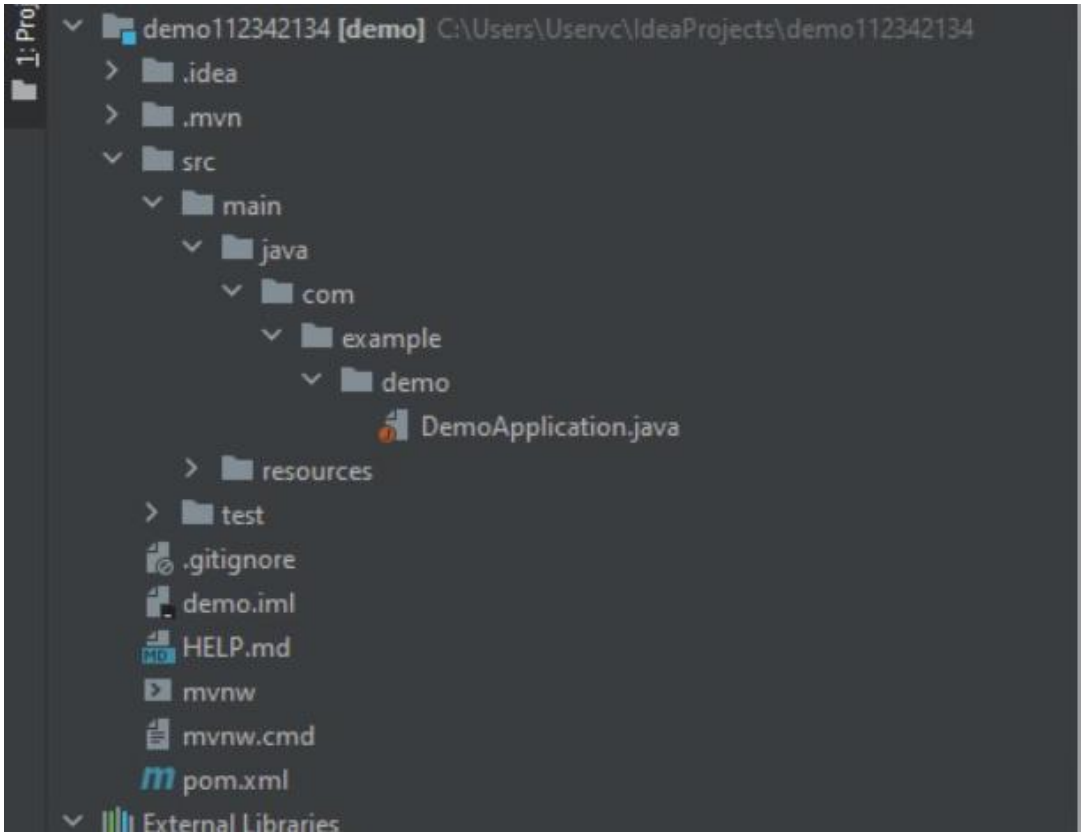


Рисунок 12 – Дерево файлов проекта

4.1.2 Настройка среды WebStorm и React

С целью создания проекта на React посредством среды разработки WebStorm, необходимо придерживаться следующей последовательности действий:

– Выполнить открытие терминала в необходимой директории;

– Установить Create React App посредством команды npm install -g create-react-app [2];

– Создать новый проект с помощью указанной команды React npx create-react-app my-react-app (рисунок 13), где my-react-app обозначает желаемое название проекта [2]. Далее необходимо ожидание, когда скрипт создаст все необходимые файлы проекта и установит зависимости.

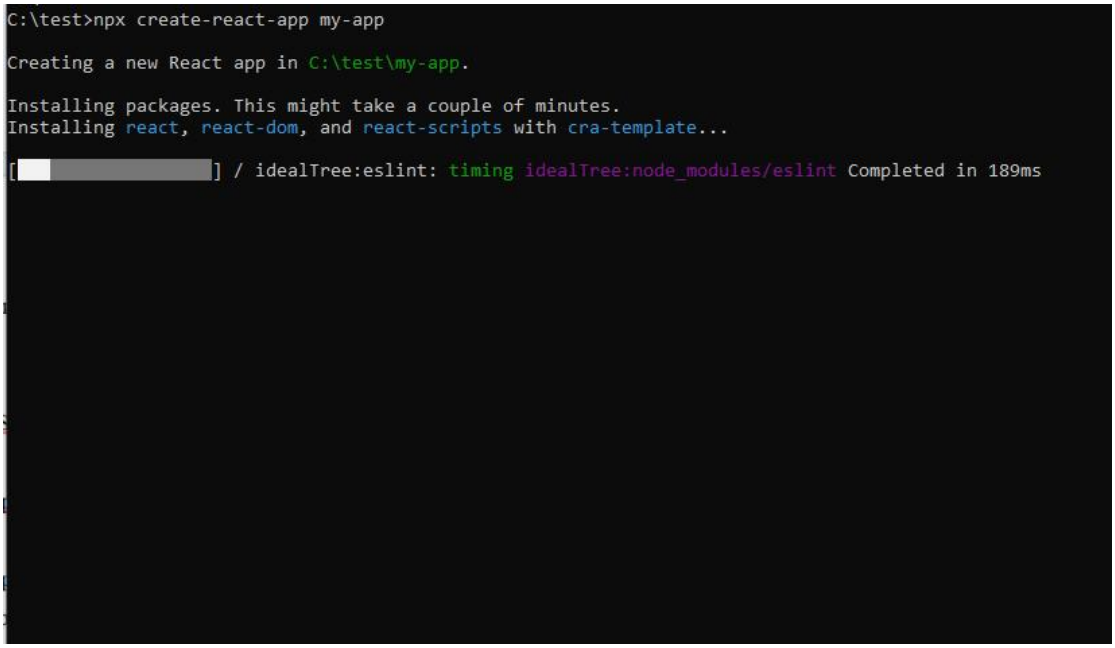


Рисунок 13 - Создание нового проекта React

По окончании создания нового проекта, следует открыть проект в WebStorm через File > Open... и выбрать папку проекта; запустить проект посредством команды **npm start** (рисунок 14), что автоматически по умолчанию откроет приложение React в браузере.

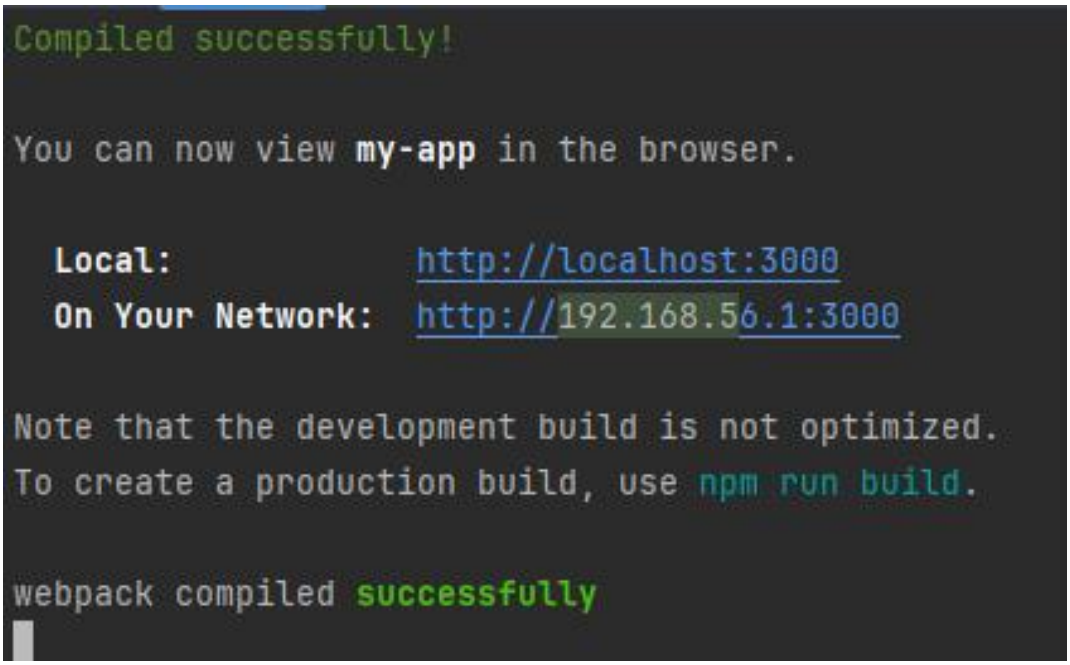


Рисунок 14 - Запуск проекта

4.1.3 Выполнение настройки среды PgAdmin

PgAdmin представляет собой современный инструмент для управления базами данных PostgreSQL, которому присущ удобный графический интерфейс для выполнения различных операций с базами данных [3].

С целью создания новой базы данных в PostgreSQL на основе графического интерфейса pgAdmin, необходимо осуществить следующую последовательность действий [3]:

– открыть pgAdmin, после запуска будет отображено главное окно pgAdmin;

– кликнуть на узле "Databases" (базы данных) правой кнопкой мыши и выбрать из контекстного меню "Create" > "Database...";

– ввести в поле "Database" название новой базы данных в открывшемся окне "Create Database" (рисунок 15).

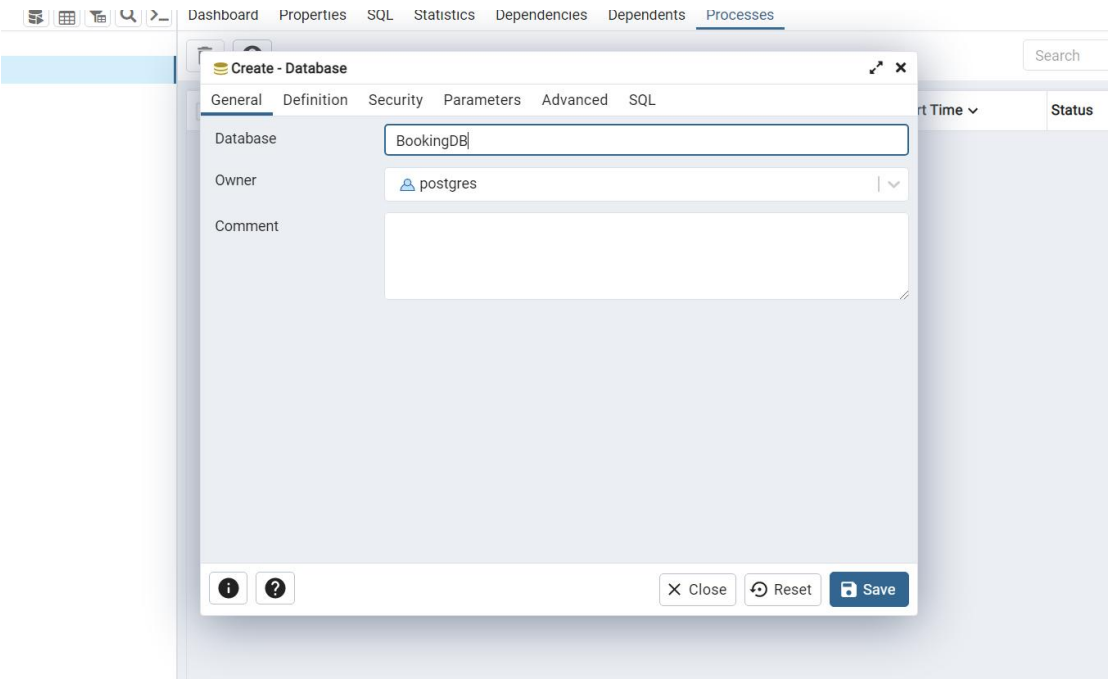


Рисунок 15 - Создание базы данных

После ввода всех данных и настроек нажмём кнопку "Save" (Сохранить), чтобы создать базу данных (рисунок 15).

Новая база данных теперь отображаться в списке баз данных (рисунок 16).

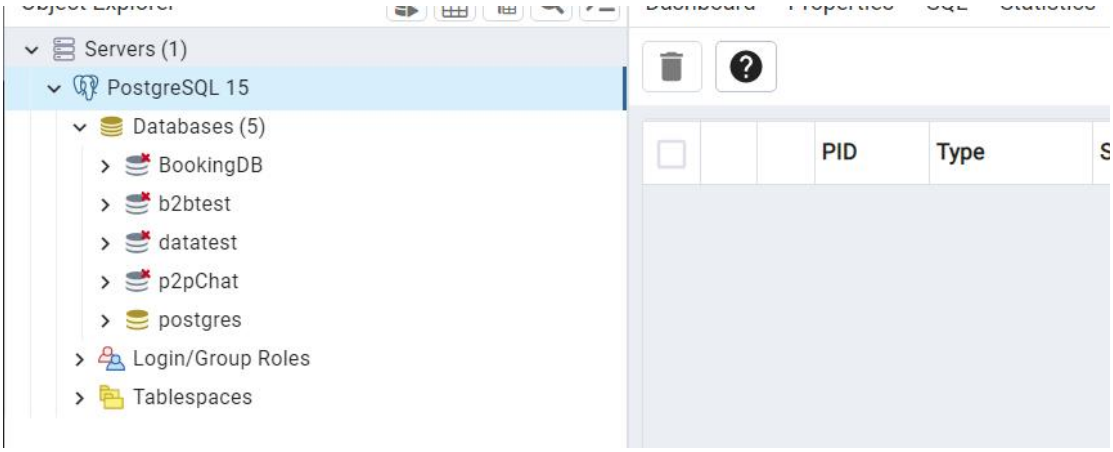


Рисунок 16 - Отображение созданной базы данных в списке

Далее необходимо кликнуть на новую базу данных в pgAdmin для её выбора, после чего можно вводить данные, выполнять запросы и использовать другие действия базой данных (рисунок 17).

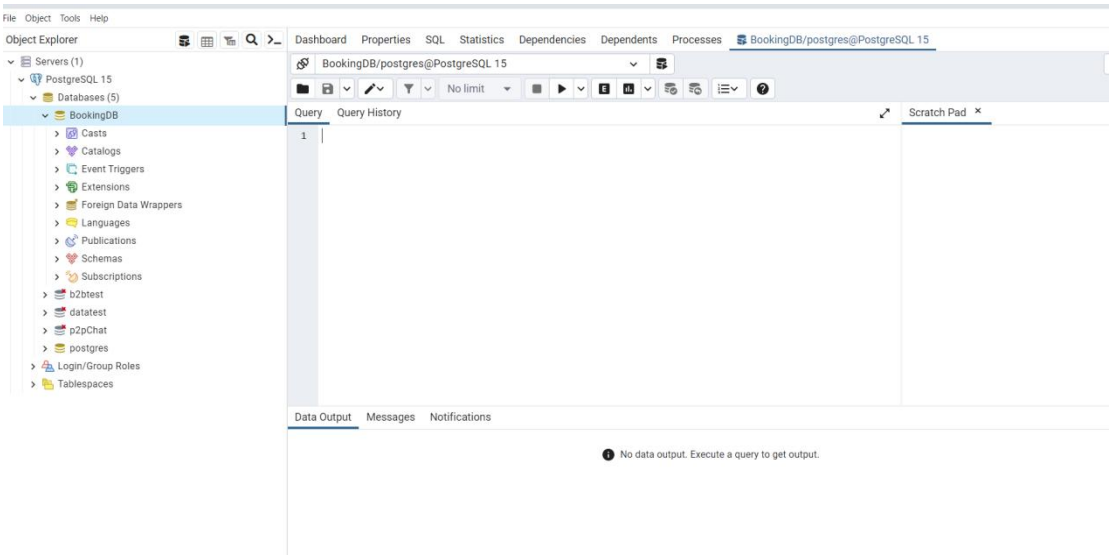


Рисунок 17 - Среда для выполнения запросов к базе данных

4.2 Разработка сервера на Spring Boot для системы управления бронированиями

Серверная часть системы бронированиями реализована с использованием фреймворка Spring Boot, который обеспечивает удобную разработку веб-приложений и REST-сервисов[1]. Основные компоненты сервера включают контроллеры, сервисы, репозитории, модели и конфигурации безопасности, каждый из которых выполняет различные функции.

При разработке системы было принято решение использовать архитектурный паттерн Model-View-Controller (MVC), который является одним из наиболее распространённых подходов для описания логики сервера на управляемые и масштабируемые части. Это решение позволило разграничить данные, пользовательский интерфейс и управляющую логику, обеспечивая тем самым получается чистая более архитектура и универсальность проекта для расширения.

4.2.1 Реализация сервера

Контроллеры являются частью паттерна MVC и обрабатывают различные входящие HTTP-запросы, управляют пользовательскими данными и возвращают раличные данные клиенту.

AuthController - отвечает за аутентификацию пользователей. Метод authenticate принимает логин и пароль, проводит аутентификацию и возвращает JWT токен[1], который используется для доступа к защищенным ресурсам.

Пример части реализации класса AuthController:

```
@RestController
@RequestMapping("/auth")
public class AuthController {
    @Autowired
```

```
        private AuthenticationManager authenticationManager;
        @Autowired
        private JwtTokenProvider jwtTokenProvider;

        @PostMapping("/login")
        public ResponseEntity<?> authenticate(@RequestBody
LoginRequest loginRequest) {
            try {
                Authentication authentication =
authenticationManager.authenticate(
                        new UsernamePasswordAuthenticationToken(
                                loginRequest.getUsername(),
loginRequest.getPassword()
                        )
                );
                String jwt =
jwtTokenProvider.createToken(authentication);
                return ResponseEntity.ok(new JwtResponse(jwt));
            } catch (AuthenticationException e) {
                return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
            }
        }
    }
```

BookingController - управляет бронированиями. Включает методы для получения всех бронирований, создания, изменения и удаления бронирования, а также поиска бронирований по ID пользователя и номеров.

Пример части реализации класса BookingController:

```
@RestController
@RequestMapping("/api/bookings")
public class BookingController {
    @Autowired
    private BookingService bookingService;
```

```
        @GetMapping
        public List<Booking> getAllBookings() {
            return bookingService.findAll();
        }

        @PostMapping
        public ResponseEntity<Booking> createBooking(@RequestBody
Booking booking) {
            Booking                  savedBooking                  =
bookingService.saveOrUpdateBooking(booking);
            return ResponseEntity.ok(savedBooking);
        }

        @PutMapping("/{id}")
        public                                ResponseEntity<Booking>
updateBooking(@PathVariable Long id, @RequestBody Booking booking)
{
            return bookingService.findById(id)
                .map(existingBooking -> {
                    booking.setId(id);
                    bookingService.saveOrUpdateBooking(booking);
                    return ResponseEntity.ok(booking);
                })
                .orElseGet(()                                        ->
ResponseEntity.notFound().build());
        }

        @DeleteMapping("/{id}")
        public  ResponseEntity<Void>  deleteBooking(@PathVariable
Long id) {
            return bookingService.findById(id)
                .map(booking -> {
                    bookingService.deleteBooking(id);
                    return ResponseEntity.ok().<Void>build();
```

```
                })
                .orElseGet(() ->
ResponseEntity.notFound().build());
        }
    }
```

ClientController - предназначен для работы с клиентами. Методы позволяют создавать, обновлять, удалять и получать информацию о пользователях. Также поддерживается поиск пользователей по имени и email.

Пример части реализации класса ClientController:

```
    @RestController
    @RequestMapping("/api/clients")
    public class ClientController {
        @Autowired
        private ClientService clientService;

        @GetMapping
        public List<Client> getAllClients() {
            return clientService.findAll();
        }

        @PostMapping
        public ResponseEntity<Client> createClient(@RequestBody
Client client) {
            Client savedClient =
clientService.saveOrUpdateUser(client);
            return ResponseEntity.ok(savedClient);
        }

        @PutMapping("/{id}")
        public ResponseEntity<Client> updateClient(@PathVariable
Long id, @RequestBody Client client) {
            return clientService.findById(id)
                .map(existingClient -> {
```

```
                client.setId(id);
                clientService.saveOrUpdateUser(client);
                return ResponseEntity.ok(client);
            })
            .orElseGet(()                                    ->
ResponseEntity.notFound().build());
        }
    }
```

EventController - контролирует действия, связанные с номерным фондом. Осуществляет добавление, обновление, удаление и получение мероприятий, а также загрузку изображений для номеров.

Пример части реализации класса EventController:

```
    @RestController
    @RequestMapping("/api/events")
    public class EventController {
        @Autowired
        private EventService eventService;

        @GetMapping
        public List<Event> getAllEvents() {
            return eventService.findAll();
        }

        @PostMapping("/create")
        public   ResponseEntity<Event>   createEvent(@RequestBody
Event event) {
            Event newEvent = eventService.saveOrUpdateEvent(event);
            return ResponseEntity.ok(newEvent);
        }
    }
```

ImageController - предоставляет доступ к изображениям, хранящимся на сервере, и поддерживает их получение по имени файла.

Сервисы в рамках данного пректа отвечают за бизнес-логику и взаимодействие с репозиториями для доступа к базе данных.

ClientService - содержит логику для управления данными клиентов, такую как создание, обновление и поиск клиентов.

BookingService - управляет операциями с бронированиями, включая сохранение, обновление и удаление броней.

EventService - занимается логикой вокруг мероприятий, включая создание, обновление и поиск мероприятий по различным критериям.

Репозитории содержат методы для запросов к базе данных и взаимодействия с моделями сущностей описанных на сервере. Репозитории в Spring в данном случае наследуются от JpaRepository, что предоставляет множество готовых методов для работы с данными.

Модели описывают структуры данных в данном случае модели содержат те же параметры что и база данных.

Далее приведены параметры реализованные в моделях сущностей базы данных:

Клиенты (clients):

– id: первичный ключ, BIGSERIAL;
– username: уникальное имя пользователя, VARCHAR(255);
– email: уникальная электронная почта, VARCHAR(255);
– password: хеш пароля, VARCHAR(255);
– roles: массив ролей пользователя, TEXT[];

Номерной фонд (events):

– id: первичный ключ, BIGSERIAL;
– name: название номера, VARCHAR(255);
– description: описание номера, TEXT;
– date_time: дата и время проведения, TIMESTAMP WITHOUT TIME ZONE;
– location: местоположение, VARCHAR(255);

– image_path: путь к изображениям номера, TEXT.

Бронирования (bookings):

– id: первичный ключ, BIGSERIAL;

– user_id: внешний ключ, ссылается на clients(id);

– event_id: внешний ключ, ссылается на events(id);

– booking_time: время создания бронирования, TIMESTAMP WITHOUT TIME ZONE;

– booking_time_start: начало времени бронирования, TIMESTAMP WITHOUT TIME ZONE;

– booking_time_end: окончание времени бронирования, TIMESTAMP WITHOUT TIME ZONE;

– status: статус бронирования, VARCHAR(50).

Данные примеры показывают, как разработанная система управляет аутентификацией, обработкой данных клиентов, мероприятий и бронирований. Spring Boot позволяет облегчить создание и поддержку таких систем, предоставляя различные инструменты для работы.

Конфигурация безопасности реализуется через Spring Security, обеспечивая аутентификацию и авторизацию на основе ролей. Включая настройку фильтров, управление доступом к API и к шифрования паролей [5].

4.2.2 Реализация аутентификации в системе на Spring Boot

Компоненты аутентификации:

– реализация UserDetailsService (ClientDetailsServiceImpl) загружает данные пользователя из базы данных с помощью класса ClientDAO, который осуществляет запросы к таблице clients;

– метод loadUserByUsername используется для поиска пользователя по его имени. Если пользователь найден, он возвращает объект UserDetails, содержащий имя пользователя, пароль и присвоенные ему роли.

Пример кода:

```
@Service
```

```
    public class ClientDetailsServiceImpl implements
UserDetailsService {
        @Autowired
        private ClientDAO userRepository;

        @Override
        public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
            Optional<Client> optionalClient =
userRepository.findByUsername(username);
            Client client = optionalClient.orElseThrow(() -> new
UsernameNotFoundException("User not found"));
            return new
org.springframework.security.core.userdetails.User(
                client.getUsername(),
                client.getPassword(),
                getAuthorities(client.getRoles())
            );
        }

        private Collection<? extends GrantedAuthority>
getAuthorities(Set<String> roles) {
            return
roles.stream().map(SimpleGrantedAuthority::new).collect(Collector
s.toList());
        }
    }
```

AuthenticationManager используется в AuthController для проверки учетных данных пользователя:

– при успешной аутентификации AuthenticationManager возвращает полностью аутентифицированный объект, содержащий имя пользователя и роли;

– JWT Token Generation and Validation;

– JwtTokenProvider создает JWT на основе имени пользователя и его ролей после успешной аутентификации. Токен включает в себя информацию о ролях пользователя, что позволяет реализовать механизмы авторизации на основе ролей;

– методы createToken и validateToken используются для создания и проверки токена соответственно.

Пример кода:

```
@Component
public class JwtTokenProvider {

    private final SecretKey secretKey = Keys.secretKeyFor(SignatureAlgorithm.HS256);

    public String createToken(Authentication authentication) {
        Claims claims = Jwts.claims().setSubject(authentication.getName());
        claims.put("roles", authentication.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority).collect(Collectors.toList()));

        Date now = new Date();
        Date validity = new Date(now.getTime() + validityInMilliseconds);

        return Jwts.builder()
            .setClaims(claims)
            .setIssuedAt(now)
            .setExpiration(validity)
            .signWith(secretKey, SignatureAlgorithm.HS256)
            .compact();
    }
```

```
        public boolean validateToken(String token) {

Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);
            return true;
        }
    }
```

Процесс аутентификации

– получение логина и пароля от пользователя;
– передача учетных данных в AuthenticationManager;
– поиск пользователя с помощью UserDetailsService;
– проверка пароля и генерация JWT;
– отправка JWT пользователю.

Аспекты безопасности:

– хранение паролей. Пароли хранятся в зашифрованном виде. В вашем текущем коде используется PasswordEncoder для шифрования паролей.
– управление токенами. JWT обеспечивает безопасную передачу учетных данных через HTTP, но требует корректной реализации механизмов обновления и отзыва токенов [5].

Данная реализация аутентификации на основе Spring Security и JWT является современным и безопасным решением для управления доступом пользователей к системе. Она обеспечивает не только безопасность, но и расширяемость для добавления дополнительных методов аутентификации и авторизации [7].

4.2.3 Реализация миграций в серверной части

Для обеспечения целостности данных и согласованности схемы базы данных в рамках разработки веб-приложения было принято решение использовать систему миграций. Миграции позволяют версионировать базу данных и обеспечивают возможность безопасного внесения изменений без

риска потери данных или нарушения функциональности существующего приложения.

Миграции баз данных организуются через последовательные скрипты, каждый из которых приводит базу данных из одного состояния в другое. Эти скрипты содержат команды SQL, которые изменяют структуру или содержимое базы данных (например, создание или изменение таблиц, добавление индексов, изменение типов данных колонок).

Приложения, основанные на следующих SQL-командах миграции обеспечивают создание начальной структуры базы данных.

```
CREATE TABLE IF NOT EXISTS clients (
    id BIGSERIAL PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    roles TEXT[] NOT NULL
);

CREATE TABLE IF NOT EXISTS events (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    date_time TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    location VARCHAR(255) NOT NULL,
    image_path TEXT
);

CREATE TABLE IF NOT EXISTS bookings (
    id BIGSERIAL PRIMARY KEY,
    user_id BIGINT REFERENCES clients(id),
    event_id BIGINT REFERENCES events(id),
    booking_time TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    booking_time_start TIMESTAMP WITHOUT TIME ZONE NOT NULL,
```

```
    booking_time_end TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    status VARCHAR(50) NOT NULL
);
```

Для управления миграциями использовался инструмент, Flyway, который интегрируются с Maven и позволяют автоматизировать процесс миграции при сборке или развертывании приложения [6].

Каждый скрипт миграции получает уникальный идентификатор версии, который помогает отслеживать историю изменений и обеспечивает возможность отката к предыдущим версиям схемы.

При интеграции с CI/CD пайплайнами миграции могут быть автоматически применены в процессе развертывания, что гарантирует, что база данных всегда синхронизирована с версией кода приложения.

Применение системы миграций в проекте обеспечивает контролируемые и обратимые изменения структуры базы данных, что является важным аспектом управления современными веб-приложениями. Это помогает поддерживать стабильность приложения и упрощает процессы разработки, тестирования и сопровождения.

4.3 Разработка клиентской части на React

В рамках проекта системы бронирования мероприятий было принято решение использовать React для разработки клиентской части. React предоставляет мощные инструменты для создания интерактивных пользовательских интерфейсов с использованием компонентного подхода. Это позволяет эффективно управлять состоянием приложения и повторно использовать код, облегчая тем самым разработку и последующее тестирование.

Клиентская часть системы включает в себя несколько основных компонентов и модальных окон, каждый из которых выполняет свои задачи:

4.3.1 Реализация окна авторизации

Компонент AuthModal обрабатывает вход в систему (рисунок 18). Пользователи вводят свои учетные данные, и если аутентификация успешна, в локальное хранилище сохраняется токен JWT.

При нажатии на кнопку "Войти" отправляется запрос на сервер. В случае успешной аутентификации пользователь переадресуется на основную страницу, в противном случае отображается сообщение об ошибке.

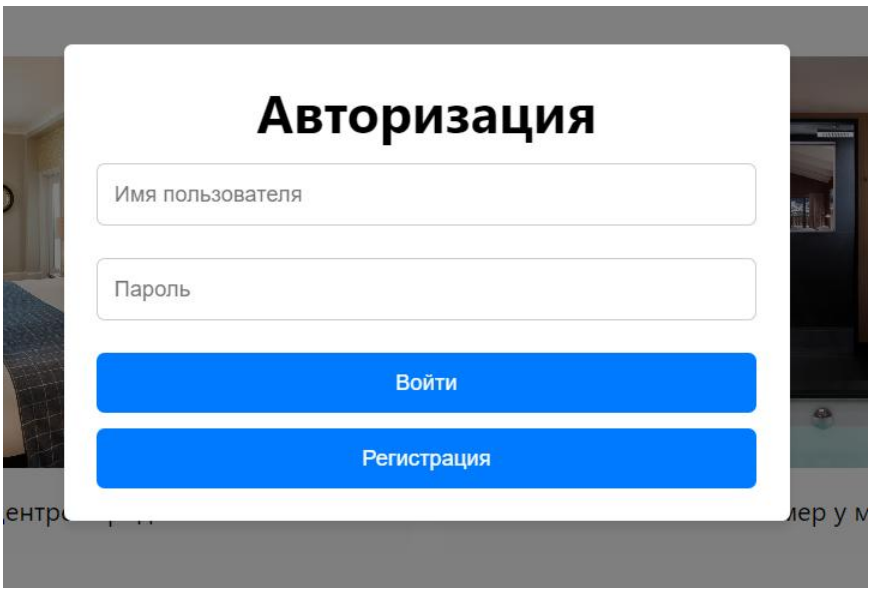


Рисунок 18 - Окно авторизации

Пример реализации:

```
import React, { useState } from 'react';
import './AuthModal.css';

const AuthModal = ({ onClose, onLogin }) => {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const [errorMessage, setErrorMessage] = useState('');

    const handleLogin = async () => {
        if (!username || !password) {
            setErrorMessage('Введите имя пользователя и
пароль');
            return;
        }

        try {
            const response = await
fetch('http://localhost:3127/auth/login', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ username, password }),
            });

            if (response.ok) {
                const { token } = await response.json();
                onLogin(token);
                onClose();
            } else {
                setErrorMessage('Неверное имя пользователя или
пароль');
            }
        } catch (error) {
            setErrorMessage('Ошибка подключения к серверу');
```

```
            }
        };

        return (
            <div className="modal-overlay" onClick={onClose}>
                <div   className="modal-content"   onClick={e   =>
e.stopPropagation()}>
                    <input
                        type="text"
                        placeholder="Имя пользователя"
                        value={username}
                        onChange={e => setUsername(e.target.value)}
                    />
                    <input
                        type="password"
                        placeholder="Пароль"
                        value={password}
                        onChange={e => setPassword(e.target.value)}
                    />
                    <button onClick={handleLogin}>Войти</button>
                    {errorMessage    &&    <div    className="error-
message">{errorMessage}</div>}
                </div>
            </div>
        );
    };
    export default AuthModal;
```

4.3.2 Реализация окна просмотра бронирования

Компонент BookingModal — модальное окно для отображения деталей бронирования, отображает информацию о конкретном бронировании. Он показывает данные о номере (гостинице), пользователе и дате бронирования (рисунок 19).

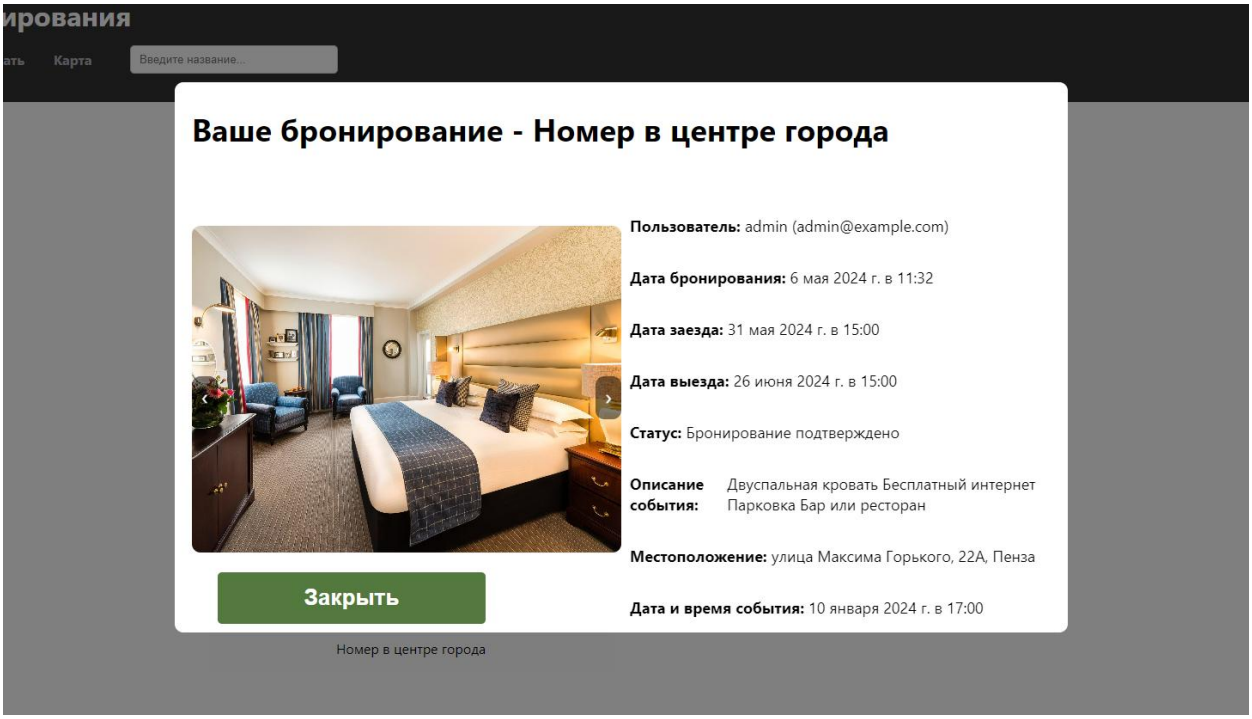


Рисунок 19 - Окно просмотра бронирования

Пользователь может закрыть модальное окно, что приведет к возврату на предыдущий экран без изменения состояния (рисунок 20).

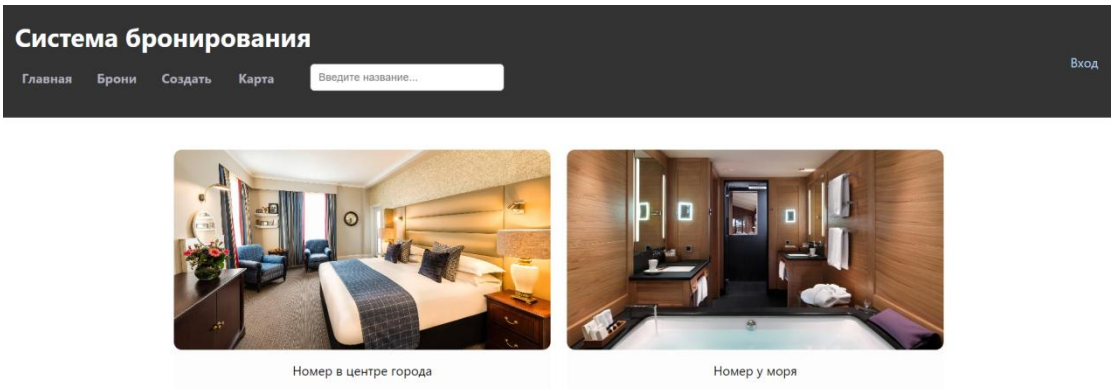


Рисунок 20 - Отображение списка бронирований

Пример реализации:

```
import React from 'react';
import './BookingModal.css';
```

```
const BookingModal = ({ booking, onClose }) => {
    return (
        <div className="modal-overlay" onClick={onClose}>
            <div   className="modal-content"   onClick={e   =>
e.stopPropagation()}>
                <h2>Бронирование для {booking.event.name}</h2>
                <p>Пользователь: {booking.user.username}</p>
                <p>Дата бронирования: {booking.event.date}</p>
                <p>Статус: {booking.status}</p>
                <button onClick={onClose}>Закрыть</button>
            </div>
        </div>
    );
};
export default BookingModal;
```

4.3.3 Реализация главного окна

Компонент EventsPage — страница, которая управляет отображением событий или бронирований, является центральным компонентом, который управляет отображением бронирований (рисунок 20) и списка номеров (рисунок 21). Пользователи могут переключаться между разделами "Главная", "Брони", "Создать" и "Карта".

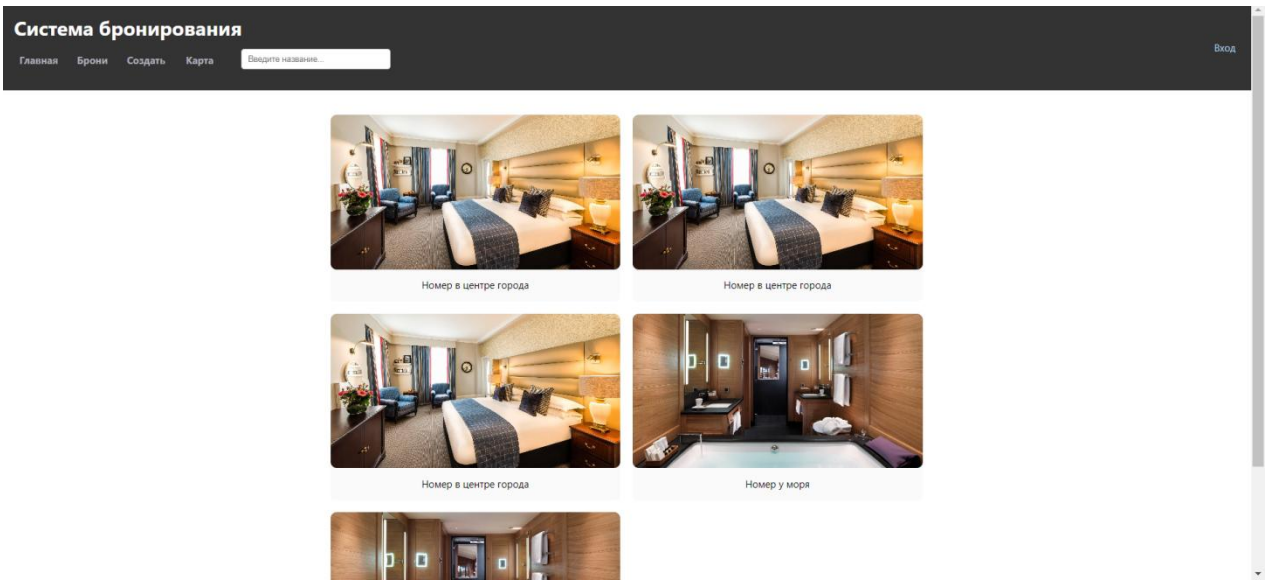


Рисунок 21 - Страница просмотра номерного фонда

В зависимости от выбранного раздела отображаются события (рисунок 21) или бронирования. Есть возможность поиска событий по названию и фильтрации по датам через SimpleCalendar.

Пример реализации:

```
import React, { useState, useEffect } from 'react';
import EventTile from './EventTile';
import BookingModal from './BookingModal';
import AuthModal from './AuthModal';
import './EventsPage.css';

const EventsPage = () => {
    const [events, setEvents] = useState([]);
    const [selectedEvent, setSelectedEvent] = useState(null);
    const [isAuthModalOpen, setIsAuthModalOpen] = useState(false);

    useEffect(() => {
        const fetchEvents = async () => {
            const response = await fetch('http://localhost:3127/api/events');
            const data = await response.json();
```

```
                setEvents(data);
            };

            fetchEvents();
        }, []);

        return (
            <div className="events-page">
                {events.map(event => (
                    <EventTile     key={event.id}     event={event}
onClick={() => setSelectedEvent(event)} />
                ))}
                {selectedEvent && (
                    <BookingModal            booking={selectedEvent}
onClose={() => setSelectedEvent(null)} />
                )}
                {isAuthModalOpen && (
                    <AuthModal             onClose={()             =>
setIsAuthModalOpen(false)} onLogin={(token) => console.log(token)}
/>
                )}
            </div>
        );
    };
    export default EventsPage;
```

4.3.4 Реализация окна просмотра и бронирования номера

Компонент Modal - модальное окно просмотра объявления для просмотра деталей событий. Включает в себя изображения события, описание, время и место проведения (рисунок 22).

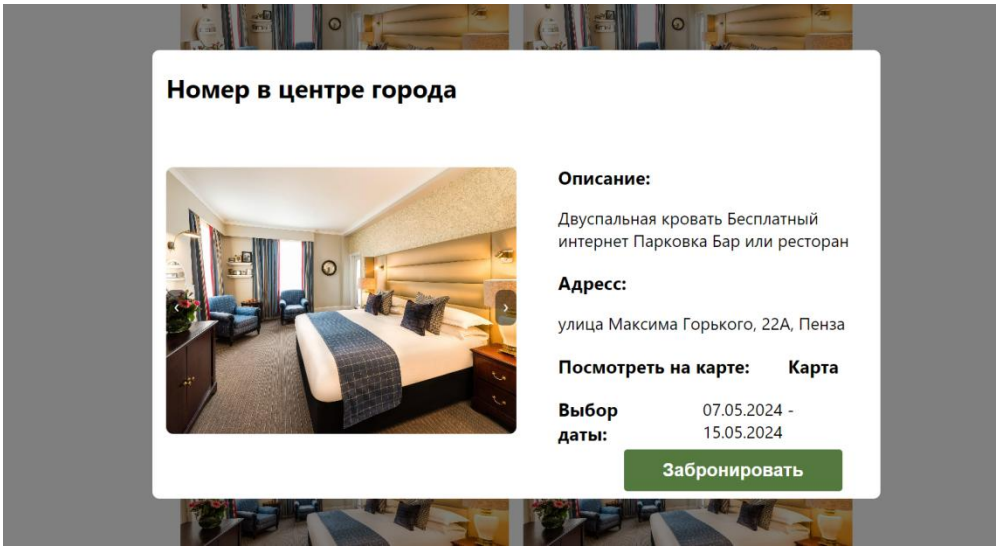


Рисунок 22 - Модальное окно просмотра объявления

Пользователи могут просматривать фотографии события, выбирать даты для бронирования и отправлять запрос на бронирование через встроенный календарь (рисунок 23).

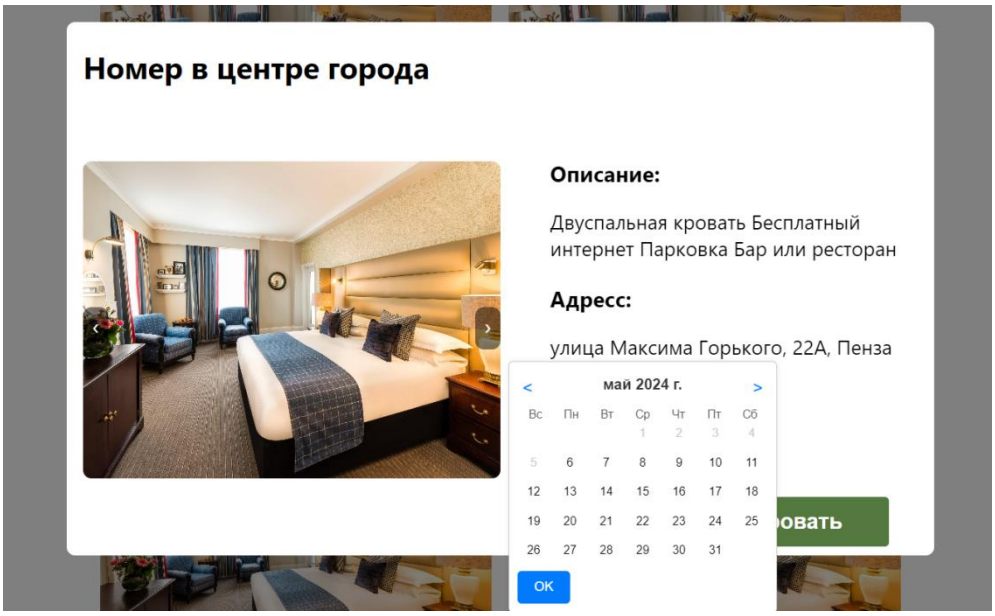


Рисунок 23 - Календарь

Календарь отображает только доступные даты для бронирования, например, уже прошедшие или занятые не будут доступны для выбора.

Пример реализации:

```
import React, { useState } from 'react';
import './Modal.css';

const Modal = ({ event, onClose }) => {
    const [currentImageIndex, setCurrentImageIndex] = useState(0);
    const imagePaths = event.imagePath.split(',');

    const handleNextImage = () => {
        setCurrentImageIndex((prevIndex) => (prevIndex + 1) % imagePaths.length);
    };

    const handlePrevImage = () => {
        setCurrentImageIndex((prevIndex) => (prevIndex - 1 + imagePaths.length) % imagePaths.length);
    };

    return (
        <div className="modal-overlay" onClick={onClose}>
            <div className="modal-content" onClick={(e) => e.stopPropagation()}>
                <h1>{event.name}</h1>
                <img src={`http://localhost:3127/api/images/${imagePaths[currentImageIndex]}`} alt="Event" />
                {imagePaths.length > 1 && (
                    <>
                        <button onClick={handlePrevImage}>Previous</button>
                        <button onClick={handleNextImage}>Next</button>
                    </>
                )}
                <p>{event.description}</p>
```

```
                    <button onClick={onClose}>Close</button>
                </div>
            </div>
        );
    };

    export default Modal;
```

4.3.5 Реализация календаря

Компонент SimpleCalendar (рис. 23) — компонент календаря для выбора дат, используется для выбора даты начала и окончания бронирования. Он предоставляет пользовательский интерфейс для взаимодействия с календарем.

Пользователи могут выбирать даты в рамках определенного периода и подтверждать свой выбор. Компонент также уведомляет другие компоненты о выборе даты через callback-функцию.

Пример реализации:

```
    import React, { useState } from 'react';
    import './SimpleCalendar.css';

    const SimpleCalendar = ({ onSelectDate }) => {
        const [selectedDate, setSelectedDate] = useState(new
Date());

        const handleDateChange = (event) => {
            setSelectedDate(new Date(event.target.value));
        };

        const handleSubmit = () => {
            onSelectDate(selectedDate);
        };

        return (
            <div className="calendar-container">
```

```
                <input                                          type="date"
value={selectedDate.toISOString().substr(0,                          10)}
onChange={handleDateChange} />
                <button                onClick={handleSubmit}>Select
Date</button>
            </div>
        );
    };
    export default SimpleCalendar;
```

4.3.6 Реализация элемента плитки объявления

EventTile — Компонент плитки объявления (рисунок 24).

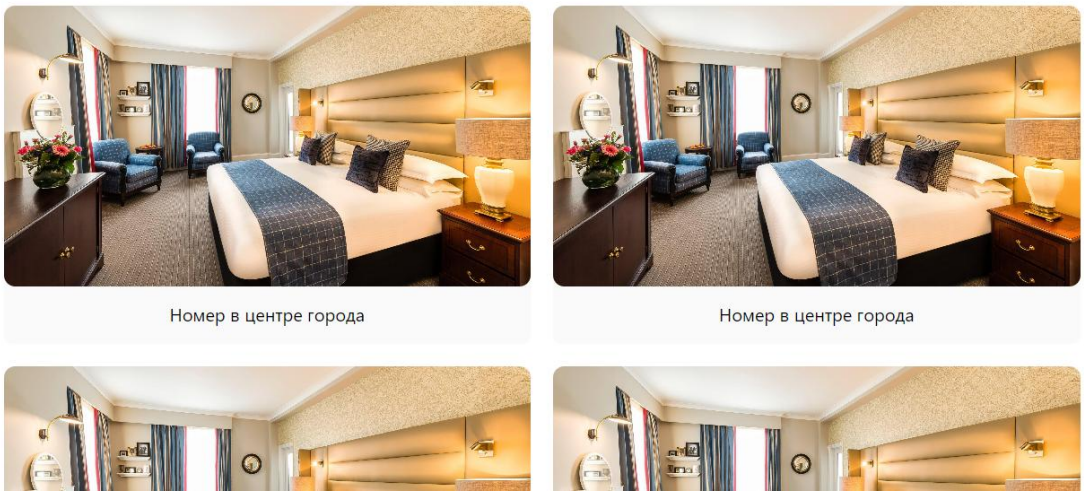


Рисунок 24 - Компонент плитки

Пример реализации:

```
    import React from 'react';
    import './EventTile.css';

    const EventTile = ({ event, onClick }) => {
        return (
            <div className="event-tile" onClick={onClick}>
                <img
src={`http://localhost:3127/api/images/${event.imagePath.split(',
')[0]}`} alt="Event" />
```

```
                <div className="event-info">
                    <h3>{event.name}</h3>
                    <p>{event.description}</p>
                </div>
            </div>
        );
    };
    export default EventTile;
```

4.3.7 Реализация окна добавления объявления

Компонент CreateEventModal представляет из себя окно обновления номерного фонда, даёт пользователю удобный способ добавления объявлений (рисунок 25). Оно содержит следующие элементы:

Поля для заполнения информации о номере: Администраторы могут ввести название, его описание, местоположение и загрузить изображения.

Кнопка "Создать": После заполнения всех необходимых полей пользователь может нажать кнопку "Создать", чтобы добавить номер в список номерного фонда. При этом данные о событии отправляются на сервер для сохранения.

Пользователь может закрыть окно добавления, если он передумал, нажав на кнопку "Закрыть".

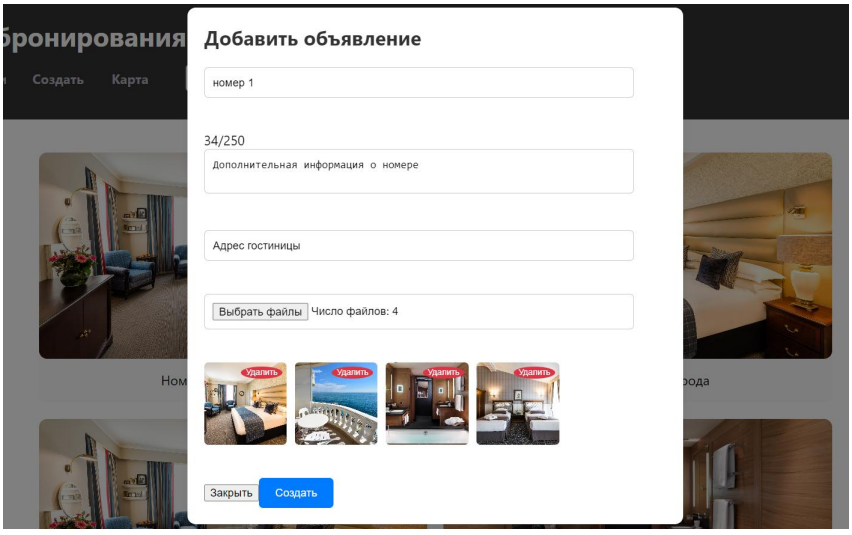


Рисунок 25 - Окно добавления объявления

4.3.8 Реализация карты

Компонент MapEvent интегрируется с API карт (Yandex Maps) [4] для отображения мероприятий на карте (рисунок 27). Пользователи могут взаимодействовать с картой, выбирая гостиницы, номера или просматривая подробную информацию о местоположении.

После выбора номера на карте пользователь может сразу перейти к просмотру деталей. Для этого используется модальное окно, которое отображает информацию о выбранном номере, включая фотографии, описание, даты и место.

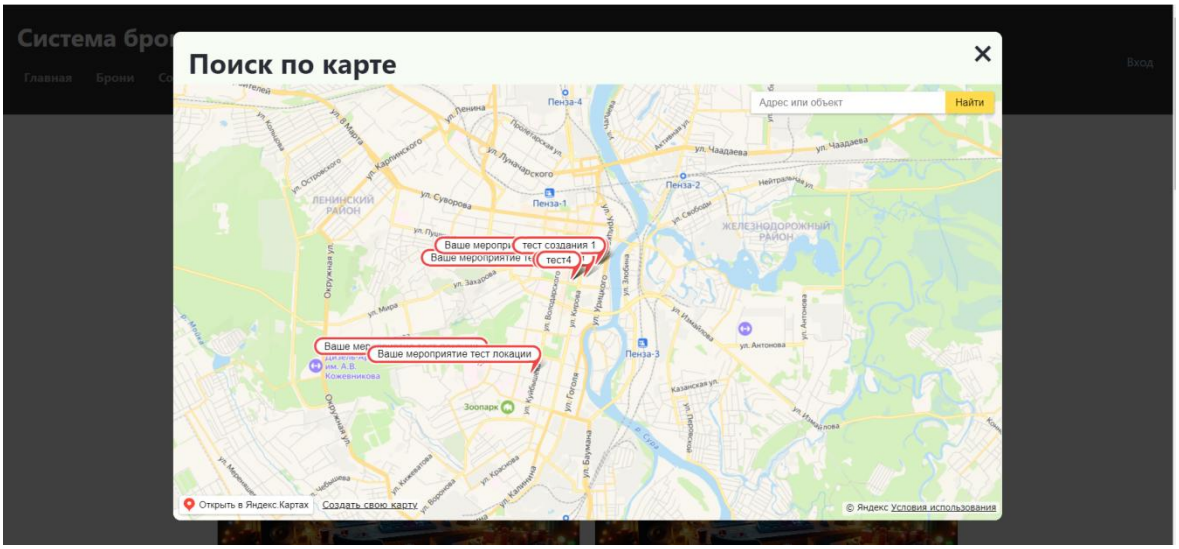


Рисунок 27 - Карта

Поиск на карте реализован с помощью интеграции с API карт и добавлением функционала поиска мест. Пользователи могут вводить названия или их части в поисковую строку, и карта автоматически центрируется на местах, которые соответствуют запросу.

Пример кода компонента MapEvent:

```
import React, { useState, useEffect } from 'react';
import { YMaps, Map, Placemark } from '@pbe/react-yandex-maps';

const MapEvent = ({ events, onSelectEvent }) => {
```

```
        const [mapState, setMapState] = useState({ center:
[55.751574, 37.573856], zoom: 10 });

        useEffect(() => {
            if (events.length > 0) {
                // Обновляем центр карты для отображения всех
мероприятий
                const firstEvent = events[0];
                setMapState({ ...mapState, center:
[firstEvent.location.lat, firstEvent.location.lon] });
            }
        }, [events]);

        return (
            <YMaps>
                <div className="map-container">
                    <Map state={mapState} width="100%"
height="400px">
                        {events.map(event => (
                            <Placemark
                                key={event.id}
                                geometry={[event.location.lat,
event.location.lon]}
                                onClick={() => onSelectEvent(event)}
                                properties={{ hintContent:
event.name }}
                            />
                        ))}
                    </Map>
                </div>
            </YMaps>
        );
    };
    export default MapEvent;
```

– ymaps: Обертка, предоставляемая библиотекой react-yandex-maps, необходимая для работы с Yandex Maps API;

– map: Компонент, который рендерит карту. state включает в себя начальные параметры центра карты и уровня масштабирования (zoom);

– placemark: Компонент, который отмечает точку на карте. Каждый Placemark связан с мероприятием и позволяет при клике на маркер вызвать функцию onSelectEvent, передавая в неё объект номера.

Интеграция карты в систему бронирования позволяет пользователю не только видеть адресс, но и управлять выбором через графический интерфейс. .

Компоненты тесно интегрированы благодаря использованию контекста React и хуков состояния. Это позволяет эффективно управлять состояниями приложения и обеспечивать своевременное обновление пользовательского интерфейса при изменении данных. Взаимодействие с сервером осуществляется через асинхронные запросы fetch, обрабатывающие данные JSON для отправки и получения информации от сервера.

Клиентская часть на React обеспечивает динамичное и интерактивное взаимодействие с пользователем, поддерживая высокий уровень абстракции и повторного использования кода. Это делает разработку и поддержку приложения более эффективными и позволяет легко масштабировать функционал с течением времени.

5. Руководство пользователя

При переходе на сайт веб приложение пользователь попадает на главную страницу, где отображается список номерного фонда гостиницы (рисунок 28).

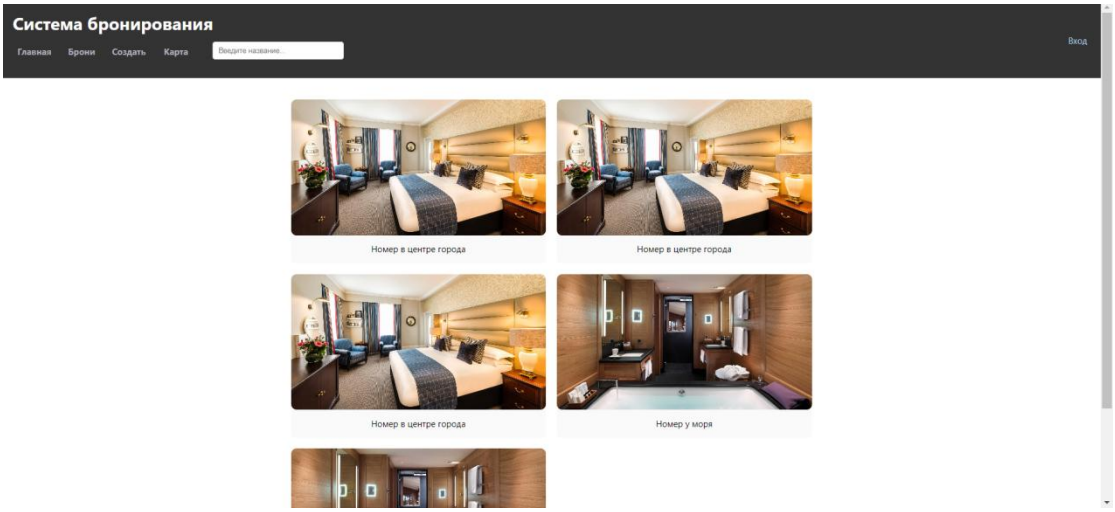


Рисунок 28 - Главная страница

На данном этапе пользователь может продолжить просмотр номеров, перейти к окну просмотра номера и бронирования (рисунок 29) или авторизироваться (рисунок 31).

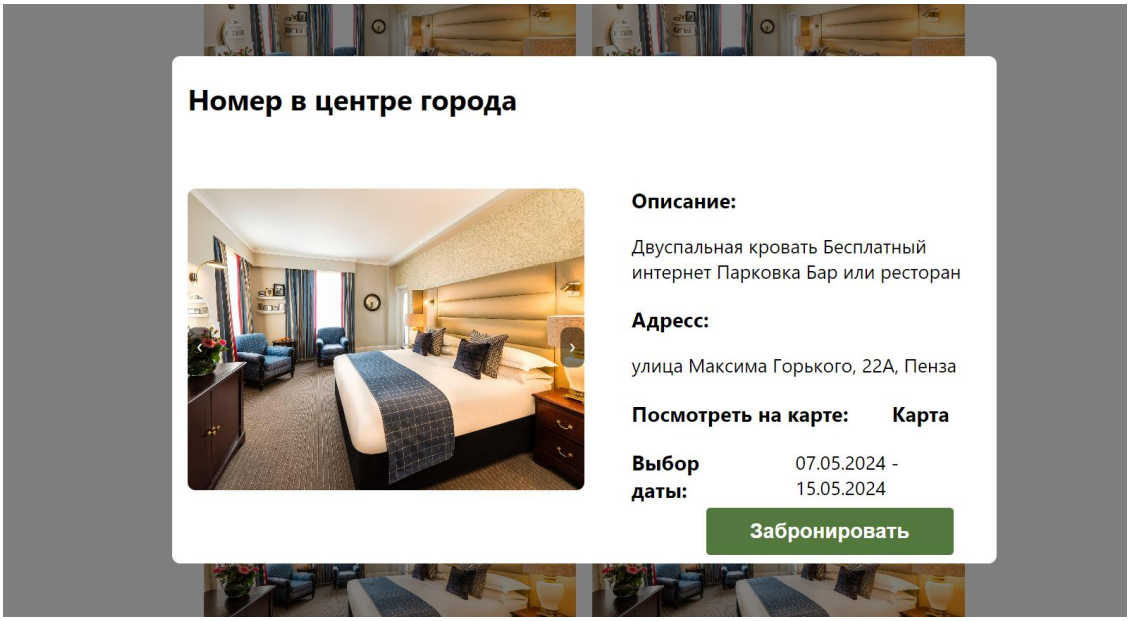


Рисунок 29 - Окно просмотра номера и бронирования

Окно просмотра номера и бронирования содержит информацию о номере, фотографии номера, возможность посмотреть местоположение конкретного номера на карте и выбрать период бронирования (рисунок 30).

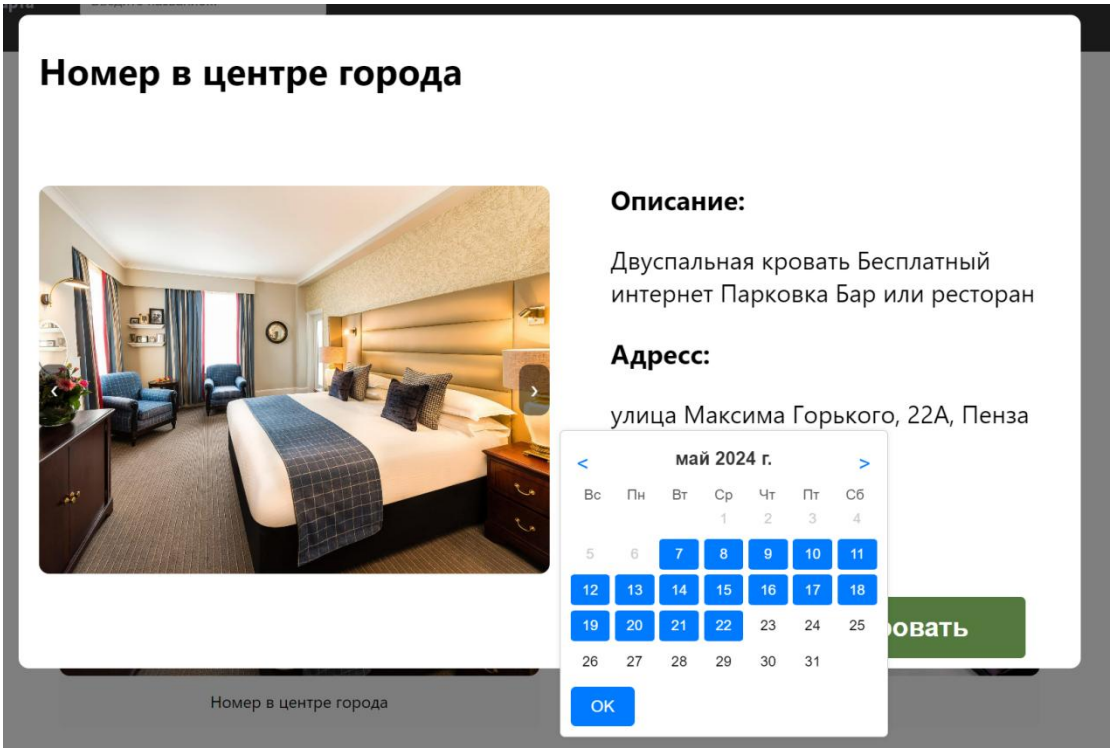


Рисунок 30 - Выбор периода бронирования в календаре

В случае если пользователь нажмёт на кнопку «Забронировать» не авторизированным, то откроется окно авторизации (рисунок 31). Где пользователь должен указать Имя пользователя и пароль, после чего по нажатию на кнопку войти произойдёт вход в систему под учётной записью пользователя или администратора в зависисмоти от указанных данных входа.

После чего пользователь сможет забронировать номер и он отобразится в разделе «бронирования» (рисунок 32).

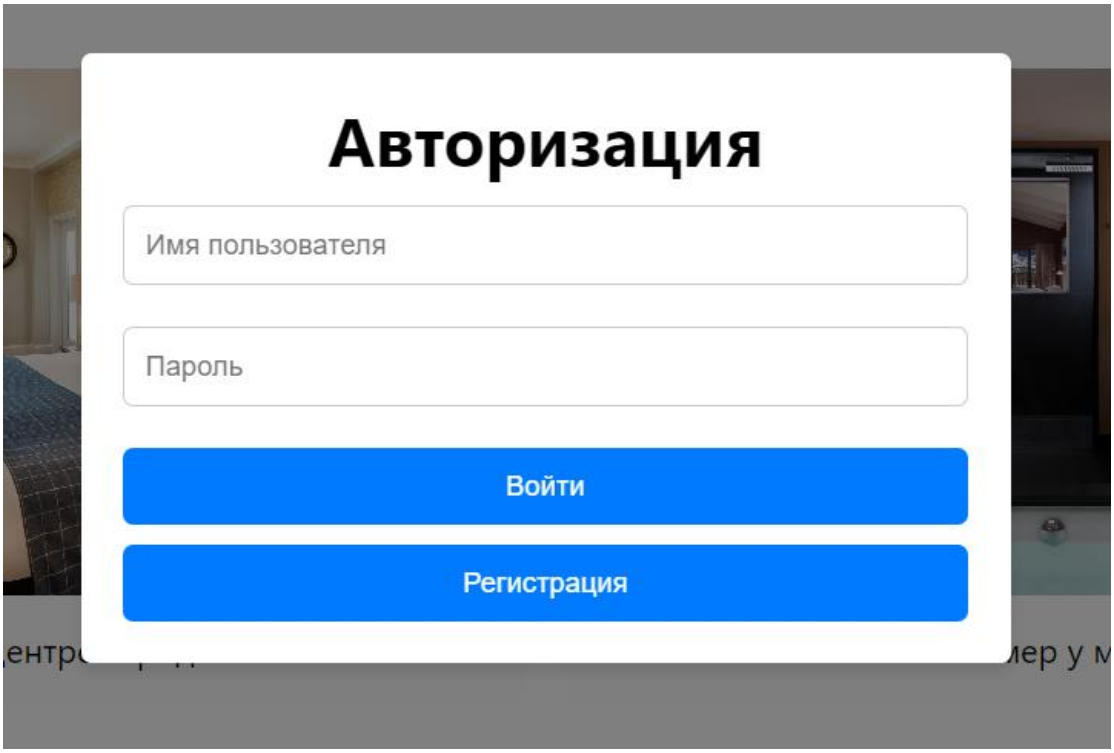


Рисунок 31 - окно авторизации

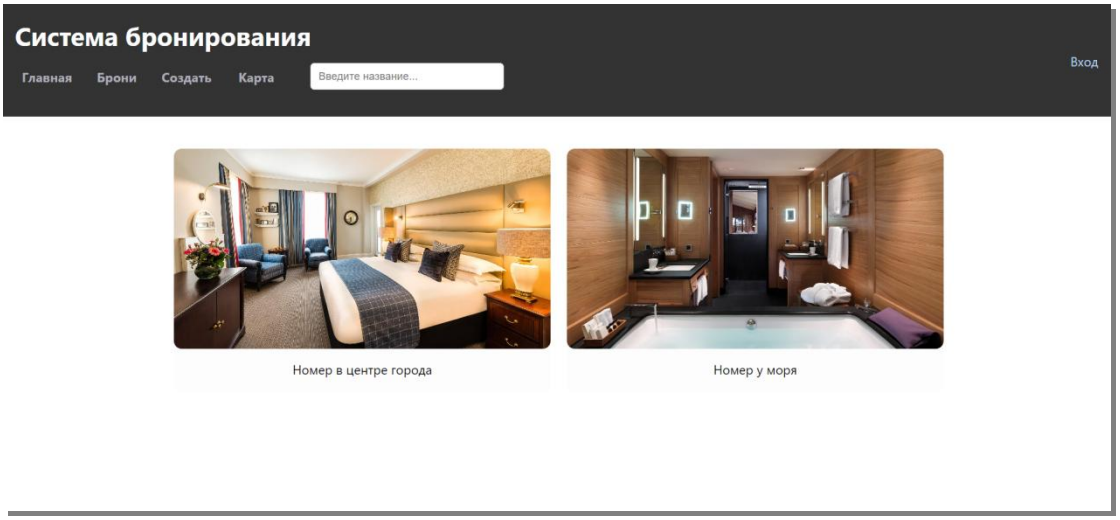


Рисунок 32 - Раздел бронирования

В разделе бронирования пользователь может нажать на любой номер который он забронировал в результате чего откроется окно просмотра бронирования, где отображается подробная информация о номере и данных бронирования включая дату заезда и выезда (рисунок 33).

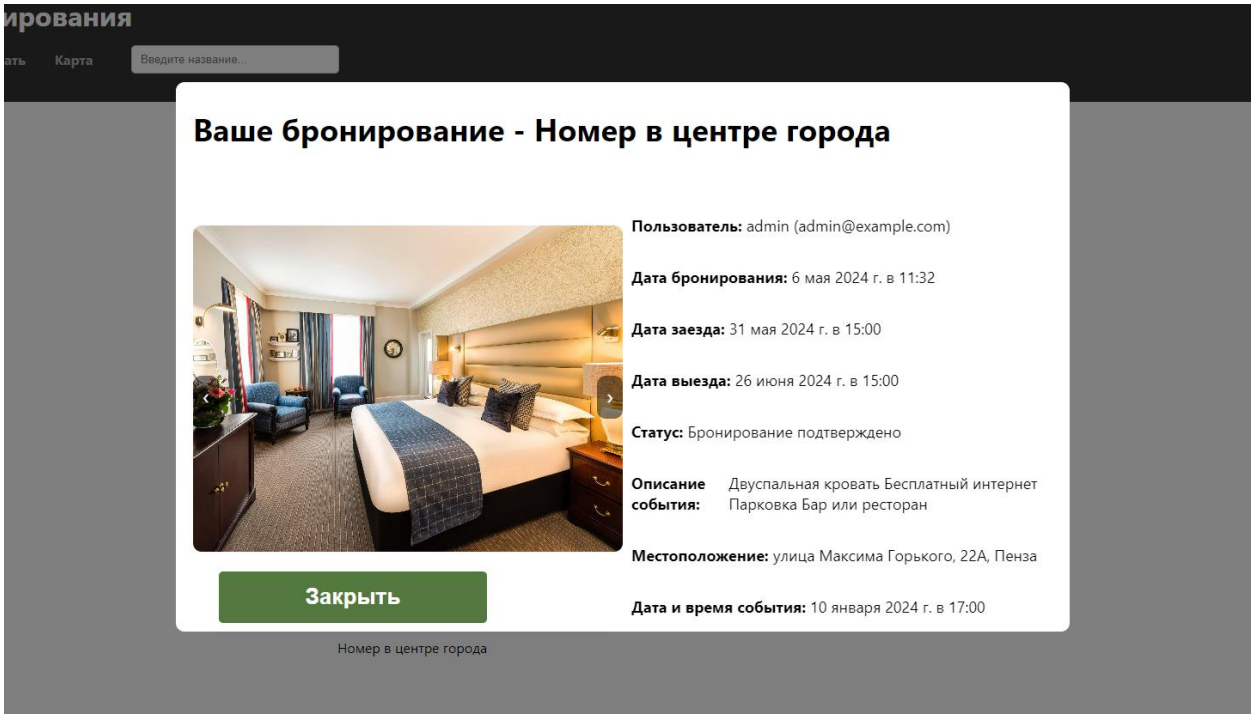


Рисунок 33 - Просмотр бронирования

В шапке главной страницы в случае если пользователь авторизован под учётной записью администратора отображается кнопка «Создать», по нажатию на которую открывается окно добавления объявления в систему (рисунок 34).

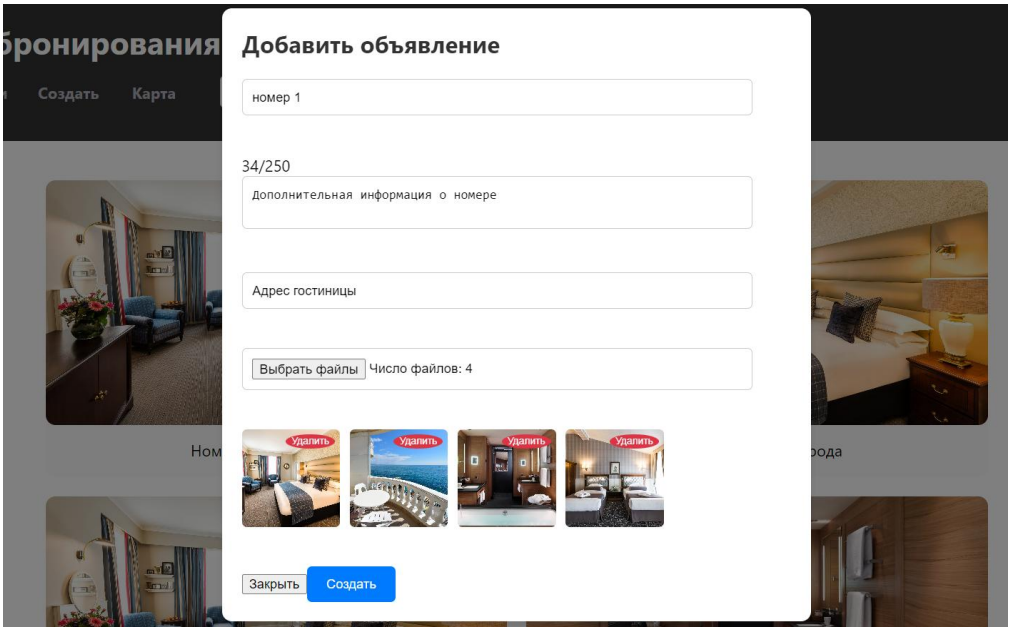


Рисунок 34 - Окно добавление объявления

6 Безопасность жизнедеятельности. Стратегии безопасности при работе с компьютером

Продолжительное нахождение за компьютером может привести к скрытому и постепенному ухудшению здоровья из-за накопления вредного воздействия. Такое ухудшение часто остается незамеченным до тех пор, пока не проявятся серьезные проблемы со здоровьем. Среди наиболее частых и серьезных проблем, связанных с долгими сессиями за компьютером, можно выделить следующие:

– синдром «сухого глаза», который возникает из-за чрезмерной нагрузки на зрение.

– разнообразные нарушения в суставах, спровоцированные неправильной посадкой в течение долгого времени.

Применение простых мер предосторожности может значительно уменьшить риски для здоровья:

– важно поддерживать оптимальное расстояние до экрана (обычно 50-70 см), чтобы уменьшить нагрузку на глаза. Рассмотрение использования специальных защитных экранов или очков для блокировки синего света также может быть полезным.

– для снижения нагрузки на шейные позвонки рекомендуется держать экран под углом приблизительно 30 градусов.

– расположение клавиатуры должно быть на расстоянии 20-30 см от края стола, на уровне, который позволяет сохранять руки в естественном и расслабленном положении. Использование подставки для запястий может помочь избежать развития синдрома запястного канала.

– крайне важно, чтобы стул или кресло поддерживали правильную осанку, обеспечивая незначительную поддержку для спины.

– ноги должны сохранять естественное положение, быть прямыми перед собой и полностью опираться на пол, при этом колени должны формировать угол в 90 градусов.

– особое внимание следует уделять регулярным перерывам: каждый час следует устраивать короткие паузы для выполнения упражнений для глаз и шеи, что помогает снижать напряжение и утомляемость.

– освещение рабочего места: Обеспечение адекватного освещения рабочего места существенно снижает нагрузку на глаза. Избегайте слишком яркого света над головой или прямого света на экран. Используйте настольную лампу с рассеянным светом, которая освещает документы, но не отражается на экране.

– программное обеспечение для напоминаний: Существуют приложения, которые могут напоминать вам о необходимости делать перерывы. Такие программы помогают организовать рабочий день таким образом, чтобы чередовать умственную активность с физической разминкой.

– регулировка влажности и температуры: Поддержание комфортной влажности и температуры в рабочей зоне также важно для общего самочувствия. Сухой воздух может усугубить синдром сухого глаза, а неоптимальная температура способствует утомляемости.

– правильное питание и гидратация: Регулярное потребление чистой воды и сбалансированное питание, богатое витаминами и минералами, поддерживает энергетический уровень и способствует общему здоровью. Особенно важны продукты, содержащие Омега-3 жирные кислоты и антиоксиданты, которые полезны для глаз.

– настройка рабочего кресла и стола: Инвестиции в эргономичное рабочее кресло и стол, которые можно настроить под индивидуальные параметры тела, способствуют правильной осанке и уменьшают риск возникновения боли в спине и суставах.

Соблюдение этих простых правил поможет значительно снизить риски для здоровья при длительной работе за компьютером.

Заключение

В процессе выполнения ВКР была разработана система управления гостиничным номерным фондом и бронирования. Клинская часть проекта представляет веб-приложение, предлагающие пользователям удобный интерфейс для просмотра доступных номеров, создания новых объявлений и бронирования номеров.

Интерфейс системы сделан таким образом, чтобы быть удобным и интуитивно понятным для пользователей, обеспечивая легкий доступ ко всем возможностям системы. Веб-приложение позволяет просматривать список номеров, добавлять новые объявления и бронировать номера.

Система обладает различными функциональными возможностями, например, добавление, редактирование и удаление данных из номерного фонда, а также управление бронированием. Также реализована система аутентификации и авторизации пользователей для обеспечения безопасности данных.

Проект разработан с использованием различных современных технологий, таких как React.js для фронтенда, Spring Framework для бэкенда и база данных PostgreSQL для хранения данных. Это обеспечивает высокую производительность и современную архитектуру системы.

Данный дипломный проект является основой для дальнейшего развития и расширения функциональности системы бронирования. В будущем можно добавить дополнительных возможностей для пользователей, улучшение интерфейса и оптимизация производительности.

Итогом выполнения дипломного проекта стала функциональная и эффективная система бронирования, предоставляющая пользователям различные инструменты для поиска, выбора и бронирования номеров. Проект демонстрирует применение современных технологий веб-разработки и может служить основой для создания аналогичных систем..

Список использованных источников

1. Pivotal Software, Inc. "Документация Java Spring Framework". URL: https://spring.io/projects/spring-framework (дата обращения 22.04.2024).
2. Meta Platforms, Inc. "Документация React". URL: https://reactjs.org/docs/getting-started.html (дата обращения 23.04.2024).
3. PostgreSQL Global Development Group. "Документация PostgreSQL". URL: https://www.postgresql.org/docs/ (дата обращения 25.04.2024).
4. Яндекс. "Документация Яндекс. Карты API". URL: https://yandex.ru/dev/maps/ (дата обращения 5.05.2024).
5. Auth0. Введение в JWT (JSON Web Tokens)". URL: https://jwt.io/introduction/ (дата обращения 25.04.2024).
6. Redgate. "Документация Flyway". URL: https://flywaydb.org/documentation/ (дата обращения 27.04.2024).
7. Pivotal Software, Inc. "Документация Spring Security". URL: https://docs.spring.io/spring-security/reference/index.html (дата обращения 28.04.2024).

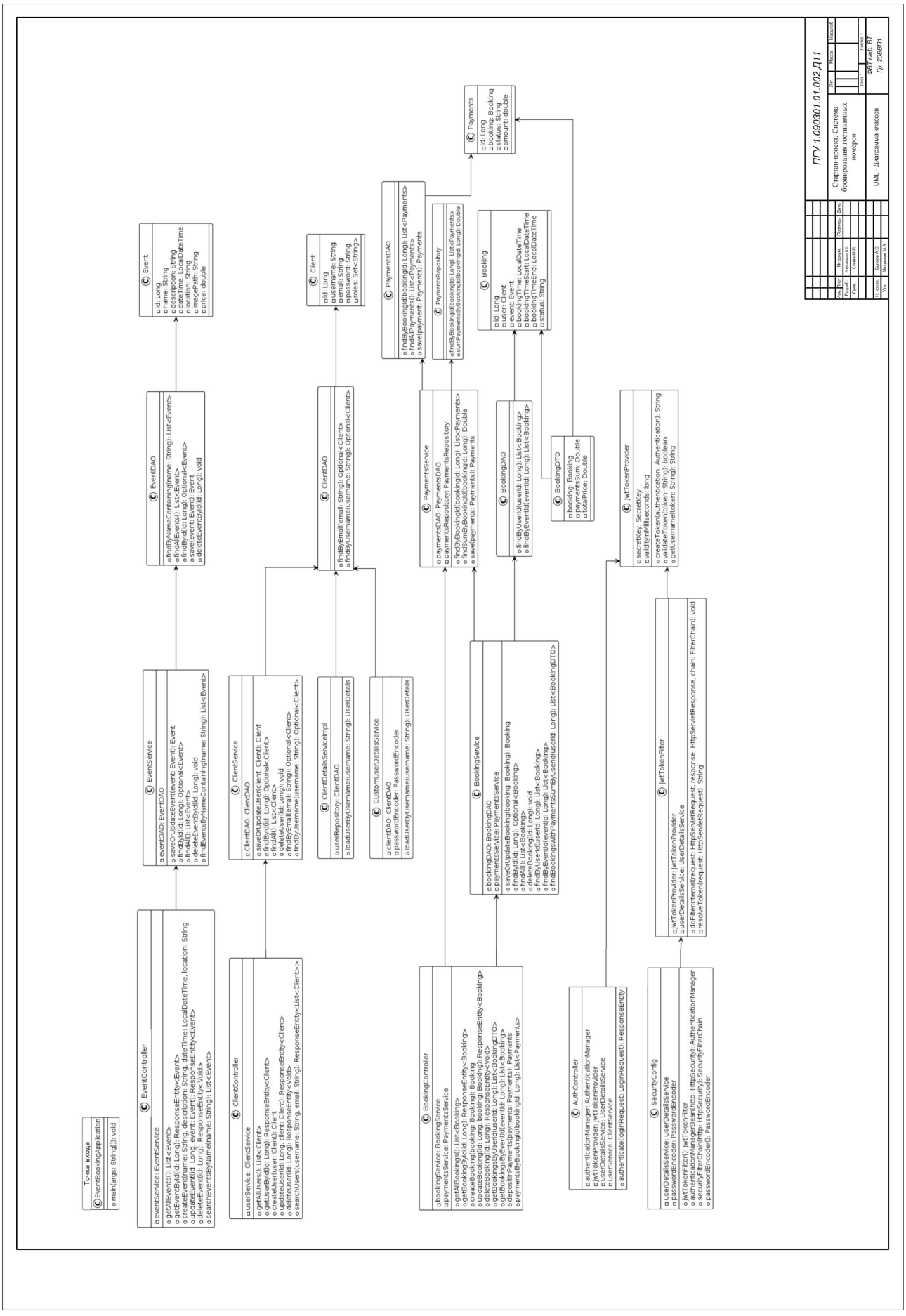
Приложение А

(Обязательное)

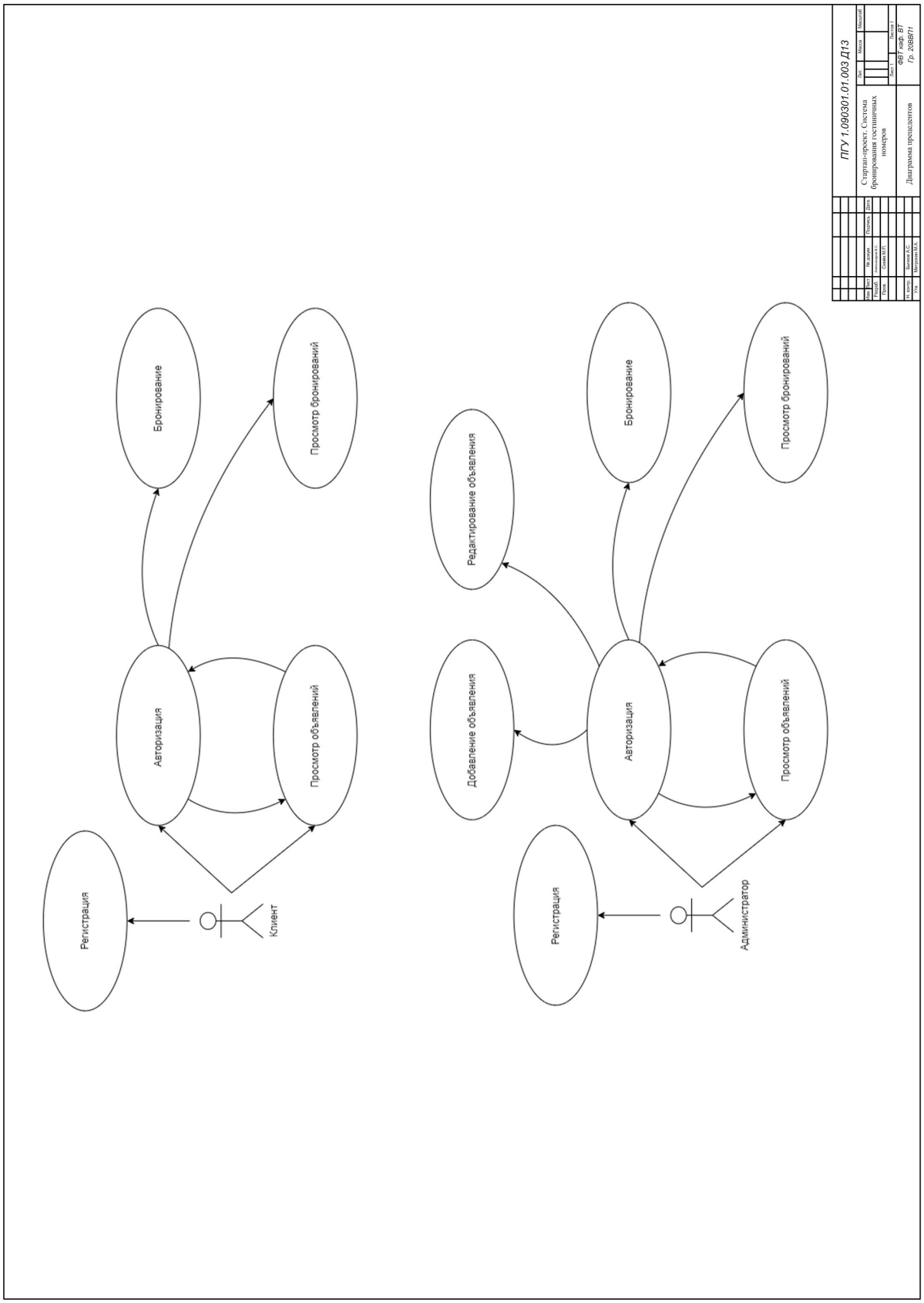
Система бронирования гостиничных номеров

UML-диаграммы

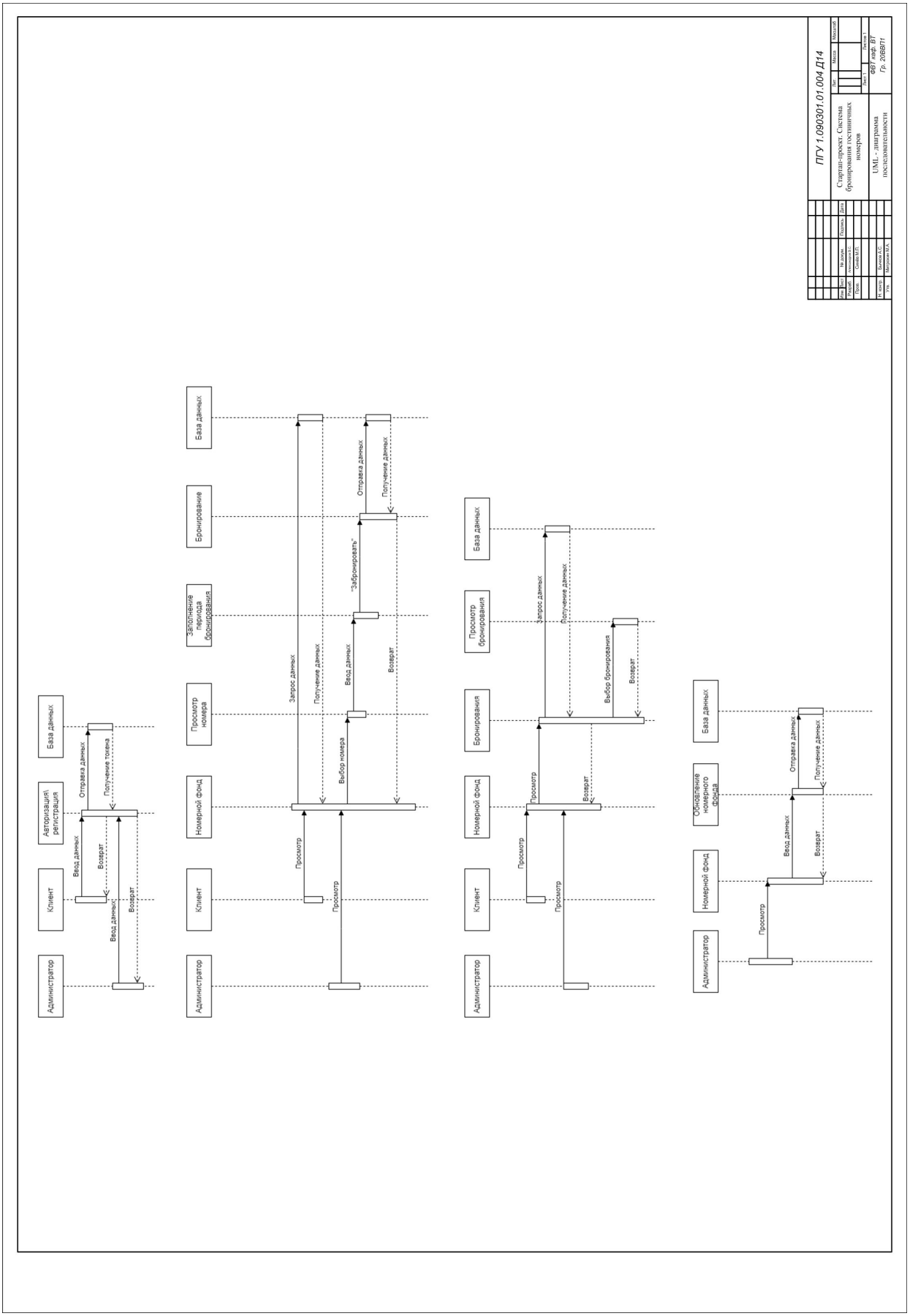
Приложение А.1 – Диаграмма классов



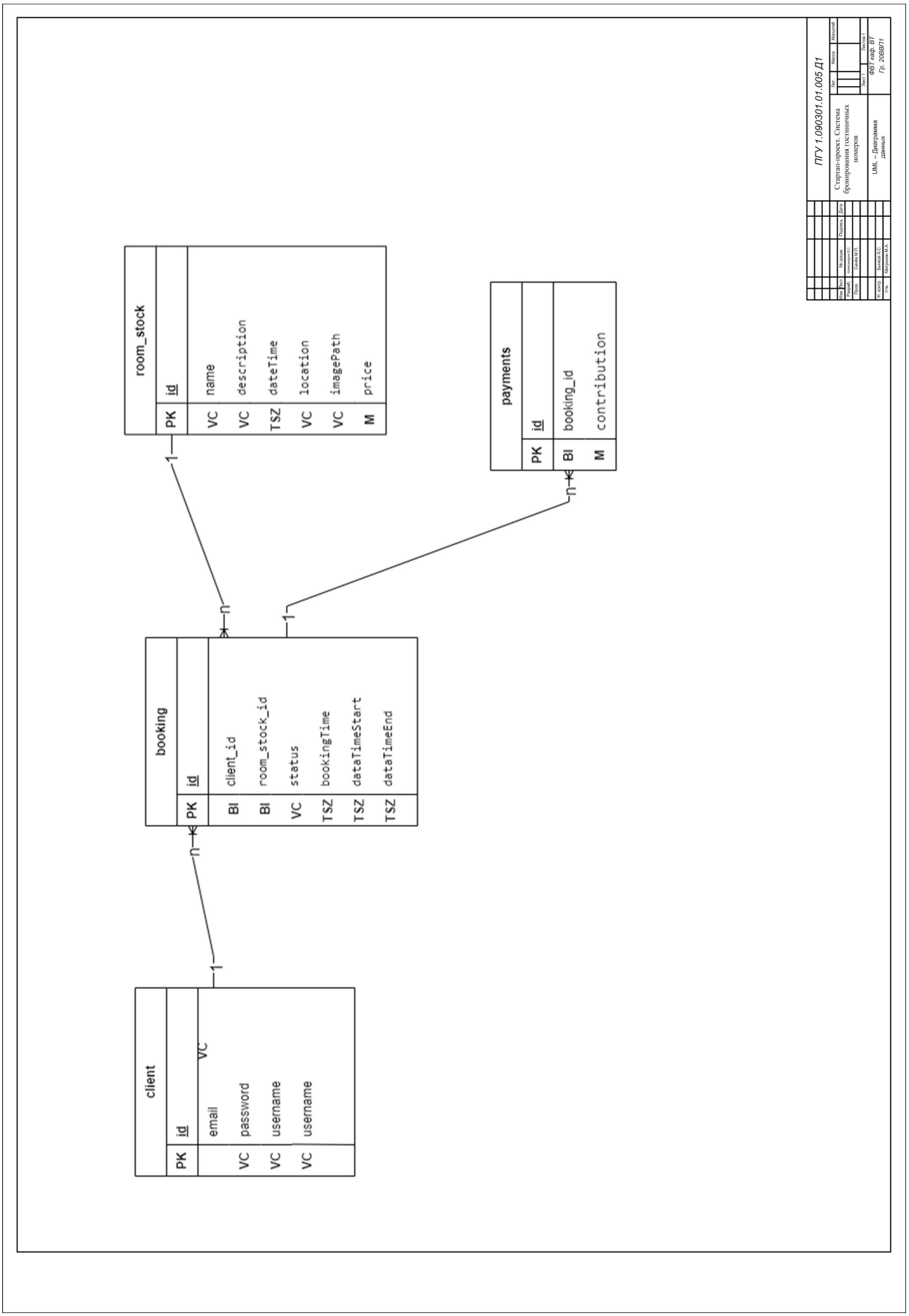
Приложение А.2 – Диаграмма прецедентов



Приложение А.3 – Диаграмма последовательности



Приложение А.4 – Диаграмма данных



Приложение Б

(Обязательное)

Система бронирования гостиничных номеров

Листинг программы

Приложение Б.1 – Листинг сервера на Spring Framework

Файл: EventBookingApplication.java

```
package com.example;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class EventBookingApplication {
	public static void main(String[] args) {
	SpringApplication.run(EventBookingApplication.class, args);
	}
}
```

Файл: JwtResponse.java

```
package com.example;
public class JwtResponse {
	private String token;
	private String type = "Bearer";
	public JwtResponse(String token) {
		this.token = token;
	}
	public String getToken() {
		return token;
	}
	public void setToken(String token) {
		this.token = token;
	}
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
}
```

Файл: JwtTokenProvider.java

```
package com.example;

import com.example.model.Client;
import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import jakarta.servlet.http.HttpServletRequest;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import javax.crypto.SecretKey;
import java.util.Date;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
@Component
public class JwtTokenProvider {
    private final SecretKey secretKey = Keys.secretKeyFor(SignatureAlgorithm.HS256);
    @Value("${jwt.expiration}")
    private long validityInMilliseconds;
    public String createToken(Authentication authentication) {
        Claims claims = Jwts.claims().setSubject(authentication.getName());
        claims.put("roles", authentication.getAuthorities());
        Date now = new Date();
        Date validity = new Date(now.getTime() + validityInMilliseconds);
        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(validity)
                .signWith(SignatureAlgorithm.HS256, secretKey)
                .compact();
    }
    public boolean validateToken(String token) {
        try {
        Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);
            return true;
```

```
            } catch (JwtException | IllegalArgumentException e) {
                throw new IllegalArgumentException("Expired or invalid JWT token");
            }
        }
        public String getUsername(String token) {
            return Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token).getBody().getSubject();
        }
}
```

Файл: LoginRequest.java

```
package com.example;
public class LoginRequest {
    private String username;
    private String password;
    public LoginRequest() {
    }
    public LoginRequest(String username, String password) {
        this.username = username;
        this.password = password;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Файл: SecurityConfig.java

```
package com.example;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuild
er;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import static org.springframework.security.config.Customizer.withDefaults;

@EnableWebSecurity
@Configuration
public class SecurityConfig {
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private PasswordEncoder passwordEncoder;
    @Bean
    public JwtTokenFilter jwtTokenFilter() {
        return new JwtTokenFilter();
    }
    @Bean
    public AuthenticationManager authenticationManagerBean(HttpSecurity http) throws Exception
{
        AuthenticationManagerBuilder authenticationManagerBuilder =
http.getSharedObject(AuthenticationManagerBuilder.class);
        authenticationManagerBuilder
                .userDetailsService(userDetailsService)
                .passwordEncoder(passwordEncoder);

        return authenticationManagerBuilder.build();
    }
```

```
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests((requests) -> requests
                        .requestMatchers("/api/clients/**").hasAuthority("ROLE_ROLE_ADMIN")
                        .requestMatchers("/api/images/**").permitAll()
                        .requestMatchers("/api/events/get_all").permitAll()
                        .requestMatchers("/api/events/search").permitAll()
                        .requestMatchers(HttpMethod.POST,
"/api/events/create").hasAuthority("ROLE_ROLE_ADMIN")

                        .anyRequest().authenticated())
                .httpBasic(withDefaults());
        http
                .csrf((csrf) -> csrf.disable())
        ;
        http.addFilterBefore(jwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }
}
```

Файл: AuthController.java

```
package com.example.controller;
import com.example.JwtResponse;
import com.example.JwtTokenProvider;
import com.example.LoginRequest;
import com.example.service.ClientService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetailsService;
```

```
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;
@RestController
@RequestMapping("/auth")
public class AuthController {
    @Autowired
    private AuthenticationManager authenticationManager;
    @Autowired
    private JwtTokenProvider jwtTokenProvider;
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private    ClientService userService;
    @CrossOrigin
    @PostMapping("/login")
    public ResponseEntity<?> authenticate(@RequestBody LoginRequest loginRequest) {
        try {
            Authentication authentication = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            loginRequest.getUsername(),
                            loginRequest.getPassword()
                    )
            );
             SecurityContextHolder.getContext().setAuthentication(authentication);
            String jwt = jwtTokenProvider.createToken(authentication);
            Object user = userService.findByUsername(loginRequest.getUsername()).map(u -> {
                return Map.of("id", u.getId(), "roles", u.getRoles());
            }).orElse(null);

            return ResponseEntity.ok(List.of(new JwtResponse(jwt), user));
        } catch (AuthenticationException e) {
            return    ResponseEntity.badRequest().body("Неверное    имя    пользователя    или
пароль");
        }
    }
}
```

Файл: BookingController.java

```
package com.example.controller;
import com.example.model.Booking;
import com.example.model.DTO.BookingDTO;
import com.example.model.Payments;
import com.example.service.BookingService;
import com.example.service.PaymentsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/bookings")
public class BookingController {
    private final BookingService bookingService;
    private final PaymentsService paymentsService;
    @Autowired
    public BookingController(BookingService bookingService, PaymentsService paymentsService) {
        this.bookingService = bookingService;
        this.paymentsService = paymentsService;
    }
    @CrossOrigin
    @GetMapping
    public List<Booking> getAllBookings() {
        return bookingService.findAll();
    }
    @CrossOrigin
    @GetMapping("/{id}")
    public ResponseEntity<Booking> getBookingById(@PathVariable Long id) {
        return bookingService.findById(id)
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
    }
    @CrossOrigin
    @PostMapping
    public Booking createBooking(@RequestBody Booking booking) {
        return bookingService.saveOrUpdateBooking(booking);
```

```
    }
    @CrossOrigin
    @PutMapping("/{id}")
    public ResponseEntity<Booking> updateBooking(@PathVariable Long id, @RequestBody Booking
booking) {
        return bookingService.findById(id)
                .map(existingBooking -> {
                    booking.setId(id);
                    return ResponseEntity.ok(bookingService.saveOrUpdateBooking(booking));
                })
                .orElseGet(() -> ResponseEntity.notFound().build());
    }

    @CrossOrigin
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteBooking(@PathVariable Long id) {
        return bookingService.findById(id)
                .map(booking -> {
                    bookingService.deleteBooking(id);
                    return ResponseEntity.ok().<Void>build();
                })
                .orElseGet(() -> ResponseEntity.notFound().build());
    }
    @CrossOrigin
    @GetMapping("/user/{userId}")
    public List<BookingDTO> getBookingsByUserId(@PathVariable Long userId) {
        return bookingService.findBookingsWithPaymentsSumByUserId(userId);
    }
    @CrossOrigin
    @GetMapping("/event/{eventId}")
    public List<Booking> getBookingsByEventId(@PathVariable Long eventId) {
        return bookingService.findByEventId(eventId);
    }
    @CrossOrigin
    @PostMapping("/payments")
    public Payments depositInPayments(@RequestBody Payments payments) {
        return paymentsService.save(payments);
    }
```

```
    @CrossOrigin
    @GetMapping("/payments/getByBookingId/{bookingId}")
    public List<Payments> paymentsByBookingId(@PathVariable Long bookingId) {
        return paymentsService.findByBookingId(bookingId);
    }
}
```

Файл: ClientController.java

```
package com.example.controller;
import com.example.model.Client;
import com.example.service.ClientService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/clients")
public class ClientController {

    private final ClientService userService;
    @Autowired
    public ClientController(ClientService userService) {
        this.userService = userService;
    }
    @CrossOrigin
    @GetMapping
    public List<Client> getAllUsers() {
        return userService.findAll();
    }
    @CrossOrigin
    @GetMapping("/{id}")
    public ResponseEntity<Client> getUserById(@PathVariable Long id) {
        return userService.findById(id)
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
    }
    @CrossOrigin
```

```
    @PostMapping
    public Client createUser(@RequestBody Client user) {

        return userService.saveOrUpdateUser(user);
    }
    @CrossOrigin
    @PutMapping("/{id}")
    public ResponseEntity<Client> updateUser(@PathVariable Long id, @RequestBody Client client) {
        return userService.findById(id)
                .map(existingUser -> {
                    client.setId(id);
                    return ResponseEntity.ok(userService.saveOrUpdateUser(client));
                })
                .orElseGet(() -> ResponseEntity.notFound().build());
    }
    @CrossOrigin
    @DeleteMapping("/{id}")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
        return userService.findById(id)
                .map(user -> {
                    userService.deleteUser(id);
                    return ResponseEntity.ok().<Void>build();
                })
                .orElseGet(() -> ResponseEntity.notFound().build());
    }

    @CrossOrigin
    @GetMapping("/search")
    public ResponseEntity<List<Client>> searchUsers(@RequestParam(required = false) String
username, @RequestParam(required = false) String email) {
        if (username != null) {
            return ResponseEntity.ok(List.of(userService.findByUsername(username).orElse(null)));
        } else if (email != null) {
            return ResponseEntity.ok(List.of(userService.findByEmail(email).orElse(null)));
        } else {
            return ResponseEntity.badRequest().build();
        }
```

```
    }
}
```

Файл: EventController.java

```
package com.example.controller;
import com.example.model.Event;
import com.example.service.EventService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.util.StringUtils;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
@RestController
@RequestMapping("/api/events")
public class EventController {
    private final EventService eventService;
    @Autowired
    public EventController(EventService eventService) {
        this.eventService = eventService;
    }
    @CrossOrigin
    @GetMapping("/get_all")
    public List<Event> getAllEvents() {
        return eventService.findAll();
    }
    @CrossOrigin
    @GetMapping("/{id}")
    public ResponseEntity<Event> getEventById(@PathVariable Long id) {
        return eventService.findById(id)
                .map(ResponseEntity::ok)
```

```
                .orElseGet(() -> ResponseEntity.notFound().build());
    }
    @CrossOrigin
    @PostMapping("/create")
    public ResponseEntity<Event> createEvent(@RequestParam("name") String name,
@RequestParam("description") String description,
@RequestParam("dateTime") LocalDateTime dateTime,
@RequestParam("location") String location,
                                             @RequestParam("images") List<MultipartFile>
images, @RequestParam("price") double price
    ) {
        try {
            List<String> imagePaths = new ArrayList<>();
            for (MultipartFile image : images) {
                String fileName = UUID.randomUUID().toString() + "_" +
StringUtils.cleanPath(image.getOriginalFilename());
                Path imagePath = Paths.get("images").resolve(fileName);
                Files.copy(image.getInputStream(), imagePath);
                imagePaths.add(fileName);
            }
            String concatenatedPaths = String.join(",", imagePaths);
            Event newEvent = new Event(name, description, dateTime, location,
concatenatedPaths, price);
            Event createdEvent = eventService.saveOrUpdateEvent(newEvent);
            return ResponseEntity.ok(createdEvent);
        } catch (IOException e) {
            e.printStackTrace();
            return ResponseEntity.status(500).build();
        }
    }
    @CrossOrigin
    @PutMapping("/{id}")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public ResponseEntity<Event> updateEvent(@PathVariable Long id, @RequestBody Event event)
{
        return eventService.findById(id)
                .map(existingEvent -> {
                    event.setId(id);
```

```
                return ResponseEntity.ok(eventService.saveOrUpdateEvent(event));
            })
            .orElseGet(() -> ResponseEntity.notFound().build());
    }

    @CrossOrigin
    @DeleteMapping("/{id}")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public ResponseEntity<Void> deleteEvent(@PathVariable Long id) {
        return eventService.findById(id)
            .map(event -> {
                eventService.deleteEventById(id);
                return ResponseEntity.ok().<Void>build();
            })
            .orElseGet(() -> ResponseEntity.notFound().build());
    }
    @CrossOrigin
    @GetMapping("/search")
    public List<Event> searchEventsByName(@RequestParam String name) {
        return eventService.findEventsByNameContaining(name);
    }
}
```

Файл: ImageController.java

```
package com.example.controller;
import org.springframework.core.io.Resource;
import org.springframework.core.io.UrlResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.io.IOException;
import java.net.MalformedURLException;
import java.nio.file.Path;
import java.nio.file.Paths;
@RestController
@RequestMapping("/api/images")
public class ImageController {
    @CrossOrigin
```

```
    @GetMapping("/{imageName:.+}")
    public ResponseEntity<Resource> getImage(@PathVariable String imageName) {
        try {
            Path imagePath = Paths.get("images").resolve(imageName);
            Resource resource = new UrlResource(imagePath.toUri());
            if (resource.exists() && resource.isReadable()) {
                return ResponseEntity.ok()
                        .header(HttpHeaders.CONTENT_TYPE, MediaType.IMAGE_JPEG_VALUE)
                        .body(resource);
            } else {
                return ResponseEntity.notFound().build();
            }
        } catch (IOException e) {
            return ResponseEntity.notFound().build();
        }
    }
}
```

Файл: BookingDAO.java

```
package com.example.dao;
import com.example.model.Booking;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface BookingDAO extends JpaRepository<Booking, Long> {
    List<Booking> findByUserId(Long userId);
    List<Booking> findByEventId(Long eventId);
}
```

Файл: ClientDAO.java

```
package com.example.dao;
import com.example.model.Client;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;
@Repository
```

```
public interface ClientDAO extends JpaRepository<Client , Long> {
    Optional<Client> findByEmail(String email);
    Optional<Client> findByUsername(String username);
}
```

Файл: EventDAO.java

```
package com.example.dao;
import com.example.model.Event;
import com.example.repository.EventRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import java.util.List;
import java.util.Optional;
@Repository
public class EventDAO {
    private final EventRepository eventRepository;
    @Autowired
    public EventDAO(EventRepository eventRepository) {
        this.eventRepository = eventRepository;
    }
    public List<Event> findByNameContaining(String name) {
        return eventRepository.findByNameContaining(name);
    }
    public List<Event> findAllEvents() {
        return eventRepository.findAll();
    }
    public Optional<Event> findById(Long id) {
        return eventRepository.findById(id);
    }
    public Event save(Event event) {
        return eventRepository.save(event);
    }

    public void deleteEventById(Long id) {
        eventRepository.deleteById(id);
    }
```

```
}
```

Файл: PaymentsDAO.java

```
package com.example.dao;
import com.example.model.Booking;
import com.example.model.Payments;
import com.example.repository.PaymentsRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public class PaymentsDAO {
    private final PaymentsRepository paymentsRepository;
    @Autowired
    public PaymentsDAO(PaymentsRepository paymentsRepository) {
        this.paymentsRepository = paymentsRepository;
    }
    public List<Payments> findByBookingId(Long bookingId) {
        return paymentsRepository.findByBookingId(bookingId);
    }
    public List<Payments> findAllPayments() {
        return paymentsRepository.findAll();
    }
    public Payments save(Payments payment) {
        return paymentsRepository.save(payment);
    }
}
```

Файл: Booking.java

```
package com.example.model;
import jakarta.persistence.*;
import org.springframework.data.relational.core.mapping.Table;
import java.time.LocalDateTime;
@Entity
@Table(name = "bookings")
public class Booking {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private Client user;
    @ManyToOne
    @JoinColumn(name = "event_id", referencedColumnName = "id")
    private Event event;
    @Column(nullable = false)
    private LocalDateTime bookingTime;
    @Column(nullable = false)
    private LocalDateTime bookingTimeStart;
    @Column(nullable = false)
    private LocalDateTime bookingTimeEnd;
    @Column(nullable = false)
    private String status;
    public Booking() {
    }
    public Booking(Client user, Event event, LocalDateTime bookingTime, LocalDateTime
bookingTimeStart, LocalDateTime bookingTimeEnd, String status) {
        this.user = user;
        this.event = event;
        this.bookingTime = bookingTime;
        this.bookingTimeStart = bookingTimeStart;
        this.bookingTimeEnd = bookingTimeEnd;
        this.status = status;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Client getUser() {
        return user;
    }
    public void setUser(Client user) {
        this.user = user;
    }
    public Event getEvent() {
```

```
        return event;
    }
    public void setEvent(Event event) {
        this.event = event;
    }
    public LocalDateTime getBookingTime() {
        return bookingTime;
    }
    public void setBookingTime(LocalDateTime bookingTime) {
        this.bookingTime = bookingTime;
    }
    public LocalDateTime getBookingTimeStart() {
        return bookingTimeStart;
    }
    public void setBookingTimeStart(LocalDateTime bookingTimeStart) {
        this.bookingTimeStart = bookingTimeStart;
    }
    public LocalDateTime getBookingTimeEnd() {
        return bookingTimeEnd;
    }
    public void setBookingTimeEnd(LocalDateTime bookingTimeEnd) {
        this.bookingTimeEnd = bookingTimeEnd;
    }
    public String getStatus() {
        return status;
    }
    public void setStatus(String status) {
        this.status = status;
    }
}
```

Файл: Client.java

```
package com.example.model;
import jakarta.persistence.*;
import org.springframework.data.relational.core.mapping.Table;
import java.util.HashSet;
import java.util.Set;
@Entity
@Table(name = "client")
```

```
public class Client {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, unique = true)
    private String username;
    @Column(nullable = false, unique = true)
    private String email;
    @Column(nullable = false)
    private String password;
    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(name = "user_roles", joinColumns = @JoinColumn(name = "user_id"))
    @Column(name = "role")
    private Set<String> roles = new HashSet<>();
    public Client() {
    }
    public Client(String username, String email, String password) {
        this.username = username;
        this.email = email;
        this.password = password;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
```

```
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public Set<String> getRoles() {
        return roles;
    }
    public void setRoles(Set<String> roles) {
        this.roles = roles;
    }
}
```

Файл: Event.java

```
        package com.example.model;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
importorg.springframework.data.relational.core.mapping.Table;
import java.time.LocalDateTime;
@Entity
@Table(name = "events")
        public class Event {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String name;
    @Column
    private String description;
    @Column(nullable = false)
    private LocalDateTime dateTime;
    @Column(nullable = false)
    private String location;
    @Column
```

```
    private String imagePath;
    @Column(nullable = false)
    private double price;
    public Event() {
    }
    public Event(String name, String description, LocalDateTime dateTime, String location, String
imagePath, double price) {
        this.name = name;
        this.description = description;
        this.dateTime = dateTime;
        this.location = location;
        this.imagePath = imagePath;
        this.price = price;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;}

    public LocalDateTime getDateTime() {
        return dateTime;
    }
    public void setDateTime(LocalDateTime dateTime) {
        this.dateTime = dateTime;
    }
```

```
    public String getLocation() {
        return location;
    }
    public void setLocation(double latitude) {
        this.location = location;
    }
    public String getImagePath() {
        return imagePath;
    }
    public void setImagePath(String imagePath) {
        this.imagePath = imagePath;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
}
```

Файл: Payments.java

```
package com.example.model;
import jakarta.persistence.*;
@Entity
@Table(name = "payments")
public class Payments {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "booking_id", referencedColumnName = "id")
    private Booking booking;

    @Column(nullable = false)
    private String status;
    @Column(nullable = false)
    private double amount;
    public Payments() {
    }
```

```
    public Payments(Booking booking, String status, double amount) {
        this.booking = booking;
        this.status = status;
        this.amount = amount;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }

    public Booking getBooking() {
        return booking;
    }
    public void setBooking(Booking booking) {
        this.booking = booking;
    }
    public String getStatus() {
        return status;
    }
    public void setStatus(String status) {
        this.status = status;
    }
    public double getAmount() {
        return amount;
    }
    public void setAmount(double amount) {
        this.amount = amount;
    }
}
```

Файл: BookingDTO.java

```
package com.example.model.DTO;
import com.example.model.Booking;
public class BookingDTO {
    private Booking booking;
    private Double paymentsSum;
    private Double totalPrice;
```

```
    public BookingDTO(Booking booking, Double paymentsSum, Double totalPrice) {
        this.booking = booking;
        this.paymentsSum = paymentsSum;
        this.totalPrice = totalPrice;
    }
    public Booking getBooking() {
        return booking;
    }
    public Double getPaymentsSum() {
        return paymentsSum;
    }
    public Double getTotalPrice() {
        return totalPrice;
    }
    public void setBooking(Booking booking) {
        this.booking = booking;
    }
    public void setPaymentsSum(Double paymentsSum) {
        this.paymentsSum = paymentsSum;
    }
    public void setTotalPrice(Double totalPrice) {
        this.totalPrice = totalPrice;
    }
}
```

Файл: ClientRepository.java

```
package com.example.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import com.example.model.Client;
public interface ClientRepository extends JpaRepository<Client, Long> {

    @Query("SELECT c FROM Client c WHERE c.email = ?1")
    Client findByEmail(String email);
    @Query("SELECT c FROM Client c WHERE c.username = :username")
    Client findByUsername(String username);
}
```

Файл: EventRepository.java

```
package com.example.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import com.example.model.Event;
import java.util.List;
@Repository
public interface EventRepository extends JpaRepository<Event, Long> {
    @Query("SELECT e FROM Event e WHERE e.name LIKE %:name%")
    List<Event> findByNameContaining(String name);
    @Query("SELECT e FROM Event e WHERE e.description LIKE %:description%")
    List<Event> findByDescriptionContaining(String description);
}
```

Файл: PaymentsRepository.java

```
package com.example.repository;
import com.example.model.Booking;
import com.example.model.Payments;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface PaymentsRepository extends JpaRepository<Payments, Long> {
    @Query("SELECT p FROM Payments p WHERE p.booking.id = :bookingId")
    List<Payments> findByBookingId(Long bookingId);

    @Query("SELECT SUM(p.amount) FROM Payments p WHERE p.booking.id = :bookingId")
    Double sumPaymentsByBookingId(@Param("bookingId") Long bookingId);

}
```

Файл: BookingService.java

```
package com.example.service;
import com.example.dao.BookingDAO;
import com.example.model.Booking;
import com.example.model.DTO.BookingDTO;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
@Service
public class BookingService {
    private final BookingDAO bookingDAO;
    private final PaymentsService paymentsService;
    @Autowired
    public BookingService(BookingDAO bookingDAO, PaymentsService paymentsService) {
        this.bookingDAO = bookingDAO;
        this.paymentsService = paymentsService;
    }
    public Booking saveOrUpdateBooking(Booking booking) {
        return bookingDAO.save(booking);
    }
    public Optional<Booking> findById(Long id) {
        return bookingDAO.findById(id);
    }
    public List<Booking> findAll() {
        return bookingDAO.findAll();
    }
    public void deleteBooking(Long id) {
        bookingDAO.deleteById(id);
    }
    public List<Booking> findByUserId(Long userId) {
        return bookingDAO.findByUserId(userId);
    }
    public List<Booking> findByEventId(Long eventId) {
        return bookingDAO.findByEventId(eventId);
    }
    public List<BookingDTO> findBookingsWithPaymentsSumByUserId(Long userId) {
        List<Booking> bookings = bookingDAO.findByUserId(userId);
        List<BookingDTO> bookingDTOs = new ArrayList<>();
        for (Booking booking : bookings) {
            Double paymentsSum = paymentsService.findSumByBookingId(booking.getId());
            Long days = ChronoUnit.DAYS.between(booking.getBookingTimeStart(),
```

```
booking.getBookingTimeEnd());
                Double totalPrice = booking.getEvent().getPrice() * days;
bookingDTOs.add(new BookingDTO(booking, paymentsSum, totalPrice));
            }
            return bookingDTOs;
        }
}
```

Файл: ClientService.java

```
        package com.example.service;
import com.example.dao.ClientDAO;
import com.example.model.Client;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class ClientService {

    private final ClientDAO ClientDAO;
    @Autowired
    public ClientService(ClientDAO ClientDAO) {
        this.ClientDAO = ClientDAO;
    }
    public Client saveOrUpdateUser(Client client) {

        return ClientDAO.save(client);
    }
    public Optional<Client> findById(Long id) {
        return ClientDAO.findById(id);
    }
    public List<Client> findAll() {
        return ClientDAO.findAll();
    }
    public void deleteUser(Long id) {
        ClientDAO.deleteById(id);
    }
    public Optional<Client> findByEmail(String email) {
        return ClientDAO.findByEmail(email);
```

```
    }
    public Optional<Client> findByUsername(String username) {
        return ClientDAO.findByUsername(username);
    }
}
```

Файл: EventService.java

```
package com.example.service;
import com.example.dao.EventDAO;
import com.example.model.Event;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class EventService {

    private final EventDAO eventDAO;
    @Autowired
    public EventService(EventDAO eventDAO) {
        this.eventDAO = eventDAO;
    }
    public Event saveOrUpdateEvent(Event event) {
        return eventDAO.save(event);
    }
    public Optional<Event> findById(Long id) {
        return eventDAO.findById(id);
    }
    public List<Event> findAll() {
        return eventDAO.findAllEvents();
    }
    public void deleteEventById(Long id) {
        eventDAO.deleteEventById(id);
    }
    public List<Event> findEventsByNameContaining(String name) {
        return eventDAO.findByNameContaining(name);
    }
```

```
}
```

Файл: PaymentsService.java

```
package com.example.service;
import com.example.dao.PaymentsDAO;
import com.example.model.Payments;
import com.example.repository.PaymentsRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class PaymentsService {
    private final PaymentsDAO paymentsDAO;
    private final PaymentsRepository paymentsRepository;
    @Autowired
    public PaymentsService(PaymentsDAO paymentsDAO, PaymentsRepository paymentsRepository)
{
        this.paymentsDAO = paymentsDAO;
        this.paymentsRepository = paymentsRepository;
    }
    public List<Payments> findByBookingId(Long bookingId) {
        return paymentsDAO.findByBookingId(bookingId);
    }
    public Double findSumByBookingId(Long bookingId) {
        return paymentsRepository.sumPaymentsByBookingId(bookingId);
    }
    public Payments save(Payments payments) {
        return paymentsDAO.save(payments);
    }
}
```

Файл: ClientDetailsServiceImpl.java

```
package com.example.service.Impl;
import com.example.model.Client;
import com.example.dao.ClientDAO;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import java.util.Collection;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;
@Service
public class ClientDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private ClientDAO userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
        Optional<Client> optionalClient = userRepository.findByUsername(username);
        Client client = optionalClient.orElseThrow(() -> new UsernameNotFoundException("User not
found"));
        return new org.springframework.security.core.userdetails.User(client.getUsername(),
client.getPassword(), getAuthorities(client.getRoles()));
    }
    private Collection<? extends GrantedAuthority> getAuthorities(Set<String> roles) {
        return roles.stream().map(SimpleGrantedAuthority::new).collect(Collectors.toList());
    }
}
```

Файл: CustomUserDetailsService.java

```
package com.example.service.Impl;
import com.example.dao.ClientDAO;
import com.example.model.Client;
import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Primary;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
```

```
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.Collection;
import java.util.stream.Collectors;
@Service
@Primary
public class CustomUserDetailsService implements UserDetailsService {
	private final ClientDAO clientDAO;
	private PasswordEncoder passwordEncoder;
	@Autowired
	private ApplicationContext context;
	@PostConstruct
	private void init() {
		this.passwordEncoder = context.getBean(PasswordEncoder.class);
	}
	@Autowired
	public CustomUserDetailsService(ClientDAO clientDAO) {
		this.clientDAO = clientDAO;
	}
	@Override
	public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
		Client client = clientDAO.findByUsername(username)
			.orElseThrow(() -> new UsernameNotFoundException("Пользователь не найден:
" + username));
		Collection<GrantedAuthority> authorities = client.getRoles().stream()
			.map(role -> new SimpleGrantedAuthority("ROLE_" + role))
			.collect(Collectors.toList());

		return User.builder()
			.username(client.getUsername())
			.password(client.getPassword())                    .authorities(authorities)
				.build();
	}
}
```

Приложение Б.2 – Листинг веб-клиента на React

Файл: App.js

```
import React from 'react';
import EventsPage from './EventsPage';
function App() {
  return (
      <div className="App">
       <EventsPage />
      </div>
  );
}
export default App;
```

Файл: App.test.js

```
import { render, screen } from '@testing-library/react';
import App from './App';
test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

Файл: AuthModal.js

```
import React, { useState } from 'react';
import './Modal.css';
const AuthModal = ({ onClose, onLogin }) => {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const [errorMessage, setErrorMessage] = useState('');
    const [showTooltip, setShowTooltip] = useState(false);
    const handleLogin = async () => {
        setShowTooltip(false);
        setErrorMessage('');
        if (!username || !password) {
            setErrorMessage('    ');
            setShowTooltip(true);
            return;
        }
        const encodedCredentials = btoa(`${username}:${password}`);
        try {
```

```
            const response = await
    fetch('http://localhost:3127/auth/login', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Basic ${encodedCredentials}`,
                },
                body: JSON.stringify({ username, password }),
            });
            if (response.ok) {
                try {
                    const responseData = await response.json();
                    const [jwtResponse, userResponse] = responseData;
                    const { token, type } = jwtResponse;
                    const { id, roles } = userResponse;
                    onLogin(`${type} ${token}`, id, roles);
                    localStorage.setItem('token', token);
                    localStorage.setItem('user_id', id);
                    setShowTooltip(false);
                    onClose();
                    window.location.reload();
                } catch (jsonError) {
                    console.error('   JSON:', jsonError);
                    setErrorMessage('    ');
                    setShowTooltip(true);
                }
            } else {
                const errorText = await response.text();
                setErrorMessage(errorText || ' ');
                setShowTooltip(true);
            }
        } catch (networkError) {
            console.error(' :', networkError);
            setErrorMessage('   ');
            setShowTooltip(true);
        }
    };



    return (
```

```
        <div className="modal-overlay" onClick={onClose}>
            <div    className="modal-content-Auth"    onClick={(e)    =>
e.stopPropagation()}>
                <h1 className="h1"></h1>
                <input
                    type="text"
                    placeholder=" "
                    value={username}
                    onChange={(e) => setUsername(e.target.value)}
                />
                <input
                    type="password"
                    placeholder=""
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                />
                {showTooltip                    &&                    <div
className="tooltip">{errorMessage}</div>}
                <button onClick={handleLogin}></button>
                <button></button>
            </div>
        </div>
    );
};
export default AuthModal;
```

Файл: BookingModal.js

```
       import React, { useState } from 'react';
import './Modal.css';
const BookingModal = ({ booking, onClose }) => {
    const { user, event, bookingTime,bookingTimeStart,bookingTimeEnd,
status } = booking;

    const imagePaths = event.imagePath ? event.imagePath.split(',') :
[];
    const [currentImageIndex, setCurrentImageIndex] = useState(0);

    const handlePrevImage = () => {
        setCurrentImageIndex(prev => prev > 0 ? prev - 1 : 0);
    };
    const handleNextImage = () => {
```

```
        setCurrentImageIndex(prev => prev < imagePaths.length - 1 ?
prev + 1 : prev);
    };
    const formatDateAndTime = (isoString) => {
        const date = new Date(isoString);
        const options = { month: 'long', day: 'numeric', year:
'numeric', hour: '2-digit', minute: '2-digit' };

        return date.toLocaleString('ru-RU', options);
    };
    return (
        <div className="modal-overlay" onClick={onClose}>
            <div     className="modal-content"     onClick={(e)     =>
e.stopPropagation()}>
                <div className="modal-header">
                    <h2>  - {event.name}</h2>
                </div>
                <div className="modal-body">
                    <div className="image-container">
                        {imagePaths.length > 1 && (
                            <button     className="image-navigation-button
prev" onClick={handlePrevImage}
                                    disabled={currentImageIndex === 0}>
                                &#10094;
                            </button>
                        )}
                        <img
src={`http://localhost:3127/api/images/${imagePaths[currentImageIndex]
}`} alt="Event"/>
                        {imagePaths.length > 1 && (
                            <button     className="image-navigation-button
next" onClick={handleNextImage}
                                    disabled={currentImageIndex         ===
imagePaths.length - 1}>
                                &#10095;
                            </button>
                        )}
                    </div>
                    <div className="event-details">
                        <div className="booking-detail-row">
                            <p className="booking-detail-header">:</p>
```

```
                        <p                  className="booking-detail-
text">{user.username} ({user.email})</p>
                    </div>
                    <div className="booking-detail-row">
                        <p className="booking-detail-header"> :</p>
                        <p                  className="booking-detail-
text">{formatDateAndTime(bookingTime)}</p>
                    </div>
                    <div className="booking-detail-row">
                        <p className="booking-detail-header"> :</p>
                        <p                  className="booking-detail-
text">{formatDateAndTime(bookingTimeStart)}</p>
                    </div>
                    <div className="booking-detail-row">
                        <p className="booking-detail-header"> :</p>
                        <p                  className="booking-detail-
text">{formatDateAndTime(bookingTimeEnd)}</p>
                    </div>
                    <div className="booking-detail-row">
                        <p className="booking-detail-header">:</p>
                        <p className="booking-detail-text">
                            {status === 'CONFIRMED' ? ' ' : '  '}
                        </p>
                    </div>
                    <div className="booking-detail-row">
                        <p className="booking-detail-header"> :</p>
                        <p                  className="booking-detail-
text">{event.description}</p>
                    </div>
                    <div className="booking-detail-row">
                        <p className="booking-detail-header">:</p>
                        <p                  className="booking-detail-
text">{event.location}</p>
                    </div>

                </div>
            </div>
            <div className="booking-bottom">
                <button                     className="book-booking"
onClick={onClose}></button>
            </div>
```

```
            </div>
        </div>
    );
};
export default BookingModal;
```

Файл: bookingService.js

```
import Modal from "./Modal";
const handleBooking = async (event, onClose, selectedStartDate,
selectedEndDate, onTokenMissing) => {
   const now = new Date();
    function formatDateToISO(dateString) {
        const parts = dateString.split('.');
        const day = parts[0];
        const month = parts[1];
        const year = parts[2];
        const isoDateString = `${year}-${month}-${day}T15:00:00`;
        return isoDateString;
    }
    const user_id = localStorage.getItem('user_id');
    const bookingData = {
        user: {
            id: user_id,
        },
        event: {
            id: event.id,
        },
        bookingTime:  now.toISOString(),
        bookingTimeStart:
formatDateToISO(selectedStartDate.toLocaleDateString()),
        bookingTimeEnd:
formatDateToISO(selectedEndDate.toLocaleDateString()),
        status: "CONFIRMED",
    };
    const token = localStorage.getItem('token');
    if (!token) {
        onTokenMissing();
        return;
    }
```

```
    try {
        const response = await
fetch('http://localhost:3127/api/bookings', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${token}`
            },
            body: JSON.stringify(bookingData),
        });
        if (response.ok) {
            console.log("  ");
            onClose();
        } else if (response.status === 401 || response.status === 403)
{

            onTokenMissing();
        } else {
            console.error(" ", response.statusText);
        }
    } catch (error) {
        console.error("    ", error);
    }
};
export { handleBooking };
```

Файл: BookingTile.js

```
      import React from 'react';
import './BookingTile.css'; const formatDateAndTime = (isoString) =>
{
    const date = new Date(isoString);
    const options = { month: 'long', day: 'numeric', year: 'numeric',
hour: '2-digit', minute: '2-digit' };
    return date.toLocaleString('ru-RU', options);
};
const BookingTile = ({ bookingData, onClick }) => {
    const { booking, paymentsSum, totalPrice} = bookingData;
    const { event, bookingTime, status } = booking;
    const { name, description, dateTime, location, imagePath, price }
= event;
```

```
    const imageToShow = imagePath ? `http://localhost:3127/api/images/${imagePath.split(',')[0]}` : 'placeholder.png';
    const formattedPaymentSum = paymentsSum !== null ? `: ${paymentsSum.toFixed(2)} ${totalPrice.toFixed(2)}` : " ";
    return (
        <div className="booking-tile" onClick={() => onClick(event)}>
            <img src={imageToShow} alt="Event" className="booking-image"/>
            <p className="event-name">{name}</p>
            <p className="payment-info"> {formattedPaymentSum}</p>
            <p className="booking-time"> : {formatDateAndTime(bookingTime)}</p>
        </div>
    );
};
export default BookingTile;
```

Файл: CreateNewModal.js

```
    import React, { useState } from 'react';
import './CreateNewModal.css';
const CreateNewModal = ({ onClose, onSubmit }) => {
    const [name, setName] = useState('');
    const [description, setDescription] = useState('');
    const [location, setLocation] = useState('');
    const [price, setPrice] = useState('');
    const [images, setImages] = useState([]);
    const handleFileChange = (e) => {
        const newImages = Array.from(e.target.files).map(file => ({
            file,
            preview: URL.createObjectURL(file),
        }));
        setImages(currentImages => [...currentImages, ...newImages]);
    };
    const handleRemoveImage = (index) => {
        const newImages = images.slice();
        URL.revokeObjectURL(newImages[index].preview);
        newImages.splice(index, 1);
        setImages(newImages);
    };
    const handleSubmit = async (e) => {
```

```
        e.preventDefault();
        const formData = new FormData();
        formData.append('name', name);
        formData.append('description', description);
        formData.append('dateTime', "2024-01-10T17:00:00");
        formData.append('location', location);
        formData.append('price', price);
        images.forEach((image) => {
            formData.append('images', image.file);        });
        onSubmit(formData);
           };
    const handleDescriptionChange = (e) => {
        const text = e.target.value;
        if (text.length <= 250) {
            setDescription(text);
        }
    };
    return (
        <div className="modal-overlay-create" onClick={onClose}>
            <div   className="modal-content-create"   onClick={(e)   =>
e.stopPropagation()}>
                <form onSubmit={handleSubmit}>
                    <div className="modal-header-create">
                        <h2> </h2>
                    </div>
                    <div className="modal-body-create">
                        <div className="form-group-create">
                            <input className="input-field-create"
                                  type="text"
                                  placeholder=""
                                  value={name}
                                  onChange={(e) => setName(e.target.value)}
                                  required
                            />
                        </div>
                        <div className="form-group-create">
                            <div className="description-counter">
                                {description.length}/250
                            </div>
                            <textarea
                                className="textarea-field-create"
```

```
                            placeholder=" ( 250 )"
                            value={description}
                            onChange={handleDescriptionChange}
                            required>
                            <div                 className="description-
counter">{description.length}/250</div>
                             </textarea>

                    </div>
                    <div className="form-group-create">
                        <input className="input-field-create"
                              type="text"
                              placeholder=""
                              value={location}
                              onChange={(e)                          =>
setLocation(e.target.value)}
                              required
                        />
                    </div>
                    <div className="form-group-create">
                        <input className="input-field-create"
                              type="text"
                              placeholder="  "
                              value={price}
                              onChange={(e)                          =>
setPrice(e.target.value)}
                              required
                        />
                    </div>
                    <div className="form-group-create">
                        <input className="input-field-create"
                              type="file"
                              multiple
  onChange={handleFileChange}
                        />
                    </div>
                    <div className="image-preview-container-create">
                        {images.map((image, index) => (
                            <div key={index} className="image-preview-
create">
                                 <img                src={image.preview}
```

```
alt="preview"/>
                                <button          onClick={()          =>
handleRemoveImage(index)}></button>
                            </div>
                        ))}
                    </div>
                    <button            className="close-button-create"
onClick={onClose}></button>
                    <button           className="submit-button-create"
type="submit"></button>

                </div>
            </form>
        </div>
    </div>
  );
};
export default CreateNewModal;
```

Файл: EventsPage.js

```
    import React, { useState, useEffect } from 'react';
import EventTile from './EventTile';
import BookingTile from './BookingTile';
import Modal from './Modal';
import BookingModal from './BookingModal';
import CreateNewModal from "./CreateNewModal";
import AuthModal from './AuthModal';
import MapEvent from "./MapEvent";
import './EventsPage.css';

function EventsPage() {
  const [activeTab, setActiveTab] = useState('events');
  const [events, setEvents] = useState([]);
  const [bookings, setBookings] = useState([]);
  const [selectedEvent, setSelectedEvent] = useState(null);
  const [selectedBooking, setSelectedBooking] = useState(null);
  const     [isCreateNewModalOpen,     setIsCreateNewModalOpen]     =
useState(false);
  const [isMapModalOpen, setIsMapModalOpen] = useState(false);
  const [isAuthModalOpen, setIsAuthModalOpen] = useState(false);
  const [searchTerm, setSearchTerm] = useState('');
```

```
    const [authToken, setAuthToken] = useState('');
    useEffect(() => {
        if (activeTab === 'events') {
            if (searchTerm) {
                fetchEventsSearch(searchTerm);
            } else {
                fetchEvents();
            }
        } else if (activeTab === 'bookings') {
            fetchBookings();
        }
    }, [activeTab, searchTerm]);
    const handleCreateEvent = async (formData) => {
        const token = localStorage.getItem('token');
        try {
            const response = await
fetch('http://localhost:3127/api/events/create', {
                method: 'POST',
                body: formData,
                headers: { 'Authorization': `Bearer ${token}` }
            });
            if (response.ok) {
                console.log('  ');
            } else {
                console.error('   :', response.statusText);
            }
        } catch (error) {
            console.error('   :', error);
        }
    };

    const fetchEvents = async () => {
        const response = await
fetch('http://localhost:3127/api/events/get_all');
        const data = await response.json();
        setEvents(data);
    };
    const fetchEventsSearch = async (name) => {
        try {
            const response = await
fetch(`http://localhost:3127/api/events/search?name=${name}`,{
```

```
            });
            if (!response.ok) {
                throw      new      Error(`HTTP      error!      status:
${response.status}`);
            }
            const text = await response.text();

            if (!text) {
                console.error('   ');
                setEvents([]);
                return;
            }
            const data = JSON.parse(text);           setEvents(data);
        } catch (error) {
            console.error('   :', error);
            setEvents([]);
        }
    };
    const fetchBookings = async () => {
        const token = localStorage.getItem('token');
        const user_id = localStorage.getItem('user_id');

        if (token) {
            const              response              =              await
fetch('http://localhost:3127/api/bookings/user/'+user_id, {
                headers: { 'Authorization': `Bearer ${token}` }
            });
            if (response.ok) {
                const data = await response.json();
                setBookings(data);
            } else {
                setIsAuthModalOpen(true);
            }
        } else {
            setIsAuthModalOpen(true);
        }
    };
    const handleSearchChange = (event) => {
        setSearchTerm(event.target.value);
    };
    const clearSearch = () => {
```

```
        setSearchTerm('');
        fetchEvents();
    };
    const handleTileClick = (item) => {
        if (activeTab === 'events') {
            setSelectedEvent(item);
            setSelectedBooking(null);
        } else if (activeTab === 'bookings') {
            setSelectedBooking(item);
            setSelectedEvent(null);
        }
    };
    const selectEventFromMap = (event) => {
        setSelectedEvent(event);
        setIsMapModalOpen(false);
    };
    const closeModal = () => {
        setSelectedEvent(null);
        setSelectedBooking(null);
        setIsCreateNewModalOpen(false);
        setIsMapModalOpen(false);
    };
    const handleProfileClick = () => {
        setIsAuthModalOpen(true);
    };
    const handleLogin = (token) => {
        setAuthToken(token);
    };
    return (
        <div>
            <header>
                <h1 className="h1"> </h1>

                <nav className="nav-panel">
                    <ul className="nav-list">
                        <li>
                            <div   className="nav-btn"   onClick={()   =>
setActiveTab('events')}></div>
                        </li>
                        <li>
                            <div   className="nav-btn"   onClick={()   =>
```

```
setActiveTab('bookings')}></div>
                    </li>
                    <li>
                        <div   className="nav-btn"   onClick={()   =>
setIsCreateNewModalOpen(true)}></div>
                    </li>
                    <li>
                        <div   className="nav-btn"   onClick={()   =>
setIsMapModalOpen(true)}></div>
                    </li>
                    <li className="search-container">
                        <input
                            className="search-input"
                            type="text"
                            placeholder=" ..."
                            value={searchTerm}
                            onChange={(e)                              =>
setSearchTerm(e.target.value)}
                        />
                        {searchTerm && (
                            <button            className="clear-search"
onClick={() => setSearchTerm('')}
                                    aria-label=" ">
                                &times;
                            </button>
                        )}
                    </li>
                </ul>
                <div                          className="profile-icon"
onClick={handleProfileClick}></div>
            </nav>
        </header>
        {activeTab === 'events' && (
            <div className="events">
                <div className="event-list">
                    {events.map(event => (
                        <EventTile    key={event.id}    event={event}
onClick={() => setSelectedEvent(event)}/>
                    ))}
                </div>
            </div>
```

```
            )}
            {activeTab === 'bookings' && (
                <div className="bookings">
                    <div className="booking-list">
                        {bookings.map(bookingData => (
                            <BookingTile      key={bookingData.booking.id}
bookingData={bookingData}                onClick={()                =>
setSelectedBooking(bookingData.booking)} />
                        ))}

                    </div>
                </div>
            )}
            {isCreateNewModalOpen  &&  <CreateNewModal  onClose={()  =>
setIsCreateNewModalOpen(false)} onSubmit={handleCreateEvent} />}
            {selectedEvent && <Modal event={selectedEvent} onClose={()
=> setSelectedEvent(null)} />}
            {selectedBooking && <BookingModal booking={selectedBooking}
onClose={() => setSelectedBooking(null)} />}
            {isAuthModalOpen     &&     <AuthModal     onClose={()     =>
setIsAuthModalOpen(false)} onLogin={handleLogin} />}
            {isMapModalOpen && <MapEvent events={events} onClose={() =>
setIsMapModalOpen(false)} />}
            {isMapModalOpen && (
                <MapEvent
                    events={events}
                    onClose={closeModal}
                    onSelectEvent={selectEventFromMap}
                />
            )}
        </div>
    );
}
export default EventsPage;
```

Файл: EventTile.js

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

const EventTile = ({ event, onClick }) => {
    const [image, setImage] = useState('');
    const username = 'User3';
```

```
    const password = 'securePassword3';

    useEffect(() => {
        const imagePath = event.imagePath.split(',')[0];   jpg
      fetch(`http://localhost:3127/api/images/${imagePath}`, {
        })
            .then(response => response.blob())
            .then(blob => {
                const imageURL = URL.createObjectURL(blob);
                setImage(imageURL);
            });
    }, [event.imagePath]);
    return (
        <div className="event-tile" onClick={() => onClick(event)}>
            <img src={image} alt="Event"/> {/*   image  src */}
            <p className="event-name">{event.name}</p>
            <p className="event-price">  : {event.price} </p>
        </div>
    );
};export default EventTile;
```

Файл: geocode.js

```
export async function geocodeAddress(address) {
    const apiKey = '535037f6-7535-4157-8a54-9be8d58b9aa2';
    try {
        const     response     =     await     fetch(`https://geocode-
maps.yandex.ru/1.x/?format=json&apikey=${apiKey}&geocode=${address}`);
        const data = await response.json();
        if (data.response.GeoObjectCollection.featureMember.length ===
0) {
            console.warn(` "${address}"  .`);
            return null;        }
        const                        coordinates                        =
data.response.GeoObjectCollection.featureMember[0].GeoObject.Point.po
s.split(' ').map(Number).reverse();
        return coordinates;     } catch (error) {
        console.error(`    "${address}": ${error}`);
        return null;
    }
```

```
}
export async function updateEventsWithCoordinates(events) {
    const promises = events.map(async (event) => {
        const coordinates = await geocodeAddress(event.location);
        if (!coordinates) {
            return event;
        }
        return { ...event, coordinates };
    });
    return Promise.all(promises);
}
```

Файл: MapEvent.js

```
      import React, { useEffect, useState } from 'react';
import { YMaps, Map, Placemark, SearchControl } from '@pbe/react-
yandex-maps';
import { updateEventsWithCoordinates } from './geocode';
const MapEvent = ({ events, onClose, onSelectEvent }) => {
    const [updatedEvents, setUpdatedEvents] = useState([]);
    const apiKey = '535037f6-7535-4157-8a54-9be8d58b9aa2';
    useEffect(() => {
        updateEventsWithCoordinates(events,
apiKey).then(setUpdatedEvents);
    }, [events, apiKey]);
    const mapState = {
        center: [53.195607, 45.018204],
        zoom: 10,
    };
    return (
        <div className="modal-overlay" onClick={onClose}>
            <div     className="modal-content-map"     onClick={e     =>
e.stopPropagation()}>
                <div className="modal-header">
                    <h2 className="map-header">  </h2>
                    <button                          className="map-header"
onClick={onClose}></button>
                </div>
                <div className="modal-body-map">
                    <YMaps   query={{   apikey:   apiKey,   suggest_apikey:
'05eafb71-34dc-48de-8e68-7f611f171f74' }}>
                        <Map state={mapState} width="100%" height="100%">
```

```
                        <SearchControl query={{ apikey: '05eafb71-
34dc-48de-8e68-7f611f171f74' }} options={{ float: 'right' }} />
                        {updatedEvents.map((event, index) => (
                            <Placemark
                                key={event.id}
  geometry={event.coordinates}
                                properties={{
                                   iconContent: event.name,
                                }}
                                options={{
                                    preset: 'islands#redStretchyIcon',
                                }}
                                onClick={() => onSelectEvent(event)}
                            />
                        ))}
                    </Map>
                </YMaps>
            </div>
        </div>
    </div>
  );
};
export default MapEvent;
```

Файл: Modal.js

```
    import React, {useEffect, useState} from 'react';
import './Modal.css';
import { handleBooking } from "./bookingService";
import SimpleCalendar from './SimpleCalendar';
import AuthModal from "./AuthModal";
import MapEvent from "./MapEvent";
const Modal = ({ event, onClose }) => {
   const [currentImageIndex, setCurrentImageIndex] = useState(0);
   const imagePaths = event.imagePath ? event.imagePath.split(',') :
[];
   const [showCalendar, setShowCalendar] = useState(false);
   const [selectedStartDate, setSelectedStartDate] = useState(null);
   const [selectedEndDate, setSelectedEndDate] = useState(null);
   const [isAuthModalOpen, setIsAuthModalOpen] = useState(false);
   const [isMapModalOpen, setIsMapModalOpen ] = useState(null);
```

```
    const [selectedEvent, setSelectedEvent] = useState(null);
    const                                    [attemptedToBookWithoutDates,
setAttemptedToBookWithoutDates] = useState(false);
    const handleMissingToken = () => {
        setIsAuthModalOpen(true);
    };
    const attemptBooking = () => {
        if (!selectedStartDate || !selectedEndDate) {
            setAttemptedToBookWithoutDates(true);
            return;
        }
        handleBooking(event,        onClose,        selectedStartDate,
selectedEndDate, handleMissingToken);
    };
    const selectEventFromMap = (event) => {
        setSelectedEvent(event);
        setIsMapModalOpen(false);
    };
    const handlePrevImage = () => {
        setCurrentImageIndex(prev => (prev > 0 ? prev - 1 : 0));
    };
    const handleNextImage = () => {
        setCurrentImageIndex(prev => (prev < imagePaths.length - 1 ?
prev + 1 : prev));
    };
    const formatDateAndTime = (isoString) => {
        const date = new Date(isoString);
        const  options  =  {  month:  'long',  day:  'numeric',  year:
'numeric', hour: '2-digit', minute: '2-digit' };
        return date.toLocaleString('ru-RU', options);
    };
    const closeModal = () => {
        setIsMapModalOpen(false);
    };
    return (
        <div className="modal-overlay" onClick={onClose}>
            <div      className="modal-content"      onClick={e      =>
e.stopPropagation()}>
                <div classNameName="modal-header">
                    <h2>{event.name}</h2>
                </div>
```

```
            <div className="modal-body">
                <div className="image-container">
                    {imagePaths.length > 1 && (
                        <button     className="image-navigation-button
prev" onClick={handlePrevImage}
          disabled={currentImageIndex === 0}>&#10094;</button>
                    )}
                    <img
src={`http://localhost:3127/api/images/${imagePaths[currentImageIndex]
}`} alt="Event"/>
                    {imagePaths.length > 1 && (
                        <button     className="image-navigation-button
next" onClick={handleNextImage}
                                disabled={currentImageIndex         ===
imagePaths.length - 1}>&#10095;</button>
                    )}
                </div>
                <div className="event-details">
                    <p className="event-description-title">:</p>
                    <p                                 className="event-
description">{event.description}</p>
                    <p className="event-location-title">:</p>
                    <p                                 className="event-
location">{event.location}</p>
                    {showCalendar          &&          <SimpleCalendar
onSelectDateRange={({start, end}) => {
                        setSelectedStartDate(start);
                        setSelectedEndDate(end);
                        setShowCalendar(false);
                    }}/>}
                    <div className="booking-detail-row">
                        <p className="event-location-title">  :</p>
                        <p   className="event-time-title"   onClick={()
=> setIsMapModalOpen(true)}></p>
                    </div>
                    <div className="booking-detail-row">
                        <p   className="event-time-title"   onClick={()
=> setShowCalendar(!showCalendar)}>
                            :</p>
                        {selectedStartDate && selectedEndDate ? (
                            <p  className="event-time"  onClick={()  =>
```

```
setShowCalendar(!showCalendar)}>
          {selectedStartDate.toLocaleDateString()}                    -
{selectedEndDate.toLocaleDateString()}
                            </p>
                        ) : null}
                        {/*    ,      */}
                        {attemptedToBookWithoutDates
&& !selectedStartDate && !selectedEndDate && (
                            <p className="tooltip">   </p>
                        )}
                    </div>
                    <p className="event-description">
                        {selectedStartDate && selectedEndDate ? (
        `                :        ${(selectedEndDate.getTime()        -
selectedStartDate.getTime()) / (1000 * 60 * 60 * 24) * event.price} `
                        ) : (
                            `  : ${event.price} `
                        )}
                    </p>
                </div>
            </div>
            <div className="event-bottom">
                <button className="book-event"
    F                    onClick={() => attemptBooking(event, onClose,
selectedStartDate, selectedEndDate, () => setIsAuthModalOpen(true))}>
                </button>
            </div>
          </div>
          {isAuthModalOpen    &&    <AuthModal       onClose={()    =>
setIsAuthModalOpen(false)} />}
          {isMapModalOpen    &&    (<MapEvent        events={[event]}
onClose={closeModal} onSelectEvent={selectEventFromMap} />)}
       </div>
   );
};
export default Modal;
```

Файл: reportWebVitals.js

```
      const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({   getCLS,   getFID,   getFCP,   getLCP,
```

```
getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};
export default reportWebVitals;
```

Файл: SimpleCalendar.js

```
      import React, { useState } from 'react';
import './SimpleCalendar.css';

const SimpleCalendar = ({ onSelectDateRange }) => {
    const [currentMonth, setCurrentMonth] = useState(new Date());
    const [startDate, setStartDate] = useState(null);
    const [endDate, setEndDate] = useState(null);

    const daysOfWeek = ["", "", "", "", "", "", ""];
    const daysInMonth = new Date(currentMonth.getFullYear(),
currentMonth.getMonth() + 1, 0).getDate();
    const firstDayOfMonth = new Date(currentMonth.getFullYear(),
currentMonth.getMonth(), 1).getDay();

    const isDateSelectable = (date) => {
        const today = new Date();
        today.setHours(0, 0, 0, 0);
        const nextYear = new Date(today.getFullYear() + 1,
today.getMonth(), today.getDate());
        return date >= today && date < nextYear;
    };
    const handleDayClick = (day) => {
        const selectedDate = new Date(currentMonth.getFullYear(),
currentMonth.getMonth(), day);
        if (!isDateSelectable(selectedDate)) {
            return;
        }
        if (!startDate || (startDate && endDate)) {
            setStartDate(selectedDate);
```

```
            setEndDate(null);
        } else if (startDate && !endDate) {
            setEndDate(selectedDate);
            if (selectedDate < startDate) {
                setEndDate(startDate);
                setStartDate(selectedDate);
            }
        }
    };
    const isSelectedDate = (day) => {
        const selectedDate = new Date(currentMonth.getFullYear(),
currentMonth.getMonth(), day);
        return (
            (startDate && !endDate && selectedDate.getTime() ===
startDate.getTime()) ||
            (startDate && endDate && selectedDate >= startDate &&
selectedDate <= endDate)
        );
    };
    const handleOkClick = () => {
        if (startDate && endDate) {
            onSelectDateRange({ start: startDate, end: endDate });
        }
    };

    const days = [];
    for (let i = 0; i < firstDayOfMonth; i++) {
        days.push(<div key={`empty-${i}`} className="day
empty"></div>);
    }
    for (let i = 1; i <= daysInMonth; i++) {
        const date = new Date(currentMonth.getFullYear(),
currentMonth.getMonth(), i);
        const isSelected = isSelectedDate(i);
        const isSelectable = isDateSelectable(date);
        days.push(
            <div
                key={i}
                className={`day ${isSelected ? 'selected' : ''}
${!isSelectable ? 'not-selectable' : ''}`}
                onClick={() => isSelectable && handleDayClick(i)}
```

```
            >
                {i}
            </div>
        );
    }
    const changeMonth = (offset) => {
        setCurrentMonth(new            Date(currentMonth.getFullYear(),
currentMonth.getMonth() + offset, 1));
    };

    return (
        <div className="simple-calendar">
            <div className="calendar-header">
                <button onClick={() => changeMonth(-1)}>&lt;</button>
               <span>{currentMonth.toLocaleString('default',  {  month:
'long', year: 'numeric' })}</span>
                <button onClick={() => changeMonth(1)}>&gt;</button>
            </div>
            <div className="days-of-week">
                {daysOfWeek.map(day  =>  <div  key={day}  className="day-
name">{day}</div>)}
            </div>
            <div className="days">
                {days}
            </div>
            <button                                    className="button-ok"
onClick={handleOkClick}>OK</button>
        </div>
    );
};

export default SimpleCalendar;
```

Приложение В

(Обязательное)

Система бронирования гостиничных номеров

Презентация

Пензенский государственный университет

Кафедра «Вычислительная техника»

Стартап - проект. Система бронирования гостиничных номеров

БАКАЛАВРСКАЯ РАБОТА

Выполнил: В.С. Александров, студент гр. 20ВВП1

Руководитель: М.П. Синев, к.т.н., доцент каф. ВТ

Пенза 4 июня 2024

1

Актуальность проекта

В мире распространены различные информационные платформы для размещения объявлений о гостиничных номерах от разных гостиниц. Целевая аудитория подобных агрегаторов – малые и средние гостиницы, не имеющие собственных систем бронирования.

В настоящее время импортозамещение актуально и для систем бронирования гостиниц.

Актуальность проекта возрастает в связи с прогнозируемым ростом туристических поездок россиян по стране.

2

Цель проекта -

разработать эффективную систему бронирования номеров в малых и средних гостиницах, обладающую интуитивно понятным интерфейсом.

Объект исследования - автоматизированная информационная система бронирования гостиничных номеров, использующая React для веб-клиента, Spring Framework на стороне сервера и базу данных PostgreSQL.

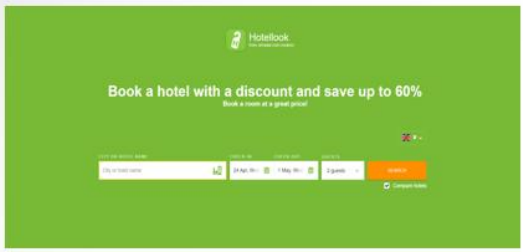
3

Постановка задачи

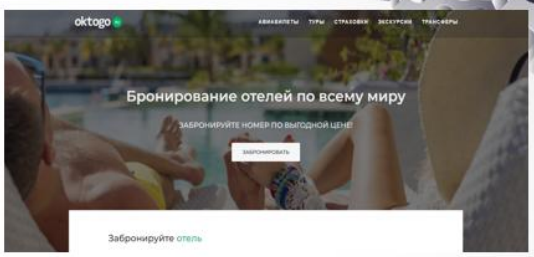
- Система должна позволять искать просматривать, бронировать номера, а так же добавлять объявления администрации гостиницы;
- Клиенская часть должна использовать React в качестве платформы веб – клиента;
- Серверная часть должна использовать в качестве платформы Spring Framework.
- Необходимо разработать бизнес-план стартап-проекта.

4

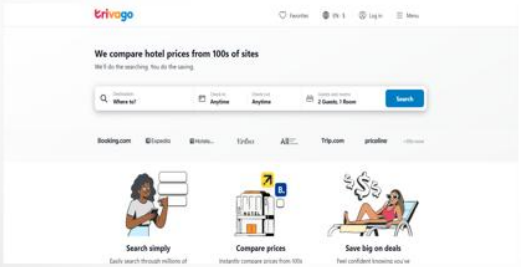
Существующие аналоги



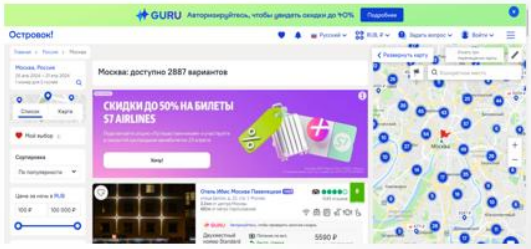
Система «Hotellook»



Система «Oktogo»



Система «Trivago»



Система «Ostrovok»

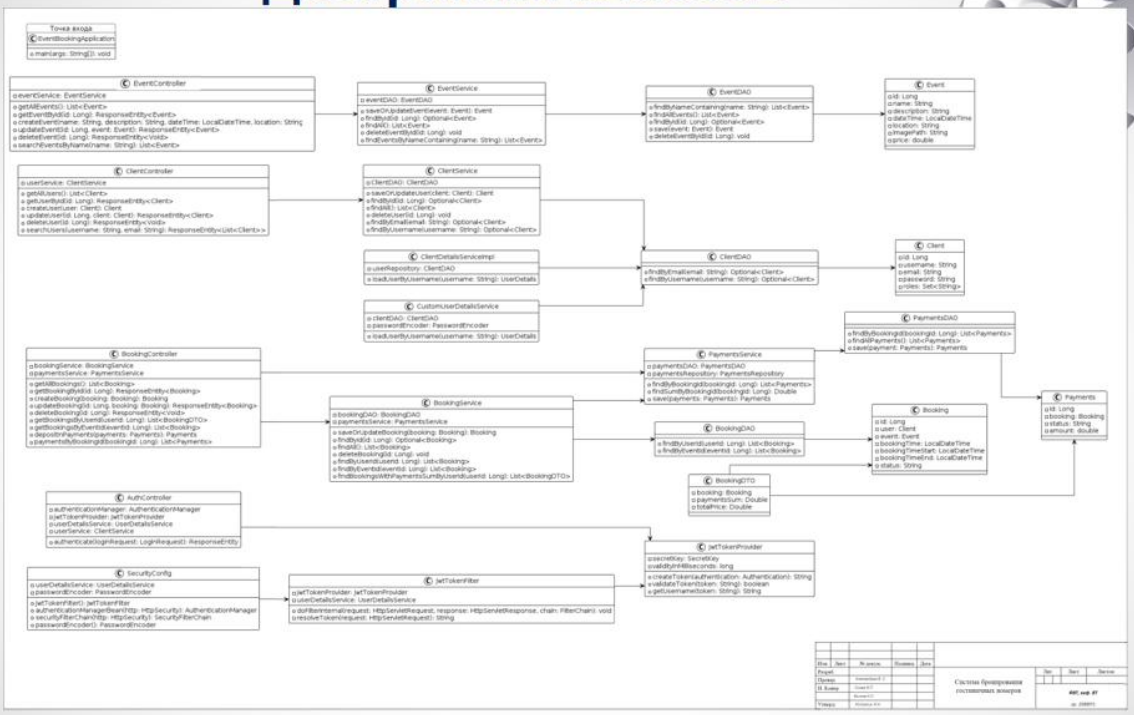
5

Стек технологий

- Java
- JavaScript
- Spring Framework (Spring Boot, Spring MVC, Spring Security, Spring Data JPA)
- PostgreSQL
- React.js
- API Яндекс Карт

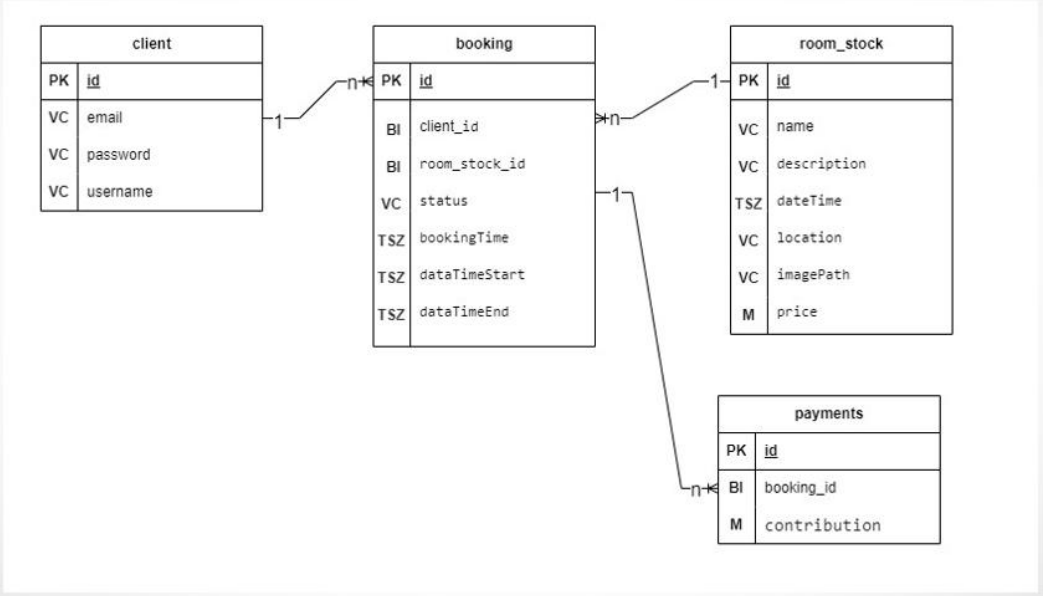
6

Диаграмма классов



Структура классов системы, показывающая основные сущности и их связи. 7

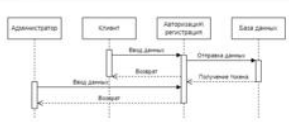
Диаграмма базы данных



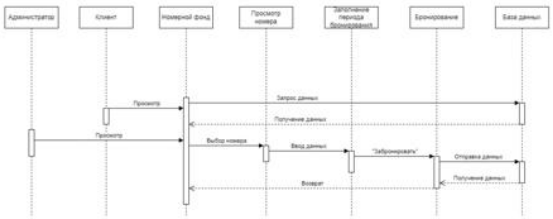
8

Диаграмма последовательности

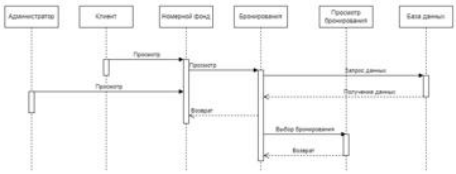
Авторизация



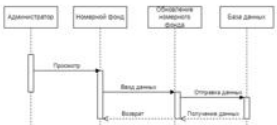
Просмотр номеров и бронирование



Просмотр бронирований



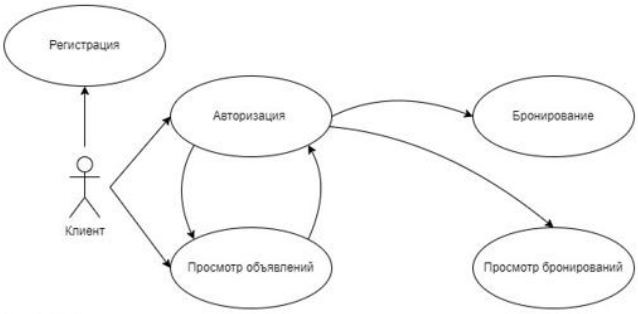
Обновление номерного фонда

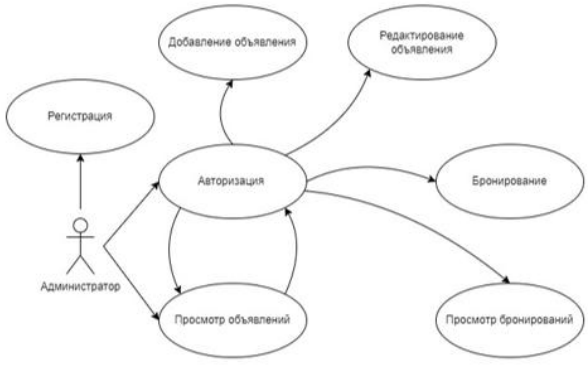


9

Диаграмма вариантов использования

Клиент



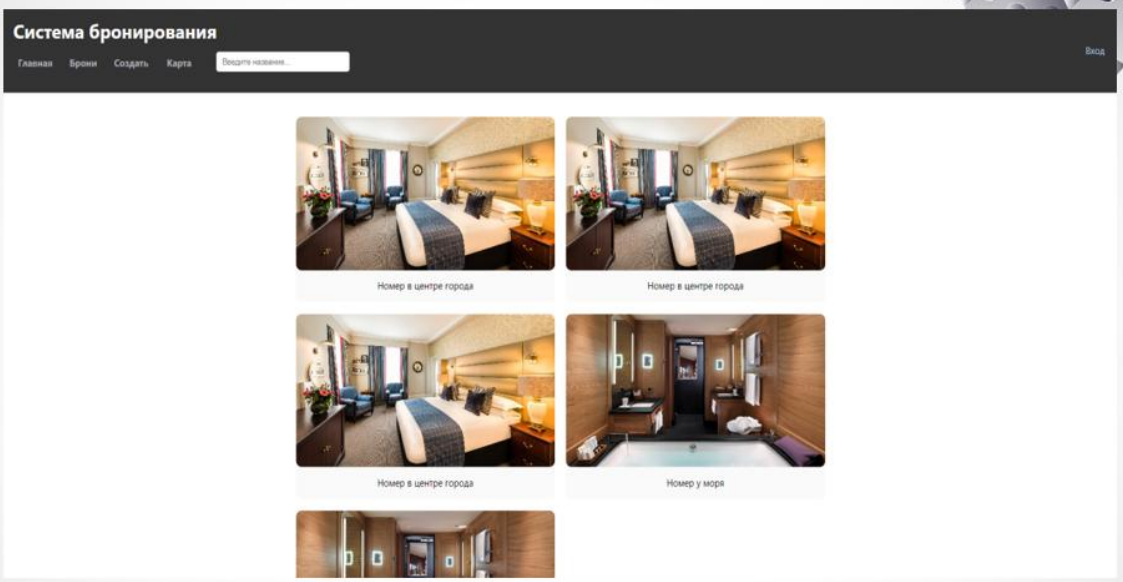


Администратор гостиницы

Основные сценарии использования системы пользователями и администраторами.

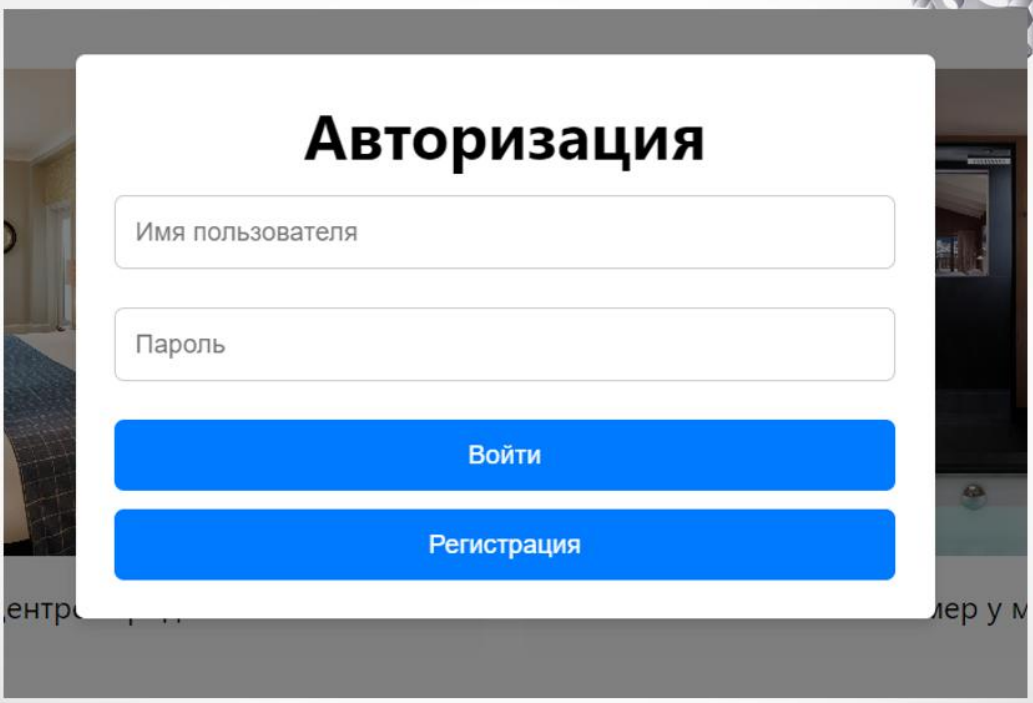
10

Главное окно системы



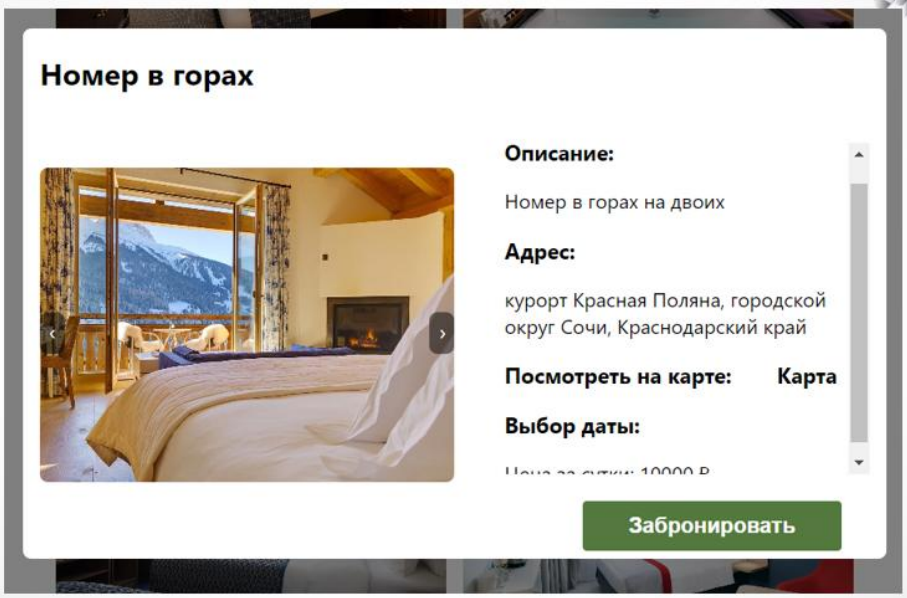
11

Окно авторизации в системе



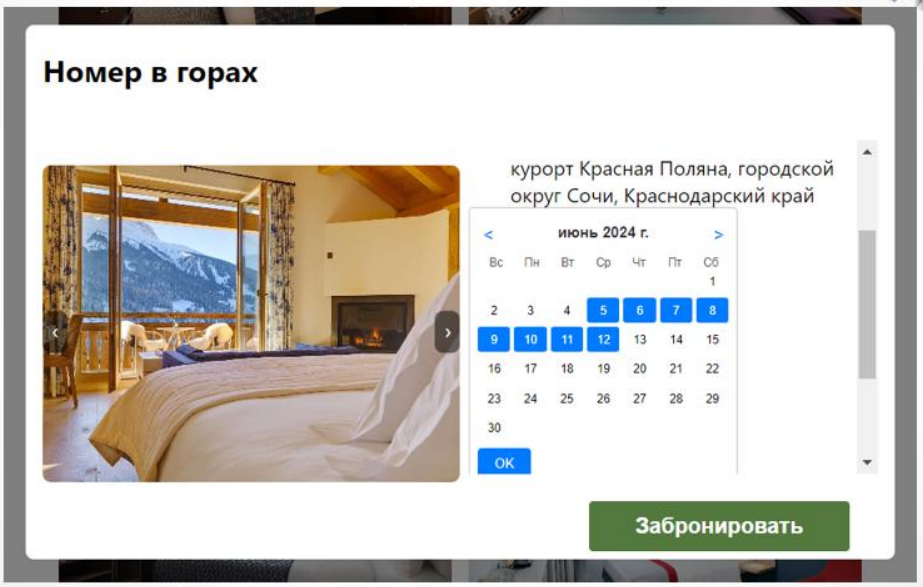
12

Окно просмотра номера



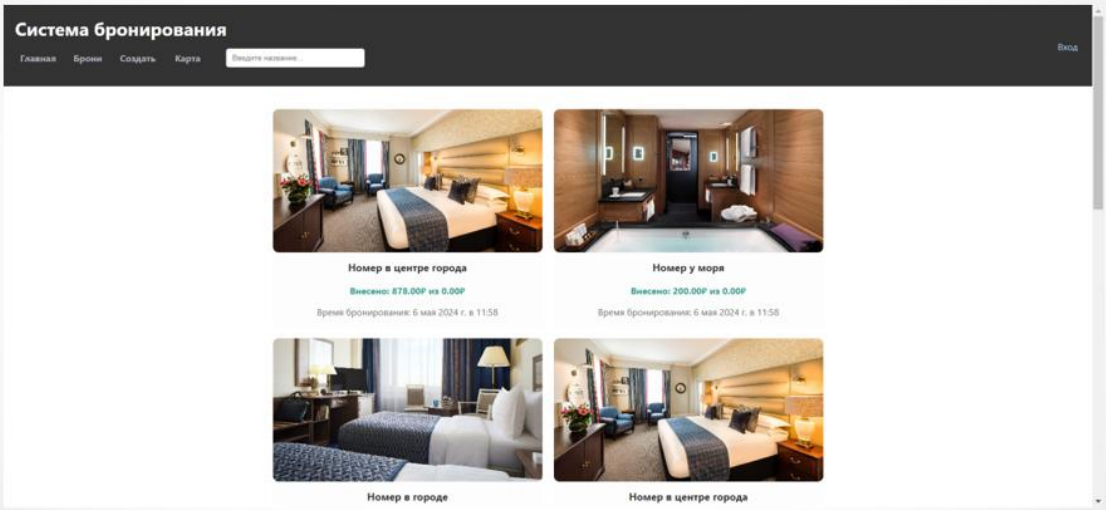
13

Календарь



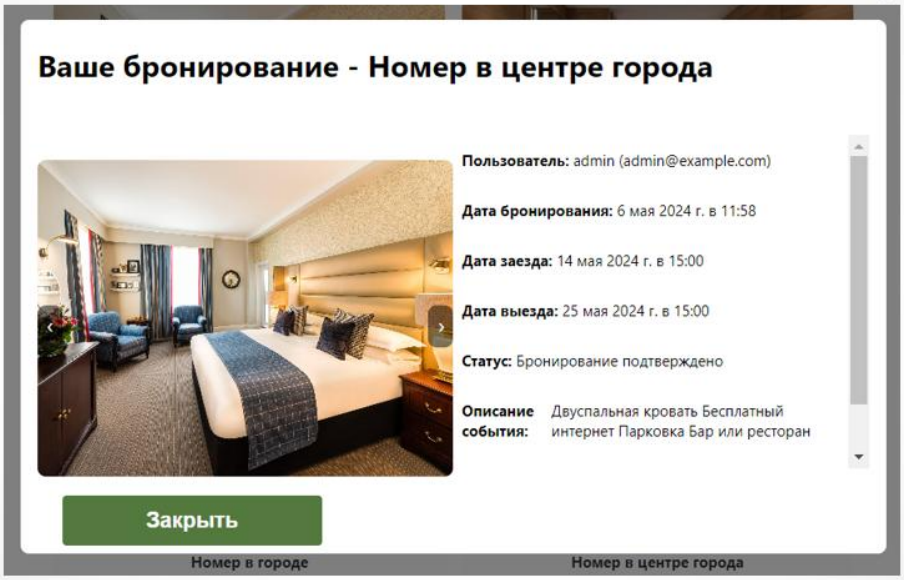
14

Раздел бронирования



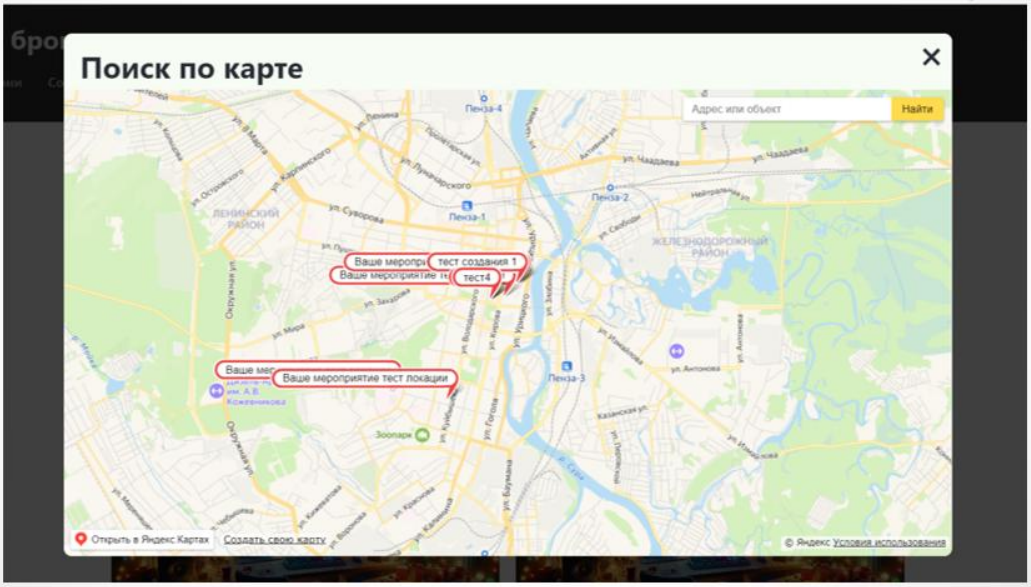
15

Окно просмотра бронирования



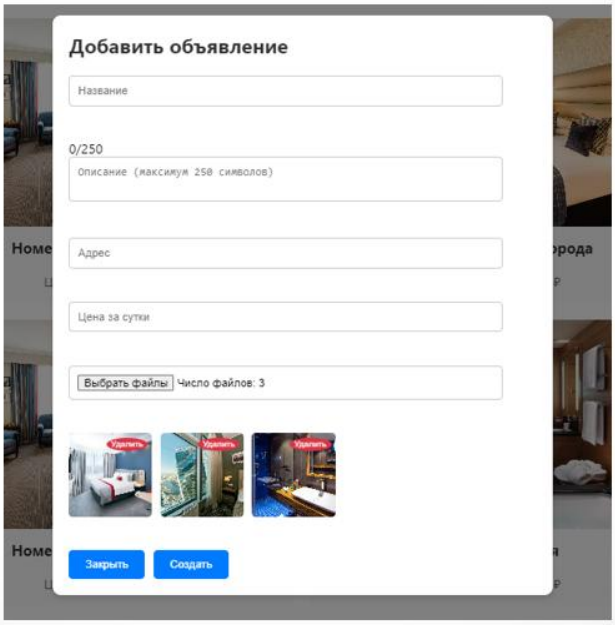
16

Местоположение на карте



17

Окно администратора



18

Заключение

В процессе работы выполнен анализ предметной области, аналитический обзор аналогов разрабатываемой автоматизированной информационной системы бронирования гостиничных номеров «Hotellook», «Trivago», «Oktogo», «Ostrovok» и др.

Разработана автоматизированная информационная система бронирования гостиничных номеров, которая обладает различными функциональными возможностями: добавление, редактирование и удаление данных из номерного фонда, а также управление бронированием и т.д.

Реализована система аутентификации и авторизации пользователей для обеспечения безопасности данных.

Разработан бизнес – план продвижения системы бронирования среди целевой аудитории.

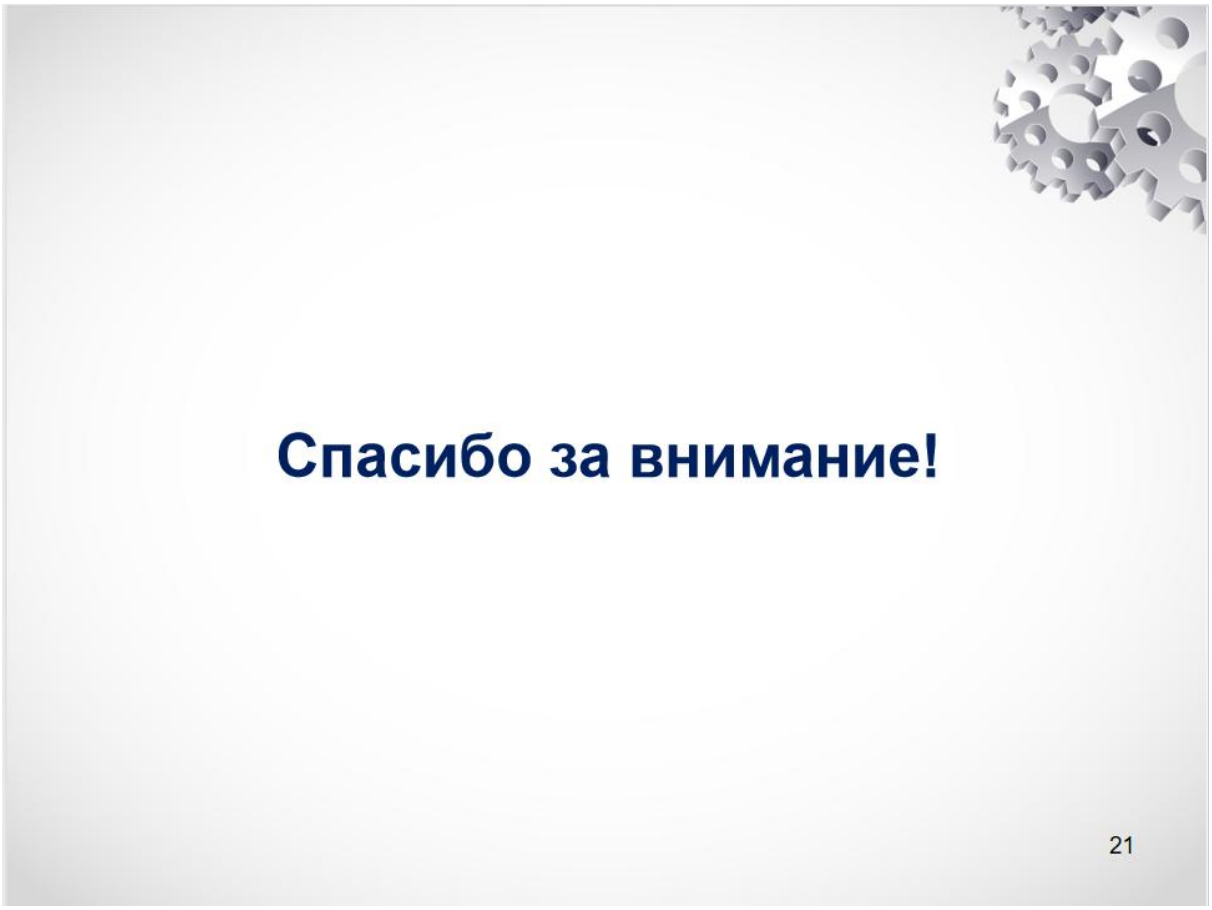
Проект разработан с использованием таких технологий, как React.js для фронтенда, Spring Framework для бэкенда и база данных PostgreSQL для хранения данных.

Планируется дальнейшее развитие системы, в частности, добавление сервиса оплаты.

19

Видео демонстрация проекта

20


Спасибо за внимание!
21

МИНОБРНАУКИ РОССИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра "Вычислительная техника"

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Митрохин М.А.
(Подпись) (Фамилия, инициалы)
«19» 04 2024 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ В ФОРМЕ СТАРТАПА

Студент Александров Владимир Сергеевич
Направление подготовки 09.03.01 Информатика и вычислительная техника.
Профиль Программное обеспечение средств вычислительной техники и автоматизированных систем
Форма обучения Очная

1. Тема: Стартап-проект. Система бронирования гостиничных номеров. «Информационная система «Система бронирования» в виде веб-приложения работающем с API реализованном на Java Spring Framework », который будет выполняться данным членом команды.

Тема одобрена экспертной комиссией и утверждена приказом ректора от «19» 04 2024 г. № 405/0

2. Срок представления работы к защите «4» 06 2024 г.

3. Общая часть ВКР:

Подготовка компьютерной презентации бизнес-проекта (стартапа).

4. Индивидуальная часть ВКР:

4.1. Исходные данные:

Разработать автоматизированную информационную систему "Система бронирования гостиничных номеров", использующую React для веб-клиента, Spring Framework на стороне сервера и базу данных PostgreSQL. Составить бизнес-модель и финансовый план. Язык программирования Java, JavaScript, PostgreSQL, среда разработки IntelliJ IDEA, WebStorm, PgAdmin. Программа должна обладать интуитивно понятным интерфейсом и использовать следующие технологии:

1. Spring Framework
2. Spring Security
3. Flyway
4. React
5. PostgreSQL

4.2. Содержание пояснительной записки ВКР (по разделам):

Анализ предметной области, актуальность предметной области, обзор аналогов, анализ аналогов, постановка задачи, выбор среды разработки, архитектура приложения, реализация приложения, руководство пользователя, БЖД, бизнес-план: финансовые и временные ограничения на проект (бюджет, сроки реализации, ожидаемые затраты и доходы и т.д.).

5. Перечень графического материала: UML-диаграммы (диаграмма классов, диаграмма развертывания, диаграмма вариантов использования, диаграмма последовательности, диаграмма архитектуры приложения)

Руководитель ВКР _____ Сиёв М.Г.
Задание принял к исполнению _____ Александров В.С.
(подпись, дата) (Ф.И.О.)

Экспертное заключение
о согласовании темы ВКР в форме стартапа

Экспертная комиссия в составе декан ФВТ д.т.н., профессор Л.Р. Фионова, доцент кафедры ВТ, к.э.н., И.В. Акифьев, эксперт по разработке ООО «Ай-теко ВЦ», к.т.н. А.Н. Токарев, рассмотрев Стартап-проект. Система бронирования гостиничных номеров, Александрова Владимира Сергеевича подтверждает его актуальность, практическую значимость и возможность коммерциализации.

Решение экспертной комиссии: выпускная квалификационная работа на тему Стартап-проект. Система бронирования гостиничных номеров, Александрова Владимира Сергеевича может выполняться в форме стартапа (бизнес-проекта).

Председатель ЭК

д.т.н., профессор
декан ФВТ — Л.Р. Фионова

Члены ЭК

к.э.н., доцент кафедры ВТ — И.В. Акифьев

к.т.н., эксперт по разработке
ООО «Ай-теко ВЦ» — А.Н. Токарев