МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Веб-приложение поиска и резервирования машино-места на автомобильной стоянке

Студент _____ Абрамов Максим Вячеславович
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И. Ю.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

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

Пенза, 2025

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

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

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент Абрамов Максим Вячеславович гр. 21ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Веб-приложение поиска и резервирования машино-места на автомобильной стоянке

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Балашова И.Ю.

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

1) Назначение: веб-приложение поиска и резервирования машино-места на автомобильной стоянке

2) Среда выполнения: Windows 11.

Языковые средства разработки: Python, JavaScript, SQL, UML.

Инструментальные средства: Visual Studio Code, Swagger, Apach Kafka, MSSQL, Visual Paradigm 17.2, Microsoft Project, PowerShell.

Технологии разработки ООП, микросервисная архитектура.

3) Основные функции приложения:

а) регистрация, авторизация и деавторизация;

б) возможность выбора автомобильной стоянки и машино-места;

в) оформление бронирования;

г) просмотр всех бронирований через личный кабинет пользователя;

д) управление парковочными местами через кабинет администратора;

е) просмотр пользовательских бронирований через кабинет администратора;

ё) визуализация результатов аналитических данных за определенный период.

5. Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и постановка задачи;*
2) *Проектирование;*
3) *Реализация;*
4) *Контроль качества программного обеспечения;*
5) *Оформление пояснительной записки.*

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 13.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 104 листа, 43 рисунка, 11 таблиц, 35 использованных источника, 7 приложений.

КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, ООП, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, MS SQL SERVER, PYTHON, KAFKA, APACHE-ZOOKEEPER

Объектом исследования – веб-приложение поиска и резервирования машино-места на автомобильной стоянке.

Цель работы – разработка прототипа веб-приложения, который может быть использован для поиска и резервирования места на автомобильной стоянке.

В качестве среды разработки использовался Visual Studio Code.

В роли СУБД был задействован MS SQL Server 20.

Разработка клиентской части выполнялась на языке программирования JavaScript. Разработка серверной части приложения производилась на языке программирования Python с применением фреймворка FastAPI.

Содержание

Введение 7
1 Анализ веб-приложения поиска и резервирования машино-места на автомобильной стоянке 8
1.1 Анализ предметной области 8
1.2 Обзор и анализ сервисов по поиску и резервированию машино-места на автомобильной стоянке 9
1.3 Сравнительный анализ 15
1.4 Постановка задачи 17
1.5 Анализ функциональных требований 18
1.6 Анализ нефункциональных требований 23
1.7 Планирование и оценка бюджета проекта 25
2 Проектирование веб-приложения поиска и резервирования машино-места на автомобильной стоянке 28
2.1 Архитектура приложения 28
2.2 Проектирование структур данных 30
2.2.1 Концептуальная модель базы данных 30
2.2.2 Логическая модель базы данных 32
2.3 Проектирование пользовательского интерфейса 33
2.4 Проектирование классов веб-приложения поиска и резервирования машино-места на автомобильной стоянке 35
3 Реализация веб-приложения поиска и резервирования машино-места на автомобильной стоянке 38
3.1 Реализация базы данных 38
3.2 Реализация серверной части приложения 38
3.3 Реализация клиентской части приложения 41
3.4 Реализация интерфейса приложения 43
4 Тестирование 55
4.1 Тестирование черным ящиком 55

4.2 Тестирование белым ящиком 58
4.3 Метрики..... 61
4.4 Интеграционное тестирование 64
Заключение 66
Список использованных источников 68
Приложение А (обязательное) Листинг серверной части приложения 72
Приложение Б (обязательное) Листинг клиентской части приложения 78
Приложение В (обязательное) Листинг интеграционных тестов 82
Приложение Г (рекомендуемое) Атрибуты сущностей базы данных 84
Приложение Д (рекомендуемое) Графическая часть 90
Приложение Е (обязательное) Результаты функционального тестирования 98
Приложение Ж (рекомендуемое) Отчет по интеграционному тестированию 103

Введение

Автомобиль уже долгое время является одним из необходимых атрибутов для человека. Это свидетельствует о важной роли автотранспорта в организации быта современного человека. В связи с этим человеку приходится чуть ли не каждый день использовать свой транспорт, чтобы добраться до нужной точки, будь то работа или любое другое место. И поэтому может возникнуть проблема с тем, где можно припарковаться на время оставления автомобиля. В большинстве городов имеются платные парковки, которые как раз решают данную проблему. Однако может вытекать и следующее неудобство: сложность с поиском и резервированием парковочного места. Ибо процесс подбора места для парковки не ограничивается нахождением самой автостоянки, также стоит учитывать ее занятость и стоимостью парковочного места на той или иной автомобильной стоянке. Поэтому целью данной работы является разрешение этой трудности, с которой могут столкнуться многие автомобилисты, которым необходимо регулярно оставлять его на время работы или другого вида занятости. Для этого будет разработано веб-приложение, которое будет учитывать все вышеперечисленные аспекты для эффективного и своевременного резервирования парковочного места.

Для достижения цели необходимо решить следующие задачи:

– провести анализ предметной области, принципов работы аналогичных систем;

– изучить существующие платные автостоянки в городе и их местонахождение;

– определить основной функционал приложения, включающий в себя поиск, бронирование и оплату парковочных мест;

– создать базу данных для хранения информации о пользователях, их бронированиях, парковках и оплатах;

– создать удобный и интуитивно понятный веб-интерфейс для пользователей, чтобы обеспечить легкость в использовании приложения.

1. Анализ веб-приложения поиска и резервирования машино-места на автомобильной стоянке

1.1 Анализ предметной области

Городская инфраструктура, как правило, создается с целью обеспечения большего количества удобств для жизни и передвижения граждан. Однако так сложилось, что чем больше город, тем сложнее у него ситуация с парковочными местами в пиковые часы.

Автомобильная стоянка представляет собой специально оборудованное место или сооружение, предназначенное для временного размещения автомобилей. Она играет важную роль в городской инфраструктуре, обеспечивая водителей возможностью безопасно оставить свой транспорт на определенный период времени. Автомобильные стоянки могут быть открытого типа, где автомобили паркуются на открытых участках без крыши или навеса, так и многоуровневого, которые, в свою очередь, строятся на нескольких этажах, что позволяет экономить пространство и увеличивать количество парковочных мест. Однако даже на многоярусных паркингах порой тяжело найти подходящее место, вследствие чего тратиться время на поиски [1,2].

Использование парковок регулируется правилами, устанавливающими, например, время работы, условия оплаты и санкции за нарушения. Для обеспечения удобства и безопасности водители взаимодействуют с инфраструктурой парковки, а также с сервисами, которые предоставляют дополнительный функционал.

Веб-приложение поиска и резервирования предлагает информацию о доступных парковках, их расположении и стоимости, что значительно упрощает процесс выбора. Пользователь может указать дату, время и продолжительность бронирования, а также оформить его через удобный и интуитивно понятный интерфейс.

1.2 Обзор и анализ сервисов по поиску и резервированию машино-места на автомобильной стоянке

Среди онлайн сервисов по нахождению и бронированию парковочного места можно выделить следующие:

«ПаркУЛЕЙ» – это онлайн-сервис для поиска и бронирования парковочных мест. Мы помогаем водителям быстро находить удобные парковки, а владельцам парковочных мест – получать дополнительный доход от сдачи их в аренду [3]. Интерфейс основных страниц данной платформы представлен на рисунках 1–3.

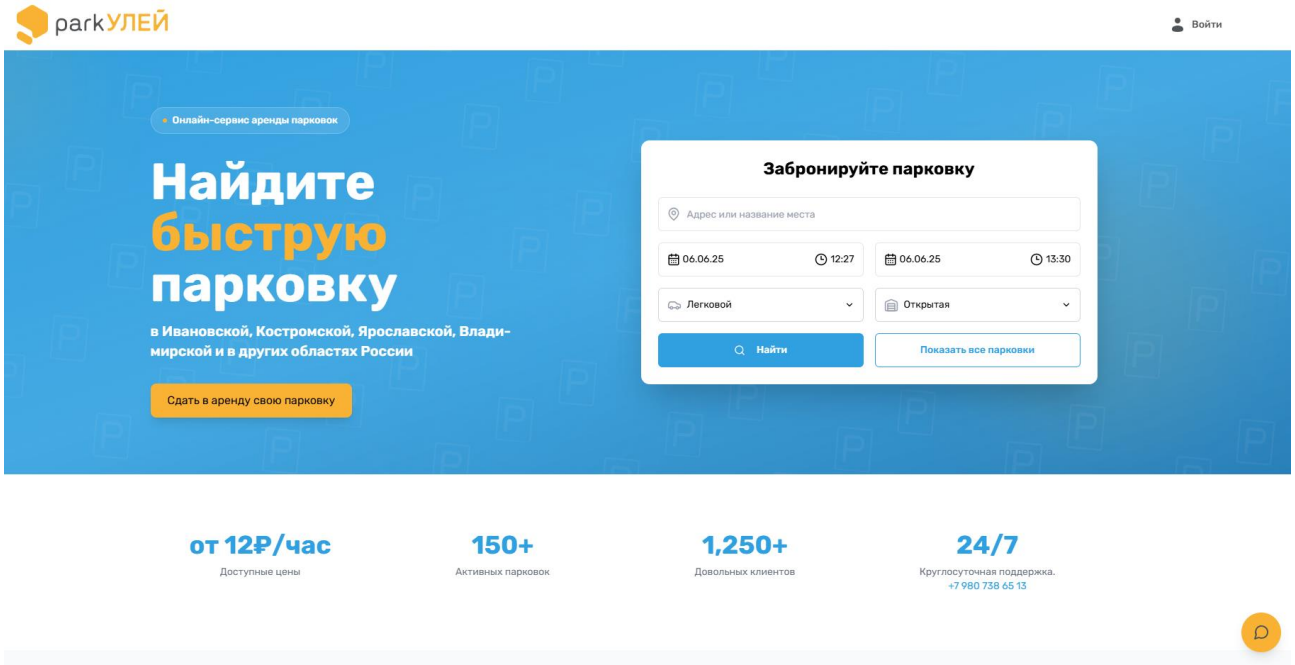

Рисунок 1 – Стартовая страница для выбора местоположения и указания промежутка времени бронирования

После выбора местоположения и предпочтительной продолжительности бронирования, происходит переход на страницу с интегрированной картой, на которой можно найти возможные автомобильные парковки со стоимостью аренды места в течение 6 часов. Помимо этого, имеется выбор типа транспортного средства и парковки. Эта страница показана на рисунке 2.

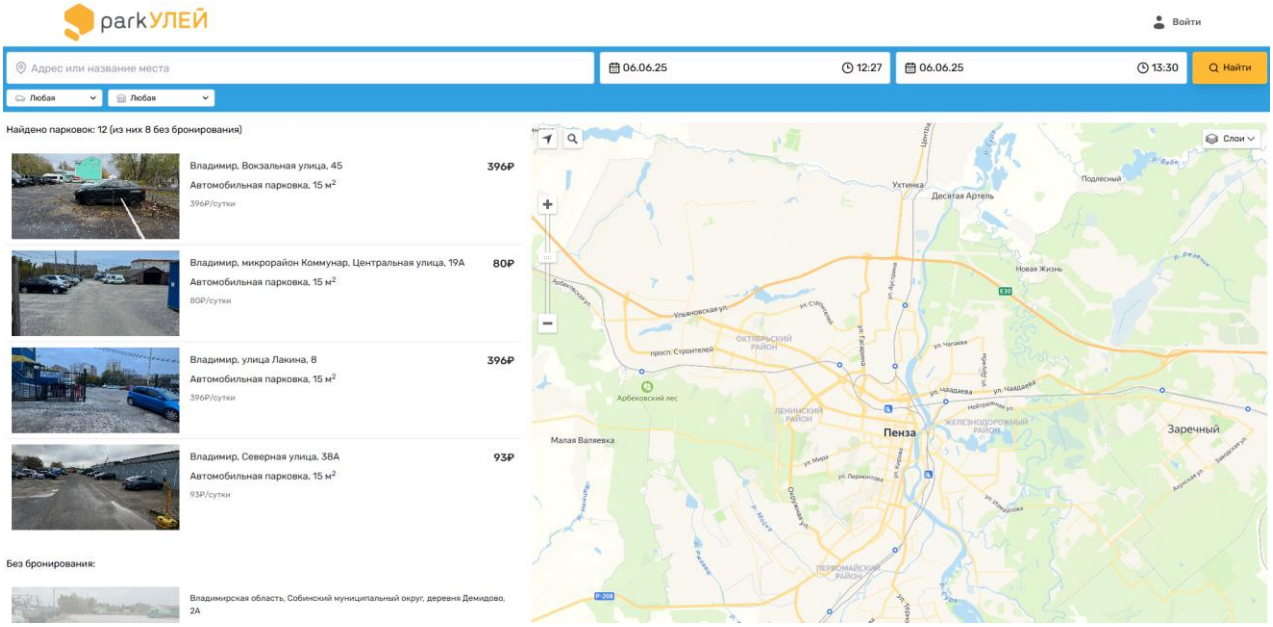

Рисунок 2 – Страница для поиска парковки

После выбора конкретной парковки открывается страница с выбором транспортного средства для оформления бронирования на указанное время. Кроме того, остается возможность выбора интересующего времени резервирования, адреса, типа транспортного средства и парковки, благодаря чему пользователь может в короткий срок найти подходящее место, воспользовавшись интегрированной картой. Данная страница представлена на рисунке 3.

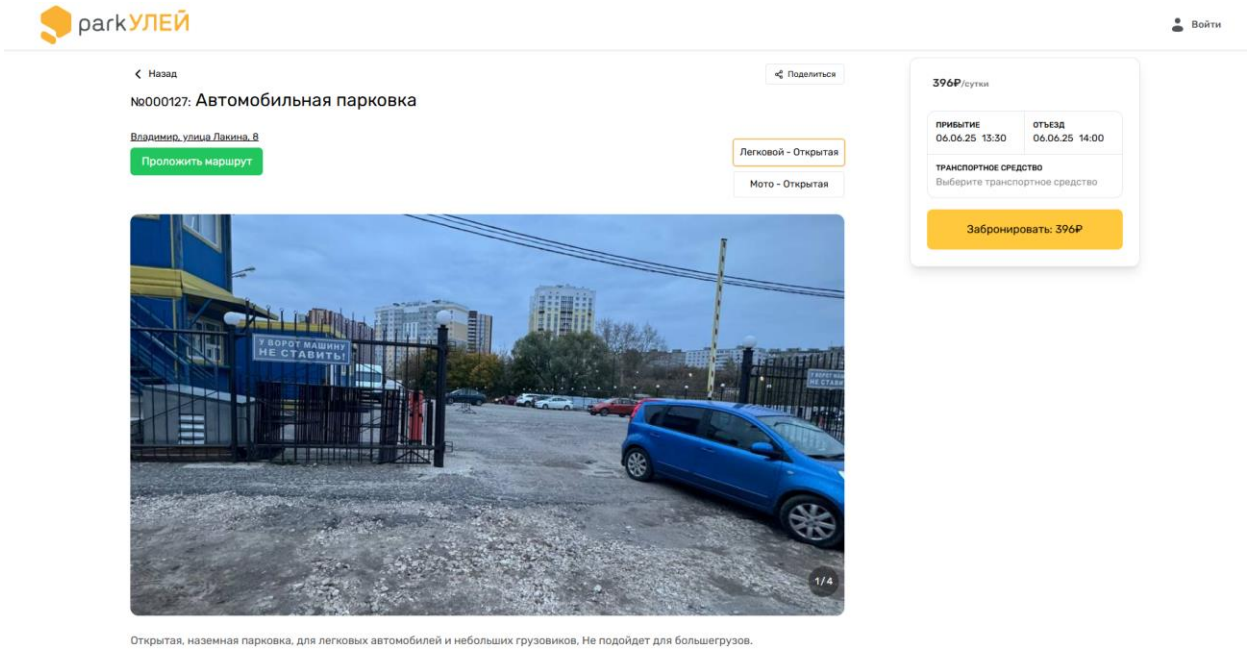

Рисунок 3 – Страница с информацией о выбранной парковке

Ключевым преимуществом является простой и быстрый процесс оформления брони. Также стоит выделить особенность, что есть возможность выбирать тип транспортного средства (легковой, мотоцикл и т д.) и автомобильной стоянки (открытая, отапливаемая и крытая).

«Паркли» – приложение для поиска, бронирования и оплаты парковок в ТЦ, аэропортах, на вокзалах и в других местах. Сервис фокусируется на сдаче мест на парковках бизнес-центров, у которых нет собственной системы для допуска на парковку кратковременных съемщиков [4]. Интерфейс основных страниц данного приложения представлен на рисунках 4–6.

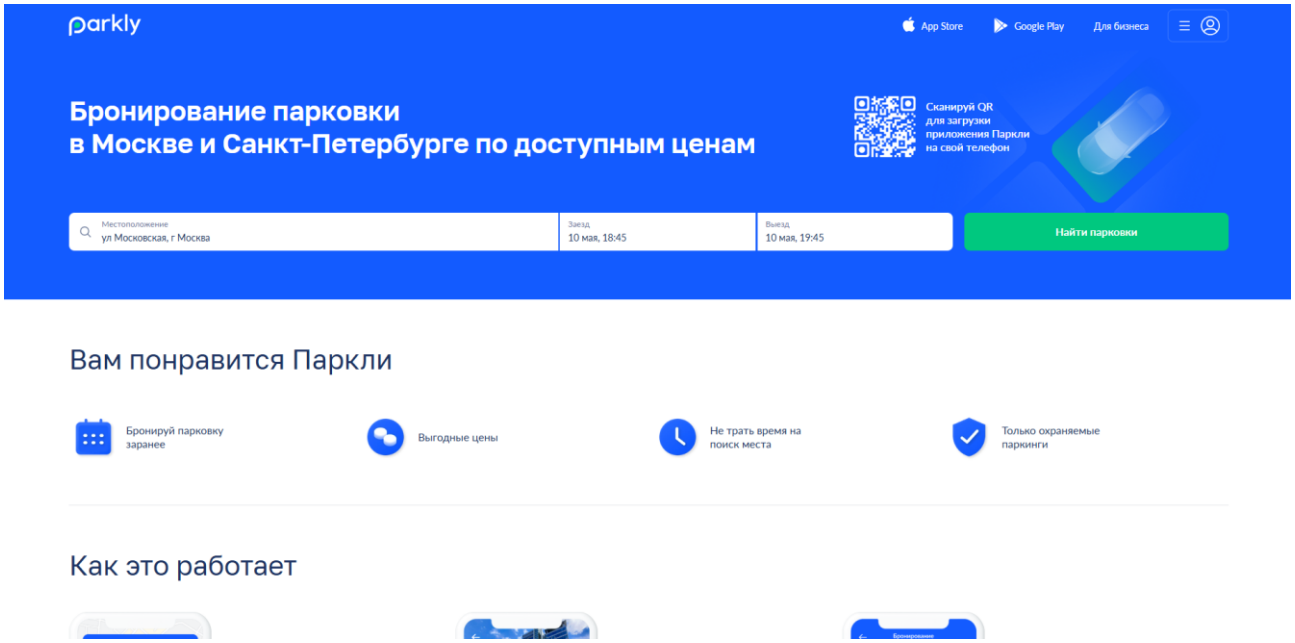

Рисунок 4 – Начальная страница для выбора места и указания промежутка времени бронирования

Также как и в первом сервисе после указания местоположения и продолжительности бронирования, нас встречает помеченная доступными парковками карта. Помимо этого, пользователю предлагается фильтр с выбором почасовым или помесячным резервированием, что помагает сразу отсеять неподходящие варианты. Но также есть возможность просмотреть все возможные парковки по указанному адресу. Данная страница изображена на рисунке 5.

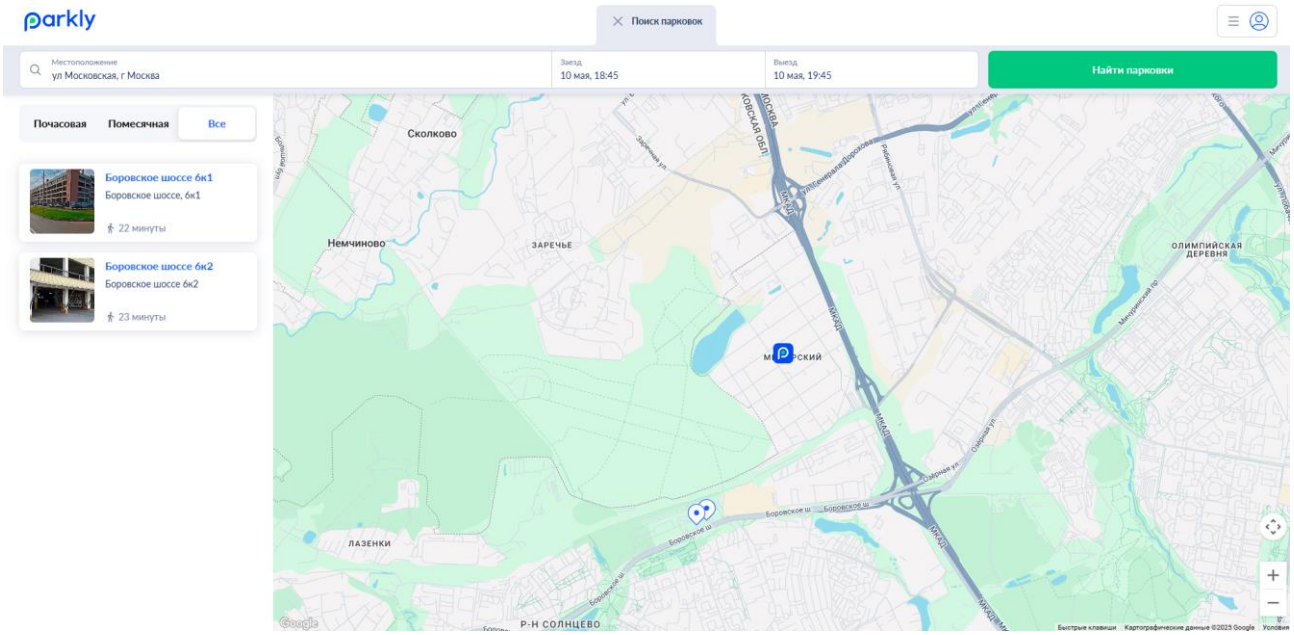

Рисунок 5 – Страница для поиска парковки по указанному адресу

Затем после выбора конкретной парковки открывается страница со стоимостями места за разные промежутки времени, описанием самого места с прикрепленными фотографиями и наличие ограничения по высоте, ширине и т п. Данная страница представлена на рисунке 6.

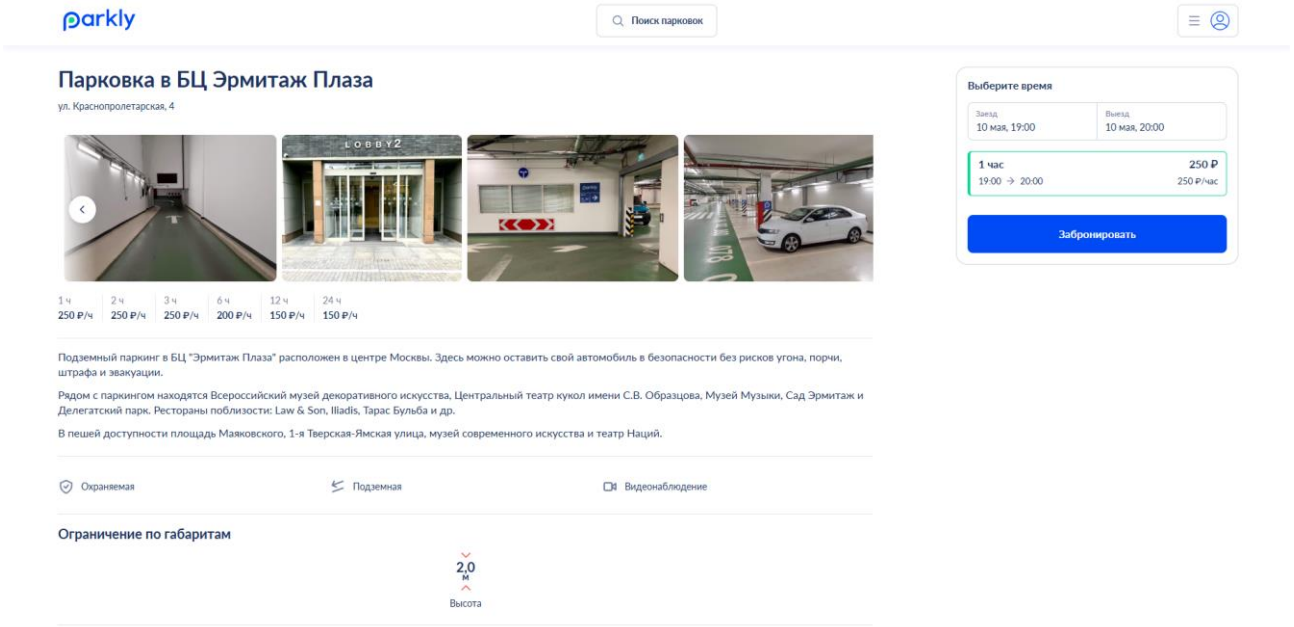

Рисунок 6 – Страница с информацией о выбранной парковке

Ключевым преимуществом «Паркли» является удобство бронирования. Приложение позволяет заблаговременно резервировать места на охраняемых

парковках, обеспеченных системой видеонаблюдения. Также стоит отметить наличие описания автомобильных стоянок, предупреждений как, например, ограничение по высоте, и фотографий для наглядного представления.

Встречаются жалобы на некорректное отображение въезда и выезда при абонементе [5]. Помимо этого, пользователи отмечают ошибки в отображении зон местных парковок [6].

«GetPark» – это цифровой сервис по аренде мест в паркингах и стоянках на территории России и СНГ [7]. Начальная страница данного сервиса изображена на рисунке 7.

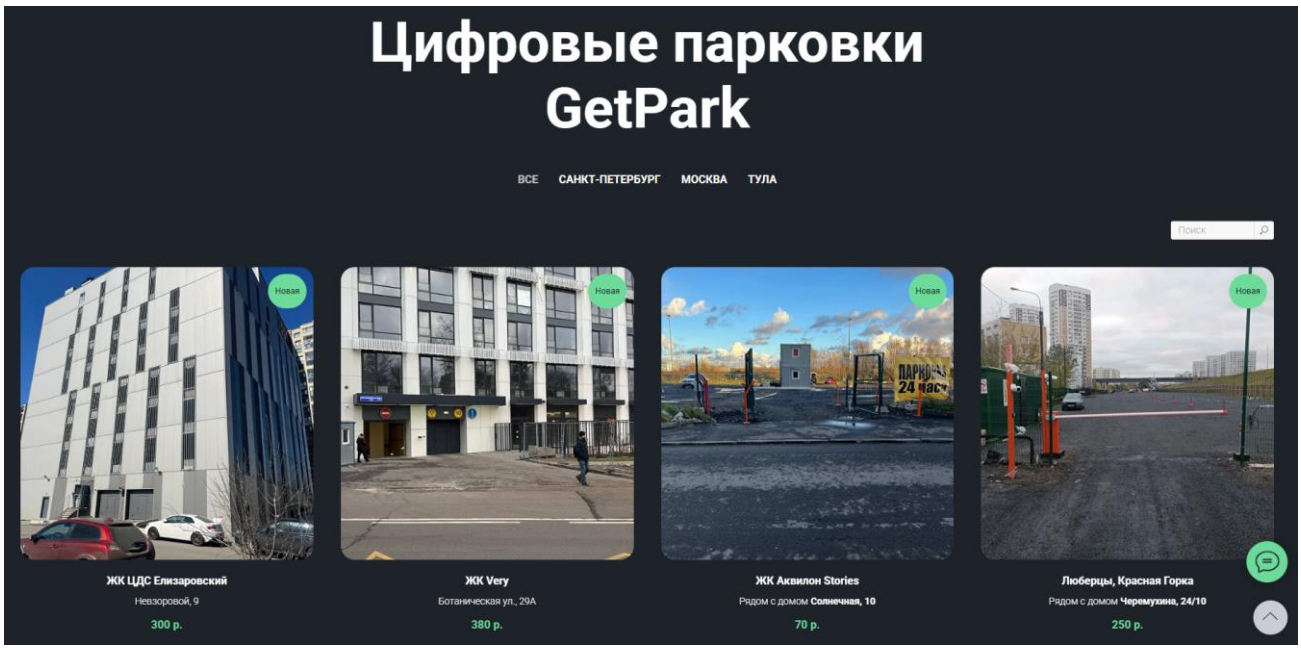

Рисунок 7 – Стартовая страница со списком всех парковок

Выбор парковки и оформление бронирование осуществляется уже в мобильном приложении. Процесс регистрации занимает немного времени, следует только указать номер телефона и ввести полученный код, после чего становится доступен весь пользовательский функционал. Страница выбора определенной парковки показана на рисунке 8. На данной странице доступна карта со всеми отмеченными автомобильными стоянками, кроме того предоставляется возможность заранее выбрать время бронирования места из предложенных вариантов.

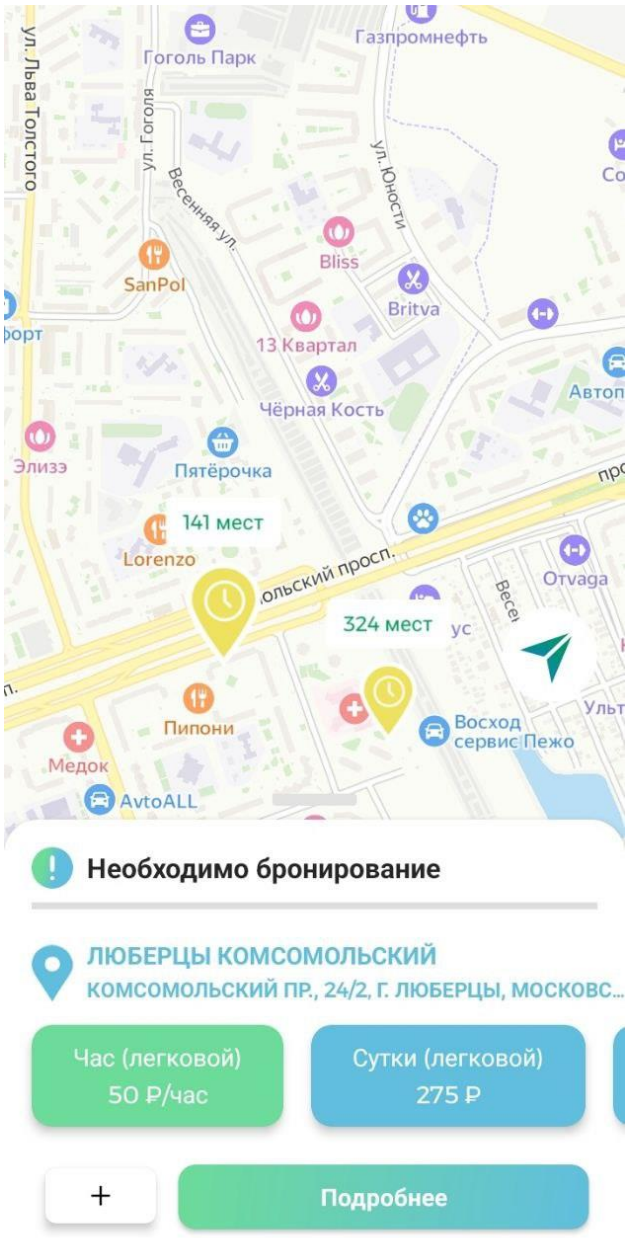

Рисунок 8 – Страница для поиска парковки по указанному адресу

Также, если перейти по кнопке «Подробнее», откроется краткая информация о выбранной парковке, которая поможет ознакомиться с ее условиями.

Процесс оформления резервирования включает в себя несколько дополнительных этапов, таких как указание номера автомобиля и добавление платежной карты. Страница с созданием бронирования представлена на рисунке 9.

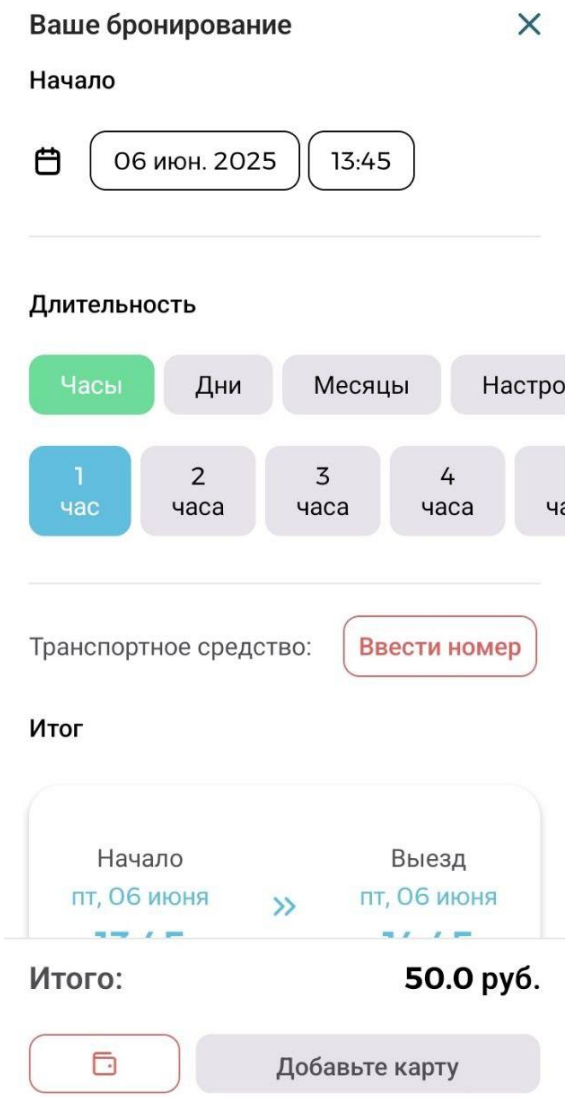

Рисунок 9 – Страница для создания бронирования

Преимуществом может послужить то, что сервис предлагает удобную систему онлайн-оплаты с бесплатной отменой бронирования до его начала. Если ваши планы поменялись, вы можете легко отменить заказ – в этом случае все деньги сразу вернутся на вашу карту без лишних неудобств.

1.3 Сравнительный анализ

Выполним сравнительный анализ рассмотренных ранее сервисов, таких как «ПаркУЛЕЙ», «Паркли» и «GetPark», по ключевым функциям.

Для проведения сравнительного анализа был определен перечень критериев, на основе которых оценивались отобранные сервисы.

Система оценивалась по следующей шкале:

– 0 баллов – функция отсутствует;

– 1 балл – функция реализована не в полной мере;

– 2 балла – функция полностью реализована.

Результаты оценки представлены в таблице 1.

Таблица 1 – Сравнение сервисов бронирования парковочных мест

№	Критерий	«ПаркУЛЕЙ»	«Паркли»	«GetPark»
1	Удобство бронирования	2	2	2
2	Понятность интерфейса	2	2	2
3	Навигация до парковки	1	0	0
4	Личный кабинет с историей бронирований	0	2	1
5	Вывод актуальной доступности мест	1	1	1
6	Простые правила отмены бронирования	1	2	2
7	Предупреждения об особенностях парковки (ограничения по высоте въезда и т п.)	0	2	0
8	Выбор типа транспортного средства и парковки	2	0	0
	Итого	9	13	10

В конечном итоге, подытоживая, справедливо сказать, что все три платформы предоставляют удобный интерфейс для поиска и бронирования

парковочных мест. Интегрированная карта позволяет быстро находить ближайшие парковочные места и оценивать расстояние до них. Помимо этого, пользователи могут просматривать список своих бронирований. Однако важно выделить, что данные сервисы имеют общие недостатки, которые связанны с системными проблемами. Один из главных – проблемы с отслеживанием доступности парковочных мест, из-за чего возникают ситуации, что свободное место остается занятым.

В результате проведенного анализа выбранных сервисов бронирования парковочного места было принято решение разработать собственное программное решение, которое устранит выявленные недостатки. Новое приложение, помимо базовой функциональности интерактивной карты с отображением доступных парковочных мест, будет включать схемы парковочных мест на автомобильных стоянках для наглядного представления расположения каждого места. Особое внимание уделено упрощению процедуры отмены бронирования – система гарантирует беспрепятственную отмену за сутки до назначенного времени при любых обстоятельствах. Для обеспечения защищенности финансовых операций в приложении будет реализована интеграция с проверенным и надежным платежным сервисом, соответствующим стандарту безопасности данных индустрии платежных карт. Дополнительно предусмотрена разработка аналитической панели для администраторов, что позволит оптимизировать работу всего сервиса.

1.4 Постановка задачи

Задачей данной работы является проектирование, разработка и тестирование программных средств прототипа приложения, которое предназначено для упрощенного и эффективного поиска и резервирования машино-мест. Кроме этого оно должно быть готовым к дальнейшим расширениям и возможным модификациям.

Предполагается, что основной функционал должен делиться на отдельные компоненты – микросервисы. Они должны быть не зависимы друг от друга и тем

самым разделять логику приложения. А она, в свою очередь, должна состоять из следующих базовых частей:

– регистрация, авторизация и деавторизация пользователей;
– осуществление поиска и резервирования парковочного места;
– реализация оплаты выбранного машино-места.

Также, помимо этого, необходимо реализовать и обеспечить администратора возможностью отслеживать и управлять местами на автомобильных стоянках, контролировать доступ к приложению, просматривать бронирования пользователей и проводить анализ, связанный с парковками, их парковочными местами и доходами, полученных после оплаты каждого зарезервированного места.

1.5 Анализ функциональных требований

В ходе изучения предметной области веб-приложения для поиска и резервирования машино-места на автостоянке были выделены следующие функциональные требования:

– регистрация, авторизация и деавторизация пользователя;
– выбор парковочного места на указанной автомобильной стоянке;
– создание бронирования;
– выполнение оплаты;
– просмотр данных созданного бронирования;
– отмена бронирования перед оплатой;
– отмена бронирования после его создания за сутки до начала действия;
– управление парковочными местами (резервирование и освобождение) через кабинет администратора;
– просмотр всех бронирований конкретного пользователя;
– внесение пользователя в черный или белый список;

– формирование диаграмм для визуализации аналитических данных по указанному периоду времени.

В соответствии с перечисленными функциональными требованиями была разработана UML-диаграмма вариантов использования, изображенная на рисунке 10.

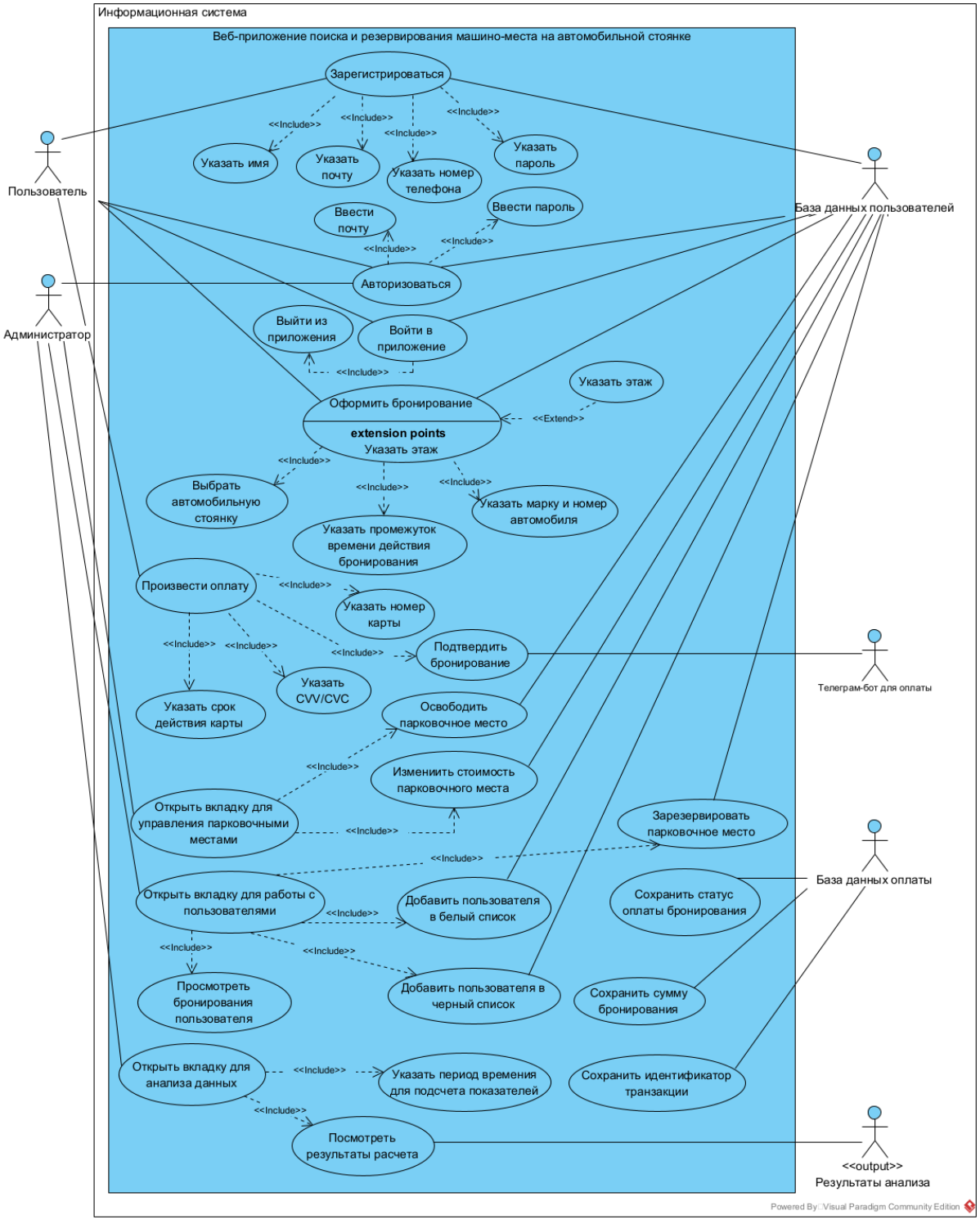

Рисунок 10 – Диаграмма вариантов использования

Ниже приведены спецификации ключевых бизнес-процессов.

Сценарий варианта использования «Зарегистрироваться» представлен на рисунке 11.

Прецедент: Зарегистрироваться
ID: 1
Описание: Регистрация пользователя в приложении
Действующие лица: 1) Пользователь; 2) База данных пользователей.
Предусловие: 1) Пользователь зашел в приложение; 2) Пользователь не зарегистрирован.
Основной поток: 1) Пользователь заходит на страницу регистрации; 2) Пользователь вводит данные для регистрации (имя, почту, номер телефона, пароль); 3) Сохранение данных нового пользователя в базу данных пользователей; 4) Пользователь, получив доступ к функционалу, попадает на основную страницу приложения.
Постусловие: 1) Пользователю доступен функционал приложения; 2) Пользователь перенаправлен на основную страницу приложения.

Рисунок 11 – Сценарий варианта использования «Зарегистрироваться»

Сценарий варианта использования «Авторизоваться» показан на рисунке 12.

Прецедент: Авторизоваться
ID: 2
Описание: Авторизация пользователя в приложении
Действующие лица: 1) Пользователь; 2) База данных пользователей.
Предусловие: 1) Пользователь зашел в приложение; 2) Пользователь не авторизован.
Основной поток: 1) Пользователь заходит на страницу авторизации; 2) Пользователь вводит данные для авторизации (почту, пароль); 3) Найти запись в базе данных пользователей с введенными данными пользователем; 4) Пользователь, получив доступ к функционалу, попадает на основную страницу приложения.
Постусловие: 1) Пользователю доступен функционал приложения; 2) Пользователь перенаправлен на основную страницу приложения.

Рисунок 12 – Сценарий варианта использования «Авторизоваться»

Сценарий варианта использования «Оформить бронирование» представлен на рисунке 13.

Прецедент: Оформить бронирование
ID: 3
Описание: Оформление бронирования через приложение
Действующие лица: 1) Пользователь; 2) База данных пользователей.
Предусловие: 1) Пользователь находиться на основной странице приложения; 2) Пользователю необходимо найти и зарезервировать парковочное место.
Основной поток: 1) Пользователь выбирает автомобильную стоянку из предложенного списка; 2) Пользователь выбирает этаж, в случае выбора многоуровневой стоянки; 3) Пользователь указывает промежуток времени, в течение которого будет действовать бронирование; 4) Пользователь указывает марку и номер автомобиля; 5) Все указанные данные сохраняются в базу данных пользователей.
Постусловие: 1) Пользователь создал бронирование; 2) В базе данных сохранились данные бронирования, которые следует пользователю подтвердить.

Рисунок 13 – Сценарий варианта использования «Оформить бронирование»

Сценарий варианта использования «Произвести оплату» изображен на рисунке 14.

Прецедент: Произвести оплату
ID: 4
Описание: Выполнение оплаты для подтверждения бронирования
Действующие лица: 1) Пользователь; 2) База данных пользователей; 3) Телеграм-бот.
Предусловие: Пользователь указал все данные для оформления бронирования
Основной поток: 1) Пользователь перенаправляется в диалог с телеграм-ботом оплаты; 2) Пользователь заполняет форму для оплаты (номер карты, cvv/cvc, срок действия карты); 3) Оплата подтверждает бронирование. 4) Данные о сумме, идентификаторе транзакции и статусе сохраняются в базу данных оплаты.
Постусловие: 1) Пользователь создал бронирование; 2) В базе данных сохранились данные бронирования, которые следует пользователю подтвердить.

Рисунок 14 – Сценарий варианта использования «Произвести оплату»

Сценарий варианта использования «Открыть вкладку для управления парковочными местами» представлен на рисунке 15.

Прецедент: Открыть вкладку для управления парковочными местами
ID: 5
Описание: Администратор во вкладке для управления парковочными местами освобождает или резервирует парковочное место и изменяет его стоимость
Действующие лица: 1) Администратор; 2) База данных пользователей;
Предусловие: Администратор авторизовался и зашел в кабинет
Основной поток: 1) Администратор открывает вкладку для управления парковочными местами; 2) При необходимости администратор освобождает зарезервированное пользователем парковочное место; 3) При необходимости администратор резервирует свободное парковочное место; 4) При необходимости администратор изменяет стоимость парковочного места.
Постусловие: 1) В случае если администратор освобождает зарезервированное пользователем парковочное место, то в базе данных меняется статус этого места; 2) В случае если администратор резервирует свободное парковочное место, то в базе данных меняется статус этого места; 3) В случае если администратор изменяет стоимость парковочного места, то в базе данных изменяется стоимость этого места;

Рисунок 15 – Сценарий варианта использования «Открыть вкладку для управления парковочными местами»

Сценарий варианта использования «Открыть вкладку для работы с пользователями» отображен на рисунке 16.

Прецедент: Открыть вкладку для работы с пользователями
ID: 6
Описание: Администратор во вкладке для работы с пользователями управляет доступом к функционала приложения и просматривает созданные пользователями бронирования
Действующие лица: 1) Администратор; 2) База данных пользователей;
Предусловие: Администратор авторизовался и зашел в кабинет
Основной поток: 1) Администратор открывает вкладку для работы с пользователями; 2) Администратор просматривает созданные пользователем бронирования; 3) При необходимости администратор добавляет пользователя в черный список; 4) При необходимости администратор добавляет пользователя в белый список.
Постусловие: 1) В случае если администратор добавляет пользователя в черный список, то в базе данных меняется статус этого пользователя; 2) В случае если администратор добавляет пользователя в белый список, то в базе данных меняется статус этого пользователя.

Рисунок 16 – Сценарий варианта использования «Открыть вкладку для работы с пользователями»

Сценарий варианта использования «Открыть вкладку для анализа данных» показан на рисунке 17.

Прецедент: Открыть вкладку для анализа данных
ID: 7
Описание: Администратор во вкладке для анализа данных просматривает и изучает различные показатели деятельности приложения за определенный период времени
Действующие лица: 1) Администратор; 2) База данных пользователей;
Предусловие: Администратор авторизовался и зашел в кабинет
Основной поток: 1) Администратор открывает вкладку для анализа данных; 2) Администратор указывает промежуток времени, в рамках которого необходимо высчитать ключевые показатели; 3) Администратор просматривает и разбирается с результатами подсчетов в виде гистограмм.
Постусловие: Результат подсчетов основных показателей представлен в виде гистограмм для наглядного восприятия

Рисунок 17 – Сценарий варианта использования «Открыть вкладку для анализа данных»

В ходе анализа функциональных требования было определено несколько первостепенных вариантов использования веб-приложения. Для отдельных вариантов использования были составлены спецификации, которые показывают четкую последовательность работы пользователя и администратора с приложением.

1.6 Анализ нефункциональных требований

Разрабатываемое веб-приложение будет состоять из трех основных компонентов: двух микросервисов и брокера сообщений. Выбор микросервисной архитектуры обусловлен тем, что ее основная концепция заключается в том, чтобы разделять сложные приложения на более мелкие и легко управляемые сервисы, что повышает масштабируемость и улучшает общую производительность системы [8]. Каждому микросервису будет принадлежать собственная база данных. Создание и управление ими будет осуществляться посредством системы управления базами данных Microsoft SQL

Server. А для составления запросов на создание и заполнение баз данных будет применен декларативный язык SQL.

Клиентская часть будет состоять из страниц, написанных на языке разметки HTML, внешний вид им будет задаваться посредством формального языка CSS и за создание динамического контента будет отвечать язык Java Script. Также такой шаблонизатор как Jinja2 будет предоставлять механизм для динамической генерации некоторых HTML-страниц на основе данных, поступающих с сервера.

Серверная же часть в виде REST API будет написана на основе фреймворка FastAPI. FastAPI – это современный и высокопроизводительный веб-фреймворк для создания API на языке программирования Python. Он разрабатывался как простой и удобный инструмент для ускорения разработки и уменьшения количества шаблонного кода [9]. Следует выделить следующие особенности этого фреймворка, которые делают его востребованным: поддержка асинхронной обработки запросов, наличие автоматической документации, что достаточно сильно упрощает процесс тестирования серверного API, и интуитивно понятная работа с HTTP-методами, такими как: GET, POST, PUT, DELETE и другими.

Немало важной деталью любого приложения является безопасность. Поэтому будет реализована аутентификация и авторизации пользователей с применением JWT (JSON Web Token). JWT – это специальный формат токена, который позволяет безопасно передавать данные между клиентом и сервером [10]. Его несомненную пользу и эффективность использования можно обосновать следующим образом:

1. Учетные данные пользователя, как правило хранятся долго (месяцы). Как бы хорошо не был зашифрован запрос, при достаточном количестве времени его можно расшифровать. Если запрос, содержащий учетные данные перехвачен злоумышленником, у него будет много времени на расшифровку. Токены доступа имеют ограниченный срок годности (обычно ~15 минут). Этого времени не достаточно, чтобы расшифровать надежный шифр. К тому времени, когда

зловредный алгоритм расшифрует запрос, токен уже выйдет из обращения и будет бесполезен.

2. Использовать учетные данные – это медленно. Для валидации учетных данных сервер должен запросить их сохраненную копию из БД и сравнить с данными, которые пришли в запросе. Обращение к БД – дорогостоящая процедура, она сильно увеличивает время обработки запроса. Токены, с другой стороны, не требуют обращения к БД для валидации. Это позволяет снизить нагрузку на БД и ускорить обработку запросов сервером [11].

Помимо этого, для обеспечения большей безопасности пароли пользователей будут хешироваться перед сохранением в базе данных. Хеширование – это технология для обеспечения безопасности и оптимизации данных. С помощью функции данные преобразовываются в уникальный код, который невозможно незаметно изменить и почти невозможно подобрать перебором [12]. Тем самым будет организовано безопасное хранения пользовательских паролей, так как даже в случае утечки данных, злоумышленник не сможет расшифровать захешированный пароль, тем самым ему не удастся получить первоначальные значения.

Взаимодействие между микросервисами, как упоминалось ранее, будет происходить посредством брокера сообщений Apache Kafka. Брокер – система, преобразующая сообщение от источника данных (продюсера) в сообщение принимающей стороны (консьюмера). Главная задача брокера – обеспечение связи и обмена информацией между приложениями или отдельными модулями в режиме реального времени [13].

1.7 Планирование и оценка бюджета проект

Бюджет проекта – это детальный план доходов и расходов, необходимых для его успешного выполнения. Он указывает, как будут распределены финансовые средства, связанные с выполнением задач, мероприятий и приобретением необходимых материалов [14].

Для создания плана по разработке проекта и представления всего жизненного цикла в виде наглядной диаграммы будет задействована программа для управления проектами Microsoft Project.

Каждый этап характеризуют задачи, представленные на рисунке 18.

Название задачи	Длительность	Начало	Окончание	Названия ресурсов
Проект разработки	**39 дней**	**Вт 08.04.25**	**Пн 02.06.25**	
Начало проекта	0 дней	Вт 08.04.25	Вт 08.04.25	
Анализ предметной области	**3,63 дней**	**Ср 09.04.25**	**Пн 14.04.25**	
Анализ готовых решений	0,63 дней	**Ср 09.04.25**	Ср 09.04.25	Компьютеры[1];Интернет;Аналитик
Постановка задачи	0,63 дней	Чт 10.04.25	Чт 10.04.25	Компьютеры[1];Интернет;Аналитик
Анализ требований	0,63 дней	Пт 11.04.25	Пт 11.04.25	Интернет;Компьютеры[1];Аналитик
Анализ функциональных требований	1 день	Сб 12.04.25	Сб 12.04.25	Интернет;Компьютеры[1];Backend-разработчик;Frontend-разработчик
Анализ нефункциональных требовани	0,63 дней	Пн 14.04.25	Пн 14.04.25	Backend-разработчик;Интернет;Компьютеры[1]
Проектирование веб-приложения	**9,25 дней**	**Вт 15.04.25**	**Пн 28.04.25**	
Выбор архитектуры приложения	0,38 дней	Вт 15.04.25	Вт 15.04.25	Интернет;Компьютеры[1];Backend-разработчик
Проектирование структур данных	3,25 дней	Ср 16.04.25	Вс 20.04.25	Интернет;Компьютеры[1];Backend-разработчик; Разработчик баз данных
Проектирование пользовательского интерфейса	4 дней	Пн 21.04.25	Чт 24.04.25	Интернет;Компьютеры[1];Frontend-разработчик;UX/UI-дизайнер
Проектирование классов веб-приложе	1,38 дней	Пт 25.04.25	Пн 28.04.25	Backend-разработчик;Интернет;Компьютеры[1]
Реализация веб-приложения	**15,88 дней**	**Вт 29.04.25**	**Вт 20.05.25**	
Реализация базы данных	2,5 дней	Вт 29.04.25	Чт 01.05.25	Интернет;Компьютеры[1];Разработчик баз данных
Реализация серверной части веб-приложения	5 дней	Сб 03.05.25	Чт 08.05.25	Интернет;Компьютеры[1];Backend-разработчик
Реализация клиентской части прилож	4 дней	Пт 09.05.25	Ср 14.05.25	Интернет;Компьютеры[1];Frontend-разработчик
Реализация интерфейса приложения	4 дней	Чт 15.05.25	Вт 20.05.25	UX/UI-дизайнер;Интернет;Компьютеры[1]
Контроль качества программного обеспечения	**8 дней**	**Ср 21.05.25**	**Вс 01.06.25**	
Выбор режима тестирования	1 день	Ср 21.05.25	Ср 21.05.25	Интернет;Компьютеры[1];QA-инженер
Тестирование клиентской части приложения	3 дней	Чт 22.05.25	Вс 25.05.25	QA-инженер;Интернет;Компьютеры[1];Frontend-разработчик
Тестирование серверной части прило	3 дней	Пн 26.05.25	Ср 28.05.25	Интернет;Компьютеры[1];Backend-разработчик;QA-инженер
Оценка качества программного кода	3 дней	Чт 29.05.25	Вс 01.06.25	QA-инженер;Интернет;Компьютеры[1]
Завершение проекта	0 дней	Пн 02.06.25	Пн 02.06.25	

Рисунок 18 – Этапы разработки веб-приложения

Для отчетливого представления этапов разработки была сформирована диаграмма Ганта. Диаграмма Ганта – это визуальное представление графика работ, построенное согласно плану проекта. На ней отражены задачи и последовательность их выполнения.

График работ состоит из ряда отрезков, размещенных вдоль временной оси. Каждый из них соответствует отдельной задаче или подзадаче. Начало и конец отрезка соответствуют моменту начала и завершения работы по задаче. Длина отрезка – продолжительность работ [15].

Данная диаграмма отображена на рисунке 19.

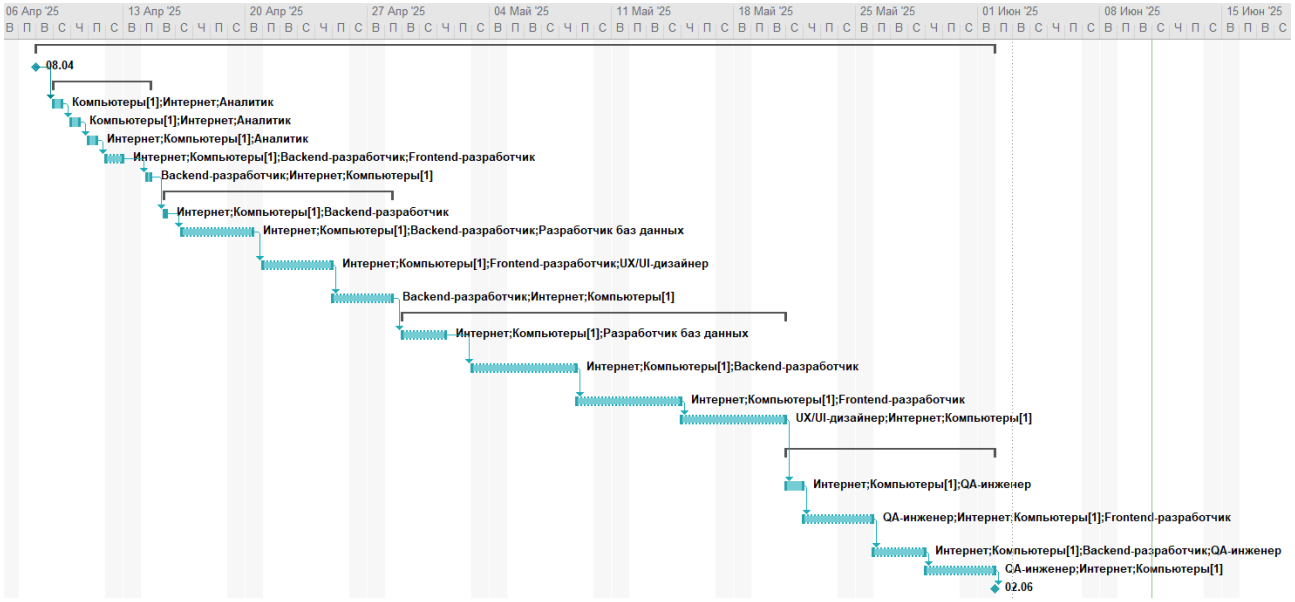

Рисунок 19 – Диаграмма Ганта с распределением ресурсов

Вся разработка делится на четыре ключевых этапов, каждый из которых включает в себя подзадачи. Для работы над ними были назначены ресурсы, которые показаны на рисунке 20.

	Название ресурса	Тип	Стандартная ставка
	Backend-разработчи	Трудовой	500,00р./час
	Frontend-разработч	Трудовой	450,00р./час
	UX/UI-дизайнер	Трудовой	450,00р./час
	QA-инженер	Трудовой	400,00р./час
	Аналитик	Трудовой	500,00р./час
	Разработчик баз данных	Трудовой	490,00р./час
	Компьютеры	Материальный	0,00р.
	Интернет	Затраты	

Рисунок 20 – Перечень ресурсов с почасовой ставкой

Общая длительность разработки проекта и суммарный объем затрат составляет 39 дней и 439120 рублей, соответственно. Тем самым, MS Project позволил создать детализированный план разработки проекта, включая задачи, сроки и ресурсы. Помимо этого, в нем легко устанавливать зависимости между задачами, чтобы видеть, как изменение одной задачи влияет на другие. Это помогает избежать задержек и обеспечивает более точное планирование [16].

2 Проектирование веб-приложения поиска и резервирования машино-места на автомобильной стоянке

2.1 Архитектура приложения

Для веб-приложения был выбран такой архитектурный шаблон проектирования как MVC (Model-View-Controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения [17].

Модель отвечает за хранение, определение структуры базы данных и обрабатывает все операции, связанные с данными. Она представляет собой совокупность ORM (Object-Relational Mapping), которые необходимы для привидения таблиц из базы данных к классам, с которыми появится возможность взаимодействовать непосредственно через программный код.

Контроллер же содержит в себе логику обработки пользовательских обращений, получаемых посредством роутера, который, в свою очередь, маршрутизирует поступающие запросы в соответствующие методы. Главной задачей данного компонента является организация связи между моделью (model) и представлением (view). Каждый метод контроллера принимает свой HTTP запрос (GET, POST или другие) и формирует результат в соответствии с той логикой, которая в нем реализована. Это может быть отображение HTML страницы, обработка полученных данных с последующим сохранением в базу данных или же наоборот извлечение данных с целью отправки и занесения их в динамический шаблон.

Третий компонент – это представление, которое определяет внешний вид приложения и обеспечивает пользовательское взаимодействие, путем представления данных, которые были получены из модели. В приложении оно реализовано в виде статических и, в частности, динамических HTML страниц, которые заполняются данными, переданными контроллером.

На рисунке 21 представлена диаграмма развертывания, которая отображает архитектуру веб-приложения поиска и резервирования машино-место на автомобильной стоянке.

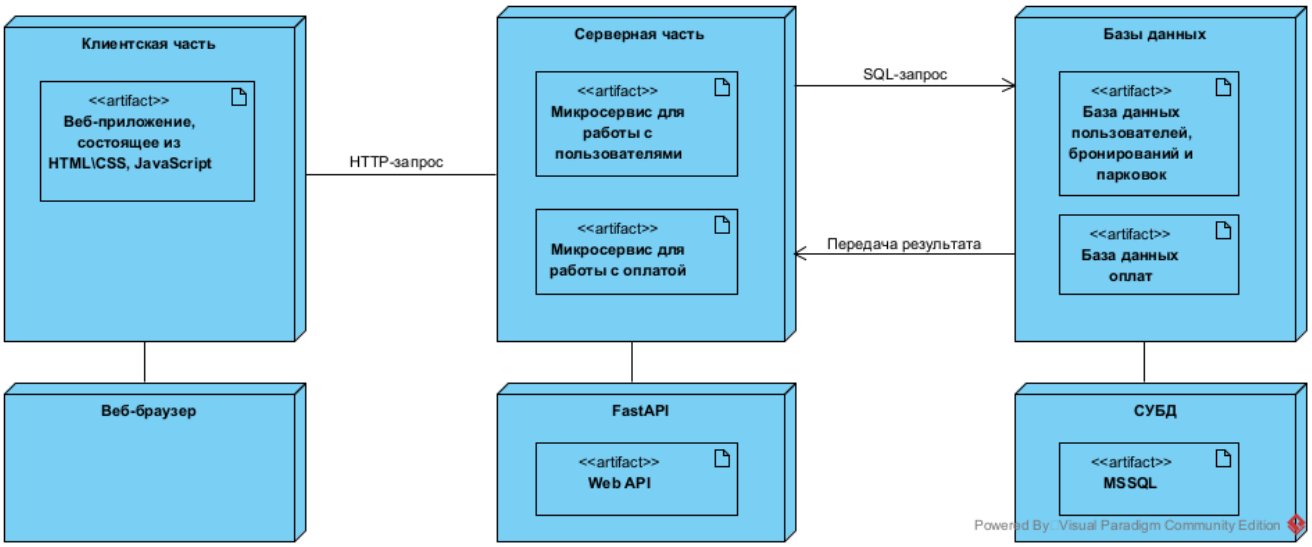

Рисунок 21 – Диаграмма развертывания

Диаграммы развертывания обычно используются для визуализации физического аппаратного и программного обеспечения системы. Используя его, можно понять, как система будет физически развернута на аппаратном обеспечении [18]. Так и в построенной диаграмме прослеживается, каким образом устроено приложение внутри, включая определяющие его узлы (клиентская часть, серверная часть, база данных т. д), которые, в свою очередь, сами состоят из отдельных программных элементов.

Для визуализации организации ключевых компонентов приложения и зависимостей между ними была построена диаграмма компонентов, которая представлена на рисунке 22. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установлиая зависимости между программными компонентами. А также она предоставляет общую картину архитектуры системы, тем самым помогая разработчикам лучше понять ее структуру и взаимосвязи, а также является полезным инструментом для коммуникации и документирования архитектурных решений.

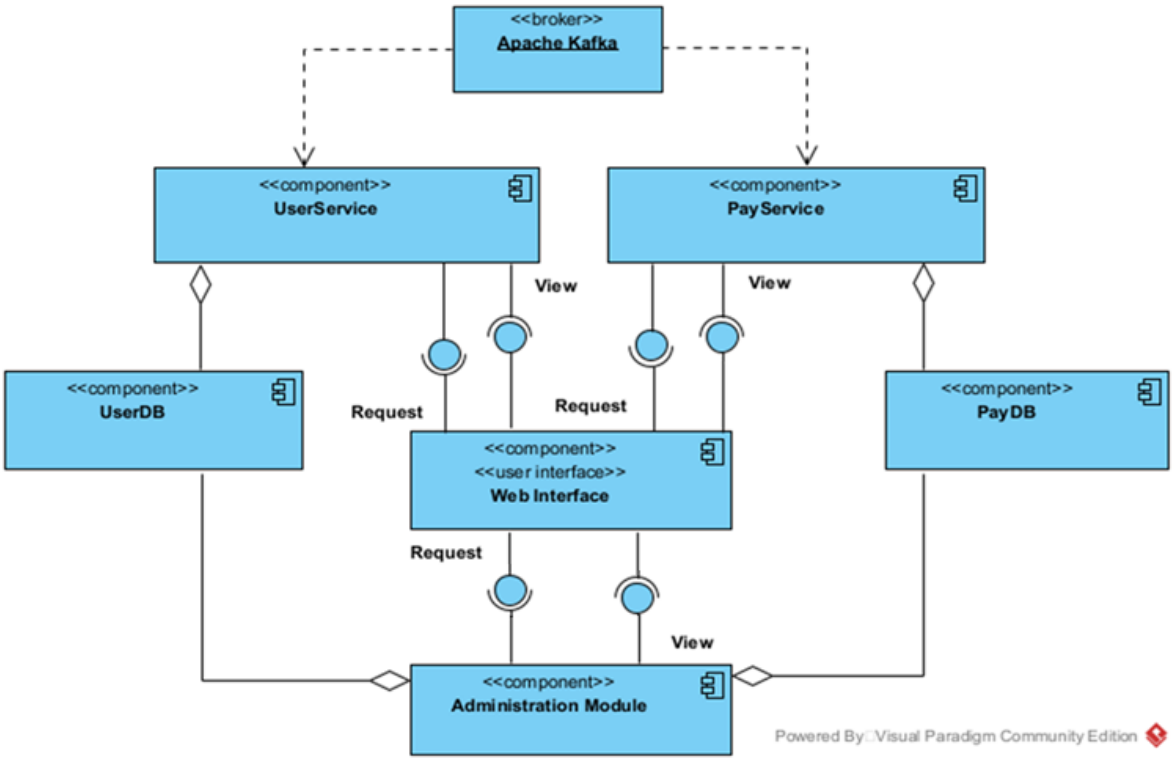

Рисунок 22 – Диаграмма компонентов

Диаграмма состоит из шести компонентов и связывающего два из них звена. Из компонентов выделяются два микросервиса и административный модуль. Все они взаимодействуют с веб-интерфейсом, задача которого отобразить полученные данные пользователю. С веб-интерфейса поступают запросы, компоненты их получают, обрабатывают и отсылают результат обратно. Каждый серверный компонент связан с базой данных связью агрегация, которая означает, что у каждого компонента есть зависимость от подключения для работы с ней, однако они не имеют строгой связи, что позволяет им существовать и функционировать независимо друг от друга.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель данных – схема наивысшего уровня с минимальным количеством подробностей. Достоинство этого подхода заключается в возможности отобразить общую структуру модели и всю архитектуру системы [19].

Для создания концептуальной модели данных необходимо составить перечень сущностей предметной области разрабатываемого приложения. Представление всех сущностей предметной области отображено в таблице 2.

Таблица 2 – Сущности базы данных

Наименование сущности	**Идентификатор сущности**	**Назначение сущности**
Пользователи	Users	Хранение информации о пользователях
Автомобильные стоянки	ParkingLocations	Хранение адресов расположения стоянок
Парковочные места	ParkingSpots	Хранение информации о конкретных парковочных местах
Бронирования	Bookings	Хранение бронирований парковочных мест пользователями
Отмененные бронирования	CancelledBookings	Хранение информации об отмененных бронированиях
Пользовательские роли	UserRoles	Хранение ролей пользователей
Сопоставленные пользовательские роли	UserRoleMapping	Сопоставление пользователей и их ролей
Автомобили	Cars	Хранение информации об автомобилях пользователей
Оплаты	Payments	Хранение платежей за бронирование парковочных мест

Графическое представление концептуальной модели базы данных отображено на рисунке 23. Все из представленных сущностей первого микросервиса имеют связь «один-ко-многим», которая означает то, что каждая отдельная запись в родительской сущности может относиться к нескольким записям в дочерней сущности.

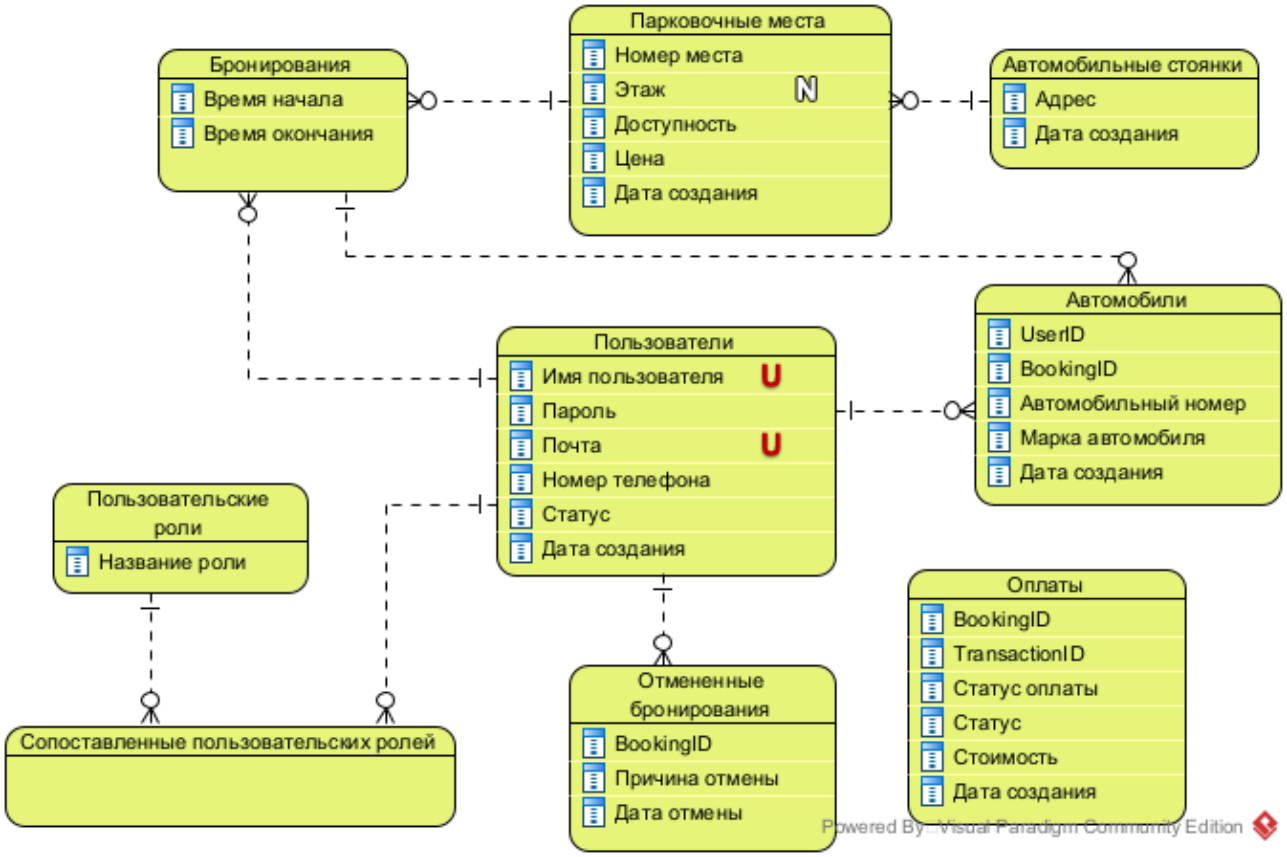

Рисунок 23 – Концептуальная модель базы данных предметной области «Поиск и резервирование машино-места на автомобильной стоянке»

Стоит обратить внимание, что в приложении реализована ролевая модель, тем самым, разграничиваются права доступа и обеспечивается разный функционал для отдельных категорий пользователей.

В приложении Г представлены наборы атрибутов для каждой сущности и их характеристики.

2.2.2 Логические модели баз данных

Одним из главных преимуществ использования стратегии базы данных микросервисов является возможность разделения данных на основе границ сервисов. Такая изоляция способствует независимости данных, тем самым устраняя конфликты и проблемы координации, часто возникающие в традиционных монолитных системах. При таком подходе каждый микросервис может самостоятельно выбирать тип базы данных или технологию в зависимости от конкретных бизнес-требований [20]. Кроме того, такое разбиение облегчает

поддержку приложения, так как, к примеру, проблемы, касающиеся одной части проекта, не будут относится к другому фрагменту приложения.

Поскольку веб-приложение предполагает два микросервиса, было принято разделить базу данных на такое же количество независимых частей. Первая отвечает за хранение пользовательских данных, их бронирований и информации об автомобильных стоянках, а вторая – за хранение сведений об оплатах.

Логическая модель данных – это расширение концептуальной модели данных. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи, которые представляют бизнес-информацию и определяют бизнес-правила [21].

Логические модели разделенных баз данных показаны на рисунках Д.1 и Д.2 в приложении Д.

2.3 Проектирование пользовательского интерфейса

Пользовательский интерфейс должен соответствовать нескольким требованиям, которые помогут сделать его удобным и понятным:

– простота – необходимо избегать перегруженности и усложнения интерфейса;

– ориентация на пользователя – создание сценария, по которому будет действовать пользователь;

– унифицированность – это требование значимо для того, чтобы у пользователя быстрее сформировалась привычка пользованием приложения;

– запоминаемость и уникальность – неотъемлемы для привлечения внимания и создания дальнейших ассоциаций пользователя [22].

Исход из вышеперечисленных критериев, был разработан интерфейс, который, в большей степени, удовлетворяет им и включает следующие страницы: регистрация, авторизация, главная страница, страница личного кабинета пользователя, страница указания данных автомобиля, кабинет администратора с тремя вкладками, каждая из которых содержит в себе отдельный функционал.

На страницах авторизации и регистрации должны быть необходимые поля для ввода необходимых данных и подсказки, на случай каких-либо некорректных данных или незаполненных полей. На главной странице должна быть размещена карта для нахождения и оценки местонахождения автомобильных стоянок, а также поля для указания данных бронирования. Помимо этого, должна быть иконка, при нажатии на которую появлялась дополнительная панель с возможностью войти в личный кабинет или выйти из приложения. На странице указания данных об автомобиле должны находиться поля для указания номера и марки автомобиля, после чего должна происходить переадресация на страницу оплаты. А в кабинете администратора должно быть реализовано разделение функционала для удобства контроля парковочными местами, управления учетными записями пользователей и анализа накопившихся данных за определенный период.

Оплата реализована через телеграм-бота, в котором встроена система оплаты от сервиса «PayMaster». Данный способ является безопасным и надежным, поскольку компания ООО «Пэймастер», разработавшая платежный сервис, имеет сертификат соответствия требованиям стандарта PCI DSS (стандарт безопасности данных индустрии платежных карт) и является в соответствии со статьей 14.1. федерального закона от 27.06.2011 N 161-ФЗ «О национальной платежной системе» банковским платежным агентом (агрегатором) крупных российских банков, таких как: «Банк Русский Стандарт», «Тинькофф Банк» и «МОСКОВСКИЙ КРЕДИТНЫЙ БАНК» [23]. Интерфейс же для оплаты устроен в виде диалога с ботом, который ищет счет за резервирование после отправки ему определенной команды. После оплаты боту следует отправлять сообщение об успешном подтверждении бронирования.

2.4 Проектирование классов веб-приложения поиска и резервирования машино-места на автомобильной стоянке

В серверной части приложения будут созданы классы для подключения и обращения к базам данных, для представления реляционных таблиц в виде объектной модели, создания единого объекта логирования (журналирования) и для валидации данных, получаемых с клиентской части. Для взаимодействия с базой данных будут реализованы классы репозитории, в которых будет передаваться объект с методом подключения к базе данных. Сам этот класс будет иметь все необходимые методы для манипуляции данными из базы данных. Таких классов будет два, поскольку столько и микросервисов. Также для валидации данных, что представляет собой проверку данных на соответствие заданным условиям и ограничениям, будут составлены классы, называемые схемами, которые будут в себе содержать статические поля и при необходимости методы для выполнения дополнительных проверок.

Диаграммы классов для каждого двух микросервисов и модуля для работы администратора представлены на рисунках Д.3, Д.4 и Д.5 в приложении Д.

Описание функциональных назначений каждого класса приведены в таблице 3.

Таблица 3 – Описание функциональных назначений классов приложения

Название класса	**Назначение**
SingletonLogger	Класс одиночка, который будет создавать единственный экземпляр для логирования.
DB_connection	Класс, использующаяся для подключения к базам данных, в зависимости какая будет подаваться при инициализации.
UserRepository	Класс, содержащий необходимые методы для обращения к базе данных первого микросервиса.
PayRepository	Класс, содержащий необходимые методы для обращения к базе данных оплаты.
User	Модель, предназначенная для связывания с таблицей пользователей.

Продолжение таблицы 3

Название класса	**Назначение**
ParkingLocation	Модель, предназначенная для связывания с таблицей автомобильных стоянок.
ParkingSpot	Модель, предназначенная для связывания с таблицей парковочных мест.
Booking	Модель, предназначенная для связывания с таблицей бронирований.
UserRole	Модель, предназначенная для связывания с таблицей пользовательских ролей.
UserRoleMapping	Модель, предназначенная для связывания с таблицей сопоставлений пользователей и ролей.
CancelledBooking	Модель, предназначенная для связывания с таблицей отмененных бронирований.
Token	Схема, используемая для представления токена доступа к приложению.
TokenData	Схема, используемая для хранения данных закодированных в токене.
SLoginForm	Схема, используемая для валидации данных авторизации.
SRegisterForm	Схема, используемая для валидации данных регистрации.
SUser	Схема, используемая для представления данных пользователя.
UserInDB	Схема, которая расширяет класс SUser, добавляя поле с хешированным паролем.
SBookingData	Схема, используемая для валидации данных бронирования пользователя.
Car	Модель, предназначенная для связывания с таблицей автомобилей пользователей.
Payment	Модель, предназначенная для связывания с таблицей оплат.
SCarInfoForm	Схема, используемая для валидации данных об автомобиле пользователя.

Окончание таблицы 3

Название класса	**Назначение**
AdminRepository	Класс, содержащий необходимые методы для обращения к базам данных.
ParkingLocationSchema	Схема, используемая для валидации данных об автомобильных стоянках.
ParkingSpotSchema	Схема, используемая для валидации данных о парковочных местах.
AdminUserSchema	Схема, используемая для валидации данных об обычных пользователях и администраторах.
UpdatePriceSchema	Схема, используемая для валидации данных о ценах парковочных мест.
BookingSchema	Схема, используемая для валидации данных о бронированиях пользователей.

Исходя из перечня и описания назначения каждого класса, можно сделать вывод, что структура серверной части приложения была разделена на несколько ключевых модулей, каждый из которых содержит в себе классы, для выполнения определенных функций, решающие отдельные задачи бизнес-логики.

3 Реализация веб-приложения поиска и резервирования машино-места на автомобильной стоянке

3.1 Реализация базы данных

Заключительным этапом в проектировании базы данных является создание физической модели данных. Физическая модель базы данных – это модель данных, которая определяет, каким образом представляются данные, и содержит все детали, необходимые СУБД для создания базы данных [24]. Для составления такой модели потребовалась логическая модель данных, ибо физические модели данных содержат дополнительные атрибуты, которых не было в логической модели данных, например триггеры, хранимые процедуры и типы данных.

Физические модели баз данных микросервисов были составлены в программе Visual Paradigm версии 17.2 с использованием нотации IDEF1X, они представлены на рисунках Д.6 и Д.7 в приложении Д.

3.2 Реализация серверной части приложени

Весь код приложения был написан в интегрированной среде разработки Visual Studio Code на языке программирования Python. Для создания быстрого, за счет асинхронных операций, приложения микросервисной архитектуры использовался фреймворк FastAPI, который также упрощает процесс тестирования, путем автоматического создания документации. А для запуска своего веб-сервера был применен инструмент из библиотеки uvicorn.

Приложение разделено на два микросервиса, каждый из которых содержит в себе модуль для взаимодействия с таблицами из базы данных, для валидации поступающих с клиентской стороны данных, для представления таблиц в виде классов и для маршрутизации HTTP-запросов по соответствующим методам. Для взаимодействия с базой данных из python кода использовалась библиотека sqlalchemy, предоставляющая все необходимые инструменты для манипуляции данными из базы данных. Для валидации же использовалась библиотека pydantic, благодаря которой легко создать схемы, которые будут являться

шаблонами для проверки корректности данных, помимо этого библиотека формирует достаточно понятные логи для того, чтобы в случае чего, можно было сразу выявить причину неправильности введенных данных пользователем. Весь код серверной части приложения представлен в приложении А.

Поскольку разработанные сервисы предоставляют REST API для взаимодействия, были разработаны методы, выполняющиеся в момент поступления HTTP-запросы по определенному URL-адресу. Реализованные методы описаны в таблице 4.

Таблица 4 – Методы API

URL запроса	**Http метод**	**Описание**
/secure-data	GET	Защищенный эндпоинт для доступа к данным, требующий аутентификации
/login	GET	Возвращает HTML-страницу для входа в систему
/login	POST	Обрабатывает данные входа, аутентифицирует пользователя и устанавливает cookie с токеном
/logout	GET	Выход из приложения, удаление токена доступа
/register	GET	Возвращает HTML-страницу регистрации
/register	POST	Обрабатывает данные регистрации нового пользователя
/token	POST	Генерирует токен доступа для аутентифицированного пользователя
/users/me/	GET	Возвращает информацию о текущем аутентифицированном пользователе
/main_page	GET	Возвращает главную страницу приложения
/api/parking_spots	GET	Возвращает список парковочных мест для выбранной автомобильной стоянки
/parking_prices	GET	Возвращает информацию о ценах на парковочные места
/user_acc/{user_id}	GET	Возвращает HTML-страницу с информацией об учетной записи пользователя и его бронированиях

Продолжение таблицы 4

URL запроса	**Http метод**	**Описание**
/cancel_booking/{booking_id}	DELETE	Отменяет указанное бронирование парковочного места
/book	POST	Создает новое бронирование парковочного места
/pay/{user_id}	GET	Возвращает HTML-страницу для указания данных об автомобиле
/pay	POST	Обрабатывает данные автомобиля и генерирует ссылку для оплаты через Telegram-бота
/save_payment	POST	Сохраняет информацию о платеже и очищает кэш бронирования
/cancel_booking/{user_id}	POST	Отменяет бронирование для указанного пользователя и очищает кэш
/waiting/{user_id}	GET	Возвращает HTML-страницу ожидания подтверждения платежа
/booking_details/{booking_id}	GET	Возвращает HTML-страницу с деталями бронирования
/admin/dashboard	GET	Возвращает HTML-панель администратора с открытой вкладкой, предназначенной для работы с парковочными местами
/admin/parkings	GET	Возвращает список всех адреса автомобильных стоянок
/admin/spots/{location_id}	GET	Возвращает список мест для конкретной стоянки
/admin/spots/{spot_id}/reserve	POST	Резервирует выбранное парковочное место
/admin/spots/{spot_id}/free	POST	Освобождает выбранное парковочное место
/admin/spots/{spot_id}/update_price	POST	Обновляет цену для выбранного места
/admin/users	GET	Возвращает список всех пользователей приложения
/admin/users/{user_id}/bookings	GET	Возвращает бронирования конкретного пользователя
/admin/users/{user_id}/status	POST	Обновляет статус пользователя
/admin/analytics/parkings	GET	Возвращает аналитику по парковкам за определенный период
/cars	POST	Сохраняет данные об автомобиле пользователя

Окончание таблицы 4

URL запроса	**Http метод**	**Описание**
/admin/analytics/spots	GET	Возвращает аналитику по парковкам за определенный период
/admin/analytics/revenue	GET	Возвращает аналитику по парковкам за определенный период
/logout	GET	Выход из системы (удаление токена)

Тем самым, для каждого действия пользователя была разработана своя точка входа на сервер, которая будет обрабатываться контроллером и при необходимости изменять состояние модели.

3.3 Реализация клиентской части приложени

Клиентская часть была реализована в том же редакторе кода Visual Studio Code. Для описания визуальной части сайта использовался гипертекстовый язык HTML. Для придания внешнего вида веб-страницам был применен формальный язык декорирования CSS. Для создания модальных окон, плавной анимации, предварительной валидации и отправки AJAX-запросов на сервер был задействован язык Java Script. А также для отображения интерактивной карты были применены инструменты библиотеки с открытым исходным кодом Leaflet. Код клиентской части приложения приведен в приложении Б.

Описание компонентов клиентской части приложения представлено в таблице 5.

Таблица 5 – Компоненты клиентской части приложения

Компонент	**Назначение**
register.html	HTML-страница регистрации новых пользователей
login.html	HTML-страница авторизации пользователей
main.html	Главная страница приложения с картой парковок и полями для создания бронирования
images/parking_schemes	Схемы парковочных мест на автомобильных стоянках

Продолжение таблицы 5

Компонент	Назначение
user_acc.html	Страница личного кабинета пользователя с информацией о бронированиях
booking_details.html	Страница с информацией об автомобиле и сумме конкретного бронирования
pay.html	Страница оплаты бронирования
waiting.hml	Страница ожидания сохранения данных о бронировании и возврата на главную страницу
admin.html	Панель администратора для управления парковками, пользователями и аналитической деятельности
register_page.css	Стили для страницы регистрации (оформление форм, кнопок, сообщений)
main_pg.css	Стили главной страницы
usr_acc_page.css	Стили страницы личного кабинета
booking_details.css	Стили страницы деталей бронирования
pay_page.css	Стили страницы оплаты
waiting_page.css	Стили страницы ожидания
admin_page.css	Стили административной панели
register_pg.js	Скрипт для обработки формы регистрации и валидации
main_pg.js	Скрипт для главной страницы
usr-acc.js	Скрипт для работы личного кабинета
payment.js	Логика обработки данных об автомобиле
wait_pg.js	Скрипт для страницы ожидания
admin_pg.js	Скрипт панели администратора

В ходе реализации были созданы все необходимые компоненты, отвечающие за визуальную составляющую приложения, что обеспечивает функционирование клиентской части.

3.4 Реализация интерфейса приложения

В соответствии с требованиями к пользовательскому интерфейсу, заявленными в пункте 2.3, были реализованы следующие страницы веб-приложения:

– страница регистрации;

– страница авторизации;

– страница для оформления бронирования;

– личный кабинет пользователя;

– страница с деталями бронирования (информацией об автомобиле и сумме бронирования);

– страница для указания данных об автомобиле;

– страница для ожидания сохранения данных о бронировании в базу данных;

– страница для перехода в телеграм-бота для оплаты.

Первая страница, которая встречает пользователя, зашедшего в приложение, это страница регистрации, которая представлена на рисунке 24.

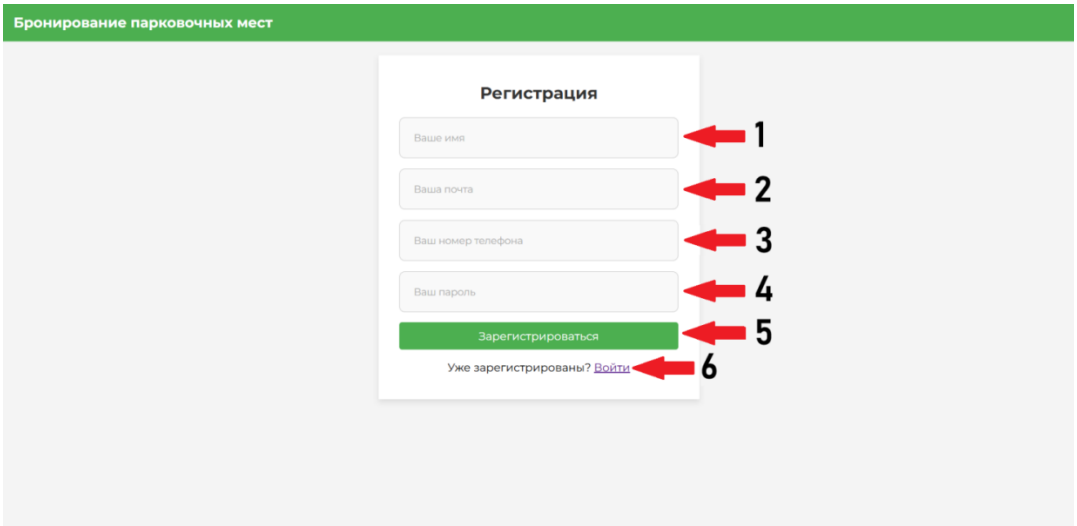

Рисунок 24 – Страница регистрации

На странице регистрации представлены следующие ключевые элементы:

1 – поле ввода имени пользователя;

2 – поле ввода почты пользователя;

3 – поле ввода номера телефона пользователя;

4 – поле ввода пароля пользователя;

5 – кнопка «Зарегистрироваться»;

6 – ссылка на страницу авторизации.

В случае если пользователь введет корректные данные для регистрации, то он будет автоматически перенаправлен на главную страницу приложения.

Если же у пользователя уже имеется учетная запись, то он может перейти по ссылке и попасть на страницу с авторизацией (рисунок 25).

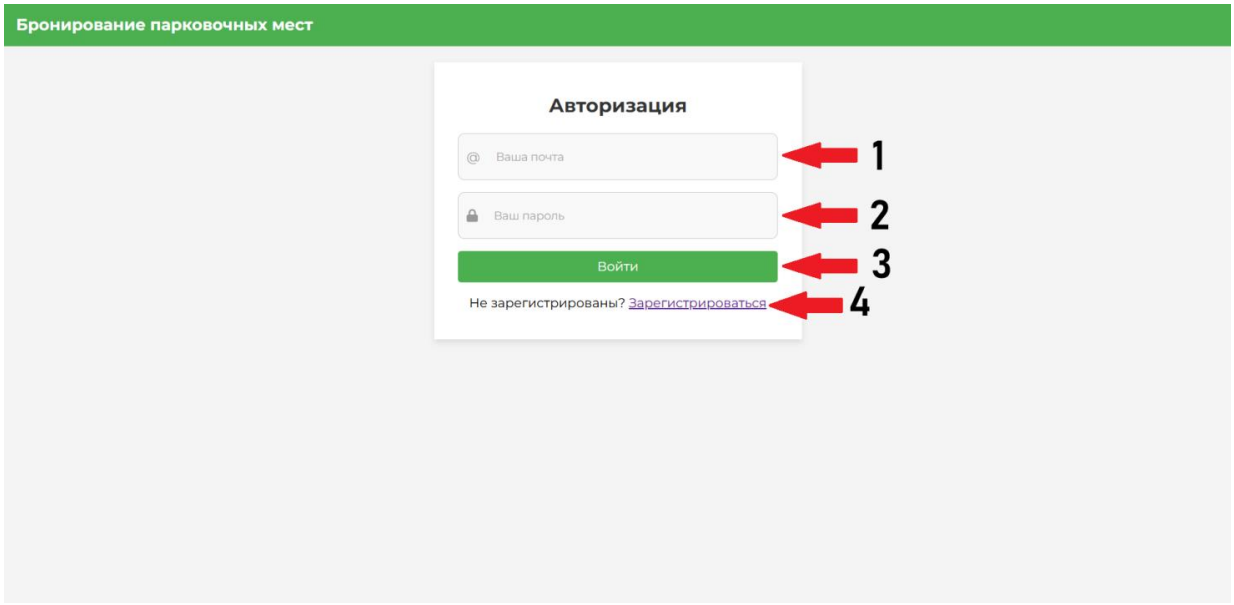

Рисунок 25 – Страница авторизации

На странице авторизации расположены следующие ключевые элементы:

1 – поле ввода почты пользователя;

2 – поле ввода пароля пользователя;

3 – кнопка «Войти»;

4 – ссылка для перехода на страницу регистрации.

После ввода существующих данных для авторизации, пользователь также непременно попадает на главную страницу приложения, которая показана на рисунке 26.

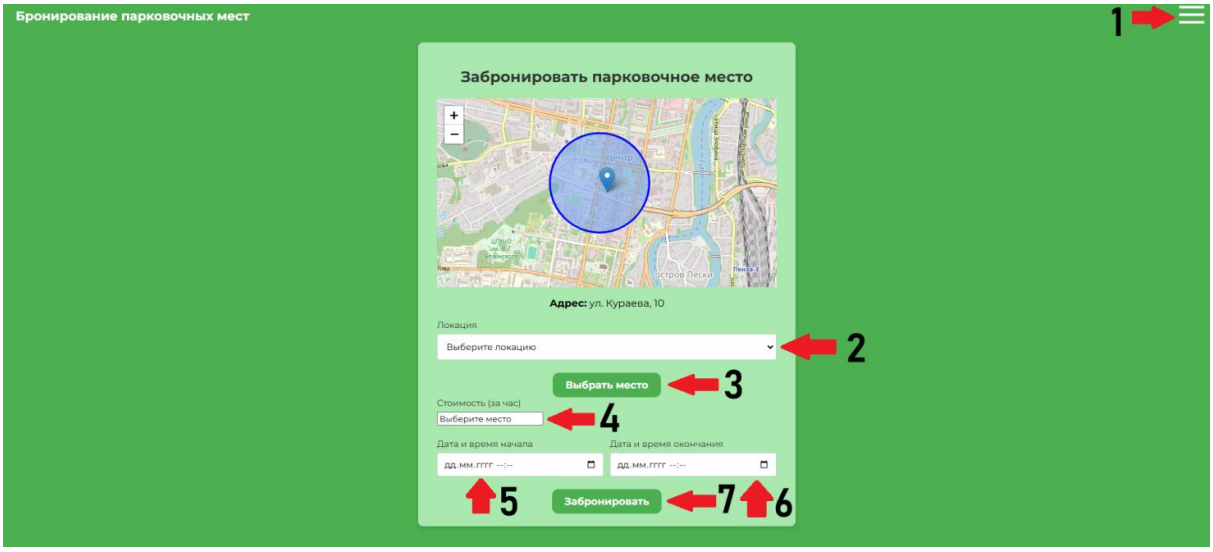

Рисунок 26 – Главная страница приложения

На главной странице расположены следующие ключевые элементы:

1 – кнопка для появления боковой панели;

2 – выпадающий список с адресами автомобильных стоянок;

3 – кнопка для открытия модального окна со всеми парковочными местами и схемой стоянки;

4 – поле для отображения стоимости выбранного места;

5 – поле для указания даты начала резервирования;

6 – поле для указания даты окончания резервирования;

7 – кнопка «Забронировать» для создания бронирования и перехода на страницу с указанием данных об автомобиле.

В противном случае появится всплывающая подсказка о некорректности введенных данных (рисунок 27).

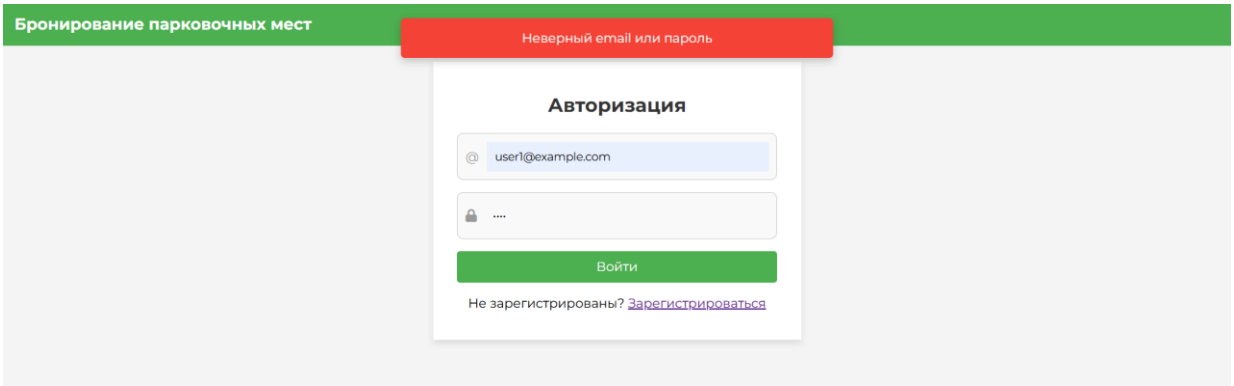

Рисунок 27 – Всплывающая подсказка о некорректности введенных данных

Боковая панель выглядит следующим образом (рисунок 28).

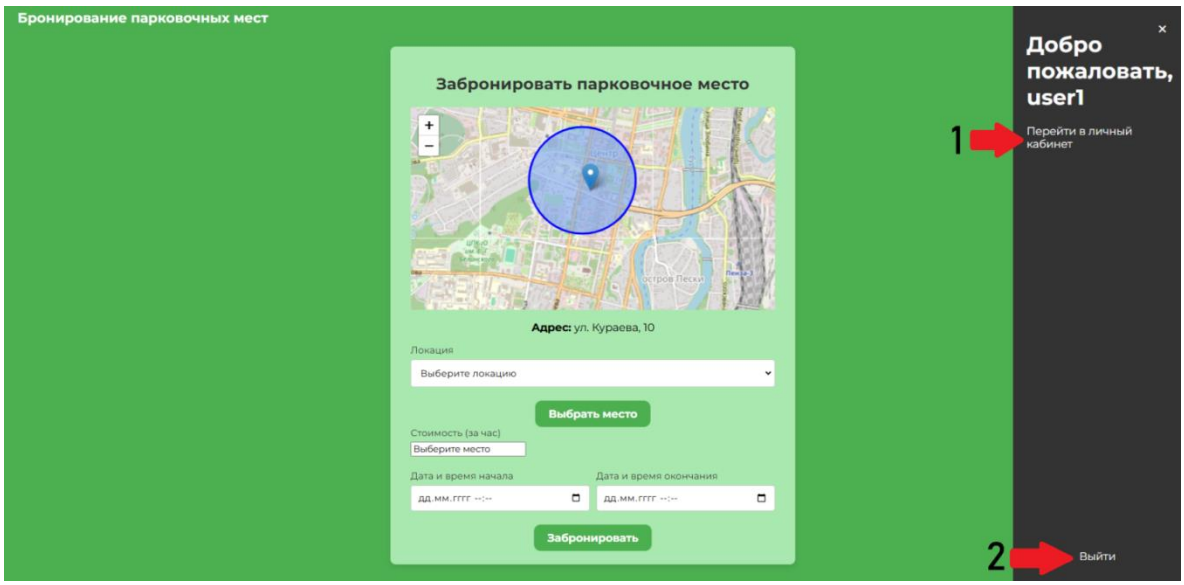

Рисунок 28 – Боковая панель главной страницы

На панели расположены две ссылки:

1 – ссылка на переход в личный кабинет;

2 – ссылка на выход из приложения.

Модальное окно с выбором места продемонстрировано на рисунке 29.

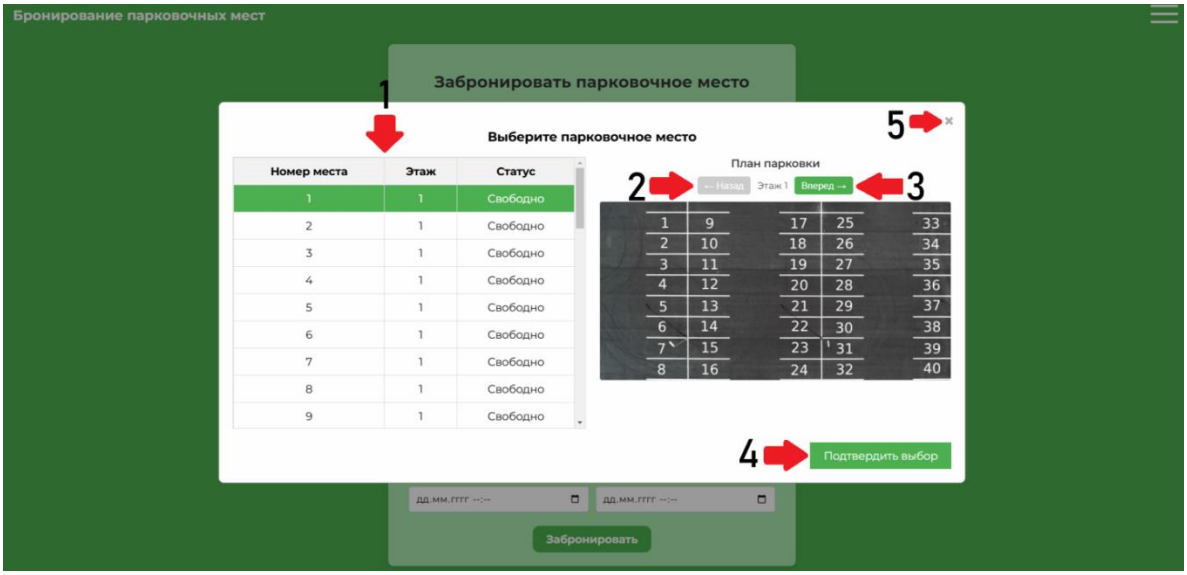

Рисунок 29 – Модальное окно с выбором места

Модальное окно имеет следующие ключевые элементы:

1 – таблица, состоящая из трех столбцов (номер места, этаж и статус);

2 – кнопка «Назад» для возврата к предыдущему этажу;

3 – кнопка «Вперед» для перехода к следующему этажу;

4 – кнопка для подтверждения выбранного места;

5 – кнопка для выхода из модального окна.

После нажатия на кнопку «Забронировать» пользователь попадает на страницу с указанием данных об автомобиле (рисунок 30).

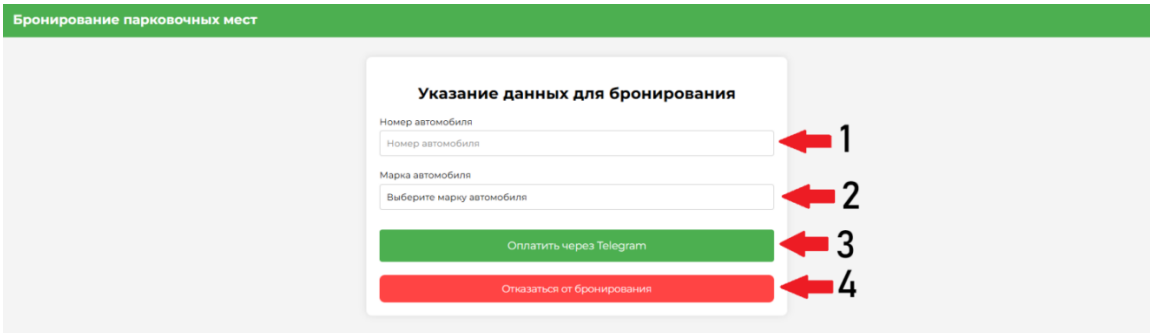

Рисунок 30 – Страница с указанием данных об автомобиле

Данная страница состоит из четырех основных элементов:

1 – поле ввода номера автомобиля;

2 – выпадающий список с марками автомобилей;

3 – кнопка «Оплатить через Telegram» для перехода в телеграм-бота для оплаты бронирования;

4 – кнопка «Отказаться от бронирования» для отмены созданного бронирования.

После заполнения полей и нажатия на кнопку «Оплатить через Telegram» произойдет переход на страницу, которая предложит перейти в телеграм-бот, чтобы выполнить оплату созданного бронирования (рисунок 31).

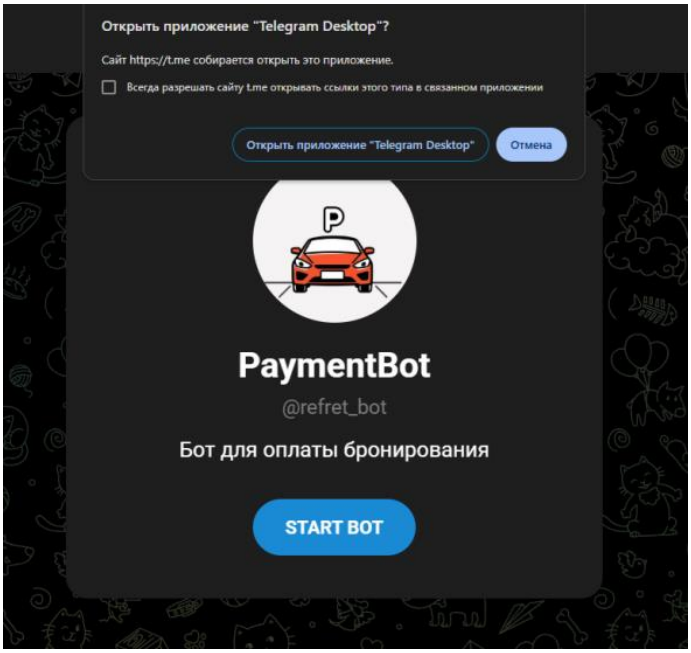

Рисунок 31 – Содержимое страницы для перехода в телеграм-бота

В это время на прошлой странице будет осуществляться сохранения данных о бронировании и возвращение на главную страницу (рисунок 32).

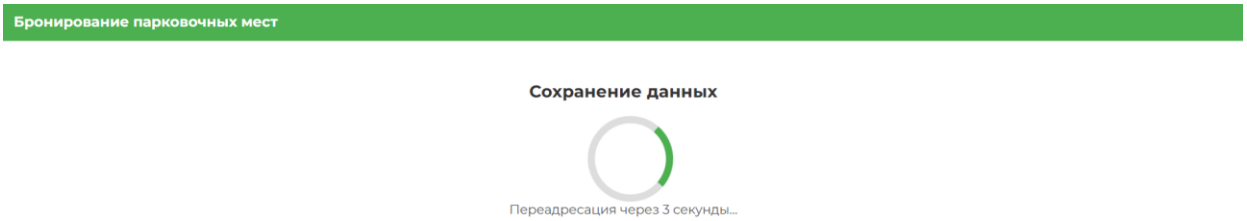

Рисунок 32 – Страница ожидания

После же перехода в телеграм-бота будет представлен интерфейс в виде диалога, в котором предусмотрены две команды для взаимодействия с ним. (рисунок 33).

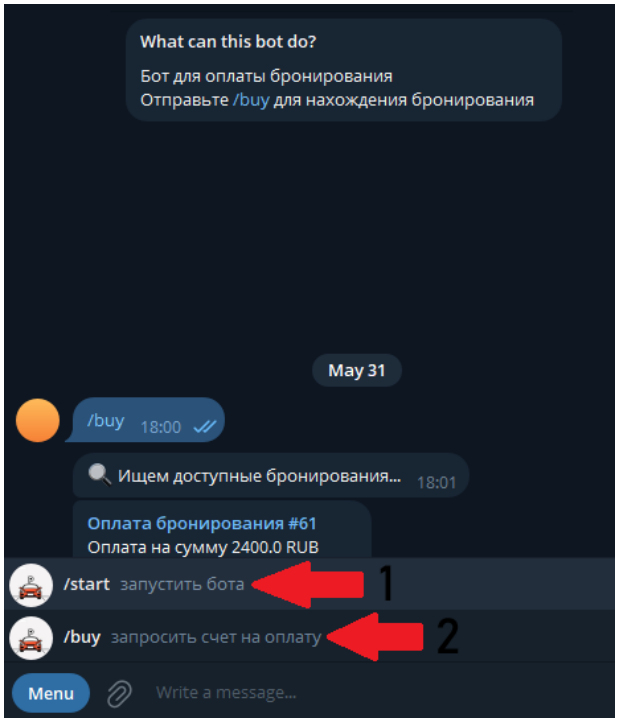

Рисунок 33 – Диалог с телеграм-ботом

Есть возможность пользоваться двумя командами:

1 – команда «start» предназначена для запуска бота;

2 – команда «buy» предназначена для нахождения созданного ранее бронирования с целью оплаты.

Форма для указания данных для оплаты представлена на рисунке 34.

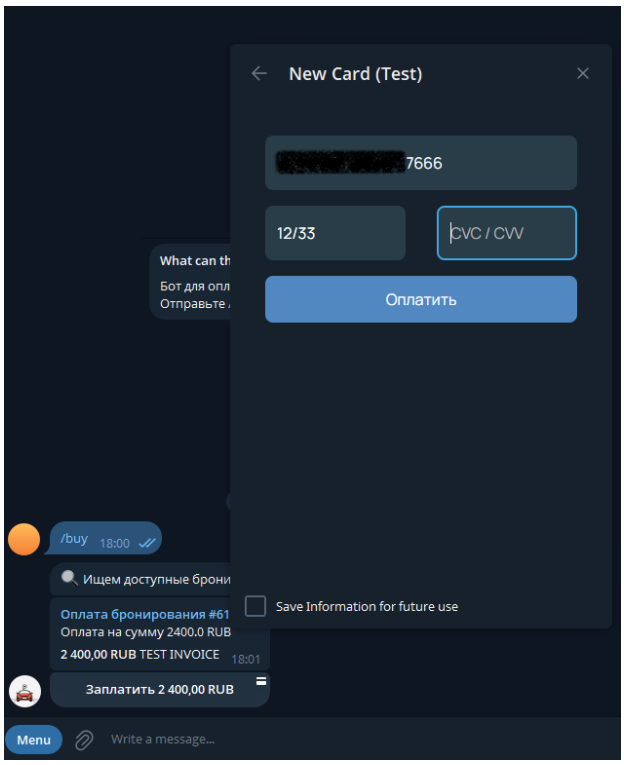

Рисунок 34 – Форма для указания данных для оплаты

Затем после успешной оплаты, в личном кабинете пользователя появится запись с информацией об оформленном резервировании (рисунок 35).

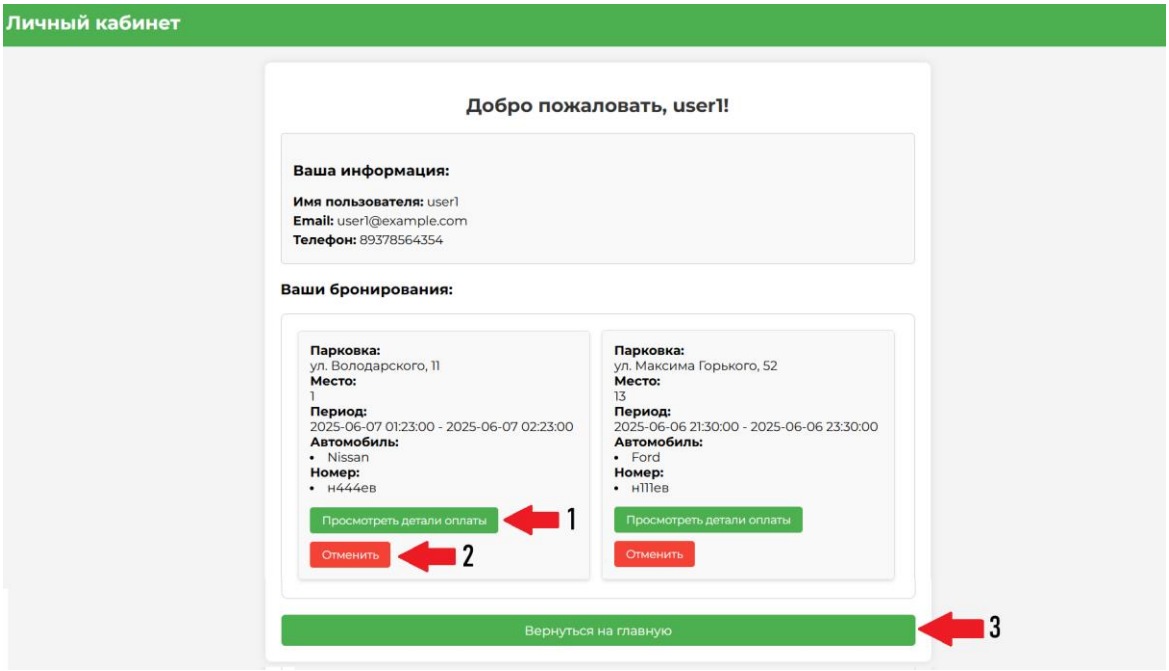

Рисунок 35 – Личный кабинет пользователя с оформленным бронированием

На странице с оформленным, как минимум, одним бронированием присутствуют следующие элементы:

1 – кнопка «Просмотреть детали оплаты», которая переносит пользователя на страницу с информацией о сумме, статусе бронирования и идентификаторе транзакции;

2 – кнопка «Отменить», которая отменяет выбранное бронирование при условии, что это будет сделано за сутки до начала резервирования;

3 – кнопка «Вернуться на главную», которая возвращает пользователя на главную страницу приложения.

Страница со сведениями сумме бронирования показана на рисунке 36.

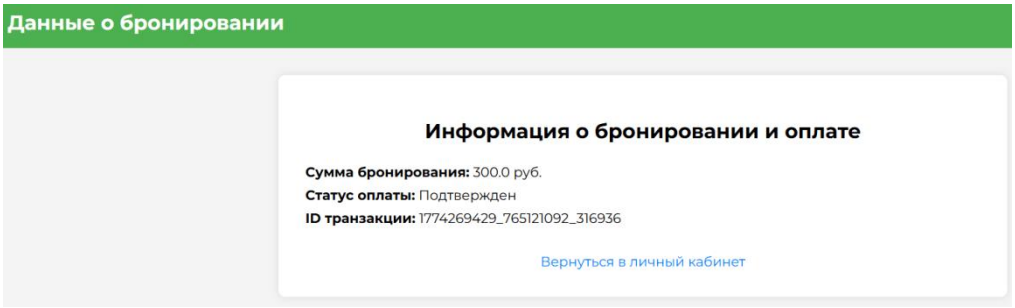

Рисунок 36 – Страница с информацией о сумме, статусе бронирования и идентификаторе транзакции

Помимо пользовательского функционала, была разработана страница с инструментами для администрирования, куда входит управление парковочными местами (рисунок 37), пользователями (рисунок 38, 39) и анализ автомобильных стоянок и доходов (рисунок 40).

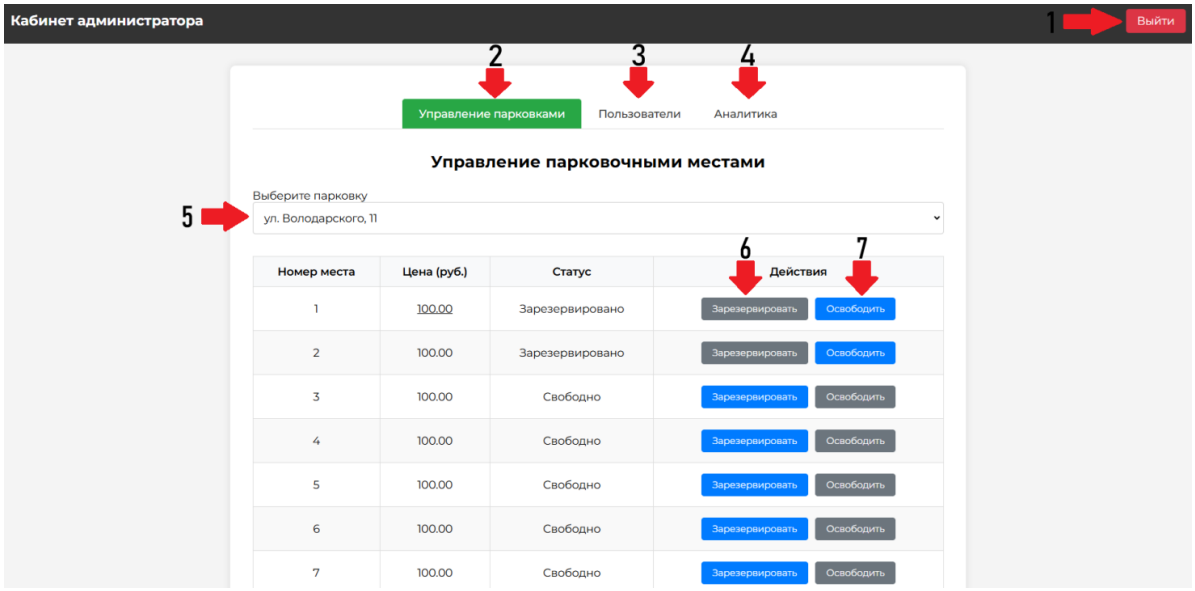

Рисунок 37 – Кабинет администратора с открытой вкладкой «Управление парковками»

На данной странице размещены следующие элементы:

1 – кнопка «Выйти», которая позволяет выйти из кабинета администратора;

2 – вкладка «Управление парковками», в которой содержится таблица со всеми местами выбранной парковки;

3 – вкладка «Пользователи», в которой содержится таблица со всеми местами пользователями приложения;

4 – вкладка «Аналитика», в которой содержатся гистограммы для представления аналитических данных;

5 – выпадающий список со всеми автомобильными стоянками;

6 – кнопка «Зарезервировать», которая позволяет администратору занять свободное место;

7 – кнопка «Освободить», которая позволяет администратору освободить занятое место.

В случае если администратор освобождает зарезервированное пользователем место, ему приходит оповещение об этом (рисунок 38).

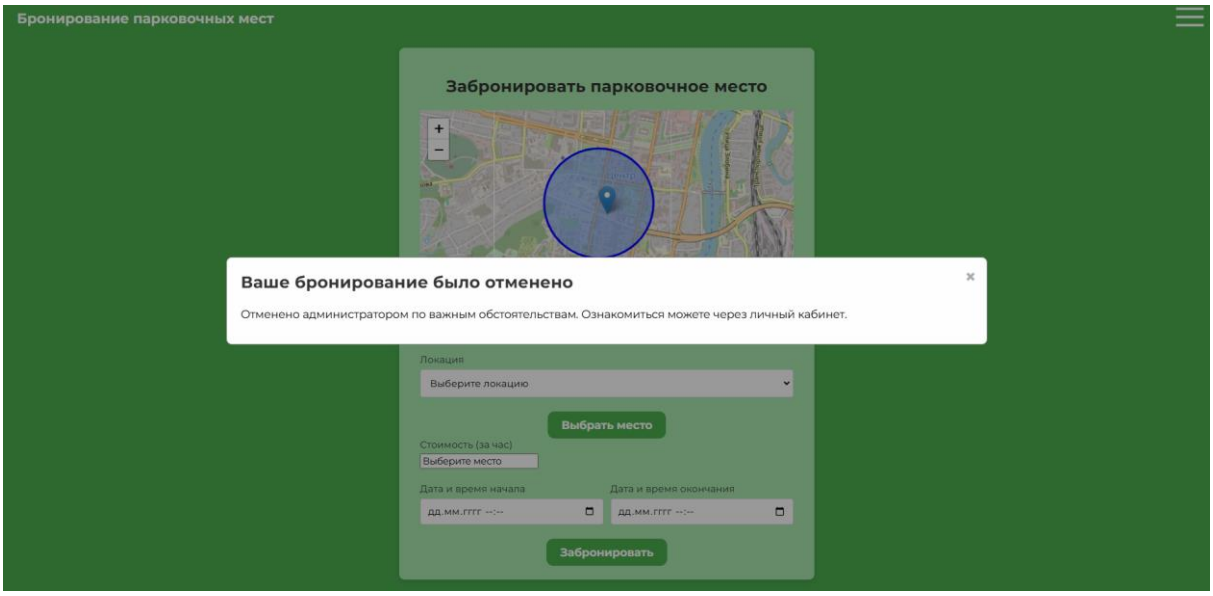

Рисунок 38 – Предупреждение о том, что бронирование было отменено администратором

Содержимое вкладки «Пользователи» представлено на рисунке 39.

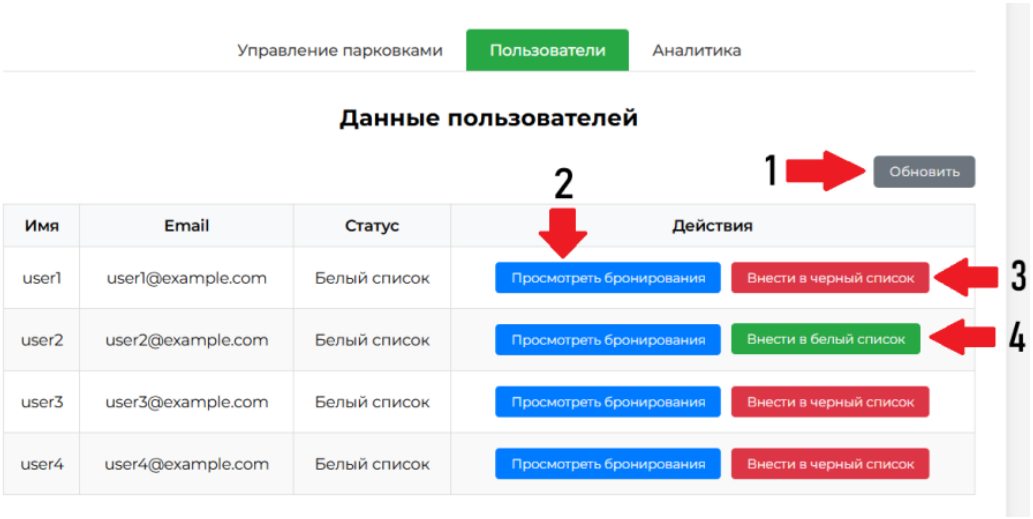

Рисунок 39 – Содержимое вкладки «Пользователи»

Вкладка включает в себя три кнопки:

1 – кнопка «Обновить» предназначена для обновления списка пользователей приложения;

2 – кнопка «Просмотреть бронирования» предназначена для открытия модального окна с информацией обо всех бронированиях пользователя;

3 – кнопка «Внести в черный список» предназначена для занесения пользователя в черный список, после чего у него не будет возможности авторизоваться в приложении;

4 – кнопка «Внести в белый список» предназначена для занесения пользователя в белый список, после чего он сможет вновь авторизоваться в приложении и пользоваться всем функционалом.

Содержимое модального окна после нажатия на кнопку «Просмотреть бронирования» показано на рисунке 40.

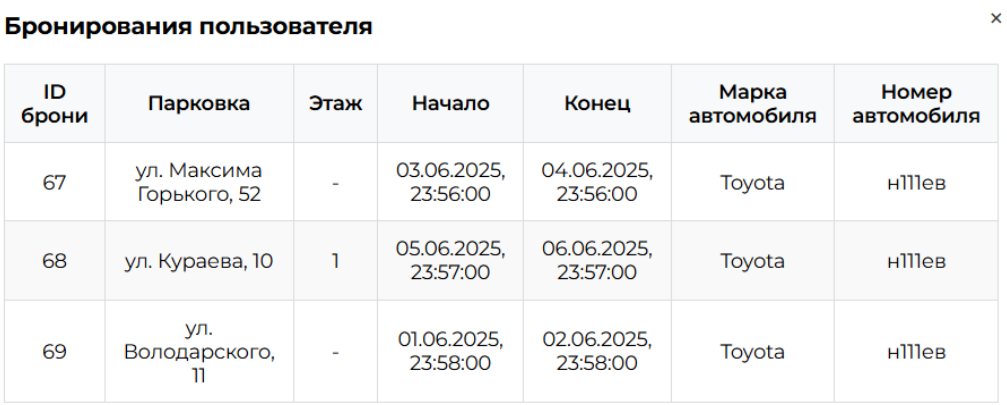

Рисунок 40 – Информация о бронированиях конкретного пользователя

И содержимое заключительной вкладки, предназначенной для проведения анализа накопившихся данных за определенный промежуток времени, представлен на рисунке 41.

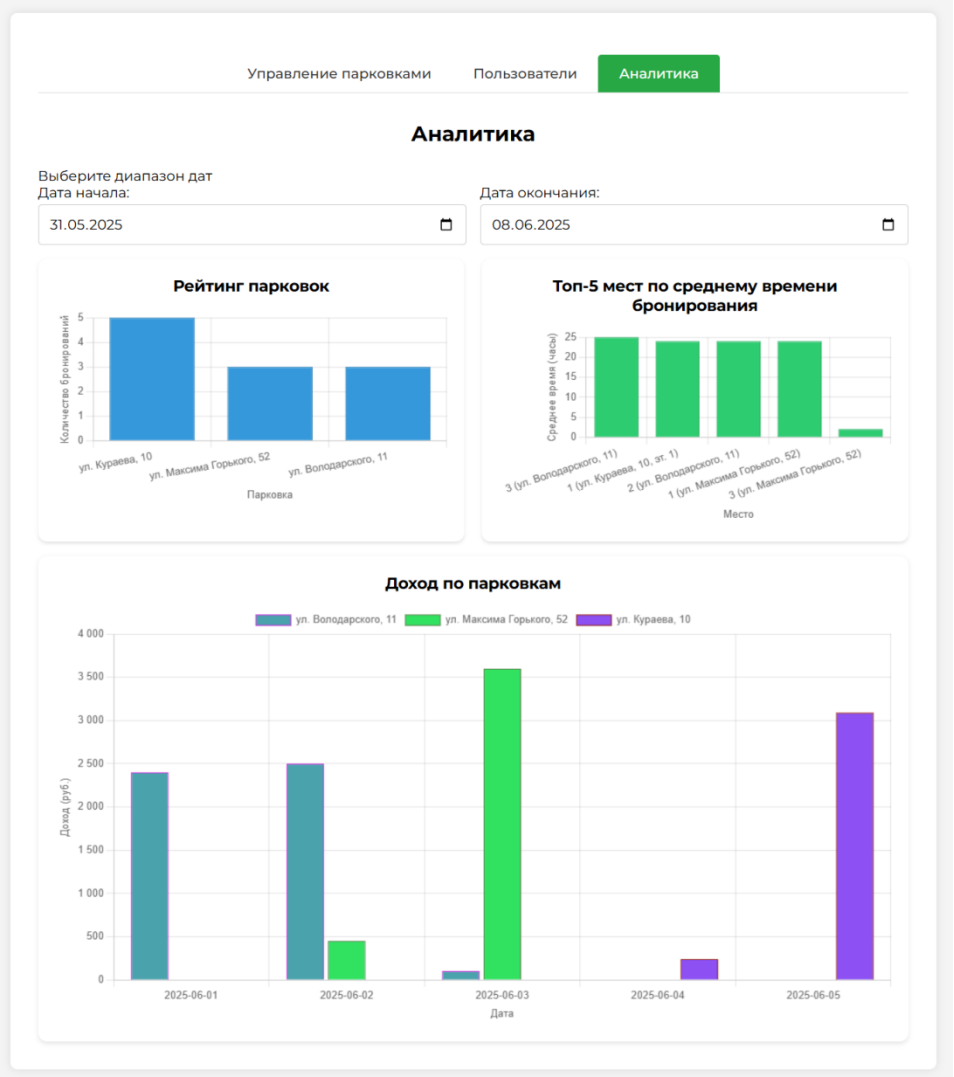

Рисунок 41 – Содержимое вкладки «Аналитика»

Для отображения гистограмм для анализа использовалась библиотека Chart.js с открытым исходным кодом на JavaScript, позволяющая создавать красивые и интерактивные диаграммы для веб-страниц. Она проста в использовании и поддерживает различные типы диаграмм, такие как линейные, столбчатые, круговые, радиальные и многие другие [25]. Было реализовано три вида анализа, такие как рейтинг парковок, определяемый по количеству забронированных мест на той или иной стоянке; пять наиболее востребованных мест по среднему времени бронирования; и доход по парковкам за указанный период времени.

4 Тестирование

4.1 Тестирование черным ящиком

Тестирование – это проверка программного обеспечения, которая показывает, соответствует ли оно ожиданиям разработчиков и правильно ли работает [26].

Тестирование по стратегии черного ящика – это процесс тестирования системы и ее поведения вне зависимости от ее внутренней структуры, архитектуры и реализации. Тестировщик осуществляет ввод, а вывод рассматривается как часть этой методики тестирования ПО. Это позволяет определять реакцию системы на ожидаемые и неожиданные действия пользователя, время реакции, сложности использования и проблемы с надежностью [27].

В качестве методики тестирования черным ящиком были выбраны классы эквивалентности (техника анализа классов эквивалентности).

Классы эквивалентности – это группировка тестовых данных, в рамках которой все элементы ведут себя одинаково. То есть, если система корректно обработает одно значение из класса, то она корректно обработает и все остальные значения этого класса [28].

Техника анализа классов эквивалентности – это метод тестирования программного обеспечения, который позволяет уменьшить количество тестовых сценариев, не уменьшая при этом покрытие тестами. Суть этой техники заключается в разделении всех возможных входных данных на классы, которые эквивалентны друг другу, и выборе по крайней мере одного представителя из каждого класса для тестирования. Это позволяет нам убедиться, что программа работает корректно для всех значений из каждого класса эквивалентности, а также покрыть все варианты возможных входных данных.

Разработка тестов посредством техники анализа классов эквивалентности осуществляется в три ключевых этапа:

1. Определить классы эквивалентности – это главный шаг техники, т к. во многом от него зависит эффективность ее применения.

2. Выполнение тестов – на этом шаге следует выполнить тесты от каждого класса эквивалентности [29].

В качестве рассматриваемых входных данных возьмем значения, которыми может быть заполнено поле ввода почтового адреса при регистрации.

Классы эквивалентности для данного поля представлены на таблице 6.

Таблица 6 – Классы эквивалентности

	Правильные КЭ	**Неправильные КЭ**
Длина почтового адреса	5-15(1)	0-4(2) 16-∞(3)
Структура	Символ «@»должен разделять имя пользователя и домен(4)	Адрес не содержит символа «@»(6)
	Домен состоит из одного или нескольких сегментов, разделенных точками(5)	Доменное имя содержит две и более точек(7)
Содержание	Должен состоять из [a-zA-Z0-9.-_](8)	Адрес содержит специальные символы(9)
		Имя пользователя содержит пробелы(10)

Построение тестов (тест-кейсов) на основе классов эквивалентности представлено на таблице 7.

Таблица 7 – Тест-кейсы

Номер теста	**Классы Эквивалентности**	**Входные данные**	**Выходные данные**
1	1, 5, 8, 9	example@gmail.com	Успешно
2	3, 5, 8, 9	verylongusername@mail.ru	Ошибка: В имени пользователя более 15-ти символов

Продолжение таблицы 7

Номер теста	**Классы Эквивалентности**	**Входные данные**	**Выходные данные**
3	1, 5, 8, 9, 11	user1@inbox..ru	Ошибка: Двойные точки
4	1, 5, 9, 10	user_2gmail.com	Ошибка: Отсутствует символ «@»
5	1, 5, 7, 8, 9	user12 3@gmail.com	Ошибка: Пробел в имени пользователя
6	1, 5, 6, 8, 9	user#1@gmail.com	Ошибка: Присутствует специальный символ в имени пользователя
7	2, 5, 8, 9	user@domain.ru	Ошибка: Длина имени пользователя меньше 5 символов
8	2, 6, 10, 11	bad!gmail...com	Ошибки: Длина имени пользователя меньше 5 символов, наличие спецсимвола вместе с тремя точками в месте разделения сегментов почтового адреса и отсутствие символа «@»

Окончание таблицы 7

Номер теста	**Классы Эквивалентности**	**Входные данные**	**Выходные данные**
9	3, 5, 6, 7, 8, 9	bad user name$@domain.com	Ошибки: Длина имени пользователя более 15 символов, наличие двух пробелов и одного спецсимвола

Таким образом, было проведено тестирование черным ящиком.

4.2 Тестирование белым ящиком

Тестирование методом белого ящика – метод тестирования ПО, который предполагает, что внутренняя структура системы известны тому, кто ее тестирует. Мы выбираем входные значения, основываясь на знании кода, который будет их обрабатывать. Точно так же мы знаем, каким должен быть результат этой обработки. Знание всех особенностей тестируемой программы и ее реализации – обязательны для этой техники. Тестирование белого ящика – углубление во внутреннее устройство системы, за пределы ее внешних интерфейсов [30].

Два основных метода проведения тестирования белого ящика:

– statement coverage testing – покрытие операторов;

– decision/branch coverage testing – покрытие решений.

Первый метод тестирования гарантирует, что каждая команда в коде будет выполнена и проверена хотя бы один раз. Вторым же методом проверяются все возможные пути в коде, где есть условные операторы [31].

Помимо данных методов был выбран еще один критерий для тестирования белого ящика – покрытие условий и решений. Согласно данному критерию необходимо составить тестовые случаи таким образом, чтобы результаты

каждого условия, решения выполнялись как минимум один раз, а также чтобы каждый оператор был выполнен хотя бы один раз.

Для тестирования белым ящиком был выбран метод для аутентификации пользователя «authenticate_user» (листинг 1).

Листинг 1 – Метод «authenticate_user»

```
async def authenticate_user(self, email: str, password: str): # a
        user = await self.get_user(email)

        if not user: # b
            logger.error("Пользователь не найден")
            raise HTTPException(status_code=401, detail="Неверный
email или пароль") # c

        logger.info(f"Stored hashed password: {user.Password}")
        logger.info(f"Entered password: {password}")

        if not self.verify_password(password, user.Password): # d
            logger.error("Ошибка верификации пароля")
            raise HTTPException(status_code=401, detail="Неверный
email или пароль") # e

        if user.Status == "Black": # f
            logger.error(f"Пользователь {email} в черном списке")
            raise HTTPException(status_code=403, detail="Вас
добавили в черный список") # g

        logger.info("Пароль успешно верифицирован")
        return user # h
```

Данный метод получает пользователя из базы данных по введенной почте, после чего сравнивает пароли на предмет соответствия и, в завершение, выполняет проверку статуса для доступа к приложению. Критерий покрытия операторов представлен в таблице 8.

Таблица 8 – Покрытие операторов

№	Операторы	Входные данные	Выходные данные
1	c, e, g	email = «user1@example.com» password=123	user = «»

Операторы были покрыты хотя бы один раз и в переменную user не было занесено новое значение. Критерий покрытие решений представлен на таблице 9.

Таблица 9 – Покрытие решений

№	Решения	Входные данные	Выходные данные
1	b – True, d – True, f – True	email = «fake-user@example.com» password = «111»	HTTPException(401, «Неверный email или пароль»)
2	b – False, d – False, f – False	email = «user1@example.com» password = «123»	Возврат user (аутентификация пройдена)

Критерий покрытие решений и условий представлен на таблице 10.

Таблица 10 – Покрытие решений и условий

№	Условие	Входные данные	Выходные данные
1	b – True	email = «fake-user1@example.com» password = «1112»	HTTPException(401, «Неверный email или пароль»)
2	b – False	email = «user1@example.com» password = «123»	user – существует, переход к проверке пароля
3	d – True	email = «user1@example.com» password = «321»	HTTPException(401, «Неверный email или пароль»)
4	d – False	email = «user1@example.com» password = «123»	переход к проверке статуса
5	f – True	email = «user2@example.com» password = «123»	HTTPException(403, «Вас добавили в черный список»)
6	f – False	email = «user1@example.com» password = «123»	успешная аутентификация, возвращается user

В ходе тестирования белым ящиком было достигнуто высокое покрытие операторов и решений, что означает, что все логические условия и ветвления в коде были выполнены хотя бы один раз. Это подтверждает, что программный

код корректно отрабатывает с различными входными данными, что минимизирует возникновение ошибок во время функционирования.

4.3 Метрики

Метрики кода представляют собой набор оценок программного обеспечения, которые дают разработчикам более глубокое представление о разрабатываемом коде. Используя преимущества метрик кода, разработчики могут понять, какие типы и методы следует переработать или более тщательно протестировать [32].

В данной работе использовались метрика сложности потока управления программы (цикломатическая сложность) и количественная метрика (метрика Холстеда).

Цикломатическая сложность (CC) – это показатель сложности кода, который рассчитывается на основании графа, построенного по точкам принятия решений, определяющим разные пути исполнения кода.

Формула для расчета CC:

$$CC = E - N + 2 \times P,$$

где E – количество ребер;

N – количество узлов;

P – количество предикативных узлов (то есть узлов, содержащих условие).

Точками принятия решений могут быть операторы и циклы. Чем их больше в одной функции и, чем больше их вложенность, тем больше и запутаннее у нас получится граф возможных путей исполнения кода этой функции.

Граф потока управления (CFG) – это графическое представление потока управления или вычислений во время выполнения программ или приложений

В графе потока управления каждый узел (вершина) графа соответствует базовому блоку – прямолинейному участку кода, не содержащему в себе ни операций передачи управления, ни точек, на которые управление передается из других частей программы. Направленные дуги используются в графе для представления инструкций перехода [33].

Управляющий граф программы представлен на рисунке 42.

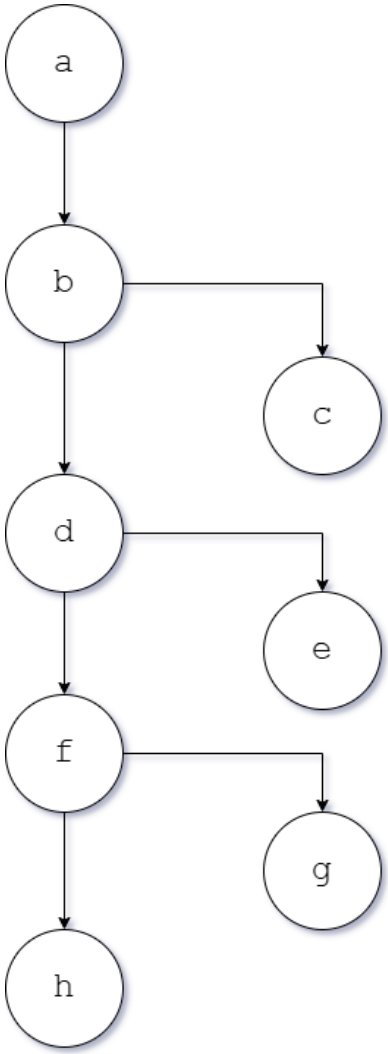

Рисунок 42 – Управляющий граф программы

В данном случае цикломатическая сложность будет равна:

$$CC = E - N + 2P = 7 - 8 + 6 = 5.$$

Значение 1–10 указывает на простоту функции. Тем самым, код легко, тестировать и отлаживать.

К группе метрик, основанных на подсчете некоторых единиц в коде программы, относят метрики Холстеда. Данные метрики основаны на следующих показателях:

– n_1 – число уникальных операторов программы, включая символы, разделители, имена процедур и знаки операций (словарь операторов);

– n_2 – число уникальных операндов программы (словарь операндов);

– N_1 – общее число операторов в программе;

– N_2 – общее число операндов в программе.

На основании этих показателей рассчитываются 4 количественных оценки:

1. Словарь программы (Halstead Program Vocabulary, HPVoc): n = $n_1 + n_2$.
2. Длина программы (Halstead Program Length, HPLen): N = $N_1 + N_2$.
3. Объем программы (Halstead Program Volume, HPVol): V = N× $\log_2$n.
4. Сложность программы (Halstead Difficulty, HDiff): D = $(\frac{n_1}{2}) \times (\frac{N2}{n_2})$.

Детальный подсчет показателей для расчета метрик Холстеда представлен на таблице 11.

Таблица 11 – Подсчет показателей для расчета метрик Холстеда

Операторы	**Число операторов**	**Операнды**	**Число операндов**
async def	1	self	1
if	3	email	2
await	1	password	2
not	2	user	5
raise	3	user.Password	2
logger.error	3	user.Status	1
logger.info	3	HTTPException	3
return	1	«Пользователь не найден»	1

В результате подсчета показателей, получились следующие значения: n_1 = 8; n_2 = 12; N_1 = 17; N_2 = 17. На их основе был выполнен расчет метрик Холстеда, который выглядит следующим образом:

Словарь программы:

$$n\ (HPVoc) = 20.$$

Длина программы:

$$N\ (HPLen) = 34.$$

Объем программы:

$$V\ (HPVol) = 34 \times log_2 20 \approx 34 \times 4{,}3219 \approx 146{,}94.$$

Сложность программы:

$$D\ (HDiff) = 4 \times 1{,}42 \approx 5{,}68.$$

Метрики Холстеда показали, что в программе содержится 20 уникальных операндов и операторов, общая длина программы составляет 34, объем программы равен приблизительно 146,94 бит и сложность программы составляет примерно 5,68. В качестве вывода, можно сказать, что полученные значения характеризуют средний уровень сложности программного кода.

4.4 Интеграционное тестирование

Интеграционное тестирование – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Целью тестирования является выявление багов при взаимодействии между программными модулями и в первую очередь направлена на проверку обмена данными между самими модулями.

Интеграционное тестирование является важной связующей частью жизненного цикла разработки программного обеспечения. Оно проводится перед системным тестированием и после модульного тестирования, где проверяется функциональная корректность самой маленькой части кода, или модуля [34].

Код, выполняющий интеграционное тестирование, представлен в приложении В на листинге В.1 и В.2.

Результат интеграционного тестирования представлен на рисунке 43.

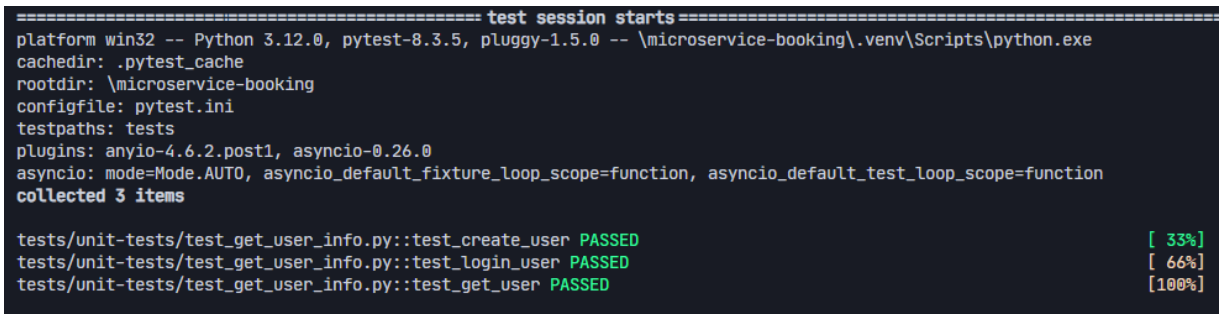

Рисунок 43 – Результат выполнения тестовых функций

Также для лучшего понимания полученных результатов был сгенерирован отчет, представленный на рисунке Ж.1 в приложении Ж.

Исходя из результатов интеграционного тестирования, можно сделать вывод, что функции, предназначенные для обращения к базе данных, корректно отрабатывают и должным образом обрабатывают поступающие данные.

Таким образом, было проведено тестирование черным ящиком, белым ящиком, рассчитаны метрики кода (цикломатическая сложность и метрики Холстеда), а также проведено интеграционное тестирование. Данные тесты показали, что программа корректно реагирует на корректные и не корректные входные данные пользователя.

Заключение

Результатом данной выпускной квалификационной работы является спроектированное и разработанное веб-приложение поиска и резервирования машино-места на автомобильной стоянке.

На стадии анализа требований проведен анализ предметной области, найдены аналогичные сервисы и выполнен их сравнительный анализ приложений, а также определены функциональные и нефункциональные требования для разработки нового программного обеспечения.

Проведено планирование разработки, а также расчет ее стоимости на основе функциональных требований. Затраты на проект составили 439120 рублей. Начало работ определено как 8 апреля, а окончание работ по проекту запланировано на 2 июня текущего года.

На стадии проектирования определена архитектура программного обеспечения, выделены классы разрабатываемого приложения, а также выявлены требования к интерфейсу и разработаны концептуальная и логическая модели базы данных.

В результате разработки веб-приложения поиска и резервирования машино-места на автомобильной стоянке реализована реляционная база данных в MS SQL Server. Серверная часть включает в себя два микросервиса, написанные на фреймворке FastAPI. А для обеспечения обмена данными между ними настроен брокер сообщений Apache Kafka. Клиентская же часть разработана с использованием динамических страниц, что улучшает взаимодействие пользователя с веб-приложением, благодаря отзывчивости интерфейса. Весь процесс разработки проводился в среде Visual Studio Code. В приложениях В, Г и Е приведены фрагменты кода, а с полным кодом можно ознакомиться на GitHub [35].

По завершении реализации проведен контроль качества разработанного веб-приложения посредством интеграционного тестирования. Оно проводилось с применением библиотеки pytest, использующейся для тестирования Python

кода, результатом успешно пройденные тесты, проверяющие корректность работы сервера. Также применено тестирование белым и черным ящиком показывающее, что алгоритм программы выполняется верно. Помимо этого, сделан подсчет метрик Холстеда, что продемонстрировал среднюю сложность программного кода. Это, в свою очередь, свидетельствует о простоте дальнейшего тестирования и поддержания структуры приложения. Результаты тестирования подтвердили корректность работы приложения и соответствие выявленным функциональным требованиям, анализ метрик показал, что программный код имеет оптимальный уровень сложности.

Таким образом, по итогу выполнения выпускной квалификационной работы разработано веб-приложение поиска и резервирования машино-места на автомобильной стоянке, позволяющее быстро бронировать место, на указанный промежуток времени. За счет простого и понятного интерфейса, как для пользователей, так и для администраторов, и хорошо организованного серверного кода, повышается удобство и эффективность пользования приложением.

Список использованных источников

1. Приложение к месту: как цифровые технологии меняют парковочную инфраструктуру // СБ. Беларусь сегодня [Электронный ресурс]. URL: https://www.sb.by/articles/prilozhenie-k-mestu.html (дата обращения: 13.05.2025).

2. Проектирование автомобильных стоянок: виды, нюансы и особенности // ПИР27 [Электронный ресурс]. URL: https://pir27.ru/blog/proektirovanie_avtomobilnykh_stojanok_vidy_njuansy_i_osobennosti/2023-09-25-62 (дата обращения: 13.05.2025).

3. Ответы на часто задаваемые вопросы // ПаркУлей [Электронный ресурс]. URL: https://www.parkuley.ru/faq#general (дата обращения: 14.05.2025)

4. Паркли: решения для паркингов и автоматизации паровок // Parking Expo Russia [Электронный ресурс]. URL: https://parking-expo.ru/ru-RU/press/news/7486.aspx (дата обращения: 14.04.2025).

5. Отзывы о приложении Parkly // RuStore [Электронный ресурс]. URL: https://www.rustore.ru/catalog/app/ru.parkly.app/reviews (дата обращения: 14.05.2025).

6. Отзывы о сервисе Parkly // ТБанк [Электронный ресурс]. URL: https://www.tbank.ru/reviews/company/parkly/102337/ (дата обращения: 14.05.2025).

7. Официальный сайт сервиса бронирования парковок // GetPark [Электронный ресурс]. URL: https://getpark.ru/ (дата обращения: 14.05.2025).

8. Что такое микросервисная архитектура // Selectel [Электронный ресурс]. URL: https://selectel.ru/blog/what-is-microservice-architecture/ (дата обращения: 16.05.2025).

9. FastAPI на Python: что это такое, как работает и зачем нужен // GitVerse [Электронный ресурс]. URL: https://gitverse.ru/blog/articles/development/169-fastapi-na-python-chto-eto-takoe-kak-rabotaet-i-zachem-nuzhen (дата обращения: 17.05.2025).

10. Аутентификация и авторизация // Hexlet [Электронный ресурс]. URL: https://ru.hexlet.io/courses/go-web-development/lessons/auth/theory_unit (дата обращения: 17.05.2025).

11. Что такое хеширование // Hexlet [Электронный ресурс]. URL: https://ru.hexlet.io/blog/posts/chto-takoe-heshirovanie (дата обращения: 18.05.2025).

12. Современные подходы к проектированию API // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/842056/ (дата обращения: 19.05.2025).

13. Apache Kafka: что это и как работает // Selectel [Электронный ресурс]. URL: https://selectel.ru/blog/apache-kafka/ (дата обращения: 18.05.2025).

14. Бюджет проекта: оценка стоимости, методы и примеры контроля // 1С-Битрикс [Электронный ресурс]. URL: https://penza.1cbit.ru/blog/byudzhet-proekta-otsenka-stoimosti-metody-i-primery-kontrolya/ (дата обращения: 15.05.2025).

15. Разбираем диаграмму Ганта: инструмент, который должен знать каждый менеджер // Skillbox [Электронный ресурс]. URL: https://skillbox.ru/media/management/razbiraem-diagrammu-ganta-instrument-kotoryy-dolzhen-znat-kazhdyy-menedzher/#stk-1 (дата обращения: 15.05.2025).

16. Обзор MS Project: возможности и функции // Sky.pro [Электронный ресурс]. URL: https://sky.pro/wiki/profession/obzor-ms-project-vozmozhnosti-i-funkcii/ (дата обращения: 16.05.2025).

17. Что такое MVC // Elbrus Bootcamp [Электронный ресурс]. URL: https://elbrusboot.camp/blog/chto-takoie-mvc/ (дата обращения: 19.05.2025).

18. Учебное пособие по диаграмме развёртывания // Creately [Электронный ресурс]. URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/ (дата обращения: 20.05.2025).

19. Руководство по ERD-диаграммам // Lucidchart [Электронный ресурс]. URL: https://www.lucidchart.com/pages/ru/erd-diagram (дата обращения: 21.05.2025).

20. База данных микросервисов // AppMaster [Электронный ресурс]. URL: https://appmaster.io/ru/glossary/baza-dannykh-mikroservisov (дата обращения: 22.05.2025).

21. Принципы проектирования REST API // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/556790/ (дата обращения: 23.05.2025).

22. Требования к современным пользовательским интерфейсам // КиберЛенинка [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/trebovaniya-k-sovremennym-polzovatelskim-interfeysam/viewer (дата обращения: 23.05.2025).

23. О компании PayMaster // PayMaster [Электронный ресурс]. URL: https://info.paymaster.ru/o-kompanii/ (дата обращения: 24.05.2025).

24. Физическая модель базы данных // Викиконспекты НИУ ВШЭ [Электронный ресурс]. URL: https://neerc.ifmo.ru/wiki/index.php?title=Физическая_модель_базы_данных (дата обращения: 25.05.2025).

25. Как использовать Chart.js для интерактивной визуализации данных // Genius Courses [Электронный ресурс]. URL: https://genius.courses/как-использовать-chart-js-для-интерактивной/ (дата обращения: 25.05.2025).

26. Какие бывают этапы и виды тестирования: подробный разбор // Hexlet [Электронный ресурс]. URL: https://ru.hexlet.io/blog/posts/vidy-testirovaniya#chto-takoe-testirovanie (дата обращения: 26.05.2025).

27. Что такое Black Box Testing? // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/ruvds/articles/700858/ (дата обращения: 26.05.2025)

28. Первые шаги в тест-дизайне: Разбираемся в базовых техниках QA// Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/768984/ (дата обращения: 27.05.2025).

29. Техника анализа классов эквивалентности // QA Evolution [Электронный ресурс]. URL: https://qaevolution.ru/testovaya-dokumentaciya/test-dizajn/texnika-analiza-klassov-ekvivalentnosti/ (дата обращения: 27.05.2025).

30. Тестирование методом белого ящика (White Box Testing)// QA Bible [Электронный ресурс]. URL: https://vladislaveremeev.gitbook.io/qa_bible/vidy-metody-urovni-testirovaniya/testirovanie-metodom-belogo-yashika-white-box-testing (дата обращения: 28.05.2025).

31. Белый ящик Пандоры // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/jugru/articles/354434/ (дата обращения: 29.05.2025)

32. Программный код и его метрики // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/intel/articles/106082/ (дата обращения: 29.05.2025).

33. Граф потока управления // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Граф_потока_управления (дата обращения: 29.05.2025).

34. Интеграционное тестирование: что это? Виды, примеры.// Logrocon [Электронный ресурс]. URL: https://logrocon.ru/news/intgration_testing (дата обращения: 29.05.2025).

35. Репозиторий microservice-booking // GitHub [Электронный ресурс] https://github.com/Refret28/microservice-booking.git (дата обращения: 30.05.2025).

Приложение А

(обязательное)

Листинг серверной части приложения

Листинг А.1 – Модуль db_conn.py

```
config = configparser.ConfigParser()
config.read('config.ini')
server = config['BD INFO']['SERVER']
username = config['BD INFO']['USERNAME']
driver = config['BD INFO']['DRIVER']

class DB_connection:
    def __init__(self, db_name):
        self.username = username
        self.server = server
        self.db_name = db_name
        self.driver = driver
        self.database_url = (
            f"mssql+aioodbc://{self.username}@{self.server}/{self.
db_name}?driver={self.driver}&trusted_connection=yes"
        )
        self.engine = create_async_engine(self.database_url,
echo=True)
        self.session_maker = async_sessionmaker(self.engine,
expire_on_commit=False)

    async def get_session(self):
        async with self.session_maker() as session:
            yield session

    async def execute_query(self, query, params=None):
        async with self.session_maker() as session:
            result = await session.execute(text(query), params)
            return result.fetchall()

    async def close(self):
        await self.engine.dispose()
```

Листинг А.2 – Модуль user_router.py

```
router = APIRouter()
templates = Jinja2Templates(directory="templates")

config = configparser.ConfigParser()
config.read('config.ini')
server = config['BD INFO']['USER_DB']
pay_server = config['BD INFO']['PAY_DB']
kafka_server = config['KAFKA SETTINGS']['KAFKA_SERVER']
topic_name = config['KAFKA SETTINGS']['TOPIC_NAME']

user_db = DB_connection(server)
pay_db = DB_connection(pay_server)

def get_user_repository() -> UserRepository:
    return UserRepository(user_db)
```

Листинг А.3 – Модуль user_models.py

```
class Base(DeclarativeBase):
    pass

class User(Base):
    __tablename__ = "Users"

    UserID: Mapped[int] = mapped_column(Integer, primary_key=True,
autoincrement=True)
    Username: Mapped[str] = mapped_column(String(50), unique=True,
nullable=False)
    Password: Mapped[str] = mapped_column(String(255),
nullable=False)
    Email: Mapped[str] = mapped_column(String(255), unique=True,
nullable=False)
    PhoneNumber: Mapped[str] = mapped_column(String(20))
    Created: Mapped[DateTime] = mapped_column(DateTime,
default=func.now())
    Status: Mapped[str] = mapped_column(String(10),
nullable=False, default="White")

    bookings: Mapped[list["Booking"]] = relationship("Booking",
back_populates="user")
    user_role_mappings: Mapped[list["UserRoleMapping"]] =
relationship("UserRoleMapping", back_populates="user")
    cancelled_bookings: Mapped[list["CancelledBooking"]] =
relationship("CancelledBooking", back_populates="user")
```

Листинг А.4 – Модуль user_schemes.py

```
import re

from pydantic import BaseModel, Field, field_validator, ConfigDict
from typing import Optional

class Token(BaseModel):
    access_token: str
    token_type: str
```

Листинг А.5 – Модуль users/db_manager.py

```
logging.basicConfig()
logging.getLogger("sqlalchemy.engine").setLevel(logging.INFO)
logging.getLogger("sqlalchemy.dialects").setLevel(logging.DEBUG)

config = configparser.ConfigParser()
config.read('config.ini')
SECRET_KEY = config['SECRET KEY']['SECRET_KEY']
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
```

```
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class UserRepository:
    def __init__(self, db_connection: DB_connection):
        self.db_connection = db_connection
```

Листинг А.6 – Модуль pay_router.py

```
import configparser
import asyncio
import json

from fastapi import FastAPI, APIRouter, HTTPException, Depends,
Request
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.background import BackgroundTasks
from aiokafka import AIOKafkaConsumer
from starlette.templating import _TemplateResponse

router = APIRouter()

templates = Jinja2Templates(directory="templates")

config = configparser.ConfigParser()
config.read('config.ini')
server = config['BD INFO']['PAY_DB']
kafka_server = config['KAFKA SETTINGS']['KAFKA_SERVER']
topic_name = config['KAFKA SETTINGS']['TOPIC_NAME']
telegram_bot_username = config['API TOKEN']['BOT_USERNAME']

user_booking_cache = {}

pay_db = DB_connection(server)

def get_pay_repository() -> PayRepository:
    return PayRepository(pay_db)
```

Листинг А.7 – Модуль main.py

```
from contextlib import asynccontextmanager
from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles
import uvicorn

from users.user_router import router as user_router
from payments.pay_router import router as pay_router
from admin.admin_router import router as admin_router
from scheduler import setup_scheduler, scheduler,
user_db_connection, pay_db_connection
from logging_manager import logger

@asynccontextmanager
```

```
async def lifespan(app: FastAPI):
    setup_scheduler()

    try:
        yield
    finally:
        await shutdown()

async def shutdown():
    logger.info("Shutting down application")
    scheduler.shutdown(wait=False)
    await user_db_connection.close()
    await pay_db_connection.close()
    logger.info("Application shutdown complete")

app = FastAPI(lifespan=lifespan)

app.mount("/static", StaticFiles(directory="static"),
name="static")
app.include_router(user_router)
app.include_router(pay_router)
app.include_router(admin_router)

if __name__ == '__main__':
    uvicorn.run("main:app", host="0.0.0.0", port=80, reload=True)
```

Приложение Б

(обязательное)

Листинг клиентской части приложения

Листинг Б.1 – Модуль register_pg.js

```
document.addEventListener("DOMContentLoaded", () => {
    console.log("register.js загружен успешно");

    const loginForm = document.getElementById("login-form");
    const registerForm = document.getElementById("register-form");

    // Проверка наличия уведомления в DOM
    const notification =
document.getElementById('authErrorNotification');
    if (!notification) {
        console.error("Элемент authErrorNotification не найден в
DOM");
    } else {
        console.log("Элемент authErrorNotification найден");
    }

    // Функция для показа всплывающего уведомления
    function showAuthErrorNotification(message) {
        console.log('Вызов showAuthErrorNotification:', message);
        if (!notification) {
            console.error('Уведомление authErrorNotification не
найдено');
            return;
        }
        notification.textContent = message;
        notification.classList.remove('hide');
        notification.classList.add('show');
        setTimeout(() => {
            notification.classList.remove('show');
            notification.classList.add('hide');
            console.log('Уведомление скрыто');
        }, 5000); // Увеличено до 5 секунд для читаемости
    }
```

Листинг Б.2 – Модуль main_pg.js

```
let openModals = 0;
let scrollPosition = 0;
let currentFloorIndex = 0;
let availableFloors = [];
let allSpots = [];
let parkingPrices = [];
let selectedSpot = null;

function showSpotErrorNotification() {
    console.log('Вызов showSpotErrorNotification');
    const notification =
document.getElementById('spotErrorNotification');
    if (!notification) {
        console.error('Уведомление spotErrorNotification не
найдено');
```

```
        return;
    }
    console.log('Показываем уведомление: Пожалуйста, выберите
парковочное место');
    notification.classList.remove('hide');
    notification.classList.add('show');
    setTimeout(() => {
        notification.classList.remove('show');
        notification.classList.add('hide');
        console.log('Уведомление скрыто');
    }, 1000);
}
```

Листинг Б.3 – Модуль usr-acc.js

```
let currentBookingId = null;

function viewPaymentDetails(bookingId) {
    fetch(`/booking_details/${bookingId}`, {
        method: 'GET',
        headers: { 'Content-Type': 'application/json' }
    })
    .then(response => {
        if (response.ok) {
            window.location.href =
`/booking_details/${bookingId}`;
        } else {
            return response.json().then(errorData => {
                if (errorData.detail === "Данные о бронировании не
найдены") {
                    const notification =
document.getElementById('deletingNotification');
                    notification.style.display = 'block';
                    setTimeout(() => {
                        notification.style.display = 'none';
                    }, 1000);
                } else {
                    console.error('Ошибка:', errorData);
                    alert('Ошибка при загрузке данных о
бронировании');
                }
            });
        }
    })
    .catch(error => {
        console.error('Ошибка при отправке запроса:', error);
        alert('Ошибка при загрузке данных о бронировании');
    });
}
```

Листинг Б.4 – Модуль payment.js

```
const formContainer = document.getElementById('form-container');
const formTitle = document.getElementById('form-title');
```

```
const form = document.getElementById('form');
const amount = window.amount || 0;
const user_id = parseInt(window.user_id || getUserIdFromUrl()); //
Преобразуем в число

form.addEventListener('click', async (event) => {
    if (event.target.id === 'submit-data') {
        const carNumberInput =
document.getElementById('car_number');
        const carBrand = document.getElementById('car_brand');
        const carNumber = carNumberInput.value;
        const carBrandValue = carBrand.value;

        const isValidCarNumber = /^[0-9a-яA-
Я]{6,10}$/.test(carNumber);

        if (!isValidCarNumber) {
            carNumberInput.setCustomValidity('Номер автомобиля
должен быть длиной 6 или 10 символов (включая буквы и цифры).');
            carNumberInput.reportValidity();
            return;
        }

        if (!carBrandValue) {
            carBrand.setCustomValidity('Пожалуйста, выберите марку
автомобиля.');
            carBrand.reportValidity();
            return;
        }

        if (isNaN(user_id)) {
            console.error("Неверный user_id:", user_id);
            showNotification("Ошибка: Неверный идентификатор
пользователя");
            return;
        }

        const data = {
            car_number: carNumber,
            car_brand: carBrandValue,
            user_id: user_id
        };
```

Листинг Б.5 – Модуль wait_pg.js

```
const path = window.location.pathname;
const userId = path.split('/').pop();
console.log("UserId:", userId)

let countdown = 3;

function updateCountdown() {
    document.getElementById('countdown-timer').innerText =
countdown;
```

```
    countdown--;

    if (countdown < 0) {
        window.location.href = `/main_page?user_id=${userId}`;
    }
}

setInterval(updateCountdown, 1000);
```

Листинг Б.6 – Модуль admin_pg.js

```
document.addEventListener("DOMContentLoaded", () => {
    if (localStorage.getItem('reloadAfterBooking')) {
        console.log('Обнаружен флаг reloadAfterBooking,
перезагружаем страницу');
        localStorage.removeItem('reloadAfterBooking');
        location.reload();
    }

    const parkingSelect = document.getElementById("parking-
select");
    const floorButtonsContainer = document.getElementById("floor-
buttons");
    const spotsTablesContainer = document.getElementById("spots-
tables");
    const notification = document.getElementById("notification");
    const usersTableBody = document.getElementById("users-table-
body");
    const bookingsModal =
document.getElementById("bookingsModal");
    const bookingsTableBody = document.getElementById("bookings-
table-body");
    const refreshUsersBtn = document.getElementById("refresh-
users-btn");
    const startDateInput = document.getElementById("start-date");
    const endDateInput = document.getElementById("end-date");
    const parkingsChartCanvas = document.getElementById("parkings-
chart");
    const spotsChartCanvas = document.getElementById("spots-
chart");
    const revenueChartCanvas = document.getElementById("revenue-
chart");
```

Приложение В

(обязательное)

Листинг интеграционных тестов

Листинг В.1 – Модуль test_get_user_info.py

```
import uuid
import pytest
from httpx import AsyncClient
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy.future import select
from app.test_users.models import User
from app.test_users.database import get_db
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

@pytest.fixture(scope="module")
async def created_user(test_client: AsyncClient, db_session:
AsyncSession):
    unique_email = f"test_{uuid.uuid4()}@example.com"
    user_data = {
        "Username": f"test_user_{uuid.uuid4().hex[:8]}",
        "Email": unique_email,
        "PhoneNumber": "+123456789",
        "Password": "testpassword123"
    }

    async def override_get_db():
        yield db_session

    from app.test_main import app
    app.dependency_overrides[get_db] = override_get_db

    response = await test_client.post("/register", json=user_data)
    assert response.status_code == 201
    user_id = response.json()["user_id"]

    yield user_id, user_data

    # Очистка после всех тестов
    result = await
db_session.execute(select(User).filter_by(UserID=user_id))
    db_user = result.scalar_one_or_none()
    if db_user:
        await db_session.delete(db_user)
        await db_session.commit()

    app.dependency_overrides.clear()
```

Листинг В.2 – Модуль conftest.py

```
import pytest
import asyncio
from typing import AsyncGenerator
from httpx import AsyncClient
from sqlalchemy.ext.asyncio import AsyncSession
```

```
from fastapi import FastAPI
from uvicorn import Config, Server
from app.test_users.database import AsyncSessionLocal, get_db

@pytest.fixture(scope="session")
def event_loop():
    loop = asyncio.get_event_loop_policy().new_event_loop()
    yield loop
    loop.close()
```

Приложение Г

(рекомендуемое)

Атрибуты сущностей базы данных

Таблица Г.1 – Атрибуты отношения «User»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
UserID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	1
Username	Логин пользователя	nvarchar	50	a-zA-Z	нет		user1
Password	Пароль пользователя	nvarchar	255	a-zA-Z0-9.-_!#	нет		t23d4
Email	Почта пользователя	nvarchar	255	a-zA-Z0-9.-_	нет		user1@example.com
PhoneNumber	Телефон пользователя	nvarchar	20	Строка, формат: +7XXXXXXXXXX	нет		+79001234567
Created	Дата создания учетной записи	datetime	8		да		2025-04-08 11:30
Status	Статус пользователя (белый или черный список)	nvarchar	10	«White», «Black»	нет		White

Таблица Г.2 – Атрибуты отношения «ParkingLocations»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
LocationID	Уникальный идентификатор инструктора	integer	10	от 1 до 2 147 483 647	нет	PK	1
Address	Адрес автомобильной стоянки	nvarchar	255	а-яА-Я0-9.-,	нет		ул. Володарского, 11
Created	Дата создания записи	datetime	8	Текущая дата и время	да		2025-04-09 10:30

Таблица Г.3 – Атрибуты отношения «ParkingSpots»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
SpotID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	1
LocationID	Идентификатор связанной автомобильной стоянки	integer	4	от 1 до 2 147 483 647	нет	FK	2
SpotNumber	Номер машино-места	nvarchar	50	1-9	нет		3
Floor	Этаж	nvarchar	1	0, 1	да		2
IsAvailable	Доступность места	bit	1	Строка, формат: +7XXXXXXXXXX	нет		1
Price	Стоимость машино-места	varchar	10	0.00-99999999.99	нет		100.25
Created	Дата создания записи	datetime	8	Текущая дата и время	да		2025-04-10 09:25

Таблица Г.4 – Атрибуты отношения «Bookings»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
BookingID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	1
UserID	Идентификатор связанного пользователя	integer	4	от 1 до 2 147 483 647	нет	FK	5

Продолжение таблицы Г.4

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
SpotID	Идентификатор связанного машино-места	integer	4	от 1 до 2 147 483 647	нет	FK	7
StartTime	Время начала бронирования	datetime	50	Дата и время в строковом формате	нет		2025-04-11 13:40
EndTime	Время окончания бронирования	datetime	50	Дата и время в строковом формате	нет		2025-04-12 17:12
Created	Дата создания записи	datetime	8	Текущая дата и время	да		2025-04-12 07:15

Таблица Г.5 – Атрибуты отношения «CancelledBookings»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
CancellationID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	1
BookingID	Идентификатор связанного бронирования	integer	4	от 1 до 2 147 483 647	нет	FK	12
UserID	Идентификатор связанного пользователя	integer	4	от 1 до 2 147 483 647	нет	FK	9
CancellationReason	Причина отмены	nvarchar	255	а-яА-Я,.	нет		Выннужденные обстоятельства
CancellationTime	Дата отмены	datetime	8	Текущая дата и время	нет		2025-04-13 12:15

Таблица Г.6 – Атрибуты отношения «UserRoles»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
RoleID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	24
RoleName	Название роли	nvarchar	50	a-zA-Z	нет		User

Таблица Г.7 – Атрибуты отношения «UserRoleMapping»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
UserRoleMappingID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	15
UserID	Идентификатор связанного пользователя	integer	4	от 1 до 2 147 483 647	нет	FK	25
RoleID	Идентификатор связанной роли	integer	4	от 1 до 2 147 483 647	нет	FK	2

Таблица Г.8 – Атрибуты отношения «Cars»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
CarID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	15
UserID	Идентификатор связанного пользователя	integer	4	от 1 до 2 147 483 647	нет		4

Продолжение таблицы Г.8

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
CarNumber	Автомобильный номер	nvarchar	50	а-яА-Яа-zA-Z0-9	нет		п321го
CarBrand	Марка автомобиля	nvarchar	100	a-zA-Z	нет		Skoda
BookingID	Идентификатор связанного бронирования	integer	4	от 1 до 2 147 483 647	да		8
Created	Дата создания записи	datetime	8	Текущая дата и время	да		2025-04-13 15:35

Таблица Г.9 – Атрибуты отношения «Payments»

Идентификатор столбца	**Описание**	**Тип данных**	**Размерность**	**Область допустимых значений**	**Возможность значения Null**	**Роль**	**Пример**
PaymentID	Уникальный идентификатор инструктора	integer	4	от 1 до 2 147 483 647	нет	PK	15
BookingID	Идентификатор связанного бронирования	integer	4	от 1 до 2 147 483 647	нет	FK	25
TransactionID	Идентификатор транзакции	nvarchar	100	от 1 до 2 147 483 647	нет	FK	2
PaymentStatus	Статус оплаты	nvarchar	50	Completed	нет		Completed
Status	Статус записи	nvarchar	15	Completed	нет		Completed
Amount	Сумма оплаты	numeric	10	0.01-99999999.99	нет		150.00
Created	Дата выполнения оплаты	datetime	8	Текущая дата и время	да		2025-04-15 16:40

Приложение Д

(рекомендуемое)

Графическая часть

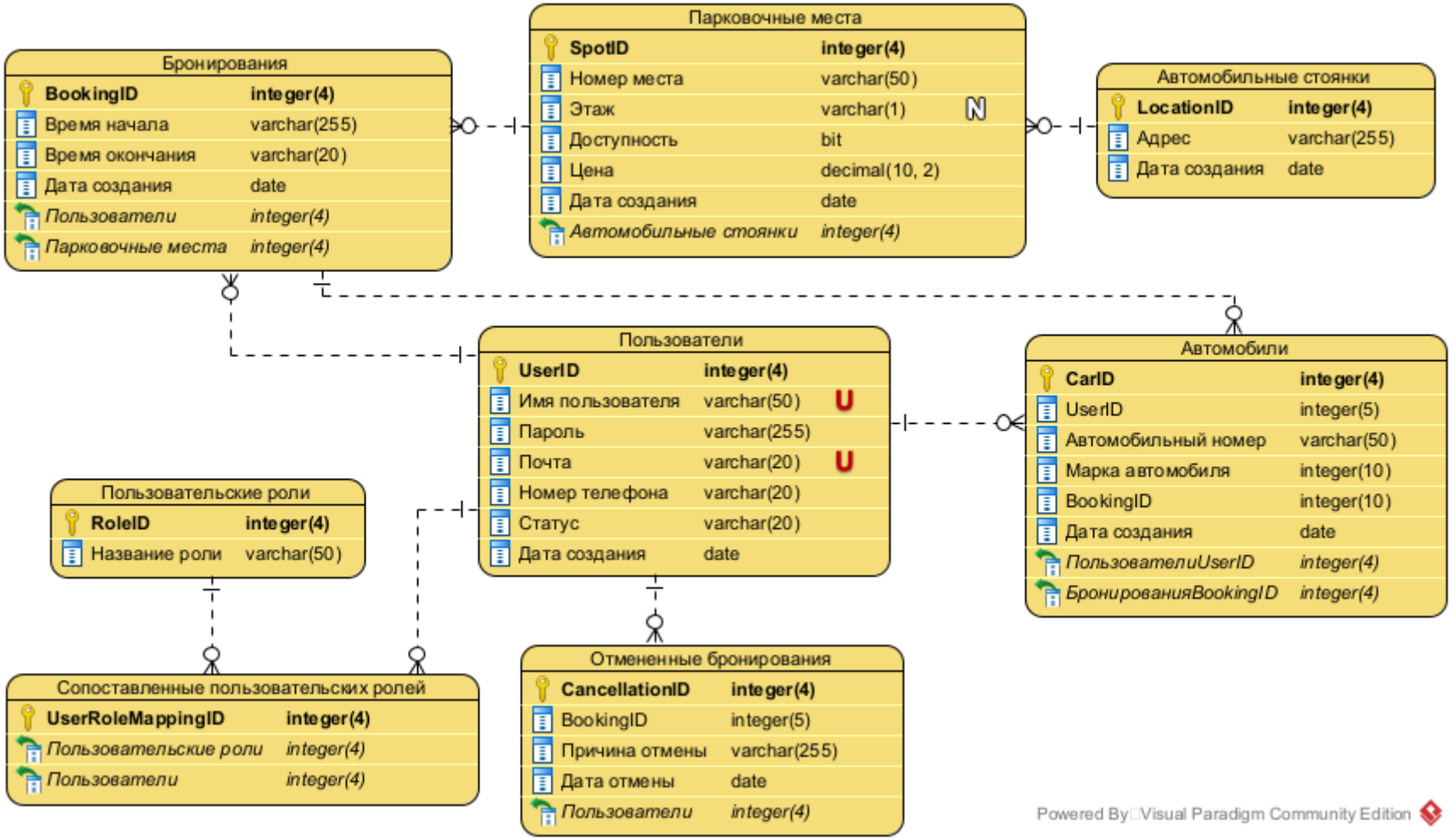

Рисунок Д.1 – Логическая модель базы данных первого микросервиса

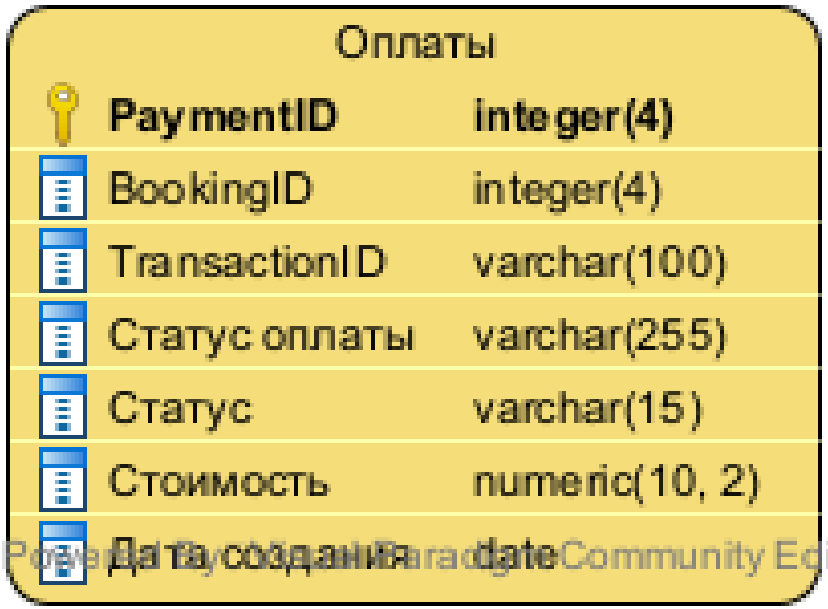

Рисунок Д.2 – Логическая модель базы данных второго микросервиса

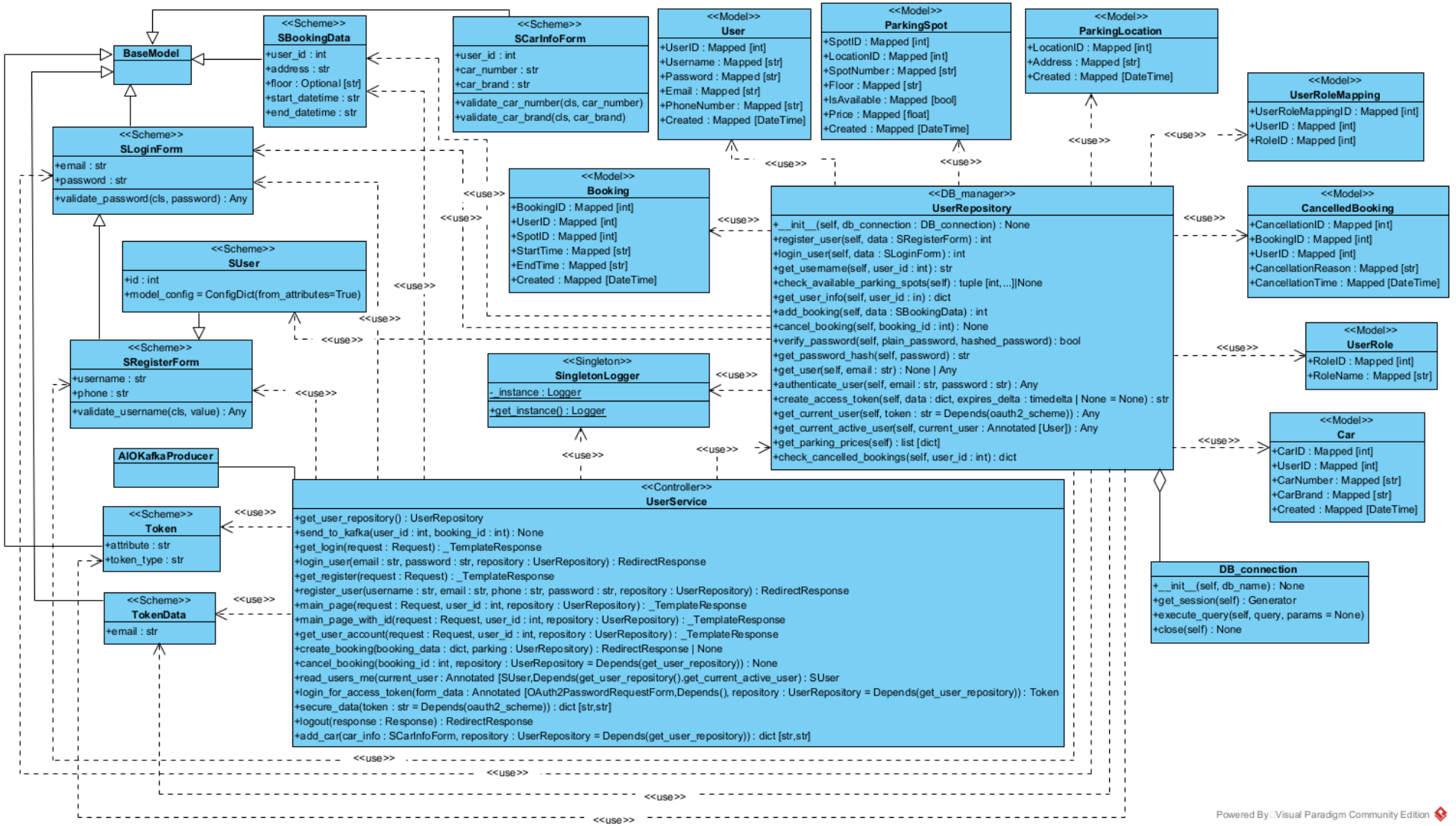

Рисунок Д.3 – Диаграмма классов микросервиса для работы с пользователями, их бронированиями и автомобильными стоянками

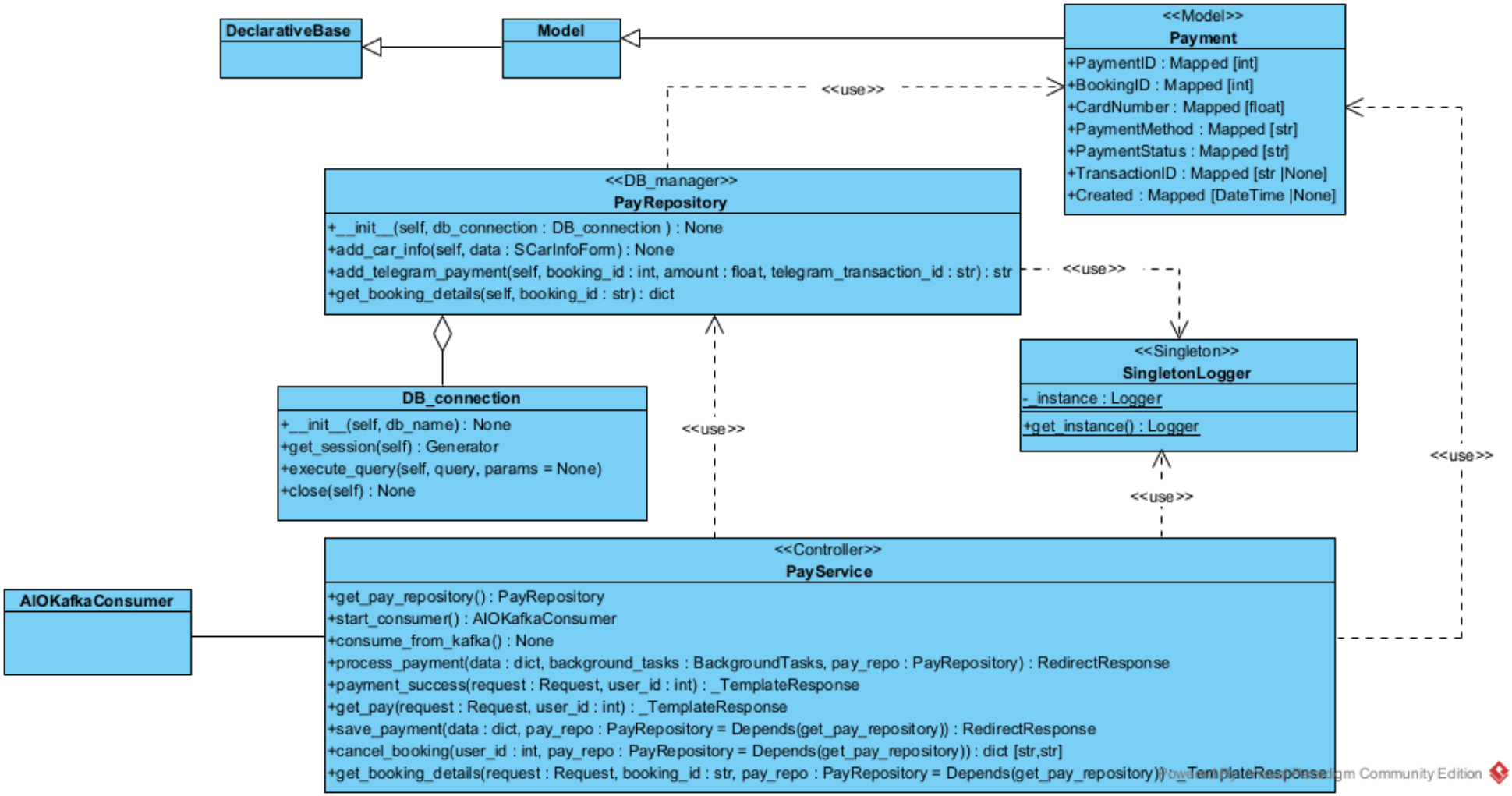

Рисунок Д.4 – Диаграмма классов микросервиса для работы с оплатой

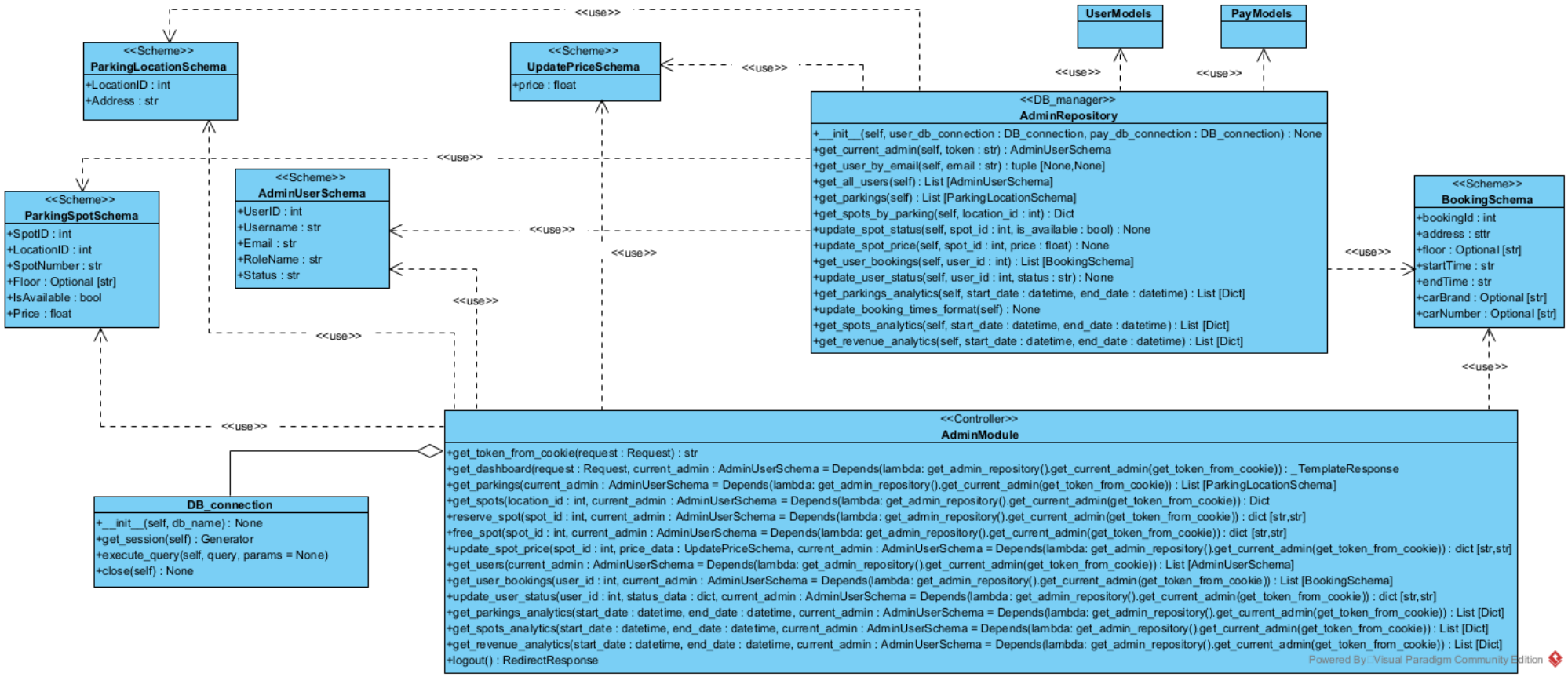

Рисунок Д.5 – Диаграмма классов модуля для работы администратора

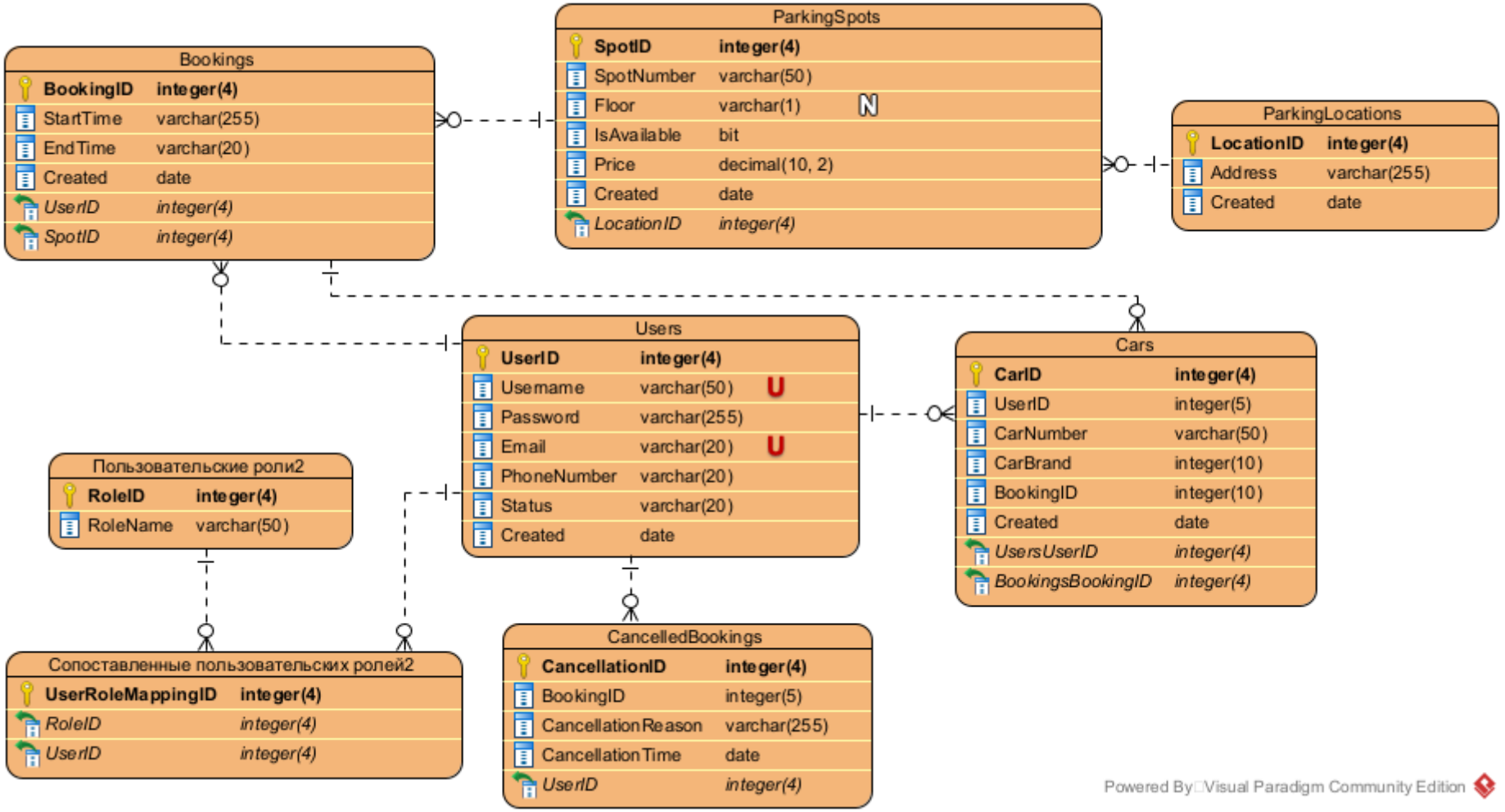

Рисунок Д.6 – Физическая модель базы данных первого микросервиса

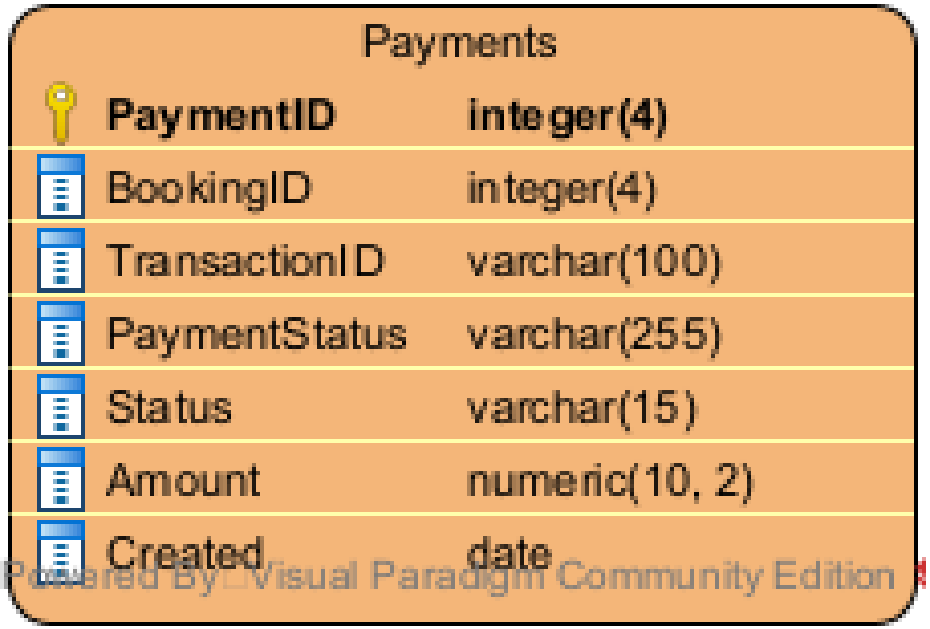

Рисунок Д.7 – Физическая модель базы данных второго микросервиса

Приложение Е

(рекомендуемое)

Результаты функционального тестирования

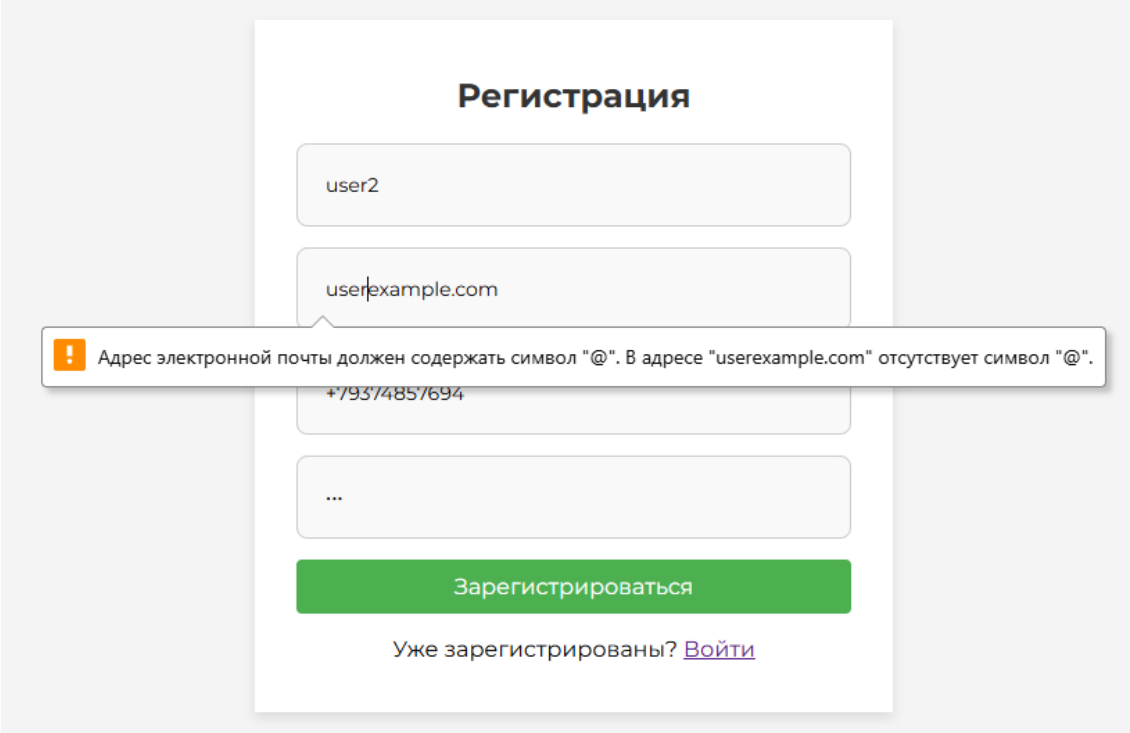

Рисунок Е.1 – Попытка зарегистрироваться с неккоректным почтовым адресом

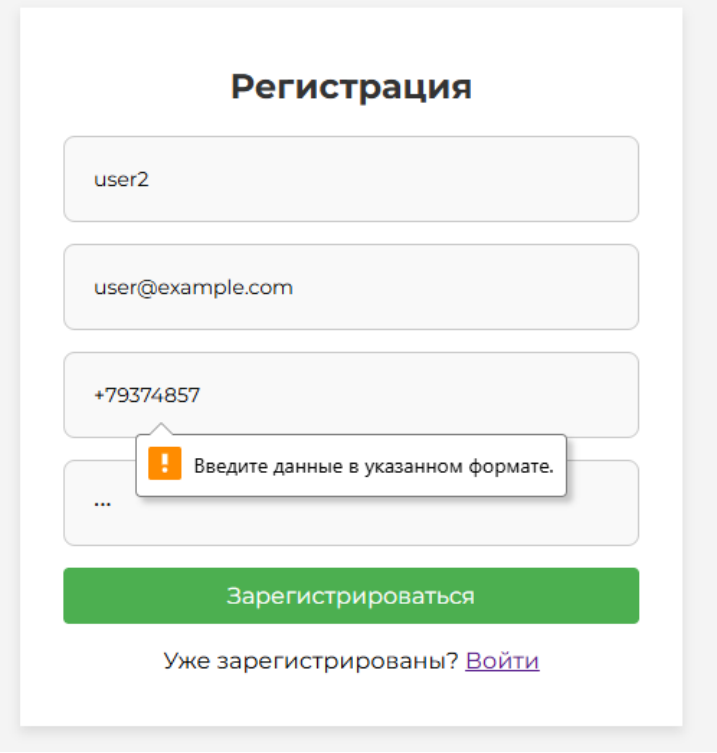

Рисунок Е.2 – Попытка зарегистрироваться с неккоректным номером телефона

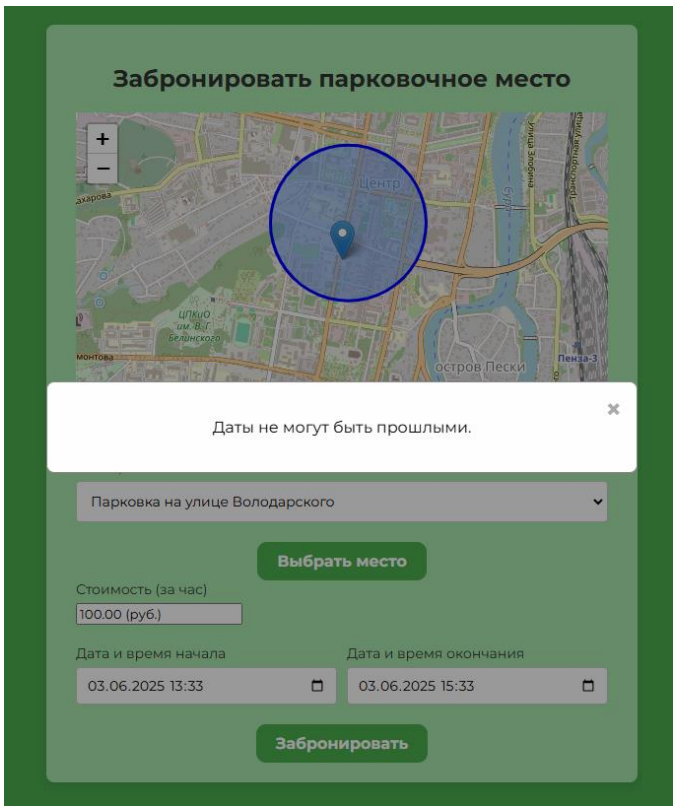

Рисунок Е.3 – Попытка создать бронирование с указанием прошлых дат

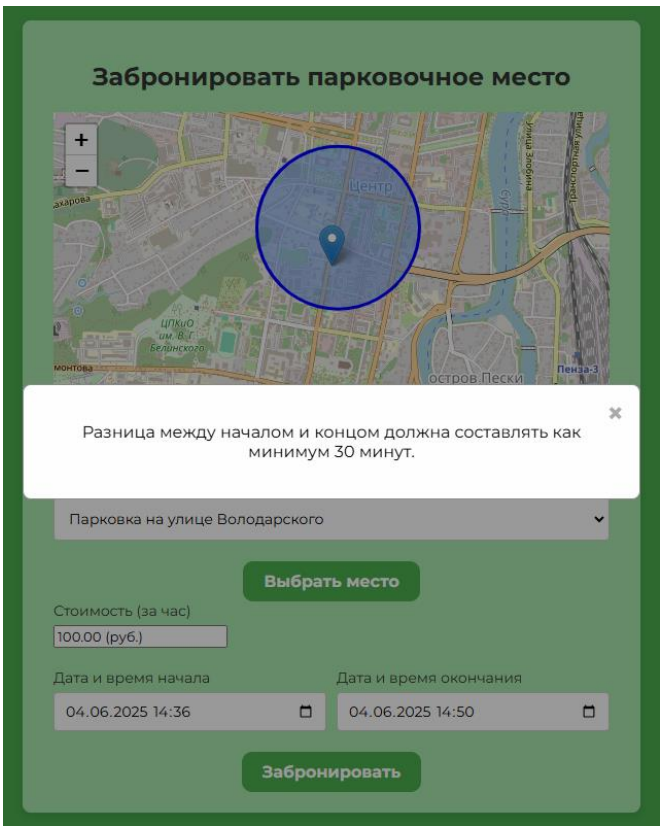

Рисунок Е.4 – Попытка создать бронирование с указанием дат с разницей менее 30 минут

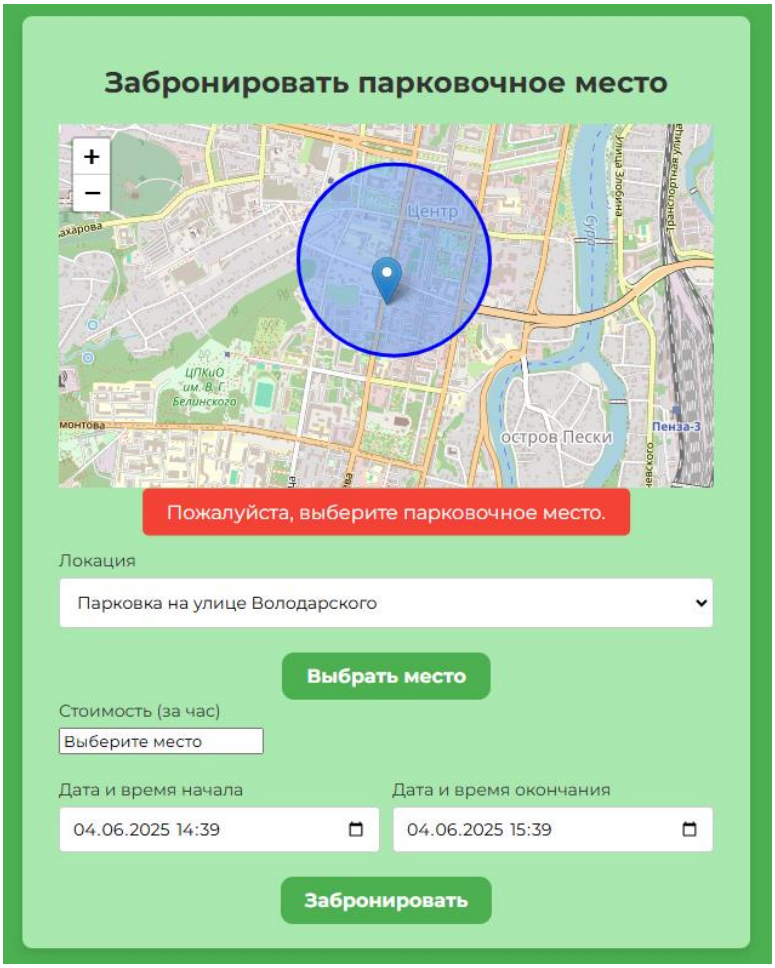

Рисунок Е.5 – Попытка создать бронирование без выбранного места

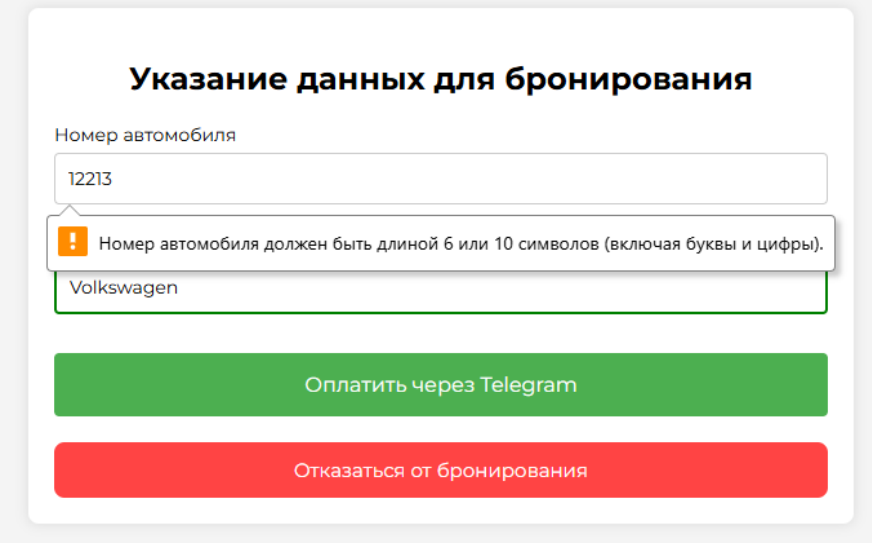

Рисунок Е.6 – Попытка указать неккоректный номер автомобиля

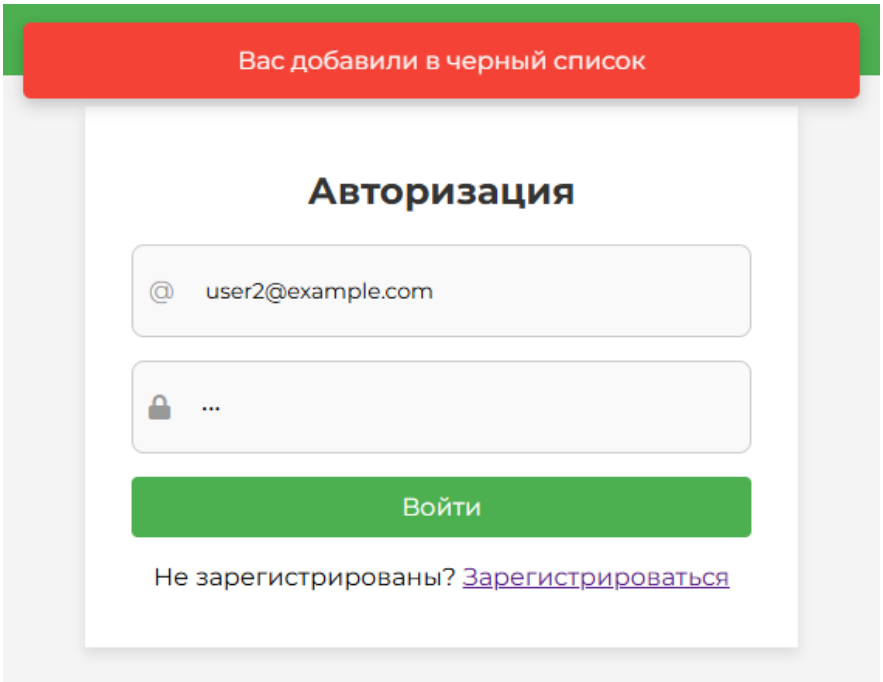

Рисунок Е.7 – Попытка авторизации после занесения в черный список

Приложение Ж

(рекомендуемое)

Отчет по интегрционному тестированию

результаты.html

Отчет создан 01-июня-2025 в 16:09:32 пользователем pytest-html v4.1.1

Среда

Питон	3.12.0
Платформа	Windows-11-10.0.22631-SP0
Пакеты	• pytest: 8.3.5 • плагин: 1.5.0
Плагины	• любой: 4.6.2.post1 • асинхро: 0.26.0 • html: 4.1.1 • метаданные: 3.1.1
JAVA_HOME	C:\Program Files\Java\jdk-21

Краткое содержание

3 теста заняли 00:00:04.

(Снимите)флажки, чтобы отфильтровать результаты.

0 Не удалось, 3 Пройдено, 0 Пропущено, 0 Ожидаемые неудачи, 0 Неожиданные пасы, 0 Ошибок, 0 Повторов

Показать все детали / Скрыть все детали

Результат	Тест	Продолжительность	Ссылки
Прошедший	тесты/unit-tests/test_get_user_info.py::test_create_user	00:00:03	
Прошедший	тесты/unit-tests/test_get_user_info.py::test_login_user	366 мс	
Прошедший	тесты/unit-тесты/test_get_user_info.py::test_get_user	00:00:01	

Рисунок Ж.1 – Отчет по интегрционному тестированию