

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Автоматизированная система
информирования, бронирования и управления
услугами для отдыхающих**

Студент

(подпись, дата)

Алёшин Вячеслав Сергеевич
(ФИО полностью)

Руководитель

(подпись, дата)

Попова Н.А.

(фамилия, инициалы)

Нормоконтролер

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026 г. №12)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Алёшин Вячеслав Сергеевич гр. 22ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Автоматизированная система информирования, бронирования и
управления услугами для отдыхающих

Тема утверждена приказом ПГУ ПГУ №1583/о от "30" декабря 2025 г.

3. Руководитель работы к.т.н. доцент Попова Н.А

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

Назначение: Автоматизированная система информирования, бронирования и
управления услугами для отдыхающих.

2) *Среда выполнения:* Windows 11.

Языковые средства разработки: C#, HTML, CSS, JavaScript, SQL, UML.

Инструментальные средства: Visual Studio 2022, SQL Server Management
Studio 20, Visual Paradigm 17.0.

Технологии разработки: ООП, многослойная архитектура.

3) *Основные функции приложения:*

а) авторизация пользователей с разграничением прав доступа;

б) просмотр проживания и дополнительных услуг;

в) бронирование домиков, номеров и услуг;

г) управление личными бронированиями отдыхающего;

д) управление вариантами проживания и услугами (создание, редактирование,
удаление);

е) управление расписанием и заказами услуг;

ж) отправка уведомлений и напоминаний по email.

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%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «20» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

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

Реферат

Пояснительная записка содержит 129 листа, 56 рисунков, 33 таблицы, 22 использованных источника и 7 приложений.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА, ВЕБ-ПРИЛОЖЕНИЕ, БРОНИРОВАНИЕ, УПРАВЛЕНИЕ УСЛУГАМИ, МНОГОСЛОЙНАЯ АРХИТЕКТУРА, UML, VISUAL STUDIO 2022, C#, HTML, CSS, JAVASCRIPT, БАЗА ДАННЫХ, SQL SERVER.

Объект исследования – процессы информирования отдыхающих, бронирования проживания и заказа дополнительных услуг.

Цель выпускной квалификационной работы – разработать автоматизированную систему информирования, бронирования и управления услугами для отдыхающих.

Технологии разработки – система реализована в среде Visual Studio 2022 с использованием ASP.NET Core MVC. Клиентская часть разработана на HTML, CSS и JavaScript, серверная логика – на языке C#. Работа с данными выполнена через Entity Framework Core и СУБД SQL Server. Для моделирования применялись UML-диаграммы.

Среда исполнения – Windows 11, веб-браузер.

Результаты работы – спроектирована и реализована веб-система, обеспечивающая просмотр вариантов проживания и услуг, оформление бронирований, заказ дополнительных услуг, демонстрационную оплату, email-уведомления и разграничение доступа по ролям. Проведено тестирование серверной логики и выполнена оценка качества программного кода.

					ПГУ 09.03.02 – 08BP211.01 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Алёшин В.С.			Автоматизированная система информирования, бронирования и управления услугами для отдыхающих Пояснительная записка	Лит.	Лист	Листов
Пров.		Попова Н.А.					4	129
Реценз.						Группа 22ВИ1		
Н. контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку системы бронирования и управления услугами.....	8
1.1 Исследование предметной области.....	8
1.2 Анализ информационных систем аналогичного назначения.....	9
1.3 Анализ функциональных требований.....	13
1.4 Выбор технологий проектирования системы.....	19
1.5 Выбор инструментальных средств разработки.....	20
1.6 Анализ требований к разрабатываемой системе.....	21
2 Проектирование информационной системы бронирования и управления услугами.....	24
2.1 Проектирование структур данных.....	24
2.2 Проектирование архитектуры приложения.....	26
2.3 Проектирование программных средств.....	30
2.4 Проектирование пользовательского интерфейса.....	33
3 Реализация информационной системы бронирования и управления услугами.....	36
3.1 Реализация базы данных.....	36
3.2 Реализация серверной части информационной системы бронирования и управления услугами.....	37
3.3 Реализация клиентской части системы бронирования.....	42
3.4 Реализация интерфейса системы бронирования и управления услугами.....	44
3.4.1 Реализация интерфейса отдыхающего.....	46
3.4.2 Реализация интерфейса администратора.....	50
3.4.3 Реализация интерфейса менеджера услуг.....	53
4 Отладка и тестирование программного обеспечения.....	56
4.1 Функциональное тестирование методом белого ящика.....	56
4.1.1 Покрывание операторов.....	56

4.1.2	Покрытие решений	59
4.1.3	Покрытие условий	62
4.2	Модульные тесты серверной части программного обеспечения.....	64
5	Оценка качества программного кода	67
5.1	Тестирование SLOC листинг сделать ссылкой на приложение	67
5.2	Метрика Холстеда.....	68
5.3	Метрика Чепина	71
5.4	Автоматическая проверка	73
6	Планирование разработки и оценка бюджета.....	75
	Заключение	80
	Список использованных источников	82
	Приложение А (справочное) Глоссарий	85
	Приложение Б (обязательное) Листинг серверной части системы.....	87
	Приложение В (обязательное) Листинг клиентской части системы	96
	Приложение Г (обязательное) Листинг файла BookingServiceTests.cs	102
	Приложение Д (рекомендуемое) Атрибуты сущностей базы данных.....	108
	Приложение Е (рекомендуемое) Графическая часть	115
	Приложение Ж (рекомендуемое) Результаты функционального тестирования.....	122

Введение

Организация отдыха связана с обработкой заявок, информированием клиентов, учетом свободных объектов проживания и планированием дополнительных услуг. При ручном выполнении этих операций возрастает риск ошибок: одно и то же время может быть занято повторно, данные о клиенте теряются, а администратору приходится сверять несколько источников информации.

Практическую значимость имеет разработка веб-системы, которая объединяет сведения о проживании, услугах, бронированиях и оплатах в единой информационной среде. Такое решение позволяет отдыхающему заранее ознакомиться с доступными вариантами размещения, выбрать услугу на нужную дату и получить уведомление по электронной почте. Администратор и менеджер услуг получают инструменты для ведения расписания, контроля заказов и обновления информации.

Объектом исследования выступает процесс информирования, бронирования и управления услугами для отдыхающих. Предметом работы является автоматизация указанных процессов средствами веб-приложения. Цель выпускной квалификационной работы состоит в разработке автоматизированной системы информирования, бронирования и управления услугами для отдыхающих.

Для достижения цели необходимо выполнить следующие задачи:

проанализировать предметную область и определить требования к разрабатываемой системе;

- рассмотреть программные решения аналогичного назначения;
- разработать архитектуру и структуру базы данных;
- реализовать ИС с учетом поставленных требований;
- провести тестирование системы и оценить ее качество.

Реализация этих задач поможет создать удобную систему информирования, бронирования и управления услугами для отдыхающих.

1 Анализ предметной области и постановка задачи на разработку системы бронирования и управления услугами

1.1 Исследование предметной области

Информирование клиентов, бронирование проживания и управление дополнительными услугами являются важными процессами в деятельности базы отдыха. Для качественного обслуживания отдыхающих необходимо хранить сведения о домиках и номерах, свободных датах, перечне услуг, заявках, оплатах, статусах бронирования и сотрудниках, отвечающих за выполнение работ.

На практике подобные процессы нередко ведутся вручную: администратор принимает заявки по телефону или через мессенджеры, фиксирует данные в рисх, отдельно согласовывает услуги с ответственными сотрудниками и вручную проверяет занятость объектов. При увеличении количества обращений такой подход приводит к ошибкам, дублированию заявок и затруднениям при контроле загрузки.

Внедрение автоматизированной системы позволяет объединить основные операции в единой информационной среде: отдыхающий получает сведения о базе отдыха и оформляет заявку через веб-интерфейс, администратор управляет бронированиями проживания, менеджер услуг контролирует расписание дополнительных услуг, а сотрудник видит назначенные задачи.

В сфере автоматизации работы базы отдыха можно выделить несколько направлений применения информационных технологий:

- ведение единой базы объектов проживания с учетом стоимости, вместимости, статуса и доступности по датам;
- управление заявками отдыхающих с разграничением прав доступа по ролям пользователей;

- учет дополнительных услуг по времени, количеству мест и ответственным сотрудникам;
- формирование отчетности по загрузке, бронированиям, заявкам на услуги и оплатам.

Для решения подобных задач существуют коммерческие гостиничные системы и PMS-платформы. Однако такие решения часто ориентированы на широкий гостиничный бизнес, содержат большое количество дополнительных модулей и требуют настройки под конкретные бизнес-процессы. Для небольшой базы отдыха целесообразно спроектировать более компактную систему, ориентированную на характерные сценарии выбранной предметной области.

Главной целью разрабатываемой системы является создание удобного веб-приложения, которое обеспечивает информирование отдыхающих, бронирование проживания и управление дополнительными услугами. Система должна снижать вероятность ошибок при работе с заявками, ускорять поиск свободных объектов и упрощать взаимодействие между отдыхающими и сотрудниками базы отдыха.

Разрабатываемая система ориентирована на использование в базе отдыха, предоставляющей проживание, баню, аренду беседок, питание, массаж, бассейн и локальные концерты. Программный продукт должен иметь публичную часть для отдыхающих и административную часть для сотрудников организации.

1.2 Анализ информационных систем аналогичного назначения

В процессе исследования рынка были выявлены основные системы аналогичного значения: «Вnovo», «1С:Отель».

Первым рассматриваемым продуктом является Vnovo – система управления отелями и гостиницами. Платформа предназначена для автоматизации бронирований, управления загрузкой, работы с каналами

продаж, модулем онлайн-бронирования, отчетностью и дополнительными функциями для объектов размещения [1].

Система обладает широким набором функциональных возможностей:

- управление бронированиями и загрузкой номерного фонда;
- использование модуля онлайн-бронирования для гостей;
- создание и настройка дополнительных, в том числе часовых, услуг;
- разграничение прав доступа пользователей;
- формирование отчетов и аналитики по работе объекта размещения.

Достоинства Vnovo являются развитая функциональность, наличие инструментов для онлайн-продаж и поддержка услуг, привязанных ко времени. Вместе с тем система является коммерческим универсальным продуктом и может содержать функции, избыточные для небольшой базы отдыха. Интерфейс платформы представлен на рисунке 1.

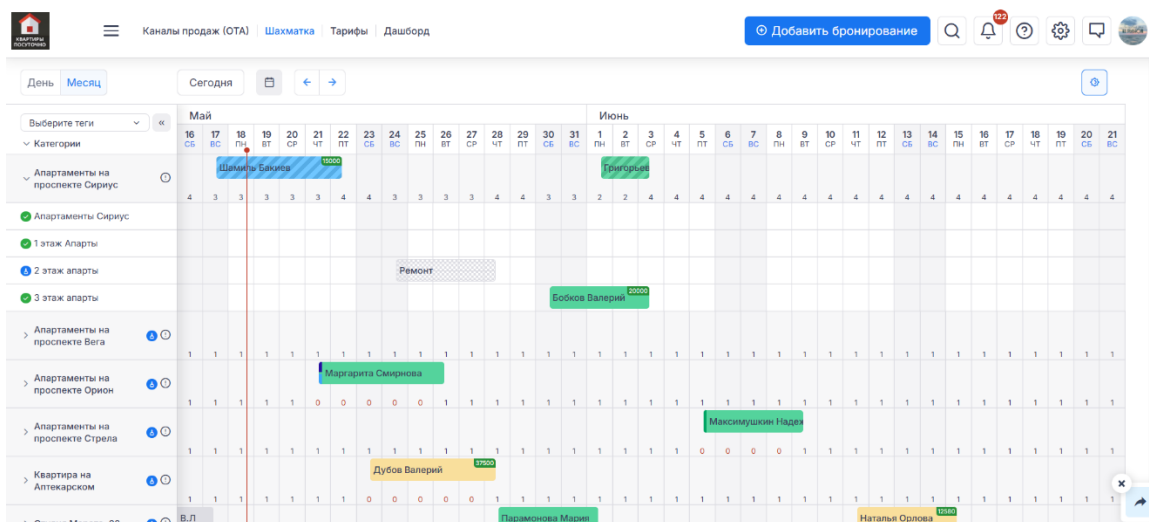


Рисунок 1 – Интерфейс системы Vnovo

Vnovo можно рассматривать как функционально близкий аналог, поскольку система поддерживает онлайн-бронирование, управление размещением, учет услуг, статусы заявок и отчетность. Эти возможности полезны при проектировании системы бронирования для отдыхающих, так как позволяют определить базовый набор функций.

При этом Вnovo ориентирован на более широкий гостиничный цикл. Для разрабатываемой системы требуется более узкая модель данных: отдыхающие, объекты проживания, услуги, бронирования, заявки на услуги, оплаты и роли пользователей. Отдельное значение имеет учет услуг и локальных мероприятий с ограничением по времени и количеству мест, поскольку для выбранной предметной области этот сценарий является самостоятельной функцией.

«1С:Отель» – это специализированная система, предназначенная для автоматизации бизнес-процессов гостиничных сетей и независимых отелей на базе платформы 1С:Предприятие [2]. Система предоставляет следующие возможности:

- ведение номерного фонда и управление бронированиями;
- оформление заселения, продления проживания, переселения и выезда гостей;
- учет дополнительных услуг и связанных с ними операций;
- разграничение прав пользователей и работа с ролями;
- формирование отчетности и расчет гостиничных показателей.

Интерфейс платформы показан на рисунке 2

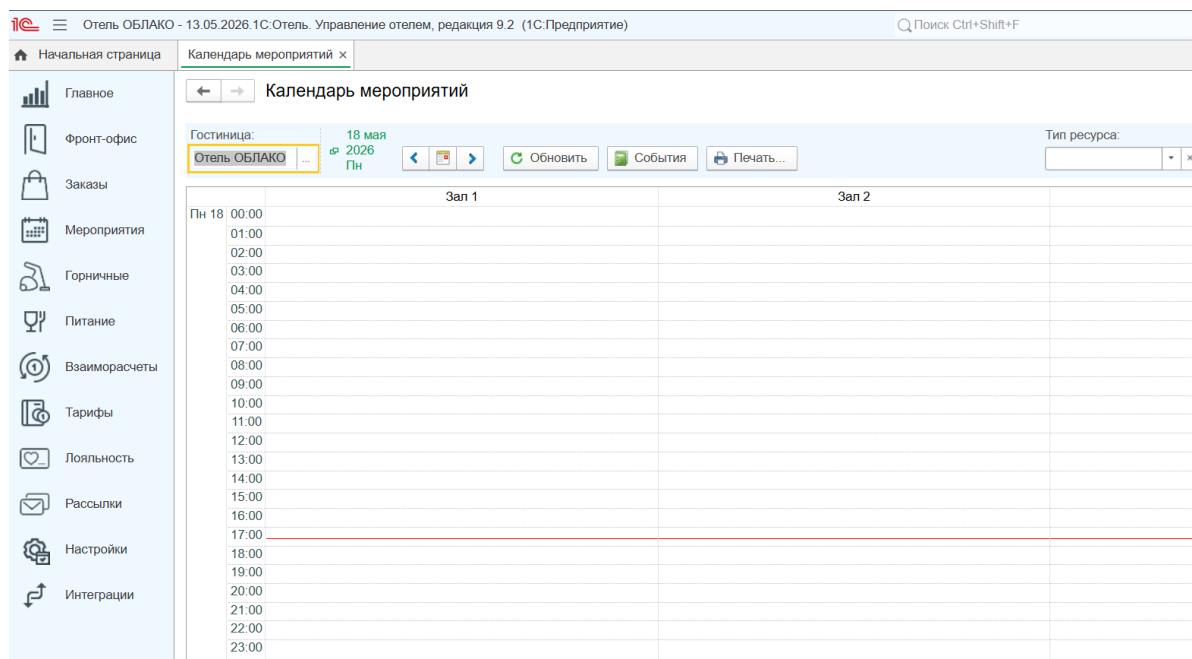


Рисунок 2 – Интерфейс решения 1С:Отель

Несомненными достоинствами продукта являются ориентация на гостиничные процессы, интеграция с экосистемой 1С и широкие возможности настройки. Вместе с тем 1С:Отель может быть сложным для внедрения в небольшой базе отдыха, поскольку решение рассчитано на более широкий набор учетных и управленческих задач.

Для проведения сравнительного анализа информационных систем отелями и гостиницами был определен ряд критериев, на основе которых оценивались существующие решения. Система оценивалась по следующей шкале:

- 0 баллов: функция отсутствует или выполнена неудовлетворительно;
- 1 балл: функция реализована частично или со значительными ограничениями;
- 2 балла: функция реализована полноценно и качественно.

Результаты оценки представлены в таблице 1.

Таблица 1 – Сравнение систем аналогичного назначения

Критерий	Вново	1С:Отель	Разрабатываемая система
Онлайн-бронирование проживания	2	1	2
Информирование о базе отдыха, номерах, услугах и ценах	2	1	2
Управление бронированиями проживания	2	2	2
Бронирование дополнительных услуг	2	2	2
Бронирование услуг по времени	2	2	2
Управление заявками на баню, беседки, массаж и бассейн	1	1	2
Учет локальных мероприятий с ограничением количества мест	1	1	2
Роли пользователей и разграничение доступа	2	2	2
Отчеты по загрузке, заявкам и услугам	2	2	2
Итого	10	5	11

В результате анализа были выявлены сильные стороны существующих аналогов. Вново и 1С:Отель обладают широким набором функций для управления объектами размещения, бронированиями, пользователями и

отчетностью. Разрабатываемая система должна отличаться специализацией под небольшую базу отдыха и заранее определенный набор услуг: проживание, баню, аренду беседок, питание, массаж, бассейн и локальные концерты.

1.3 Анализ функциональных требований

В ходе анализа предметной области и состава задач автоматизированной системы информирования, бронирования и управления услугами для отдыхающих выделены следующие функциональные требования:

- регистрация и авторизация пользователей с разграничением ролей;
- просмотр публичной информации об объектах проживания и дополнительных услугах;
- оформление бронирования проживания с выбором дат, количества гостей и контактных данных;
- автоматическая проверка занятости объекта проживания перед созданием брони;
- оформление заказа дополнительной услуги на выбранную дату и время;
- контроль свободных мест при заказе услуги с ограниченной вместимостью;
- отображение активных и отмененных заявок в личном кабинете отдыхающего;
- имитация оплаты бронирований и заказов услуг с изменением статуса оплаты;
- отправка email-уведомлений после оформления бронирования или заказа услуги;
- ведение администратором пользователей, объектов проживания, бронирований и отчетов;

- ведение менеджером услуг перечня услуг, расписания сеансов и заявок отдыхающих;
- управление фотографиями объектов проживания и дополнительных услуг.

Для формализации указанных требований построены UML-диаграммы вариантов использования. Они показывают, какие действия доступны каждой роли пользователя и как эти действия соотносятся с функциями системы.

Диаграмма вариантов использования описывает функциональность системы с точки зрения внешних участников. В рамках данной работы она применяется для фиксации пользовательских сценариев без привязки к внутренней структуре базы данных и программных классов.

Актёром называется пользовательская роль, которая взаимодействует с системой через её интерфейс. В рамках проекта выделено три актёра: отдыхающий, администратор и менеджер услуг. Прямые связи между актёрами не задаются — каждый из них работает с системой самостоятельно, выполняя собственный набор операций.

Вариант использования — это действие или последовательность действий, которые приводят к наблюдаемому и значимому результату. Каждый вариант использования описывает законченный сценарий: вход в систему, создание брони, заказ услуги, изменение расписания или обработку заявки [3].

Отдыхающий взаимодействует с публичной частью системы и личным кабинетом. Публичная часть доступна без авторизации и позволяет свободно просматривать каталог вариантов проживания и перечень услуг. После регистрации и авторизации отдыхающий попадает в личный кабинет, где ему становятся доступны основные операции: оформление бронирования проживания, заказ дополнительной услуги, оплата выставленной заявки, отмена брони и просмотр истории своих обращений. Все перечисленные сценарии собраны на диаграмме вариантов использования для отдыхающего, которая приведена на рисунке 3.



Рисунок 3 – Диаграмма вариантов использования (отдыхающий)

Менеджер услуг отвечает за операционные действия, связанные с дополнительными услугами. Он ведет справочник услуг, прикрепляет изображения, создает и редактирует сеансы в календаре, настраивает повторение, удаляет события и просматривает заказы отдыхающих. Диаграмма вариантов использования для менеджера услуг представлена на рисунке 4.



Рисунок 4 – Диаграмма вариантов использования (менеджер услуг)

Администратор получает доступ к общему управлению системой. В его функции входят ведение пользователей, объектов проживания, фотографий, бронирований, заказов услуг, статусов оплаты, календаря сеансов и отчетных данных. Диаграмма вариантов использования для администратора представлена на рисунке 5.



Рисунок 5 – Диаграмма вариантов использования (администратор)

Ниже приведены спецификации основных вариантов использования, отражающих базовые сценарии работы системы.

Спецификация варианта использования «Авторизоваться» приведена на рисунке 6.

Прецедент: Авторизоваться
ID: 1
Краткое описание: Пользователь авторизуется в системе
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: 1. Пользователь не авторизован 2. Открыта страница авторизации
Основной поток: 1. Пользователь вводит email и пароль 2. Пользователь нажимает кнопку «Войти» 3. Система проверяет учетные данные 4. Система открывает раздел, доступный роли пользователя
Постусловия: 1. Пользователь авторизован 2. В системе отображаются доступные ему страницы

Рисунок 6 – Спецификация варианта использования «Авторизоваться»

Спецификация варианта использования «Забронировать проживание» приведена на рисунке 7.

Прецедент: Забронировать проживание
ID: 2
Краткое описание: Отдыхающий оформляет бронь выбранного домика или номера
Главные актеры: Отдыхающий
Второстепенные актеры: Нет
Предусловие: 1. Пользователь авторизован 2. Открыта форма бронирования проживания
Основной поток: 1. Пользователь выбирает объект проживания 2. Пользователь указывает даты заезда и выезда 3. Пользователь вводит количество гостей и комментарий 4. Система проверяет вместимость и занятость объекта 5. Система создает бронь проживания
Постусловия: 1. Бронь сохранена в системе 2. В личном кабинете отображается новая заявка

Рисунок 7 – Спецификация варианта использования «Забронировать проживание»

Спецификация варианта использования «Заказать дополнительную услугу» приведена на рисунке 8.

Прецедент: Заказать дополнительную услугу
ID: 3
Краткое описание: Отдыхающий бронирует услугу на выбранную дату и время
Главные актеры: Отдыхающий
Второстепенные актеры: Нет
Предусловие: 1. Пользователь авторизован 2. Выбран активный сеанс услуги
Основной поток: 1. Пользователь выбирает услугу и сеанс 2. Пользователь указывает количество мест 3. Пользователь при необходимости связывает услугу с бронью проживания 4. Система проверяет наличие свободных мест 5. Система создает заказ услуги
Постусловия: 1. Заказ услуги сохранен 2. Заказ отображается в личном кабинете и панели управления

Рисунок 8 – Спецификация варианта использования «Заказать дополнительную услугу»

Спецификация варианта использования «Управлять расписанием услуг» приведена на рисунке 9.

Прецедент: Управлять расписанием услуг
ID: 4
Краткое описание: Менеджер услуг или администратор ведет календарь сеансов
Главные актеры: Менеджер услуг, администратор
Второстепенные актеры: Нет
Предусловие: 1. Пользователь авторизован 2. Открыт календарь расписания услуг
Основной поток: 1. Пользователь выбирает дату и время в календаре 2. Пользователь указывает услугу, длительность и вместимость 3. Пользователь при необходимости настраивает повторение 4. Система сохраняет сеанс 5. Система отображает сеанс в календаре
Постусловия: 1. Расписание обновлено 2. Сеанс доступен отдыхающим для заказа

Рисунок 9 – Спецификация варианта использования «Управлять расписанием услуг»

Спецификация варианта использования «Управлять объектами проживания» приведена на рисунке 10.

Прецедент: Управлять объектами проживания
ID: 5
Краткое описание: Администратор ведет справочник домиков и номеров
Главные актеры: Администратор
Второстепенные актеры: Нет
Предусловие: 1. Пользователь авторизован как администратор 2. Открыт раздел управления проживанием
Основной поток: 1. Администратор открывает карточку объекта 2. Администратор заполняет название, тип, вместимость и цену 3. Администратор добавляет описание и статус доступности 4. Администратор прикрепляет или удаляет фотографии 5. Система сохраняет изменения
Постусловия: 1. Данные объекта проживания обновлены 2. Изменения отображаются в публичном каталоге

Рисунок 10 – Спецификация варианта использования «Управлять объектами проживания»

В результате анализа функциональных требований определены роли пользователей, состав доступных действий и основные сценарии работы системы. Полученные диаграммы и спецификации используются далее при проектировании структуры приложения, пользовательского интерфейса и логики обработки заявок.

1.4 Выбор технологий проектирования системы

При проектировании автоматизированной системы были изучены средства визуального моделирования, базирующиеся на унифицированном языке моделирования UML. Этот язык является распространенным стандартом в области проектирования программных систем и позволяет наглядно представить структуру ролей, взаимодействие пользователей с системой и основные функциональные процессы.

В ходе работы рассматривались два инструмента: Draw.io (diagrams.net) и Visual Paradigm Community Edition. Draw.io является бесплатным редактором диаграмм, доступным в браузере и в настольной версии. Он удобен для быстрой подготовки схем, но имеет ограниченные средства проверки корректности моделей.

Visual Paradigm Community Edition представляет собой профессиональную CASE-среду, поддерживающую UML-диаграммы и другие виды проектных моделей. Инструмент предоставляет возможности построения диаграмм вариантов использования, классов, последовательностей и деятельности, что делает его удобным для оформления материалов выпускной квалификационной работы [8].

В итоге в качестве основного средства проектирования была выбрана среда Visual Paradigm Community Edition. Выбор обусловлен поддержкой UML, удобством построения диаграмм, возможностью экспорта изображений и достаточным набором функций для учебного проектирования автоматизированной системы.

1.5 Выбор инструментальных средств разработки

Разработка системы бронирования требует согласованного набора инструментов: языка программирования, веб-фреймворка, средств доступа к данным, системы управления базой данных, среды разработки и инструментов моделирования. При выборе стека учитывались практическая применимость технологий, поддержка веб-интерфейса, удобство работы с базой данных и возможность реализации ролей пользователей.

К инструментальным средствам предъявлялись следующие требования:

- поддержка разработки веб-приложения с публичной частью и панелями управления;
- возможность разграничения доступа для разных ролей пользователей;
- удобная работа с реляционной базой данных;
- производительность и надежность при работе с данными о бронированиях и услугах;
- наличие качественной документации и интеграции с платформой .NET.

В качестве языка программирования был выбран C# – современный объектно-ориентированный язык платформы .NET. C# обладает строгой типизацией, развитой стандартной библиотекой и активно применяется при разработке корпоративных, настольных и веб-приложений.

Для реализации серверной части и пользовательского интерфейса выбран ASP.NET Core MVC [4]. Данный фреймворк позволяет разрабатывать веб-приложения по архитектурному шаблону Model-View-Controller, разделяя модели данных, представления и контроллеры, отвечающие за обработку запросов пользователя.

В качестве системы управления базами данных может быть использован Microsoft SQL Server [6] или другая реляционная СУБД, совместимая с Entity Framework Core. Реляционная модель подходит для хранения связанных

данных о пользователях, объектах проживания, услугах, бронированиях, оплатах и статусах заявок.

Entity Framework Core является объектно-реляционным маппером для платформы .NET [5]. Он позволяет работать с данными через объектную модель приложения, описывать связи между сущностями и выполнять миграции базы данных без ручного написания большого количества SQL-запросов.

Клиентская часть приложения построена на Razor Views, HTML, CSS и JavaScript [12, 14, 15, 16]. Эти средства применяются для отображения страниц, форм бронирования, календаря услуг, модальных окон оплаты и элементов управления фотографиями. Для моделирования диаграмм используется Visual Paradigm [8].

Таким образом, выбранный стек технологий представляет собой целостное решение для разработки веб-приложения: C#, ASP.NET Core MVC, Entity Framework Core и реляционная база данных. Все компоненты хорошо интегрируются в рамках платформы .NET и подходят для реализации дипломного проекта.

1.6 Анализ требований к разрабатываемой системе

Разрабатываемая автоматизированная система должна обеспечивать обработку заявок от момента просмотра информации отдыхающим до подтверждения бронирования, оказания услуги или отмены заявки. При этом интерфейс должен быть понятным для пользователей с разным уровнем технической подготовки.

Основные функциональные требования к системе сформированы исходя из анализа предметной области и типовых сценариев работы базы отдыха.

Управление бронированиями проживания является ключевым функциональным блоком. Система должна обеспечивать выбор дат заезда и выезда, проверку доступности объекта размещения, оформление заявки с

указанием данных отдыхающего, количества гостей, выбранного домика или номера, а также последующее изменение статуса бронирования.

Управление справочниками предполагает ведение данных об объектах проживания, услугах, категориях, ценах и пользователях. Администратор должен иметь возможность редактировать сведения о домиках и номерах, а менеджер услуг – управлять перечнем услуг, расписанием и доступностью временных интервалов.

Разграничение прав доступа является обязательным требованием к системе [17, 18]. В зависимости от роли пользователь получает доступ только к тем функциям, которые необходимы ему для выполнения задач: отдыхающий оформляет заявки, администратор управляет бронированиями и пользователями, менеджер услуг работает с услугами.

Контроль доступности проживания и услуг должен выполняться автоматически на основе уже существующих заявок и расписания. Система должна предотвращать двойное бронирование одного объекта на пересекающиеся даты, а также не позволять оформить услугу на занятое время или при отсутствии свободных мест.

Формирование отчетности должно позволять получать сведения о загрузке объектов проживания, количестве заявок, популярности услуг и оплатах за выбранный период. Такие данные необходимы администратору для анализа работы базы отдыха и планирования загрузки ресурсов.

Дополнительные требования к системе:

- интерфейс должен быть понятным и не вызывать затруднений у пользователей с базовым уровнем компьютерной грамотности;
- названия элементов интерфейса, кнопок и разделов должны соответствовать их назначению;
- система должна быть доступна через веб-браузер и обеспечивать работу с публичной частью и административной панелью;

- время отклика на типовые операции, такие как открытие списка бронирований, проверка доступности и формирование отчета, не должно вызывать заметных задержек при штатном объеме данных.

Таким образом, сформированный перечень требований включает весь необходимый функционал: информирование отдыхающих, управление бронированиями, работу с услугами, разграничение прав доступа и формирование отчетности. Эти требования стали основой для последующего проектирования и разработки системы.

2 Проектирование информационной системы бронирования и управления услугами

2.1 Проектирование структур данных

Проектирование структуры данных начинается с выделения объектов предметной области и определения связей между ними. Сначала формируется концептуальное представление базы данных, которое позволяет описать состав сущностей без привязки к конкретной СУБД и способу физического хранения.

Концептуальная модель – это упрощенное описание предметной области, которое показывает, какие объекты существуют, какими свойствами они обладают и как связаны между собой. Она не зависит от конкретных технологий или способов хранения данных и служит основой для дальнейшего проектирования информационной системы [11].

Для составления концептуальной модели данных требуется выделить список всех основных сущностей предметной области разрабатываемой системы. Описание сущностей предметной области представлено в таблице 2.

Таблица 2 – Сущности БД

Название сущности	Идентификатор сущности	Назначение сущности
Пользователь	User	Содержит сведения о пользователях системы и их ролях
Объект проживания	Accommodation	Содержит сведения о домиках и номерах
Фотография проживания	AccommodationPhoto	Содержит сведения об изображениях объектов проживания
Бронирование проживания	Booking	Содержит сведения о бронировании объектов проживания
Услуга	Service	Содержит сведения о дополнительных услугах
Фотография услуги	ServicePhoto	Содержит сведения об изображениях дополнительных услуг
Расписание услуги	ServiceSchedule	Содержит сведения о датах, времени и доступности услуг
Заказ услуги	ServiceOrder	Содержит сведения о бронировании дополнительных услуг

Графическое представление концептуальной модели базы данных представлено на рисунке 11.



Рисунок 11 – Концептуальная модель базы данных системы информирования, бронирования и управления услугами для отдыхающих

Таким образом была разработана концептуальная модель базы данных системы информирования, бронирования и управления услугами для отдыхающих.

После определения состава сущностей и связей выполняется детализация модели. На этом этапе концептуальные элементы уточняются до уровня атрибутов, типов данных и связей, необходимых для последующей реализации в реляционной базе данных.

Логическая модель данных представляет собой более детальную и технически проработанную версию концептуальной модели. В ней осуществляется детальная проработка сущностей, их атрибутов и взаимосвязей, что создает основу для последующего перехода к физической реализации [11].

Логическая модель проектируемой базы данных представлена на рисунке 12.

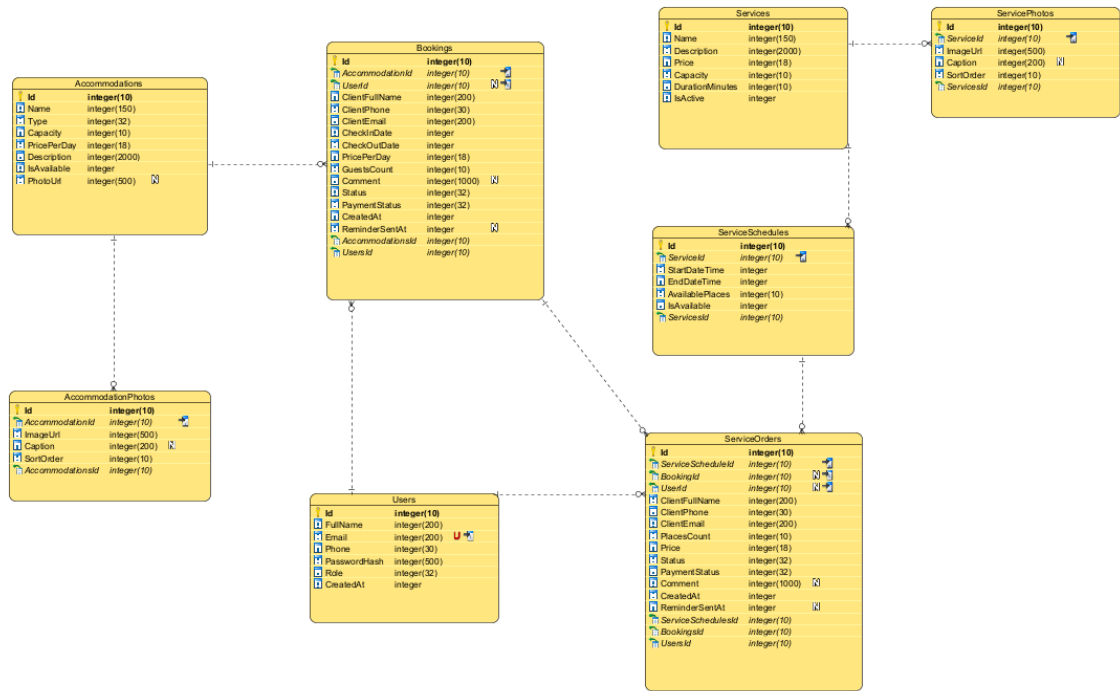


Рисунок 12 – Логическая модель базы данных системы

Таким образом, разработанные модели полностью отражают структуру предметной области и обеспечат основу для последующего перехода к физической реализации.

Наборы атрибутов для каждой сущности предметной области и их характеристики приведены в приложении Д.

2.2 Проектирование архитектуры приложения

Проектирование приложения направлено на получение поддерживаемой веб-системы, которая обрабатывает бронирования проживания, заказы дополнительных услуг, оплату и уведомления пользователей. Для такой задачи требуется разделить пользовательский интерфейс, бизнес-правила и работу с данными, чтобы изменение одного блока не нарушало остальные части системы.

Система построена по клиент-серверной архитектуре. Клиентская часть отвечает за отображение страниц, форм, календаря услуг и элементов управления. Серверная часть принимает запросы пользователя, выполняет проверки, изменяет данные и формирует ответ. Такой подход подходит для веб-приложения, доступного через браузер, и позволяет централизованно хранить данные о бронированиях, услугах и пользователях [9].

В основе приложения используется архитектурный шаблон Model-View-Controller. Модели описывают сущности предметной области: пользователей, объекты проживания, услуги, расписание, бронирования, заказы услуг и статусы оплаты. Представления формируют HTML-страницы для отдыхающего, администратора и менеджера услуг. Контроллеры принимают запросы, выбирают необходимый сценарий обработки и передают данные в представления.

Логика приложения разделена на несколько уровней. Слой представления реализован с помощью Razor Views, HTML, CSS и JavaScript. Он обеспечивает работу публичных страниц, личного кабинета, административных разделов, календаря услуг и модальных окон оплаты.

Слой бизнес-логики содержит правила обработки бронирований и заказов. В нем выполняется проверка пересечения дат проживания, контроль доступных мест на услугу, изменение статусов, отмена заявок, обработка демонстрационной оплаты и подготовка email-уведомлений. Отдельные сервисы отвечают за бронирование проживания, заказ услуг и отправку уведомлений.

Слой доступа к данным реализован через Entity Framework Core. Приложение работает с базой данных через объектную модель C#, а структура таблиц обновляется с помощью миграций [5, 19]. Такой способ снижает объем ручных SQL-запросов и упрощает сопровождение схемы данных.

База данных размещается в Microsoft SQL Server. В ней хранятся учетные записи пользователей, роли, сведения об объектах проживания, услугах, фотографиях, расписании, бронированиях, заказах услуг и оплатах.

Администрирование базы выполняется через SQL Server Management Studio [6]. Это даёт возможность централизованно хранить все данные и оперативно к ним обращаться.

Клиентская часть реализуется средствами Razor Views, HTML, CSS и JavaScript. Через нее пользователь получает доступ к публичным страницам, формам бронирования, личному кабинету, календарю услуг и административным разделам. Интерфейс строится с учетом простоты навигации и понятного расположения элементов управления. Благодаря этому все действия выполняются интуитивно и не требуют долгого привыкания.

Серверная часть разрабатывается на ASP.NET Core MVC. Она обрабатывает запросы пользователей, выполняет проверку доступности проживания и услуг, управляет статусами бронирований, оплатами и уведомлениями. Вся основная логика сосредоточена именно здесь, что позволяет держать данные и процессы под единым контролем. Для хранения данных используется Microsoft SQL Server, взаимодействие с которым организовано через Entity Framework Core. Такая связка обеспечивает надёжное и удобное управление информацией, накопленной в ходе работы системы.

Основные компоненты системы:

- модуль авторизации и разграничения прав доступа;
- модуль просмотра проживания и дополнительных услуг;
- модуль бронирования домиков, номеров и услуг;
- модуль календарного управления расписанием услуг;
- модуль управления заказами и статусами оплаты;
- модуль загрузки и удаления фотографий;
- модуль email-уведомлений и напоминаний;
- модуль отчетности для административной панели.

На рисунке 13 представлена диаграмма развертывания, отражающая архитектуру системы информирования, бронирования и управления услугами для отдыхающих.

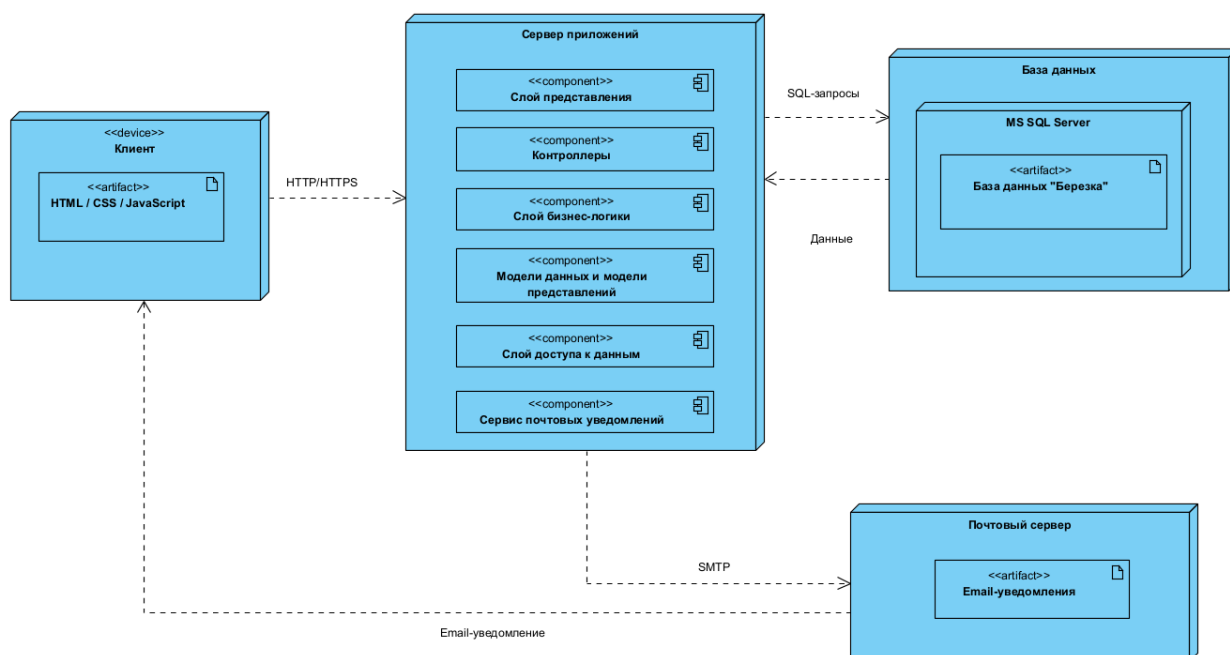


Рисунок 13 – Диаграмма развертывания системы информирования, бронирования и управления услугами для отдыхающих

Для визуализации взаимодействия компонентов была создана диаграмма компонентов. Диаграмма компонентов отображена на рисунке 14.

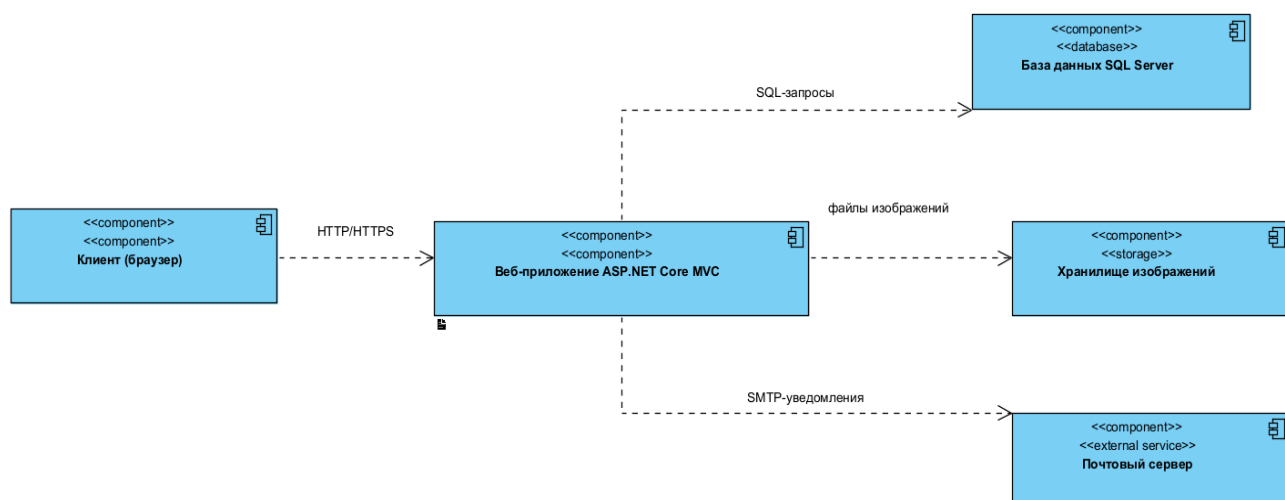


Рисунок 14 – Диаграмма компонентов серверной части системы

Таким образом было выполнено высокоуровневое проектирование системы информирования, бронирования и управления услугами для отдыхающих.

2.3 Проектирование программных средств

После определения общей архитектуры для основных модулей системы сформированы диаграммы классов. Они отражают состав контроллеров, сервисов, моделей данных и компонентов доступа к базе данных. Диаграммы классов основных модулей представлены в приложении Е.

Серверная часть разработана на платформе ASP.NET Core MVC. Приложение построено по многослойному принципу: обработка пользовательских запросов, прикладные проверки, работа с постоянными данными и формирование страниц разделены между самостоятельными компонентами. В составе системы выделены следующие уровни:

- слой контроллеров (Controllers): принимает HTTP-запросы, подготавливает данные для представлений и направляет операции в сервисный слой;
- слой представлений (Razor Views): формирует HTML-страницы публичной части, личного кабинета и административных панелей;
- слой сервисов (Services): выполняет проверку доступности проживания и услуг, создает бронирования, заказы и почтовые уведомления;
- слой доступа к данным (ApplicationDbContext): связывает объектную модель приложения с таблицами SQL Server через Entity Framework Core;
- слой моделей и представлений (Models, ViewModels): описывает сущности предметной области и данные, передаваемые на страницы системы [4].

Краткая спецификация контроллеров представлена в таблице 3.

Таблица 3 – Контроллеры

Название класса	Назначение
AccountController	Выполняет регистрацию, вход, выход и обработку отказа в доступе
HomeController	Формирует стартовую страницу и базовые информационные представления

Продолжение таблицы 3

Название класса	Назначение
AccommodationsController	Выводит каталог объектов проживания и галереи фотографий
BookingsController	Оформляет бронирование проживания и показывает подтверждение
ServicesController	Отображает услуги, расписание и создает заказы
CustomerController	Управляет личным кабинетом, отменами и оплатой
AdminController	Обрабатывает административные функции и панель менеджера услуг

Контроллеры не содержат всю прикладную логику. Основные проверки, связанные с доступностью проживания, расписанием услуг, оплатой и уведомлениями, вынесены в сервисные классы. Спецификация сервисов приведена в таблице 4.

Таблица 4 – Сервисы

Название класса	Назначение
BookingService	Проверяет доступность проживания и создает бронирования
ServiceOrderService	Рассчитывает свободные места и оформляет заказы услуг
BookingEmailNotificationService	Формирует письма об оплате и напоминаниях
BookingReminderBackgroundService	Отправляет напоминания по будущим броням
SmtpEmailSender	Передает сообщения через SMTP или тестовый каталог
OperationResult<T>	Возвращает результат операции с сообщением об ошибке
EmailOptions	Хранит параметры подключения к почтовому серверу
EmailMessage	Описывает адресата, тему и текст письма
BookingRequests	Передает входные данные в сервисы бронирования

Для основных сервисов предусмотрены интерфейсы. Они задают контракт выполнения операций и позволяют подключать реализации через встроенный контейнер зависимостей ASP.NET Core.

Описание интерфейсов сервисов представлено в таблице 5.

Таблица 5 – Интерфейсы сервисов

Название интерфейса	Назначение
IBookingService	Определяет операции проверки доступности и создания брони
IServiceOrderService	Определяет расчет мест и оформление заказов услуг
IBookingEmailNotificationService	Задаёт методы платёжных писем и напоминаний
IEmailSender	Описывает единый метод отправки email-сообщений

Отдельный слой репозитория в проекте не выделялся: доступ к постоянным данным выполняется через `ApplicationDbContext`. Контекст описывает наборы сущностей, ограничения, связи и начальные данные, а Entity Framework Core формирует SQL-запросы к базе данных.

Перечень компонентов доступа к данным приведен в таблице 6.

Таблица 6 – Компоненты доступа к данным

Название компонента	Назначение
<code>ApplicationDbContext</code>	Описывает DbSet, связи, ограничения и индексы
<code>DatabaseSeeder</code>	Создает начальных пользователей и расписание услуг
<code>Users</code>	Хранит учетные записи и роли пользователей
<code>Accommodations</code>	Хранит домики, номера и параметры проживания
<code>Bookings</code>	Хранит сведения о бронированиях проживания
<code>Services</code>	Хранит справочник дополнительных услуг
<code>ServiceSchedules</code>	Хранит дату, время и доступность сеансов
<code>ServiceOrders</code>	Хранит заказы услуг и статусы оплаты
<code>AccommodationPhotos</code>	Хранит фотографии объектов проживания
<code>ServicePhotos</code>	Хранит фотографии услуг

Предметные данные представлены набором моделей. Эти классы используются контроллерами, сервисами и контекстом базы данных как единая объектная схема приложения.

Модели предметной области представлены в таблице 7.

Таблица 7 – Модели предметной области

Название интерфейса	Назначение
AppUser	Описывает пользователя, контакты и роль в системе
Accommodation	Описывает объект проживания и его стоимость
AccommodationPhoto	Связывает фотографию с объектом проживания
Booking	Хранит параметры бронирования проживания
Service	Описывает дополнительную услугу
ServicePhoto	Связывает изображение с услугой
ServiceSchedule	Описывает сеанс услуги по дате и времени
ServiceOrder	Хранит заказ услуги и данные клиента
Enums	Фиксирует роли, типы и статусы операций

Диаграммы классов, сопровождающие описание программных средств, показывают распределение обязанностей между MVC-компонентами, сервисами, моделями и слоем доступа к данным.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс разрабатываемой системы выполнен в формате веб-приложения, открываемого через браузер. Страницы формируются средствами Razor Views в ASP.NET Core MVC, а визуальное оформление строится на HTML, CSS, Bootstrap и отдельных сценариях JavaScript [12, 13, 14, 15, 16].

Проектирование интерфейса ориентировано на три группы пользователей: отдыхающего, администратора и менеджера услуг. Для

каждой роли оставлены только те разделы, которые нужны для ее сценариев работы. Такой подход снижает количество лишних действий и делает навигацию предсказуемой.

В системе предусмотрены следующие основные страницы и элементы интерфейса:

- главная публичная страница – содержит краткое описание назначения системы и переходы к основным разделам;
- каталог проживания – отображает домики и номера с ценой, вместимостью, описанием, статусом доступности и фотографиями;
- форма бронирования проживания – позволяет выбрать даты заезда и выезда, указать количество гостей и контактные данные;
- каталог услуг – показывает доступные услуги, расписание сеансов, стоимость, количество свободных мест и изображения;
- форма заказа услуги – используется для выбора сеанса, указания числа мест и связи услуги с бронью проживания при необходимости;
- личный кабинет отдыхающего – содержит активные и отмененные бронирования, заказы услуг, статусы оплаты и действия пользователя;
- страницы регистрации и авторизации – обеспечивают создание учетной записи, вход в систему и разграничение доступа по ролям;
- панель администратора – предоставляет сводные показатели, отчеты, управление проживанием, бронями, услугами и заказами;
- панель менеджера услуг – предназначена для ведения услуг, расписания сеансов, фотографий и заявок на дополнительные услуги;
- календарь расписания – отображает сеансы услуг в дневном, недельном и месячном представлении;
- окно демонстрационной оплаты – имитирует ввод банковской карты и переводит заявку в статус оплаченной.

Взаимодействие пользовательской части с сервером построено по схеме ASP.NET Core MVC. Основные операции выполняются через HTTP-запросы GET и POST, формы Razor передают данные контроллерам, а защита от

подделки запросов обеспечивается antiforgery-токенами. JavaScript применяется точно: для масок ввода, модальных окон оплаты, галерей изображений и элементов календаря.

В интерфейсе предусмотрена обработка ошибок:

- при некорректном вводе данных пользователь получает сообщение рядом с соответствующим полем или в верхней части формы;
- после успешного действия отображается уведомление, например при отмене бронирования, оплате заявки или изменении статуса заказа.

Навигация построена с учетом роли пользователя. Неавторизованный посетитель видит публичные разделы и формы входа, отдыхающий получает доступ к личному кабинету, администратор – к панели управления, а менеджер услуг – к инструментам работы с расписанием и заявками.

3 Реализация информационной системы бронирования и управления услугами

3.1 Реализация базы данных

На основе концептуальной и логической модели, была создана физическая модель данных, представленная на рисунке 15. Физическая модель данных – это точное представление структуры базы данных, предназначенное для конкретной СУБД. Она определяет, как логическая модель будет реализована на физическом уровне [11].

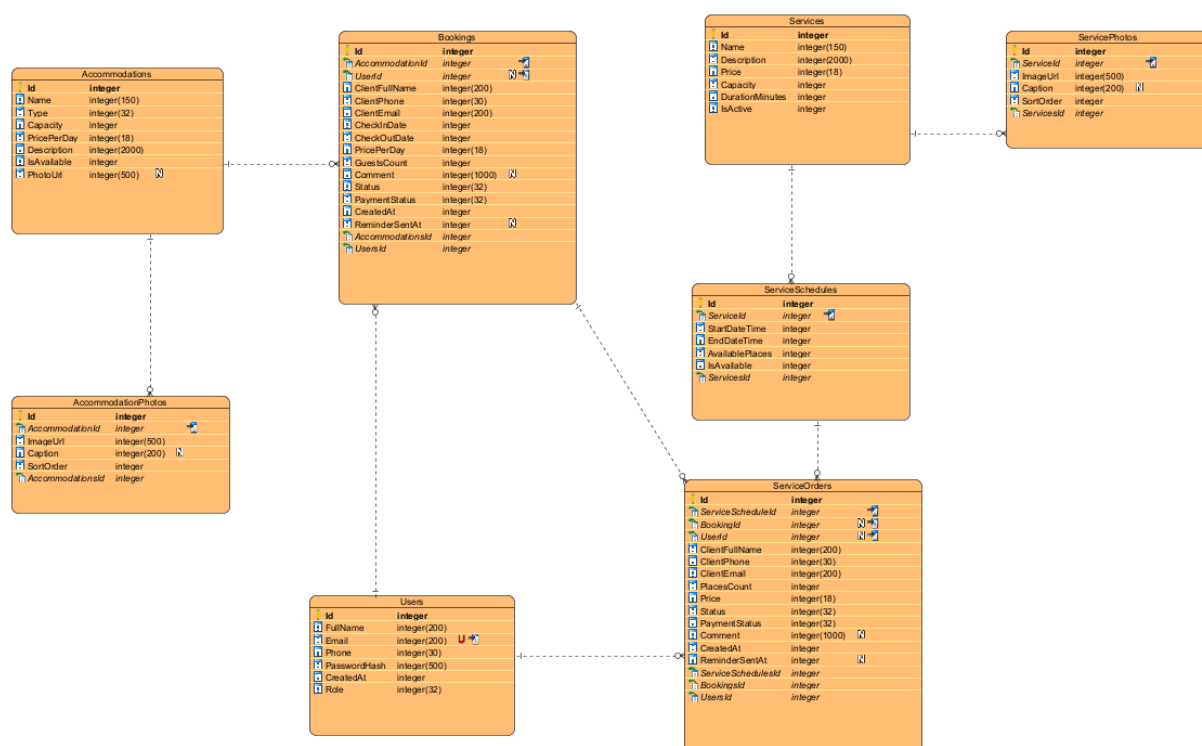


Рисунок 15 – Физическая модель базы данных системы информирования, бронирования и управления услугами для отдыхающих

Согласно физической модели с помощью СУБД MS SQL Server была развернута база данных информирования, бронирования и управления услугами для отдыхающих. В этой базе хранятся все сведения о зарегистрированных пользователях, доступных вариантах проживания, дополнительных услугах, оформленных бронях и истории заказов.

базы данных. В серверной части выделены следующие программные элементы:

- модели предметной области, описывающие пользователей, проживание, услуги, бронирования, расписание и заказы;
- модели представлений, передающие на страницы только те данные, которые требуются конкретному экрану;
- сервисы и их интерфейсы, содержащие правила проверки доступности, оформления заявок и отправки уведомлений;
- контекст базы данных `ApplicationDbContext`, обеспечивающий связь объектной модели с таблицами SQL Server;
- контроллеры, принимающие HTTP-запросы и возвращающие Razor-представления или результат операции.

Такое распределение и использование внедрения зависимостей снижает связанность кода и упрощает доработку отдельных функций без изменения всей системы [10].

Обмен между пользовательским интерфейсом и серверной частью реализован средствами ASP.NET Core MVC. Для открытия страниц используются GET-запросы, а операции изменения данных выполняются через POST-запросы с antiforgery-защитой. Отдельные действия календаря возвращают данные в формате JSON, остальные операции завершаются переходом на представление или обновлением страницы.

Реализованные действия модуля учетных записей описаны в таблице 8.

Таблица 8 – Действия контроллера учетных записей

URL запроса	HTTP метод	Описание
/Account/Login	GET	открывает форму входа пользователя
/Account/Login	POST	проверяет учетные данные и создает cookie-сеанс
/Account/Register	GET	открывает форму регистрации отдыхающего
/Account/Register	POST	создает учетную запись с ролью отдыхающего
/Account/Logout	POST	завершает сеанс авторизованного пользователя
/Account/AccessDenied	GET	отображает страницу отказа в доступе

Действия публичных страниц системы приведены в таблице 9.

Таблица 9 – Действия публичного модуля

URL запроса	HTTP метод	Описание
/	GET	открывает главную публичную страницу системы
/Home/Index	GET	открывает главную страницу по стандартному маршруту
/Home/Privacy	GET	отображает информационную страницу
/Home/Error	GET	возвращает представление ошибки приложения

Действия каталога проживания представлены в таблице 10.

Таблица 10 – Действия каталога проживания

URL запроса	HTTP метод	Описание
/Accommodations	GET	выводит список домиков и номеров
/Accommodations/Index	GET	формирует каталог проживания с фотографиями
/Bookings/Create?accommodationId={id}	GET	переходит к форме бронирования выбранного объекта

Действия оформления бронирования проживания приведены в таблице 11.

Таблица 11 – Действия модуля бронирования проживания

URL запроса	HTTP метод	Описание
/Bookings/Create?accommodationId={id}	GET	загружает данные объекта и недоступные даты
/Bookings/Create	POST	создает бронь проживания после проверки доступности
/Bookings/Confirmation/{id}	GET	показывает подтверждение созданной брони

Действия модуля дополнительных услуг отображены в таблице 12.

Таблица 12 – Действия модуля дополнительных услуг

URL запроса	HTTP метод	Описание
/Services	GET	выводит список услуг и будущих сеансов
/Services/Index	GET	формирует публичный каталог дополнительных услуг
/Services/Order?scheduleId={id}	GET	открывает форму заказа выбранного сеанса
/Services/Order	POST	создает заказ услуги с учетом свободных мест
/Services/Confirmation/{id}	GET	показывает подтверждение заказа услуги
/Customer/Index	GET	переходит к заказам пользователя после оформления услуги

Действия личного кабинета отдыхающего приведены в таблице 13.

Таблица 13 – Действия личного кабинета отдыхающего

URL запроса	HTTP метод	Описание
/Customer/Index	GET	отображает активные бронирования и заказы услуг
/Customer/Index?tab=canceled	GET	отображает архив отмененных заявок
/Customer/CancelBooking	POST	отменяет бронь проживания пользователя
/Customer/CancelServiceOrder	POST	отменяет заказ дополнительной услуги
/Customer/PayBooking	POST	изменяет статус оплаты брони проживания
/Customer/PayServiceOrder	POST	изменяет статус оплаты заказа услуги

Административные действия для просмотра сводных данных, отчетов и заявок представлены в таблице 14.

Таблица 14 – Действия административной панели

URL запроса	HTTP метод	Описание
/Admin/Index	GET	отображает сводные показатели администратора
/Admin/Reports	GET	формирует отчеты по выбранному периоду
/Admin/Bookings	GET	выводит список бронирований проживания
/Admin/UpdateBookingStatus	POST	изменяет статус бронирования и оплаты
/Admin/ServiceOrders	GET	выводит список заказов дополнительных услуг
/Admin/UpdateServiceOrderStatus	POST	изменяет статус заказа услуги и оплаты

Действия для управления каталогами проживания и услуг приведены в таблице 15.

Таблица 15 – Действия управления каталогами

URL запроса	HTTP метод	Описание
/Admin/Accommodations	GET	отображает справочник объектов проживания
/Admin/CreateAccommodation	GET	открывает форму добавления проживания
/Admin/CreateAccommodation	POST	создает объект проживания и сохраняет фотографии
/Admin/EditAccommodation/{id}	GET	открывает форму редактирования проживания
/Admin/EditAccommodation	POST	обновляет данные проживания и набор фотографий
/Admin/DeleteAccommodation	POST	удаляет объект проживания при отсутствии броней
/Admin/Services	GET	отображает справочник дополнительных услуг
/Admin/CreateService	GET	открывает форму добавления услуги

Продолжение таблицы 15

URL запроса	HTTP метод	Описание
/Admin/CreateService	POST	создает услугу и сохраняет фотографии
/Admin/EditService/{id}	GET	открывает форму редактирования услуги
/Admin/EditService	POST	обновляет услугу и прикрепленные изображения
/Admin/DeleteService	POST	удаляет услугу при отсутствии связанных сеансов

Действия для работы с календарем расписания услуг представлены в таблице 16.

Таблица 16 – Действия календаря расписания услуг

URL запроса	HTTP метод	Описание
/Admin/Schedules	GET	открывает календарь расписания услуг
/Admin/CreateSchedule	GET	открывает форму создания сеанса
/Admin/CreateSchedule	POST	создает один или повторяющийся сеанс услуги
/Admin/EditSchedule/{id}	GET	открывает форму редактирования сеанса
/Admin/EditSchedule	POST	обновляет время и доступность сеанса
/Admin/MoveSchedule	POST	переносит сеанс в календаре
/Admin/DeleteSchedule	POST	удаляет выбранный сеанс или серию сеансов

Действия панели менеджера услуг приведены в таблице 17.

Таблица 17 – Действия панели менеджера услуг

URL запроса	HTTP метод	Описание
/Admin/Services	GET	открывает панель управления услугами
/Admin/Schedules	GET	отображает календарь сеансов услуг
/Admin/ServiceOrders	GET	показывает заказы дополнительных услуг
/Admin/CreateSchedule	POST	добавляет сеанс услуги в расписание
/Admin/MoveSchedule	POST	изменяет дату и время сеанса из календаря
/Admin/UpdateServiceOrderStatus	POST	обновляет статус заказа и оплаты услуги

Серверная часть реализована как единое приложение, однако ее внутренние компоненты разделены по назначению. Контроллеры отвечают за маршруты и права доступа, сервисы выполняют прикладные проверки, а Entity Framework Core обеспечивает согласованную работу с базой данных.

3.3 Реализация клиентской части системы бронирования

Клиентская часть автоматизированной системы информирования, бронирования и управления услугами для отдыхающих отвечает за отображение интерфейса, ввод данных и передачу пользовательских действий в серверную часть. В проекте она реализована средствами ASP.NET Core MVC: страницы формируются Razor-представлениями, а данные для них подготавливаются контроллерами и моделями представления. Такой подход позволяет разделить разметку, сценарии обработки и бизнес-логику, не вынося клиентскую часть в отдельное приложение.

Для клиентской части системы разработана диаграмма компонентов, позволяющая зафиксировать состав пользовательского интерфейса и связи между его основными элементами [3, 8]. На диаграмме отражаются Razor-представления, общий макет страниц, подключаемые таблицы стилей, JavaScript-сценарии и библиотеки, используемые для валидации форм и интерактивных элементов. Диаграмма компонентов представлена на рисунке Е.5 в приложении Е.

В таблице 18 приведено описание основных компонентов клиентской части системы.

Таблица 18 – Описание компонентов клиентской части системы бронирования и управления услугами

Компонент	Назначение
Shared/_Layout.cshtml	Общий макет страниц, навигация, подключение стилей и скриптов
Shared/_ValidationScriptsPartial.cshtml	Подключение клиентской проверки форм
Home/Index.cshtml	Главная публичная страница системы
Account/Login.cshtml	Форма авторизации пользователя
Account/Register.cshtml	Форма регистрации отдыхающего
Account/AccessDenied.cshtml	Сообщение об ограничении прав доступа
Accommodations/Index.cshtml	Каталог домиков и номеров с фотографиями
Bookings/Create.cshtml	Форма бронирования проживания
Bookings/Confirmation.cshtml	Подтверждение созданной брони
Services/Index.cshtml	Каталог услуг и доступных сеансов

Продолжение таблицы 18

Компонент	Назначение
Services/Confirmation.cshtml	Подтверждение заказа дополнительной услуги
Customer/Index.cshtml	Личный кабинет отдыхающего с активными и отмененными заявками
Services/Order.cshtml	Форма заказа выбранной услуги
Admin/Index.cshtml	Сводная панель администратора
Admin/Reports.cshtml	Отчеты по загрузке, услугам и оплатам
Admin/Bookings.cshtml	Список бронирований проживания
Admin/ServiceOrders.cshtml	Список заказов дополнительных услуг
Admin/Accommodations.cshtml	Справочник объектов проживания
Admin/AccommodationForm.cshtml	Форма создания и редактирования объекта проживания
Admin/Services.cshtml	Справочник дополнительных услуг
Admin/ServiceForm.cshtml	Форма создания и редактирования услуги
Admin/Schedules.cshtml	Календарь расписания услуг
Admin/ScheduleForm.cshtml	Форма создания и редактирования сеанса
site.css	Единое оформление публичных страниц, кабинетов, таблиц и календаря
site.js	Клиентские сценарии для форм, календаря, оплаты и галерей
bootstrap.min.css	Сетка и базовые компоненты интерфейса
bootstrap.bundle.min.js	Модальные окна, навигация и интерактивные элементы Bootstrap
jquery.min.js	Вспомогательная библиотека для клиентских сценариев
jquery.validate.min.js	Проверка корректности введенных данных на стороне клиента
jquery.validate.unobtrusive.min.js	Связь атрибутов валидации ASP.NET Core с jQuery Validation
AccommodationListItemViewModel	Данные карточки проживания для каталога
BookingCreateViewModel	Данные формы бронирования проживания
BookingConfirmationViewModel	Данные страницы подтверждения брони
ServiceListItemViewModel	Данные карточки услуги и сеансов
ServiceOrderCreateViewModel	Данные формы заказа услуги
ServiceOrderConfirmationViewModel	Данные подтверждения заказа услуги
CustomerViewModels	Данные личного кабинета отдыхающего
LoginViewModel	Данные формы входа в систему
AdminViewModels	Данные сводки, отчетов, броней и заказов
AdminCatalogViewModels	Данные форм проживания, услуг и расписания
_ViewImports.cshtml	Подключение пространств имен и tag helpers для представлений
_ViewStart.cshtml	Назначение общего макета для Razor-представлений
Shared/Error.cshtml	Представление ошибки приложения

Клиентская часть системы реализована с использованием Razor Views, HTML, CSS, JavaScript и Bootstrap в среде разработки Visual Studio 2022. Для

проверки корректности введенных данных применяются jQuery Validation и unobtrusive validation, связанные с атрибутами моделей ASP.NET Core.

Взаимодействие с серверной частью выполняется через MVC-маршруты ASP.NET Core. Данные передаются формами GET и POST, а JavaScript используется точно: для масок ввода, модальных окон оплаты, календаря расписания, удаления элементов и просмотра изображений. Код клиентской части системы приведен в приложении В.

3.4 Реализация интерфейса системы бронирования и управления услугами

Интерфейс системы разработан с учетом разделения функций между неавторизованным пользователем, отдыхающим, администратором и менеджером услуг. Пользовательские страницы сгруппированы по сценариям работы: просмотр информации, оформление заявки, контроль личных бронирований и администрирование справочников.

В системе предусмотрены следующие основные страницы:

- главная публичная страница;
- страница авторизации и регистрации;
- каталог объектов проживания;
- форма бронирования проживания;
- каталог дополнительных услуг;
- форма заказа услуги;
- личный кабинет отдыхающего;
- панель администратора;
- календарь расписания услуг;
- справочники проживания и услуг;
- панель менеджера услуг;
- окно демонстрационной оплаты.

При открытии веб-приложения пользователь попадает на главную страницу. Она доступна без авторизации и содержит краткое описание

назначения системы, переходы к каталогам проживания и услуг, а также навигацию по публичным разделам (рисунок 17).

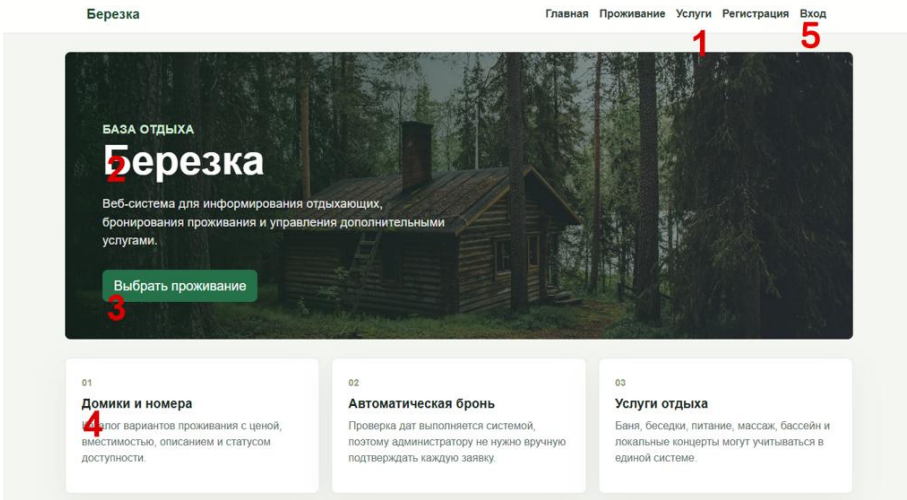


Рисунок 17 – Главная публичная страница

Основные элементы главной страницы:

- 1 – шапка сайта с основными разделами;
- 2 – информационный блок о назначении системы;
- 3 – кнопка перехода к выбору проживания;
- 4 – краткое описание возможностей системы.
- 5 – кнопки входа и регистрации.

При нажатии на кнопку «Вход» откроется страница авторизации (рисунок 18).

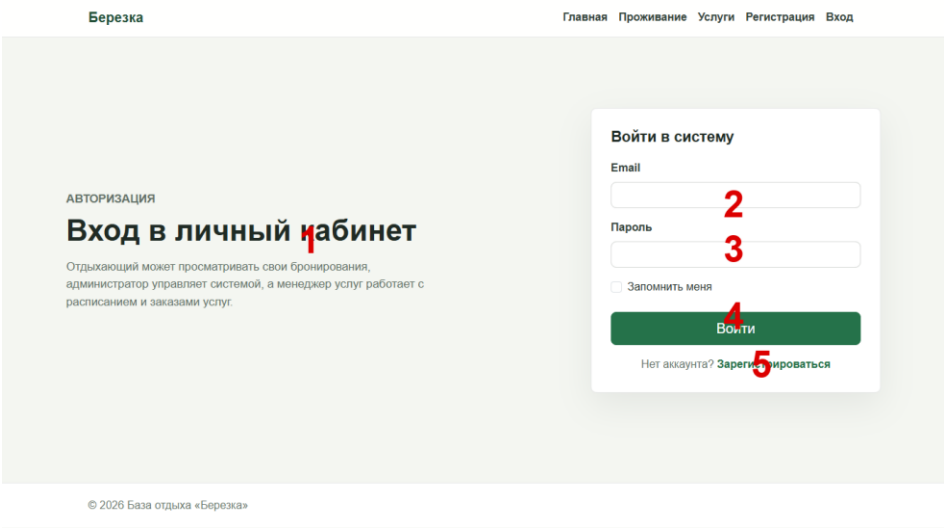


Рисунок 18 – Страница авторизации

Основные элементы страницы авторизации:

- 1 – информационный блок с пояснением назначения входа;
- 2 – поле ввода email;
- 3 – поле ввода пароля;
- 4 – кнопка входа в систему;
- 5 – ссылка на регистрацию отдыхающего.

3.4.1 Реализация интерфейса отдыхающего

Неавторизованный пользователь может просматривать каталог проживания. В карточках отображаются название объекта, тип размещения, вместимость, стоимость за сутки, описание, фотографии и доступность для бронирования (рисунок 19).

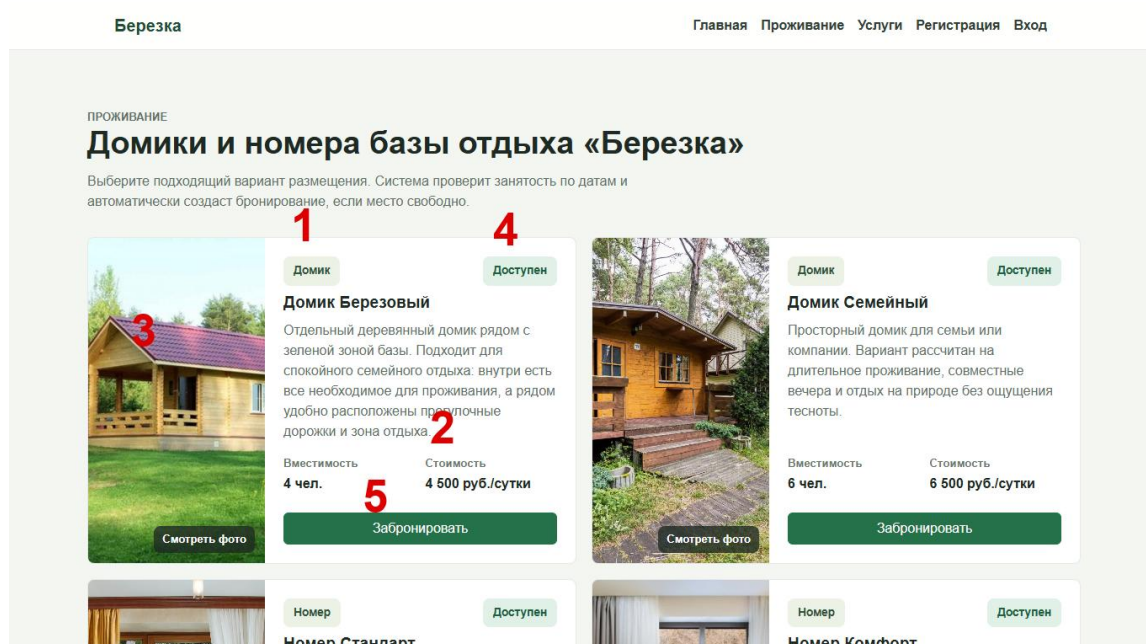


Рисунок 19 – Каталог объектов проживания

Основные элементы каталога проживания:

- 1 – карточки домиков и номеров;
- 2 – стоимость и вместимость объекта;
- 3 – фотографии объекта проживания;
- 4 – статус доступности;
- 5 – кнопка перехода к бронированию.

После выбора объекта проживания авторизованный отдыхающий переходит к форме бронирования. На странице отображаются сведения о выбранном объекте, занятые периоды, данные клиента, даты заезда и выезда, количество гостей и комментарий к заявке (рисунок 20).

The screenshot shows a web interface for booking a house named 'Домик Березовый'. The form is divided into several sections:

- Selected variant (1):** A box on the left containing the object name 'Домик Березовый', capacity '4 чел.', cost '4 500 руб./сутки', and a list of occupied dates (2):
 - 12.06.2026 - 18.06.2026
 - 20.12.2026 - 23.12.2026
- Client data (3):** A section on the right for personal information, including a text field for the full name (filled with 'Иванов Иван Иванович'), a phone number field (filled with '+7 (900) 123-45-67'), and an email field (filled with 'demo.vacationer@example.com').
- Living parameters (4):** A section for trip details, including:
 - Check-in date (filled with '24.12.2026')
 - Check-out date (filled with '27.12.2026')
 - Number of guests (dropdown menu set to '2')
- Commentary (5):** A large text area at the bottom for the user to leave a comment on the booking.

Рисунок 20 – Форма бронирования проживания

Основные элементы формы бронирования проживания:

- 1 – краткая сводка по выбранному объекту;
- 2 – список занятых дат;
- 3 – поля контактных данных отдыхающего;
- 4 – поля выбора периода проживания;
- 5 – поле комментария к заявке.

Каталог дополнительных услуг содержит перечень услуг, которые отдыхающий может заказать на конкретную дату и время. Для каждой услуги указываются описание, стоимость, формат бронирования, длительность и фотографии (рисунок 21). Все позиции каталога доступны для просмотра в публичной части системы, а авторизованный пользователь может сразу перейти к бронированию выбранной услуги на свободное время.

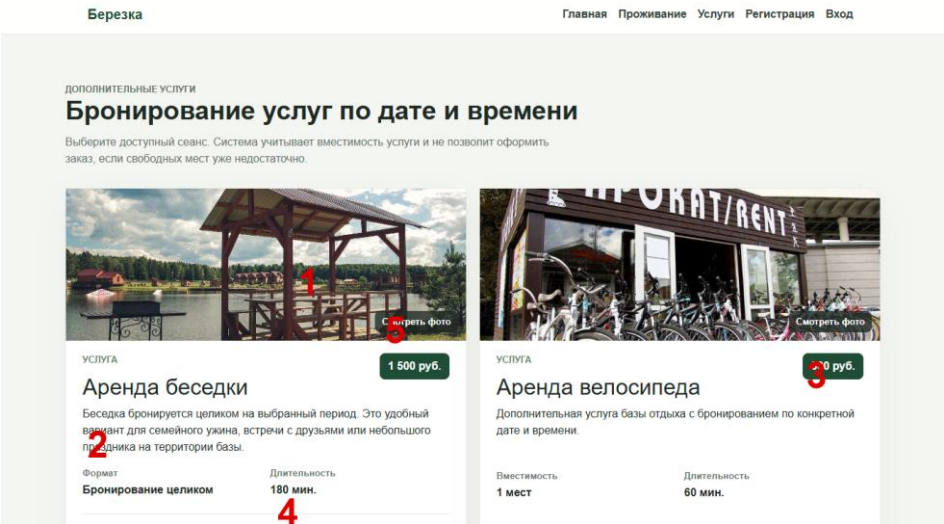


Рисунок 21 – Каталог дополнительных услуг

Основные элементы каталога услуг:

- 1 – карточки дополнительных услуг;
- 2 – фотографии и описание услуги;
- 3 – стоимость оказания услуги;
- 4 – формат бронирования и длительность услуги;
- 5 – кнопка просмотра фотографий.

Форма заказа услуги открывается после выбора свободного сеанса. В ней отображаются данные услуги, дата и время оказания, количество свободных мест, контактные данные клиента и возможность связать услугу с бронью проживания (рисунок 22).

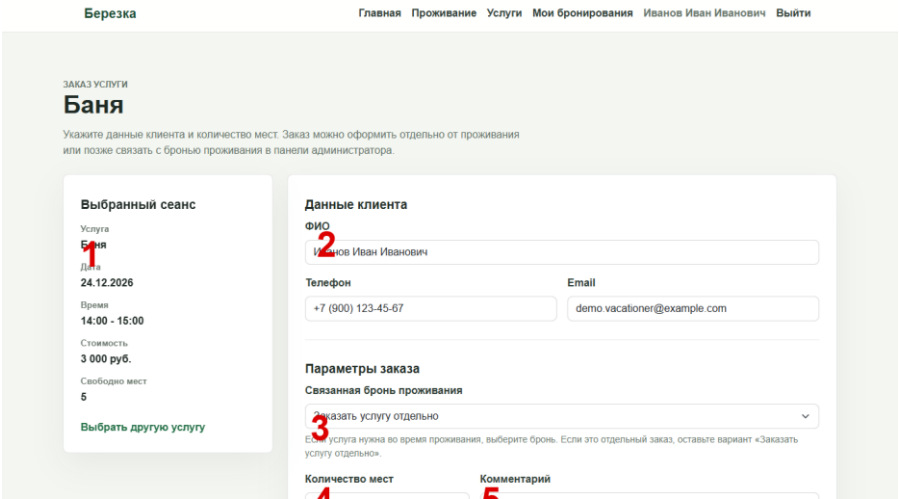


Рисунок 22 – Форма заказа дополнительной услуги

Основные элементы формы заказа услуги:

- 1 – сводка по выбранному сеансу;
- 2 – контактные данные отдыхающего;
- 3 – выбор связанной брони проживания;
- 4 – поле количества мест;
- 5 – поле комментария к заказу.

Личный кабинет отдыхающего предназначен для контроля собственных бронирований и заказов услуг. На странице показаны данные аккаунта, активные заявки, отмененные заявки, статусы оплаты и действия, доступные для пользователя (рисунок 23).

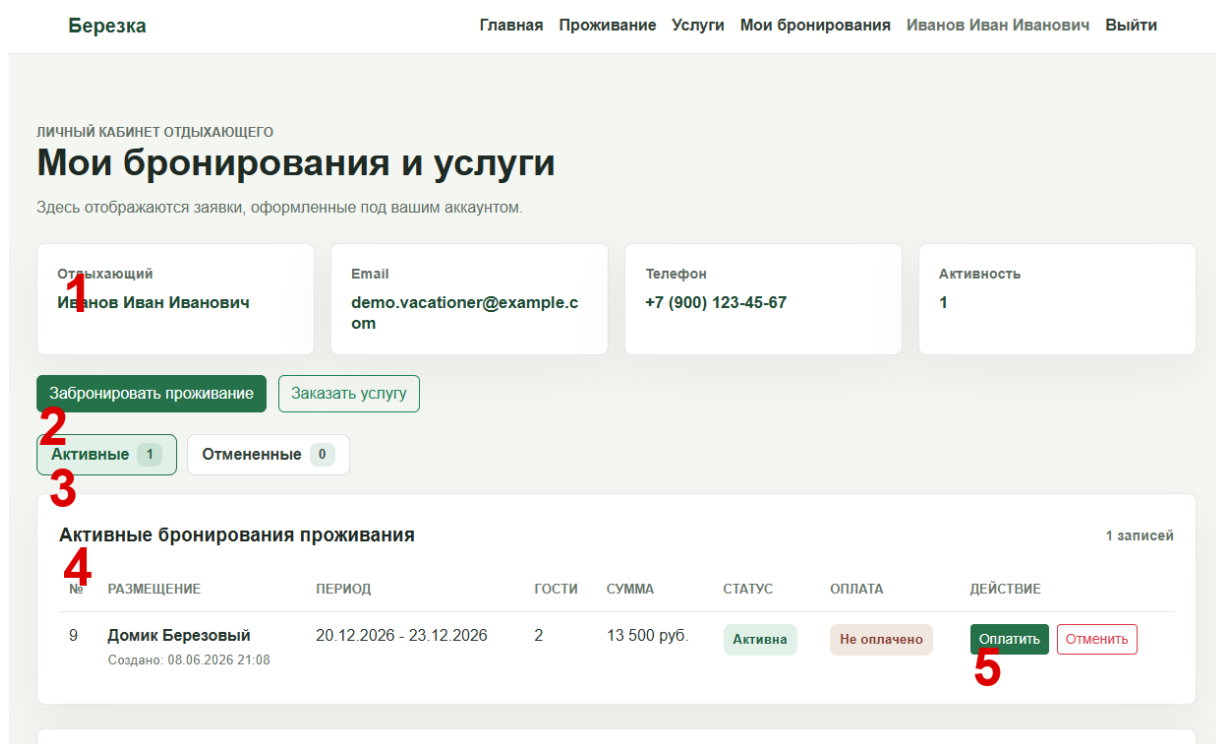


Рисунок 23 – Личный кабинет отдыхающего

Основные элементы личного кабинета отдыхающего:

- 1 – данные авторизованного пользователя;
- 2 – кнопки перехода к бронированию проживания и услуг;
- 3 – вкладки активных и отмененных заявок;
- 4 – таблица бронирований проживания;
- 5 – кнопки оплаты и отмены заявки.

Для демонстрации процесса оплаты в системе реализовано модальное окно. Оно имитирует ввод данных банковской карты и после подтверждения меняет только статус оплаты заявки; платежные данные в базе не сохраняются (рисунок 24).

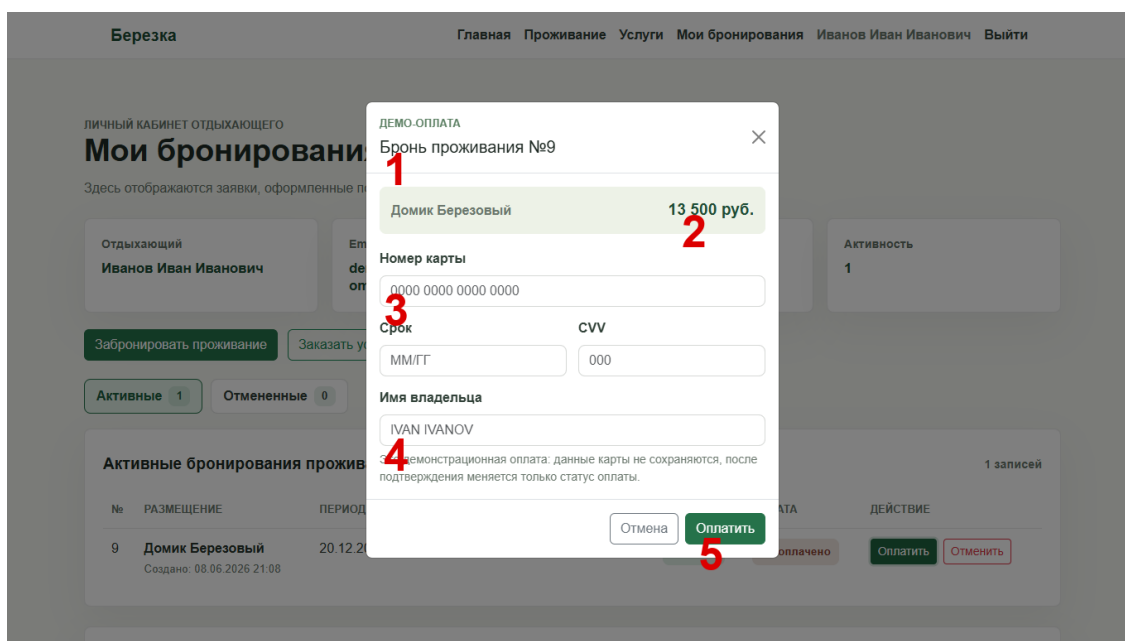


Рисунок 24 – Окно демонстрационной оплаты

Основные элементы окна демонстрационной оплаты:

- 1 – название оплачиваемой заявки;
- 2 – сумма к оплате;
- 3 – поля реквизитов карты;
- 4 – поле имени владельца карты;
- 5 – кнопка подтверждения оплаты.

3.4.2 Реализация интерфейса администратора

После входа под учетной записью администратора открывается панель управления. В ней собраны сводные показатели по объектам проживания, бронированиям, заказам услуг, оплатам и отчетам (рисунок 25). С помощью этих сводок администратор может быстро оценить текущую загрузку, проконтролировать выполнение заказов и перейти к детализированным отчетам.

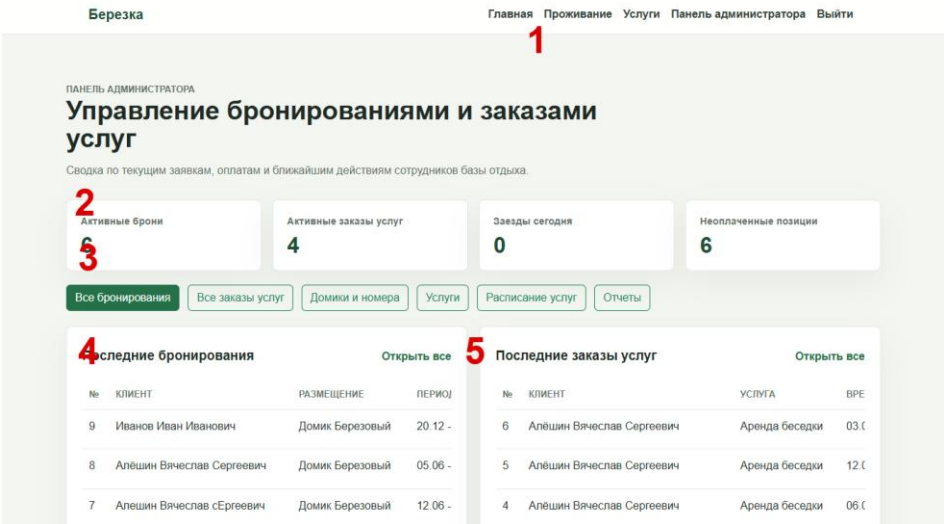


Рисунок 25 – Панель администратора

Основные элементы панели администратора:

- 1 – навигация по административным разделам;
- 2 – сводные показатели работы системы;
- 3 – быстрые переходы к справочникам;
- 4 – последние бронирования проживания;
- 5 – последние заказы дополнительных услуг.

Календарь расписания позволяет администратору управлять сеансами дополнительных услуг. В интерфейсе предусмотрены переключение дня, недели и месяца, создание сеанса, редактирование времени и контроль доступности по конкретным датам (рисунок 26).

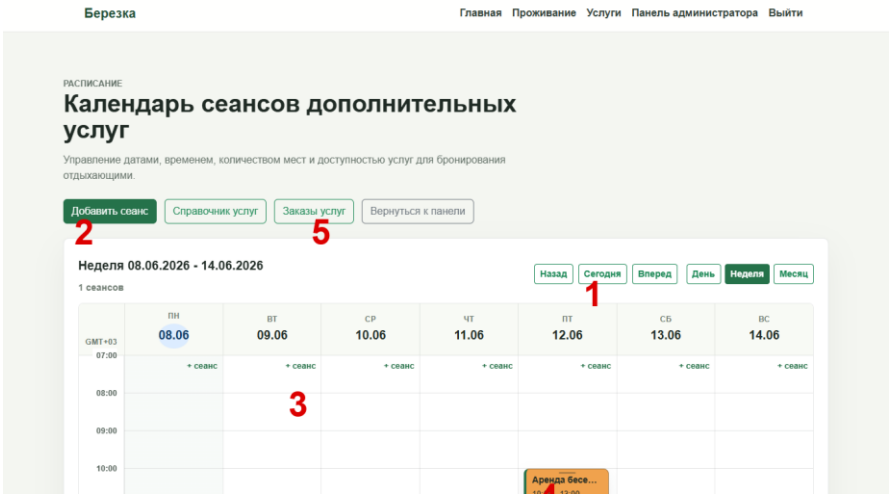


Рисунок 26 – Календарь расписания услуг администратора

Основные элементы календаря расписания:

- 1 – переключение периода отображения;
- 2 – кнопка добавления сеанса;
- 3 – календарная сетка по дням и времени;
- 4 – карточки запланированных сеансов;
- 5 – переход к заказам услуг.

Справочник объектов проживания используется для ведения домиков и номеров. Администратор может добавлять новые объекты, изменять параметры, управлять доступностью и прикреплять фотографии, которые затем отображаются в публичном каталоге (рисунок 27).

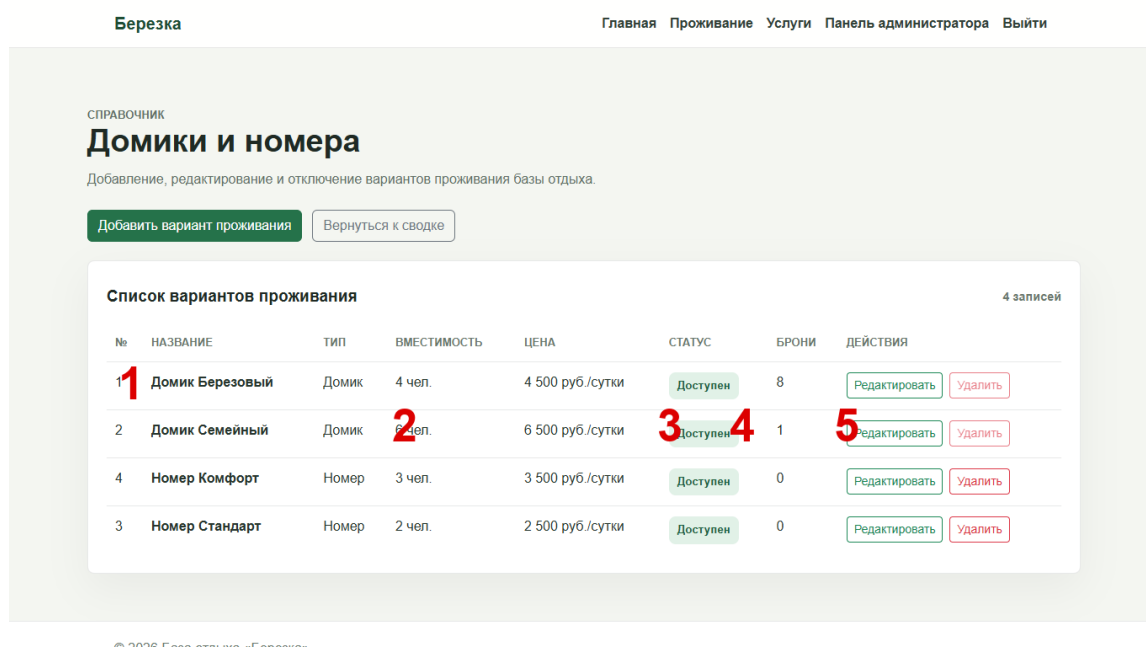


Рисунок 27 – Справочник объектов проживания

Основные элементы справочника проживания:

- 1 – таблица домиков и номеров;
- 2 – стоимость и вместимость объекта;
- 3 – статус доступности;
- 4 – количество прикрепленных фотографий;
- 5 – кнопки редактирования и удаления.

Календарь заказов услуг используется для контроля заявок отдыхающих. Администратор видит распределение заказов по датам и времени, может перейти к расписанию, справочнику услуг или открыть связанную заявку для изменения статуса и оплаты (рисунок 28).

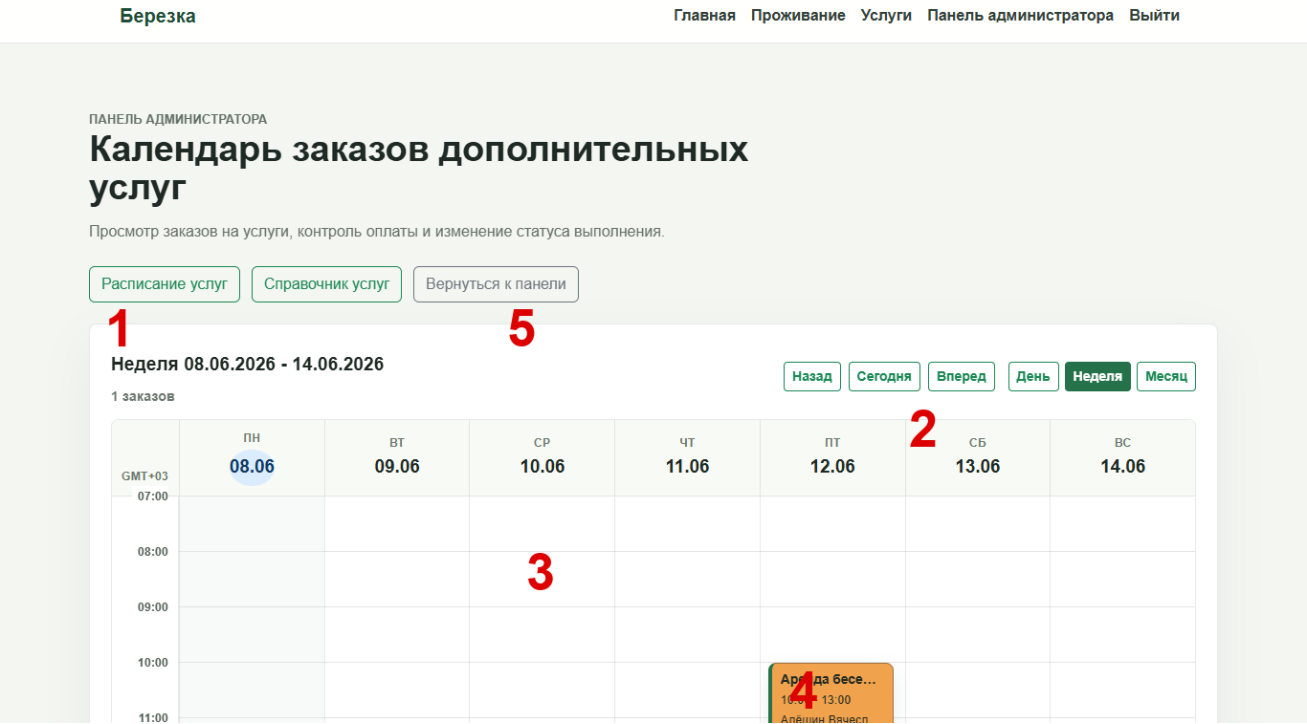


Рисунок 28 – Календарь заказов дополнительных услуг

Основные элементы календаря заказов услуг:

- 1 – панель перехода к расписанию и справочнику услуг;
- 2 – переключение периода отображения;
- 3 – календарная сетка заказов;
- 4 – карточка заказа услуги;
- 5 – кнопка возврата к панели управления.

3.4.3 Реализация интерфейса менеджера услуг

Менеджер услуг получает доступ только к функциям, связанным с дополнительными услугами. Основной рабочий экран роли представляет собой календарь расписания, где можно создавать, переносить, редактировать и удалять сеансы (рисунок 29).

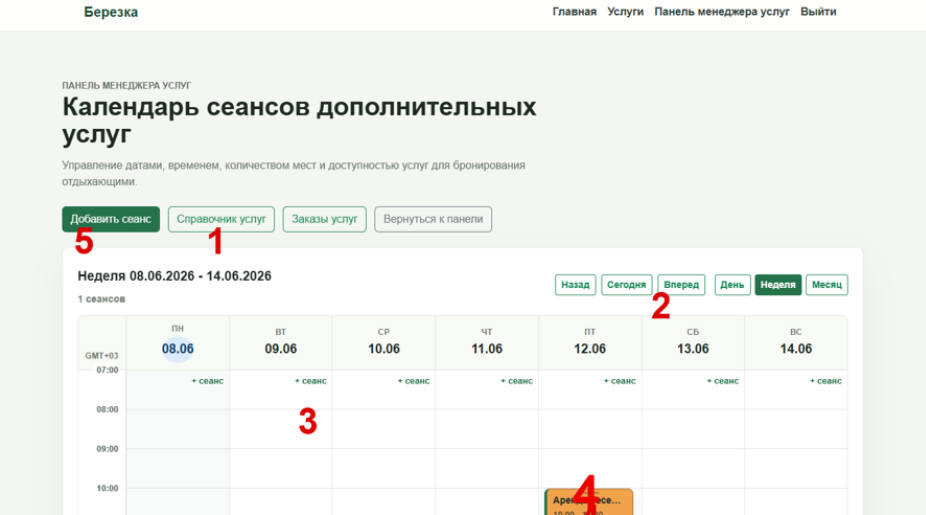


Рисунок 29 – Календарь расписания менеджера услуг

Основные элементы календаря менеджера услуг:

- 1 – панель перехода к услугам и заказам;
- 2 – переключение режима день, неделя и месяц;
- 3 – сетка времени;
- 4 – карточки сеансов;
- 5 – кнопки создания новых сеансов.

Справочник услуг в панели менеджера предназначен для добавления, редактирования и отключения дополнительных услуг. На странице отображаются стоимость, вместимость, длительность, статус, количество сеансов и число оформленных заказов (рисунок 30).

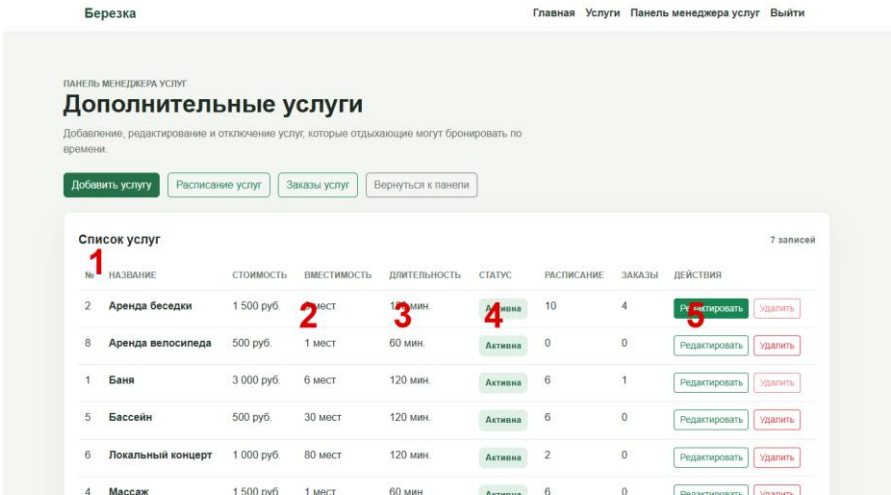


Рисунок 30 – Справочник услуг менеджера

Основные элементы справочника услуг:

1 – список дополнительных услуг;

2 – стоимость и вместимость;

3 – длительность оказания услуги;

4 – статус активности;

5 – кнопки редактирования и удаления.

Разработанный интерфейс отражает основные сценарии использования системы: информирование отдыхающих, бронирование проживания, заказ дополнительных услуг, просмотр личных заявок, демонстрационную оплату и управление расписанием со стороны сотрудников. Разделение страниц по ролям снижает количество лишних действий для пользователя и ограничивает доступ к операциям, которые не относятся к его задачам.

4 Отладка и тестирование программного обеспечения

4.1 Функциональное тестирование методом белого ящика

4.1.1 Покрытие операторов

Для проверки серверной логики автоматизированной системы информирования, бронирования и управления услугами проведено функциональное тестирование методом белого ящика. Данный подход опирается на анализ исходного кода и дает возможность проследить, как отрабатывают проверки, ветвления и операторы, связанные с созданием брони проживания.

Метод белого ящика выбран, поскольку исходный код сервисного слоя доступен для анализа, а ошибки в логике создания брони могут привести к двойному бронированию объекта проживания.

Объектом тестирования выбран метод `CreateBookingAsync` сервиса `BookingService`. Метод принимает данные бронирования, проверяет корректность дат и контактов, ищет объект проживания, контролирует пересечение с уже созданными бронями и сохраняет новую запись в базе данных. Код метода приведен в приложении Б, листинг Б.10.

Согласно критерию покрытия операторов, набор тестов должен вызывать выполнение каждого оператора метода не менее одного раза. Для метода `CreateBookingAsync` выделены следующие операторы:

- а – приведение дат заезда и выезда к типу `Date`;
- b – проверка порядка дат;
- c – возврат ошибки при неверном периоде;
- d – проверка даты заезда на прошедший период;
- e – возврат ошибки при дате заезда раньше текущей даты;
- f – проверка количества гостей;
- g – возврат ошибки при нулевом или отрицательном числе гостей;
- h – проверка обязательных контактных данных;

- i – возврат ошибки при пустых контактах;
- j – открытие транзакции с уровнем Serializable;
- k – получение объекта проживания из базы данных;
- l – проверка существования объекта;
- m – возврат ошибки при отсутствии объекта;
- n – проверка статуса доступности;
- o – возврат ошибки при недоступности объекта;
- p – сравнение числа гостей с вместимостью;
- q – возврат ошибки при превышении вместимости;
- r – проверка пересечения с другими бронями;
- s – возврат ошибки при занятости периода;
- t – создание экземпляра Booking;
- u – добавление брони в контекст Entity Framework Core;
- v – сохранение изменений;
- w – фиксация транзакции;
- x – возврат успешного результата.

Блок-схема метода CreateBookingAsync представлена в приложении Е, рисунок Е.6.

Для покрытия всех операторов сформирован набор тестов, в котором совмещены успешные и ошибочные сценарии. Если в строке таблицы не указаны иные значения, используется базовый набор входных данных.

Базовый набор входных данных:

- AccommodationId = 1;
- Accommodation.Name = "Домик Березовый";
- Accommodation.IsAvailable = true;
- Accommodation.Capacity = 4;
- Accommodation.PricePerDay = 4500;
- UserId = 1;
- ClientFullName = "Алешин Вячеслав Сергеевич";
- ClientPhone = "+7 (999) 111-22-33";

- ClientEmail = "user@example.com";
- CheckInDate = 2026-08-10;
- CheckOutDate = 2026-08-13;
- GuestsCount = 2;
- Comment = null;

При проверке сценария с занятым периодом в базу данных добавляется активная бронь этого же объекта на период с 2026-08-11 по 2026-08-14. При проверке отмененной брони используется тот же период, но со статусом Cancelled.

Результаты тестирования приведены в таблице 19.

Таблица 19 – Тест-кейсы для покрытия операторов метода
CreateBookingAsync

№	Операторы	Входные данные	Ожидаемый результат	Фактический результат
1	a, b, c	CheckInDate = 2026-08-10; CheckOutDate = 2026-08-10	Failure: дата выезда должна быть позже даты заезда	Получена ошибка о неверном периоде, результат соответствует ожидаемому
2	a, b, d, e	CheckInDate = 2026-01-10; CheckOutDate = 2026-01-13	Failure: дата заезда не может быть раньше текущей даты	Получена ошибка о прошедшей дате, результат соответствует ожидаемому
3	a, b, d, f, g	GuestsCount = 0	Failure: количество гостей должно быть больше нуля	Получена ошибка о количестве гостей, результат соответствует ожидаемому
4	a, b, d, f, h, i	ClientFullName = ""; ClientPhone и ClientEmail заполнены	Failure: необходимо указать ФИО, телефон и email клиента	Получена ошибка о контактных данных, результат соответствует ожидаемому
5	a, b, d, f, h, j, k, l, m	AccommodationId = 999	Failure: выбранный объект проживания не найден	Получена ошибка об отсутствии объекта, результат соответствует ожидаемому
6	n, o	Accommodation.IsAvailable = false	Failure: выбранный объект проживания временно недоступен	Получена ошибка о недоступности объекта, результат соответствует ожидаемому
7	p, q	GuestsCount = 5; Accommodation.Capacity = 4	Failure: количество гостей превышает вместимость объекта проживания	Получена ошибка о превышении вместимости, результат соответствует ожидаемому
8	r, s	Есть активная бронь: 2026-08-11 - 2026-08-14	Failure: на выбранные даты объект проживания уже забронирован	Получена ошибка о занятости периода, результат соответствует ожидаемому

Продолжение таблицы 19

№	Операторы	Входные данные	Ожидаемый результат	Фактический результат
9	r, t, u, v, w, x	Базовый набор	Success: создана активная бронь со статусом оплаты NotPaid	Бронь создана, статус Active, оплата NotPaid, результат соответствует ожидаемому
10	t, u, v, w, x	Базовый набор; Comment = "Поздний заезд"	Success: бронь создана, комментарий сохранен	Бронь создана, комментарий сохранен, результат соответствует ожидаемому
11	r, t, u, v, w, x	Существующая бронь: 2026-08-07 - 2026-08-10; новая бронь начинается 2026-08-10	Success: соседние периоды не считаются пересечением	Бронь создана, результат соответствует ожидаемому
12	r, t, u, v, w, x	Есть отмененная бронь: 2026-08-11 - 2026-08-14	Success: отмененная бронь не блокирует период	Бронь создана, результат соответствует ожидаемому
13	a-x	Базовый набор; ClientFullName, ClientPhone, ClientEmail содержат пробелы по краям	Success: данные клиента сохраняются без лишних пробелов	Бронь создана, строковые данные очищены, результат соответствует ожидаемому

Для покрытия операторов метода `CreateBookingAsync` подготовлены сценарии, охватывающие все основные варианты выполнения. Тесты проверяют неверные даты, прошедший период, нулевое число гостей, незаполненные контакты, отсутствующий или недоступный объект, превышение вместимости и пересечение с ранее созданной бронью.

Отдельно проверены корректные сценарии: создание брони, сохранение комментария, разрешение смежных периодов, игнорирование отмененных броней и удаление лишних пробелов в строковых полях. По итогам тестов установлено, что метод выполняет валидацию данных до сохранения, а создание брони происходит только при соблюдении всех бизнес-правил

4.1.2 Покрытие решений

В соответствии с критерием покрытия решений необходимо составить такие тесты, при которых каждое ветвление метода принимает истинное и ложное значение не менее одного раза. Для метода `CreateBookingAsync` данный критерий позволяет проверить обработку ошибочных ситуаций до сохранения брони и подтвердить корректность успешного сценария создания записи.

Для метода `CreateBookingAsync` были выделены следующие решения:

- b – дата выезда не позже даты заезда;
- d – дата заезда находится раньше текущей даты;
- f – количество гостей меньше либо равно 0;
- h1 – ФИО клиента не заполнено;
- h2 – телефон клиента не заполнен;
- h3 – email клиента не заполнен;
- l – объект проживания не найден;
- n – объект проживания недоступен для бронирования;
- p – количество гостей превышает вместимость объекта проживания;
- r – период пересекается с активной или неотмененной бронью;
- r1 – отмененная бронь не блокирует выбранный период;
- r2 – смежные даты не образуют пересечение;
- t1 – комментарий к брони отсутствует;
- t2 – комментарий к брони заполнен.

Результаты тестирования представлены в таблице 20.

Таблица 20 – Тест-кейсы для покрытия решений метода CreateBookingAsync

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	b – да	CheckInDate = 2026-08-10; CheckOutDate = 2026-08-10	Failure: дата выезда должна быть позже даты заезда	Получена ошибка о неверном периоде, результат соответствует ожидаемому
2	b – нет, d – да	CheckInDate = 2026-01-10; CheckOutDate = 2026-01-13	Failure: дата заезда не может быть раньше текущей даты	Получена ошибка о прошедшей дате, результат соответствует ожидаемому
3	b – нет, d – нет, f – да	Базовый набор; GuestsCount = 0	Failure: количество гостей должно быть больше нуля	Получена ошибка о количестве гостей, результат соответствует ожидаемому
4	f – нет, h1 – да	Базовый набор; ClientFullName = ""	Failure: необходимо указать ФИО, телефон и email клиента	Получена ошибка о контактных данных, результат соответствует ожидаемому
5	h1 – нет, h2 – да	Базовый набор; ClientPhone = ""	Failure: необходимо указать ФИО, телефон и email клиента	Получена ошибка о контактных данных, результат соответствует ожидаемому
6	h2 – нет, h3 – да	Базовый набор; ClientEmail = ""	Failure: необходимо указать ФИО, телефон и email клиента	Получена ошибка о контактных данных, результат соответствует ожидаемому

Продолжение таблицы 20

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
7	h3 – нет, l – да	Базовый набор; AccommodationId = 999	Failure: выбранный объект проживания не найден	Получена ошибка об отсутствии объекта, результат соответствует ожидаемому
8	l – нет, n – да	Базовый набор; Accommodation.Is Available = false	Failure: выбранный объект проживания временно недоступен	Получена ошибка о недоступности объекта, результат соответствует ожидаемому
9	n – нет, p – да	Базовый набор; GuestsCount = 5; Accommodation.Capacity = 4	Failure: количество гостей превышает вместимость объекта проживания	Получена ошибка о превышении вместимости, результат соответствует ожидаемому
10	p – нет, r – да	Базовый набор; существует активная бронь на период 2026-08-11 – 2026-08-14	Failure: на выбранные даты объект проживания уже забронирован	Получена ошибка о занятости периода, результат соответствует ожидаемому
11	r – нет, r1 – да	Базовый набор; существует отмененная бронь на период 2026-08-11 – 2026-08-14	Success: отмененная бронь не блокирует создание новой брони	Бронь создана, результат соответствует ожидаемому
12	r – нет, r2 – да	Базовый набор; существующая бронь завершается 2026-08-10	Success: смежные периоды не считаются пересечением	Бронь создана, результат соответствует ожидаемому
13	t1 – да	Базовый набор; Comment = ""	Success: бронь создана, комментарий сохранен как null	Бронь создана без комментария, результат соответствует ожидаемому
14	t2 – да	Базовый набор; Comment = "Поздний заезд"	Success: бронь создана, комментарий сохранен	Бронь создана с комментарием, результат соответствует ожидаемому
15	b – нет, d – нет, f – нет, h – нет, l – нет, n – нет, p – нет, r – нет	Базовый набор; свободный объект проживания; пересечения отсутствуют	Success: создана активная бронь со статусом оплаты NotPaid	Создана бронь со статусами Active и NotPaid, результат соответствует ожидаемому

Для достижения покрытия решений метода CreateBookingAsync были составлены сценарии, в которых каждое условие получает как истинное, так и ложное значение. Ошибочные тесты проверяют неверный период проживания, прошедшую дату заезда, некорректное количество гостей, отсутствие контактных данных, недоступный объект проживания, превышение вместимости и занятый период.

Успешные сценарии проверяют создание брони при свободном объекте проживания, сохранение комментария, обработку пустого комментария, разрешение смежных периодов и игнорирование отмененных броней при проверке занятости. В результате подтверждено, что метод возвращает ошибку до записи данных при нарушении бизнес-правил и создает бронь только при корректных входных данных.

4.1.3 Покрытие условий

В соответствии с критерием покрытия условий необходимо сформировать такой набор тестов, при котором каждое простое условие внутри проверяемого метода получает истинное и ложное значение не менее одного раза.

Для метода `CreateBookingAsync` данный критерий особенно важен, поскольку создание брони выполняется только после последовательной проверки дат, контактных данных, вместимости объекта проживания и занятости выбранного периода.

Для метода `CreateBookingAsync` были выделены следующие условия:

- `b` – дата выезда меньше либо равна дате заезда;
- `d` – дата заезда находится раньше текущей даты;
- `f` – количество гостей меньше либо равно 0;
- `h1` – ФИО клиента не заполнено;
- `h2` – телефон клиента не заполнен;
- `h3` – email клиента не заполнен;
- `l` – объект проживания не найден;
- `n` – объект проживания недоступен для бронирования;
- `p` – количество гостей превышает вместимость объекта проживания;
- `r1` – найдено бронирование по выбранному объекту проживания;
- `r2` – найденное бронирование не отменено;
- `r3` – дата заезда раньше даты выезда существующей брони;

- r4 – дата выезда позже даты заезда существующей брони;
- t1 – комментарий к брони не заполнен;
- t2 – комментарий к брони заполнен.

Составные проверки контактных данных и пересечения периодов были разделены на простые условия. Это позволяет проверить не только общий отказ в создании брони, но и отдельные причины, из-за которых метод возвращает ошибочный результат.

Результаты тестирования представлены в таблице 21.

Таблица 21 – Тест-кейсы для покрытия условий метода CreateBookingAsync

№	Условия	Входные данные	Ожидаемый результат	Фактический результат
1	b – да	CheckInDate = 2026-08-10; CheckOutDate = 2026-08-10	Failure: дата выезда должна быть позже даты заезда	Получена ошибка о неверном периоде, результат соответствует ожидаемому
2	b – нет, d – да	CheckInDate = 2026-01-10; CheckOutDate = 2026-01-13	Failure: дата заезда не может быть раньше текущей даты	Получена ошибка о прошедшей дате, результат соответствует ожидаемому
3	b – нет, d – нет, f – да	Базовый набор; GuestsCount = 0	Failure: количество гостей должно быть больше нуля	Получена ошибка о количестве гостей, результат соответствует ожидаемому
4	f – нет, h1 – да	Базовый набор; ClientFullName = «»	Failure: необходимо указать ФИО, телефон и email клиента	Получена ошибка о контактных данных, результат соответствует ожидаемому
5	h1 – нет, h2 – да	Базовый набор; ClientPhone = «»	Failure: необходимо указать ФИО, телефон и email клиента	Получена ошибка о контактных данных, результат соответствует ожидаемому
6	h2 – нет, h3 – да	Базовый набор; ClientEmail = «»	Failure: необходимо указать ФИО, телефон и email клиента	Получена ошибка о контактных данных, результат соответствует ожидаемому
7	h3 – нет, l – да	Базовый набор; AccommodationId = 999	Failure: выбранный объект проживания не найден	Получена ошибка об отсутствии объекта, результат соответствует ожидаемому
8	l – нет, n – да	Базовый набор; Accommodation.Is Available = false	Failure: выбранный объект проживания временно недоступен	Получена ошибка о недоступности объекта, результат соответствует ожидаемому
9	n – нет, p – да	Базовый набор; GuestsCount = 5; Accommodation.Capacity = 4	Failure: количество гостей превышает вместимость объекта проживания	Получена ошибка о превышении вместимости, результат соответствует ожидаемому
10	p – нет, r1 – да, r2 – да, r3 – да, r4 – да	Базовый набор; активная бронь на период 2026-08-11 – 2026-08-14	Failure: на выбранные даты объект проживания уже забронирован	Получена ошибка о занятости периода, результат соответствует ожидаемому

Продолжение таблицы 21

№	Условия	Входные данные	Ожидаемый результат	Фактический результат
11	r1 – да, r2 – нет	Базовый набор; найденная бронь имеет статус Cancelled	Success: отмененная бронь не блокирует выбранный период	Бронь создана, результат соответствует ожидаемому
12	r1 – да, r2 – да, r3 – нет	Базовый набор; существующая бронь завершается 2026-08-10	Success: смежный период после существующей брони разрешен	Бронь создана, результат соответствует ожидаемому
13	r1 – да, r2 – да, r3 – да, r4 – нет	Базовый набор; существующая бронь начинается 2026-08-13	Success: смежный период до существующей брони разрешен	Бронь создана, результат соответствует ожидаемому
14	r1 – нет	Базовый набор; брони по выбранному объекту отсутствуют	Success: бронь создана при свободном объекте проживания	Бронь создана, результат соответствует ожидаемому
15	t1 – да	Базовый набор; Comment = «»	Success: бронь создана, комментарий сохранен как null	Бронь создана без комментария, результат соответствует ожидаемому
16	t2 – да	Базовый набор; Comment = "Поздний заезд"	Success: бронь создана, комментарий сохранен	Бронь создана с комментарием, результат соответствует ожидаемому

Для достижения покрытия условий метода CreateBookingAsync были составлены тесты, в которых каждое простое условие принимает истинное и ложное значение. Отдельно проверены составные условия: наличие контактных данных отдыхающего и логика пересечения периода с существующими бронями.

Все простые условия метода CreateBookingAsync были проверены в обоих значениях. Полученные результаты подтверждают, что метод не допускает создание некорректной брони и сохраняет запись только при выполнении всех бизнес-правил

4.2 Модульные тесты серверной части программного обеспечения

Для проверки корректности работы метода CreateBookingAsync сервиса BookingService было выбрано Unit-тестирование. Модульное тестирование позволяет проверять отдельный фрагмент серверной логики без запуска

пользовательского интерфейса и ручного ввода данных через браузер. Это ускоряет обратную связь и повышает надёжность каждого изменения. Каждый модульный тест был построен по шаблону AAA:

- Arrange – подготовка входных данных и тестовой базы;
- Act – вызов тестируемого метода `CreateBookingAsync`;
- Assert – сравнение полученного результата с ожидаемым.

Для реализации модульных тестов использовался фреймворк `xUnit` [20]. В качестве хранилища тестовых данных применялся провайдер `Entity Framework Core InMemory` [21], что позволило изолировать каждый тестовый сценарий от рабочей базы данных.

Разработанные тесты охватывают основные сценарии использования метода `CreateBookingAsync`. Полный код модульных тестов приведен в приложении Г. Классификация тестов представлена в таблице 22.

Таблица 22 – Классификация Unit-тестов метода `CreateBookingAsync`

Категория сценариев	Количество тестов	Примеры
Проверка дат проживания	2	Дата выезда совпадает с датой заезда; дата заезда находится в прошлом
Проверка данных отдыхающего	2	Количество гостей равно 0; не заполнены контактные данные клиента
Проверка объекта проживания	3	Объект не найден; объект недоступен; превышена вместимость
Проверка пересечения броней	3	Активная бронь пересекается; смежные даты разрешены; отмененная бронь не блокирует период
Проверка статусов брони	1	После создания бронь получает статус <code>Active</code> и статус оплаты <code>NotPaid</code>
Проверка фиксации цены	1	В бронь записывается цена объекта проживания на момент создания
Проверка строковых данных	1	ФИО, телефон, email и комментарий сохраняются без лишних пробелов

Общее количество тестов: 13.

Результаты выполнения модульных тестов представлены на рисунке 31

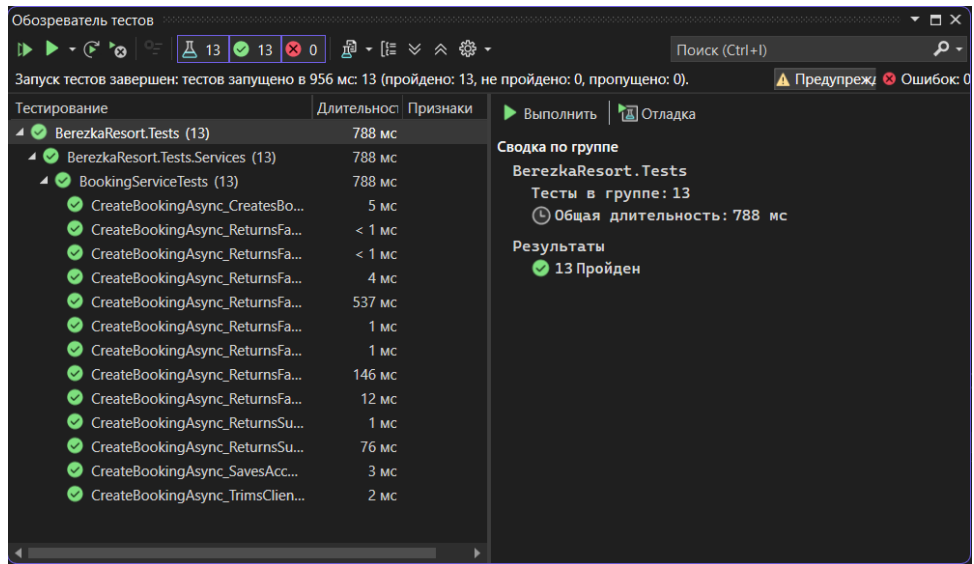


Рисунок 31 – Результаты выполнения модульных тестов

Все разработанные Unit-тесты были успешно пройдены. Это подтверждает корректность работы метода `CreateBookingAsync` при основных вариантах использования. Тесты охватывают проверку дат проживания, обязательных данных клиента, вместимости объекта, занятости периода, фиксации цены, статусов брони и обработки строковых полей.

5 Оценка качества программного кода

5.1 Тестирование SLOC

Для оценки качества программного кода серверной части автоматизированной системы информирования, бронирования и управления услугами применены метрики программного обеспечения. Количественный анализ позволяет определить размер метода, долю пустых строк, наличие комментариев и фактический объем исполняемых инструкций.

Объектом анализа выбран метод `CreateBookingAsync` сервиса `BookingService`. Метод оформляет бронь проживания: нормализует даты, проверяет входные данные, получает объект проживания из базы данных, контролирует доступность и пересечение с уже созданными бронированиями, после чего сохраняет заявку в транзакции с уровнем изоляции `Serializable`.

Метрика SLOC используется для оценки объема программного кода. Она показывает количество физических строк, пустых строк, строк комментариев и значимых строк, которые непосредственно участвуют в выполнении программной логики.

При подсчете строк кода учитывались следующие показатели

- LOC (Lines of Code) – общее количество физических строк метода;
- BLANK (Blank Lines) – количество пустых строк;
- CLOC (Comment Lines of Code) – количество строк, содержащих комментарии;
- SLOC (Source Lines of Code) – количество значимых строк кода без учета пустых строк и комментариев;
- LLOC (Logical Lines of Code) – количество логических строк кода, определяемое по операторам завершения инструкций и управляющим конструкциям `if`.

Код метода приведен в приложении Б. листинг Б.10.

Результаты анализа SLOC для метода `CreateBookingAsync`:

- LOC – 79;

- BLANK – 13;
- CLOC – 0;
- SLOC – $79 - 13 - 0 = 66$;
- LLOC – 25.

Метод содержит 79 физических строк, из которых 66 относятся к значимому коду. Значение LLOC равно 25, что указывает на умеренную логическую насыщенность: в методе сосредоточены проверки дат, контактных данных, вместимости и занятости объекта проживания, однако каждая ветвь завершает обработку явным результатом операции.

Пустые строки занимают около 16% объема метода и отделяют проверки, обращение к базе данных, формирование объекта бронирования и сохранение результата. Комментарии в методе отсутствуют, при этом назначение операций читается по именам переменных, модели запроса и возвращаемому типу `OperationResult<Booking>`.

5.2 Метрика Холстеда

Метрики Холстеда применяются для количественной оценки сложности программного кода через состав операторов и операндов. К операторам отнесены ключевые слова C#, условные конструкции, вызовы методов, присваивания, скобки и служебные разделители. Операндами выступают переменные, свойства моделей, константы и строковые значения, с которыми работает метод.

Сначала выполнен подсчет операторов. Результат представлен в таблице 23.

Таблица 23 – Подсчет операторов метода `CreateBookingAsync`

Оператор	Количество повторений
public	1
async	1
Task	1
return	9
if	8
await	6
using	1

Продолжение таблицы 23

Оператор	Количество повторений
var	5
new	1
is	1
. (вызов метода и обращение к свойству)	53
=	19
>=	1
<	1
<=	1
	2
!	1
>	1
?:	1
=>	1
()	32
{}	10
;	17
,	18

Уникальных операторов (n1) – 24.

Всего операторов (N1) – 192.

Наиболее часто встречаются обращения к свойствам и методам через точку, круглые скобки, оператор присваивания и оператор return. Такая структура связана с тем, что метод последовательно проверяет входные данные, обращается к Entity Framework Core и формирует объект бронирования перед сохранением.

К операндам отнесены переменные метода, свойства запроса, поля сущностей, значения перечислений, константы и сообщения об ошибках. Результаты подсчетов операндов представлены в таблице 24.

Таблица 24 – Подсчет операндов метода CreateBookingAsync

Операнд	Количество повторений
request	16
accommodation	7
checkInDate	5
checkOutDate	4
GuestsCount	4
dbContext	4
ClientFullName	3
ClientPhone	3
ClientEmail	3
Id	3

Продолжение таблицы 24

Операнд	Количество повторений
booking	3
Comment	3
CheckInDate	2
CheckOutDate	2
transaction	2
x	2
AccommodationId	2
UserId	2
PricePerDay	2
PaymentStatus	2
null	2
Операнды, встречающиеся один раз	9
Числовая константа 0	1
Строковые сообщения об ошибках	8

Уникальных операндов (n_2) – 39.

Всего операндов (N_2) – 94.

К операндам, встречающимся один раз, относятся Database, IsolationLevel, Accommodations, IsAvailable, Capacity, Status, BookingStatus, CreatedAt и Bookings.

Чаще всего используется операнд request, так как через объект запроса метод получает идентификатор проживания, даты, число гостей и контактные данные клиента. Также заметную долю занимают accommodation, checkInDate, checkOutDate и dbContext, что соответствует логике проверки доступности и сохранения бронирования.

После подсчета операторов и операндов выполняется расчет основных метрик Холстеда. На основе полученных данных сформированы следующие значения:

- словарь программы (n): $n = n_1 + n_2 = 63$;
- длина программы (N): $N = N_1 + N_2 = 286$;
- объем программы (V): $V = N * \log_2(n) = 1709,50$;
- сложность программы (D): $D = (n_1/2) * (N_2/n_2) = 28,92$;
- усилие на понимание (E): $E = D * V = 49444,10$.

Словарь программы отражает ограниченный, но разнородный набор конструкций. Метод использует проверки, асинхронные обращения к базе

данных, объектную инициализацию и возврат результата через `OperationResult<Booking>`.

Длина программы обусловлена не расчетными циклами, а числом защитных проверок. Код отсекает некорректные даты, пустые контактные данные, превышение вместимости, недоступность объекта проживания и пересечение с уже созданными бронями.

Объем программы показывает, что метод содержит насыщенную бизнес-логику оформления брони. При этом последовательное расположение условий снижает риск ошибочного прочтения алгоритма.

Сложность программы находится на умеренном уровне. На показатель влияют условные операторы, асинхронные вызовы `Entity Framework Core` и формирование объекта `Booking` с набором обязательных полей.

Усилие на понимание подтверждает, что метод требует внимательного анализа при сопровождении. Основная причина связана с совмещением валидации, проверки занятости и сохранения данных в одной операции.

Расчет метрик Холстеда показал, что метод `CreateBookingAsync` обладает умеренной сложностью. Полученные значения согласуются с назначением метода: он выполняет проверку заявки на проживание, предотвращает двойное бронирование и фиксирует результат в базе данных.

5.3 Метрика Чепина

Метрика Чепина применяется для оценки информационной сложности программного кода. В отличие от метрик, основанных на подсчете операторов и операндов, данный подход анализирует переменные метода и распределяет их по группам с учетом роли в алгоритме.

В метрике Чепина используются следующие группы переменных:

- Р – вводимые переменные, через которые метод получает исходные данные;
- М – модифицируемые переменные, значения которых создаются или изменяются при выполнении метода;

- С – управляющие переменные, влияющие на выбор ветвей выполнения программы;
- Т – транзитные переменные, которые используются как вспомогательные и не определяют основную бизнес-логику.

Для анализа выбран метод CreateBookingAsync сервиса BookingService. В данном методе используются переменные, связанные с заявкой на бронирование проживания, объектом размещения, проверкой занятости, транзакцией базы данных и формированием записи Booking.

Распределение переменных по группам Чепина представлено в таблице 25.

Таблица 25 – Распределение переменных метода CreateBookingAsync по группам Чепина

Группа	Вес	Переменные	Количество
Р	$a1 = 1$	request, cancellationToken, accommodation, request.AccommodationId, request.UserId, request.ClientFullName, request.ClientPhone, request.ClientEmail, request.Comment, request.GuestsCount, accommodation.Id, accommodation.PricePerDay	12
М	$a2 = 2$	transaction, booking, dbContext.Bookings, BookingStatus.Active, PaymentStatus.NotPaid, CreatedAt	6
С	$a3 = 3$	checkInDate, checkOutDate, request.GuestsCount, request.ClientFullName, request.ClientPhone, request.ClientEmail, accommodation, accommodation.IsAvailable, accommodation.Capacity, HasBlockingOverlappingBookingAsync	10
Т	$a4 = 0.5$	x	1

Расчет метрики Чепина выполняется по формуле:

$$Q=a1 \times P+a2 \times M+a3 \times C+a4 \times T.$$

Подставляем значения:

$$Q = (1 \times 12) + (2 \times 6) + 3 \times 10) + (0.5 \times 1) = 12 + 12 + 30 + 0.5 = 54.5.$$

Информационная сложность метода CreateBookingAsync составляет

$$Q = 54.5.$$

Наибольший вклад в сложность метода вносят управляющие переменные группы С. Это объясняется набором проверок, от которых зависит дальнейшее выполнение операции: корректность дат, заполненность

контактных данных, существование объекта проживания, его доступность, вместимость и наличие пересекающихся бронирований.

Модифицируемые переменные группы М отражают изменение состояния системы. К ним относятся создаваемая бронь, транзакция, коллекция бронирований контекста данных, а также значения статусов, фиксируемые при сохранении новой записи.

Вводимые переменные группы Р задают исходные данные для оформления брони. Метод получает их из объекта запроса, параметра отмены операции и найденной сущности проживания, поэтому данная группа описывает как пользовательский ввод, так и данные, считанные из базы.

К транзитным переменным отнесена переменная х, используемая в лямбда-выражении при поиске объекта проживания. Она выполняет вспомогательную роль и не изменяет смысл бизнес-операции.

5.4 Автоматическая проверка

Автоматическая проверка программного кода проводилась по набору показателей, используемых инструментом Lizard [22]. Данный подход позволяет оценить количество строк кода, цикломатическую сложность, число токенов, количество параметров метода и физическую длину функции.

В рамках выпускной квалификационной работы проверка выполнена для файла BookingService.cs, так как в нем расположен метод CreateBookingAsync, отвечающий за оформление бронирования проживания и контроль доступности объекта размещения.

Результат проверки файла BookingService.cs представлен на рисунке 32.

```
> lizard BerezkaResort.Web\Services\BookingService.cs
=====
NLOC   CCN   token  PARAM  length  location
-----
66      9    419     2     79   CreateBookingAsync@42-120@BookingService.cs
-----
1 file analyzed.
=====
Average NLOC: 66.00
Average cyclomatic complexity: 9.00
Total nloc: 66
```

Рисунок 32 – Результат автоматической проверки BookingService.cs

Расшифровка результатов автоматической проверки:

- NLOC – количество строк кода без учета пустых строк и комментариев (для метода `CreateBookingAsync` данный показатель составил 66 строк);
- CCN – цикломатическая сложность метода (для метода `CreateBookingAsync` значение CCN составило 9);
- token – количество лексем, т. е. отдельных элементов программного кода (для метода `CreateBookingAsync` количество токенов составило 419);
- PARAM – количество параметров метода (метод `CreateBookingAsync` принимает 2 параметра: `CreateBookingRequest request` и `CancellationToken cancellationToken`);
- length – физическая длина метода в строках (для метода `CreateBookingAsync` данный показатель составил 79 строк).

В ходе автоматической проверки установлено, что метод `CreateBookingAsync` обладает умеренной цикломатической сложностью и не содержит критических признаков чрезмерной запутанности. Полученные результаты согласуются с ручным анализом метрик SLOC, Холстеда и Чепина.

6 Планирование разработки и оценка бюджета

Планирование разработки позволяет определить состав работ, сроки выполнения, необходимые ресурсы и ориентировочную стоимость создания программного обеспечения. В рамках выпускной квалификационной работы было выполнено планирование разработки автоматизированной системы информирования, бронирования и управления услугами для отдыхающих.

Для планирования разработки использовалась программа ProjectLibre. Данное программное средство позволяет формировать список задач проекта, задавать длительность работ, устанавливать зависимости между задачами, назначать ресурсы и рассчитывать стоимость выполнения проекта. Также с помощью ProjectLibre была построена диаграмма Ганта, отражающая календарный план разработки системы.

Разработка системы была разделена на следующие основные этапы:

- анализ предметной области и требований;
- планирование разработки и оценка бюджета;
- проектирование системы;
- разработка программного обеспечения;
- тестирование программного обеспечения;
- контроль качества программного обеспечения;
- оформление пояснительной записки.

На этапе анализа предметной области и требований выполнялось исследование процессов информирования отдыхающих, бронирования проживания, заказа дополнительных услуг и разграничения ролей пользователей.

Этап планирования разработки и оценки бюджета включал составление календарного плана, определение исполнителей, назначение ресурсов, расчет трудозатрат и подготовку диаграммы Ганта.

На этапе проектирования системы выполнялась разработка архитектуры приложения, концептуальной, логической и физической моделей базы

данных, UML-диаграмм, структуры серверной части и пользовательского интерфейса.

На этапе разработки программного обеспечения выполнялась настройка проекта ASP.NET Core MVC, реализация моделей данных, миграций, авторизации, бронирования проживания, заказов услуг, календаря, демонстрационной оплаты, email-уведомлений и загрузки фотографий.

Этап тестирования программного обеспечения включал функциональное тестирование метода бронирования, разработку модульных тестов серверной части и исправление выявленных дефектов.

Этап контроля качества программного обеспечения включал расчет метрик SLOC, Холстеда, Чепина и автоматическую оценку кода с помощью инструмента Lizard.

Перечень задач разработки представлен на рисунке 33.

	№	Название	Продолжительность	Начало	Окончание
1		Начало проекта	0 дней	02.03.26 8:00	02.03.26 8:00
2		☒ Анализ предметной области и требований	5 дней	02.03.26 8:00	06.03.26 17:00
3		Изучение предметной области	2 дней	02.03.26 8:00	03.03.26 17:00
4		Анализ систем аналогичного назначения	1 день	04.03.26 8:00	04.03.26 17:00
5		Формирование функциональных требований	2 дней	05.03.26 8:00	06.03.26 17:00
6		☒ Планирование разработки и оценка бюджета	3 дней	09.03.26 8:00	11.03.26 17:00
7		Составление календарного плана	1 день	09.03.26 8:00	09.03.26 17:00
8		Определение ресурсов и ставок	1 день	10.03.26 8:00	10.03.26 17:00
9		Построение диаграммы Ганта	1 день	11.03.26 8:00	11.03.26 17:00
10		☒ Проектирование системы	9 дней	12.03.26 8:00	24.03.26 17:00
11		Проектирование архитектуры приложения	2 дней	12.03.26 8:00	13.03.26 17:00
12		Разработка UML-диаграмм	2 дней	16.03.26 8:00	17.03.26 17:00
13		Проектирование базы данных	3 дней	18.03.26 8:00	20.03.26 17:00
14		Проектирование пользовательского интерфейса	2 дней	23.03.26 8:00	24.03.26 17:00
15		☒ Разработка программного обеспечения	17 дней	25.03.26 8:00	16.04.26 17:00
16		Настройка проекта ASP.NET Core MVC	1 день	25.03.26 8:00	25.03.26 17:00
17		Реализация моделей и миграций БД	3 дней	26.03.26 8:00	30.03.26 17:00
18		Реализация авторизации и ролей	2 дней	31.03.26 8:00	01.04.26 17:00
19		Реализация бронирования проживания	3 дней	02.04.26 8:00	06.04.26 17:00
20		Реализация услуг и календаря	4 дней	07.04.26 8:00	10.04.26 17:00
21		Реализация оплаты и email-уведомлений	2 дней	13.04.26 8:00	14.04.26 17:00
22		Реализация фотографий и доработка интерфейса	2 дней	15.04.26 8:00	16.04.26 17:00
23		☒ Тестирование программного обеспечения	7 дней	17.04.26 8:00	27.04.26 17:00
24		Функциональное тестирование	3 дней	17.04.26 8:00	21.04.26 17:00
25		Разработка модульных тестов	2 дней	22.04.26 8:00	23.04.26 17:00
26		Исправление выявленных дефектов	2 дней	24.04.26 8:00	27.04.26 17:00
27		☒ Контроль качества программного обеспечения	3 дней	28.04.26 8:00	30.04.26 17:00
28		Расчет метрик SLOC, Холстеда и Чепина	1 день	28.04.26 8:00	28.04.26 17:00
29		Автоматическая оценка кода Lizard	1 день	29.04.26 8:00	29.04.26 17:00
30		Анализ результатов контроля качества	1 день	30.04.26 8:00	30.04.26 17:00
31		☒ Оформление пояснительной записки	6 дней	01.05.26 8:00	08.05.26 17:00
32		Подготовка текстовых разделов	2 дней	01.05.26 8:00	04.05.26 17:00
33		Оформление рисунков и таблиц	2 дней	05.05.26 8:00	06.05.26 17:00
34		Финальная вычитка и нормоконтроль	2 дней	07.05.26 8:00	08.05.26 17:00
35		Конец проекта	0 дней	08.05.26 17:00	08.05.26 17:00

Рисунок 33 – Перечень задач разработки системы

На рисунке 34 представлена диаграмма Ганта, которая показывает последовательность выполнения задач и продолжительность каждого этапа.

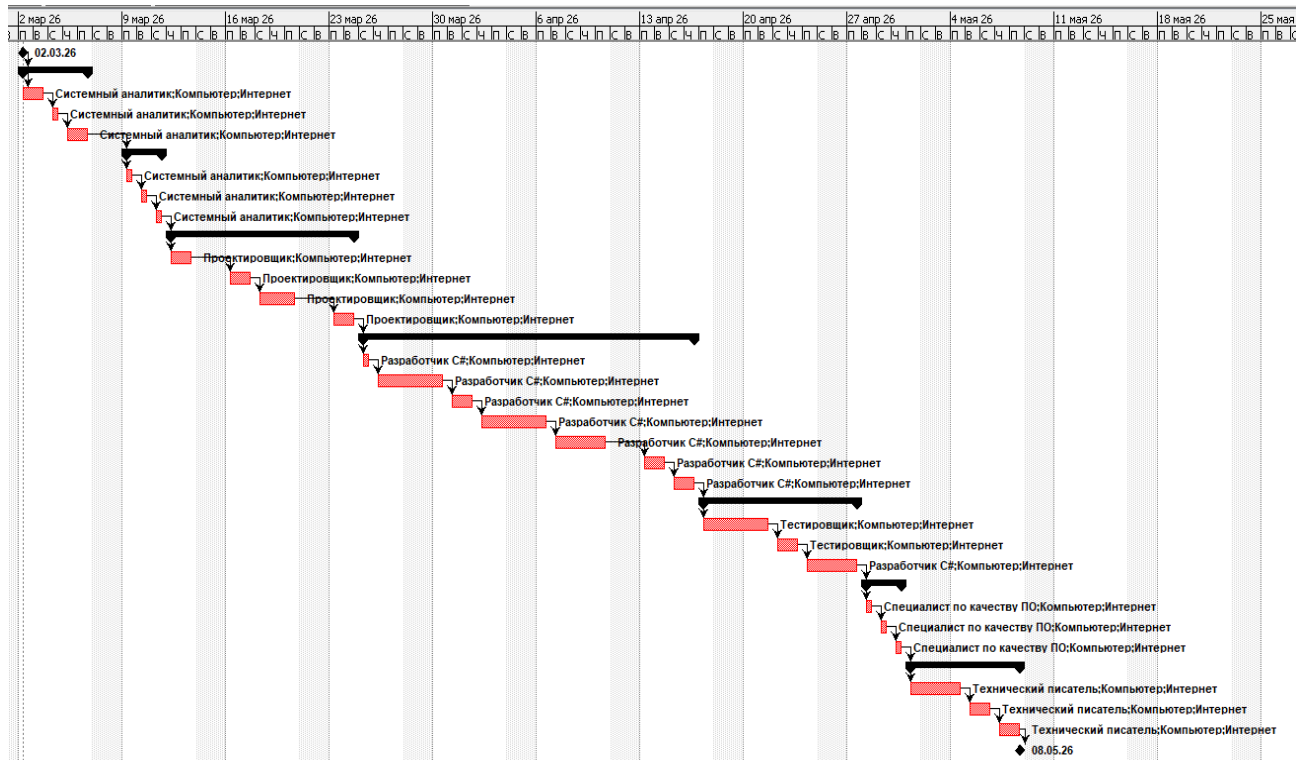


Рисунок 34 – Диаграмма Ганта

Общая продолжительность работы над проектом составила 50 рабочих дней. Каждый этап состоит из набора связанных задач, выполнение которых необходимо для перехода к следующему этапу.

В проекте были выделены следующие ресурсы:

- системный аналитик;
- проектировщик;
- разработчик С#;
- тестировщик;
- специалист по качеству ПО;
- технический писатель;
- компьютер;
- интернет.

На рисунке 35 приведена стоимость рабочего времени задействованных ресурсов.

	Ⓜ	Название	Тип	Стандартная ставка
1	🌸	Системный аналитик	Работа	450/час
2	🌸	Проектировщик	Работа	500/час
3	🌸	Разработчик C#	Работа	650/час
4	🌸	Тестировщик	Работа	450/час
5	🌸	Специалист по качеству ПК	Работа	500/час
6	🌸	Технический писатель	Работа	400/час
7	🌸	Компьютер	Материал	0
8	🌸	Интернет	Материал	0

Рисунок 35 – Стоимость ресурсов

Общие затраты проекта с учетом продолжительности задач представлены на рисунке 36.

	Ⓜ	Название	Стоимость
1		Начало проекта	0 руб.
2		☐ Анализ предметной области и требований	18000 руб.
3		Изучение предметной области	7200 руб.
4		Анализ систем аналогичного назначения	3600 руб.
5		Формирование функциональных требований	7200 руб.
6		☐ Планирование разработки и оценка бюджета	10800 руб.
7		Составление календарного плана	3600 руб.
8		Определение ресурсов и ставок	3600 руб.
9		Построение диаграммы Ганта	3600 руб.
10		☐ Проектирование системы	36000 руб.
11		Проектирование архитектуры приложения	8000 руб.
12		Разработка UML-диаграмм	8000 руб.
13		Проектирование базы данных	12000 руб.
14		Проектирование пользовательского интерфейса	8000 руб.
15		☐ Разработка программного обеспечения	88400 руб.
16		Настройка проекта ASP.NET Core MVC	5200 руб.
17		Реализация моделей и миграций БД	15600 руб.
18		Реализация авторизации и ролей	10400 руб.
19		Реализация бронирования проживания	15600 руб.
20		Реализация услуг и календаря	20800 руб.
21		Реализация оплаты и email-уведомлений	10400 руб.
22		Реализация фотографий и доработка интерфейса	10400 руб.
23		☐ Тестирование программного обеспечения	28400 руб.
24		Функциональное тестирование	10800 руб.
25		Разработка модульных тестов	7200 руб.
26		Исправление выявленных дефектов	10400 руб.
27		☐ Контроль качества программного обеспечения	12000 руб.
28		Расчет метрик SLOC, Холстеда и Чепина	4000 руб.
29		Автоматическая оценка кода Lizard	4000 руб.
30		Анализ результатов контроля качества	4000 руб.
31		☐ Оформление пояснительной записки	19200 руб.
32		Подготовка текстовых разделов	6400 руб.
33		Оформление рисунков и таблиц	6400 руб.
34		Финальная вычитка и нормоконтроль	6400 руб.
35		Конец проекта	0 руб.
36		Итого:	212800 руб.

Рисунок 36 – Общие затраты проекта

Итоговая стоимость проекта составила 212 800 рублей. Распределение затрат по этапам выглядит следующим образом:

- анализ предметной области и требований – 18 000 руб;
- планирование разработки и оценка бюджета – 10 800 руб;
- проектирование системы – 36 000 руб;
- разработка программного обеспечения – 88 400 руб;
- тестирование программного обеспечения – 28 400 руб;
- контроль качества программного обеспечения – 12 000 руб;
- оформление пояснительной записки – 19 200 руб;

Проведенное планирование позволяет заранее определить последовательность разработки, оценить трудоемкость проекта и зафиксировать ориентировочную стоимость создания автоматизированной системы.

Заключение

В рамках выпускной квалификационной работы спроектирована и реализована автоматизированная система информирования, бронирования и управления услугами для отдыхающих.

На этапе анализа предметной области определены основные сценарии работы системы: просмотр информации о проживании и услугах, оформление бронирования, заказ дополнительных услуг, учет оплаты, администрирование заявок и управление расписанием.

Архитектура приложения построена по клиент-серверной модели с разделением ответственности между представлениями, контроллерами, сервисами бизнес-логики и слоем доступа к данным. Такой подход упрощает сопровождение кода и позволяет дорабатывать отдельные функциональные блоки без переработки всей системы.

Для базы данных разработаны концептуальная, логическая и физическая модели. В них отражены пользователи, роли, объекты проживания, фотографии, бронирования, услуги, расписание сеансов, заказы услуг и статусы оплаты.

В ходе проектирования подготовлены UML-диаграммы, описывающие назначение системы, варианты использования, состав компонентов и развертывание приложения. Диаграммы позволили зафиксировать связи между пользователями, веб-приложением и базой данных.

Разработка выполнена на языке C# с применением ASP.NET Core MVC, Entity Framework Core и Microsoft SQL Server. Пользовательский интерфейс реализован на Razor Views с использованием HTML, CSS, JavaScript и Bootstrap.

В системе реализованы регистрация и авторизация пользователей, разграничение прав доступа для отдыхающего, администратора и менеджера услуг, оформление брони проживания, заказ услуг по расписанию,

демонстрационная оплата, работа с фотографиями и отправка email-уведомлений.

Проверка качества включала анализ программного кода по метрикам SLOC, Холстеда, Чепина, автоматическую оценку сложности и модульное тестирование сервиса бронирования. Полученные результаты показали умеренную сложность ключевого метода и корректную работу основных проверок доступности.

Поставленная цель достигнута: разработанное веб-приложение автоматизирует основные процессы информирования, бронирования проживания и управления дополнительными услугами. Система может использоваться как основа для дальнейшего расширения, включая интеграцию с реальным платежным сервисом, личными уведомлениями и дополнительными отчетами.

Список использованных источников

1. Bnovo – автоматизированная система управления гостиницами и отелями // Bnovo [Электронный ресурс]. URL: <https://bnovo.ru/> (дата обращения: 15.06.2026)
2. 1С:Отель – система управления для гостиничного бизнеса // 1С:Отель [Электронный ресурс]. URL: <https://1c-hotel.ru/> (дата обращения: 15.06.2026)
3. Unified Modeling Language Specification Version 2.5.1 // Object Management Group [Электронный ресурс]. URL: <https://www.omg.org/spec/UML/2.5.1/About-UML> (дата обращения: 15.06.2026)
4. Overview of ASP.NET Core MVC // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-10.0> (дата обращения: 15.06.2026)
5. Overview of Entity Framework Core // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/> (дата обращения: 15.06.2026)
6. SQL Server technical documentation // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17> (дата обращения: 15.06.2026)
7. Visual Studio documentation // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/visualstudio/windows/?view=visualstudio> (дата обращения: 15.06.2026)
8. What is Use Case Diagram? // Visual Paradigm [Электронный ресурс]. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/> (дата обращения: 15.06.2026)
9. ASP.NET Core fundamentals overview // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en->

us/aspnet/core/fundamentals/?view=aspnetcore-10.0 (дата обращения: 15.06.2026)

10. Dependency injection in ASP.NET Core // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-10.0> (дата обращения: 15.06.2026)

11. What Is Data Modeling? // IBM [Электронный ресурс]. URL: <https://www.ibm.com/think/topics/data-modeling> (дата обращения: 15.06.2026)

12. Views in ASP.NET Core MVC // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-10.0> (дата обращения: 15.06.2026)

13. Get started with Bootstrap // Bootstrap [Электронный ресурс]. URL: <https://getbootstrap.com/docs/5.3/getting-started/introduction/> (дата обращения: 15.06.2026)

14. HTML: HyperText Markup Language // MDN Web Docs [Электронный ресурс]. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (дата обращения: 15.06.2026)

15. CSS: Cascading Style Sheets // MDN Web Docs [Электронный ресурс]. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (дата обращения: 15.06.2026)

16. JavaScript // MDN Web Docs [Электронный ресурс]. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (дата обращения: 15.06.2026)

17. Overview of ASP.NET Core Authentication // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-10.0> (дата обращения: 15.06.2026)

18. Role-based authorization in ASP.NET Core // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authorization/role-based?view=aspnetcore-10.0>

us/aspnet/core/security/authorization/roles?view=aspnetcore-10.0 (дата обращения: 15.06.2026)

19. Migrations Overview // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/> (дата обращения: 15.06.2026)

20. Unit testing C# in .NET using dotnet test and xUnit // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-xunit> (дата обращения: 15.06.2026)

21. EF Core In-Memory Database Provider // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/providers/in-memory/> (дата обращения: 15.06.2026)

22. lizard // PyPI [Электронный ресурс]. URL: <https://pypi.org/project/lizard/> (дата обращения: 15.06.2026)

Приложение А
(справочное)
Глоссарий

База данных (БД) – организованная совокупность данных, предназначенная для хранения, обработки и получения сведений о пользователях, проживании, услугах, бронированиях и оплатах.

Веб-браузер – прикладная программа, через которую пользователь открывает страницы веб-приложения и выполняет действия в системе.

Веб-приложение – клиент-серверное программное решение, доступное через браузер и обрабатывающее запросы пользователей на стороне сервера.

Информационная система (ИС) – совокупность программных средств, данных и процедур, обеспечивающих сбор, хранение, обработку и предоставление информации пользователям.

СУБД (система управления базами данных) – программное средство для создания, изменения, защиты и использования базы данных.

Фреймворк – программная платформа, задающая структуру приложения и предоставляющая готовые механизмы для разработки типовых функций.

API (Application Programming Interface) – набор правил и программных средств, через которые отдельные части приложения обмениваются данными и вызывают функции друг друга.

JSON – текстовый формат обмена данными, применяемый для передачи структурированной информации между клиентской частью и сервером.

Microsoft SQL Server Management Studio – среда администрирования Microsoft SQL Server, используемая для работы с базой данных, выполнения запросов и просмотра таблиц.

.NET – платформа разработки, включающая среду выполнения, библиотеки и инструменты для создания приложений на языке C#.

ASP.NET Core MVC – веб-фреймворк платформы .NET, основанный на разделении приложения на модели, представления и контроллеры.

Entity Framework Core – объектно-реляционный механизм доступа к данным, позволяющий работать с таблицами базы данных через классы C#.

Razor Views – механизм формирования HTML-страниц в ASP.NET Core MVC с использованием шаблонов, в которые может встраиваться код C#.

Приложение Б
(обязательное)
Листинг серверной части системы

Листинг Б.1 – Enums.cs

```
namespace BerezkaResort.Web.Models;
```

```
public enum UserRole
{
    Vacationer = 1,
    Administrator = 2,
    ServiceManager = 3
}

public enum BookingStatus
{
    Active = 1,
    Cancelled = 2,
    Completed = 3
}
```

```
public enum PaymentStatus
{
    NotPaid = 1,
    PartiallyPaid = 2,
    Paid = 3
}
```

Листинг Б.2 – AppUser.cs

```
namespace BerezkaResort.Web.Models;
```

```
public class AppUser
{
    public int Id { get; set; }
    public string FullName { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
    public string Phone { get; set; } = string.Empty;
    public string PasswordHash { get; set; } = string.Empty;
    public UserRole Role { get; set; } = UserRole.Vacationer;
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    public ICollection<Booking> Bookings { get; set; } = new
List<Booking>();
    public ICollection<ServiceOrder> ServiceOrders { get; set; } =
new List<ServiceOrder>();
}
```

Листинг Б.3 – Accommodation.cs

```
namespace BerezkaResort.Web.Models;
```

```
public class Accommodation
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
```

```

    public AccommodationType Type { get; set; }
    public int Capacity { get; set; }
    public decimal PricePerDay { get; set; }
    public string Description { get; set; } = string.Empty;
    public bool IsAvailable { get; set; } = true;
    public string? PhotoUrl { get; set; }

    public ICollection<Booking> Bookings { get; set; } = new
List<Booking>();
    public ICollection<AccommodationPhoto> Photos { get; set; } =
new List<AccommodationPhoto>();
}

```

Листинг Б.4 – Booking.cs

```

namespace BerezkaResort.Web.Models;

public class Booking
{
    public int Id { get; set; }
    public int AccommodationId { get; set; }
    public Accommodation Accommodation { get; set; } = null!;
    public int? UserId { get; set; }
    public AppUser? User { get; set; }

    public string ClientFullName { get; set; } = string.Empty;
    public string ClientPhone { get; set; } = string.Empty;
    public string ClientEmail { get; set; } = string.Empty;
    public DateTime CheckInDate { get; set; }
    public DateTime CheckOutDate { get; set; }
    public decimal PricePerDay { get; set; }
    public int GuestsCount { get; set; }
    public BookingStatus Status { get; set; } =
BookingStatus.Active;
    public PaymentStatus PaymentStatus { get; set; } =
PaymentStatus.NotPaid;
}

```

Листинг Б.5 – ServiceSchedule.cs

```

namespace BerezkaResort.Web.Models;

public class ServiceSchedule
{
    public int Id { get; set; }
    public int ServiceId { get; set; }
    public Service Service { get; set; } = null!;
    public DateTime StartDateTime { get; set; }
    public DateTime EndDateTime { get; set; }
    public int AvailablePlaces { get; set; }
    public bool IsAvailable { get; set; } = true;
}

```

```

        public ICollection<ServiceOrder> ServiceOrders { get; set; } =
new List<ServiceOrder>();
    }

```

Листинг Б.6 – ServiceOrder.cs

```

namespace BerezkaResort.Web.Models;

public class ServiceOrder
{
    public int Id { get; set; }
    public int ServiceScheduleId { get; set; }
    public ServiceSchedule ServiceSchedule { get; set; } = null!;
    public int? BookingId { get; set; }
    public Booking? Booking { get; set; }
    public int? UserId { get; set; }
    public AppUser? User { get; set; }

    public int PlacesCount { get; set; }
    public decimal Price { get; set; }
    public OrderStatus Status { get; set; } = OrderStatus.Active;
    public PaymentStatus PaymentStatus { get; set; } =
PaymentStatus.NotPaid;
}

```

Листинг Б.7 – ApplicationDbContext.cs

```

public class
ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options) : DbContext(options)
{
    public DbSet<AppUser> Users => Set<AppUser>();
    public DbSet<Accommodation> Accommodations =>
Set<Accommodation>();
    public DbSet<AccommodationPhoto> AccommodationPhotos =>
Set<AccommodationPhoto>();
    public DbSet<Booking> Bookings => Set<Booking>();
    public DbSet<Service> Services => Set<Service>();
    public DbSet<ServicePhoto> ServicePhotos =>
Set<ServicePhoto>();
    public DbSet<ServiceSchedule> ServiceSchedules =>
Set<ServiceSchedule>();
    public DbSet<ServiceOrder> ServiceOrders =>
Set<ServiceOrder>();

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        ConfigureUsers(modelBuilder);
        ConfigureAccommodations(modelBuilder);
        ConfigureBookings(modelBuilder);
        ConfigureServices(modelBuilder);
    }
}

```

```

        ConfigureServiceSchedules(modelBuilder);
        ConfigureServiceOrders(modelBuilder);
    }
}

```

Листинг Б.8 – Program.cs

```

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("De
faultConnection")));
builder.Services.AddScoped<IBookingService, BookingService>();
builder.Services.AddScoped<IServiceOrderService,
ServiceOrderService>();
builder.Services.Configure<EmailOptions>(builder.Configuration.Get
Section("Email"));
builder.Services.AddScoped<IEmailSender, SmtplibEmailSender>();
builder.Services.AddScoped<IBookingEmailNotificationService,
BookingEmailNotificationService>();
builder.Services.AddHostedService<BookingReminderBackgroundService
>();
builder.Services.AddScoped<IPasswordHasher<AppUser>,
PasswordHasher<AppUser>>();
builder.Services.AddControllersWithViews();

var app = builder.Build();
await DatabaseSeeder.SeedAsync(app.Services);
app.UseAuthentication();
app.UseAuthorization();

```

Листинг Б.9 – BookingsController.cs

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(BookingCreateViewModel
model, CancellationToken cancellationToken)
{
    await FillAccommodationInfoAsync(model, cancellationToken);
    await FillCurrentUserInfoAsync(model, cancellationToken);
    await FillUnavailablePeriodsAsync(model, cancellationToken);

    if (!ModelState.IsValid)
    {
        return View(model);
    }

    var result = await bookingService.CreateBookingAsync(
        new CreateBookingRequest(
            model.AccommodationId, GetCurrentUserId(),
            model.ClientFullName,

```

```

        model.ClientPhone, model.ClientEmail,
model.CheckInDate,
        model.CheckOutDate, model.GuestsCount, model.Comment),
        cancellationToken);

    return result.IsSuccess && result.Value is not null
        ? RedirectToAction(nameof(Confirmation), new { id =
result.Value.Id })
        : View(model);
}

```

Листинг Б.10 – BookingService.cs

```

public async Task<OperationResult<Booking>> CreateBookingAsync(
    CreateBookingRequest request,
    Cancellation_token cancellationToken = default)
{
    var checkInDate = request.CheckInDate.Date;
    var checkOutDate = request.CheckOutDate.Date;

    if (checkInDate >= checkOutDate)
    {
        return OperationResult<Booking>.Failure("Дата выезда
должна быть позже даты заезда.");
    }

    if (checkInDate < DateTime.Today)
    {
        return OperationResult<Booking>.Failure("Дата заезда не
может быть раньше текущей даты.");
    }

    await using var transaction = await
dbContext.Database.BeginTransactionAsync(
        IsolationLevel.Serializable, cancellationToken);

    var accommodation = await dbContext.Accommodations
        .FirstOrDefaultAsync(x => x.Id == request.AccommodationId,
cancellationToken);

    if (accommodation is null || !accommodation.IsAvailable)
    {
        return OperationResult<Booking>.Failure("Объект проживания
недоступен.");
    }

    var booking = new Booking
    {
        AccommodationId = accommodation.Id,
        UserId = request.UserId,
        ClientFullName = request.ClientFullName.Trim(),
        CheckInDate = checkInDate,
        CheckOutDate = checkOutDate,
    }
}

```

```

        PricePerDay = accommodation.PricePerDay,
        Status = BookingStatus.Active,
        PaymentStatus = PaymentStatus.NotPaid
    };

    dbContext.Bookings.Add(booking);
    await dbContext.SaveChangesAsync(cancellationToken);
    await transaction.CommitAsync(cancellationToken);
    return OperationResult<Booking>.Success(booking);
}

```

Листинг Б.11 – ServiceOrderService.cs

```

public async Task<int> GetAvailablePlacesAsync(
    int serviceScheduleId,
    CancellationToken cancellationToken = default)
{
    var schedule = await dbContext.ServiceSchedules
        .Include(x => x.Service)
        .AsNoTracking()
        .FirstOrDefaultAsync(x => x.Id == serviceScheduleId,
            cancellationToken);

    if (schedule is null || !schedule.IsAvailable ||
        !schedule.Service.IsActive
        || schedule.StartDateTime <= DateTime.Now)
    {
        return 0;
    }

    if (IsWholeObjectService(schedule.Service.Name))
    {
        var hasActiveOrder = await
            dbContext.ServiceOrders.AnyAsync(
                order => order.ServiceScheduleId == serviceScheduleId
                && order.Status != OrderStatus.Cancelled,
                cancellationToken);

        return hasActiveOrder ? 0 : 1;
    }

    var reservedPlaces = await
        GetReservedPlacesAsync(serviceScheduleId, cancellationToken);
    return Math.Max(0, schedule.AvailablePlaces - reservedPlaces);
}

```

Листинг Б.12 – CustomerController.cs

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> PayBooking(int id,
    CancellationToken cancellationToken)
{

```

```

var userId = GetCurrentUserId();
if (!userId.HasValue)
{
    return Challenge();
}

var booking = await dbContext.Bookings
    .Include(x => x.Accommodation)
    .FirstOrDefaultAsync(x => x.Id == id && x.UserId ==
userId.Value, cancellationTokens);

if (booking is null)
{
    return NotFound();
}

if (booking.Status == BookingStatus.Cancelled)
{
    return RedirectToAction(nameof(Index), new { tab =
"cancelled" });
}

booking.PaymentStatus = PaymentStatus.Paid;
await dbContext.SaveChangesAsync(cancellationTokens);
await
emailNotificationService.SendBookingPaymentConfirmationAsync(booking,
cancellationTokens);

return RedirectToAction(nameof(Index));
}

```

Листинг Б.13 – SmtпEmailSender.cs

```

public async Task<bool> SendAsync(EmailMessage message,
CancellationTokens cancellationTokens)
{
    var emailOptions = options.Value;

    if (!emailOptions.Enabled ||
string.IsNullOrEmpty(message.ToEmail))
    {
        return false;
    }

    using var mailMessage = new MailMessage
    {
        From = new MailAddress(emailOptions.FromEmail,
emailOptions.FromName, Encoding.UTF8),
        Subject = message.Subject,
        Body = message.Body,
        BodyEncoding = Encoding.UTF8,
        SubjectEncoding = Encoding.UTF8,
        IsBodyHtml = false
    }
}

```

```
};

    mailMessage.To.Add(new MailAddress(message.ToEmail,
message.ToName, Encoding.UTF8));
    using var client = CreateClient(emailOptions);
    await client.SendMailAsync(mailMessage, cancellationTokens);
    return true;
}
```

Приложение В
(обязательное)
Листинг клиентской части системы

Листинг В.1 – _Layout.cshtml

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>@ViewData["Title"] - Бездка</title>
    <link rel="stylesheet"
href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-
version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-lg navbar-light app-navbar">
            <a class="navbar-brand brand-mark" asp-controller="Home" asp-
action="Index">Бездка</a>
            <ul class="navbar-nav ms-auto">
                <li><a class="nav-link" asp-controller="Home" asp-
action="Index">Главная</a></li>
                @if (!User.IsInRole("ServiceManager"))
                {
                    <li><a class="nav-link" asp-
controller="Accommodations" asp-action="Index">Проживание</a></li>
                }
                <li><a class="nav-link" asp-controller="Services" asp-
action="Index">Услуги</a></li>
                @if (User.IsInRole("Administrator"))
                {
                    <li><a class="nav-link" asp-controller="Admin" asp-
action="Index">Панель администратора</a></li>
                }
                @if (User.IsInRole("ServiceManager"))
                {
                    <li><a class="nav-link" asp-controller="Admin" asp-
action="Services">Панель менеджера услуг</a></li>
                }
                @if (User.IsInRole("Vacationer"))
                {
                    <li><a class="nav-link" asp-controller="Customer"
asp-action="Index">Мои бронирования</a></li>
                }
            </ul>
        </nav>
    </header>
    <main role="main">
        @RenderBody()
    </main>
    <script
src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
</body>
</html>
```

Листинг В.2 – Register.cshtml

```
@model RegisterViewModel
```

```

@{
    ViewData["Title"] = "Регистрация";
}

<section class="auth-layout">
    <form class="auth-panel" asp-controller="Account" asp-
action="Register" method="post">
        @Html.AntiForgeryToken()
        <h2>Создать аккаунт</h2>
        <div asp-validation-summary="ModelOnly" class="alert alert-
danger" role="alert"></div>

        <div class="mb-3">
            <label asp-for="FullName" class="form-label"></label>
            <input asp-for="FullName" class="form-control"
autocomplete="name" data-full-name />
            <span asp-validation-for="FullName" class="text-
danger"></span>
        </div>

        <div class="mb-3">
            <label asp-for="Phone" class="form-label"></label>
            <input asp-for="Phone"
                class="form-control"
                autocomplete="tel"
                inputmode="tel"
                maxlength="18"
                placeholder="+7 (999) 123-45-67"
                data-russian-phone />
            <span asp-validation-for="Phone" class="text-danger"></span>
        </div>

        <div class="mb-3">
            <label asp-for="Email" class="form-label"></label>
            <input asp-for="Email" class="form-control"
autocomplete="email" />
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>

        <button class="btn btn-success btn-lg w-100"
type="submit">Зарегистрироваться</button>
    </form>
</section>

```

Листинг В.3 – Create.cshtml

```

@model BookingCreateViewModel

<aside class="summary-panel">
    <h2>Выбранный вариант</h2>
    <dl class="summary-list">
        <dt>Название</dt>
        <dd>@Model.AccommodationName</dd>
        <dt>Вместимость</dt>
        <dd>@Model.AccommodationCapacity чел.</dd>
        <dt>Стоимость</dt>

```

```

        <dd>@Model.PricePerDay.ToString("N0") руб./сутки</dd>
    </dl>
    <ul class="availability-list" data-booking-unavailable>
        @foreach (var period in Model.UnavailablePeriods)
        {
            <li data-start="@period.CheckInDate.ToString("yyyy-MM-dd") "
                data-end="@period.CheckOutDate.ToString("yyyy-MM-dd") ">
                @period.CheckInDate.ToString("dd.MM.yyyy") -
                @period.CheckOutDate.ToString("dd.MM.yyyy")
            </li>
        }
    </ul>
</aside>

<form class="booking-form" asp-action="Create" method="post" data-
booking-form>
    @Html.AntiForgeryToken()
    <input asp-for="AccommodationId" type="hidden" />

    <label asp-for="ClientFullName" class="form-label"></label>
    <input asp-for="ClientFullName" class="form-control"
autocomplete="name" />

    <label asp-for="CheckInDate" class="form-label"></label>
    <input asp-for="CheckInDate"
        class="form-control"
        min="@DateTime.Today.ToString("yyyy-MM-dd") "
        data-booking-check-in />

    <label asp-for="CheckOutDate" class="form-label"></label>
    <input asp-for="CheckOutDate"
        class="form-control"
        min="@DateTime.Today.AddDays(1).ToString("yyyy-MM-dd") "
        data-booking-check-out />

    <div class="alert alert-warning d-none" data-booking-availability-
message></div>
    <button class="btn btn-success btn-lg" type="submit" data-booking-
submit>Создать бронь</button>
</form>

```

Листинг В.4 – site.js

```

document.addEventListener("input", (event) => {
    const target = event.target;

    if (!(target instanceof HTMLInputElement)) {
        return;
    }

    if (target.matches("[data-payment-card]")) {
        const digits = target.value.replace(/\D/g, "").slice(0,
16);
        target.value = digits.replace(/(.{4})/g, "$1 ").trim();
        return;
    }
}

```

```

    if (target.matches("[data-payment-cvv]")) {
        target.value = target.value.replace(/\D/g, "").slice(0,
3);
        return;
    }

    if (target.matches("[data-payment-holder]")) {
        target.value = target.value
            .replace(/[^A-Za-z\s]/g, "")
            .replace(/\s{2,}/g, " ")
            .toUpperCase()
            .slice(0, 80);
        return;
    }

    if (target.matches("[data-russian-phone]")) {
        target.value = formatRussianPhoneInput(target.value);
    }
});

function initBookingAvailability() {
    const form = document.querySelector("[data-booking-form]");

    if (!form) {
        return;
    }

    const checkInInput = form.querySelector("[data-booking-check-
in]");
    const checkOutInput = form.querySelector("[data-booking-check-
out]");
    const submitButton = form.querySelector("[data-booking-
submit]");
    const message = form.querySelector("[data-booking-
availability-message]");
    const periods = Array.from(document.querySelectorAll("[data-
booking-unavailable] [data-start] [data-end]"))
        .map((item) => ({
            start: parseDateOnly(item.dataset.start),
            end: parseDateOnly(item.dataset.end),
            label: item.textContent.trim()
        }));

    const validateDates = () => {
        const checkIn = parseDateOnly(checkInInput.value);
        const checkOut = parseDateOnly(checkOutInput.value);
        const overlappingPeriod = periods.find((period) => checkIn
< period.end && checkOut > period.start);

        if (overlappingPeriod) {

```

```

        message.textContent = `Выбранные даты пересекаются с
занятым периодом: ${overlappingPeriod.label}.`;
        message.classList.remove("d-none");
        submitButton.disabled = true;
        return false;
    }

    message.classList.add("d-none");
    submitButton.disabled = false;
    return true;
};

checkInInput.addEventListener("change", validateDates);
checkOutInput.addEventListener("change", validateDates);
form.addEventListener("submit", (event) => {
    if (!validateDates()) {
        event.preventDefault();
    }
});
}

```

Приложение Г

(обязательное)

Листинг файла BookingServiceTests.cs

```

using BerezkaResort.Web.Data;
using BerezkaResort.Web.Models;
using BerezkaResort.Web.Services;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Diagnostics;
using Xunit;

namespace BerezkaResort.Tests.Services;

public class BookingServiceTests
{
    private static readonly DateTime FutureCheckIn =
DateTime.Today.AddDays(30);
    private static readonly DateTime FutureCheckOut =
DateTime.Today.AddDays(33);

    [Fact]
    public async Task
CreateBookingAsync_ReturnsFailure_WhenCheckOutDateIsNotAfterChec
kInDate()
    {
        await using var dbContext = CreateDbContext();
        await SeedAccommodationAsync(dbContext);
        var service = new BookingService(dbContext);

        var result = await
service.CreateBookingAsync(CreateRequest(
            checkInDate: FutureCheckIn,
            checkOutDate: FutureCheckIn));

        Assert.False(result.IsSuccess);
        Assert.Contains("Дата выезда", result.ErrorMessage);
    }

    [Fact]
    public async Task
CreateBookingAsync_ReturnsFailure_WhenCheckInDateIsInPast()
    {
        await using var dbContext = CreateDbContext();
        await SeedAccommodationAsync(dbContext);
        var service = new BookingService(dbContext);

        var result = await
service.CreateBookingAsync(CreateRequest(
            checkInDate: DateTime.Today.AddDays(-1),
            checkOutDate: DateTime.Today.AddDays(2)));

        Assert.False(result.IsSuccess);
        Assert.Contains("Дата заезда", result.ErrorMessage);
    }

    [Fact]

```

```

        public async Task
        CreateBookingAsync_ReturnsFailure_WhenActiveBookingOverlaps()
        {
            await using var dbContext = CreateDbContext();
            await SeedAccommodationAsync(dbContext);
            await SeedBookingAsync(dbContext,
                FutureCheckIn.AddDays(1), FutureCheckOut.AddDays(1));
            var service = new BookingService(dbContext);

            var result = await
            service.CreateBookingAsync(CreateRequest());

            Assert.False(result.IsSuccess);
            Assert.Contains("уже забронирован",
                result.ErrorMessage);
        }

        [Fact]
        public async Task
        CreateBookingAsync_ReturnsSuccess_WhenOverlappingBookingIsCancelled()
        {
            await using var dbContext = CreateDbContext();
            await SeedAccommodationAsync(dbContext);
            await SeedBookingAsync(
                dbContext,
                FutureCheckIn.AddDays(1),
                FutureCheckOut.AddDays(1),
                BookingStatus.Cancelled);
            var service = new BookingService(dbContext);

            var result = await
            service.CreateBookingAsync(CreateRequest());

            Assert.True(result.IsSuccess);
        }

        [Fact]
        public async Task
        CreateBookingAsync_CreatesBookingWithActiveAndNotPaidStatuses()
        {
            await using var dbContext = CreateDbContext();
            await SeedAccommodationAsync(dbContext);
            var service = new BookingService(dbContext);

            var result = await
            service.CreateBookingAsync(CreateRequest());

            Assert.True(result.IsSuccess);
            Assert.NotNull(result.Value);
            Assert.Equal(BookingStatus.Active, result.Value.Status);
        }

```

```

        Assert.Equal(PaymentStatus.NotPaid,
result.Value.PaymentStatus);
    }

    [Fact]
    public async Task
CreateBookingAsync_TrimsClientDataAndComment()
    {
        await using var dbContext = CreateDbContext();
        await SeedAccommodationAsync(dbContext);
        var service = new BookingService(dbContext);

        var result = await
service.CreateBookingAsync(CreateRequest(
    clientFullName: "  Алешин Вячеслав Сергеевич  ",
    clientPhone: "  +7 (999) 111-22-33  ",
    clientEmail: "  user@example.com  ",
    comment: "  Поздний заезд  "));

        Assert.True(result.IsSuccess);
        Assert.Equal("Алешин Вячеслав Сергеевич",
result.Value?.ClientFullName);
        Assert.Equal("+7 (999) 111-22-33",
result.Value?.ClientPhone);
        Assert.Equal("user@example.com",
result.Value?.ClientEmail);
        Assert.Equal("Поздний заезд", result.Value?.Comment);
    }

    private static ApplicationDbContext CreateDbContext()
    {
        var options = new
DbContextOptionsBuilder<ApplicationDbContext>()
            .UseInMemoryDatabase(Guid.NewGuid().ToString())
            .ConfigureWarnings(warnings =>
warnings.Ignore(InMemoryEventId.TransactionIgnoredWarning))
            .Options;

        return new ApplicationDbContext(options);
    }

    private static async Task SeedAccommodationAsync(
        ApplicationDbContext dbContext,
        int id = 1,
        int capacity = 4,
        bool isAvailable = true,
        decimal pricePerDay = 4500m)
    {
        dbContext.Accommodations.Add(new Accommodation
        {
            Id = id,
            Name = "Домик Березовый",

```

```

        Type = AccommodationType.House,
        Capacity = capacity,
        PricePerDay = pricePerDay,
        Description = "Домик для тестирования
бронирования.",
        IsAvailable = isAvailable
    });

    await dbContext.SaveChangesAsync();
}

private static async Task SeedBookingAsync(
    ApplicationDbContext dbContext,
    DateTime checkInDate,
    DateTime checkOutDate,
    BookingStatus status = BookingStatus.Active)
{
    dbContext.Bookings.Add(new Booking
    {
        AccommodationId = 1,
        ClientFullName = "Тестовый Клиент",
        ClientPhone = "+7 (999) 000-00-00",
        ClientEmail = "client@example.com",
        CheckInDate = checkInDate.Date,
        CheckOutDate = checkOutDate.Date,
        PricePerDay = 4500m,
        GuestsCount = 2,
        Status = status,
        PaymentStatus = PaymentStatus.NotPaid,
        CreatedAt = DateTime.UtcNow
    });

    await dbContext.SaveChangesAsync();
}

private static CreateBookingRequest CreateRequest(
    int accommodationId = 1,
    string clientFullName = "Алешин Вячеслав Сергеевич",
    string clientPhone = "+7 (999) 111-22-33",
    string clientEmail = "user@example.com",
    DateTime? checkInDate = null,
    DateTime? checkOutDate = null,
    int guestsCount = 2,
    string? comment = null)
{
    return new CreateBookingRequest(
        accommodationId,
        UserId: null,
        clientFullName,
        clientPhone,
        clientEmail,
        checkInDate ?? FutureCheckIn,

```

```
        checkOutDate ?? FutureCheckOut,  
        guestsCount,  
        comment);  
    }  
}
```

Приложение Д
(рекомендуемое)
Атрибуты сущностей базы данных

Таблица Д.1 – Атрибуты отношения «Users»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор пользователя	integer	10	от 1 до 2 147 483 647	нет	PK	1
FullName	ФИО пользователя	nvarchar	200	буквы, пробелы, дефис	нет		Алешин Вячеслав Сергеевич
Email	Адрес электронной почты пользователя	nvarchar	200	строка, формат email	нет	UQ	user@example.com
Phone	Номер телефона пользователя	nvarchar	30	строка, формат +7 (XXX) XXX-XX-XX	нет		+7 (999) 111-22-33
PasswordHash	Хеш пароля пользователя	nvarchar	500	строка, сформированная PasswordHasher	нет		AQAAAA...
Role	Роль пользователя в системе	nvarchar	32	Vacationer, Administrator, ServiceManager	нет		Vacationer
CreatedAt	Дата и время создания учетной записи	datetime2		01.01.2020 – 01.01.2100	нет		2026-05-20 12:00

Таблица Д.2 – Атрибуты отношения «Accommodations»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор объекта проживания	integer	10	от 1 до 2 147 483 647	нет	PK	1
Name	Название объекта проживания	nvarchar	150	строка до 150 символов	нет		Домик Березовый
Type	Тип объекта проживания	nvarchar	32	House, Room	нет		House
Capacity	Вместимость объекта проживания	integer	10	целое число больше 0	нет		4
PricePerDay	Стоимость проживания за сутки	decimal	18,2	число не меньше 0	нет		4500.00

Продолжение таблицы Д.2

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Description	Описание объекта проживания	nvarchar	2000	строка до 2000 символов	нет		Отдельный домик для семейного отдыха
IsAvailable	Признак доступности для бронирования	bit		0 – нет, 1 – да	нет		1
PhotoUrl	Основная ссылка на фотографию	nvarchar	500	путь или URL изображения	да		/uploads/accommodations/house.jpg

Таблица Д.3 – Атрибуты отношения «AccommodationPhotos»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор фотографии проживания	integer	10	от 1 до 2 147 483 647	нет	PK	1
AccommodationId	Идентификатор объекта проживания	integer	10	от 1 до 2 147 483 647	нет	FK	1
ImageUrl	Ссылка на изображение объекта проживания	nvarchar	500	путь или URL изображения	нет		/uploads/accommodations/photo1.jpg
Caption	Подпись к фотографии	nvarchar	200	строка до 200 символов	да		Вид снаружи
SortOrder	Порядок вывода фотографии	integer	10	целое число от 0	нет		1

Таблица Д.4 – Атрибуты отношения «Bookings»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор бронирования проживания	integer	10	от 1 до 2 147 483 647	нет	PK	1
AccommodationId	Идентификатор объекта проживания	integer	10	от 1 до 2 147 483 647	нет	FK	1
UserId	Идентификатор пользователя, оформившего бронь	integer	10	от 1 до 2 147 483 647	да	FK	1
ClientFullName	ФИО клиента	nvarchar	200	буквы, пробелы, дефис	нет		Алешин Вячеслав Сергеевич
ClientPhone	Контактный телефон клиента	nvarchar	30	строка, формат +7 (XXX) XXX-XX-XX	нет		+7 (999) 111-22-33
ClientEmail	Электронная почта клиента	nvarchar	200	строка, формат email	нет		user@example.com
CheckInDate	Дата заезда	datetime 2		не раньше текущей даты	нет		2026-06-20
CheckOutDate	Дата выезда	datetime 2		позже даты заезда	нет		2026-06-23
PricePerDay	Цена за сутки на момент бронирования	decimal	18,2	число не меньше 0	нет		4500.00
GuestsCount	Количество гостей	integer	10	целое число больше 0	нет		2
Comment	Комментарий клиента к бронированию	nvarchar	1000	строка до 1000 символов	да		Поздний заезд
Status	Статус бронирования проживания	nvarchar	32	Active, Cancelled, Completed	нет		Active
PaymentStatus	Статус оплаты бронирования	nvarchar	32	NotPaid, PartiallyPaid, Paid	нет		NotPaid
CreatedAt	Дата и время создания бронирования	datetime 2		01.01.2020 – 01.01.2100	нет		2026-06-01 10:30
ReminderSentAt	Дата отправки напоминания о брони	datetime 2		01.01.2020 – 01.01.2100	да		2026-06-19 09:00

Таблица Д.5 – Атрибуты отношения «Services»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор услуги	integer	10	от 1 до 2 147 483 647	нет	PK	1
Name	Название услуги	nvarchar	150	строка до 150 символов	нет		Баня
Description	Описание услуги	nvarchar	2000	строка до 2000 символов	нет		Бронирование бани на выбранное время
Price	Стоимость услуги	decimal	18,2	число не меньше 0	нет		3000.00
Capacity	Количество доступных мест	integer	10	целое число больше 0	нет		6
DurationMinutes	Длительность услуги в минутах	integer	10	целое число больше 0	нет		120
IsActive	Признак активности услуги	bit		0 – нет, 1 – да	нет		1

Таблица Д.6 – Атрибуты отношения «ServicePhotos»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор фотографии услуги	integer	10	от 1 до 2 147 483 647	нет	PK	1
ServiceId	Идентификатор услуги	integer	10	от 1 до 2 147 483 647	нет	FK	1
ImageUrl	Ссылка на изображение услуги	nvarchar	500	путь или URL изображения	нет		/uploads/services/banya.jpg
Caption	Подпись к фотографии	nvarchar	200	строка до 200 символов	да		Парная
SortOrder	Порядок вывода фотографии	integer	10	целое число от 0	нет		1

Таблица Д.7 – Атрибуты отношения «ServiceSchedules»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор сеанса услуги	integer	10	от 1 до 2 147 483 647	нет	PK	1
ServiceId	Идентификатор услуги	integer	10	от 1 до 2 147 483 647	нет	FK	1
StartDateTime	Дата и время начала сеанса	datetime 2		01.01.2020 – 01.01.2100	нет		2026-06-20 12:00
EndDateTime	Дата и время окончания сеанса	datetime 2		позже времени начала	нет		2026-06-20 14:00
AvailablePlaces	Количество мест на сеанс	integer	10	целое число больше 0	нет		6
IsAvailable	Признак доступности сеанса	bit		0 – нет, 1 – да	нет		1

Таблица Д.8 – Атрибуты отношения «ServiceOrders»

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор заказа услуги	integer	10	от 1 до 2 147 483 647	нет	PK	1
ServiceScheduleId	Идентификатор сеанса услуги	integer	10	от 1 до 2 147 483 647	нет	FK	1
BookingId	Идентификатор связанной брони проживания	integer	10	от 1 до 2 147 483 647	да	FK	1
UserId	Идентификатор пользователя, оформившего заказ	integer	10	от 1 до 2 147 483 647	да	FK	1
ClientFullName	ФИО клиента	nvarchar	200	буквы, пробелы, дефис	нет		Алешин Вячеслав Сергеевич
ClientPhone	Контактный телефон клиента	nvarchar	30	строка, формат +7 (XXX) XXX-XX-XX	нет		+7 (999) 111-22-33
ClientEmail	Электронная почта клиента	nvarchar	200	строка, формат email	нет		user@example.com
PlacesCount	Количество бронируемых мест	integer	10	целое число больше 0	нет		2
Price	Стоимость услуги на момент заказа	decimal	18,2	число не меньше 0	нет		3000.00

Продолжение таблицы Д.8

Идентификатор столбца	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Status	Статус заказа услуги	nvarchar	32	Active, Cancelled, Completed	нет		Active
PaymentStatus	Статус оплаты заказа услуги	nvarchar	32	NotPaid, PartiallyPaid, Paid	нет		NotPaid
Comment	Комментарий клиента к заказу	nvarchar	1000	строка до 1000 символов	да		Нужны два места
CreatedAt	Дата и время создания заказа	datetime 2		01.01.2020 – 01.01.2100	нет		2026-06-01 11:00
ReminderSentAt	Дата отправки напоминания о заказе	datetime 2		01.01.2020 – 01.01.2100	да		2026-06-19 09:00

Приложение Е
(рекомендуемое)
Графическая часть

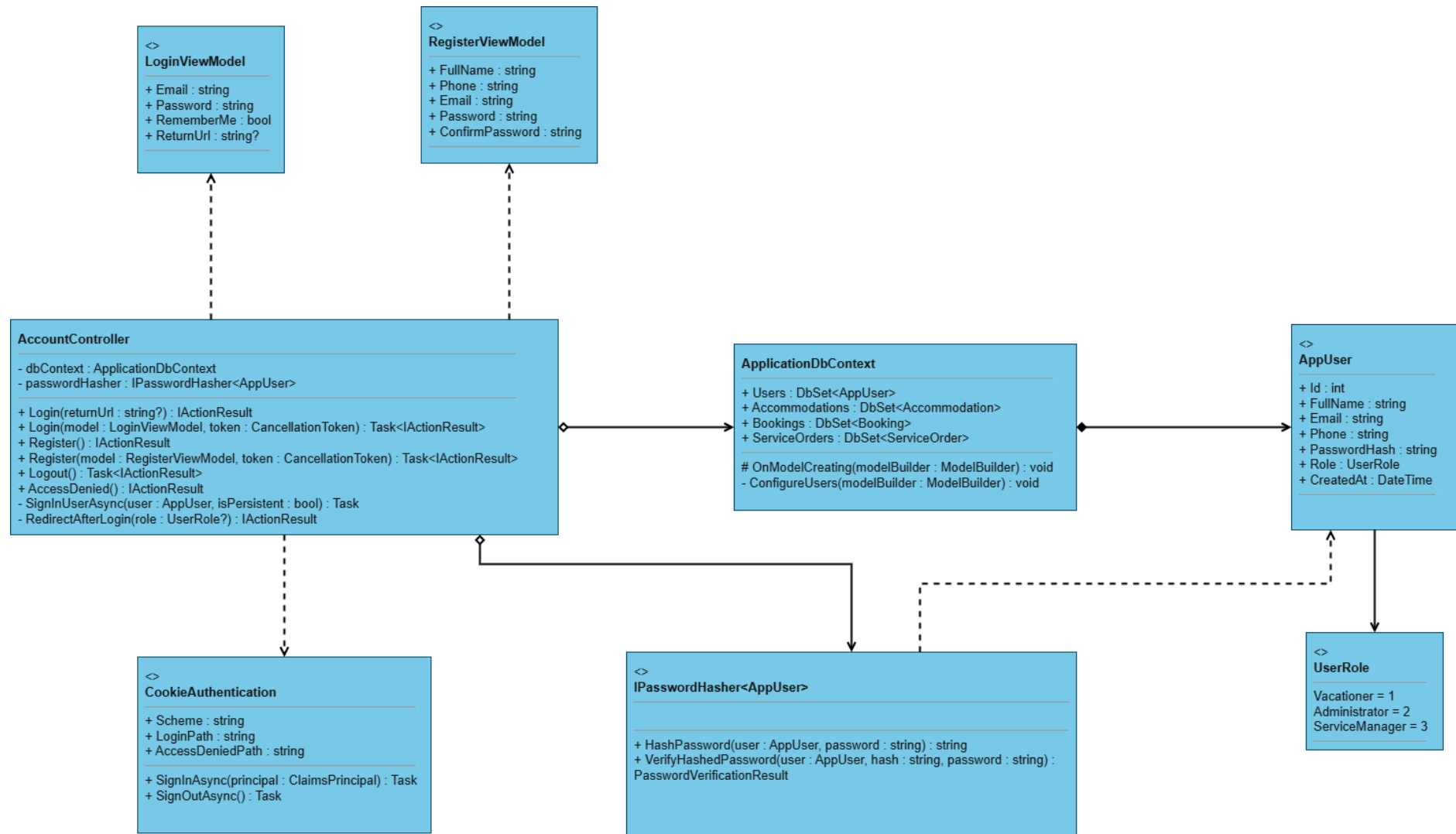


Рисунок Е.1 – Диаграмма классов модуля «Аутентификация»

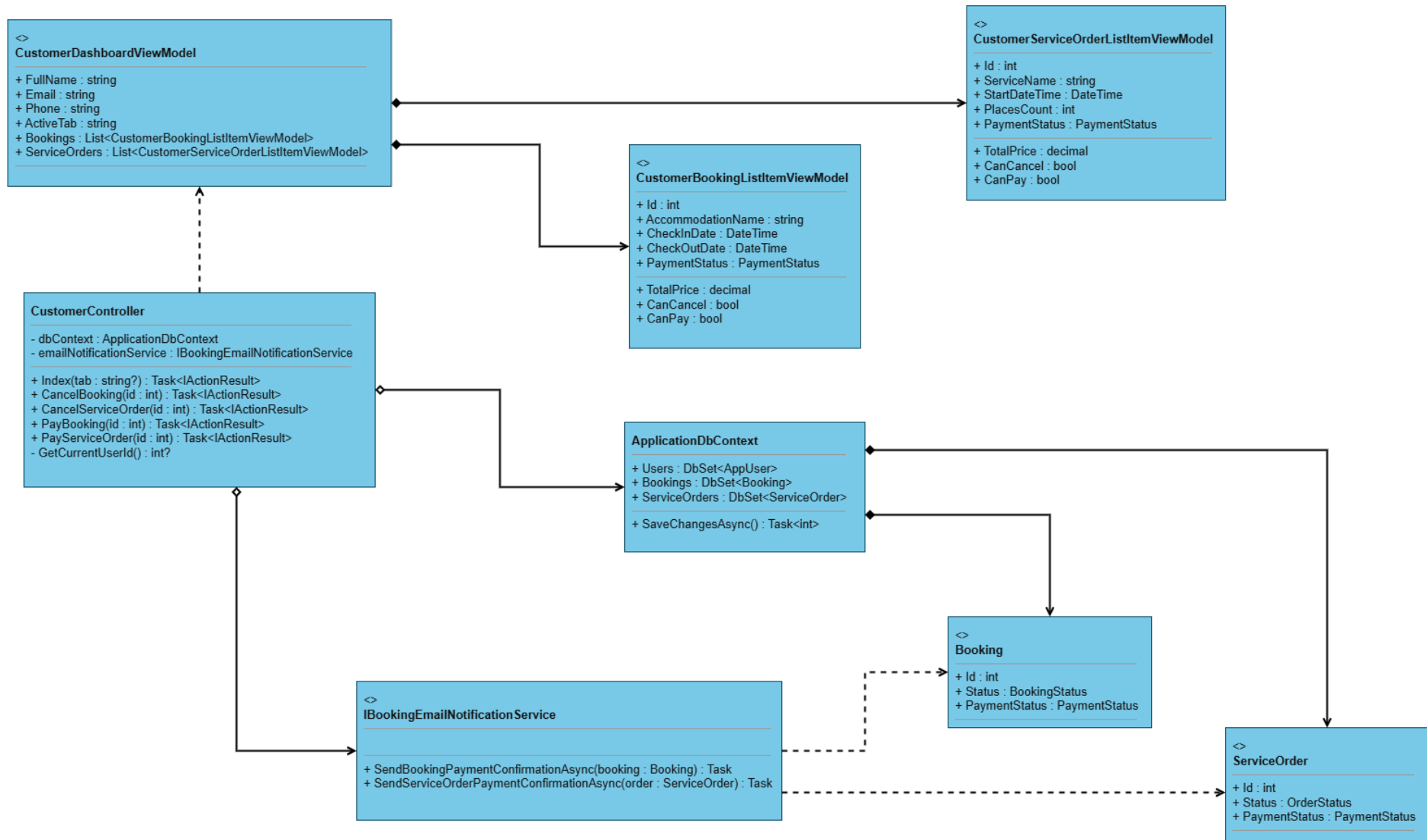


Рисунок Е.2 – Диаграмма классов модуля «Личный кабинет отдыхающего»

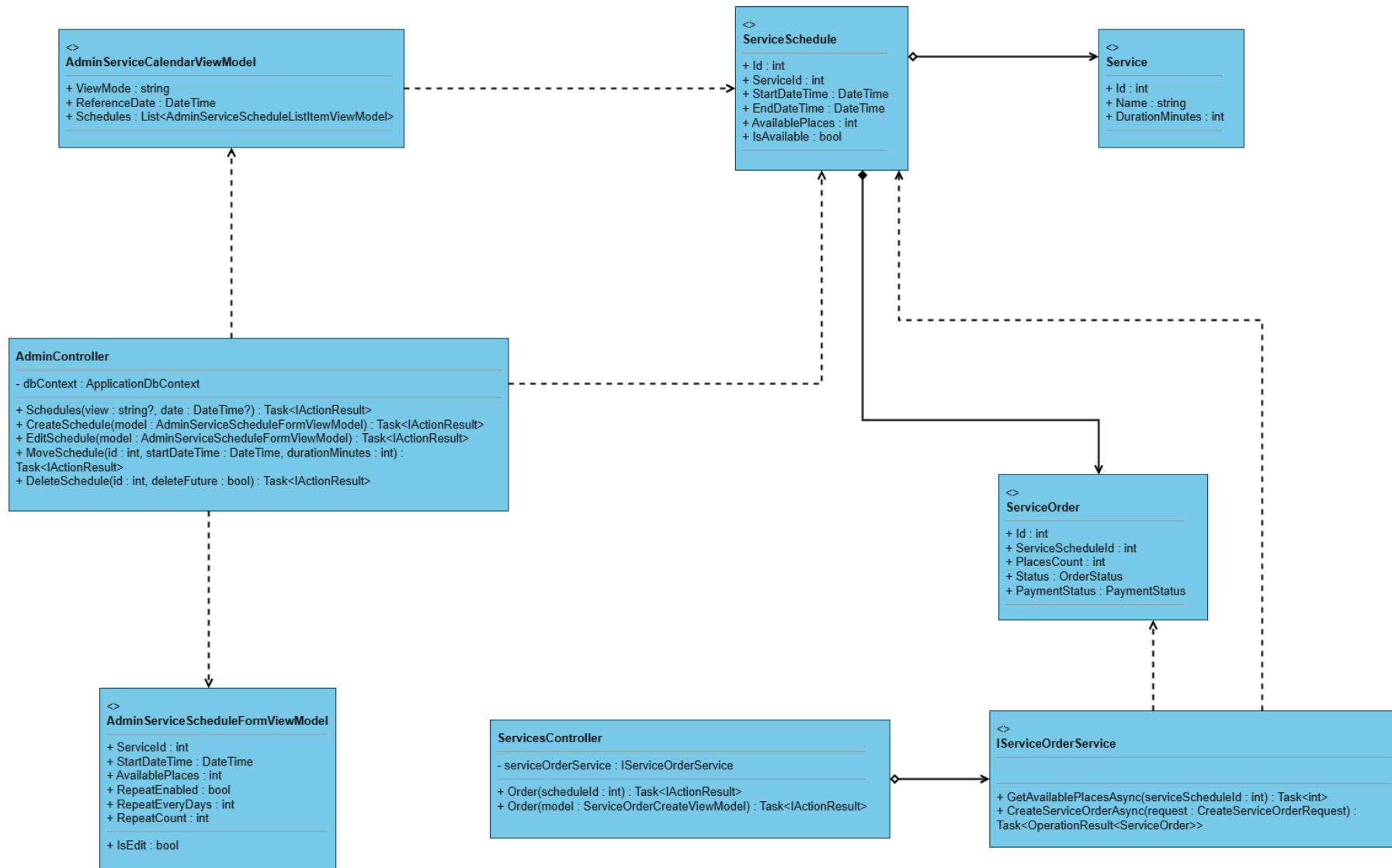


Рисунок Е.3 – Диаграмма классов модуля «Расписание услуг»

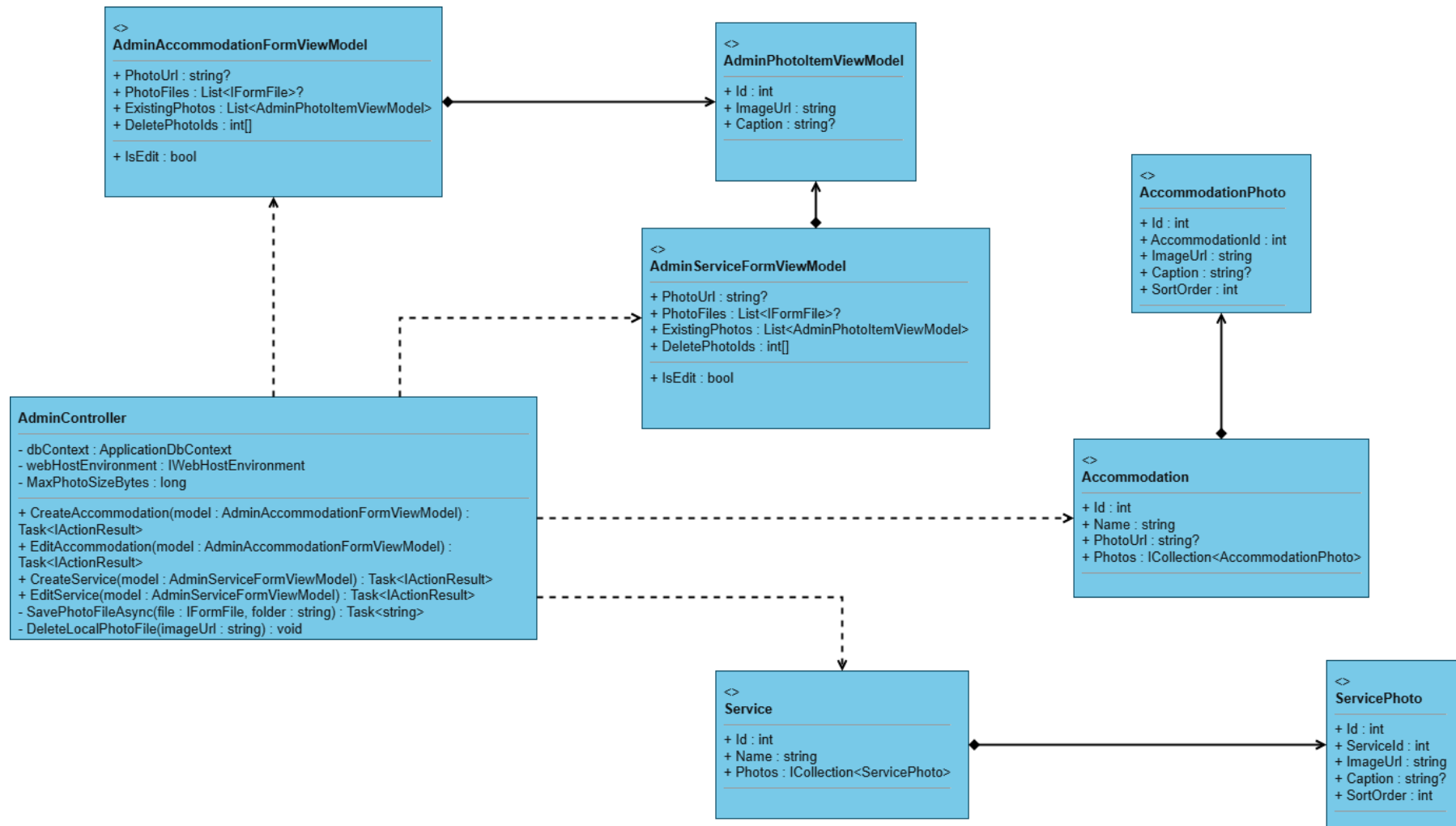


Рисунок Е.4 – Диаграмма классов модуля «Фотографии»

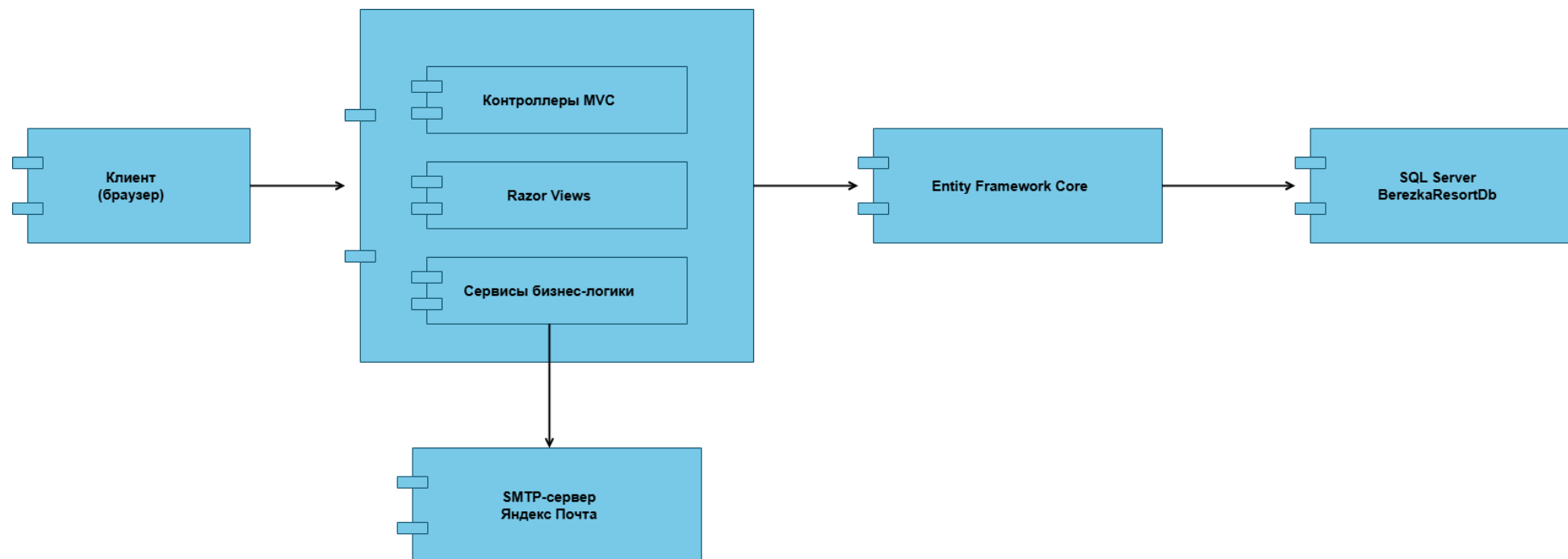


Рисунок Е.5 – Диаграмма компонентов системы

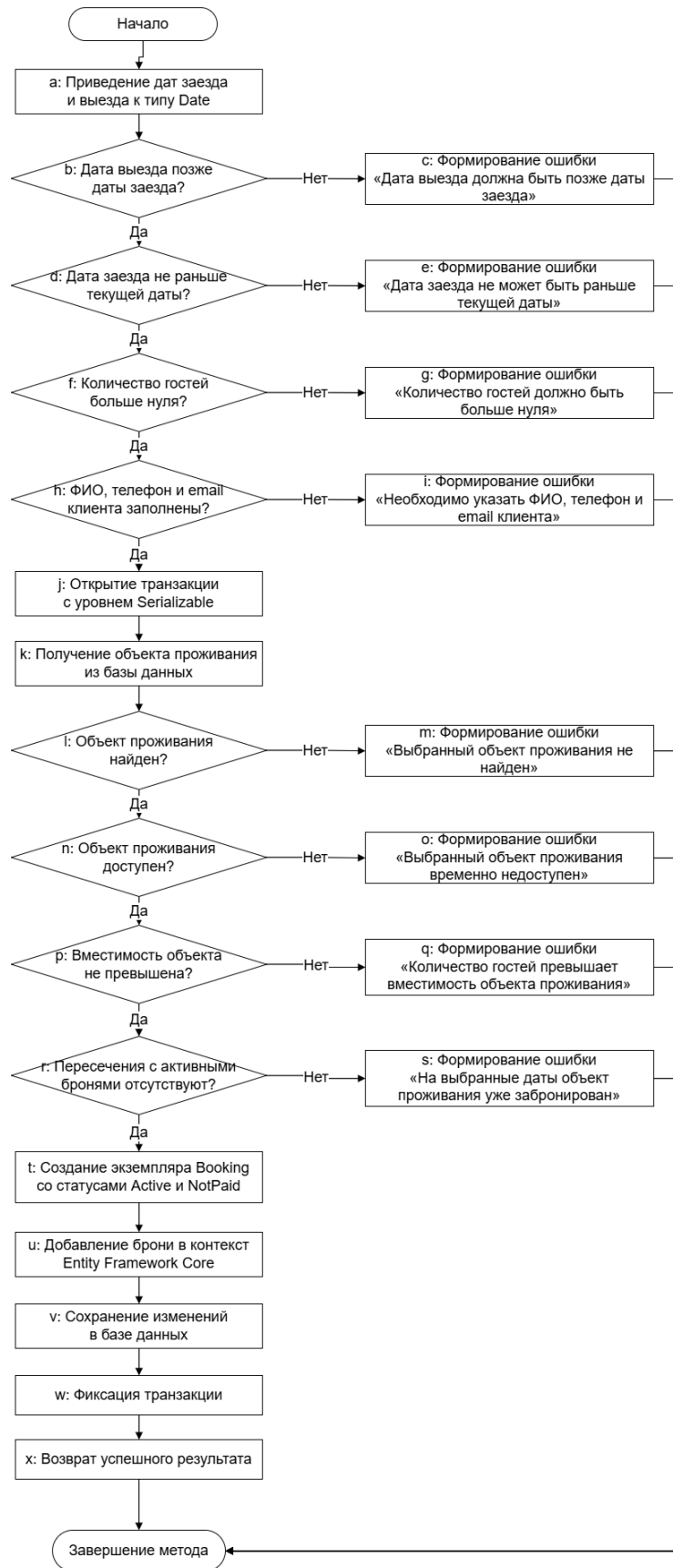


Рисунок Е.6 – Блок-схема метода CreateBookingAsync

Приложение Ж
(рекомендуемое)
Результаты функционального тестирования

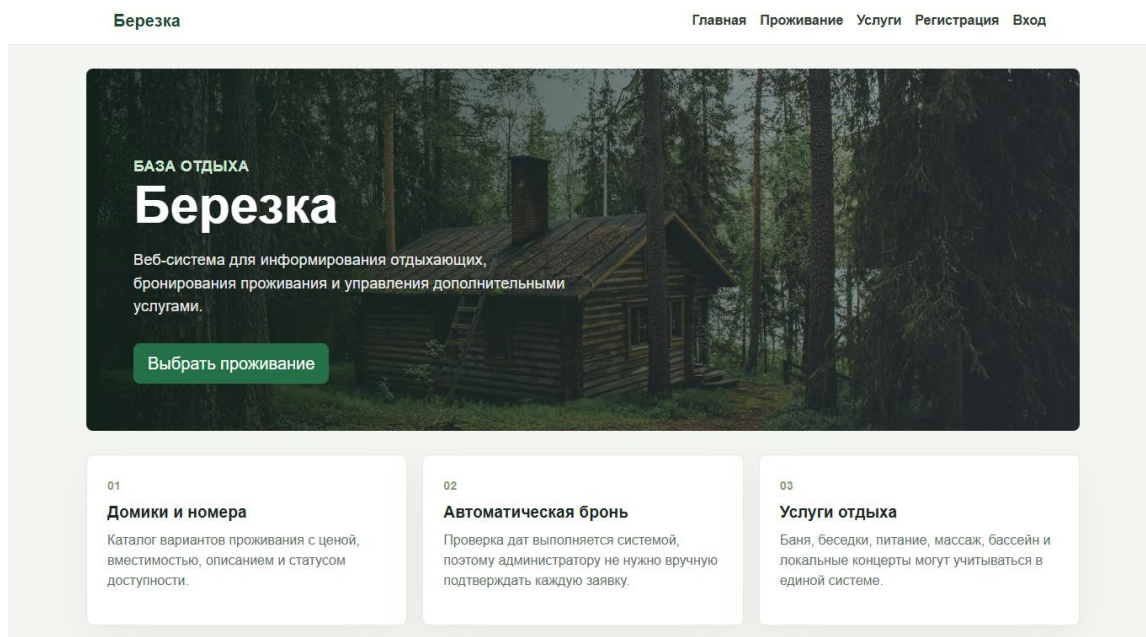


Рисунок Ж.1 – Главная публичная страница

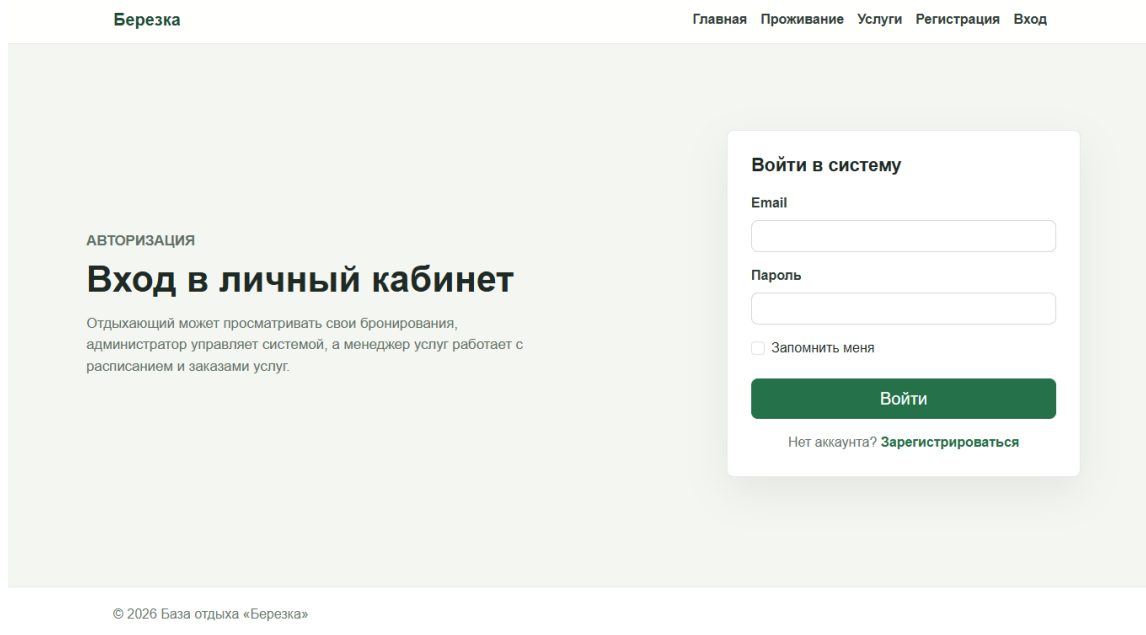


Рисунок Ж.2 – Страница авторизации пользователя

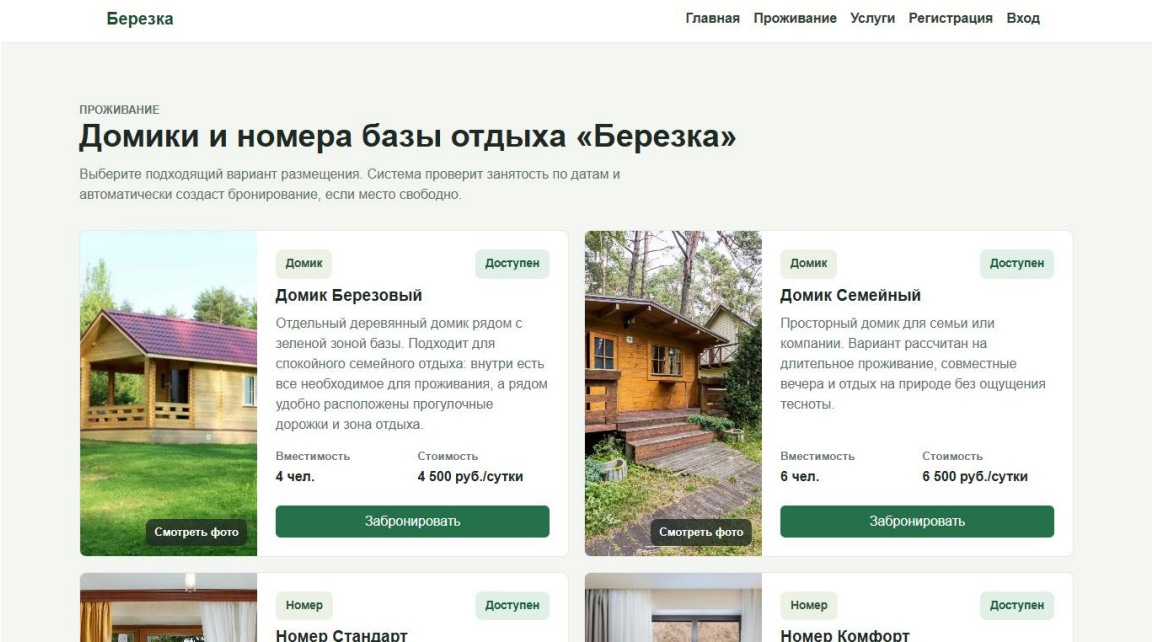


Рисунок Ж.3 – Каталог объектов проживания

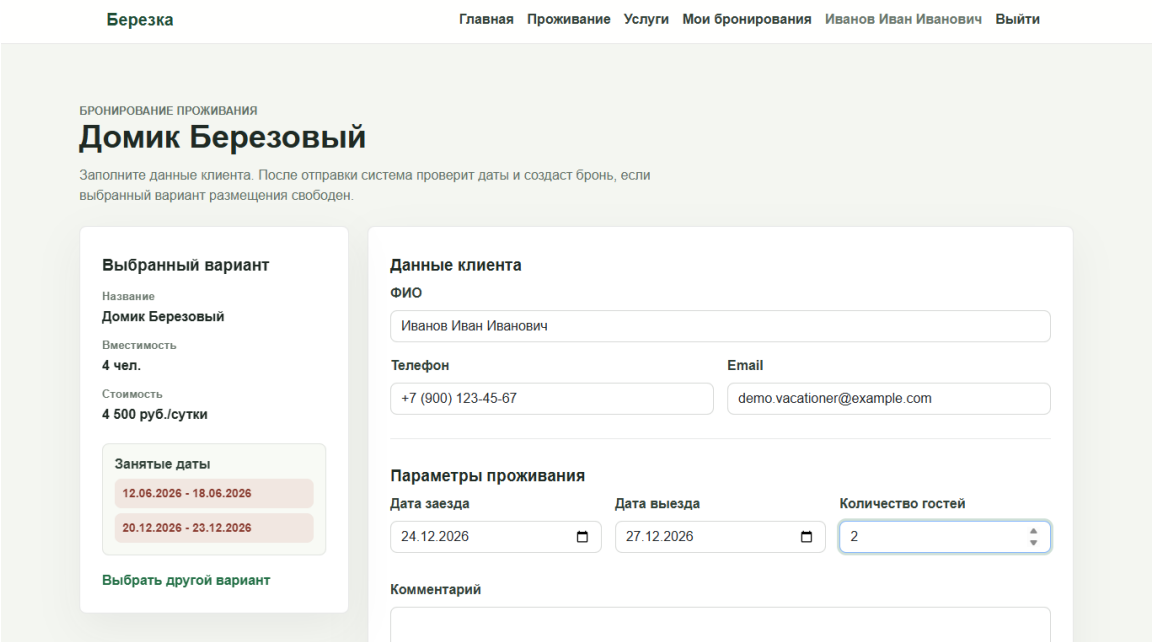


Рисунок Ж.4 – Форма бронирования проживания

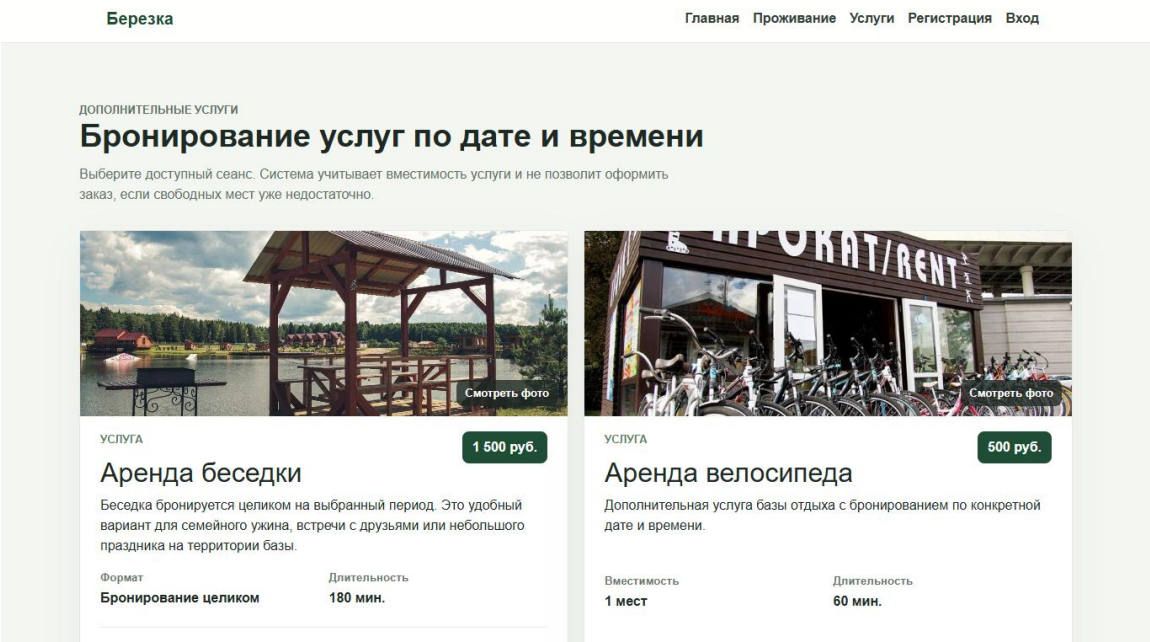


Рисунок Ж.5 – Каталог дополнительных услуг

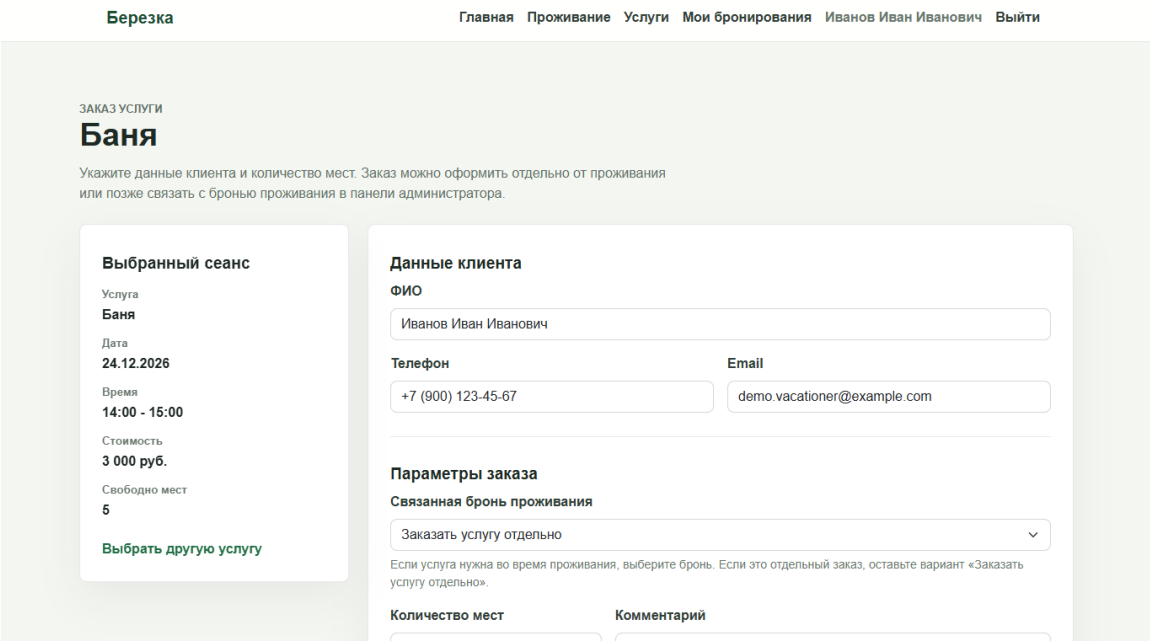


Рисунок Ж.6 – Форма заказа дополнительной услуги

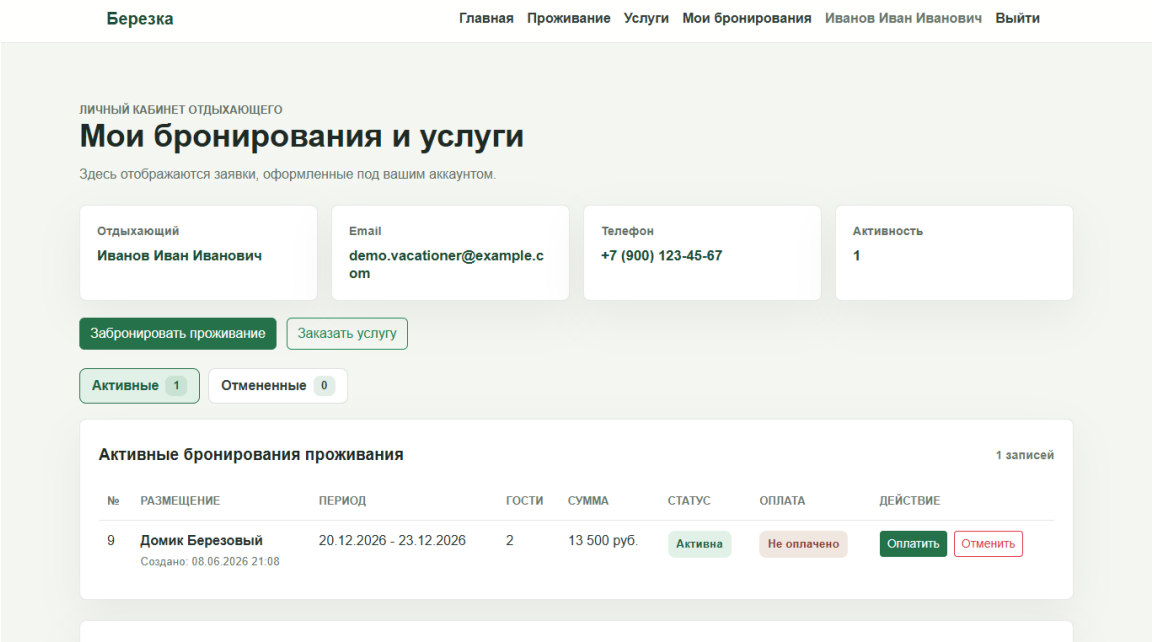


Рисунок Ж.7 – Личный кабинет отдыхающего

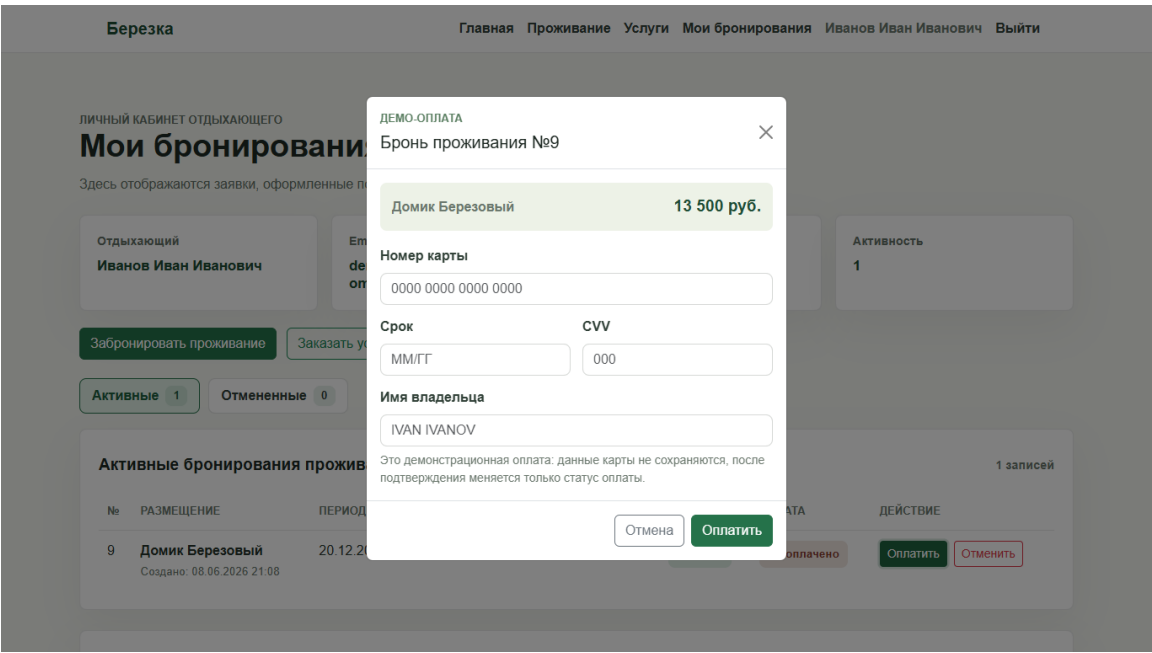


Рисунок Ж.8 – Окно демонстрационной оплаты заявки

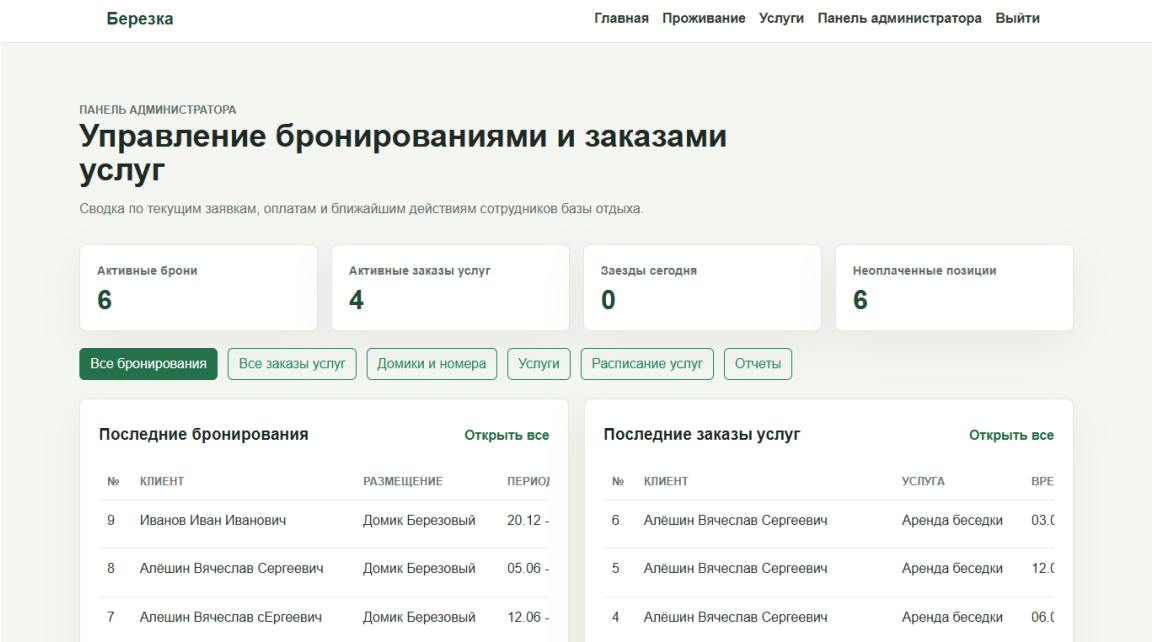


Рисунок Ж.9 – Панель администратора

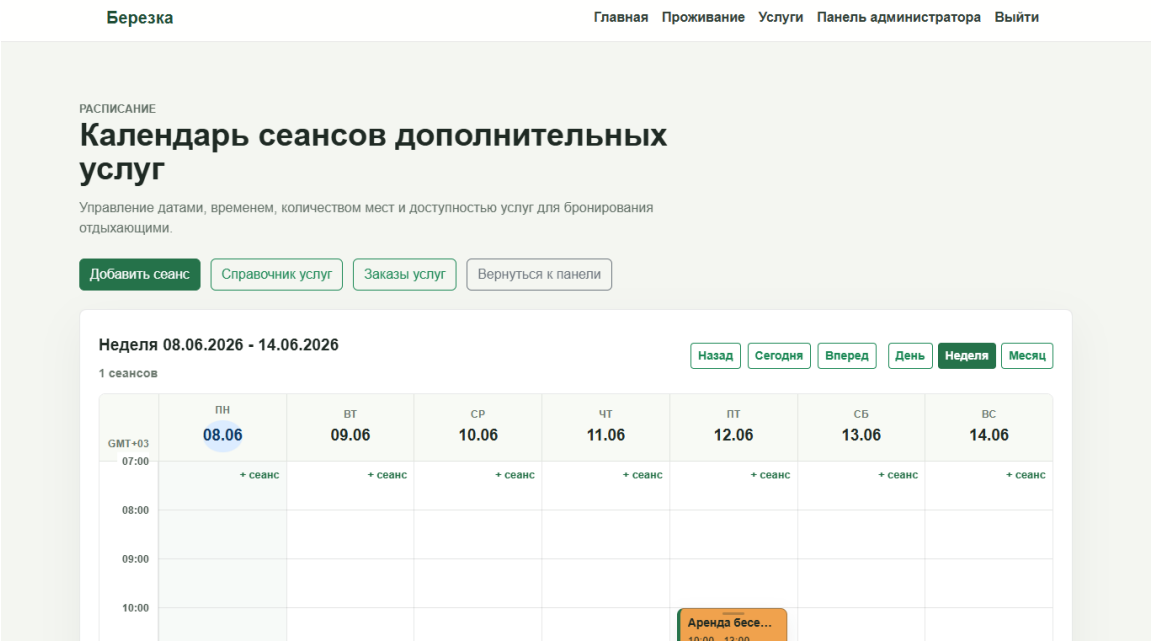
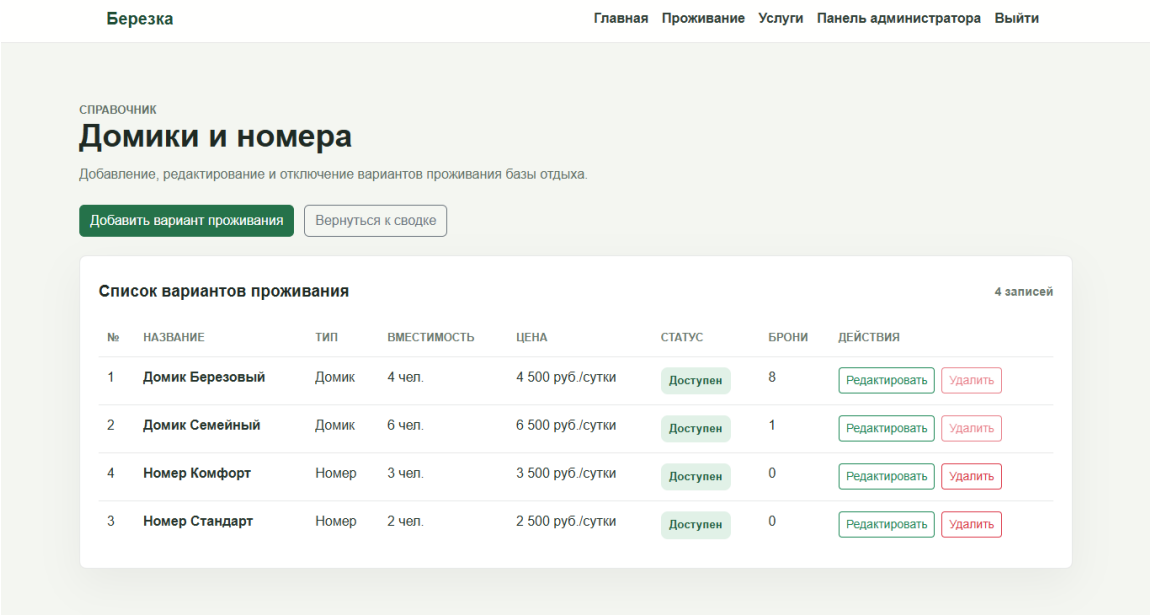


Рисунок Ж.10 – Календарь расписания услуг администратора



© 2026 Березка. Все права защищены.

Рисунок Ж.11 – Раздел управления объектами проживания

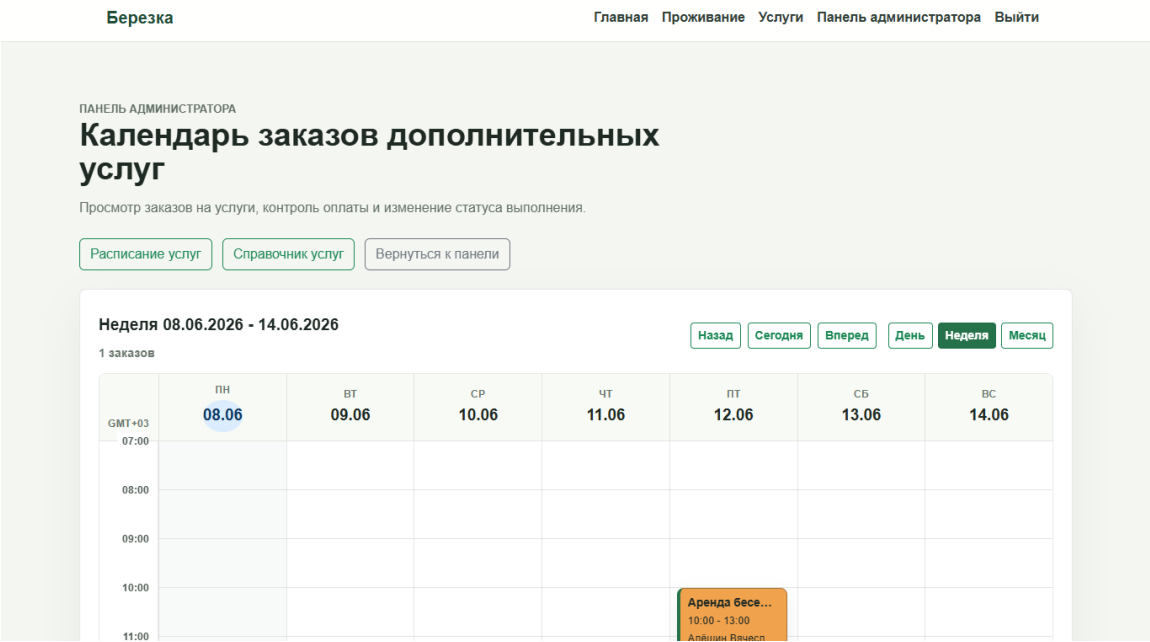


Рисунок Ж.12 – Раздел заказов дополнительных услуг

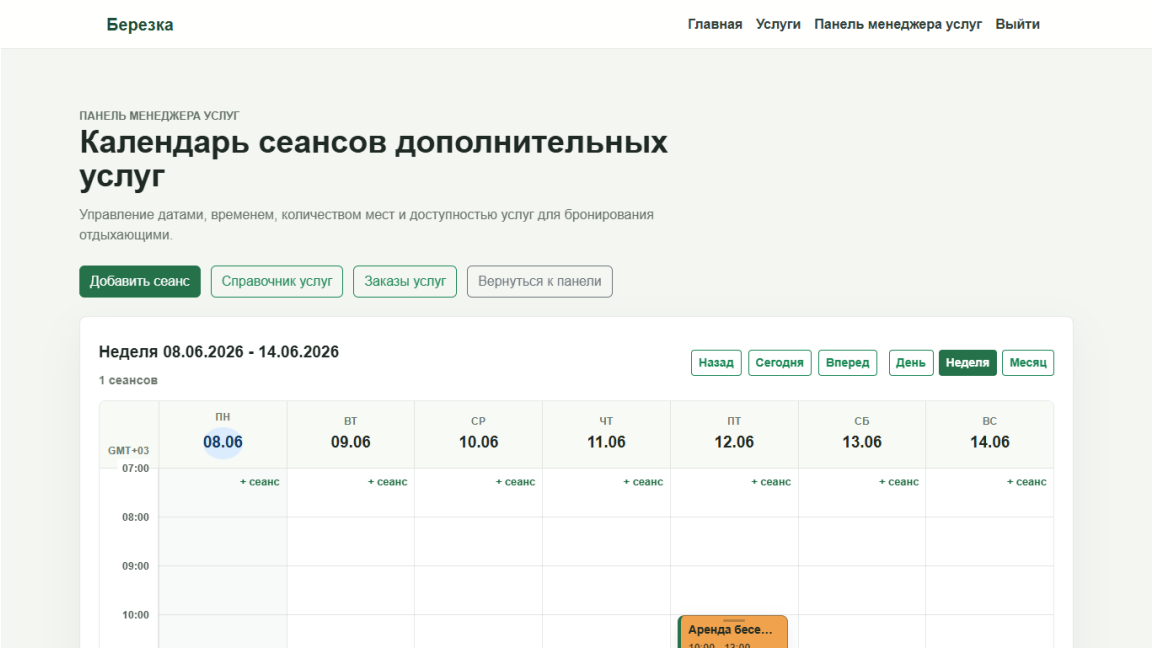


Рисунок Ж.13 – Календарь менеджера услуг

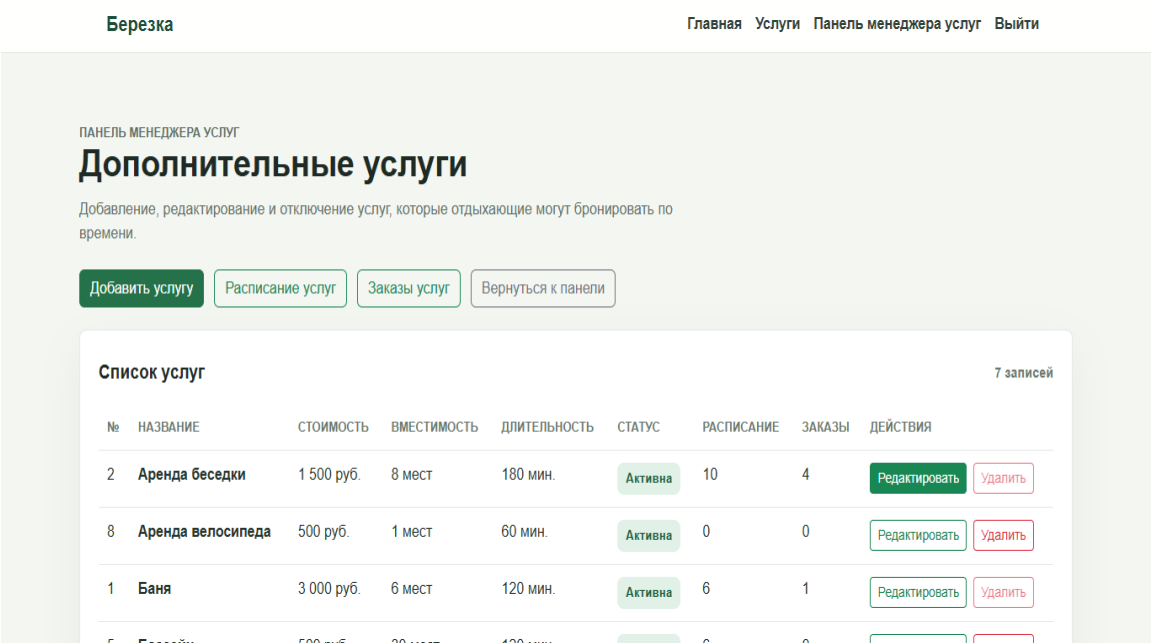


Рисунок Ж.14 – Раздел управления справочником услуг