

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 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Разработка бэкенд-компонента информационной
системы автоматизации бронирования и учета
услуг рыболовного хозяйства**

Студент _____ Гришин Максим Михайлович
(подпись, дата) (ФИО полностью)

Руководитель _____ Самуйлов С.В.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

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

Пенза, 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 Гришин Максим Михайлович гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Разработка бэкенд-компонента информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства

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

3. Руководитель работы к.т.н., доцент кафедры "МО и ПЭВМ" Самуйлов С.В.

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

1) Назначение: разработка серверной части информационной системы для автоматизации обработки заявок на бронирование, учета услуг и управления данными рыболовного хозяйства

2) Среда выполнения: Windows 10.

Языковые средства разработки: C#, SQL, UML.

Инструментальные средства: Visual Studio 2022, PostgreSQL, pgAdmin 4, Swagger, Visual Paradigm 17.3.

Технологии разработки: ООП, многослойная архитектура, REST API, Entity Framework Core, JWT-аутентификация.

3) Основные функции приложения:

а) авторизация администратора с разграничением доступа;

б) просмотр и фильтрация рыболовных секторов;

в) просмотр информации о секторах, тарифах и товарах;

г) расчет стоимости отдыха и оформление бронирования сектора;

д) просмотр и отмена бронирования;
е) административное управление бронированиями, секторами, тарифами и товарами;
ж) формирование уведомлений и аналитических отчетов.

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 «26» 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 120 листов, 29 рисунков, 33 таблицы, 21 использованный источник и 6 приложений.

АВТОМАТИЗАЦИЯ БРОНИРОВАНИЯ, РЫБОЛОВНОЕ ХОЗЯЙСТВО, BACKEND-КОМПОНЕНТ, WEB-ПРИЛОЖЕНИЕ, REST API, UML, SQL, C#, ASP.NET CORE, POSTGRESQL, БАЗА ДАННЫХ.

Объект разработки – бэкенд-компонент информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства.

Цель разработки: спроектировать и реализовать серверную часть информационной системы, предназначенную для автоматизации процессов бронирования рыболовных секторов, расчёта стоимости услуг, управления данными клиентов, товаров, тарифов и бронирований.

Результаты разработки: в рамках выпускной квалификационной работы была спроектирована и реализована серверная часть информационной системы. Разработанное программное обеспечение обеспечивает просмотр секторов, расчёт стоимости отдыха, оформление бронирований, управление заявками, тарифами, товарами и формирование отчётных данных. Все предъявленные требования к созданию бэкенд-компонента выполнены.

Средства разработки: серверная часть приложения была реализована в среде разработки Microsoft Visual Studio 2022 на языке программирования C# с использованием фреймворка ASP.NET Core. Для взаимодействия с базой данных применялась технология Entity Framework Core. В качестве системы управления базами данных была выбрана PostgreSQL. Для проектирования моделей и диаграмм использовалась среда Visual Paradigm, для проверки REST API – Swagger, для планирования разработки и оценки бюджета – ProjectLibre.

					ПГУ 09.03.04-8BP222.05 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Разработка бэкенд-компонента информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства	Лит.	Лист	Листов
Разраб.		Гришин М.М.					4	120
Провер.		Самуйлов С.В.				Группа 22ВП2		
Реценз.								
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение	7
1 Постановка задачи и анализ требований	9
1.1 Постановка задачи на разработку приложения	9
1.2 Анализ аналогов	10
1.2.1 Анализ системы «Carp Fishing Belarus»	10
1.2.2 Анализ системы «World-Fish»	11
1.2.3 Анализ системы «Вороний пруд»	13
1.2.4 Сравнение аналогов	14
1.3 Анализ функциональных требований	15
1.4 Анализ нефункциональных требований	22
2 Проектирование приложения	24
2.1 Архитектура приложения	24
2.2 Проектирование структур данных	26
2.2.1 Концептуальная модель базы данных	26
2.2.2 Логическая модель базы данных	28
2.3 Проектирование программных средств	29
2.4 Проектирование пользовательского интерфейса	34
2.5 Выбор технологий и инструментальных средств разработки приложения	36
2.5.1 Выбор технологий разработки приложения	36
2.5.2 Выбор инструментальных средств разработки приложения	38
3 Разработка программного обеспечения	40
3.1 Разработка физической модели базы данных	40
3.2 Реализация серверной части информационной системы	41
3.3 Реализация клиентской части информационной системы	44
4 Отладка и тестирование программного обеспечения	45
4.1 Функциональное тестирование методом белого ящика	45
4.1.1 Покрытие операторов	45
4.1.2 Покрытие решений	50

4.1.3	Покрытие условий.....	53
4.1.4	Покрытие решений и условий	56
4.2	Модульные тесты серверной части программного обеспечения	60
5	Оценка качества программного кода	62
5.1	Тестирование SLOC	62
5.2	Метрика Холстеда	63
5.3	Метрика Чепина	66
5.4	Автоматическая проверка	68
6	Планирование разработки и оценка бюджета	70
	Заключение	74
	Список использованных источников	76
	Приложение А (справочное) Глоссарий	78
	Приложение Б (рекомендуемое) Атрибуты сущностей базы данных.....	81
	Приложение В (обязательное) Листинг серверной части системы.....	88
	Приложение Г (рекомендуемое) Листинг модульных тестов	100
	Приложение Д (рекомендуемое) Графическая часть	111
	Приложение Е (рекомендуемое) Акт о внедрении.....	119

Введение

В современном мире внутренний туризм и загородный отдых становятся всё более популярными среди жителей крупных городов. Вместо длительных поездок за границу люди все чаще выбирают короткие поездки на выходные, отдых на природе и рыбалку. Однако владельцы частных рыболовных хозяйств зачастую сталкиваются с проблемой неэффективной организации процессов бронирования и учета. В большинстве случаев используются устаревшие методы, такие как запись по телефону и ведение бумажных журналов. Это приводит к ряду проблем: двойное бронирование одного сектора на одну дату, потеря заявок клиентов, невозможность оперативно получить информацию о загруженности пруда и актуальных остатках товаров. Со стороны клиентов также возникают трудности: чтобы забронировать сектор, необходимо позвонить владельцу или администратору водоёма в рабочее время, узнать о свободных секторах, согласовать стоимость, а затем надеяться, что запись не потерялась. Отсутствие возможности самостоятельно рассчитать стоимость отдыха с учетом всех услуг, увидеть фотографии секторов и ознакомиться с их характеристиками снижает привлекательность подобных объектов для современных пользователей, привыкших к цифровым сервисам.

Таким образом, возникает противоречие между растущим спросом на организованный загородный отдых и низким уровнем автоматизации малого бизнеса в этой сфере. Требуется создание доступного и функционального программного обеспечения, которое позволит автоматизировать ключевые бизнес-процессы рыболовного водоёма, сделав его конкурентоспособным и удобным для клиентов.

Целью выпускной квалификационной работы является разработка бэкенд-компонента информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства. Разработка включает в себя несколько этапов:

- постановка задачи и анализ требований;

- проектирование архитектуры приложения, программных средств и структуры базы данных;
- программная реализация приложения;
- тестирование и отладка приложения;
- оценка качества программного кода с использованием метрик;
- планирование разработки и оценка бюджета проекта.

Каждый из перечисленных этапов является важной частью разработки программного продукта. Успешное выполнение всех этапов позволит создать надёжный, масштабируемый и производительный бэкенд-компонент, который станет основой для полноценной информационной системы рыболовного хозяйства и обеспечит удобный и прозрачный процесс взаимодействия между клиентами и владельцем пруда.

1 Постановка задачи и анализ требований

1.1 Постановка задачи на разработку приложения

Владельцы частных рыболовных водоёмов для отдыха сталкиваются с необходимостью организации эффективного процесса бронирования секторов, учета дополнительных услуг и взаимодействия с клиентами. В настоящий момент времени большинство операций выполняется вручную: заявки принимаются по телефону или через мессенджеры, информация о бронированиях фиксируется в таблицах или бумажных блокнотах, а расчет стоимости пребывания на водоёме осуществляется администратором самостоятельно. Подобный подход приводит к ряду проблем, начиная от отсутствия инструментов анализа деятельности хозяйства, заканчивая высокой вероятностью ошибки при обработке заявок и возможного двойного бронирования одного сектора. Кроме того, клиент не имеет возможности самостоятельно ознакомиться с доступными секторами, посмотреть их описание и фотографии, заранее рассчитать стоимость отдыха и быстро оформить бронирование.

Для решения указанных проблем требуется разработка информационной системы, обеспечивающей автоматизацию процессов бронирования и управления рыболовным хозяйством.

Разрабатываемое приложение должно обеспечивать возможность:

- просмотра и фильтрации секторов по характеристикам;
- отображения подробной информации о секторах;
- ознакомления с тарифами;
- расчета стоимости отдыха с учетом количества посетителей и дополнительных услуг;
- оформления бронирования;
- просмотра и отмены бронирования по коду и номеру телефона;
- управления бронирования со стороны администратора;

- управления секторами, тарифами и товарами;
- формирования аналитических отчетов.

В системе должны быть предусмотрены механизмы проверки корректности вводимых данных, контроля доступности секторов на выбранные даты, ограничения бронирования по сезону, а также защиты административного функционала с использованием авторизации. В результате разработки должна быть создана серверная часть информационной системы, обеспечивающая централизованное хранение данных, автоматизацию ключевых бизнес-процессов и возможность дальнейшего расширения функциональности приложения.

1.2 Анализ аналогов

В ходе изучения рынка были рассмотрены системы аналогичного назначения, предоставляющие услуги платной рыбалки. В качестве аналогов были выбраны следующие системы: сайты «Carp Fishing Belarus», «World-Fish» и сообщество «Вороний пруд» в социальной сети «ВКонтакте».

1.2.1 Анализ системы «Carp Fishing Belarus»

Сайт Carp Fishing Belarus представляет собой информационный web-ресурс рыболовного хозяйства, предназначенный для ознакомления клиентов с предоставляемыми услугами [1]. Пользователю доступны сведения о секторах, правилах пребывания, инфраструктуре, стоимости услуг, а также фотографии территории и мест для рыбалки.

К преимуществам данной системы можно отнести:

- наличие фотографий территории и инфраструктуры;
- предоставление информации о тарифах и правилах пребывания.

Несмотря на подробное предоставление информации, система обладает рядом недостатков, как для клиентов, так и для администратора рыболовного хозяйства:

- отсутствуют разделы по каждой категории информации;

- отсутствует онлайн-бронирование рыболовных секторов;
- отсутствует «калькулятор» для расчета стоимости отдыха;
- отсутствует возможность просмотра свободных секторов по датам;
- отсутствует возможность самостоятельного управления бронированием со стороны клиента;
- отсутствует административная система управления бронированиями и товарами;
- отсутствует аналитика и формирование отчетности.

Интерфейс сайта представлен на рисунке 1.

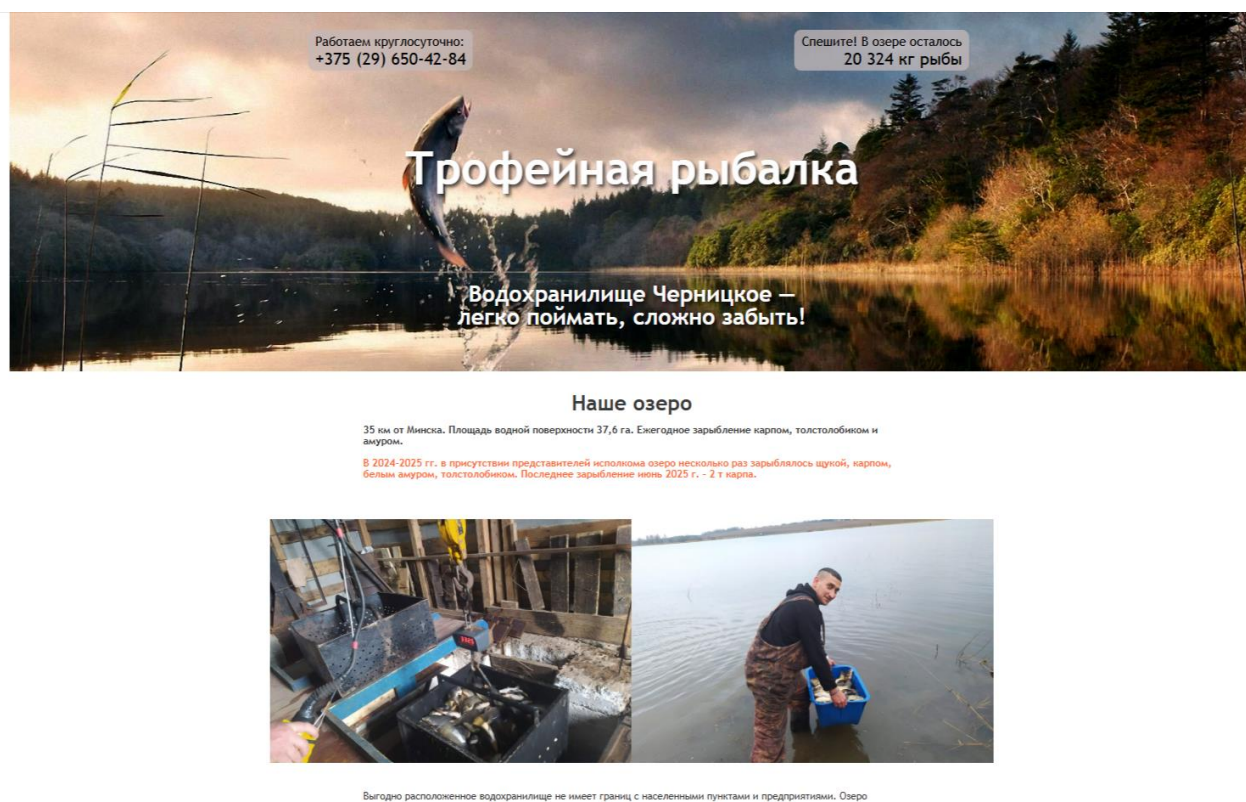


Рисунок 1 – Интерфейс сайта «Carp Fishing Belarus»

Рассмотренный сайт позволяет пользователю получить общее представление о рыболовном хозяйстве, но без дополнительного функционала.

1.2.2 Анализ системы «World-Fish»

Система World-Fish представляет собой сайт, который содержит как информацию о водоёме и условиях отдыха на его территории, так и

возможность рассчитать стоимость предоставляемых услуг, указав основные параметры отдыха, и оставить заявку на бронирование рыболовного сектора, для дальнейшей связи с администратором водоёма [2]. Интерфейс данной системы представлен на рисунке 2.

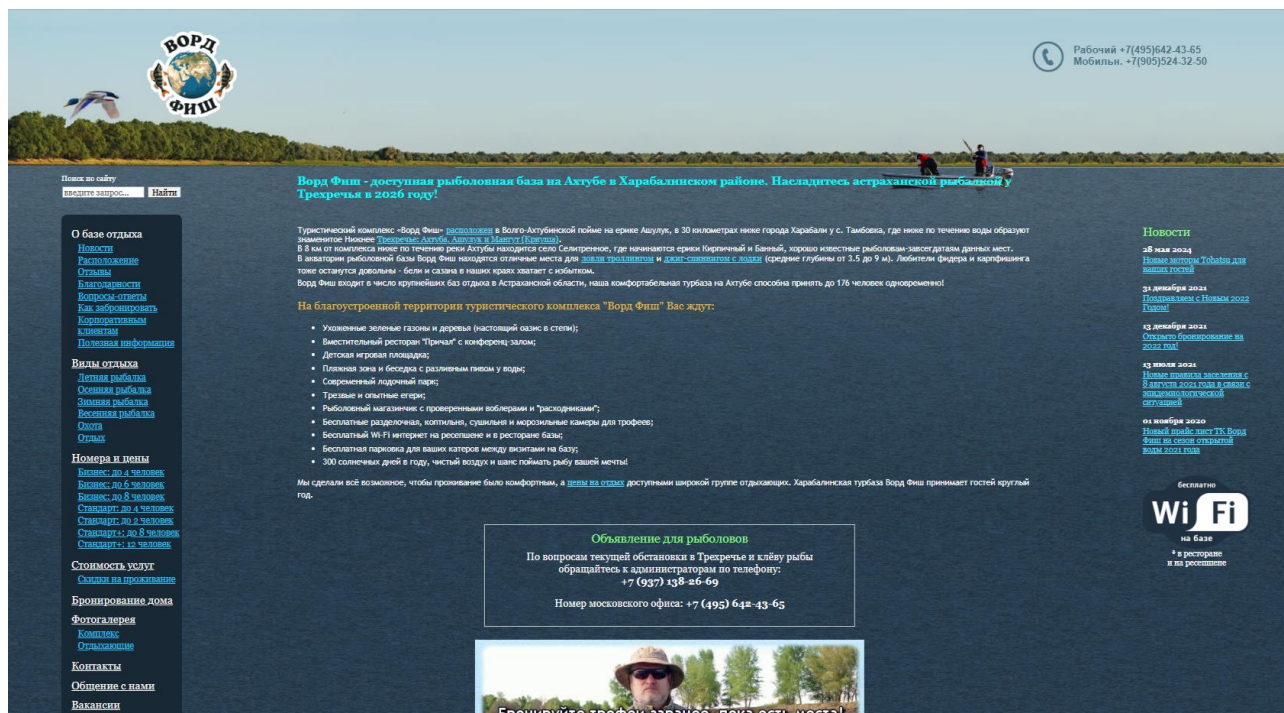


Рисунок 2 – Интерфейс сайта «World-Fish»

К преимуществам данной системы можно отнести:

- наличие разделов по каждой категории информации;
- возможность ознакомиться с секторами и тарифами;
- наличие «калькулятора» расчёта стоимости отдыха;
- возможность отправить онлайн-заявку на бронь.

Минусами данной системы являются:

- отсутствие поиска секторов, с применением фильтрации;
- отсутствие полноценного функционала онлайн-бронирования;
- отсутствие системы управления доступностью секторов;
- отсутствие административной системы управления товарами и дополнительными услугами;
- отсутствие централизованного хранения информации о бронированиях;

– отсутствие инструментов аналитики и формирования отчетности.

1.2.3 Анализ системы «Вороний пруд»

Сообщество «Вороний пруд» в социальной сети «ВКонтакте» используется в качестве платформы для публикации информации о рыболовном хозяйстве и взаимодействия с клиентами [3]. В сообществе размещаются фотографии секторов и уловов, объявления, правила пребывания на территории водоёма и контактные данные администрации. Его интерфейс представлен на рисунке 3.

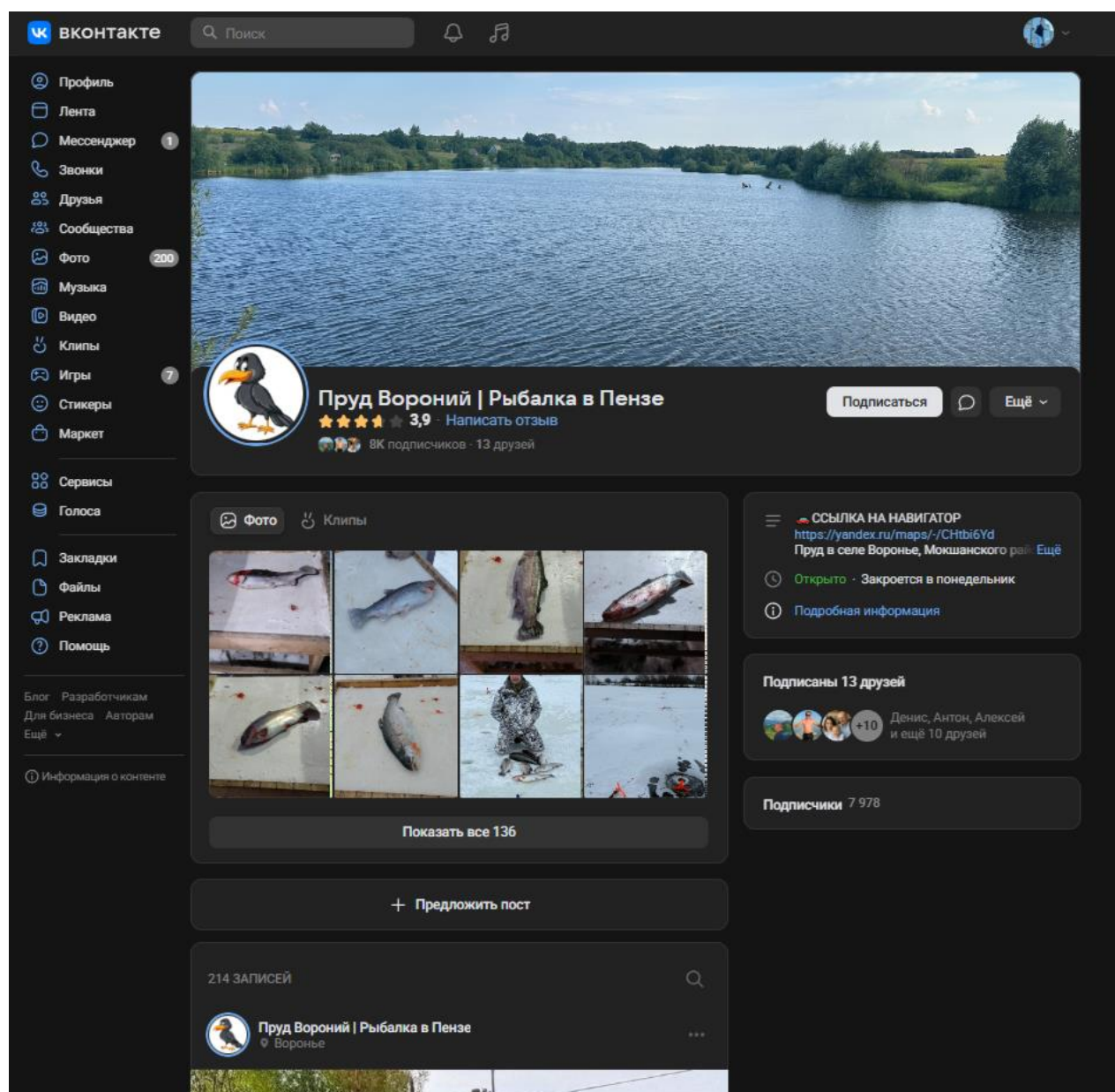


Рисунок 3 – Интерфейс интернет-сообщества «Вороний пруд»

К преимуществам данного подхода можно отнести:

- простоту публикации информации;
- возможность быстрого взаимодействия с клиентами;
- наличие фотографий и пользовательских отзывов.

Однако использование социальной сети в качестве основного инструмента управления имеет существенный ряд недостатков:

- отсутствует большая часть разделов информации;
- отсутствует централизованное хранение данных;
- бронирование осуществляется через сообщения;
- отсутствует автоматическая проверка занятости секторов;
- отсутствует система управления товарами и услугами;
- отсутствует возможность просмотра и отмены бронирования;
- отсутствует административная аналитика и отчетность;
- высокая вероятность потери или дублирования заявок.

1.2.4 Сравнение аналогов

Для проведения сравнительного анализа рассмотренных аналогов были выбраны основные критерии, характеризующие уровень автоматизации процессов взаимодействия с клиентами и управления рыболовным хозяйством. Была использована трехбалльная система оценки:

- 0 баллов – критерий не реализован;
- 1 балл – критерий реализован частично;
- 2 балла – критерий полностью реализован.

Результаты сравнительного анализа представлены в таблице 1.

Таблица 1 – Результаты сравнительного анализа

Критерий	Carp Fishing Belarus	World-Fish	Вороний пруд VK
Наличие информационных разделов	0	2	1
Просмотр информации о секторах	2	2	0

Продолжение таблицы 1

Критерий	Carp Fishing Belarus	World-Fish	Вороний пруд VK
Просмотр фотографий секторов	2	2	1
Онлайн-расчет стоимости	0	2	0
Онлайн-бронирование секторов	0	1	0
Проверка доступности секторов	0	0	0
Фильтрация и поиск секторов	0	1	0
Просмотр и отмена бронирования	0	0	0
Управление товарами и услугами	0	1	0
Управление тарифами	2	2	2
Управление бронированиями	0	1	0
Формирование отчетности	0	0	0
Итого	6	14	5

Данное сравнение показало, что рассмотренные аналоги обладают ограниченным уровнем автоматизации. Большинство существующих решений ориентированы преимущественно на предоставление информации клиентам и не обеспечивают полноценную автоматизацию процессов управления рыболовным хозяйством. При проектировании и реализации разрабатываемой информационной системы будут учтены все достоинства и недостатки проанализированных систем, что поможет разработать удобное приложение с полезным функционалом.

1.3 Анализ функциональных требований

На основе проведенного анализа предметной области и рассмотренных программ-аналогов были определены функциональные требования к разрабатываемой информационной системе автоматизации рыболовного хозяйства.

Функциональные требования играют важную роль при разработке программного продукта, так как они определяют, какие действия должна

выполнять система и какие возможности должны быть доступны пользователям. В рамках данной работы функциональные требования описывают поведение и возможности системы, необходимые для автоматизации основных бизнес-процессов, связанных с бронированием рыболовных секторов, управлением услугами и взаимодействием пользователей с приложением.

Разрабатываемая система предусматривает две основные категории пользователей: клиент и администратор. Для каждой категории пользователей предусмотрен отдельный набор функциональных возможностей. На основе данных функциональных возможностей для каждой категории пользователей была разработана диаграмма вариантов использования UML [4].

Диаграмма вариантов использования – это диаграмма, описывающая какой функционал разрабатываемой программной системы доступен каждой группе пользователей [5]. Диаграмма вариантов использования служит для описания функциональности и поведения проектируемой системы. На ней видно, кто взаимодействует с системой, и какие действия может выполнить.

Актером является тот, кто использует систему. В качестве актера может выступать человек, внешняя сущность, класс или другая система. Актеры не связаны между собой напрямую.

Вариант использования – это выполняемые системой действия, приводящие к наблюдаемым актерами результатам. Каждый вариант использования описывает конкретный сценарий – последовательность шагов, которые система выполняет, чтобы получить нужный результат.

Клиентская часть системы обеспечивает возможности: ознакомиться с секторами (по желанию, применяя фильтрацию по нужным ему критериям), ознакомиться с тарифами рыбалки, рассчитать стоимость рыбалки, забронировать сектор, просмотреть свои бронирования и отменить бронирование.

Для визуального взаимодействия клиента с системой была разработана диаграмма вариантов использования, представленная на рисунке 4.

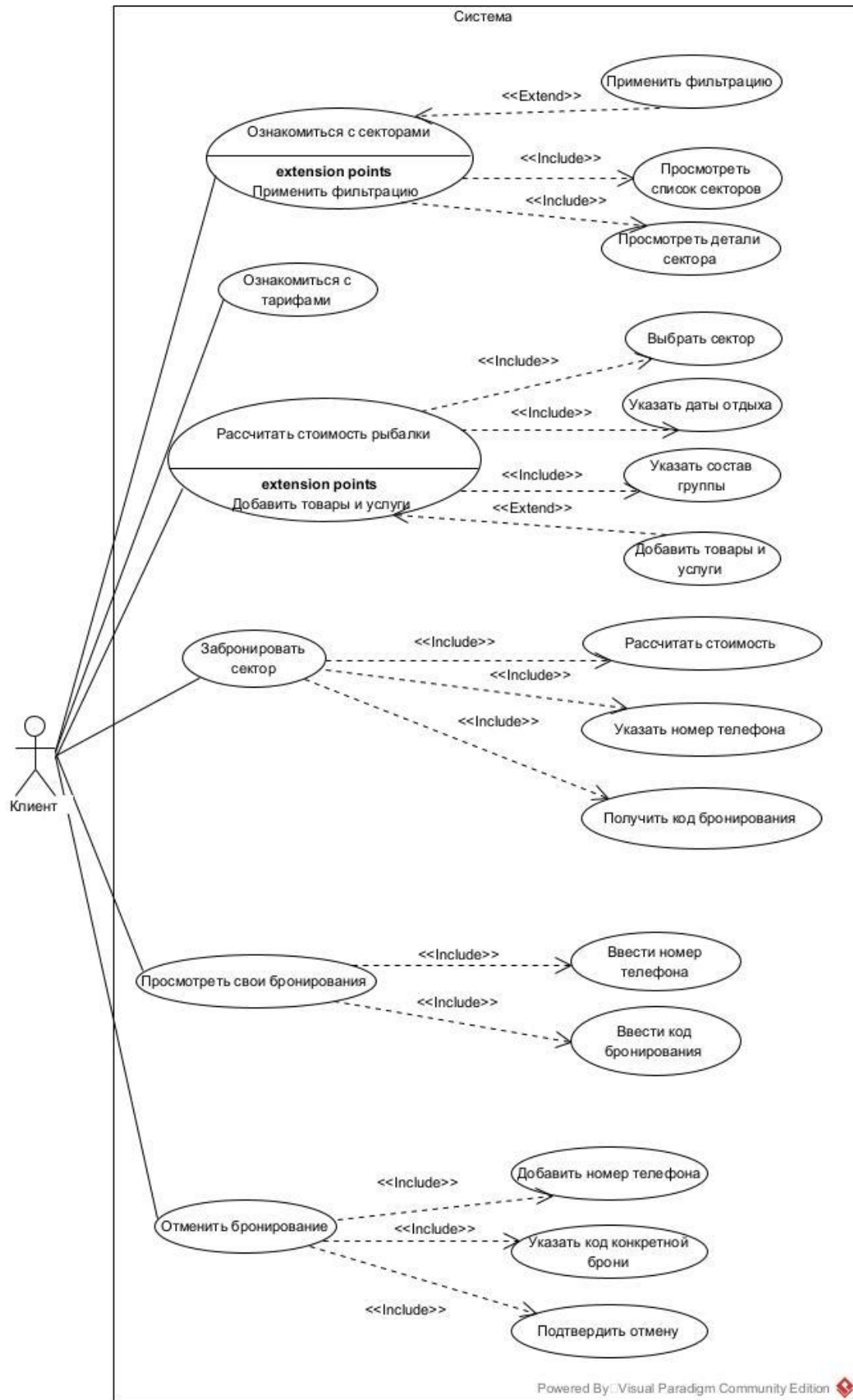


Рисунок 4 – Диаграмма вариантов использования клиентской части системы

Чтобы детализировать и описать основные действия работы наиболее важных вариантов использования у клиентской стороны были разработаны их спецификации.

Спецификация варианта использования «Ознакомиться с секторами» показана на рисунке 5.

Прецедент	Ознакомиться с секторами
ID	1
Краткое описание	Клиент просматривает список рыболовных секторов и получает подробную информацию о выбранном секторе
Главные актёры	Клиент
Второстепенные актёры	Нет
Предусловие	1. Система доступна пользователю 2. В системе существуют доступные сектора
Основной поток	1. Клиент открывает список секторов 2. Система отображает список доступных секторов 3. Клиент может применить фильтрацию по характеристикам 4. Клиент выбирает интересующий сектор 5. Система отображает подробную информацию о секторе 6. Клиент просматривает фотографии и характеристики сектора
Альтернативные потоки	1а. Сектора по заданным фильтрам отсутствуют. 1б. Система выводит сообщение об отсутствии подходящих секторов
Постусловие	Пользователь ознакомлен с информацией о выбранном секторе

Рисунок 5 – Спецификация варианта использования «Ознакомиться с секторами»

Спецификация варианта использования «Рассчитать стоимость рыбалки» показана на рисунке 6.

Прецедент	Рассчитать стоимость рыбалки
ID	3
Краткое описание	Клиент выполняет расчет стоимости отдыха с учетом выбранного сектора, дат отдыха, состава группы и дополнительных услуг
Главные актёры	Клиент
Второстепенные актёры	Нет
Предусловие	1. Система доступна пользователю 2. В системе существуют доступные тарифы и сектора
Основной поток	1. Клиент выбирает сектор 2. Клиент указывает даты въезда на водоём и выезда 3. Клиент указывает состав группы 3.1 Клиент указывает количество рыбаков-мужчин 3.2 Клиент указывает количество рыбаков-женщин 3.3 Клиент указывает количество отдыхающих мужчин 3.4 Клиент указывает количество отдыхающих женщин 3.5 Клиент указывает количество отдыхающих детей 4. Клиент может добавить товары, которые хочет приобрести 5. Клиент может добавить товары, которые хочет арендовать на время отдыха 6. Клиент указывает, будет ли осуществлена бронь или нет 7. Система выполняет расчет стоимости отдыха 8. Система отображает итоговую стоимость отдыха
Альтернативные потоки	1а. Пользователь указывает некорректные даты отдыха 1б. Система выводит сообщение об ошибке 2а. Выбранный сектор недоступен 2б. Система выводит сообщение о невозможности расчета стоимости
Постусловие	Пользователь получает информацию о стоимости отдыха

Рисунок 6 – Спецификация варианта использования «Рассчитать стоимость рыбалки»

Спецификация варианта использования «Забронировать сектор» показана на рисунке 7.

Прецедент	Забронировать сектор
ID	4
Краткое описание	Клиент оформляет бронирование выбранного сектора на указанный период времени
Главные актёры	Клиент
Второстепенные актёры	Нет
Предусловие	1. Система доступна пользователю 2. Сектор доступен для бронирования
Основной поток	1. Клиент выбирает сектор 2. Клиент указывает даты въезда на водоём и выезда 3. Клиент указывает состав группы 3.2 Клиент указывает количество рыбаков-мужчин 3.3 Клиент указывает количество рыбаков-женщин 3.4 Клиент указывает количество отдыхающих мужчин 3.5 Клиент указывает количество отдыхающих женщин 3.6 Клиент указывает количество отдыхающих детей 4. Клиент может добавить товары, которые хочет приобрести 5. Клиент может добавить товары, которые хочет арендовать на время отдыха 6. Клиент вводит номер телефона 7. Система выполняет расчет стоимости отдыха 8. Клиент подтверждает бронирование 9. Система создает бронирование 10. Система присваивает код бронирования 11. Система отображает информацию о созданном бронировании
Альтернативные потоки	1а. Выбранный сектор уже занят на указанные даты 1б. Система выводит сообщение о недоступности сектора 2а. Пользователь вводит некорректные данные 2б. Система выводит сообщение об ошибке ввода данных
Постусловие	Бронирование успешно создано и сохранено в системе

Рисунок 7 – Спецификация варианта использования «Забронировать сектор»

Представленные спецификации вариантов использования клиентской части позволяют детализировать основные сценарии взаимодействия пользователя с системой.

Административная часть системы обеспечивает возможности: авторизоваться, управлять бронированиями, управлять секторами, управлять тарифами, управлять товарами, формировать отчеты.

Для визуального взаимодействия администратора с системой была разработана диаграмма вариантов использования, представленная на рисунке 8.

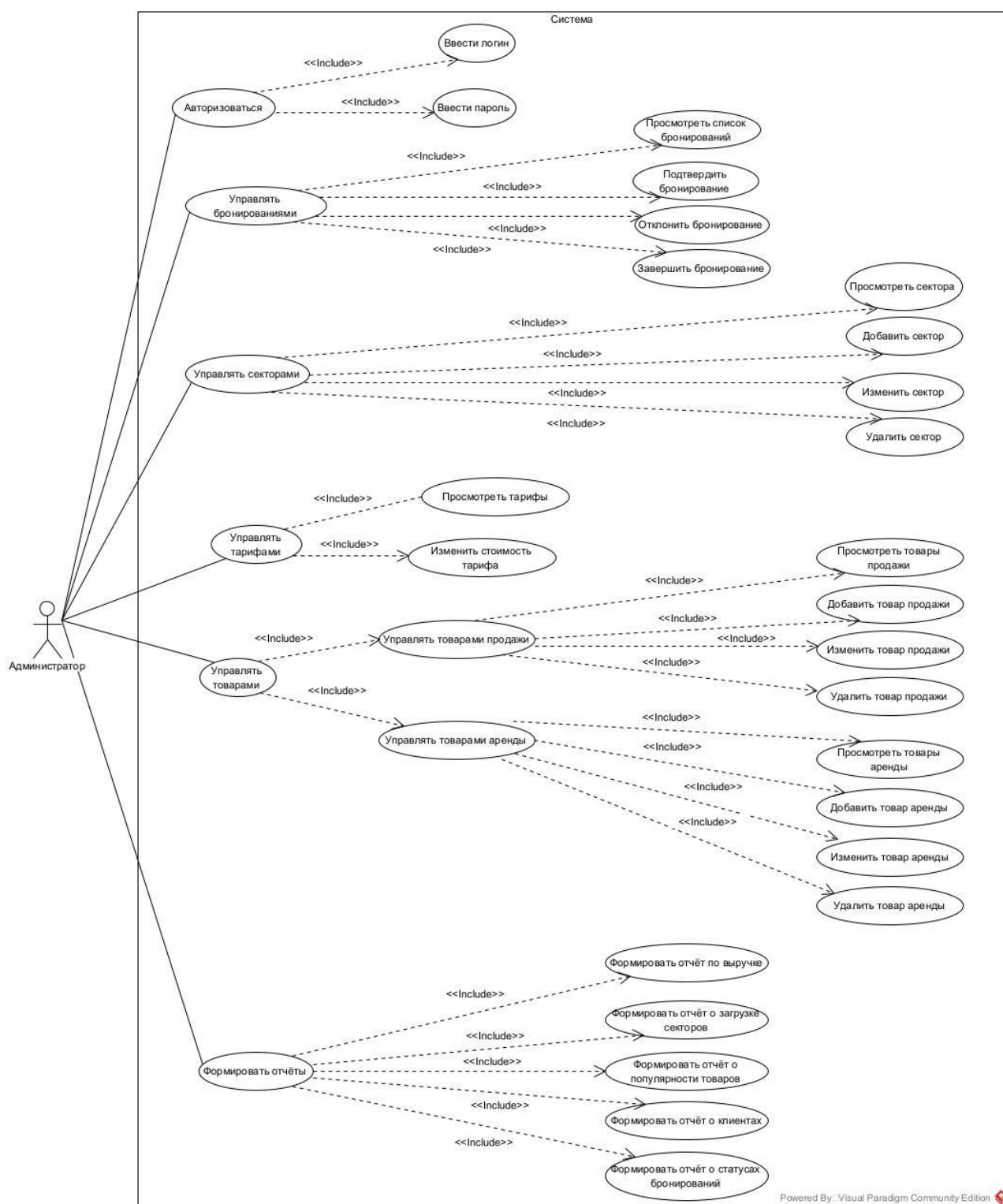


Рисунок 8 – Диаграмма вариантов использования административной части системы

Чтобы детализировать и описать основные действия работы наиболее важных вариантов использования у административной стороны были разработаны их спецификации.

Спецификация варианта использования «Управлять бронированиями» показана на рисунке 9.

Прецедент	Управлять бронированиями
ID	2
Краткое описание	Администратор просматривает и обрабатывает бронирования клиентов
Главные актёры	Администратор
Второстепенные актёры	Нет
Предусловие	1. Администратор авторизован в системе 2. В системе существуют бронирования секторов
Основной поток	1. Администратор открывает список бронирований секторов 2. Система отображает список бронирований 3. Администратор выбирает конкретное бронирование 4. Система отображает подробную информацию о бронировании 5. Администратор подтверждает, отклоняет или завершает бронирование 6. Система изменяет статус бронирования
Альтернативные потоки	1а. Администратор не авторизован 1б. Система запрещает доступ к функционалу 2а. Бронирования отсутствуют 2б. Система выводит соответствующее сообщение
Постусловие	Статус бронирования обновлён в системе

Рисунок 9 – Спецификация варианта использования «Управлять бронированиями»

Спецификация варианта использования «Формировать отчеты» показана на рисунке 10.

Прецедент	Формировать отчёты
ID	6
Краткое описание	Администратор формирует аналитические отчёты по деятельности рыболовного хозяйства
Главные актёры	Администратор
Второстепенные актёры	Нет
Предусловие	1. Администратор авторизован в системе 2. В системе существуют данные для формирования отчётов
Основной поток	1. Администратор открывает раздел отчётов 2. Администратор выбирает тип отчёта 3. Администратор указывает временной период 4. Система формирует отчёт 5. Система отображает результат отчёта
Альтернативные потоки	1а. Администратор не авторизован 1б. Система запрещает доступ к функционалу 2а. За выбранный период отсутствуют данные 2б. Система выводит соответствующее сообщение
Постусловие	Отчёт успешно сформирован и отображён администратору

Рисунок 10 – Спецификация варианта использования «Формировать отчеты»

Результатом анализа функциональных требований являются определение основных категорий пользователей системы и формирование перечня функциональных возможностей клиентской и административной частей приложения. Проведённый анализ позволил сформировать основу для дальнейшего проектирования архитектуры приложения, структуры базы данных и программной реализации системы.

1.4 Анализ нефункциональных требований

Кроме функциональных возможностей, разрабатываемая информационная система должна удовлетворять ряду нефункциональных требований, определяющих качество и характеристики работы системы, не связанные напрямую с конкретными функциями, обеспечивающие стабильную работу приложения и возможность его дальнейшего развития [6].

Для информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства можно выделить следующие нефункциональные требования: производительность, безопасность, удобство использования, модульность и расширяемость.

Производительность: при получении данных о секторах, стоимости отдыха и результатах бронирования система должна сохранять стабильность и корректно обрабатывать поступающие запросы, особенно при одновременной работе нескольких клиентов, а пользователь должен быстро получать информацию без длительного ожидания обработки запросов. Также приложение должно обеспечить эффективную работу с базой данных, так как в процессе эксплуатации количество броней, товаров и остальной информации будет постепенно увеличиваться. Для повышения производительности планируется использовать асинхронную обработку операций, оптимизированные запросы к базе данных и разделение системы на отдельные уровни архитектуры.

Безопасность: так как приложение предусматривает наличие административной части и хранение данных клиентов, необходимо ограничить

доступ к функциям управления и защитить систему от нежелательных пользователей. Административный функционал должен быть доступен только после успешной авторизации пользователя. Помимо этого, система должна выполнять проверку корректности вводимых данных и предотвращать создание некорректных бронирований. Для обеспечения безопасности планируется использовать JWT-аутентификацию, серверную валидацию данных и механизмы разграничения прав доступа.

Удобство использования: пользователь должен иметь возможность выполнять нужный функционал без выполнения большого количества действий. Все сообщения системы должны быть понятными и информативными, чтобы пользователь легко понимал результат выполняемых операций и причины возможных ошибок.

Модульность и расширяемость: В будущем может возникнуть необходимость добавления новых бизнес-процессов, отчетов и сущностей. По этой причине система должна обладать модульной структурой, позволяющей вносить изменения и расширять функциональность без существенной переработки существующего кода. Для этого планируется использование многослойной архитектуры, обеспечивающей разделение бизнес-логики, работы с данными и пользовательского взаимодействия.

2 Проектирование приложения

2.1 Архитектура приложения

Основной целью проектирования является создание гибкой, расширяемой и поддерживаемой информационной системы автоматизации рыболовного хозяйства. Особое внимание было уделено проектированию архитектуры приложения, так как именно она определяет общую структуру программного продукта и порядок взаимодействия его основных компонентов. От выбранной архитектуры напрямую зависят стабильность работы системы, удобство сопровождения программного кода, возможность дальнейшего расширения функциональности и масштабирования приложения.

Так как разрабатываемая система предполагает наличие большого количества бизнес-процессов, при проектировании было принято решение использовать монолитную многослойную архитектуру приложения. Такой подход позволяет разделить систему на независимые логические компоненты и снизить связанность между отдельными частями приложения. Благодаря разделению на слои каждый компонент отвечает за собственную область работы, что делает структуру приложения более понятной и удобной для дальнейшей доработки.

Архитектура приложения построена по принципу последовательного взаимодействия слоёв. Пользовательский запрос сначала поступает в слой представления, где происходит обработка HTTP-запроса и маршрутизация обращения к необходимому компоненту системы. Далее запрос передается в слой бизнес-логики, в котором выполняется проверка входных данных, обработка бизнес-правил и выполнение основных операций приложения. При необходимости получения или изменения данных система обращается к слою доступа к данным, который взаимодействует с базой данных. После завершения обработки результат возвращается пользователю в виде HTTP-ответа. Для более полного понимания принципа работы данной архитектуры на рисунке 11 представлена топология монолитной многослойной архитектуры.

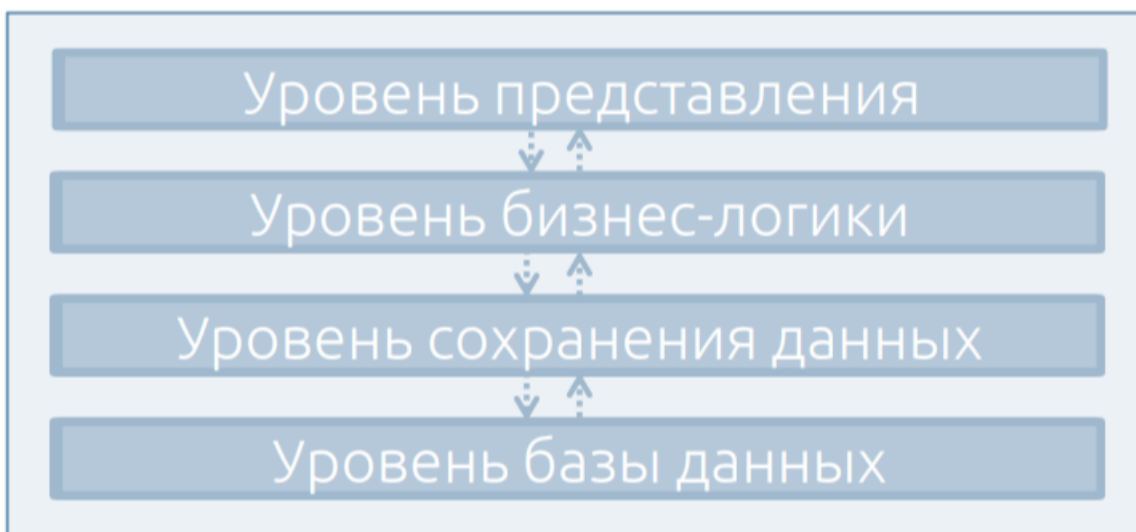


Рисунок 11 – Топология монолитной многослойной архитектуры

Структура разработанного приложения включает следующие основные слои:

- слой представления (API);
- слой бизнес-логики (Application);
- слой доступа к данным (Infrastructure);
- слой доменных сущностей (Domain).

Слой представления предназначен для взаимодействия системы с клиентскими приложениями. В данном слое реализованы REST API-контроллеры, обеспечивающие выполнение операций, связанных с просмотром секторов, расчетом стоимости отдыха, оформлением бронирований, управлением товарами и административным функционалом. Также данный слой отвечает за обработку HTTP-запросов и формирование ответов пользователю.

Слой бизнес-логики содержит основную функциональность приложения. В данном слое реализованы сервисы, выполняющие обработку бронирований, расчет стоимости отдыха, проверку доступности секторов, управление товарами, формирование отчетов и выполнение других бизнес-процессов. Кроме того, слой содержит DTO-модели, интерфейсы сервисов и механизмы валидации данных.

Слой доступа к данным обеспечивает взаимодействие приложения с базой данных. В данном слое реализованы репозитории, предназначенные для выполнения операций чтения, добавления, изменения и удаления данных.

Слой доменных сущностей содержит основные модели предметной области, описывающие структуру данных системы. В данном слое располагаются сущности бронирований, рыболовных секторов, тарифов, товаров продажи и аренды, а также перечисления и вспомогательные модели.

Диаграмма компонентов серверной части представлена в приложении Д, рисунок Д.1.

Выбранный архитектурный подход обеспечивает модульность, расширяемость и устойчивость приложения, а также создаёт основу для дальнейшего развития данной информационной системы.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Одним из ключевых этапов проектирования базы данных информационной системы является разработка концептуальной модели. Концептуальная модель – это упрощенное описание предметной области, которое показывает, какие объекты существуют, какими свойствами они обладают и как связаны между собой. На данном этапе формируется общее представление о структуре данных системы без привязки к конкретной системе управления базами данных и особенностями физической реализации.

Для составления концептуальной модели данных требуется выделить список всех основных сущностей предметной области разрабатываемой системы. В результате анализа бизнес-процессов для концептуальной модели были выделены сущности, описание которых представлено в таблице 2.

Таблица 2 – Описание сущностей

Название сущности	Идентификатор сущности	Описание сущности
Клиент	Client	Содержит сведения о клиенте рыболовного хозяйства: фамилию, имя, отчество и контакты

Продолжение таблицы 2

Название сущности	Идентификатор сущности	Описание сущности
Администратор	Admin	Содержит учетные данные администратора системы, необходимые для авторизации и выполнения административных действий
Бронирование	Booking	Содержит информацию о бронировании сектора клиентом, включая даты отдыха, данные клиента, статус бронирования, стоимость услуг и выбранный сектор
Тариф	Tariff	Содержит информацию о тарифе, его названии, описании и стоимости. Используется при расчете итоговой стоимости бронирования
Сектор	Sector	Предназначена для хранения информации о рыболовных секторах, их характеристиках, вместимости, глубине, наличии беседки, фотографиях и доступности для бронирования
Сторона водоёма	WaterSide	Содержит сведения о стороне водоёма или части водоёма, на которой расположен рыболовный сектор
Товар продажи	SaleProduct	Хранит информацию о товарах, доступных для покупки клиентами во время отдыха
Товар аренды	RentProduct	Содержит данные о товарах и оборудовании, доступных клиентам в аренду на период отдыха
Уведомление	Notification	Содержит уведомление, связанное с изменением состояния бронирования

Графическое представление концептуальной модели базы данных представлено на рисунке 12.

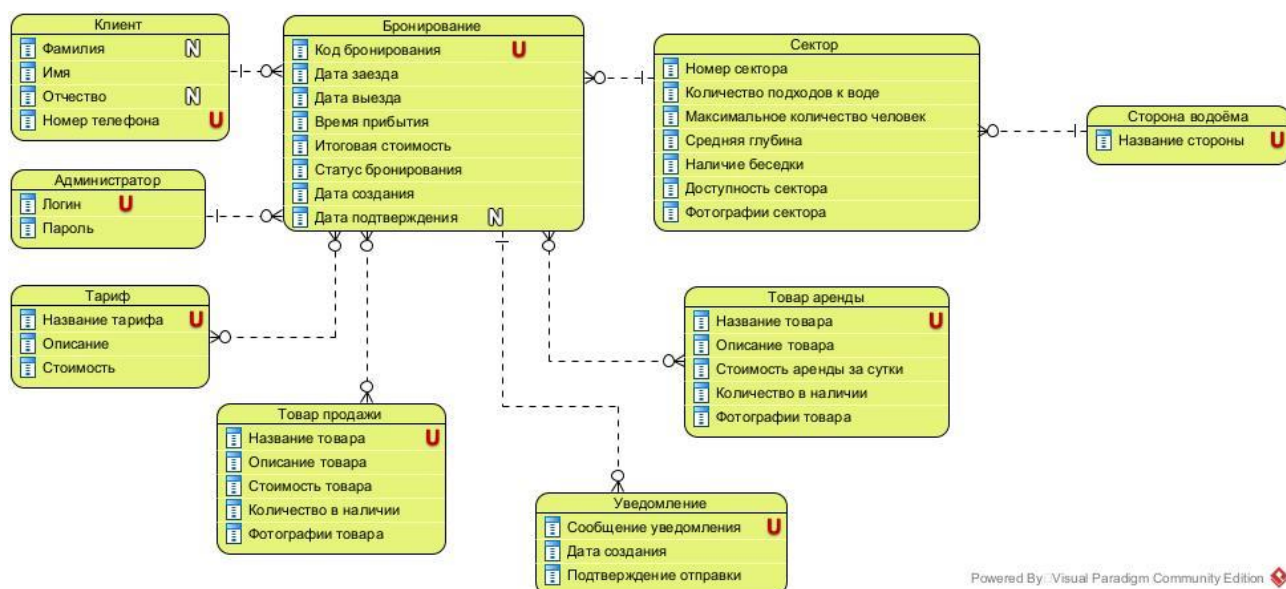


Рисунок 12 – Концептуальная модель базы данных

Таким образом, была разработана концептуальная модель разрабатываемой базы данных информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства.

2.2.2 Логическая модель базы данных

Логическая модель базы данных представляет собой более детальную и технически проработанную версию концептуальной модели [7]. Данная модель уточняет связи и ограничения данных, добавляет первичные и вторичные внешние ключи. Логическая модель является основой для последующего перехода к реализации физической модели базы данных.

На этапе построения логической модели были уточнены сущности, выделенные в концептуальной модели: «Клиент», «Администратор», «Сектор», «Сторона водоёма», «Бронирование», «Тариф», «Товар продажи», «Товар аренды» и «Уведомление». Также были добавлены промежуточные таблицы, необходимые для реализации связей типа «многие ко многим»: «Позиция бронирования», «Позиция товара продажи», «Позиция товара аренды». Логическая модель базы данных представлена на рисунке 13.

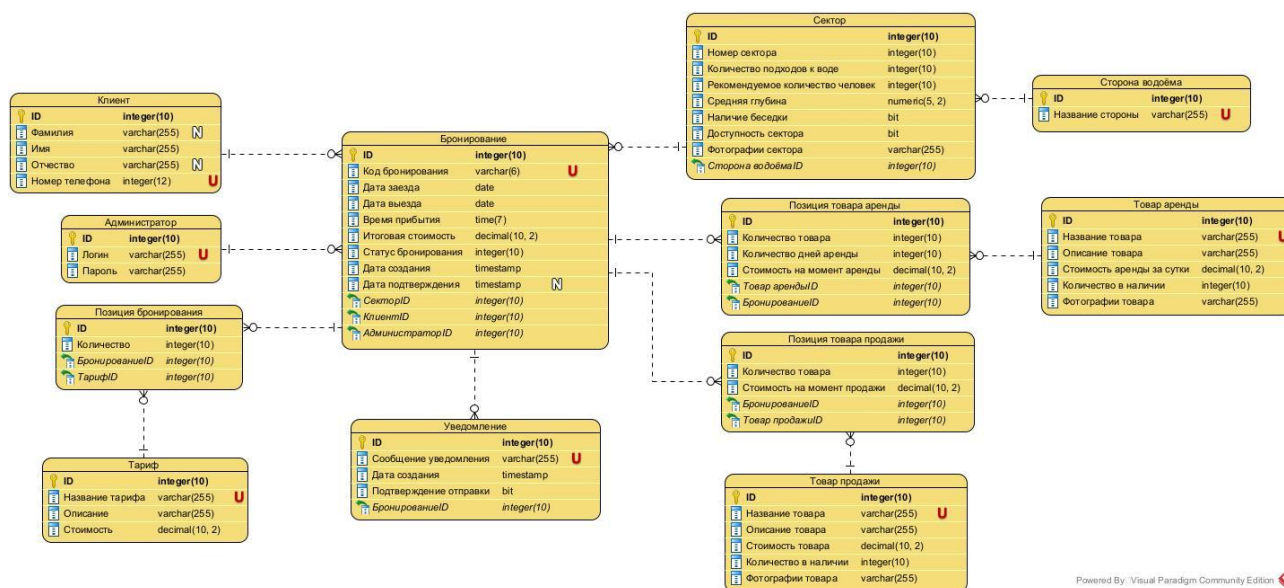


Рисунок 13 – Логическая модель базы данных

Описание атрибутов каждой сущности представлено в приложении Б.

2.3 Проектирование программных средств

На основании разработанной архитектуры приложения и логической модели базы данных выполнено проектирование программных средств информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства. Были определены основные программные компоненты серверной части приложения, их назначение и взаимодействие между собой. Также были построены диаграммы классов, разделённые на модули. Диаграммы классов основных модулей представлены в приложении Д.

Серверная часть системы реализуется с использованием фреймворка ASP.NET Core [8]. Данный фреймворк позволяет разрабатывать web-приложения и REST API, обрабатывать HTTP-запросы, выполнять маршрутизацию, подключать сервисы через механизм внедрения зависимостей и организовывать взаимодействие с базой данных.

Серверное приложение построено по многослойной архитектуре, разделяя логику системы на следующие уровни:

- слой контроллеров;
- слой бизнес-логики;
- слой интерфейсов;
- слой доступа к данным;
- слой доменных сущностей.

Слой контроллеров обрабатывает входящие HTTP-запросы, принимает данные от клиентской части, вызывает соответствующие методы сервисов и возвращает ответ.

Слой бизнес-логики содержит основные правила работы системы.

Слой интерфейсов определяет контракты взаимодействия между сервисами и репозиториями.

Слой доступа к данным выполняет операции с базой данных.

Слой доменных сущностей описывает основные объекты предметной области.

Краткая спецификация контроллеров представлена в таблице 3.

Таблица 3 – Контроллеры

Название класса	Назначение
AdminAuthController	Обрабатывает запросы, связанные с авторизацией администратора и получением JWT-токена
AdminBookingsController	Предназначен для административного управления бронированиями: просмотра, подтверждения, отклонения и завершения заявок
AdminRentProductsController	Обеспечивает управление товарами аренды со стороны администратора
AdminReportsController	Обрабатывает запросы администратора на формирование отчетов по деятельности рыболовного хозяйства
AdminSaleProductsController	Обеспечивает управление товарами продажи со стороны администратора
AdminSectorsController	Предназначен для управления рыболовными секторами: добавления, изменения и удаления данных
AdminTariffsController	Обеспечивает управление тарифами, используемыми при расчете стоимости отдыха
BookingsController	Обрабатывает клиентские операции, связанные с расчетом стоимости, созданием, просмотром и отменой бронирования
ProductsController	Предоставляет клиенту возможность просматривать товары продажи и товары аренды
SectorsController	Предоставляет клиенту возможность просматривать список секторов, применять фильтрацию и получать подробную информацию о секторе
TariffsController	Позволяет клиенту просматривать действующие тарифы

Каждый контроллер взаимодействует с соответствующим сервисом, который изолирует бизнес-логику от слоя представления. Контроллеры не выполняют сложную обработку данных, а только принимают запрос, передают данные в сервис и возвращают результат клиентской части.

Спецификация сервисов представлена в таблице 4.

Таблица 4 – Сервисы

Название класса	Назначение
AdminAuthService	Реализует логику авторизации администратора и формирования JWT-токена
BookingService	Реализует создание бронирования, просмотр данных брони, отмену бронирования и изменение статуса заявок
PricingService	Выполняет расчет стоимости отдыха на основе выбранного сектора, дат, состава группы, тарифов и дополнительных товаров
RentProductService	Реализует бизнес-логику работы с товарами аренды
TariffService	Обеспечивает получение и изменение данных о тарифах

Продолжение таблицы 4

Название класса	Назначение
SaleProductService	Реализует бизнес-логику с товарами продажи
ReportService	Формирует отчеты по выручке, загрузке секторов, популярности товаров, клиентам и статусам бронирований
SectorService	Обрабатывает получение, фильтрацию, создание, изменение и удаление данных о рыболовных секторах

Сервисы содержат основные правила работы системы. Например, при создании бронирования сервис проверяет существование выбранного сектора, корректность дат отдыха, доступность сектора на указанный период, рассчитывает итоговую стоимость и сохраняет данные бронирования в базе данных.

Каждый сервис имеет интерфейс, который определяет набор методов для взаимодействия с другими компонентами приложения. Использование интерфейсов позволяет определить описание доступных операций от их конкретной реализации.

Описание интерфейсов сервисов представлено в таблице 5.

Таблица 5 – Интерфейсы сервисов

Название класса	Назначение
IAdminAuthService	Определяет методы авторизации администратора
IBookingService	Определяет методы создания, просмотра, отмены и изменения статусов бронирований
IPricingService	Определяет методы расчета стоимости бронирований
IRentProductService	Определяет методы работы с товарами аренды
ITariffService	Определяет методы получения и изменения тарифов
IReportService	Определяет методы формирования аналитических отчетов
ISaleProductService	Определяет методы работы с товарами продажи
ISectorService	Определяет методы работы с рыболовными секторами

Репозитории обеспечивают доступ к данным, хранящимся в базе данных. Каждый репозиторий реализует методы для выполнения операций чтения, добавления, изменения и удаления данных. Использование репозитория позволяет отделить бизнес-логику от конкретных запросов к базе данных.

Перечень используемых репозитория приведен в таблице 6

Таблица 6 – Репозитории

Название класса	Назначение
AdminRepository	Обеспечивает доступ к учетным данным администратора
BookingRepository	Выполняет операции с бронированиями клиентов
NotificationRepository	Выполняет операции с уведомлениями, связанными с бронированиями
RentProductRepository	Выполняет операции с товарами аренды
SaleProductRepository	Выполняет операции с товарами продажи
SectorRepository	Выполняет операции с данными рыболовных секторов
TariffRepository	Обеспечивает доступ к данным тарифов
WaterSideRepository	Обеспечивает получение данных о сторонах водоёма

Для каждого репозитория также создан интерфейс, который определяет набор операций, доступных для работы с данными. Благодаря этому сервисы взаимодействуют не с конкретной реализацией репозитория, а с его интерфейсом.

Интерфейсы репозитория представлены в таблице 7.

Таблица 7 – Интерфейсы репозитория

Название класса	Назначение
IAdminRepository	Определяет методы доступа к учетным данным администратора
IBookingRepository	Определяет методы работы с бронированиями
INotificationRepository	Определяет методы работы с уведомлениями
IRentProductRepository	Определяет методы работы с товарами аренды
ISaleProductRepository	Определяет методы работы с товарами продажи
ISectorRepository	Определяет методы работы с данными рыболовных секторов
ITariffRepository	Определяет методы доступа к тарифам
IWaterSideRepository	Определяет методы получения данных о сторонах водоёма

Были выделены доменные сущности, соответствующие основным объектам предметной области. Их описание представлено в таблице 8.

Таблица 8 – Доменные сущности

Название класса	Назначение
Admin	Содержит данные администратора системы
Client	Содержит сведения о клиенте рыболовного хозяйства
Booking	Содержит информацию о бронировании сектора клиентом
BookingPosition	Содержит тарифную позицию бронирования, включая ссылку на бронирование, тариф и количество
Sector	Содержит характеристики рыболовного сектора

Продолжение таблицы 8

Название класса	Назначение
SaleProduct	Содержит данные о товаре, доступном для продажи
BookingRentProduct	Связывает бронирование с товарами аренды и хранит количество выбранного товара
BookingSaleProduct	Связывает бронирование с товарами продажи и хранит количество выбранного товара
Notification	Содержит уведомление, связанное с изменением состояния бронирования
RentProduct	Содержит данные о товаре, доступном для аренды
Tariff	Содержит информацию о тарифе, участвующем в расчете стоимости
WaterSide	Содержит данные о стороне водоёма, к которой относится сектор

Для настройки отображения доменных сущностей на таблицы базы данных используются конфигурационные классы. Они позволяют задать названия таблиц, первичные и внешние ключи, типы полей и связи между сущностями.

Конфигурационные классы представлены в таблице 9.

Таблица 9 – Конфигурации сущностей

Название класса	Назначение
AdminConfiguration	Определяет настройки таблицы администраторов
ClientConfiguration	Определяет настройки таблицы клиентов
BookingConfiguration	Определяет настройки таблицы бронирований
BookingPositionConfiguration	Определяет настройки таблицы тарифных позиций бронирования
RentProductConfiguration	Определяет настройки таблицы товаров аренды
SaleProductConfiguration	Определяет настройки таблицы товаров продажи
BookingRentProductConfiguration	Определяет настройки позиций товаров аренды в бронировании
BookingSaleProductConfiguration	Определяет настройки позиций товаров продажи в бронировании
NotificationConfiguration	Определяет настройки таблицы уведомлений
SectorConfiguration	Определяет настройки таблицы рыболовных секторов
TariffConfiguration	Определяет настройки таблицы тарифов
WaterSideConfiguration	Определяет настройки таблицы сторон водоёма

Для передачи данных между клиентской и серверной частью используются DTO-модели. Они позволяют не передавать клиенту напрямую доменные сущности и ограничивать набор возвращаемых данных. В проекте DTO-модели разделены по назначению: авторизация, бронирования, товары, отчеты, сектора, настройки и тарифы.

В проекте также предусмотрены вспомогательные классы и исключения, позволяющие обработать различные ошибочные ситуации в бизнес-логике приложения. Обработка ошибок выполняется централизованно.

Для обеспечения безопасности в приложении предусмотрены два класса. Первый отвечает за формирование JWT-токена для администратора, а второй используется для хеширования и проверки пароля.

Таким образом, в рамках проектирования программных средств были определены основные компоненты серверной части информационной системы, их назначение и взаимодействие. Разделение приложения на контроллеры, сервисы, интерфейсы, репозитории, конфигурации и доменные сущности обеспечивает модульность, удобство сопровождения и возможность дальнейшего расширения функциональности системы.

2.4 Проектирование пользовательского интерфейса

Для разрабатываемой информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства были определены основные требования к пользовательскому интерфейсу.

Интерфейс программного продукта планируется реализовать с использованием HTML, Tailwind CSS, TypeScript и фреймворка React. Выбор данных средств разработки обусловлен тем, что они позволяют создавать адаптивный, удобный и масштабируемый интерфейс, который может корректно отображаться на различных устройствах [9].

При разработке пользовательского интерфейса необходимо учитывать принципы удобства, единообразия визуального оформления, читаемости элементов и логичности переходов между страницами. Интерфейс должен быть понятен как администраторам системы, так и обычным клиентам.

Для администратора системы предусмотрены следующие страницы:

- окно регистрации: позволяет пройти аутентификацию и авторизацию администраторам системы, используя логин и пароль;

- окно управления секторами: позволяет контролировать список имеющихся секторов, добавляя или удаляя любой из секторов;
- окно просмотра сектора: позволяет перейти на выбранный сектор, при необходимости возможно изменение параметров для текущего сектора;
- окно управления бронированиями: позволяет открыть список имеющихся броней, которые администратор либо подтверждает в случае успешного выполнения всех условий, либо же отклоняет текущую бронь;
- окно управления тарифами: позволяет ознакомиться с имеющимися тарифами, которые используются в итоговом подсчете стоимости бронирования, а также изменить уже существующий тариф;
- окно управления товарами на аренду и продажу: позволяет ознакомиться с текущим каталогом товаров для аренды или продажи, которые используются в конечном подсчёте суммы бронирования сектора; при необходимости возможно добавление или удаление товара, а также его изменение;
- окно аналитики: позволяет получить текущую статистику системы по группе параметров: выручке за выбранный период, популярности секторов или загрузке секторов.

Для клиента рыболовного хозяйства предусматривается следующий набор страниц:

- окно главной страницы: позволяет ознакомиться с основной информацией о рыболовном хозяйстве, узнать его историю и структуру;
- окно списка секторов: позволяет перейти к каталогу имеющихся секторов для отдыха на выбор;
- окно осуществления бронирования: позволяет перейти к конкретному сектору и подсчитать сумму бронирования, которая зависит от введенных клиентом параметров: даты заезда и выезда, количества людей, наличия беседки или добавления желаемых товаров, а также осуществить само бронирование;

- окно просмотра броней: позволяет узнать список собственных броней и ознакомиться с их статусом в ходе продвижения; при обновлении статус брони изменяется на соответствующий;

- окно тарифов: позволяет посмотреть имеющиеся тарифы для разных категорий людей и узнать их стоимость;

- окно товаров: позволяет изучить каталог товаров аренды и продажи, который клиент может добавить в собственную бронь на выбор.

Каждая страница пользовательского интерфейса должна взаимодействовать с серверной частью системы через REST API. Для выполнения операций используются соответствующие HTTP-запросы (GET, POST, PUT и DELETE).

При вводе некорректных или недопустимых данных система должна отображать пользователю понятное текстовое сообщение об ошибке. При успешном выполнении операции пользователь должен получать уведомление о результате действия.

2.5 Выбор технологий и инструментальных средств разработки приложения

2.5.1 Выбор технологий разработки приложения

При создании сложных проектов от разработчиков требуется знание современных методологий, принципов и идей программирования. Одной из таких методологий является парадигма объектно-ориентированного программирования (ООП). ООП позволяет представить элементы предметной области в виде классов и объектов. Класс – это шаблон какого-либо объекта, который содержит данные и методы, описывающие объект и работающие с ним. Объект – это конкретный экземпляр класса, который уже содержит определённые данные.

Использование ООП позволяет разделить программный код на отдельные классы, каждый из которых выполняет свою задачу. Классы доменных

сущностей описывают структуру данных, сервисы содержат бизнес-логику, репозитории отвечают за доступ к данным, а контроллеры обрабатывают запросы пользователей. Данное разделение делает код более понятным и простым, а также позволяет в дальнейшем расширять функциональность системы без полной переработки существующих компонентов. Помимо этого, согласно принципам ООП, важным его аспектом является инкапсуляция – это механизм ограничения доступа к внутреннему состоянию объекта и объединения данных с методами их обработки. Учитывая все эти аспекты, в разработке данного проекта в качестве методологии была выбрана парадигма ООП.

В качестве основного языка программирования выбран С# [10]. Данный язык поддерживает объектно-ориентированное программирование, строго типизирован, хорошо интегрируется с платформой .NET и позволяет разрабатывать надежные серверные приложения.

Для реализации серверной части была выбрана технология ASP.NET Core, так как она позволяет разрабатывать REST API, поддерживает внедрение зависимостей, удобную маршрутизацию запросов и хорошо интегрируется с Entity Framework Core.

REST API выбран для организации взаимодействия между клиентской и серверной частью приложения [11], так как данный подход позволяет передавать данные через HTTP-запросы в формате JSON.

Для обмена данными между клиентом и сервером используется формат JSON. Данный формат удобен для передачи структурированных данных и широко применяется в современных web-приложениях.

Для доступа к базе данных используется технология Entity Framework Core [12]. Она позволяет работать с данными не напрямую через SQL-запросы, а через классы и объекты приложения. Это упрощает разработку, так как таблицы базы данных сопоставляются с доменными сущностями системы.

В качестве системы управления базами данных (СУБД) используется PostgreSQL [13]. Данная СУБД подходит для хранения структурированных

данных, поддерживает связи между таблицами, внешние ключи, ограничения целостности и различные типы данных.

Для защиты административной части приложения используется технология JWT-аутентификации [14]. После успешной авторизации администратор получает токен доступа, который используется при выполнении защищенных запросов. Это позволяет разграничить клиентскую и административную части системы.

В разрабатываемом приложении используется многослойный подход к организации программного кода. Система разделена на 4 проекта. Такое разделение позволяет вынести контроллеры, бизнес-логику, доменные сущности и доступ к данным в отдельные части приложения. Благодаря этому упрощается сопровождение проекта, а изменения в одном слое оказывают меньшее влияние на остальные части системы.

2.5.2 Выбор инструментальных средств разработки приложения

После выбора технологий разработки необходимо определить инструментальные средства, с помощью которых будет выполняться создание, настройка и тестирование информационной системы. Инструментальные средства должны обеспечивать удобную разработку серверной части приложения, работу с базой данных, проектирование диаграмм и проверку работоспособности API.

В качестве среды разработки была выбрана Microsoft Visual Studio 2022. Данная среда разработки удобна для создания приложений на языке C# и хорошо подходит для работы с проектами ASP.NET Core. Visual Studio предоставляет разработчику встроенный редактор кода, средства отладки, подсветку синтаксиса, автодополнение, управление зависимостями и возможность запуска приложения непосредственно из среды разработки. Для разрабатываемой информационной системы Visual Studio 2022 является основным инструментом, так как проект состоит из нескольких частей. Такое разделение проекта удобно поддерживать именно в Visual Studio, поскольку

обозреватель решений позволяет наглядно видеть структуру приложения, быстро переходить между контроллерами, сервисами, репозиториями, DTO-моделями и доменными сущностями.

Для управления базой данных используется pgAdmin. Данный инструмент предоставляет графический интерфейс для работы с СУБД PostgreSQL. С его помощью можно создавать и просматривать таблицы, выполнять SQL-запросы, проверять добавленные данные и анализировать связи между таблицами.

Для проверки работы серверной части используется Swagger. Swagger – это инструмент для описания, просмотра и тестирования REST API [15]. Он автоматически формирует web-интерфейс, в котором отображаются доступные запросы серверной части приложения, их параметры, формат входных данных и структура возвращаемого ответа. Благодаря Swagger можно проверить работу API без разработки отдельного пользовательского интерфейса.

Для проектирования диаграмм была выбрана среда Visual Paradigm [16]. Данный инструмент используется для построения UML-диаграмм и моделей базы данных. В рамках выпускной квалификационной работы Visual Paradigm применяется для разработки диаграмм вариантов использования, концептуальной и логической модели базы данных, а также других графических материалов, необходимых для пояснительной записки. Использование Visual Paradigm позволяет наглядно представить структуру предметной области, связи между сущностями и взаимодействие пользователей с системой.

Для планирования разработки и оценки бюджета программного обеспечения была выбрана среда ProjectLibre [17]. Этот продукт позволяет составить различные планы по затратам на разработку приложения, рассчитать общую стоимость проекта и составить все необходимые диаграммы, связанные с планированием.

3 Разработка программного обеспечения

3.1 Разработка физической модели базы данных

Физическая модель базы данных – это точное представление структуры базы данных, предназначенное для конкретной СУБД. Она определяет, как логическая модель будет реализована на физическом уровне [18].

На основе ранее разработанных концептуальной и логической моделей, была разработана физическая модель базы данных, представленная на рисунке 14.

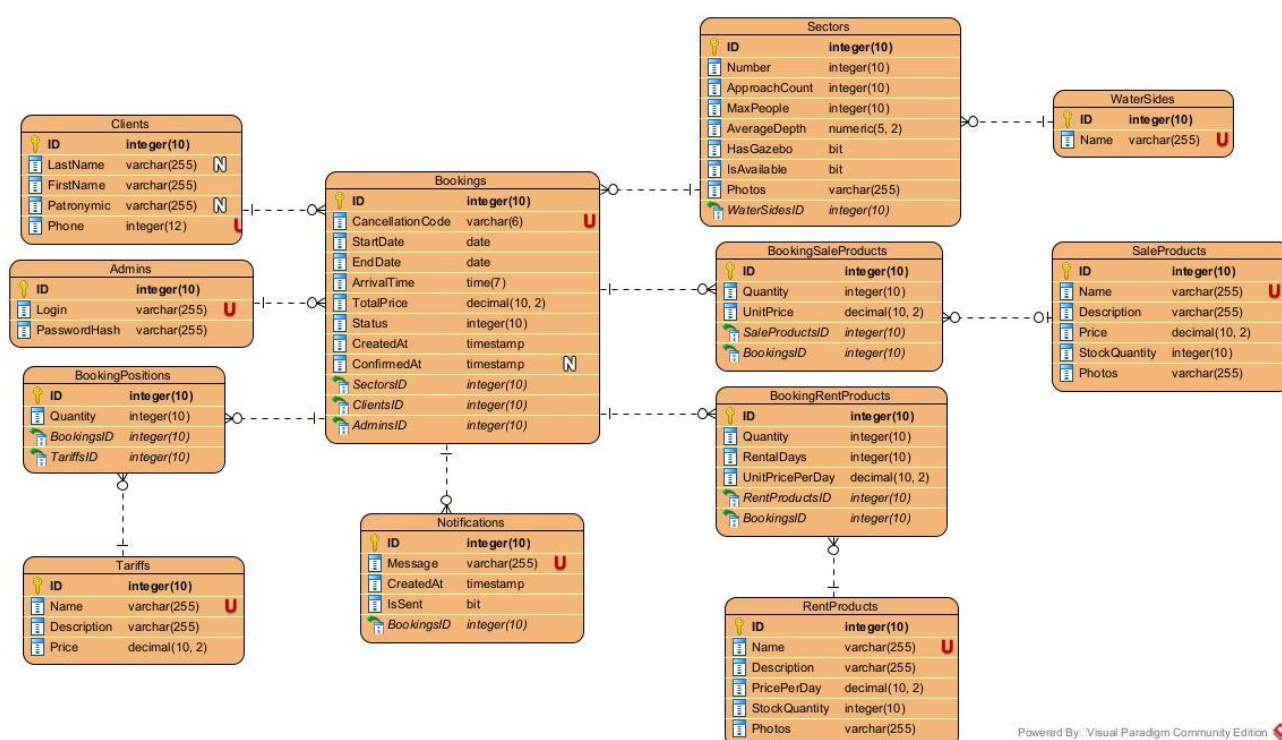


Рисунок 14 – Физическая модель базы данных

Согласно физической модели с помощью СУБД PostgreSQL была развернута база данных информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства.

Таким образом, была реализована база данных системы автоматизации бронирования и учета услуг рыболовного хозяйства, которая обеспечивает хранение, обработку и управление данными, а также поддерживает связи

между таблицами, необходимые для корректной работы основных бизнес-процессов приложения.

3.2 Реализация серверной части информационной системы

Серверная часть информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства была реализована в среде разработки Microsoft Visual Studio 2022 на языке программирования C# с использованием фреймворка ASP.NET Core. Фрагменты кода серверной части системы приведены в приложении В. С полным кодом можно ознакомиться на GitHub [19].

Серверная часть приложения отвечает за обработку запросов от клиентской части, выполнение бизнес-логики, взаимодействие с базой данных, расчет стоимости отдыха, оформление бронирований, управление административными функциями и формирование отчетов.

Разработанная серверная часть построена по принципу монолитной многослойной архитектуры. В рамках реализации были выделены следующие проекты:

- FishingPondBookingSystem.API;
- FishingPondBookingSystem.Application;
- FishingPondBookingSystem.Domain;
- FishingPondBookingSystem.Infrastructure.

Проект FishingPondBookingSystem.API содержит контроллеры, точку входа приложения, настройки маршрутизации, подключение сервисов и промежуточный компонент для централизованной обработки ошибок.

Проект FishingPondBookingSystem.Application содержит основную бизнес-логику приложения. В нём реализованы сервисы, DTO-модели, интерфейсы сервисов и репозиториях, вспомогательные классы и пользовательские исключения. Данный слой отвечает за обработку бронирований, расчет стоимости, управление секторами, товарами, тарифами и отчетами.

Проект FishingPondBookingSystem.Domain содержит доменные сущности и перечисления предметной области. В данном проекте описаны основные объекты системы: бронирование, сектор, сторона водоёма, товар продажи, товар аренды, тариф, администратор и уведомление.

Проект FishingPondBookingSystem.Infrastructure реализует доступ к базе данных. В нем находятся контекст базы данных, репозитории, конфигурации сущностей, миграции и классы, связанные с безопасностью приложения.

Взаимодействие между клиентской и серверной частью осуществляется через REST API. Клиентская часть отправляет HTTP-запросы на сервер, после чего контроллер принимает запрос, передает данные в соответствующий сервис, сервис выполняет бизнес-логику и при необходимости обращается к репозиторию для работы с базой данных. Результат выполнения операции возвращается клиенту в формате JSON.

Основные модули серверной части представлены в таблице 10.

Таблица 10 – Основные модули серверной части

Модуль	Назначение
Модуль авторизации администратора	Обеспечивает вход администратора в систему и выдачу JWT-токена
Модуль клиентов	Обеспечивает хранение и получение данных клиентов, связанных с бронированиями
Модуль бронирований	Отвечает за расчет стоимости, создание, просмотр, отмену и изменение статуса бронирования
Модуль секторов	Обеспечивает получение, фильтрацию, добавление, изменение и удаление рыболовных секторов
Модуль товаров продажи	Реализует просмотр и административное управление товарами, доступными для покупки
Модуль товаров аренды	Реализует просмотр и административное управление товарами, доступными для аренды
Модуль тарифов	Обеспечивает просмотр и изменение тарифов, используемых при расчете стоимости отдыха
Модуль отчетов	Формирует аналитические отчеты по деятельности рыболовного хозяйства
Модуль уведомлений	Создает уведомления, связанные с изменением состояния бронирования

Для обработки запросов были реализованы контроллеры, каждый из которых отвечает за отдельную группу функций. Клиентские контроллеры предназначены для просмотра информации и оформления бронирования, а административные контроллеры обеспечивают управление данными системы.

Методы API клиентской части представлены в таблице 11.

Таблица 11 – Методы API клиентской части

URL запроса	HTTP-метод	Описание
/sectors	GET	Получение списка рыболовных секторов с возможностью фильтрации
/sectors/{id}	GET	Получение подробной информации о выбранном секторе
/tariffs	GET	Получение списка действующих тарифов
/products/sale	GET	Получение списка товаров продажи
/products/rent	GET	Получение списка товаров аренды
/bookings	GET	Получение информации о бронированиях
/bookings/{id}	GET	Получение подробной информации о конкретном бронировании
/bookings	POST	Создание бронирования
/bookings/{id}	DELETE	Отмена бронирования клиентом
/bookings/calculate	POST	Расчет стоимости отдыха на основе выбранного сектора, дат, состава группы и дополнительных товаров

Административная часть серверного приложения защищена с помощью JWT-аутентификации. Для доступа к административным методам пользователь должен пройти авторизацию и получить токен доступа. Полученный токен передается в заголовке запроса и проверяется сервером при обращении к защищенным методам.

Методы API административной части представлены в таблице 12.

Таблица 12 – Методы API административной части

URL запроса	HTTP-метод	Описание
/admin/login	POST	Авторизация администратора и получение JWT-токена
/admin/bookings	GET	Получение списка бронирований для администратора
/admin/bookings/{id}/status	PUT	Изменение статуса бронирования администратором
/admin/products/rent	GET	Получение списка товаров аренды для администратора
/admin/products/rent	POST	Добавление нового товара аренды
/admin/products/rent/{id}	PUT	Изменение данных товара аренды
/admin/products/rent/{id}	DELETE	Удаление товара аренды
/admin/products/sale	GET	Получение списка товаров продажи для администратора
/admin/products/sale	POST	Добавление нового товара продажи
/admin/products/sale/{id}	PUT	Изменение данных товара продажи
/admin/products/sale/{id}	DELETE	Удаление товара продажи
/admin/sectors	GET	Получение списка рыболовных секторов для администратора
/admin/sectors	POST	Добавление нового рыболовного сектора
/admin/sectors/{id}	PUT	Изменение данных рыболовного сектора
/admin/sectors/{id}	DELETE	Удаление рыболовного сектора
/admin/tariffs	GET	Получение списка тарифов для администратора
/admin/tariffs/{id}	PUT	Изменение данных тарифа
/admin/reports/revenue	GET	Формирование отчета по выручке

Продолжение таблицы 12

URL запроса	HTTP-метод	Описание
/admin/reports/sector-load	GET	Формирование отчета по загрузке секторов
/admin/reports/top-products	GET	Формирование отчета по популярности товаров
/admin/reports/customers	GET	Формирование отчета по клиентам
/admin/reports/booking-statuses	GET	Формирование отчета по статусам бронирований

Несмотря на то, что система реализована в виде монолитного приложения, ее внутренняя структура разделена на отдельные функциональные модули и слои. Такое построение снижает связанность компонентов и создает основу для дальнейшего расширения системы, а также при необходимости может упростить постепенный переход к микросервисной архитектуре.

3.3 Реализация клиентской части информационной системы

При разработке информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства отдельное внимание было уделено клиентской части приложения. Описание основных компонентов, используемых в клиентском приложении, представлено в таблице 13.

Таблица 13 – Описание компонентов клиентской части

Название компонента	Назначение компонента
AdminPanel.tsx	Главная страница администратора
BookingForm.tsx	Страница оформления бронирования
BookingList.tsx	Страница просмотра текущих броней администратором
ClientHome.tsx	Главная страница клиента
ClientNotifications.tsx	Страница просмотра уведомлений клиентом
ClientProductPage.tsx	Страница просмотра товаров
ClientTariffPage.tsx	Страница просмотра тарифов
LoginForm.tsx	Страница входа для администратора
ProductTable.tsx	Страница управления товарами
SectorTable.tsx	Страница управления секторами
StatsPage.tsx	Страница управления аналитикой
TariffTable.tsx	Страница управления тарифами
UserBooking.tsx	Страница просмотра текущих броней клиентом
App.tsx	Структура главной страницы
App.css	Стилизация главной страницы

Клиентская часть была реализована с использованием HTML и CSS, а также фреймворка Tailwind CSS для оформления интерфейса. Для разработки логики пользовательской части применялись язык программирования TypeScript и фреймворк React. Взаимодействие клиентской части с сервером выполнялось посредством HTTP-запросов через REST API.

4 Отладка и тестирование программного обеспечения

4.1 Функциональное тестирование методом белого ящика

4.1.1 Покрытие операторов

Для проверки корректности работы серверной части информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства было проведено функциональное тестирование методом белого ящика (White-box testing). Данный метод основан на анализе внутренней структуры программного кода и позволяет проверить выполнение операторов, условий и ветвлений внутри тестируемого метода.

Метод белого ящика был выбран по следующим причинам:

- исходный код полностью доступен для анализа;
- основная бизнес-логика системы реализована в сервисах приложения;
- расчёт стоимости бронирования содержит несколько условий, проверок и циклов;
- корректность расчёта стоимости напрямую влияет на итоговую стоимость бронирования;
- метод позволяет оценить полноту проверки программного кода.

В качестве объекта тестирования был выбран метод `CalculateAsync` сервиса `PricingService`. Данный метод выполняет расчёт итоговой стоимости бронирования на основе выбранного сектора, дат отдыха, состава группы отдыхающих, тарифов, товаров продажи, товаров аренды и дополнительных услуг. Код данного метода представлен в приложении В, листинг В.4.

В соответствии с критерием покрытия операторов необходимо составить такие тесты, при которых каждый оператор программы выполнится хотя бы один раз. Для метода `CalculateAsync` были выделены следующие основные операторы:

- а – проверка корректности дат бронирования;
- b – получение сектора по идентификатору;

- c – проверка существования сектора;
- d – формирование исключения при отсутствии сектора;
- e – проверка доступности сектора;
- f – формирование исключения при недоступности сектора;
- g – подсчет общего количества отдыхающих;
- h – проверка вместимости сектора;
- i – формирование исключения при превышении вместимости;
- j – вычисление количества дней отдыха;
- k – создание списка тарифных позиций;
- l-r – получение тарифов из базы данных;
- s-w – добавление тарифных позиций по категориям отдыхающих;
- x – расчёт стоимости проживания;
- y – инициализация стоимости беседки;
- z – проверка наличия беседки;
- aa-ab – добавление позиции беседки и расчёт её стоимости;
- ac – инициализация фиксированной стоимости бронирования;
- ad – проверка необходимости включения стоимости бронирования сектора;
- ae-af – добавление позиции бронирования сектора и расчет её стоимости;
- ag – инициализация стоимости товаров продажи;
- ah – цикл обработки товаров продажи;
- ai – проверка количества товара продажи;
- aj – формирование исключения при некорректном количестве товара продажи;
- ak – получение товара продажи по идентификатору;
- al – проверка существования товара продажи;
- am – формирование исключения при отсутствии товара продажи;
- an – расчёт стоимости товаров продажи;

- ao – инициализация стоимости товаров аренды;
- ap – цикл обработки товаров аренды;
- aq – проверка количества товара аренды;
- ar – формирование исключения при некорректном количестве товара аренды;
- as – получение товара аренды по идентификатору;
- at – проверка существования товара аренды;
- au – формирование исключения при отсутствии товара аренды;
- av – расчёт стоимости товаров аренды;
- aw – расчёт итоговой стоимости бронирования;
- ax – возврат результата расчёта.

На рисунке 15 представлена блок-схема метода CalculateAsync.

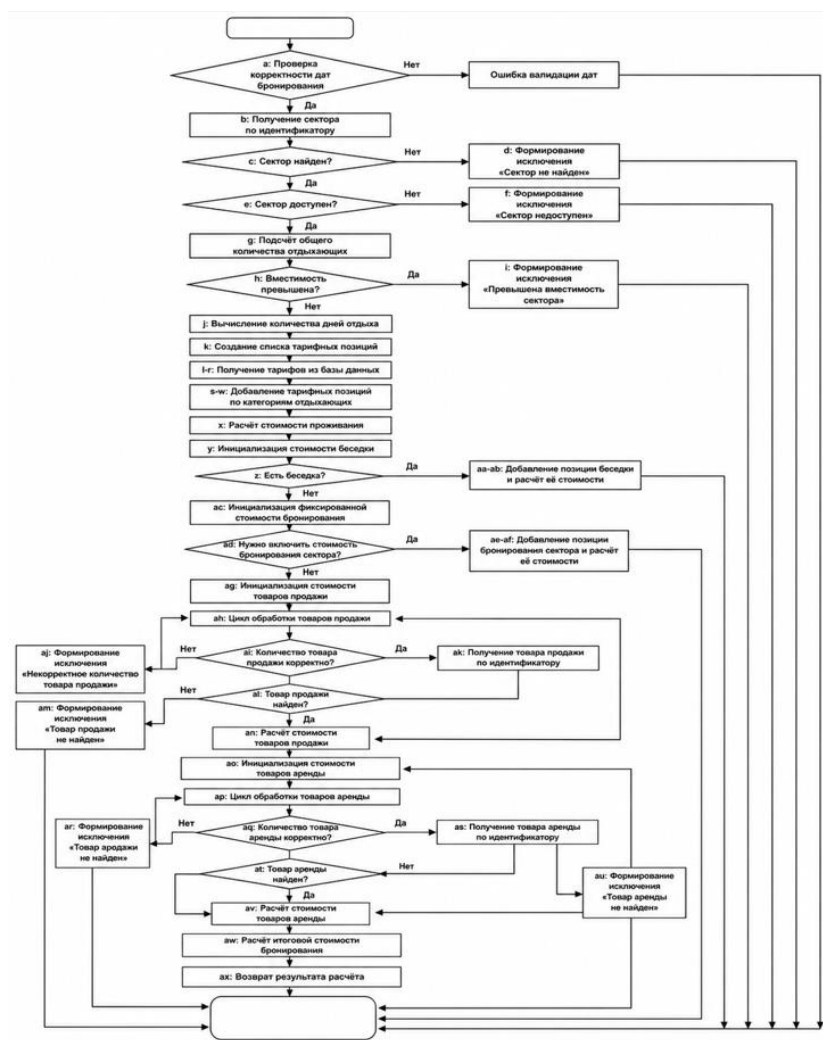


Рисунок 15 – Блок-схема метода CalculateAsync

Чтобы выполнить каждый оператор не менее одного раза, необходимо составить набор тестов, включающий как успешные, так и ошибочные сценарии.

Был задан базовый набор входных данных, который используется в тест-кейсах, если в строке таблицы не указано иное.

Базовый набор входных данных:

- SectorId = 1;
- IsAvailable = true;
- MaxPeople = 10;
- HasGazebo = false;
- IncludeBookingFee = true;
- StartDate = 2026-08-10;
- EndDate = 2026-08-13;
- FishermenCount = 2;
- FemaleFishermenCount = 1;
- MaleGuestsCount = 1;
- FemaleGuestsCount = 1;
- ChildrenCount = 1;
- SaleProducts = [];
- RentProducts = [].

Для расчёта стоимости использовались следующие тарифы:

- FishermanTariff = 800;
- FemaleFishermanTariff = 500;
- MaleGuestTariff = 200;
- FemaleGuestTariff = 0;
- ChildTariff = 0;
- GazeboTariff = 200;
- SectorBookingTariff = 200.

Также в тестах использовались следующие товары:

- SaleProduct.Id = 1;
- SaleProduct.Price = 180;
- RentProduct.Id = 1;
- RentProduct.PricePerDay = 300.

Результаты тестирования представлены в таблице 14.

Таблица 14 – Тест-кейсы для покрытия операторов метода CalculateAsync

№	Операторы	Входные данные	Ожидаемый результат	Фактический результат
1	a	StartDate = 2026-08-10; EndDate = 2026-08-10	ValidationException: Дата выезда должна быть позже даты заезда	ValidationException: Дата выезда должна быть позже даты заезда (результат соответствует ожидаемому)
2	a, b, c, d	Базовый набор; SectorId = 999;	NotFoundException: Сектор не найден	NotFoundException: Сектор не найден (результат соответствует ожидаемому)
3	a, b, c, e, f	Базовый набор; IsAvailable = false	BusinessRuleException: Сектор недоступен для бронирования	BusinessRuleException: Сектор недоступен для бронирования (результат соответствует ожидаемому)
4	a, b, c, e, g, h, i	Базовый набор; MaxPeople = 2	BusinessRuleException: Количество человек превышает допустимую вместимость сектора	BusinessRuleException: Количество человек превышает допустимую вместимость сектора (результат соответствует ожидаемому)
5	a, b, c, e, g, h, j, k, l-r, s-w, x, y, z, ac, ad, ae-af, ag, ah, ao, ap, aw, ax	Базовый набор;	TotalPrice = 7100	TotalPrice = 7100 (результат соответствует ожидаемому)
6	z, aa-ab	Базовый набор; HasGazebo = true	TotalPrice = 7700	TotalPrice = 7700 (результат соответствует ожидаемому)
7	ah, ai, aj	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество товара продажи должно быть больше нуля	ValidationException: Количество товара продажи должно быть больше нуля (результат соответствует ожидаемому)
8	ah, ai, ak, al, am	Базовый набор; SaleProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар продажи с id=999 не найден	NotFoundException: Товар продажи с id=999 не найден (результат соответствует ожидаемому)
9	ah, ai, ak, al, an	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 7640	TotalPrice = 7640 (результат соответствует ожидаемому)
10	aq, ar	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество арендного товара должно быть больше нуля	ValidationException: Количество арендного товара должно быть больше нуля (результат соответствует ожидаемому)

Продолжение таблицы 14

№	Операторы	Входные данные	Ожидаемый результат	Фактический результат
11	ap, aq, as, at, au	Базовый набор; RentProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар аренды с id=999 не найден	NotFoundException: Товар аренды с id=999 не найден (результат соответствует ожидаемому)
12	ap, aq, as, at, av	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 9800	TotalPrice = 9800 (результат соответствует ожидаемому)
13	a, b, c, e, g, h, j, k, l-r, s- w, x, y, z, aa- ab, ac, ad, ae-af, ag, ah, ai, ak, al, an, ao, ap, aq, as, at, av, aw, ax	Базовый набор; HasGazebo=true; SaleProducts = [{ProductId = 1; Quantity = 3}]; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 10940	TotalPrice = 10940 (результат соответствует ожидаемому)

Для достижения покрытия операторов метода CalculateAsync были составлены тестовые сценарии, проверяющие основные варианты выполнения метода.

В тестах были рассмотрены ошибочные ситуации, связанные с некорректными датами, отсутствием сектора, недоступностью сектора, превышением вместимости и некорректным количеством товаров. Также были проверены успешные сценарии расчёта стоимости: без товаров, с беседкой, с товарами продажи, с товарами аренды и полный сценарий с использованием всех основных элементов расчета.

4.1.2 Покрытие решений

В соответствии с критерием покрытия решений необходимо составить такие тесты, при которых каждое решение, т. е. каждый условный оператор программы, примет как истинное значение, так и ложное значение хотя бы 1 раз. Данный критерий позволяет проверить корректность работы всех ветвлений метода и убедиться, что программа правильно обрабатывает как успешные, так и ошибочные сценарии.

Для метода CalculateAsync были выделены следующие решения:

- a1 – дата выезда меньше даты заезда либо совпадает с ней;
- a2 – дата заезда находится в прошлом;

- a3 – даты бронирования находятся вне допустимого сезона;
- c – выбранный сектор не найден;
- e – выбранный сектор недоступен для бронирования;
- h – количество отдыхающих превышает рекомендуемую вместимость сектора;
- z – сектор имеет беседку;
- ad – необходимо включить фиксированную стоимость бронирования сектора;
- ah – список товаров продажи содержит элементы;
- ai – количество товара продажи меньше либо равно 0;
- al – товар продажи не найден;
- ar – список товаров аренды содержит элементы;
- aq – количество товара аренды меньше либо равно 0;
- at – товар аренды не найден.

Результаты тестирования представлены в таблице 15.

Таблица 15 – Тест-кейсы для покрытий решений метода CalculateAsync

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	a1 – да	StartDate = 2026-08-10; EndDate = 2026-08-10	ValidationException: Дата выезда должна быть позже даты заезда	ValidationException: Дата выезда должна быть позже даты заезда (результат соответствует ожидаемому)
2	a1 – нет, a2 – да	StartDate = 2026-04-30; EndDate = 2026-08-10	ValidationException: Дата заезда не может быть в прошлом	ValidationException: Дата заезда не может быть в прошлом (результат соответствует ожидаемому)
3	a1 – нет, a2 – нет, a3 – да	StartDate = 2026-01-10; EndDate = 2026-10-10	ValidationException: Бронирование доступно только в установленный сезон	ValidationException: Бронирование доступно только в установленный сезон (результат соответствует ожидаемому)
4	a1 – нет, a2 – нет, a3 – нет, c – да	Базовый набор; SectorId = 999;	NotFoundException: Сектор не найден	NotFoundException: Сектор не найден (результат соответствует ожидаемому)
5	c – нет, e – да	Базовый набор; IsAvailable = false	BusinessRuleException: Сектор недоступен для бронирования	BusinessRuleException: Сектор недоступен для бронирования (результат соответствует ожидаемому)

Продолжение таблицы 15

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
6	с – нет, е – нет, h – да	Базовый набор; MaxPeople = 2	BusinessRuleException: Количество человек превышает допустимую вместимость сектора	BusinessRuleException: Количество человек превышает допустимую вместимость сектора (результат соответствует ожидаемому)
7	h – нет, z – нет, ad – да, ah – нет, ap – нет	Базовый набор;	TotalPrice = 7100	TotalPrice = 7100 (результат соответствует ожидаемому)
8	h – нет, z – да, ad – да, ah – нет, ap – нет	Базовый набор; HasGazebo = true	TotalPrice = 7700	TotalPrice = 7700 (результат соответствует ожидаемому)
9	ah – да, ai – да	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество товара продажи должно быть больше нуля	ValidationException: Количество товара продажи должно быть больше нуля (результат соответствует ожидаемому)
10	ah – да, ai – нет, al – да	Базовый набор; SaleProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар продажи с id=999 не найден	NotFoundException: Товар продажи с id=999 не найден (результат соответствует ожидаемому)
11	ah – да, ai – нет, al – нет	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 7640	TotalPrice = 7640 (результат соответствует ожидаемому)
12	ap – да, aq – да	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество арендного товара должно быть больше нуля	ValidationException: Количество арендного товара должно быть больше нуля (результат соответствует ожидаемому)
13	ap – да, aq – нет, at – да	Базовый набор; RentProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар аренды с id=999 не найден	NotFoundException: Товар аренды с id=999 не найден (результат соответствует ожидаемому)
14	ap – да, aq – нет, at – нет	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 9800	TotalPrice = 9800 (результат соответствует ожидаемому)
15	a1 – нет, a2 – нет, a3 – нет, с – нет, е – нет, h – нет, z – да, ad – да, ah – да, ai – нет, al – нет, ap – да, aq – нет, at – нет	Базовый набор; HasGazebo=true; SaleProducts = [{ProductId = 1; Quantity = 3}]; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 10940	TotalPrice = 10940 (результат соответствует ожидаемому)

Для достижения покрытия решений метода `CalculateAsync` были составлены тесты, в которых каждое решение принимает как истинное, так и ложное значение. Ошибочные сценарии позволяют проверить обработку некорректных дат, отсутствующего сектора, недоступного сектора, превышения вместимости, некорректного количества товаров и отсутствия товаров в базе данных.

Успешные сценарии позволяют проверить выполнение ветвей, связанных с наличием и отсутствием беседки, учетом фиксированной стоимости бронирования, обработкой товаров продажи и товаров аренды. Полный успешный сценарий проверяет совместную работу всех основных ветвей метода.

4.1.3 Покрытие условий

В соответствии с критерием покрытия условий необходимо составить такое количество тестов, при котором все возможные результаты каждого простого условия в решении будут выполнены хотя бы один раз.

Следует учитывать, что данный критерий не всегда приводит к выполнению каждого оператора программы хотя бы один раз. Поэтому дополнительно необходимо следить за тем, чтобы в результате составления тестов каждый основной оператор метода также был выполнен не менее одного раза.

Для метода `CalculateAsync` были выделены следующие условия:

- a1 – дата выезда меньше либо равна дате заезда;
- a2 – дата заезда находится в прошлом;
- a3.1 – дата заезда находится раньше начала сезона;
- a3.2 – дата выезда находится позже окончания сезона;
- c – выбранный сектор не найден;
- e – выбранный сектор недоступен для бронирования;
- h – количество отдыхающих превышает рекомендуемую вместимость сектора;

- z – сектор имеет беседку;
- ad – необходимо включить фиксированную стоимость бронирования сектора;
- ah – список товаров продажи содержит элементы;
- ai – количество товара продажи меньше либо равно 0;
- al – товар продажи не найден;
- ar – список товаров аренды содержит элементы;
- aq – количество товара аренды меньше либо равно 0;
- at – товар аренды не найден.

Составное условие проверки сезона было разбито на два простых условия: a3.1 и a3.2. Это связано с тем, что бронирование может быть недоступно как из-за слишком ранней даты заезда, так и из-за слишком поздней даты выезда.

Результаты тестирования представлены в таблице 16.

Таблица 16 – Тест-кейсы для покрытия условий метода CalculateAsync

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	a1 – да	StartDate = 2026-08-10; EndDate = 2026-08-10	ValidationException: Дата выезда должна быть позже даты заезда	ValidationException: Дата выезда должна быть позже даты заезда (результат соответствует ожидаемому)
2	a1 – нет, a2 – да	StartDate = 2026-04-30; EndDate = 2026-08-10	ValidationException: Дата заезда не может быть в прошлом	ValidationException: Дата заезда не может быть в прошлом (результат соответствует ожидаемому)
3	a1 – нет, a2 – нет, a3.1 – да, a3.2 – нет	StartDate = 2026-01-10; EndDate = 2026-08-10	ValidationException: Бронирование доступно только в установленный сезон	ValidationException: Бронирование доступно только в установленный сезон (результат соответствует ожидаемому)
4	a1 – нет, a2 – нет, a3.1 – нет, a3.2 – да	StartDate = 2026-08-10; EndDate = 2026-11-10	ValidationException: Бронирование доступно только в установленный сезон	ValidationException: Бронирование доступно только в установленный сезон (результат соответствует ожидаемому)
5	a1 – нет, a2 – нет, a3.1 – нет, a3.2 – нет, c – да	Базовый набор; SectorId = 999;	NotFoundException: Сектор не найден	NotFoundException: Сектор не найден (результат соответствует ожидаемому)
6	c – нет, e – да	Базовый набор; IsAvailable = false	BusinessRuleException: Сектор недоступен для бронирования	BusinessRuleException: Сектор недоступен для бронирования (результат соответствует ожидаемому)

Продолжение таблицы 16

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
7	c – нет, e – нет, h – да	Базовый набор; MaxPeople = 2	BusinessRuleException: Количество человек превышает допустимую вместимость сектора	BusinessRuleException: Количество человек превышает допустимую вместимость сектора (результат соответствует ожидаемому)
8	h – нет, z – нет, ad – да, ah – нет, ap – нет	Базовый набор;	TotalPrice = 7100	TotalPrice = 7100 (результат соответствует ожидаемому)
9	h – нет, z – да, ad – да, ah – нет, ap – нет	Базовый набор; HasGazebo = true	TotalPrice = 7700	TotalPrice = 7700 (результат соответствует ожидаемому)
10	ah – да, ai – да	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество товара продажи должно быть больше нуля	ValidationException: Количество товара продажи должно быть больше нуля (результат соответствует ожидаемому)
11	ah – да, ai – нет, al – да	Базовый набор; SaleProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар продажи с id=999 не найден	NotFoundException: Товар продажи с id=999 не найден (результат соответствует ожидаемому)
12	ah – да, ai – нет, al – нет	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 7640	TotalPrice = 7640 (результат соответствует ожидаемому)
13	ap – да, aq – да	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество арендного товара должно быть больше нуля	ValidationException: Количество арендного товара должно быть больше нуля (результат соответствует ожидаемому)
14	ap – да, aq – нет, at – да	Базовый набор; RentProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар аренды с id=999 не найден	NotFoundException: Товар аренды с id=999 не найден (результат соответствует ожидаемому)
15	ap – да, aq – нет, at – нет	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 9800	TotalPrice = 9800 (результат соответствует ожидаемому)
16	a1 – нет, a2 – нет, a3.1 – нет, a3.2 – нет, c – нет, e – нет, h – нет, z – да, ad – да, ah – да, ai – нет, al – нет, ap – да, aq – нет, at – нет	Базовый набор; HasGazebo=true; SaleProducts = [{ProductId = 1; Quantity = 3}]; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 10940	TotalPrice = 10940 (результат соответствует ожидаемому)

Для достижения покрытия условий метода `CalculateAsync` были составлены тесты, в которых каждое простое условие принимает как истинное, так и ложное значение. Особое внимание было уделено проверке составного условия ограничения сезона бронирования, которое было разделено на два простых условия: `a3.1` и `a3.2`.

Все простые условия метода `CalculateAsync` были проверены в истинном и ложном значении. Полученные результаты подтверждают корректную работу условий, влияющих на расчет итоговой стоимости бронирования.

4.1.4 Покрытие решений и условий

В соответствии с критерием покрытия решений и условий необходимо составить такие тесты, при которых результаты каждого решения и каждого простого условия выполняются хотя бы один раз. Кроме того, каждый основной оператор программы должен быть выполнен не менее одного раза.

Критерий покрытия решений и условий объединяет в себе требования двух предыдущих критериев. Он позволяет проверить как истинные и ложные значения условий, так и различные ветви выполнения программы. Если после составления тестов остаются непокрытые операторы, набор тестов необходимо дополнить таким образом, чтобы каждый оператор выполнялся хотя бы один раз.

В методе `CalculateAsync` часть условий находится внутри метода проверки дат `ValidateDates`. Поэтому оператор «а» обозначает вызов проверки дат, а отдельные условия внутри данной проверки обозначены как «a1», «a2», «a3.1» и «a3.2».

Такое разделение необходимо для более точного анализа выполнения метода, так как проверка дат влияет на дальнейший ход расчёта. Если даты указаны некорректно, выполнение метода завершается до получения сектора и расчёта стоимости бронирования

Результаты тестирования представлены в таблице 17.

Таблица 17 – Тест-кейсы для покрытия решений и условий метода
CalculateAsync

№	Условие	Входные данные	Ожидаемый результат	Фактический результат
1	a, a1 (да)	StartDate = 2026-08-10; EndDate = 2026-08-10	ValidationException: Дата выезда должна быть позже даты заезда	ValidationException: Дата выезда должна быть позже даты заезда (результат соответствует ожидаемому)
2	a, a1 (нет), a2 (да)	StartDate = 2026-04-30; EndDate = 2026-08-10	ValidationException: Дата заезда не может быть в прошлом	ValidationException: Дата заезда не может быть в прошлом (результат соответствует ожидаемому)
3	a, a1 (нет), a2 (нет), a3.1 (да), a3.2 (нет)	StartDate = 2026-01-10; EndDate = 2026-08-10	ValidationException: Бронирование доступно только в установленный сезон	ValidationException: Бронирование доступно только в установленный сезон (результат соответствует ожидаемому)
4	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (да)	StartDate = 2026-08-10; EndDate = 2026-11-10	ValidationException: Бронирование доступно только в установленный сезон	ValidationException: Бронирование доступно только в установленный сезон (результат соответствует ожидаемому)
5	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (да), d	Базовый набор; SectorId = 999;	NotFoundException: Сектор не найден	NotFoundException: Сектор не найден (результат соответствует ожидаемому)
6	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (да), f	Базовый набор; IsAvailable = false	BusinessRuleException: Сектор недоступен для бронирования	BusinessRuleException: Сектор недоступен для бронирования (результат соответствует ожидаемому)

Продолжение таблицы 17

№	Условие	Входные данные	Ожидаемый результат	Фактический результат
7	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (да), i	Базовый набор; MaxPeople = 2	BusinessRuleException: Количество человек превышает допустимую вместимость сектора	BusinessRuleException: Количество человек превышает допустимую вместимость сектора (результат соответствует ожидаемому)
8	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (нет), aa, ab, ac, ad (да), ae, af, ag, ah (нет), ao, ap (нет), aw, ax	Базовый набор;	TotalPrice = 7100	TotalPrice = 7100 (результат соответствует ожидаемому)
9	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (да), aa, ab, ac, ad (да), ae, af, ag, ah (нет), ao, ap (нет), aw, ax	Базовый набор; HasGazebo = true	TotalPrice = 7700	TotalPrice = 7700 (результат соответствует ожидаемому)
10	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (нет), ac, ad (да), ae, af, ag, ah (да), ai (да), aj	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество товара продажи должно быть больше нуля	ValidationException: Количество товара продажи должно быть больше нуля (результат соответствует ожидаемому)
11	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (нет), ac, ad (да), ae, af, ag, ah (да), ai (нет), ak, al (да), am	Базовый набор; SaleProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар продажи с id=999 не найден	NotFoundException: Товар продажи с id=999 не найден (результат соответствует ожидаемому)
12	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (нет), ac, ad (да), ae, af, ag, ah (да), ai (нет), ak, al (нет), an, ao, ap (нет), aw, ax	Базовый набор; SaleProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 7640	TotalPrice = 7640 (результат соответствует ожидаемому)

Окончание таблицы 17

№	Условие	Входные данные	Ожидаемый результат	Фактический результат
13	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (нет), ac, ad (да), ae, af, ag, ah (нет), ao, ap (да), aq (да), ar	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 0}]	ValidationException: Количество арендного товара должно быть больше нуля	ValidationException: Количество арендного товара должно быть больше нуля (результат соответствует ожидаемому)
14	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (нет), ac, ad (да), ae, af, ag, ah (нет), ao, ap (да), aq (нет), as, at (да), au	Базовый набор; RentProducts = [{ProductId = 999; Quantity = 1}]	NotFoundException: Товар аренды с id=999 не найден	NotFoundException: Товар аренды с id=999 не найден (результат соответствует ожидаемому)
15	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (нет), ac, ad (да), ae, af, ag, ah (нет), ao, ap (да), aq (нет), as, at (нет), av, aw, ax	Базовый набор; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 9800	TotalPrice = 9800 (результат соответствует ожидаемому)
16	a, a1 (нет), a2 (нет), a3.1 (нет), a3.2 (нет), b, c (нет), e (нет), g, h (нет), j, k, l-r, s-w, x, y, z (да), aa, ab, ac, ad (да), ae, af, ag, ah (да), ai (нет), ak, al (нет), an, ao, ap (да), aq (нет), as, at (нет), av, aw, ax	Базовый набор; HasGazebo=true; SaleProducts = [{ProductId = 1; Quantity = 3}]; RentProducts = [{ProductId = 1; Quantity = 3}]	TotalPrice = 10940	TotalPrice = 10940 (результат соответствует ожидаемому)

Для достижения покрытия решений и условий метода CalculateAsync были составлены тесты, охватывающие основные маршруты выполнения метода. Все возможные комбинации результатов условий и решений были проверены. Все тесты пройдены успешно, что подтверждает корректную работу метода CalculateAsync при всех возможных сценариях выполнения.

В ходе функционального тестирования метода CalculateAsync методом белого ящика были последовательно применены четыре критерия: покрытие операторов, покрытие решений, покрытие условий и покрытие решений и условий. Для каждого критерия были разработаны соответствующие тест-кейсы, количество которых составило 13, 15, 16 и 16 соответственно. Все

разработанные тесты успешно пройдены, что подтверждает корректность реализации бизнес-логики расчета стоимости бронирования.

4.2 Модульные тесты серверной части программного обеспечения

Для проверки корректности работы метода `CalculateAsync` сервиса `PricingService` было выбрано Unit-тестирование. Модульное тестирование позволяет проверить отдельный программный модуль изолированно от остальных частей приложения. В рамках данной работы такой подход применяется для проверки бизнес-логики расчёта стоимости бронирования без необходимости запускать всё приложение целиком.

Каждый модульный тест был построен по шаблону AAA:

- Arrange – подготовка входных данных;
- Act – вызов тестируемого метода `CalculateAsync`;
- Assert – сравнение полученного результата с ожидаемым.

Для реализации модульных тестов использовался фреймворк `xUnit` [20]. Данный фреймворк позволяет создавать автоматизированные тесты для приложений, разработанных на платформе `.NET`.

Разработанные тесты охватывают основные сценарии использования метода `CalculateAsync`. Полный код модульных тестов приведён в приложении Г. Классификация тестов представлена в таблице 18.

Таблица 18 – Классификация Unit-тестов метода `CalculateAsync`

Категория сценариев	Количество тестов	Примеры
Проверка дат бронирования	3	Дата выезда совпадает с датой заезда; дата заезда находится в прошлом, даты находятся вне сезона
Проверка сектора	3	Сектор не найден; сектор недоступен; превышена вместимость сектора
Базовый расчёт стоимости	1	Расчёт стоимости без беседки и без дополнительных товаров
Расчёт с беседкой	1	В итоговую стоимость добавляется тариф за беседку
Расчёт товаров продажи	2	Успешный расчёт товара продажи; некорректное количество товара продажи
Расчёт товаров аренды	2	Успешный расчет товара аренды с учетом количества дней; некорректное количество товара аренды
Полный сценарий	1	Расчёт с разными категориями отдыхающих, беседкой, товарами продажи и аренды

Общее количество тестов: 13.

Результаты выполнения модульных тестов представлены на рисунке 16.

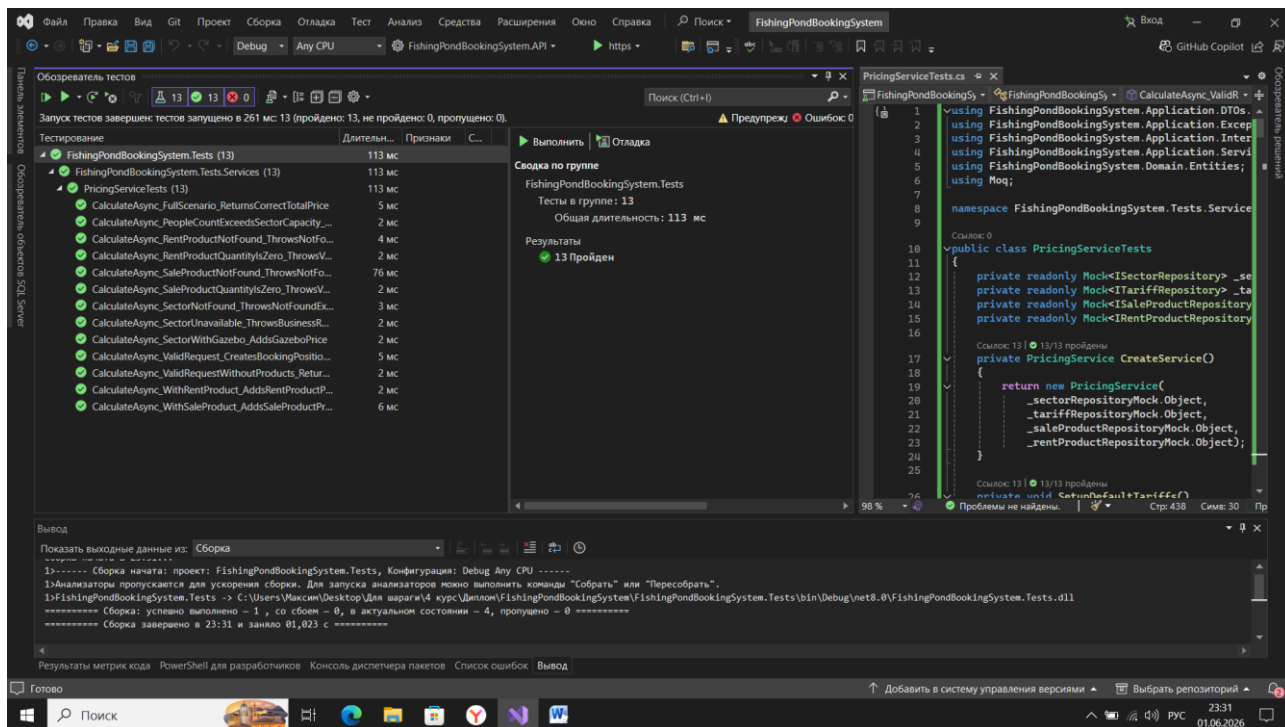


Рисунок 16 – Результат выполнения модульных тестов

Все разработанные Unit-тесты были успешно пройдены. Это подтверждает корректность работы метода `CalculateAsync` при основных вариантах использования. Тесты охватывают проверку дат, проверку сектора, расчёт стоимости по тарифам, обработку дополнительных товаров, учет беседки, а также граничные и ошибочные ситуации.

5 Оценка качества программного кода

5.1 Тестирование SLOC

Для оценки качества программного кода серверной части информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства были применены метрики программного обеспечения. Использование метрик позволяет количественно оценить такие характеристики программного кода, как объём, сложность, читаемость и пригодность к дальнейшему сопровождению.

В качестве объекта анализа был выбран метод `CalculateAsync` сервиса `PricingService`. К данному методу были применены следующие метрики: SLOC (анализ строк кода), метрики Холстеда [21], метрики Чепина, а также проведена автоматическая проверка с помощью инструмента `Lizard`.

Метрика SLOC используется для оценки объёма программного кода. Она позволяет определить количество физических строк кода, пустых строк, строк комментариев и значимых строк, непосредственно участвующих в выполнении программы.

При подсчёте 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 (считаются по количеству точек с запятой и управляющих конструкций «if», «for»).

Код метода `CalculateAsync` представлен в приложении В, листинг В.4.

Результаты анализа SLOC для метода `CalculateAsync`:

- LOC – 114;
- BLANK – 25;
- CLOC – 0;
- SLOC – $114 - 25 - 0 = 89$;
- LLOC – 50.

Несмотря на то, что физический размер файла составляет 114 строк, реальный исполняемый объём почти в два раза меньше – 50 логических инструкций. Высокое соотношение комментариев и пустых строк (почти 25% от всего объёма) говорит о хорошей читаемости и документированности кода.

5.2 Метрика Холстеда

Метрика Холстеда используется для количественной оценки сложности программного кода на основе количества операторов и операндов. Операторами считаются ключевые слова языка программирования, арифметические и логические операции, операторы присваивания, вызовы методов и служебные символы. Операндами являются переменные, свойства объектов, константы и значения, над которыми выполняются операции.

В первую очередь был произведён подсчёт операторов. Результат представлен в таблице 19.

Таблица 19 – Подсчёт операторов метода CalculateAsync

Оператор	Количество повторений
public	1
async	1
Task	1
return	1
if	9
foreach	2
throw	7
await	10
var	17
decimal	4
new	9
. (вызов метода)	45
=	31
+=	2
+	8
is	3

Продолжение таблицы 19

Оператор	Количество повторений
-	1
*	4
>	1
<=	2
!	1
=>	1
()	39
{}	15
;	39
,	27

Уникальных операторов (n1) – 26.

Всего операторов (N1) – 281.

Наиболее часто встречаются операторы обращений к свойствам и методам (.), оператор присваивания (=), круглые скобки, точки с запятой, а также ключевое слово var. Это связано с тем, что метод активно обращается к свойствам объекта запроса, получает данные из репозитория и формирует объект результата.

К операндам были отнесены переменные, свойства DTO-объектов, поля сущностей, константы и строковые сообщения исключений. Результаты подсчётов операндов представлены в таблице 20.

Таблица 20 – Подсчёт операндов метода CalculateAsync

Операнд	Количество повторений
request	18
days	10
position	10
item	10
product	6
sector	5
Quantity	4
ProductId	4
bookingFee	4
gazeboPrice	4
saleProductsPrice	4
rentProductsPrice	4
Price	3
accommodationPrice	3
gazeboTariff	3
null	3
FishermenCount	2
FemaleFishermenCount	2
MaleGuestsCount	2
sectorBookingTariff	3

Продолжение таблицы 20

Операнд	Количество повторений
FemaleGuestsCount	2
ChildrenCount	2
DayNumber	2
TotalPrice	2
fishermanTariff	2
femaleFishermanTariff	2
maleGuestTariff	2
femaleGuestTariff	2
childTariff	2
totalPeople	2
totalPrice	2
id	2
x	2
Операнды, встречающиеся один раз	27
Числовые константы 0 и 1	8
Строковые сообщения исключения	7

Уникальных операндов (n_2) – 69.

Всего операндов (N_2) – 172.

К операндам, встречающимся один раз, относятся отдельные свойства и переменные метода: SectorId, StartDate, EndDate, IsAvailable, RecommendedPeople, HasGazebo, IncludeBookingFee, SaleProducts, RentProducts, PricePerDay, Days, AccommodationPrice, GazeboPrice, BookingFee, SaleProductsPrice, RentProductsPrice, Positions, а также константы названий тарифов и поля репозитория.

Наиболее часто встречается операнд request, так как метод получает большинство входных данных из объекта запроса. Также часто используются операнды days, positions и item, что связано с расчётом количества дней, формированием тарифных позиций и обработкой списков товаров.

После подсчёта операторов и операндов производится расчёт основных метрик Холстеда. На основе полученных данных были получены следующие результаты:

- словарь программы (n): $n = n_1 + n_2 = 95$;
- длина программы (N): $N = N_1 + N_2 = 453$;
- объём программы (V): $V = N * \log_2(n) = 2976,14$;
- сложность программы (D): $D = (n_1/2) * (N_2/n_2) = 32,41$;

– усилие на понимание (E): $E = D * V = 96444,34$.

Значение словаря программы указывает на достаточно большое разнообразие используемых операторов и операндов. Это связано с тем, что метод активно использует свойства объекта запроса, переменные расчёта, тарифы, товары и DTO-объекты.

Длина программы показывает, что метод содержит значительное количество операций. Однако данный объём обусловлен не избыточностью кода, а количеством выполняемых проверок и расчётов, необходимых для корректного формирования стоимости бронирования.

Объём программы свидетельствует о том, что метод содержит достаточно насыщенную бизнес-логику. При этом код метода остаётся читаемым благодаря последовательной структуре и понятным названиям переменных.

Сложность программы является умеренно повышенной. Это связано с наличием условных операторов, циклов обработки товаров и обращений к репозиториям. Данный показатель показывает, что метод желательно поддерживать в хорошо структурированном виде.

Усилие на понимание отражает, что метод требует внимательного анализа при сопровождении, так как в нём объединены разные части расчёта стоимости. Однако сложность метода является оправданной, поскольку он выполняет ключевую бизнес-функцию серверной части информационной системы.

Расчёт метрик Холстеда показал, что метод CalculateAsync обладает умеренно повышенной сложностью, связанной с большим количеством проверок и расчётных операций. Полученные значения подтверждают, что метод является важным элементом бизнес-логики приложения.

5.3 Метрика Чепина

Метрика Чепина используется для оценки информационной сложности программного кода. В отличие от метрики Холстеда, которая основывается на подсчёте операторов и операндов, метрика Чепина анализирует переменные

программы и распределяет их по группам в зависимости от роли, которую они выполняют в алгоритме.

В метрике Чепина используются следующие группы переменных:

- Р – вводимые переменные, используемые для получения исходных данных;
- М – модифицируемые переменные, значения которых изменяются в ходе выполнения метода;
- С – управляющие переменные, влияющие на выбор ветвей выполнения программы;
- Т – транзитные переменные, которые используются как вспомогательные и не оказывают существенного влияния на логику программы.

Для анализа был выбран метод CalculateAsync сервиса PricingService. В данном методе используются переменные, связанные с запросом на расчёт стоимости, выбранным сектором, тарифами, товарами продажи, товарами аренды и итоговой стоимостью бронирования.

Распределение переменных по группам Чепина представлено в таблице 21.

Таблица 21 – Распределение переменных метода CalculateAsync по группам Чепина

Группа	Вес	Переменные	Количество
Р	$a_1 = 1$	request, sector, fishermanTariff, femaleFishermanTariff, maleGuestTariff, femaleGuestTariff, childTariff, gazeboTariff, sectorBookingTariff, days, item, product	12
М	$a_2 = 2$	totalPeople, positions, accommodationPrice, gazeboPrice, bookingFee, saleProductsPrice, rentProductsPrice, totalPrice	8
С	$a_3 = 3$	sector, sector.IsAvailable, totalPeople, sector.HasGazebo, request.IncludeBookingFee, item.Quantity, product	7
Т	$a_4 = 0,5$	x	1

Расчёт метрики Чепина выполняется по формуле:

$$Q = a_1 \times P + a_2 \times M + a_3 \times C + a_4 \times T.$$

Подставляем значения:

$$Q = (1 * 12) + (2 * 8) + (3 * 7) + (0,5 * 1) = 12 + 16 + 21 + 0,5 = 49,5.$$

Информационная сложность метода CalculateAsync составляет $Q = 49,5$.

Наибольший вклад в сложность метода вносят управляющие переменные группы С. Это объясняется тем, что метод содержит несколько проверок, от которых зависит дальнейшее выполнение алгоритма: проверка существования сектора, доступности сектора, вместимости, наличия беседки, необходимости включения фиксированного сбора, корректности количества товаров и существования товаров в базе данных.

Модифицируемые переменные группы М также оказывают значительное влияние на сложность метода. К ним относятся переменные, в которых накапливаются отдельные части итоговой стоимости: стоимость проживания, стоимость беседки, фиксированная стоимость бронирования, стоимость товаров продажи, стоимость товаров аренды и итоговая стоимость.

Вводимые переменные группы Р отражают исходные данные, необходимые для расчёта. К ним относятся объект запроса, выбранный сектор, тарифы, товары и количество дней бронирования. Данная группа имеет достаточно большое количество переменных, так как метод использует данные из нескольких источников: запроса пользователя, таблицы секторов, таблицы тарифов, таблицы товаров продажи и таблицы товаров аренды.

К транзитным переменным была отнесена переменная x , используемая в лямбда-выражении при подсчёте стоимости тарифных позиций. Она выполняет вспомогательную роль и не влияет напрямую на бизнес-логику метода.

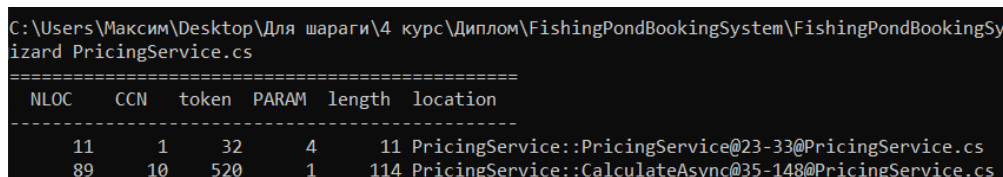
5.4 Автоматическая проверка

Автоматическая проверка программного кода проводилась с помощью инструмента Lizard. Данный инструмент предназначен для анализа сложности программного кода и позволяет определить такие показатели, как количество строк кода, цикломатическая сложность, количество токенов, количество параметров метода и физическая длина функции.

В рамках выпускной квалификационной работы автоматическая проверка была выполнена для файла PricingService.cs.

Данный файл был выбран, так как в нём расположен метод CalculateAsync, отвечающий за расчёт итоговой стоимости бронирования.

Результат работы инструмента Lizard для файла PricingService.cs представлен на рисунке 17.



The screenshot shows the command prompt path: C:\Users\Максим\Desktop\Для шараги\4 курс\Диплом\FishingPondBookingSystem\FishingPondBookingSystem\Lizard PricingService.cs. Below the path is a table with the following data:

NLOC	CCN	token	PARAM	length	location
11	1	32	4	11	PricingService::PricingService@23-33@PricingService.cs
89	10	520	1	114	PricingService::CalculateAsync@35-148@PricingService.cs

Рисунок 17 – Результат работы Lizard

Расшифровка результатов автоматической проверки:

– NLOC – количество строк кода без учета пустых строк и комментариев (для метода CalculateAsync данный показатель составил 89 строк);

– CCN – цикломатическая сложность метода (для метода CalculateAsync значение CCN составило 10);

– token – количество лексем, т. е. отдельных элементов программного кода (для метода CalculateAsync количество токенов составило 520);

– PARAM – количество параметров метода (метод CalculateAsync принимает 1 параметр – объект BookingCalculationRequestDto);

– length – физическая длина метода в строках (для метода CalculateAsync данный показатель составил 114 строк).

В ходе автоматической проверки было установлено, что метод CalculateAsync обладает умеренной цикломатической сложностью и не содержит критических признаков чрезмерной запутанности. Полученные результаты подтверждают выводы, сделанные при ручном анализе метрик SLOC, Холстеда и Чепина.

6 Планирование разработки и оценка бюджета

Планирование разработки является важным этапом создания программного обеспечения (ПО), так как позволяет определить последовательность выполнения работ, сроки реализации, необходимые ресурсы и ориентировочную стоимость проекта. В рамках выпускной квалификационной работы было выполнено планирование разработки бэкенд-компонента информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства.

Для планирования разработки использовалась программа ProjectLibre. Данное программное средство позволяет создавать 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, наглядно отражающая календарный план выполнения работ.

Разработка бэкенд-компонента была разделена на следующие основные этапы:

- анализ предметной области и требований;
- проектирование приложения;
- разработка программного обеспечения;
- тестирование программного обеспечения;
- контроль качества программного обеспечения;
- оформление пояснительной записки.

На этапе анализа предметной области и требований выполнялось исследование предметной области рыболовного хозяйства, анализ систем аналогичного назначения, а также формирование функциональных и нефункциональных требований к разрабатываемой системе.

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

проектирование программных средств, пользовательского интерфейса, а также выбор технологий и инструментальных средств разработки.

На этапе разработки программного обеспечения выполнялась разработка физической модели базы данных и реализация серверной части информационной системы.

Этап тестирования программного обеспечения включал функциональное тестирование методом белого ящика и разработку модульных тестов серверной части программного обеспечения.

Этап контроля качества программного обеспечения включал расчёт метрик качества программного кода и реализацию автоматической проверки. На данном этапе были рассчитаны метрики SLOC, Холстеда, Чепина, а также выполнена автоматическая проверка кода с помощью инструмента Lizard.

Заключительным этапом являлось оформление пояснительной записки. На данном этапе были подготовлены текстовые разделы работы, таблицы, рисунки, приложения и итоговое оформление документа.

Перечень задач разработки бэкенд-компонента представлен на рисунке 18.

	Название	Продолжительность	Начало	Окончание
1	Начало проекта	0 дней	07.04.2026, 08:00	07.04.2026, 08:00
2	Анализ предметной области и требований	6 дней	07.04.2026, 08:00	14.04.2026, 17:00
3	Исследование предметной области рыболовного хозяйства	1 день	07.04.2026, 08:00	07.04.2026, 17:00
4	Анализ систем аналогичного назначения	2 дней	08.04.2026, 08:00	09.04.2026, 17:00
5	Формирование функциональных требований	2 дней	10.04.2026, 08:00	13.04.2026, 17:00
6	Формирование нефункциональных требований	1 день	14.04.2026, 08:00	14.04.2026, 17:00
7	Проектирование приложения	9 дней	15.04.2026, 08:00	27.04.2026, 17:00
8	Проектирование архитектуры приложения	2 дней	15.04.2026, 08:00	16.04.2026, 17:00
9	Проектирование концептуальной модели базы данных	0,5 дней	17.04.2026, 08:00	17.04.2026, 13:00
10	Проектирование логической модели базы данных	0,5 дней	17.04.2026, 13:00	17.04.2026, 17:00
11	Проектирование программных средств	3 дней	20.04.2026, 08:00	22.04.2026, 17:00
12	Проектирование пользовательского интерфейса	2 дней	23.04.2026, 08:00	24.04.2026, 17:00
13	Выбор технологий и инструментальных средств разработки приложения	1 день	27.04.2026, 08:00	27.04.2026, 17:00
14	Разработка программного обеспечения	8 дней	28.04.2026, 08:00	07.05.2026, 17:00
15	Разработка физической модели базы данных	0,5 дней	28.04.2026, 08:00	28.04.2026, 13:00
16	Реализация серверной части информационной системы	7,5 дней	28.04.2026, 13:00	07.05.2026, 17:00
17	Тестирование программного обеспечения	5 дней	08.05.2026, 08:00	14.05.2026, 17:00
18	Функциональное тестирование методом белого ящика	3 дней	08.05.2026, 08:00	12.05.2026, 17:00
19	Модульные тесты серверной части ПО	2 дней	13.05.2026, 08:00	14.05.2026, 17:00
20	Контроль качества программного обеспечения	4 дней	15.05.2026, 08:00	20.05.2026, 17:00
21	Расчет метрик качества ПО	3 дней	15.05.2026, 08:00	19.05.2026, 17:00
22	Реализация автоматической проверки	1 день	20.05.2026, 08:00	20.05.2026, 17:00
23	Оформление пояснительной записки	9 дней	21.05.2026, 08:00	02.06.2026, 17:00
24	Конец проекта	0 дней	02.06.2026, 17:00	02.06.2026, 17:00

Рисунок 18 – Перечень задач разработки бэкенд-компонента

На рисунке 19 представлена диаграмма Ганта, которая показывает последовательность выполнения всех задач и продолжительность каждого этапа.

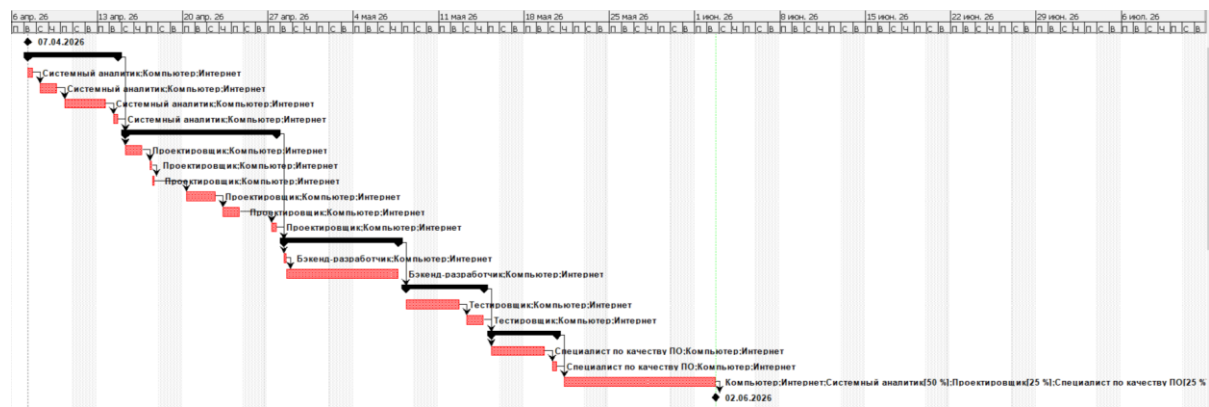


Рисунок 19 – Диаграмма Ганта

Общая продолжительность работы над проектом составила 41 рабочий день. Каждый этап, за исключением последнего, состоит из нескольких задач, выполнение которых необходимо для перехода к следующему этапу.

В проекте были выделены следующие ресурсы:

- системный аналитик;
- проектировщик;
- бэкенд-разработчик;
- тестировщик;
- специалист по качеству ПО;
- компьютер;
- интернет.

На рисунке 20 приведена стоимость рабочего времени задействованных ресурсов.

		Название	Тип	Стандартная ставка
1	🌸	Системный аналитик	Работа	400/час
2	🌸	Проектировщик	Работа	450/час
3	🌸	Бэкенд-разработчик	Работа	500/час
4	🌸	Тестировщик	Работа	350/час
5	🌸	Специалист по качеству ПО	Работа	400/час
6	🌸	Компьютер	Материал	0
7	🌸	Интернет	Материал	0

Рисунок 20 – Стоимость ресурсов

Общие затраты проекта с учетом продолжительности задач представлены на рисунке 21.

	Название	Стоимость
1	Начало проекта	0,00 Р
2	Анализ предметной области и требований	19200,00 Р
3	Исследование предметной области рыболовного хозяйства	3200,00 Р
4	Анализ систем аналогичного назначения	6400,00 Р
5	Формирование функциональных требований	6400,00 Р
6	Формирование нефункциональных требований	3200,00 Р
7	Проектирование приложения	32400,00 Р
8	Проектирование архитектуры приложения	7200,00 Р
9	Проектирование концептуальной модели базы данных	1800,00 Р
10	Проектирование логической модели базы данных	1800,00 Р
11	Проектирование программных средств	10800,00 Р
12	Проектирование пользовательского интерфейса	7200,00 Р
13	Выбор технологий и инструментальных средств разработки приложения	3600,00 Р
14	Разработка программного обеспечения	32000,00 Р
15	Разработка физической модели базы данных	2000,00 Р
16	Реализация серверной части информационной системы	30000,00 Р
17	Тестирование программного обеспечения	14000,00 Р
18	Функциональное тестирование методом белого ящика	8400,00 Р
19	Модульные тесты серверной части ПО	5600,00 Р
20	Контроль качества программного обеспечения	12800,00 Р
21	Расчет метрик качества ПО	9600,00 Р
22	Реализация автоматической проверки	3200,00 Р
23	Оформление пояснительной записки	13155,56 Р
24	Конец проекта	0,00 Р
		Итого: 123555,56

Рисунок 21 – Общие затраты проекта

Итоговая стоимость проекта составила 123 555,56 рублей. Распределение затрат по этапам выглядит следующим образом:

- анализ предметной области и требований – 19 200 рублей;
- проектирование приложения – 32 400 рублей;
- разработка программного обеспечения – 32 000 рублей;
- тестирование программного обеспечения – 14 000 рублей;
- контроль качества программного обеспечения – 12 800 рублей;
- оформление пояснительной записки – 13 155,56 рублей.

Проведённое планирование позволит заранее определить все основные этапы работы и обеспечит выполнение проекта в запланированные сроки.

Заключение

В рамках выполнения выпускной квалификационной работы была выполнена разработка бэкенд-компонента информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства.

На первом этапе был проведён анализ предметной области. Были рассмотрены особенности работы рыболовного хозяйства, процессы бронирования рыболовных секторов, учета клиентов, товаров продажи, товаров аренды, тарифов и уведомлений. Также были определены основные проблемы, связанные с ручной обработкой заявок, хранением данных и расчётом стоимости бронирования. Был проведён сравнительный анализ аналогичных решений. На основе проведённого анализа были сформулированы функциональные и нефункциональные требования к системе.

На этапе проектирования была выбрана многослойная архитектура приложения, позволяющая разделить серверную часть на отдельные логические уровни. Также были разработаны концептуальная и логическая модели базы данных, определены основные сущности предметной области, их атрибуты и связи между ними. Помимо этого, были спроектированы программные средства приложения и выбраны технологии и инструментальные средства разработки.

На третьем этапе была разработана физическая модель базы данных и реализована серверная часть информационной системы. Для реализации приложения использовался язык программирования C# и фреймворк ASP.NET Core. В качестве системы управления базами данных была выбрана PostgreSQL, а взаимодействие с базой данных осуществлялось с помощью Entity Framework Core. В приложении были реализованы доменные сущности, конфигурации, репозитории, сервисы бизнес-логики и API-контроллеры.

Разработанный бэкенд-компонент обеспечивает выполнение основных функций информационной системы: просмотр и фильтрацию рыболовных секторов, получение информации о секторах, тарифах и товарах, расчет стоимости отдыха, оформление бронирования, просмотр и отмену

бронирования, административное управление заявками, секторами, тарифами и товарами, а также формирование уведомлений и аналитических отчетов.

Для проверки корректности работы программного обеспечения были выполнены отладка и тестирование. В рамках функционального тестирования методом белого ящика были рассмотрены основные операторы, решения, условия, а также покрытие решений и условий метода расчета стоимости бронирования. Кроме того, были разработаны модульные тесты серверной части приложения. Результаты тестирования подтвердили корректность работы основных функций бэкенд-компонента и соответствие разработанного программного обеспечения заявленным требованиям.

На этапе контроля качества программного обеспечения были рассчитаны метрики программного кода. Полученные результаты позволили оценить объём, сложность и качество разработанного программного кода и показали, что программный код пригоден для сопровождения и дальнейшего развития.

Выполнено планирование и оценка стоимости разработки системы. Общая трудоёмкость разработки составила 41 рабочий день, а оценочная стоимость – 123 555,56 рублей.

Исходя из вышесказанного, можно сделать вывод, что цель выпускной квалификационной работы была достигнута. В результате был разработан бэкенд-компонент информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства. Все поставленные задачи были выполнены в полном объеме. Результаты выполненной работы внедрены в деятельность индивидуального предпринимателя Гришиной Светланы Николаевны, являющейся владельцем рыболовного пруда «Орловский», что подтверждается актом о внедрении, представленным в приложении Е.

Список использованных источников

1. Сайт рыболовного хозяйства Carp Fishing Belarus // carp-fishing.by [Электронный ресурс]. URL: <https://carp-fishing.by/> (дата обращения: 08.04.2026)
2. Сайт рыболовной тематики World-Fish // w-fish.ru [Электронный ресурс]. URL: <https://w-fish.ru/> (дата обращения: 08.04.2026)
3. Страница рыболовного хозяйства «Вороний пруд» // vk.com [Электронный ресурс]. URL: <https://vk.com/voroniy58> (дата обращения: 08.04.2026)
4. Фраулер, М. UML. Основы. / М. Фраулер. – 3-е изд. : пер. с англ. – СПб. : Символ-Плюс, 2004. – 192 с.
5. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/566218/> (дата обращения: 12.04.2026)
6. Что такое нефункциональные требования: примеры, определение, полное руководство // VISURESOLUTIONS [Электронный ресурс]. URL: <https://visuresolutions.com/ru/Блог/нефункциональные-требования/> (дата обращения: 12.04.2026)
7. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. – 3-е изд. – М. : Вильямс, 2003. – 1400 с.
8. ASP.NET Core // WORKSPACE [Электронный ресурс]. URL: <https://workspace.ru/tools/framework/asp-dot-net-core/> (дата обращения: 15.04.2026)
9. Основы кроссплатформенной верстки // Sky.pro [Электронный ресурс]. URL: <https://sky.pro/wiki/html/osnovy-krossplatforming-verstki/> (дата обращения: 20.04.2026)
10. Документация по языку C# // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 25.04.2026)

11. Fielding, R. RFC 9110: HTTP Semantics / R. Fielding, M. Nottingham, J. Reschke // RFC Editor [Электронный ресурс]. URL: <https://www.rfc-editor.org/info/rfc9110> (дата обращения: 25.04.2026)
12. Обзор Entity Framework Core // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/ef/core/> (дата обращения: 25.04.2026)
13. PostgreSQL Documentation // postgresql.org [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения: 25.04.2026)
14. Jones, M. RFC 7519: JSON Web Token (JWT) / M. Jones, J. Bradley, N. Sakimura // IETF Datatracker [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата обращения: 25.04.2026)
15. OpenAPI Specification // swagger.io [Электронный ресурс]. URL: <https://swagger.io/specification/> (дата обращения: 25.04.2026)
16. Visual Paradigm User's Guide // Visual Paradigm [Электронный ресурс]. URL: <https://www.visual-paradigm.com/support/documents/vpuserguide.jsp> (дата обращения: 25.04.2026)
17. ProjectLibre Documentation // ProjectLibre [Электронный ресурс]. URL: <https://projectlibre.com/projectlibre-documentation/> (дата обращения: 25.04.2026)
18. Физическая модель данных // AppMaster [Электронный ресурс]. URL: <https://appmaster.io/ru/glossary/fizicheskaiia-model-dannykh> (дата обращения: 27.04.2026)
19. Репозиторий кода FishingPondBookingSystem: серверная часть // GitHub [Электронный ресурс]. URL: <https://github.com/maksibon17/FishingPondBookingSystem/tree/main> (дата обращения: 27.04.2026)
20. xUnit.net Documentation // xunit.net [Электронный ресурс]. URL: <https://xunit.net/> (дата обращения: 18.05.2026)
21. Холстед, М. Наука о программировании / М. Холстед ; пер. с англ. – М. : Мир, 1981. – 270 с.

Приложение А
(справочное)
Глоссарий

Информационная система – программный комплекс, предназначенный для хранения, обработки, передачи и предоставления данных пользователям в удобном виде.

Backend – серверная часть приложения, которая отвечает за обработку запросов, выполнение основной логики, работу с базой данных и формирование ответа клиенту. Пользователь напрямую не видит backend, но именно он обеспечивает выполнение большинства функций системы.

API – набор правил, методов и форматов, с помощью которых разные части программной системы взаимодействуют между собой.

REST API – способ организации взаимодействия между клиентом и сервером через HTTP-запросы. REST API позволяет выполнять операции получения, добавления, изменения и удаления данных с помощью стандартных методов (GET, POST, PUT и DELETE).

HTTP – протокол передачи данных, который используется для обмена информацией между клиентом и сервером. С его помощью клиент отправляет запрос, а сервер возвращает ответ с необходимыми данными или результатом выполнения операции.

C# – объектно-ориентированный язык программирования, применяемый для разработки приложений на платформе .NET.

ASP.NET Core – фреймворк для разработки серверных web-приложений и API на платформе .NET. Он предоставляет средства маршрутизации запросов, внедрения зависимостей, обработки HTTP-запросов и настройки безопасности приложения.

Фреймворк – набор готовых инструментов, библиотек и правил, упрощающих разработку программного обеспечения.

База данных – организованное хранилище информации, предназначенное для сохранения, поиска, изменения и удаления данных.

СУБД – система управления базами данных, предназначенная для создания, настройки и обслуживания баз данных. СУБД обеспечивает хранение

данных, выполнение запросов, поддержку связей между таблицами и контроль целостности информации.

Entity Framework Core – технология объектно-реляционного отображения, позволяющая работать с базой данных через классы и объекты приложения.

DTO – объект передачи данных, используемый для обмена информацией между слоями приложения или между клиентом и сервером. Позволяет передавать только необходимые данные и не раскрывать внутреннюю структуру доменных сущностей.

Приложение Б
(рекомендуемое)
Атрибуты сущностей базы данных

Таблица Б.1 – Атрибуты сущности «Клиент»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор клиента	integer	10	от 1 до 2 147 483 467	нет	PK	1
Фамилия	Фамилия клиента	varchar	255	а-я А-Я -	да		Иванов
Имя	Имя клиента	varchar	255	а-я А-Я -	нет		Александр
Отчество	Отчество клиента	varchar	255	а-я А-Я -	да		Сергеевич
Номер телефона	Номер телефона клиента	varchar	12	строка, формат +7XXXXXXXXXX	нет	U	+79996120242

Таблица Б.2 – Атрибуты сущности «Администратор»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор администратора	integer	10	от 1 до 2 147 483 467	нет	PK	1
Логин	Логин администратора для входа в систему	varchar	255	а-z A-Z 0-9 . _ -	нет	U	admin
Пароль	Хешированный пароль администратора	varchar	255	а-z A-Z 0-9 . _ - ! & @ * +	нет		maxibon_17!

Таблица Б.3 – Атрибуты сущности «Тариф»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор тарифа	integer	10	от 1 до 2 147 483 467	нет	PK	1
Название тарифа	Название тарифа или категории услуги	varchar	255	а-я А-Я 0-9 , -	нет	U	Рыбак-мужчина
Описание	Описание тарифа	varchar	255	а-я А-Я 0-9 , -	нет		Стоимость суточной рыбалки для рыбака-мужчины
Стоимость	Стоимость тарифа	decimal	10,2	от 0 до 99999999,99	нет		800,00

Таблица Б.4 – Атрибуты сущности «Сторона водоёма»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор стороны водоёма	integer	10	от 1 до 2 147 483 467	нет	PK	1
Название стороны	Название стороны водоёма	varchar	255	а-я А-Я 0-9 , -	нет	U	Жёлтая сторона

Таблица Б.5 – Атрибуты сущности «Товар продажи»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор товара продажи	integer	10	от 1 до 2 147 483 467	нет	PK	1
Название товара	Название товара продажи	varchar	255	а-я А-Я -	нет	U	Кукуруза
Описание товара	Описание товара продажи	varchar	255	а-я А-Я -	нет		Насадка для рыбалки
Стоимость товара	Стоимость единицы товара продажи	varchar	10,2	от 0 до 99999999,99	нет		180,00
Количество в наличии	Количество товара продажи в наличии	varchar	10	от 0 до 2 147 483 467	нет	U	50
Фотографии товара	Ссылка на фотографию товара продажи	varchar	255	а-z А-Z 0-9 . _ - /	нет		corn.jpg

Таблица Б.6 – Атрибуты сущности «Товар аренды»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор товара аренды	integer	10	от 1 до 2 147 483 467	нет	PK	1
Название товара	Название товара аренды	varchar	255	а-я А-Я -	нет	U	Удочка
Описание товара	Описание товара аренды	varchar	255	а-я А-Я -	нет		Поплавочная удочка
Стоимость аренды за сутки	Стоимость аренды товара за сутки	varchar	10,2	от 0 до 99999999,99	нет		300,00

Продолжение таблицы Б.6

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
Количество в наличии	Количество товара аренды в наличии	varchar	10	от 0 до 2 147 483 467	нет	U	20
Фотографии товара	Ссылка на фотографии товара аренды	varchar	255	a-z A-Z 0-9 . _ - /	нет		rod.jpg

Таблица Б.7 – Атрибуты сущности «Сектор»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор сектора	integer	10	от 1 до 2 147 483 467	нет	PK	1
Номер сектора	Номер рыболовного сектора	integer	10	a-я A-Я -	нет		17
Количество подходов к воде	Количество подходов к воде в секторе	integer	10	a-я A-Я -	нет		2
Максимально разрешенное количество человек	Максимальное разрешенное количество отдыхающих на секторе	integer	10	от 1 до 2 147 483 467	нет		5
Средняя глубина	Средняя глубина водоёма в зоне сектора, м	numeric	5,2	от 0 до 999,99	нет	U	2,50
Наличие беседки	Признак наличия беседки в секторе	bit	1	0 – нет, 1 – да	нет		1
Доступность сектора	Признак доступности сектора для бронирования	bit	1	0 – нет, 1 – да	нет		1
Фотографии сектора	Ссылка на фотографии сектора	varchar	255	a-z A-Z 0-9 . _ - /	нет		sector6.jpg
Сторона водоёмаID	Идентификатор стороны водоёма	integer	10	от 1 до 2 147 483 467	нет	FK	1

Таблица Б.8 – Атрибуты сущности «Бронирование»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор бронирования	integer	10	от 1 до 2 147 483 467	нет	PK	1
Код бронирования	Уникальный код бронирования клиента	varchar	6	a-z A-Z 0-9	нет	U	A7K92B
Дата заезда	Дата начала бронирования	date		20.04.2026 – 31.12.2100	нет		10.08.2026
Дата выезда	Дата окончания бронирования	date		20.04.2026 – 31.12.2100	нет		13.08.2026
Время прибытия	Планируемое время прибытия клиента	time	7	00:00:00 – 23:59:59	нет	U	08:00:00
Итоговая стоимость	Итоговая стоимость бронирования	decimal	10,2	от 0 до 99999999,99	нет		7400,00
Статус бронирования	Текущий статус бронирования	integer	10	0-9	нет		1
Дата создания	Дата и время создания бронирования	timestamp		20.04.2026 00:00 – 31.12.2100 23:59	нет		10.08.2026 12:30
Дата подтверждения	Дата и время подтверждения бронирования	timestamp		20.04.2026.00:00:00 – 31.12.2100.23:59:59	да		10.08.2026 13:00
СекторID	Идентификатор выбранного сектора	integer	10	от 1 до 2 147 483 467	нет	FK	6
КлиентID	Идентификатор клиента, оформившего бронирование	integer	10	от 1 до 2 147 483 467	нет	FK	1
АдминистраторID	Идентификатор администратора, изменившего статус бронирования	integer	10	от 1 до 2 147 483 467	нет	FK	1

Таблица Б.9 – Атрибуты сущности «Позиция бронирования»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор позиции бронирования	integer	10	от 1 до 2 147 483 467	нет	PK	1
Количество	Количество единиц выбранного тарифа	integer	10	от 1 до 2 147 483 467	нет		2
БронированиеID	Идентификатор связанного бронирования	integer	10	от 1 до 2 147 483 467	нет	FK	1
ТарифID	Идентификатор выбранного тарифа	integer	10	от 1 до 2 147 483 467	нет	FK	1

Таблица Б.10 – Атрибуты сущности «Позиция товара продажи»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор позиции товара продажи	integer	10	от 1 до 2 147 483 467	нет	PK	1
Количество товара	Количество выбранного товара продажи	integer	10	от 1 до 2 147 483 467	нет		3
Стоимость на момент продажи	Стоимость товара продажи на момент оформления бронирования	decimal	10,2	от 0 до 99999999,99	нет		180,00
БронированиеID	Идентификатор связанного бронирования	integer	10	от 1 до 2 147 483 467	нет	FK	1
Товар продажиID	Идентификатор товара продажи	integer	255	от 1 до 2 147 483 467	нет	FK	1

Таблица Б.11 – Атрибуты сущности «Позиция товара аренды»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор позиции товара аренды	integer	10	от 1 до 2 147 483 467	нет	PK	1
Количество товара	Количество выбранного товара аренды	integer	10	от 1 до 2 147 483 467	нет		3
Количество дней аренды	Количество дней аренды товара	integer	10	от 1 до 2 147 483 467	нет		3
Стоимость на момент аренды	Стоимость товара аренды на момент оформления бронирования	decimal	10,2	от 0 до 99999999,99	нет		300,00
БронированиеID	Идентификатор связанного бронирования	integer	10	от 1 до 2 147 483 467	нет	FK	1
Товар арендыID	Идентификатор товара аренды	integer	255	от 1 до 2 147 483 467	нет	FK	1

Таблица Б.12 – Атрибуты сущности «Уведомление»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Null	Роль	Пример
ID	Уникальный идентификатор уведомления	integer	10	от 1 до 2 147 483 467	нет	PK	1
Сообщение уведомления	Текст уведомления, связанного с бронированием	varchar	255	а-я А-Я 0-9 . , -	нет	U	Бронирование подтверждено
Дата создания	Дата и время создания уведомления	timestamp		20.04.2026 00:00-31.12.2100 23:59	нет		01.08.2026 13:00
Подтверждение отправки	Признак подтверждения отправки уведомления	bit	1	0 – нет, 1 – да	нет		1
БронированиеID	Идентификатор связанного бронирования	integer	10	от 1 до 2 147 483 467	нет	FK	1

Приложение В
(обязательное)
Листинг серверной части системы

Листинг В.1 – BookingsController.cs

```

using FishingPondBookingSystem.Application.DTOs.BookingDtos;
using FishingPondBookingSystem.Application.Interfaces.Services;
using Microsoft.AspNetCore.Mvc;

namespace FishingPondBookingSystem.API.Controllers;

[ApiController]
[Route("bookings")]
public class BookingsController : ControllerBase
{
    private readonly IPricingService _pricingService;
    private readonly IBookingService _bookingService;

    public BookingsController(
        IPricingService pricingService,
        IBookingService bookingService)
    {
        _pricingService = pricingService;
        _bookingService = bookingService;
    }

    [HttpPost("calculate")]
    public async Task<ActionResult<BookingCalculationResultDto>>
    Calculate(
        [FromBody] BookingCalculationRequestDto request)
    {
        var result = await
        _pricingService.CalculateAsync(request);
        return Ok(result);
    }

    [HttpPost]
    public async Task<ActionResult<BookingResponseDto>> Create(
        [FromBody] CreateBookingDto request)
    {
        var result = await _bookingService.CreateAsync(request);

        return CreatedAtAction(
            nameof(GetById),
            new { id = result.Id, phone = result.ClientPhone, code
= result.CancellationCode },
            result);
    }

    [HttpGet]
    public async
    Task<ActionResult<IEnumerable<BookingResponseDto>>>
    GetByPhoneAndCode(
        [FromQuery] string phone,
        [FromQuery] string code)

```

```

    {
        var result = await
_bookingService.GetByPhoneAndCodeAsync(phone, code);
        return Ok(result);
    }

    [HttpGet("{id:int}")]
    public async Task<ActionResult<BookingDetailsDto>> GetById(
        int id,
        [FromQuery] string phone,
        [FromQuery] string code)
    {
        var result = await _bookingService.GetByIdAsync(id, phone,
code);
        return Ok(result);
    }

    [HttpDelete("{id:int}")]
    public async Task<ActionResult> Cancel(
        int id,
        [FromQuery] string phone,
        [FromQuery] string code)
    {
        await _bookingService.CancelAsync(id, phone, code);
        return NoContent();
    }
}

```

Листинг В.2 – SectorsController.cs

```

using FishingPondBookingSystem.Application.DTOs.SectorDtos;
using FishingPondBookingSystem.Application.Interfaces.Services;
using Microsoft.AspNetCore.Mvc;

namespace FishingPondBookingSystem.API.Controllers;

[ApiController]
[Route("sectors")]
public class SectorsController : ControllerBase
{
    private readonly ISectorService _sectorService;

    public SectorsController(ISectorService sectorService)
    {
        _sectorService = sectorService;
    }

    [HttpGet]
    public async
Task<ActionResult<IEnumerable<SectorResponseDto>>> GetAll(
        [FromQuery] SectorFilterDto filter)
    {
        var sectors = await _sectorService.GetAllAsync(filter);
    }
}

```

```

        return Ok(sectors);
    }

    [HttpGet("{id:int}")]
    public async Task<ActionResult<SectorDetailsDto>> GetById(int
id)
    {
        var sector = await _sectorService.GetByIdAsync(id);
        return Ok(sector);
    }
}

```

Листинг В.3 – PricingService.cs

```

using FishingPondBookingSystem.Application.DTOs.BookingDtos;
using FishingPondBookingSystem.Application.Exceptions;
using
FishingPondBookingSystem.Application.Interfaces.Repositories;
using FishingPondBookingSystem.Application.Interfaces.Services;

namespace FishingPondBookingSystem.Application.Services;

public class PricingService : IPricingService
{
    private const string FishermanTariffName = "Рыбак";
    private const string FemaleFishermanTariffName = "Женщина-
рыбачка";
    private const string MaleGuestTariffName = "Мужчина-гость";
    private const string FemaleGuestTariffName = "Женщина-гость";
    private const string ChildTariffName = "Ребёнок";
    private const string GazeboTariffName = "Беседка";
    private const string SectorBookingTariffName = "Бронирование
сектора";

    private readonly ISectorRepository _sectorRepository;
    private readonly ITariffRepository _tariffRepository;
    private readonly ISaleProductRepository
_saleProductRepository;
    private readonly IRentProductRepository
_rentProductRepository;

    public PricingService(
        ISectorRepository sectorRepository,
        ITariffRepository tariffRepository,
        ISaleProductRepository saleProductRepository,
        IRentProductRepository rentProductRepository)
    {
        _sectorRepository = sectorRepository;
        _tariffRepository = tariffRepository;
        _saleProductRepository = saleProductRepository;
        _rentProductRepository = rentProductRepository;
    }
}

```

```

    public async Task<BookingCalculationResultDto>
    CalculateAsync(BookingCalculationRequestDto request)
    {
        ValidateDates(request);

        var sector = await
        _sectorRepository.GetByIdAsync(request.SectorId);

        if (sector is null)
        {
            throw new NotFoundException("Сектор не найден.");
        }

        if (!sector.IsAvailable)
        {
            throw new BusinessException("Сектор недоступен для
            бронирования.");
        }

        var totalPeople = request.FishermenCount +
            request.FemaleFishermenCount +
            request.MaleGuestsCount +
            request.FemaleGuestsCount +
            request.ChildrenCount;

        if (totalPeople > sector.MaxPeople)
        {
            throw new BusinessException("Количество человек
            превышает допустимую вместимость сектора.");
        }

        var days = request.EndDate.DayNumber -
        request.StartDate.DayNumber;

        var positions = new List<BookingCalculationPositionDto>();

        var fishermanTariff = await
        GetTariffAsync(FishermanTariffName);
        var femaleFishermanTariff = await
        GetTariffAsync(FemaleFishermanTariffName);
        var maleGuestTariff = await
        GetTariffAsync(MaleGuestTariffName);
        var femaleGuestTariff = await
        GetTariffAsync(FemaleGuestTariffName);
        var childTariff = await GetTariffAsync(ChildTariffName);
        var gazeboTariff = await GetTariffAsync(GazeboTariffName);
        var sectorBookingTariff = await
        GetTariffAsync(SectorBookingTariffName);

        AddDailyPosition(positions, fishermanTariff,
        request.FishermenCount, days);
    }

```

```

        AddDailyPosition(positions, femaleFishermanTariff,
request.FemaleFishermenCount, days);
        AddDailyPosition(positions, maleGuestTariff,
request.MaleGuestsCount, days);
        AddDailyPosition(positions, femaleGuestTariff,
request.FemaleGuestsCount, days);
        AddDailyPosition(positions, childTariff,
request.ChildrenCount, days);

        var accommodationPrice = positions.Sum(x => x.TotalPrice);

        decimal gazeboPrice = 0;

        if (sector.HasGazebo)
        {
            AddDailyPosition(positions, gazeboTariff, 1, days);
            gazeboPrice = gazeboTariff.Price * days;
        }

        decimal bookingFee = 0;

        if (request.IncludeBookingFee)
        {
            AddFixedPosition(positions, sectorBookingTariff, 1);
            bookingFee = sectorBookingTariff.Price;
        }
        decimal saleProductsPrice = 0;

        foreach (var item in request.SaleProducts)
        {
            if (item.Quantity <= 0)
            {
                throw new ValidationException("Количество товара
продажи должно быть больше нуля.");
            }

            var product = await
_saleProductRepository.GetByIdAsync(item.ProductId);

            if (product is null)
            {
                throw new NotFoundException($"Товар продажи с
id={item.ProductId} не найден.");
            }

            saleProductsPrice += product.Price * item.Quantity;
        }

        decimal rentProductsPrice = 0;

        foreach (var item in request.RentProducts)
        {

```

```

        if (item.Quantity <= 0)
        {
            throw new ValidationException("Количество
арендного товара должно быть больше нуля.");
        }

        var product = await
_rentProductRepository.GetByIdAsync(item.ProductId);

        if (product is null)
        {
            throw new NotFoundException($"Товар аренды с
id={item.ProductId} не найден.");
        }

        rentProductsPrice += product.PricePerDay * days *
item.Quantity;
    }

    var totalPrice = accommodationPrice + gazeboPrice +
bookingFee + saleProductsPrice + rentProductsPrice;

    return new BookingCalculationResultDto
    {
        Days = days,
        AccommodationPrice = accommodationPrice,
        GazeboPrice = gazeboPrice,
        BookingFee = bookingFee,
        SaleProductsPrice = saleProductsPrice,
        RentProductsPrice = rentProductsPrice,
        TotalPrice = totalPrice,
        Positions = positions
    };
}

private async Task<Domain.Entities.Tariff>
GetTariffAsync(string name)
{
    var tariff = await _tariffRepository.GetByNameAsync(name);

    if (tariff is null)
    {
        throw new NotFoundException($"Тариф \"{name}\" не
найден.");
    }

    return tariff;
}

private static void AddDailyPosition(
    List<BookingCalculationPositionDto> positions,
    Domain.Entities.Tariff tariff,

```

```

        int quantity,
        int days)
    {
        if (quantity <= 0)
        {
            return;
        }

        positions.Add(new BookingCalculationPositionDto
        {
            TariffId = tariff.Id,
            TariffName = tariff.Name,
            Quantity = quantity,
            UnitPrice = tariff.Price,
            TotalPrice = tariff.Price * quantity * days
        });
    }

    private static void AddFixedPosition(
        List<BookingCalculationPositionDto> positions,
        Domain.Entities.Tariff tariff,
        int quantity)
    {
        if (quantity <= 0)
        {
            return;
        }

        positions.Add(new BookingCalculationPositionDto
        {
            TariffId = tariff.Id,
            TariffName = tariff.Name,
            Quantity = quantity,
            UnitPrice = tariff.Price,
            TotalPrice = tariff.Price * quantity
        });
    }

    private static void ValidateDates(BookingCalculationRequestDto
request)
    {
        if (request.StartDate >= request.EndDate)
        {
            throw new ValidationException("Дата выезда должна быть
позже даты заезда.");
        }

        if (request.StartDate <
DateOnly.FromDateTime(DateTime.Today))
        {
            throw new ValidationException("Дата заезда не может
быть в прошлом.");
        }
    }

```

```

        ValidateBookingSeason(request.StartDate, request.EndDate);
    }

    private static void ValidateBookingSeason(DateOnly startDate,
        DateOnly endDate)
    {
        var currentYear = DateTime.Today.Year;

        var seasonStart = new DateOnly(currentYear, 4, 20);
        var seasonEnd = new DateOnly(currentYear, 9, 20);

        if (startDate < seasonStart || endDate > seasonEnd)
        {
            throw new ValidationException(
                $"Бронирование доступно только в сезон: с
{seasonStart:dd.MM.yyyy} по {seasonEnd:dd.MM.yyyy}.");
        }
    }
}

```

Листинг В.4 – Метод CalculateAsync:

```

    public async Task<BookingCalculationResultDto>
    CalculateAsync(BookingCalculationRequestDto request)
    {
        ValidateDates(request); // a
        var sector = await
        _sectorRepository.GetByIdAsync(request.SectorId); // b
        if (sector is null) // c
        {
            throw new NotFoundException("Сектор не найден."); //d
        }

        if (!sector.IsAvailable) // e
        {
            throw new BusinessException("Сектор недоступен для
бронирования."); // f
        }
        var totalPeople = request.FishermenCount +
            request.FemaleFishermenCount +
            request.MaleGuestsCount +
            request.FemaleGuestsCount +
            request.ChildrenCount; //g

        if (totalPeople > sector.RecommendedPeople) //h
        {
            throw new BusinessException("Количество человек
превышает допустимую вместимость сектора."); //i
        }

        var days = request.EndDate.DayNumber -
        request.StartDate.DayNumber; //j
    }
}

```

```

        var positions = new List<BookingCalculationPositionDto>();
// k

        var fishermanTariff = await
GetTariffAsync(FishermanTariffName); // l
        var femaleFishermanTariff = await
GetTariffAsync(FemaleFishermanTariffName); // m
        var maleGuestTariff = await
GetTariffAsync(MaleGuestTariffName); //n
        var femaleGuestTariff = await
GetTariffAsync(FemaleGuestTariffName); //o
        var childTariff = await GetTariffAsync(ChildTariffName);
// p
        var gazeboTariff = await GetTariffAsync(GazeboTariffName);
// q
        var sectorBookingTariff = await
GetTariffAsync(SectorBookingTariffName); // r

        AddDailyPosition(positions, fishermanTariff,
request.FishermenCount, days); // s
        AddDailyPosition(positions, femaleFishermanTariff,
request.FemaleFishermenCount, days); // t
        AddDailyPosition(positions, maleGuestTariff,
request.MaleGuestsCount, days); // u
        AddDailyPosition(positions, femaleGuestTariff,
request.FemaleGuestsCount, days); // v
        AddDailyPosition(positions, childTariff,
request.ChildrenCount, days); // w

        var accommodationPrice = positions.Sum(x => x.TotalPrice);
// x

        decimal gazeboPrice = 0; // y

        if (sector.HasGazebo) // z
        {
            AddDailyPosition(positions, gazeboTariff, 1, days); //
aa
            gazeboPrice = gazeboTariff.Price * days; // ab
        }

        decimal bookingFee = 0; // ac

        if (request.IncludeBookingFee) // ad
        {
            AddFixedPosition(positions, sectorBookingTariff, 1);
// ae
            bookingFee = sectorBookingTariff.Price; // af
        }
        decimal saleProductsPrice = 0; // ag

```

```

        foreach (var item in request.SaleProducts) // ah
        {
            if (item.Quantity <= 0) // ai
            {
                throw new ValidationException("Количество товара
продажи должно быть больше нуля."); // aj
            }

            var product = await
_saleProductRepository.GetByIdAsync(item.ProductId); // ak

            if (product is null) // al
            {
                throw new NotFoundException($"Товар продажи с
id={item.ProductId} не найден."); // am
            }

            saleProductsPrice += product.Price * item.Quantity; //
an
        }

        decimal rentProductsPrice = 0; // ao

        foreach (var item in request.RentProducts) // ap
        {
            if (item.Quantity <= 0) // aq
            {
                throw new ValidationException("Количество
арендного товара должно быть больше нуля."); // ar
            }

            var product = await
_rentProductRepository.GetByIdAsync(item.ProductId); // as

            if (product is null) // at
            {
                throw new NotFoundException($"Товар аренды с
id={item.ProductId} не найден."); // au
            }

            rentProductsPrice += product.PricePerDay * days *
item.Quantity; // av
        }

        var totalPrice = accommodationPrice + gazeboPrice +
bookingFee + saleProductsPrice + rentProductsPrice; // aw

        return new BookingCalculationResultDto // ax
        {
            Days = days,
            AccommodationPrice = accommodationPrice,
            GazeboPrice = gazeboPrice,

```

```
        BookingFee = bookingFee,  
        SaleProductsPrice = saleProductsPrice,  
        RentProductsPrice = rentProductsPrice,  
        TotalPrice = totalPrice,  
        Positions = positions  
    };  
}
```

Приложение Г
(рекомендуемое)
Листинг модульных тестов

```

using FishingPondBookingSystem.Application.DTOS.BookingDtos;
using FishingPondBookingSystem.Application.Exceptions;
using
FishingPondBookingSystem.Application.Interfaces.Repositories;
using FishingPondBookingSystem.Application.Services;
using FishingPondBookingSystem.Domain.Entities;
using Moq;

namespace FishingPondBookingSystem.Tests.Services;

public class PricingServiceTests
{
    private readonly Mock<ISectorRepository> _sectorRepositoryMock
= new();
    private readonly Mock<ITariffRepository> _tariffRepositoryMock
= new();
    private readonly Mock<ISaleProductRepository>
_saleProductRepositoryMock = new();
    private readonly Mock<IRentProductRepository>
_rentProductRepositoryMock = new();

    private PricingService CreateService()
    {
        return new PricingService(
            _sectorRepositoryMock.Object,
            _tariffRepositoryMock.Object,
            _saleProductRepositoryMock.Object,
            _rentProductRepositoryMock.Object);
    }

    private void SetupDefaultTariffs()
    {
        SetupTariff("Рыбак", 800m);
        SetupTariff("Женщина-рыбачка", 500m);
        SetupTariff("Мужчина-гость", 200m);
        SetupTariff("Женщина-гость", 0m);
        SetupTariff("Ребёнок", 0m);
        SetupTariff("Беседка", 200m);
        SetupTariff("Бронирование сектора", 200m);
    }

    private void SetupTariff(string name, decimal price)
    {
        _tariffRepositoryMock
            .Setup(x => x.GetByNameAsync(name))
            .ReturnsAsync(new Tariff
            {
                Id = Math.Abs(name.GetHashCode()),
                Name = name,
                Description = $"Тариф: {name}",
                Price = price
            });
    }
}

```

```

    }

    private static BookingCalculationRequestDto
    CreateValidRequest()
    {
        return new BookingCalculationRequestDto
        {
            SectorId = 1,
            StartDate = new DateOnly(2026, 8, 10),
            EndDate = new DateOnly(2026, 8, 13),
            FishermenCount = 2,
            FemaleFishermenCount = 1,
            MaleGuestsCount = 1,
            FemaleGuestsCount = 1,
            ChildrenCount = 1,
            IncludeBookingFee = true,
            SaleProducts = new List<BookingSaleProductItemDto>(),
            RentProducts = new List<BookingRentProductItemDto>()
        };
    }

    private void SetupAvailableSector(bool hasGazebo = false, int
    maxPeople = 10)
    {
        _sectorRepositoryMock
            .Setup(x => x.GetByIdAsync(1))
            .ReturnsAsync(new Sector
            {
                Id = 1,
                Number = 1,
                MaxPeople = maxPeople,
                IsAvailable = true,
                HasGazebo = hasGazebo
            });
    }

    [Fact]
    public async Task
    CalculateAsync_ValidRequestWithoutProducts_ReturnsCorrectTotalPrice()
    {
        // Arrange
        SetupDefaultTariffs();
        SetupAvailableSector(hasGazebo: false);

        var service = CreateService();
        var request = CreateValidRequest();

        // Act
        var result = await service.CalculateAsync(request);

        // Assert
    }

```

```

        Assert.Equal(3, result.Days);
        Assert.Equal(6900m, result.AccommodationPrice);
        Assert.Equal(0m, result.GazeboPrice);
        Assert.Equal(200m, result.BookingFee);
        Assert.Equal(0m, result.SaleProductsPrice);
        Assert.Equal(0m, result.RentProductsPrice);
        Assert.Equal(7100m, result.TotalPrice);
    }

    [Fact]
    public async Task
    CalculateAsync_SectorWithGazebo_AddsGazeboPrice()
    {
        // Arrange
        SetupDefaultTariffs();
        SetupAvailableSector(hasGazebo: true);

        var service = CreateService();
        var request = CreateValidRequest();

        // Act
        var result = await service.CalculateAsync(request);

        // Assert
        Assert.Equal(3, result.Days);
        Assert.Equal(6900m, result.AccommodationPrice);
        Assert.Equal(600m, result.GazeboPrice);
        Assert.Equal(200m, result.BookingFee);
        Assert.Equal(7700m, result.TotalPrice);
    }

    [Fact]
    public async Task
    CalculateAsync_WithSaleProduct_AddsSaleProductPrice()
    {
        // Arrange
        SetupDefaultTariffs();
        SetupAvailableSector(hasGazebo: false);

        _saleProductRepositoryMock
            .Setup(x => x.GetByIdAsync(1))
            .ReturnsAsync(new SaleProduct
            {
                Id = 1,
                Name = "Кукуруза рыболовная",
                Description = "Насадка для рыбалки",
                Price = 180m,
                StockQuantity = 10
            });

        var service = CreateService();

```

```

var request = CreateValidRequest();
request.SaleProducts.Add(new BookingSaleProductItemDto
{
    ProductId = 1,
    Quantity = 3
});

// Act
var result = await service.CalculateAsync(request);

// Assert
Assert.Equal(540m, result.SaleProductsPrice);
Assert.Equal(7640m, result.TotalPrice);
}

[Fact]
public async Task
CalculateAsync_WithRentProduct_AddsRentProductPrice()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: false);

    _rentProductRepositoryMock
        .Setup(x => x.GetByIdAsync(1))
        .ReturnsAsync(new RentProduct
        {
            Id = 1,
            Name = "Удочка поплавочная",
            Description = "Удочка для аренды",
            PricePerDay = 300m,
            StockQuantity = 10
        });

    var service = CreateService();

    var request = CreateValidRequest();
    request.RentProducts.Add(new BookingRentProductItemDto
    {
        ProductId = 1,
        Quantity = 3
    });

    // Act
    var result = await service.CalculateAsync(request);

    // Assert
    Assert.Equal(2700m, result.RentProductsPrice);
    Assert.Equal(9800m, result.TotalPrice);
}

[Fact]

```

```

public async Task
CalculateAsync_FullScenario_ReturnsCorrectTotalPrice()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: true);

    _saleProductRepositoryMock
        .Setup(x => x.GetByIdAsync(1))
        .ReturnsAsync(new SaleProduct
        {
            Id = 1,
            Name = "Кукуруза рыболовная",
            Description = "Насадка для рыбалки",
            Price = 180m,
            StockQuantity = 10
        });

    _rentProductRepositoryMock
        .Setup(x => x.GetByIdAsync(1))
        .ReturnsAsync(new RentProduct
        {
            Id = 1,
            Name = "Удочка поплавочная",
            Description = "Удочка для аренды",
            PricePerDay = 300m,
            StockQuantity = 10
        });

    var service = CreateService();

    var request = CreateValidRequest();

    request.SaleProducts.Add(new BookingSaleProductItemDto
    {
        ProductId = 1,
        Quantity = 3
    });

    request.RentProducts.Add(new BookingRentProductItemDto
    {
        ProductId = 1,
        Quantity = 3
    });

    // Act
    var result = await service.CalculateAsync(request);

    // Assert
    Assert.Equal(3, result.Days);
    Assert.Equal(6900m, result.AccommodationPrice);
    Assert.Equal(600m, result.GazeboPrice);
}

```

```

        Assert.Equal(200m, result.BookingFee);
        Assert.Equal(540m, result.SaleProductsPrice);
        Assert.Equal(2700m, result.RentProductsPrice);
        Assert.Equal(10940m, result.TotalPrice);
    }

    [Fact]
    public async Task
    CalculateAsync_SectorNotFound_ThrowsNotFoundException()
    {
        // Arrange
        SetupDefaultTariffs();

        _sectorRepositoryMock
            .Setup(x => x.GetByIdAsync(999))
            .ReturnsAsync((Sector?)null);

        var service = CreateService();

        var request = CreateValidRequest();
        request.SectorId = 999;

        // Act & Assert
        await Assert.ThrowsAsync<NotFoundException>(() =>
            service.CalculateAsync(request));
    }

    [Fact]
    public async Task
    CalculateAsync_SectorUnavailable_ThrowsBusinessRuleException()
    {
        // Arrange
        SetupDefaultTariffs();

        _sectorRepositoryMock
            .Setup(x => x.GetByIdAsync(1))
            .ReturnsAsync(new Sector
            {
                Id = 1,
                Number = 1,
                MaxPeople = 10,
                IsAvailable = false,
                HasGazebo = false
            });

        var service = CreateService();
        var request = CreateValidRequest();

        // Act & Assert
        await Assert.ThrowsAsync<BusinessRuleException>(() =>
            service.CalculateAsync(request));
    }

```

```

[Fact]
public async Task
CalculateAsync_PeopleCountExceedsSectorCapacity_ThrowsBusinessRule
Exception()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: false, maxPeople: 2);

    var service = CreateService();
    var request = CreateValidRequest();

    // Act & Assert
    await Assert.ThrowsAsync<BusinessRuleException>(() =>
        service.CalculateAsync(request));
}

[Fact]
public async Task
CalculateAsync_SaleProductQuantityIsZero_ThrowsValidationException
()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: false);

    var service = CreateService();

    var request = CreateValidRequest();
    request.SaleProducts.Add(new BookingSaleProductItemDto
    {
        ProductId = 1,
        Quantity = 0
    });

    // Act & Assert
    await Assert.ThrowsAsync<ValidationException>(() =>
        service.CalculateAsync(request));
}

[Fact]
public async Task
CalculateAsync_RentProductQuantityIsZero_ThrowsValidationException
()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: false);

    var service = CreateService();

```

```

var request = CreateValidRequest();
request.RentProducts.Add(new BookingRentProductItemDto
{
    ProductId = 1,
    Quantity = 0
});

// Act & Assert
await Assert.ThrowsAsync<ValidationException>(() =>
    service.CalculateAsync(request));
}

[Fact]
public async Task
CalculateAsync_SaleProductNotFound_ThrowsNotFoundException()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: false);

    _saleProductRepositoryMock
        .Setup(x => x.GetByIdAsync(999))
        .ReturnsAsync((SaleProduct?)null);

    var service = CreateService();

    var request = CreateValidRequest();
    request.SaleProducts.Add(new BookingSaleProductItemDto
    {
        ProductId = 999,
        Quantity = 1
    });

    // Act & Assert
    await Assert.ThrowsAsync<NotFoundException>(() =>
        service.CalculateAsync(request));
}

[Fact]
public async Task
CalculateAsync_RentProductNotFound_ThrowsNotFoundException()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: false);

    _rentProductRepositoryMock
        .Setup(x => x.GetByIdAsync(999))
        .ReturnsAsync((RentProduct?)null);

    var service = CreateService();

```

```

var request = CreateValidRequest();
request.RentProducts.Add(new BookingRentProductItemDto
{
    ProductId = 999,
    Quantity = 1
});

// Act & Assert
await Assert.ThrowsAsync<NotFoundException>(() =>
    service.CalculateAsync(request));
}

[Fact]
public async Task
CalculateAsync_ValidRequest_CreatesBookingPositionsWithOriginalQuantities()
{
    // Arrange
    SetupDefaultTariffs();
    SetupAvailableSector(hasGazebo: true);

    var service = CreateService();
    var request = CreateValidRequest();

    // Act
    var result = await service.CalculateAsync(request);

    // Assert
    Assert.Contains(result.Positions, x =>
        x.TariffName == "Рыбак" &&
        x.Quantity == 2);

    Assert.Contains(result.Positions, x =>
        x.TariffName == "Женщина-рыбачка" &&
        x.Quantity == 1);

    Assert.Contains(result.Positions, x =>
        x.TariffName == "Мужчина-гость" &&
        x.Quantity == 1);

    Assert.Contains(result.Positions, x =>
        x.TariffName == "Женщина-гость" &&
        x.Quantity == 1);

    Assert.Contains(result.Positions, x =>
        x.TariffName == "Ребёнок" &&
        x.Quantity == 1);

    Assert.Contains(result.Positions, x =>
        x.TariffName == "Беседка" &&
        x.Quantity == 1);
}

```

```
Assert.Contains(result.Positions, x =>
    x.TariffName == "Бронирование сектора" &&
    x.Quantity == 1);
}
```

Приложение Д
(рекомендуемое)
Графическая часть

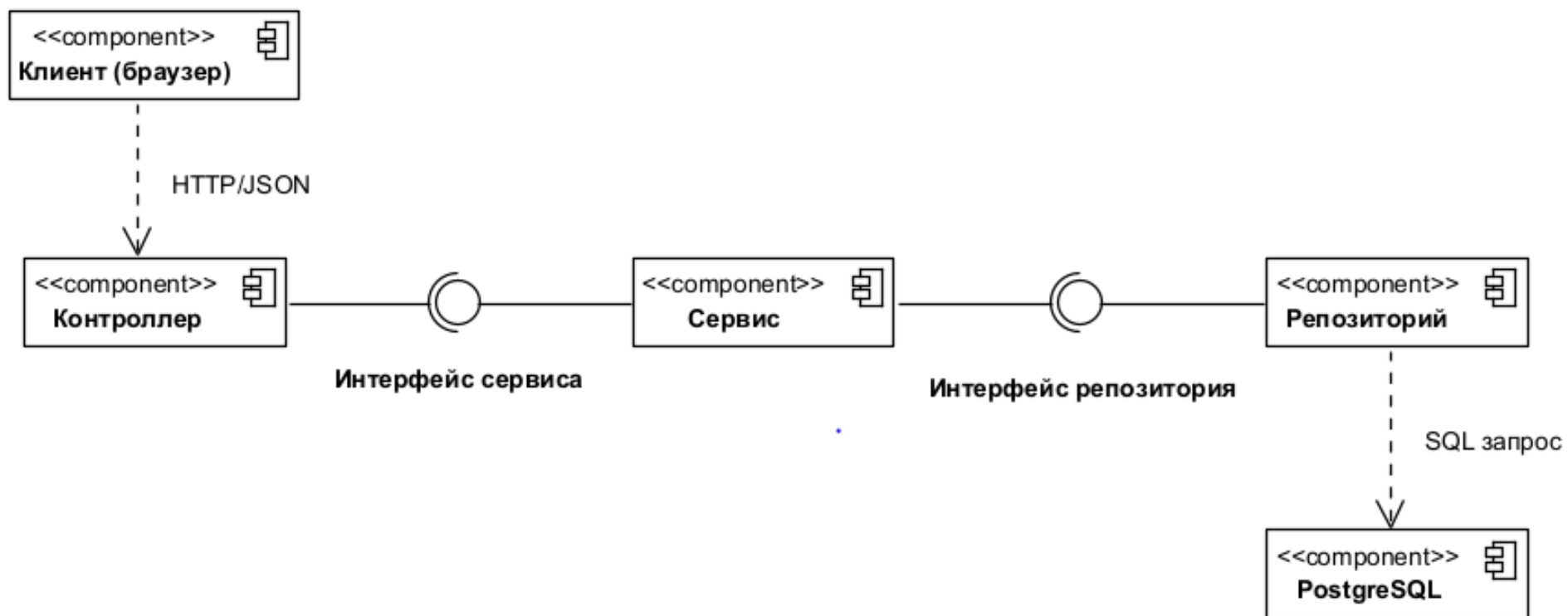


Рисунок Д.1 – Диаграмма компонентов серверной части

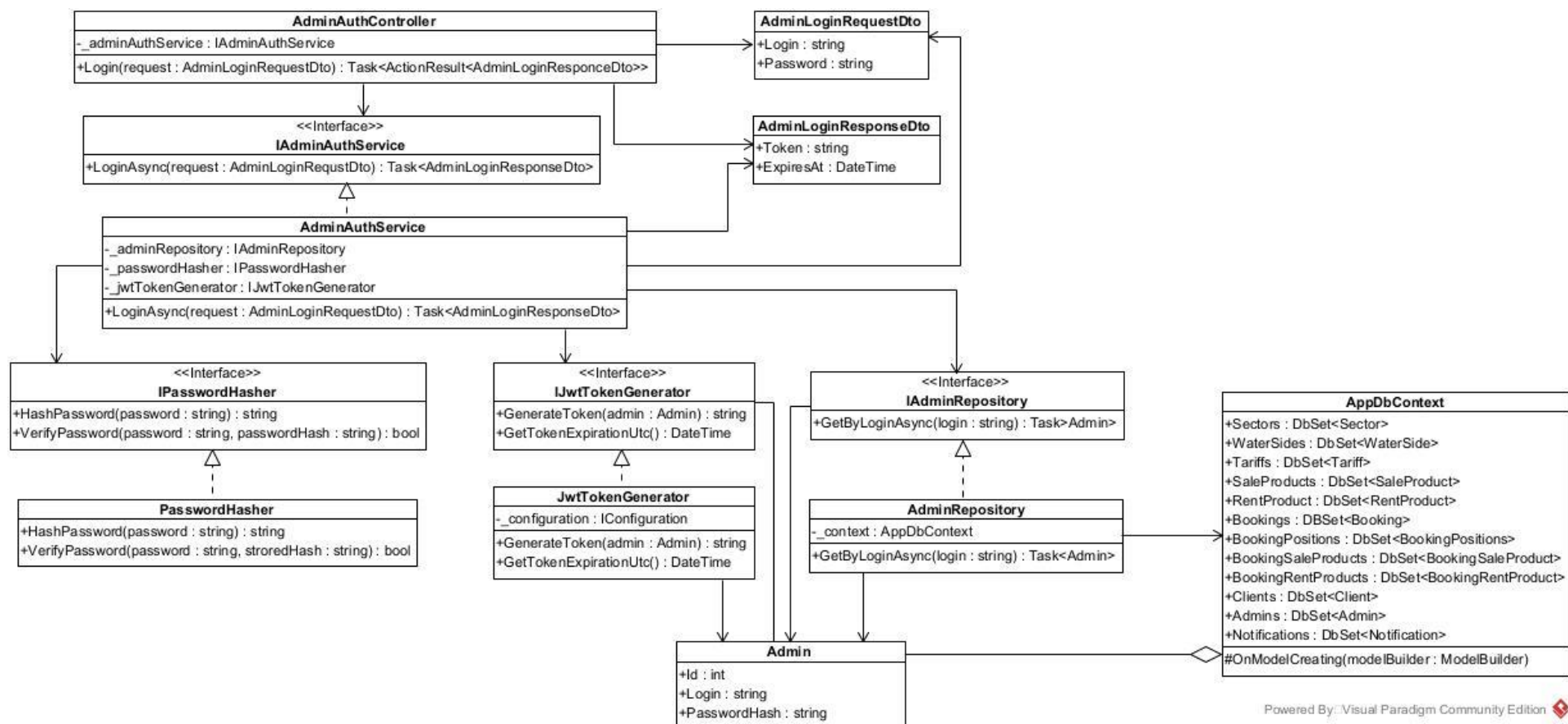


Рисунок Д.2 – Диаграмма классов модуля администратора

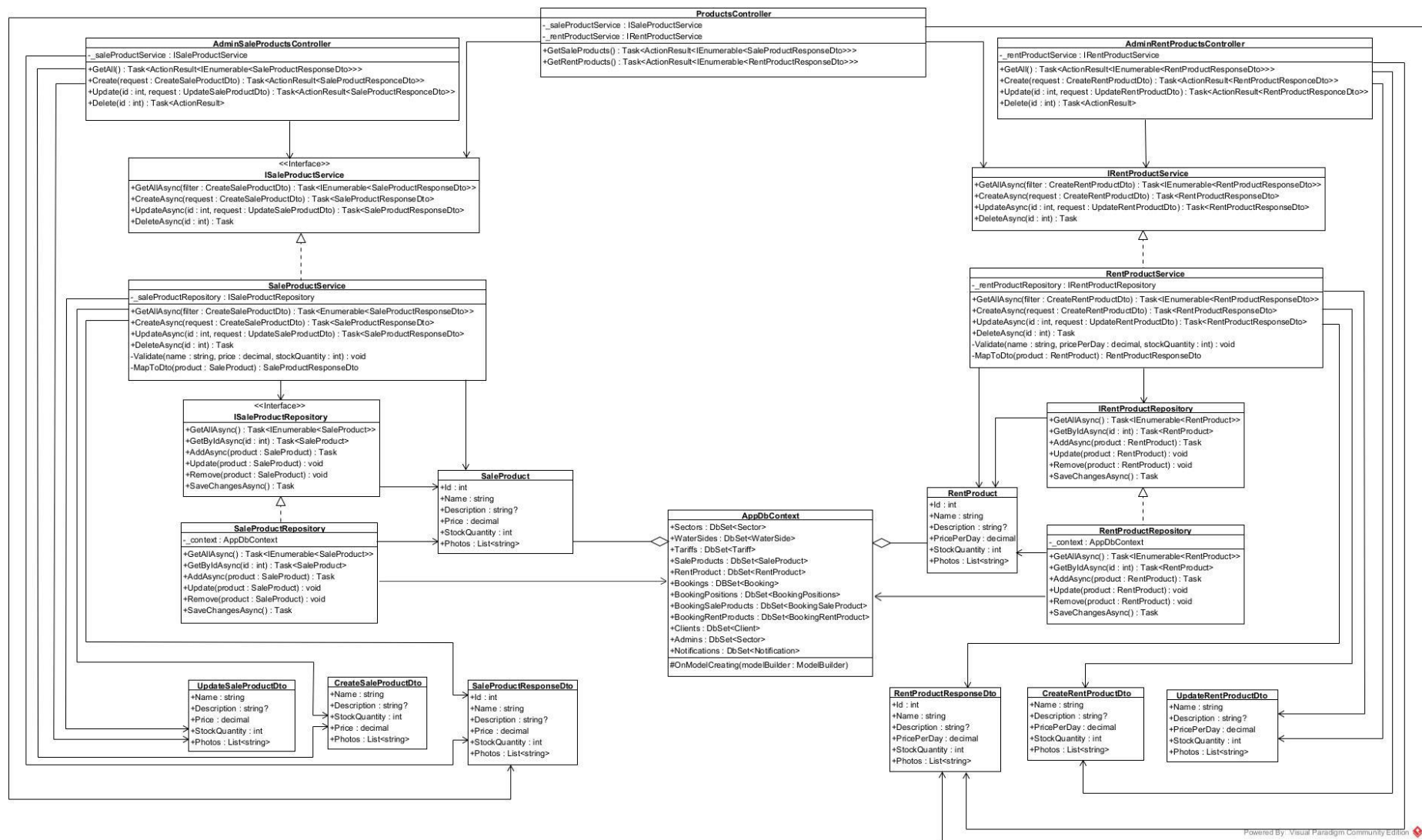


Рисунок Д.4 – Диаграмма классов модуля товаров

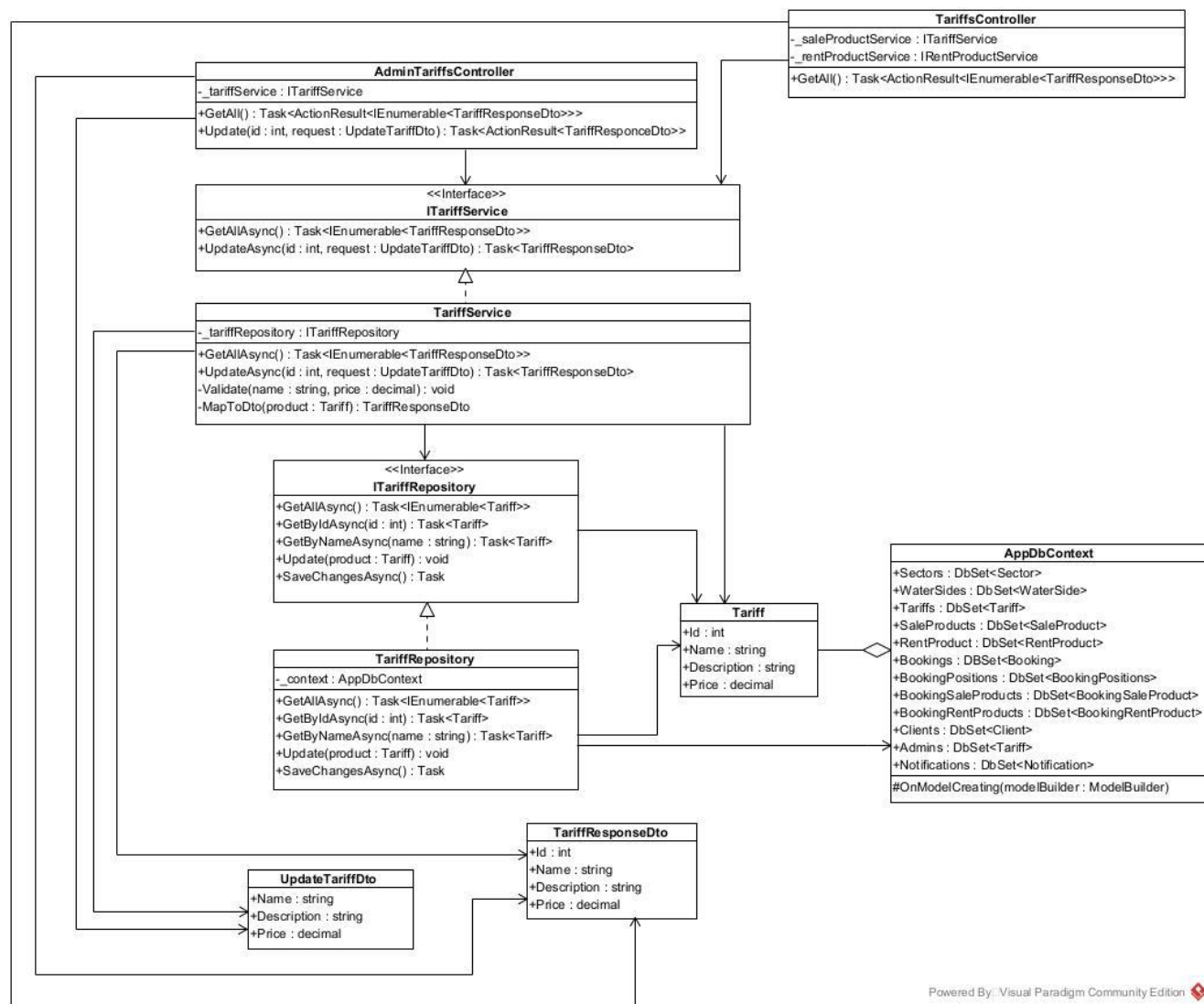


Рисунок Д.5 – Диаграмма классов модуля тарифов

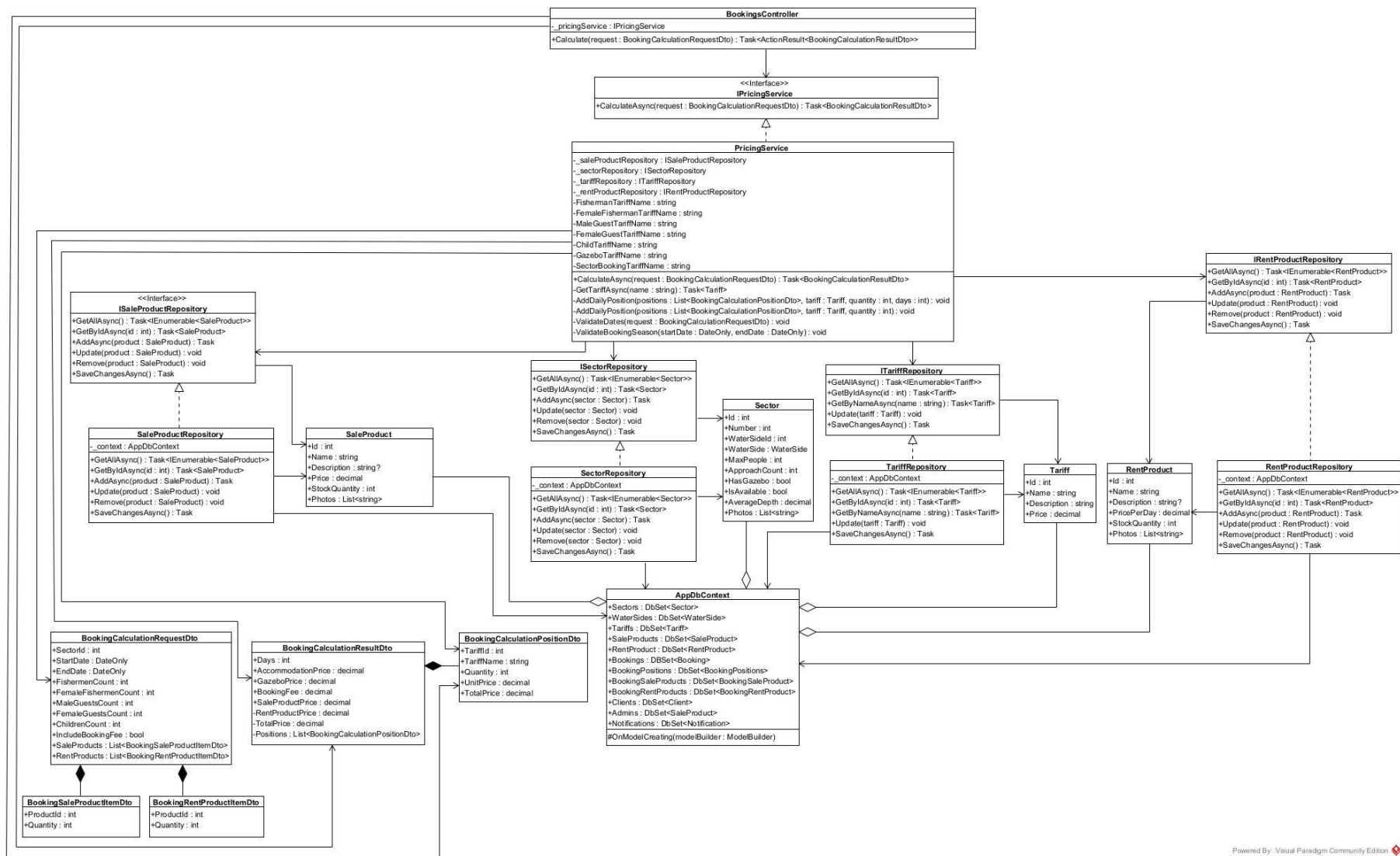


Рисунок Д.6 – Диаграмма классов модуля расчёта

Приложение Е
(рекомендуемое)
Акт о внедрении

**ИНДИВИДУАЛЬНЫЙ ПРЕДПРИНИМАТЕЛЬ
ГРИШИНА СВЕТЛАНА НИКОЛАЕВНА
ИНН 583401990443
ОГРН 317583500021289**

440061, г. Пенза, ул. Луначарского, д. 7а, кв. 3
тел. +7 902 343-51-01, E-mail: grishinasvetlana1973@gmail.com
Р/с: 40802810429170003445 в ФИЛИАЛ "НИЖЕГОРОДСКИЙ" АО "АЛЬФА-БАНК"
г. Нижний Новгород БИК: 042202824 К/с: 30101810200000000824

21.05.2026г.

Акт о внедрении

Результаты теоретических и практических исследований, приведённые в выпускной квалификационной работе Гришина Максима Михайловича на тему «Разработка бэкенд-компонента информационной системы автоматизации бронирования и учета услуг рыболовного хозяйства», внедрены в деятельность ИП Гришиной Светланы Николаевны, являющейся владельцем рыболовного хозяйства «Пруд "Орловский"», местоположение: на ручье без названия в 1,5 км на северо-запад от с. Кучки, Пензенский район Пензенской области.

Индивидуальный Предприниматель _____



/Гришина С.Н./