

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА
на тему
Разработка веб – платформы
туристического маркетплейса

Студент _____ Бакаева Алина Денисовна _____
(подпись, дата) (ФИО полностью)

Руководитель _____ Афонин А.Ю. _____
(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 Бакаева Алина Денисовна гр. 22ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Разработка веб – платформы туристического маркетплейса

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

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

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

1) *Назначение: веб – платформы туристического маркетплейса*

2) *Среда выполнения: Windows 11.*

Языковые средства разработки: TypeScript, CSS, HTML, SQL, UML.

Инструментальные средства: npm, Visual Studio Code, PostgreSQL, Prisma, Visual Paradigm 18.

Технологии разработки ООП, многослойная архитектура.

3) *Основные функции приложения:*

а) автоматизация операций;

б) управление контентом;

в) расчет стоимости туристических услуг;

г) оформление онлайн-заявки на бронирование;

д) хранение и систематизация информации;

е) управление и редактирование туристических карточек.

ж) управление отзывами клиентов;

5. Перечень подлежащих разработке вопросов

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

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 _____ 26.12.2025г.
(подпись, дата)

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

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

Реферат

Пояснительная записка содержит 109 листов, 44 рисунка, 13 таблиц, 21 использованный источник и 6 приложений.

ТУРИСТИЧЕСКИЙ МАРКЕТПЛЕЙС, ВЕБ-ПЛАТФОРМА, МОНОЛИТНАЯ АРХИТЕКТУРА, REST API, UML, VISUAL STUDIO CODE, HTML, CSS, TYPESCRIPT, БАЗА ДАННЫХ, POSTGRESQL, REACT, NODE.

Объект исследования – комплексные веб-системы, направленные на агрегацию туристических предложений и оптимизацию процесса выбора и бронирования услуг.

Цель выпускной квалификационной работы – проектирование и программная реализация платформы «туристический маркетплейс», предназначенной для централизованного управления базой туров и автоматизации взаимодействия между провайдерами и конечными пользователями.

Технологии разработки – разработка клиентской части приложения проводилась на языке программирования TypeScript с использованием библиотеки React в среде разработки Visual Studio Code. Разработка серверной части приложения также производилась на языке программирования TypeScript в аналогичной среде разработки с использованием фреймворка Node.JS. Разработка базы данных проводилась в СУБД PostgreSQL.

Среда исполнения – Windows 11, веб-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были спроектированы и внедрены клиентский и серверный уровни программного комплекса для туристического маркетплейса. Разработанный функционал обеспечивает стабильную работу, отвечая исходным техническим требованиям и задачам проектирования.

					ПГУ 09.03.02 – 08BP221.03 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Бакаева А.Д.			Разработка веб-платформы туристического маркетплейса. Пояснительная записка	Лит.	Лист	Листов
Пров.		Афонин А.Ю.					4	109
Реценз.						Группа 22ВИ1		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение	7
1. Анализ предметной области и постановка задачи на разработку веб-платформы туристического маркетплейса.....	9
1.1 Исследование предметной области.....	9
1.2 Обзор существующих веб-платформ туристического маркетплейса ..	10
1.2.1 Веб-платформа Tourvisor	10
1.2.2 Веб-платформа Travelata.....	11
1.2.3 Сравнение веб-платформ туристического маркетплейса	12
1.3 Анализ функциональных требований.....	14
1.4 Анализ нефункциональных требований.....	18
1.5 Планирование разработки и оценка бюджета.....	20
2 Проектирование программных средств веб – платформы туристического маркетплейса.....	24
2.1 Архитектура приложения.....	24
2.2 Проектирование структур данных	25
2.3 Проектирование пользовательского интерфейса.....	28
2.4 Проектирование классов программных средств для веб – платформы туристического маркетплейса	30
3 Реализация программных средств веб – платформы туристического маркетплейса.....	33
3.1 Реализация базы данных веб – платформы	33
3.2 Реализация серверной части веб – платформы.....	34
3.3 Реализация клиентской части веб – платформы.....	36
3.4 Реализация интерфейса веб – платформы	39

4	Контроль качества программного обеспечения	55
4.1	Обоснования выбора режима тестирования и метрик кода	55
4.2	Тестирование программных средств для веб – платформы туристического маркетплейса	56
4.2.1	Функциональное тестирование.....	56
4.2.2	Кросс-браузерное тестирование	59
4.3	Оценка качества программного кода	60
4.3.1	Метрики Холстеда	60
4.3.2	Цикломатическая сложность.....	61
	Заключение.....	65
	Список использованных источников.....	67
	Приложение А (справочное) Глоссарий	69
	Приложение Б (обязательное) Атрибуты сущности базы данных	72
	Приложение В (обязательное) Листинг серверной части приложения	76
	Приложение Г (обязательное) Листинг клиентской части приложения.....	87
	Приложение Д (рекомендуемое) Результаты функционального тестирования	101
	Приложение Е (рекомендуемое) Результаты кросс-браузерного тестирования	107

Введение

Цифровизация привела к существенным изменениям в индустрии путешествий. Туристическая деятельность, являясь информационно насыщенной индустрией, быстро реагирующей на все изменения, одной из первых ощутила на себе воздействие глобальной цифровизации. Внедрение цифровых технологий в сферу туризма формирует новые тенденции ее развития. Одним из факторов, способствующих цифровизации индустрии туризма, является доступность Интернет-среды [1]. Классический формат покупки путевок в физических офисах уступает место технологичным маркетплейсам, которые собирают предложения от разных туроператоров в едином интерфейсе и позволяют планировать отдых без лишних посредников.

Обоснованность разработки туристического маркетплейса связана с увеличением спроса на самостоятельные путешествия по России. Пользователи больше не хотят переплачивать за готовые пакетные решения с фиксированной логистикой. Разрабатываемая веб-платформа маркетплейса призвана решить данную задачу путем автоматизации поиска, структурирования туристических предложений по категориям и упрощения процесса бронирования.

Цель выпускной квалификационной работы заключается в разработке веб-платформы туристического маркетплейса для поиска и бронирования туристических предложений.

Для достижения указанной цели были поставлены следующие задачи:

- проведение анализа предметной области, изучение существующих рыночных аналогов и формулирование технических требований к сервису;
- проектирование архитектуры программного обеспечения и структуры базы данных;
- разработка программных модулей веб-платформы;
- тестирование разработанного приложения и контроль качества его функционирования.

Выполнение обозначенных задач направлено на создание функциональной и коммерчески перспективной веб-платформы маркетплейса с интуитивно понятным пользовательским интерфейсом.

1. Анализ предметной области и постановка задачи на разработку веб-платформы туристического маркетплейса

1.1 Исследование предметной области

Индустрия путешествий претерпела существенные изменения в формате взаимодействия с потребителем. В настоящее время гибкое формирование туров интегрирует услуги отельеров и локальных гидов, способствуя объединению проживания и экскурсий в единый цифровой продукт. Традиционные печатные буклеты и разрозненные веб-сайты отдельных гостиниц утратили прежнюю актуальность. Агрегатор собирает наземные маршруты в едином интерфейсе, обеспечивая концентрацию внимания пользователя и трансформируя многоэтапный поиск в рационализированный процесс. Эффективная программная реализация структурирует несистематизированные запросы в логичную систему бронирования. Это позволяет минимизировать опасения клиента и предоставляет пользователям, предпочитающим самостоятельное планирование логистики, расширенные возможности для индивидуализации маршрутов.

Рынок внутреннего туризма переживает значительные структурные изменения. Разработчики осуществляют сегментацию данного сектора, классифицируя предложения по категориям и географическим тегам. Изначально развитие интернет-бронирования наблюдалось преимущественно в сфере крупных отельных сетей. Однако в последующий период цифровые решения распространились на узкоспециализированные продукты – от городских туров до авторских экскурсий и пляжного отдыха в регионах России. Это предоставило локальным операторам возможность представлять уникальные маршруты, расширять клиентскую базу и осуществлять непосредственное взаимодействие с аудиторией. Данная выпускная квалификационная работа посвящена разработке веб-платформы туристического маркетплейса. Этот инструмент предназначен для интеграции разных предложений, предоставляя пользователю возможность гибкого формирования индивидуальной программы отпуска.

Разрабатываемая веб-платформа отличается рядом специфических особенностей, явно выделяющих среди стандартных информационных онлайн-платформ:

- динамическое визуальное оформление: внедрение высококачественных мультимедийных материалов (демонстрация локаций, отелей и экскурсионных объектов) для эффективного привлечения внимания потенциальных клиентов;
- интерактивное взаимодействие с аудиторией: внедрение гибких систем фильтрации предложений по категориям, стоимости или регионам, а также использование отзывчивых элементов интерфейса для повышения удобства навигации;
- структурированная архитектура веб-платформы: интуитивно понятное распределение логики по ключевым разделам платформы, включая единый каталог туров, личный кабинет пользователя, модули онлайн-бронирования и формы обратной связи.

1.2 Обзор существующих веб-платформ туристического маркетплейса

1.2.1 Веб-платформа Tourvisor

Tourvisor – это агрегатор туристических предложений с прямым доступом к базе данных более чем сто двадцать туроператоров [2]. Пользователь указывает направление и даты, программа подбрасывает предложения со стоимостью. Фильтры предполагают выбор типа питания, звездности отеля, цены и продолжительности поездки. Для каждого тура предусмотрены фотографии, описание объекта размещения, геолокация и кнопка бронирования. На рисунке 1 изображена страница с турами данного агрегатора.

Платформа справилась отлично со своей основной функциональной ролью – поиск и предварительное сравнение туристических предложений. Заказ совершается либо с помощью формы заявки на самой платформе, либо непосредственно на сайте туроператора, в зависимости от поставщика.

Эффективно, но создается очень некомфортная для пользователя ситуация – часть бронирования остается на Tourvisor, часть – на сторонних площадках.

Самые низкие цены

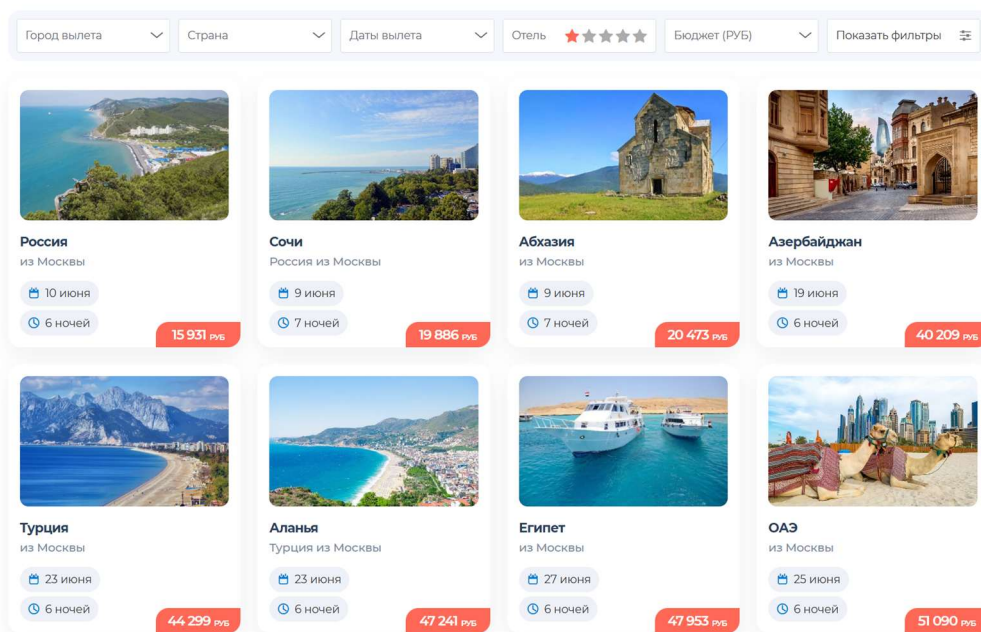


Рисунок 1 – Веб-платформа Tourvisor

Администрирование также является проблемой. Управлять туристическими компаниями, модерировать статусы заявок и отзывы невозможно при помощи внутреннего интерфейса. Все данные решаются на стороне поставщика услуг. Отзывы пользователей не привязаны к конкретному туру, они существуют отдельно и никак не влияют на ранжирование предложений.

1.2.2 Веб-платформа Travelata

Travelata ориентирована на сектор отдыха на пляжах и экскурсий. Перечень обновляется в режиме реального времени, а цены подтягиваются с помощью туроператоров непосредственно [3]. Путеводная карточка туристической программы подробнее, чем у Tourvisor: в ней отражается расписание программы по дням, питание, класс номера, а также информация о туроператоре – включая его логотип и краткое описание фирмы.

У Travelata имеется собственный кабинет. Все данные по истории бронирования туристических программ сохраняются, а также отчеты с

комментариями об отелях можно читать на странице соответствующей программы. Это ближе всего к потребностям простого пользователя. На рисунке 2 изображена страница с туристическим каталогом.

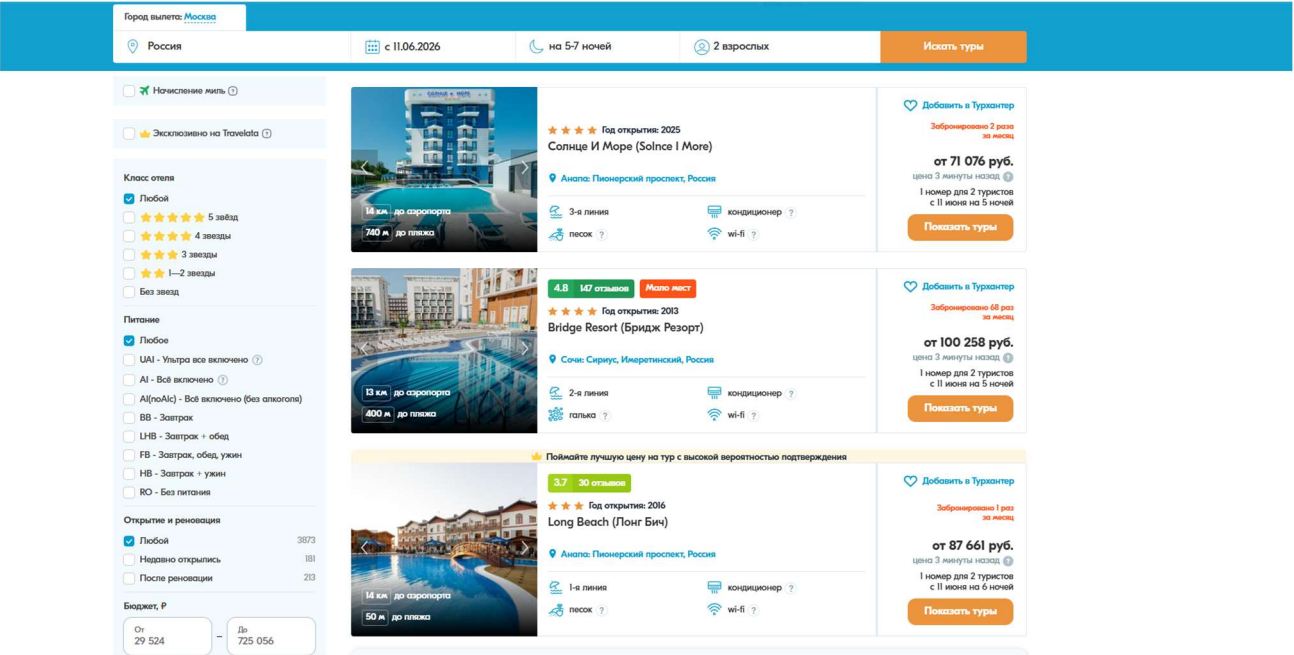


Рисунок 2 – Веб-платформа Travelata

Платформа почти лишилась административного интерфейса. Невозможно добавить нового поставщика, внести изменения в статус бронирования вручную, удалить сомнительный отзыв. Категории туров зафиксированы и не редактируются. По сути, витрина без склада: фасад проработан, а внутренняя механика скрыта от оператора.

1.2.3 Сравнение веб-платформ туристического маркетплейса

Выполним детальный анализ для сравнения рассмотренных выше веб – платформ, таких как, Travelata и Tourvisor с разрабатываемой платформой. Анализ будет осуществляться по двенадцати критериям.

Для оценивания будет применяться шкала из трех баллов. В таблице 1 представлены параметры оценки.

Таблица 1 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) полностью отсутствует

Продолжение таблицы 1

Балл	Характеристика оценки
1	существует частичная реализация функции (качества)
2	функция (качество) полностью реализована

Сравнение веб – платформ аналогичного назначения с разрабатываемой платформой представлены в таблице 2.

Таблица 2 – Сравнение веб – платформ туристических маркетплейсов

Критерий\Платформа	Tourvisor	Travelata	TravelMarket (разрабатываемая)
1. Каталог туров с фильтрацией	2	2	2
2. Карточка тура с подробным описанием	2	2	2
3. Система бронирования туров	2	1	2
4. Личный кабинет пользователя	2	2	2
5. Управление статусами бронированием	0	1	2
6. Система отзывов и оценок туров	2	1	2
7. Управление туроператорами (провайдерами)	0	1	2
8. Панель администратора	2	0	2
9. Категоризация туров	2	2	2
10. Загрузка нескольких фото к туру	2	1	2
11. Фильтрация по цене и категории	2	2	2
12. Отображение логотипа туроператора на карточке тура	1	0	2
ИТОГО	19	15	24

Сводные цифровые показатели отображают реальное положение дел. В части основных функций пользовательского интерфейса, включающих: поисковый каталог, карточки туристических продуктов и персональный кабинет клиента, все три платформы демонстрируют максимальную оценку в два балла. Однако на этом этапе сходство прекращается. Расхождения в показателях начинаются в области внутренней работы с контентом, включая: ручное

отслеживание статус бронирований, координацию деятельности поставщиков услуг, пакетную загрузку изображений и гибкую модерацию отзывов.

Архитектура Tourvisor не предусматривает выполнение указанных задач, что проявляется в нулевых показателях по администрированию и управлению статусами заявок. Итоговый результат данной платформы составил девятнадцать баллов. Платформа Travelata предпринимает попытки устранить эти недостатки. Однако реализация данной функциональности неполна, что приводит к снижению оценки до одного балла. Это обусловлено строгими ограничениями на загрузку медиафайлов и полным отсутствием функциональной административной панели. В итоге сервис Travelata получает только пятнадцать баллов.

Выявленный дефицит в управленческой функциональности послужил основой для определения вектора проектирования. Разрабатываемая веб – платформа достигает максимальной оценки в двадцать четыре балла. Принципы построения ядра системы изначально предусматривают отсутствие изолированных функциональных модулей. Разработка осуществлялась с целью обеспечения возможности точечного контроля всего жизненного цикла данных одним администратором: от верификации новых локальных туроператоров до оперативного изменения статуса конкретного заказа, все это в рамках единого рабочего пространства.

1.3 Анализ функциональных требований

В разработке веб – платформы туристического маркетплейса были определены три объекта пользования сайта – это администратор, гость и клиент.

Особенности функциональных требований для приложения можно отобразить с помощью диаграммы вариантов использования UML [4].

На рисунке 3 представлено наследование прав доступа пользователями.

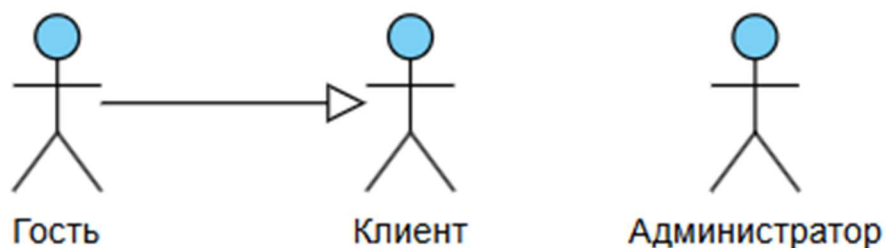


Рисунок 3 – Диаграмма вариантов использования пользователей системы веб – платформы

Администратор сайта обладает следующим набором функциональных требований:

- авторизация;
- создание конкретного тура;
- изменение конкретного тура;
- удаление конкретного тура из каталога;
- добавление категории тура;
- редактирование категории тура;
- удаление категории тура;
- добавление провайдера в библиотеку туроператоров;
- изменение карточки туроператора;
- удаление туроператора из библиотеки;
- просмотр и модерация отзывов пользователей;
- удаление некорректных отзывов;
- просмотр заявок на бронирование туров от клиентов;
- изменение статуса заявки на бронирование (ожидание, подтверждение, отклонение).

Гость сайта обладает следующим набором функциональных требований:

- регистрация в системе для создания учетной записи;
- авторизация в системе под своими учетными данными;
- просмотр главной страницы маркетплейса;

- просмотр каталога доступных туров;
- поиск тура по категории;
- поиск тура по названию и направлению;
- просмотр подробной карточки тура;
- просмотр отзывов других клиентов.

Клиент сайта обладает следующим набором функциональных требований:

- просмотр данных своего личного профиля;
- изменение пароля в личном кабинете;
- изменение заглавного фото в личном кабинете;
- создание новой заявки на бронирование тура;
- просмотр списка личных бронирований;
- отслеживание изменения статуса своей заявки на бронирование.

Диаграмма вариантов использования для веб –платформы представлена на рисунке 4.

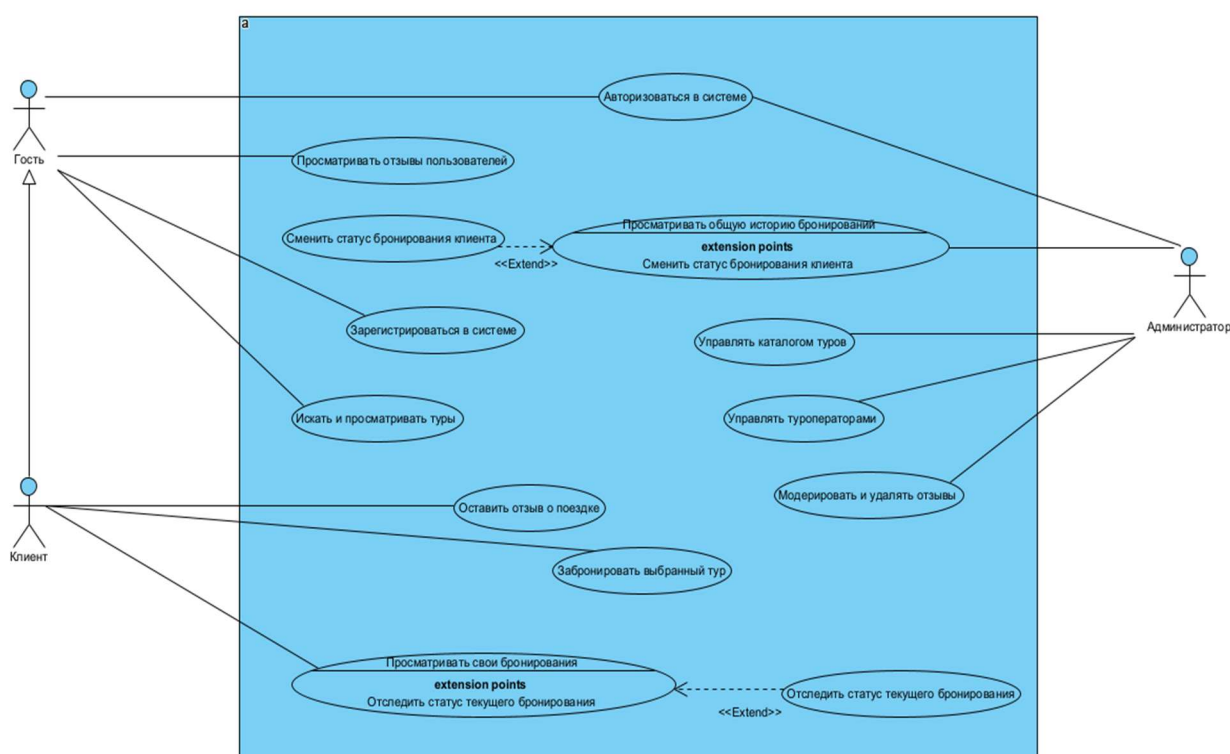


Рисунок 4 – Диаграмма вариантов использования веб – платформы

Выделенные функциональные требования четко разграничивают возможности каждого пользователя. Данные из этого списка станут основой для дальнейшей разработки веб –платформы туристического маркетплейса.

Ниже приведены спецификации наиболее важных бизнес-процессов.

Сценарий варианта использования «Авторизоваться в системе» представлен на рисунке 5.

Прецедент: авторизоваться в системе
ID: 1
Краткое описание: Гость авторизуется на платформе
Главные актеры: Гость
Второстепенные актеры: Нет
Предусловие: 1. Пользователь открыл страницу входа 2. Пользователь не был авторизован
Основной поток: 1. Гость переходит на страницу авторизации 2. Вводит адрес электронной почты 3. Вводит пароль 4. Нажимает кнопку «Войти» 5. Система проверяет учётные данные 6. Система создаёт сессию и перенаправляет пользователя в каталог туров
Постусловия: 1. Пользователь авторизован в системе 2. Пользователю доступны функции личного кабинета
Альтернативный поток: 1. Учётные данные некорректны — система отображает сообщение об ошибке 2. Пользователь повторяет ввод

Рисунок 5 – Сценарий варианта использования «Авторизоваться в системе»

Сценарий варианта использования «Редактировать тур» представлен на рисунке 6.

Прецедент: редактировать тур
ID: 2
Краткое описание: Администратор обновляет информацию о существующем туре через административную панель
Главные актеры: Администратор
Второстепенные актеры: Нет
Предусловие: 1. Администратор авторизован в системе 2. Тур существует в базе данных
Основной поток: 1. Администратор открывает раздел «Туры» в административной панели 2. Находит нужный тур в списке 3. Нажимает кнопку «Изменить» 4. Система отображает форму редактирования с текущими данными тура 5. Администратор изменяет нужные поля (название, описание, цену, фото и др.) 6. Нажимает кнопку «Сохранить» 7. Система валидирует данные и сохраняет изменения 8. Тур отображается в каталоге с обновлёнными данными
Постусловия: 1. Данные тура обновлены в системе 2. Обновлённые данные отображаются в каталоге туров
Альтернативный поток: 1. Поле «Название» или «Цена» пустое — система выводит сообщение об ошибке валидации 2. Данные не сохраняются до исправления ошибки

Рисунок 6 – Сценарий варианта использования «Редактировать тур»

Сценарий варианта использования «Забронировать выбранный тур» представлен на рисунке 7.

Прецедент: забронировать выбранный тур
ID: 3
Краткое описание: Клиент оформляет бронирование на выбранный тур через карточку тура
Главные актеры: Клиент
Второстепенные актеры: Нет
Предусловие: 1. Клиент авторизован в системе 2. Тур существует и доступен в каталоге
Основной поток: 1. Клиент открывает карточку тура в каталоге 2. Изучает описание, фотографии и данные туроператора 3. Нажимает кнопку «Забронировать тур» 4. Система создаёт запись о бронировании со статусом «Ожидает» 5. Клиент перенаправляется в раздел «Мои бронирования» 6. Клиент видит актуальный статус своей заявки
Постусловия: 1. Создана запись о бронировании со статусом «Ожидает» 2. Администратор видит заявку в административной панели
Альтернативный поток: 1. Клиент не авторизован — система перенаправляет на страницу входа 2. После авторизации клиент возвращается на карточку тура

Рисунок 7 – Сценарий варианта использования «Забронировать выбранный тур»

Анализ функциональных требований позволил выявить ключевые сценарии взаимодействия с разрабатываемой системой маркетплейса. Функционирование приложения без данных последовательностей операций существенно снижает его практическую ценность. системы. Данные описания служат основой для последующей разработки пользовательских интерфейсов и реализации серверной логики веб – платформы.

1.4 Анализ нефункциональных требований

Реляционное хранилище данных веб – платформы туристического маркетплейса реализовано с применением СУБД PostgreSQL [5]. Выбор этой системы с открытым исходным кодом обусловлен ее способностью обеспечивать эффективную изоляцию транзакций в условиях высокой нагрузки. Архитектура данной СУБД обеспечивает отказоустойчивость, высокую производительность при выполнении сложных JOIN-запросов и встроенную поддержку неструктурированных данных формата JSON. Интеграция серверных

программных слоев с таблицами СУБД реализована посредством инструмента Prisma ORM [6]. Данный инструмент автоматически генерирует типы данных на основе схемы базы, что снижает риски некорректного приведения типов в ходе выполнения транзакций.

Клиентский интерфейс маркетплейса разработан с использованием библиотеки React и строго типизированного языка программирования TypeScript. Визуальные интерфейсы и логика клиентских компонентов разрабатывались в среде Visual Studio Code. Для преобразования исходных скриптов, минимизации кода и окончательной сборки дистрибутива применяется инструмент Vite. Данное высокоскоростное решение заменяет традиционный сборщик Webpack, что значительно ускоряет процесс отладки приложения за счет функции Hot Module Replacement и оптимизирует размер конечного интерфейсного бандла.

Серверная часть системы реализована в рамках монолитной архитектуры, которая консолидирует логику приложения в единой кодовой базе. Подобный подход упрощает сквозное тестирование компонентов, снижает издержки на сетевое взаимодействие и облегчает развертывание сервиса. Серверная часть разработана на языке TypeScript для платформы Node.js с использованием веб-фреймворка Express. Взаимодействие между клиентскими и серверными модулями осуществляется в соответствии с архитектурным стилем REST API [6]. Клиент отправляет асинхронные HTTP-запросы к строго определенным конечным точкам маршрутизатора, а сервер формирует структурированные JSON-ответы, обеспечивая динамическое обновление интерфейса.

Моделирование архитектурных слоев и визуализация графических представлений системы осуществляется с использованием унифицированного языка моделирования UML 2.0. Структурные схемы, такие как диаграммы классов, прецедентов использования и физического распределения компонентов, построены с использованием инструментария Visual Paradigm [7]. Эта программная среда поддерживает нотации объектно-ориентированного проектирования как в локальном, так и в облачном режимах.

Технические требования к разрабатываемому веб-приложению включают требование кроссбраузерной оптимизации для обеспечения стабильного отображения интерфейсных компонентов. Клиентская часть маркетплейса обеспечивает корректное отображение и выполнение скриптов в актуальных версиях популярных веб-браузеров: Google Chrome (сборки 80 и выше), Mozilla Firefox (версии 75 и выше), и Microsoft Edge (версии 80 и выше). Это проектное решение минимизирует риски некорректного отображения стилей и обеспечивает стабильную реактивность интерфейса на устройствах под управлением различных операционных систем.

1.5 Планирование разработки и оценка бюджета

Управление процессом разработки и расчет финансовой составляющей являются основополагающими для успешной реализации программных продуктов. В современной инженерии проектов выделяют два основных методологических подхода к ведению проектов: гибкий «Agile» и каскадный «Waterfall». Выбор методологии непосредственно определяет дальнейший вектор планирования [8].

В рамках данной выпускной квалификационной работы используется классическая каскадная модель управления. Процесс разработки в данной модели строго линеен. Переход к последующему этапу возможен только после полного завершения текущего. Для фиксации временных интервалов, мониторинга задач и расчета финансовых издержек задействована среда ProjectLibre. Эта программа является кроссплатформенным, свободно распространяемым аналогом проприетарного пакета Microsoft Office Project 2007.

Начальный этап планирования включал инициализацию проектной документации. Были зафиксированы название проекта, дата его начала и параметры проектного менеджера. Дальнейшее формирование структуры включало декомпозицию процесса на отдельные фазы и распределение проектных ресурсов. В контексте данной методологии под ресурсами

понимается совокупность всех факторов, задействованных в производстве: кадровый состав, техническое обеспечение, расходные материалы и бюджетные ассигнования [9].

Производственный цикл создания маркетплейса подразделен на 4 основных этапа и 18 локальных задач. Начало проекта определено на 8 апреля 2026 года. Завершение и сдача готового веб – платформы запланированы на 4 июня 2026 года.

Сведения об этапах разработки представлены на рисунке 8.

Название	Продолжитель...	Начало	Окончание
Начало проекта	0 дней	08.04.2026, 08:00	08.04.2026, 08:00
Е Анализ предметной области и постано	9 дней	09.04.2026, 08:00	21.04.2026, 17:00
Анализ предметной области	0,25 дней	09.04.2026, 08:00	09.04.2026, 10:00
Выбор приложений-аналогов	0,375 дней	14.04.2026, 08:00	14.04.2026, 11:00
Обзор приложений-аналогов	0,75 дней	15.04.2026, 08:00	15.04.2026, 15:00
Сравнительный анализ приложений-анал	0,5 дней	17.04.2026, 08:00	17.04.2026, 13:00
Результаты сравнительного анализа	0,062 дней	17.04.2026, 08:00	17.04.2025, 08:30
Анализ функциональных требований	1,5 дней	18.04.2026, 08:00	21.04.2026, 13:00
Анализ нефункциональных требований	1,5 дней	21.04.2026, 08:00	21.04.2026, 17:00
Е Проектирование структур данных	7,167 дней	21.04.2026, 08:00	30.04.2026, 09:20
Проектирование архитектуры приложения	5 дней	21.04.2026, 08:00	25.04.2026, 17:00
Проектирование структур данных	2,667 дней	24.04.2026, 08:00	24.04.2026, 14:20
Проектирование пользовательского интер	1,5 дней	24.04.2025, 14:20	28.04.2025, 09:20
Проектирование классов программных сре	2 дней	28.04.2025, 09:20	30.04.2025, 09:20
Е Реализация веб-приложения	11,333 дней	03.05.2026, 08:00	20.05.2026, 10:40
Реализация базы данных	2 дней	03.05.2026, 08:00	06.05.2026, 17:00
Реализация серверной части приложения	5 дней	07.05.2026, 08:00	13.05.2026, 17:00
Реализация клиентской части приложения	2,667 дней	14.05.2026, 08:00	16.05.2026, 14:20
Реализация пользовательского интерфейса	1,667 дней	16.05.2026, 14:20	20.05.2026, 10:40
Е Контроль качества программного обес	7,375 дней	25.05.2026, 08:00	04.06.2026, 11:00
Определение режима тестирования	1 день	25.05.2026, 08:00	26.05.2026, 17:00
Тестирование программных средств прило	4,5 дней	27.05.2026, 08:00	02.06.2026, 13:00
Оценка качества программного кода	0,625 дней	03.06.2026, 15:00	04.06.2026, 11:00
Конец проекта	0 дней	04.06.2026, 11:00	04.06.2026, 11:00

Рисунок 8 – Сведения об этапах разработки приложения веб – платформы

Производственный цикл создания маркетплейса подразделен на 4 основных этапа и 18 локальных задач. Начало проекта определено на 8 апреля 2026 года. Завершение и сдача готового веб – платформы запланированы на 4 июня 2026 года.

Визуализация календарного планирования и взаимных зависимостей между задачами представлена диаграммой Ганта, сгенерированной соответствующей средой, приведенной на рисунке 9 [10].

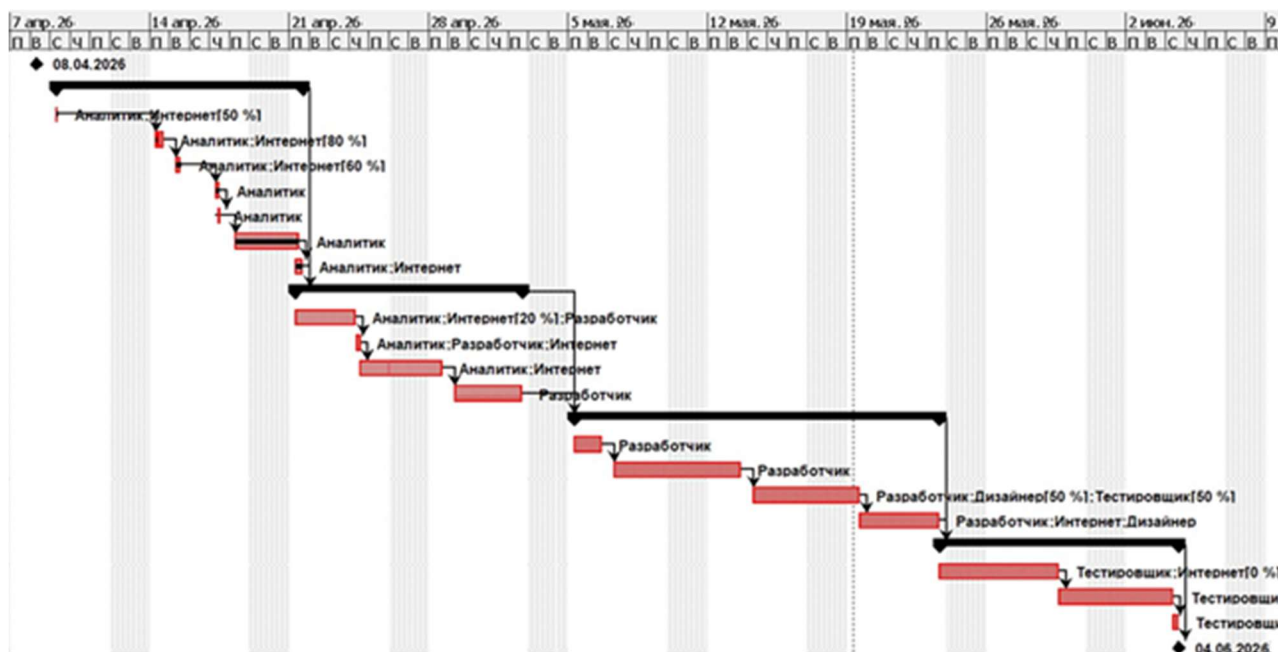


Рисунок 9 – Диаграмма Ганта по этапам разработки веб – платформы

Расчет материальных затрат на производство программного обеспечения требует четкого разграничения трудовых ресурсов и определения стоимости их работы (рисунок 10).

№	Название	Тип	Стандартная ставка	Начислять	Основной календарь
1	Аналитик	Работа	400,00 Р/час	Пропорционально	Пятидневка
2	Разработчик	Работа	400,00 Р/час	Пропорционально	Пятидневка
3	Дизайнер	Работа	250,00 Р/час	Пропорционально	Пятидневка
4	Тестировщик	Работа	350,00 Р/час	Пропорционально	Пятидневка
5	Интернет	Работа	0,00 Р/час	Пропорционально	Пятидневка

Рисунок 10 – Трудовые ресурсы и их стоимость

Интеграция указанных параметров позволяет оценить суммарные трудозатраты. При стандартной пятидневной рабочей неделе общий объем рабочего времени специалистов составляет 279 часов. Финансовая оценка каждого этапа основывается на производстве отработанных нормативных часов, общего количества привлеченных профильных специалистов и их персональной почасовой тарифной ставки. Последующее суммирование локальных затрат позволяет определить итоговую финансовую стоимость всего проекта. Сводный бюджет разработки представлен на рисунке 11.

Название	Стоимость
Анализ предметной области и постановка задачи	30200,00 Р
Анализ предметной области	800,00 Р
Выбор приложений-аналогов	1200,00 Р
Обзор приложений-аналогов	2400,00 Р
Сравнительный анализ приложений-аналогов	1600,00 Р
Результаты сравнительного анализа	200,00 Р
Анализ функциональных требований	4800,00 Р
Анализ нефункциональных требований	3200,00 Р
Проектирование структур данных	33333,33 Р
Проектирование архитектуры приложения	6400,00 Р
Проектирование структур данных	4266,67 Р
Проектирование пользовательского интерфейса	4800,00 Р
Проектирование классов программных средств	6400,00 Р
Реализация веб-приложения	43830,56 Р
Реализация базы данных	6400,00 Р
Реализация серверной части приложения	16000,00 Р
Реализация клиентской части приложения	10666,67 Р
Реализация пользовательского интерфейса	8666,67 Р
Контроль качества программного обеспечения	18200,00 Р
Определение режима тестирования	1050,00 Р
Тестирование программных средств приложения	12600,00 Р
Оценка качества программного кода	1750,00 Р
Конец проекта	0,00 Р
Итого: 125 563,89 руб.	

Рисунок 11 – Оценка бюджета

Итоговый расчет стоимости реализации программного комплекса составляет 125 563 рубля и 89 копеек.

2 Проектирование программных средств веб – платформы туристического маркетплейса

2.1 Архитектура приложения

Веб – платформа туристического маркетплейса основана на принципах монолитной архитектуры. Этот паттерн проектирования обеспечивает простоту развертывания проекта и гарантирует изоляцию компонентов в рамках единой технологической экосистемы. Консолидация программных модулей в единой кодовой базе значительно сокращает накладные расходы. Это исключает необходимость проектирования сложных сетевых протоколов для межсервисного взаимодействия, настройки распределенных сред и оркестрации контейнеров, поскольку компоненты интегрируются напрямую.

Клиентское веб-приложение выступает в роли фронтального интерфейса для визуализации данных. Оно передает команды пользователя на сервер, используя асинхронный механизм REST API [6].

Коммуникационные потоки распределены между изолированными конечными точками. Для отображения логической структуры и разграничения ответственности модулей была спроектирована UML-диаграмма компонентов, представленная на рисунке 12.

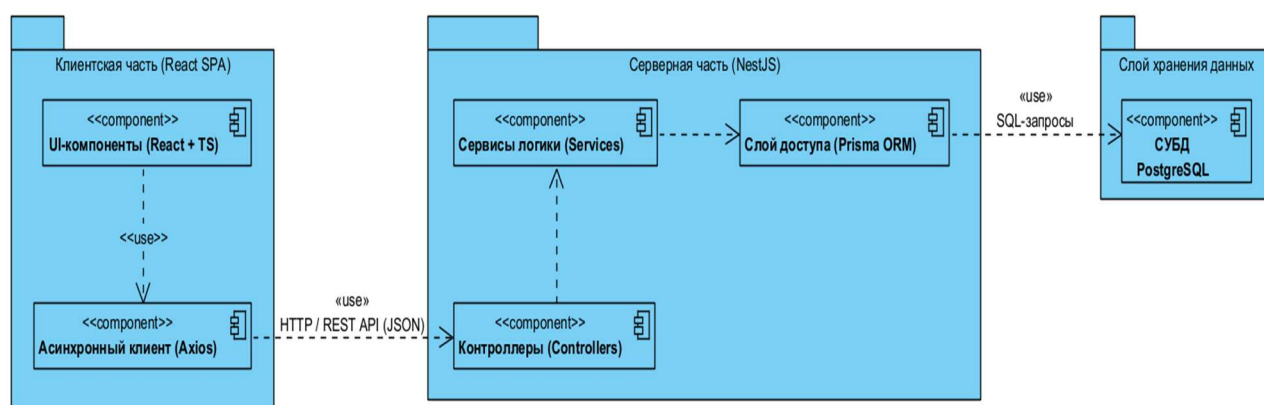


Рисунок 12 – Диаграмма компонентов структуры веб – платформы туристического маркетплейса

Физическое распределение ресурсов и сетевая топология представлены диаграммой развертывания на рисунке 13.

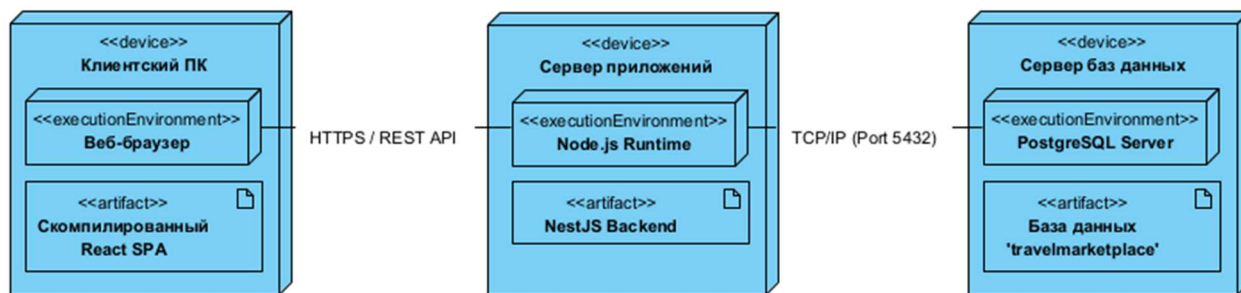


Рисунок 13 – Диаграмма развертывания архитектуры веб – платформы туристического маркетплейса

В контексте UML, диаграмма развертывания входит в семейство структурных диаграмм [11]. Схема отражает статичное состояние аппаратных узлов и описывает процесс проецирования программных артефактов на физические серверные мощности. В данной архитектуре серверный модуль функционирует как интеллектуальный шлюз, балансирующий потоки данных между базой данных и внешними запросами от клиентских браузеров.

2.2 Проектирование структур данных

Проектирование реляционного хранилища данных начинается с построения концептуальной модели. Концептуальная модель – это совокупность концептов (понятий, смыслов) и их связей [12]. Эта абстракция чисто отражает логику предметной области, исключая специфику конкретных систем управления базами данных (СУБД).

Проектирование концептуальной схемы предполагает точную идентификацию информационных сущностей. Для маркетплейса было спроектировано шесть базовых сущностей. Их функциональное назначение представлено в таблице 3.

Таблица 3 – Сущности базы данных

Название сущности	Идентификатор сущности	Название сущности
Пользователь	users	Содержит личные данные пользователей и их системные роли для авторизации
Категория	categories	Содержит сведения о категориях туров
Провайдер	providers	Содержит информацию о провайдерах, предоставляющих туры
Тур	tours	Содержит информацию о конкретном туре
Бронирование	bookings	Содержит всю информацию о заказах туров пользователями
Отзыв	reviews	Содержит оценки, текстовые комментарии и отзывы

Визуальное представление концептуальной схемы представлено на рисунке 14. Логика связывания этих объектов представлена посредством топологии отношений, зафиксированной в таблице 4.

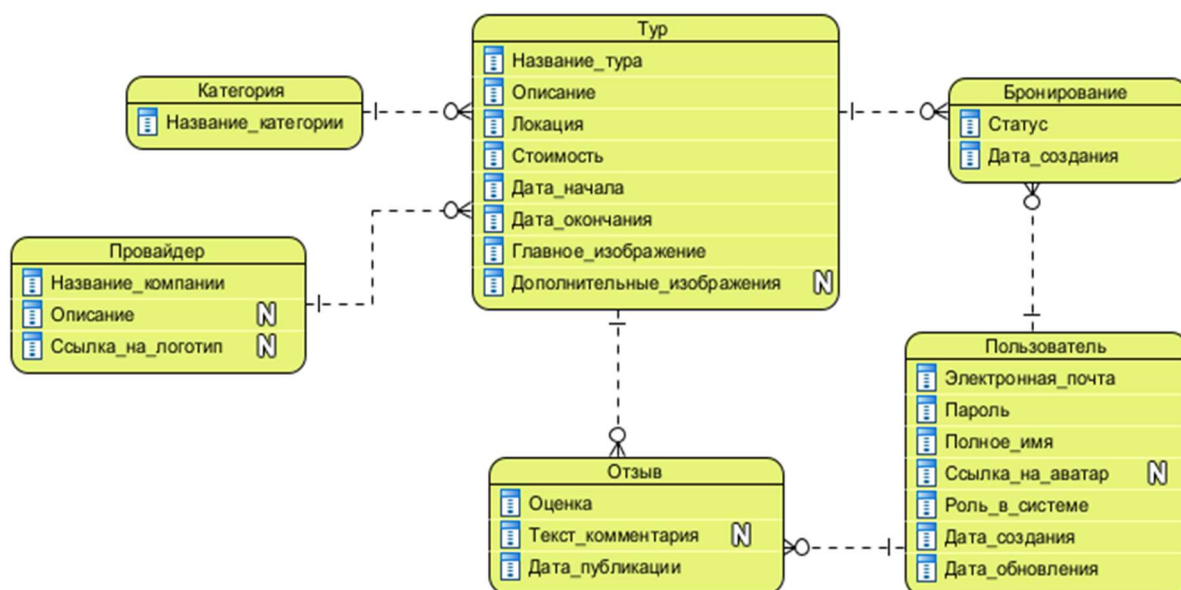


Рисунок 14 – Концептуальная модель веб – платформы туристического маркетплейса

Таблица 4 – Описание связей между сущностями

Наименование связи	Описание	Мощность
Провайдер – Тур	Каждая компания – провайдер может предлагать и размещать на маркетплейсе множество различных туров	1:M
Категория – Тур	В рамках одной категории может быть представлено несколько туров	1:M
Пользователь – Бронирование	Один зарегистрированный клиент имеет право оформить множество заявок на бронирование	1:M
Тур – Бронирование	На один и тот же тур может быть создано множество заявок на бронирование от клиентов	1:M
Пользователь –Отзыв	Каждый пользователь системы может оставить множество отзывов	1:M
Тур – Отзыв	Один тур может получить множество отзывов	1:M

Полный спектр полей, типы данных и специфические ограничения для каждого атрибута представлены в таблицах 1–6 приложения А. На основе концептуального каркаса и описанных взаимосвязей была сформирована логическая модель данных (рисунок 15).

Построение концептуального и логического представлений осуществлялось в инструментальной среде Visual Paradigm 18 с использованием методологического стандарта нотации IDEF1X. Спроектированная логическая схема является основой для генерации физической модели, которая затем транслируется непосредственно в SQL-код для развертывания таблиц на сервере.

Логическая модель данных является уточнением и детализацией концептуальной модели данных [12].

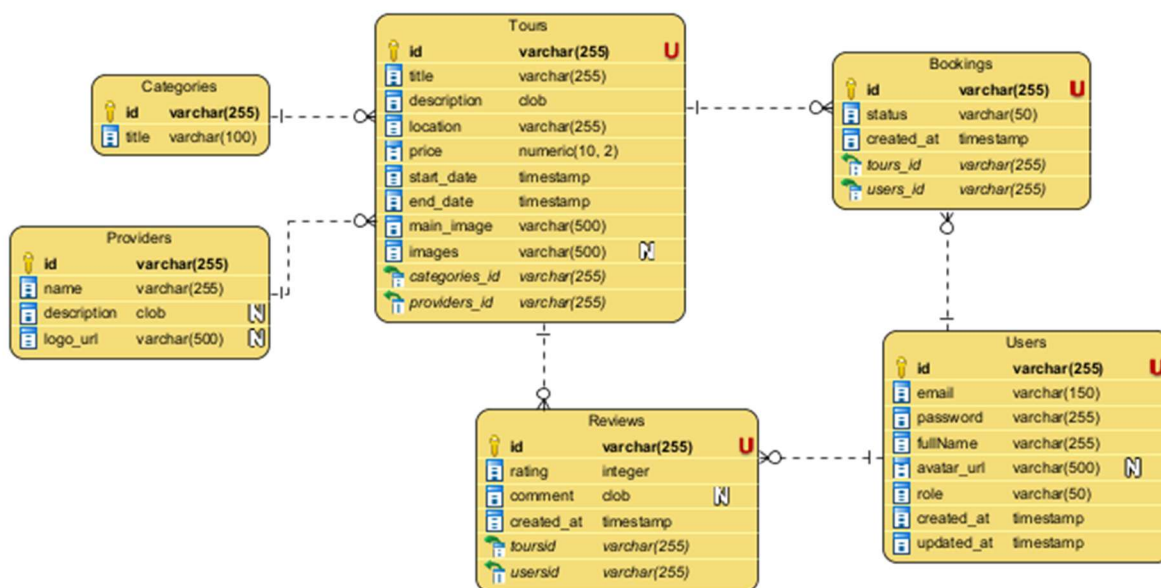


Рисунок 15 – Логическая модель веб – платформы туристического маркетплейса

Описанная логическая модель учитывает все ключевые сущности предметной области, нормализована до необходимой степени и служит прямой основой для последующей генерации скриптов создания в СУБД.

2.3 Проектирование пользовательского интерфейса

Проектирование визуальной среды направлено на снижение когнитивной нагрузки на пользователя. Навигационные пути должны быть прозрачными и предсказуемыми.

Проектированная архитектура интерфейса предусматривает следующие страницы:

- «Авторизация»;
- «Регистрация»;
- «Главная страница»;
- «Страница тура»;
- «Личный профиль пользователя»;
- «Мои бронирования»;
- «Админ-панель».

На главной странице интегрирован баннер, после которого пользователь перенаправляется к каталогу предложений. В левой части экрана расположен адаптивный блок фильтрации, обеспечивающий пользователю возможность оперативно уточнять параметры поиска путем ввода ключевого слова, выбора определенной категории или ограничения ценового диапазона. Правая часть экрана отведена под сетку карточек. Каждый элемент каталога тура включает изображение, описание, метку категории, название, географическую локацию, актуальную цену и дату поездки. Навигационное меню расположено в верхней части интерфейса и является сквозным, обеспечивая постоянный доступ.

Выбор карточки тура инициирует переход пользователя на персональную страницу тура. Данная страница является основным источником детальной информации о туре. Здесь представлена полномасштабная галерея изображений, информация о проживании и подробная программа экскурсий с разбивкой по дням. Справа закреплен коммерческий блок, содержащий точные даты заезда, фиксированную стоимость и функциональную кнопку «Забронировать тур». Ниже основного описания интегрирован блок пользовательских отзывов, который включает оценки и комментарии, способствующие формированию доверия к продукту.

После подтверждения операции бронирования информация регистрируется в личном кабинете пользователя. Раздел «Мои бронирования» выполняет функцию персонализированного отслеживания поездок. Здесь отображается список всех выбранных поездок с привязкой к датам, контактами организатора и текущим статусом («Ожидает», «Подтверждено» или «Отклонено»). Минималистичный дизайн данного раздела направлен на снижение пользовательской неопределенности. Для смены пароля или имени существует раздел «Профиль», в нем также можно сменить главное фото пользователя.

Для администрации веб – платформы разработана изолированная административная панель. Доступ к данной панели строго регламентирован. Архитектура административной панели структурирована посредством

логических вкладок, расположенных в левой боковой панели: «Туры», «Категории», «Провайдеры», «Бронирования» и «Отзывы».

Интерфейс панели управления характеризуется полной интерактивностью. На вкладке «Туры» отображается список существующих предложений с функциями «Изменить» и «Удалить», а также форма для создания новых предложений. В эту форму интегрированы поля для текстового описания, селекторы дат, выпадающие списки категорий и туроператоров, а также область для загрузки изображений. Вкладки «Категории» и «Провайдеры» устроены схожим образом; система предусматривает динамическое расширение базы данных посредством добавления новых провайдеров или категорий через лаконичные текстовые формы. Вкладка «Бронирования» функционирует как интерфейс для управления заказами. Модератору доступны сведения о заявителе и времени оформления заявки, а также функции ее подтверждения или отклонения.

Реализация интерфейсных элементов осуществлена с использованием компонентной архитектуры библиотеки React, что обеспечивает эффективное повторное использование кода [13]. Каждый изолированный компонент инкапсулирует собственное состояние. Этот подход обеспечивает высокую скорость масштабирования кодовой базы и способствует повторному использованию компонентов, тем самым сокращая размер конечного дистрибутива приложения.

2.4 Проектирование классов программных средств для веб – платформы туристического маркетплейса

Диаграмма классов предназначена для представления внутренней структуры программы в виде классов и связей между ними [14]. Диаграмма классов представляет внутреннюю объектную структуру серверного модуля. Архитектура серверной части веб – платформы соответствует классическому паттерну «Model-View-Controller» (MVC). Этот паттерн обеспечивает изоляцию

логики хранения данных от механизмов их обработки, что упрощает рефакторинг и параллельную разработку модулей.

Спроектированная UML-схема приведена на рисунке 16.

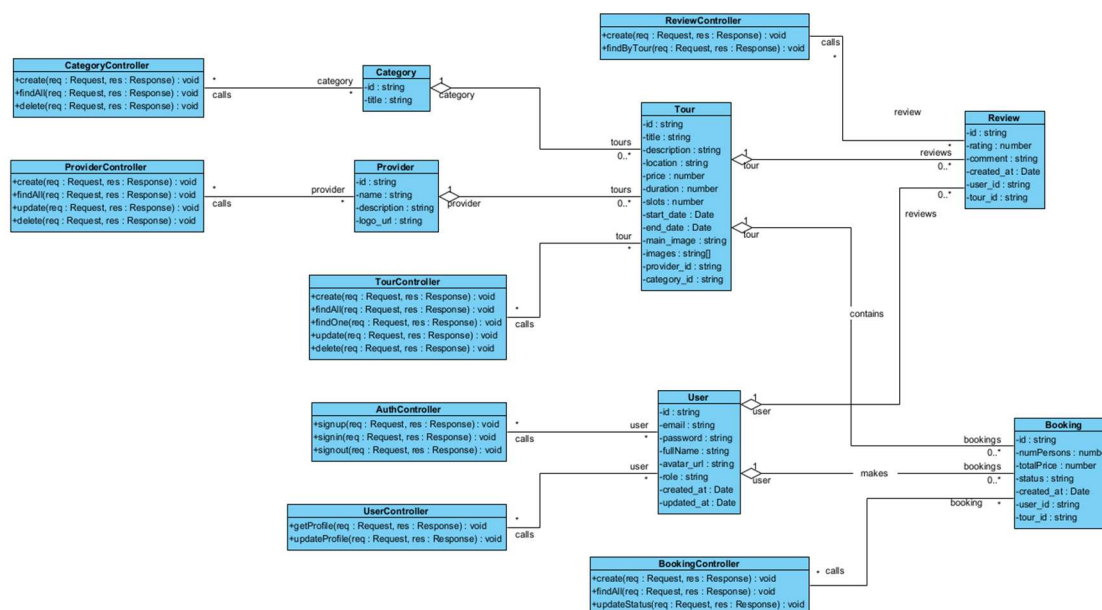


Рисунок 16 – Диаграмма классов программных средств веб – платформы

Зоны ответственности спроектированных классов распределены и детализированы в таблице 5.

Таблица 5 – Компоненты диаграммы классов приложения

Компонент	Назначение
Category	Модель данных для категорий туров (хранит уникальный строковый идентификатор и название категории).
Provider	Модель данных для туроператоров (хранит информацию о названии компании, текстовое описание и ссылку на логотип).
Tour	Модель данных для туристических путевок (содержит сведения о локации, стоимости, длительности, свободных местах, датах проведения и массиве изображений).
User	Модель данных для пользователей платформы (хранит электронную почту, хэш пароля, полное имя, ссылку на аватар и роль в системе).
Booking	Модель данных для заказов (фиксирует количество человек, итоговую стоимость, статус бронирования и дату создания).

Продолжение таблицы 5

Компонент	Назначение
Review	Модель данных для пользовательских отзывов (содержит цифровую оценку-рейтинг, текст комментария и штамп даты публикации).
CategoryController	Контроллер, отвечающий за добавление, просмотр и удаление тематических категорий отдыха.
BookingController	Контроллер для обработки заявок на бронирование и интерактивного изменения их статусов модератором.
ReviewController	Контроллер, реализующий функции добавления новых отзывов и фильтрации комментариев по конкретным турам.
AuthController	Контроллер, реализующий логику аутентификации, регистрации новых пользователей и выхода из учетных записей.
UserController	Контроллер для работы с учетными записями, отвечающий за извлечение и редактирование данных профиля.
TourController	Контроллер CRUD-операций для управления каталогом туров (создание, чтение, обновление, удаление карточек).
ProviderController	Контроллер CRUD-операций для добавления, изменения списков и удаления компаний-туроператоров.

Таким образом, были спроектированы классы программных средств для разрабатываемого веб-приложения, где шесть основных моделей (Category, Provider, User, Tour, Booking, Review) отражают сущности предметной области, а контроллеры (AuthController, TourController и т. д.) реализуют над ними бизнес-логику и CRUD-операции. Таблица помогает быстро понять назначение каждого компонента: модели отвечают за структуру данных, контроллеры – за прием запросов и взаимодействие с данными.

3 Реализация программных средств веб – платформы туристического маркетплейса

3.1 Реализация базы данных веб – платформы

Физическая модель базы данных – это модель данных, которая определяет, каким образом представляются данные, и содержит все детали, необходимые СУБД для создания базы данных. [15]

Построение физического представления базы данных «travelmarketplace» осуществлялось в инструментальной среде Visual Paradigm 18 с использованием методологического стандарта нотации IDEF1X. Данная модель представлена на рисунке 17.

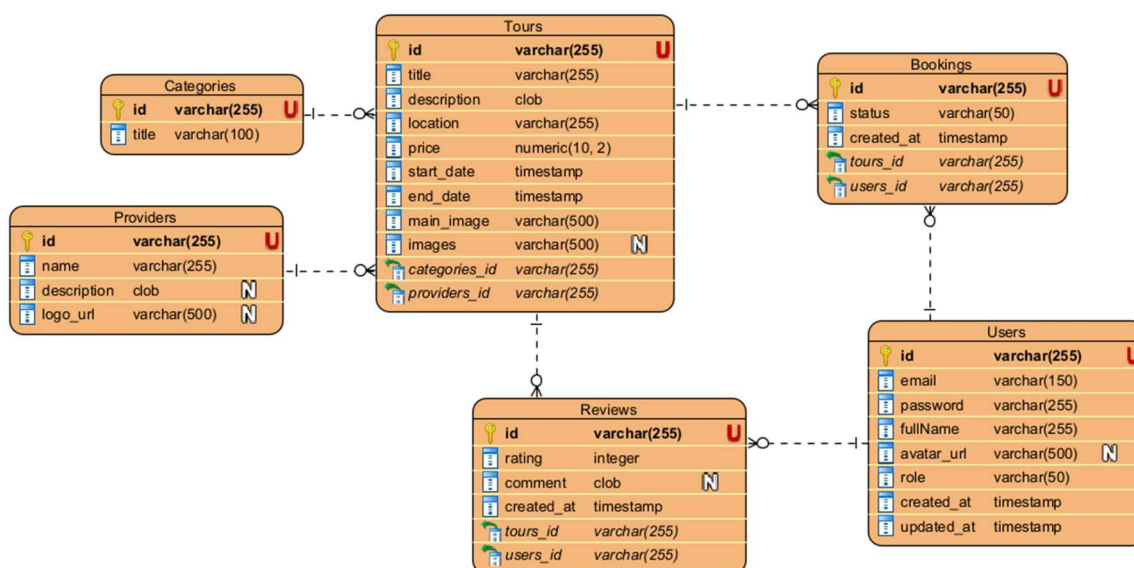


Рисунок 17 – Физическая модель базы данных «travelmarketplace» веб – платформы туристического маркетплейса

Для интерактивного администрирования реляционной базы данных использовался веб-интерфейс облачной платформы Supabase. Этот инструмент предоставляет графический инструментарий для управления миграциями, непосредственной модификации записей в таблицах и отладки изолированных SQL-запросов к СУБД PostgreSQL.

На этапе первичного тестирования функциональности развернутого туристического маркетплейса потребовалась генерация демонстрационных данных. Заполнение целевых таблиц тестовыми сущностями осуществлялось вручную с использованием встроенного редактора СУБД.

Объектно-реляционное отображение (ORM) Prisma выступает в роли связующего звена между реляционными таблицами базы данных и серверной логикой Express. Эта библиотека, предназначенная для платформы Node.js, абстрагирует слой данных до уровня стандартных классов и объектов TypeScript. Таким образом, серверные контроллеры взаимодействуют с информацией без прямого написания низкоуровневого SQL-кода.

Асинхронные потоки данных из хранилища динамически отображаются на пользовательском интерфейсе маркетплейса. Например, при доступе пользователя к главной странице система извлекает доступные тематические направления. Запись новой категории отдыха в базу данных инициируется методом `CategoryController` посредством обращения к Prisma (`prisma.category.create`). Аналогично, чтение и последующий вывод полного списка доступных категорий в навигационном меню обеспечивается методом `prisma.category.findMany` [13].

Безопасность учетных записей модераторов и клиентов обеспечивается криптографическими механизмами. Пароль администратора перед сохранением в таблице 'users' подвергается процедуре одностороннего хеширования с использованием алгоритма `bcrypt`. В процессе аутентификации пользователя серверный модуль перехватывает введенные данные, генерирует проверочный хеш и выполняет его сверку с сохраненным значением. Прямое декодирование или хранение паролей в открытом текстовом виде в СУБД полностью исключено.

3.2 Реализация серверной части веб – платформы

Серверный программный модуль агрегатора туристических предложений реализован на строго типизированном языке программирования TypeScript и функционирует в интегрированной среде разработки Visual Studio Code.

Демонстрационные фрагменты исходного кода серверной части представлены в приложении В. Полная кодовая база проекта доступна в открытом удаленном репозитории на платформе GitHub [16].

Архитектура серверной части основана на событийно-ориентированной платформе Node.js в сочетании с минималистичным веб-фреймворком Express. Такое технологическое решение обеспечивает высокую пропускную способность серверной части при обработке входящих сетевых потоков и предоставляет гибкие средства для маршрутизации асинхронных вызовов [17].

Модульная структура серверного приложения организует логические блоки в специализированных директориях единой кодовой базы:

- «prisma» – содержит декларативную схему данных schema.prisma, которая координирует миграции и генерирует типы для PostgreSQL;
- «src/controllers» – инкапсулирует бизнес-логику серверной части, отвечая за валидацию входящих параметров и формирование ответов;
- «src/routes» – определяет маршруты API, связывая определенные URI-эндпоинты со слоем контроллеров;
- «src/middleware» – организует промежуточные программные фильтры для проверки JWT-токенов, верификации ролей пользователей (клиент/администратор) и централизованной обработки исключений.

В таблице 6 представлен систематизированный перечень ключевых эндпоинтов REST API, обеспечивающих манипуляцию сущностями маркетплейса.

Таблица 6 – Эндпоинты REST API веб – платформы туристического маркетплейса

URL запроса	HTTP метод	Функциональное описание метода
/api/auth/register	POST	Регистрация нового пользовательского аккаунта в системе
/api/auth/login	POST	Аутентификация пользователя с выдачей JWT-токена доступа
/api/auth/logout	POST	Завершение текущей сессии и инвалидация сессионных данных

Продолжение таблицы 6

URL запроса	HTTP метод	Функциональное описание метода
/api/auth/me	GET	Извлечение профиля текущего авторизованного пользователя
/api/tours	POST	Публикация новой карточки тура (доступно администратору)
/api/tours	GET	Извлечение полного каталога туров с поддержкой фильтрации
/api/tours/:id	GET	Получение детальных сведений о конкретном туристическом продукте
/api/tours/:id	PUT	Модификация параметров существующей карточки тура
/api/tours/:id	DELETE	Безвозвратное удаление тура из каталога маркетплейса
/api/categories	POST	Создание новой тематической категории отдыха
/api/categories	GET	Получение полного листинга доступных категорий
/api/categories/:id	DELETE	Удаление выбранной категории из базы данных
/api/providers	POST	Регистрация новой компании-туроператора (провайдера)
/api/providers	GET	Вывод списка всех верифицированных поставщиков услуг
/api/providers/:id	DELETE	Исключение туроператора из партнерской сети платформы
/api/bookings	POST	Оформление и отправка новой заявки на бронирование путевки
/api/bookings	GET	Запрос истории заказов (формируется динамически под роль пользователя)
/api/bookings/:id	PATCH	Интерактивное изменение статуса бронирования модератором
/api/reviews	POST	Добавление текстового отзыва и оценки к совершенной поездке
/api/reviews/tour/:tourId	GET	Извлечение массива комментариев для конкретной карточки тура

Описанные интерфейсы взаимодействия обеспечивают сопряжение реактивного клиентского интерфейса с серверной логикой. Такая интеграция способствует высокой отзывчивости маркетплейса при выполнении пользовательских транзакций.

3.3 Реализация клиентской части веб – платформы

Клиентский модуль веб-платформы реализован на строго типизированном языке программирования TypeScript и функционирует в интегрированной среде разработки Visual Studio Code. Решение основано на компонентной архитектуре пользовательского интерфейса с использованием библиотеки React. Ключевые

демонстрационные фрагменты исходного кода клиентской части представлены в приложении Г. Полная структура интерфейсных скриптов и архитектура клиентской части доступны в открытом удаленном репозитории проекта на платформе GitHub [16].

Процесс трансформации программных модулей, оптимизации исходных скриптов и финальной компоновки интерфейсного пакета в единый дистрибутив выполняется с использованием современного инструмента Vite. Это решение заменяет традиционный сборщик Webpack. Интеграция Vite значительно сокращает время сборки проекта благодаря нативной поддержке модулей и существенно ускоряет процесс отладки компонентов (Hot Module Replacement) в реальном времени.

Статическая структура клиентской части, иерархия интерфейсных слоев и механизмы обмена данными между изолированными модулями представлены на диаграмме компонентов, приведенной ранее на рисунке 18.

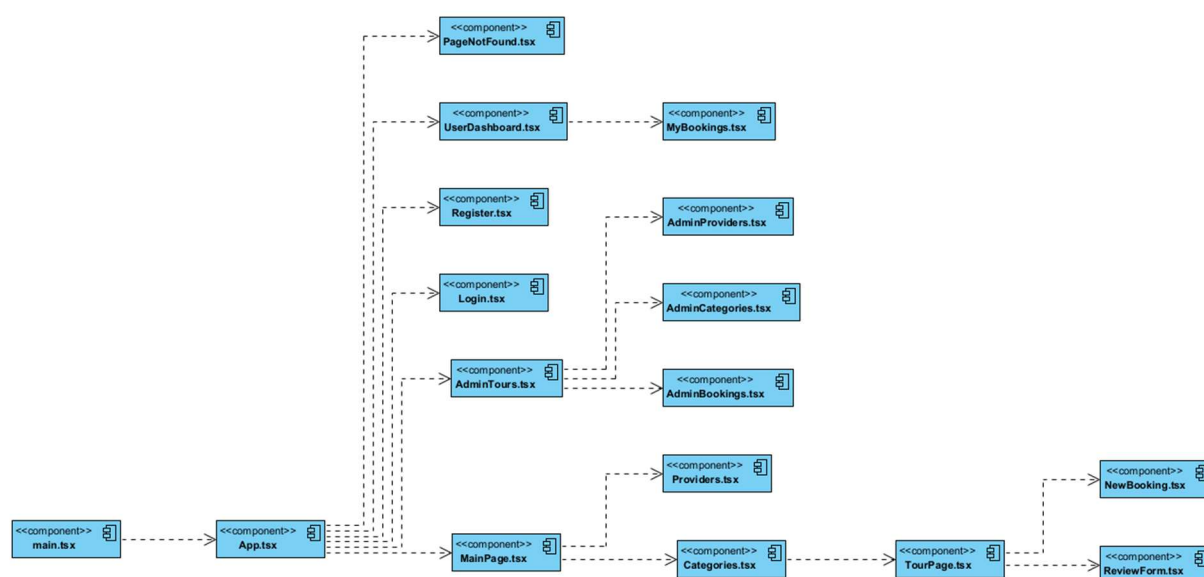


Рисунок 18 – Диаграмма компонентов клиентской части веб – платформы туристического маркетплейса

В таблице 7 приведено описание компонентов клиентской части веб – платформы

Таблица 7 – Компоненты клиентской части веб – платформы

Компонент	Назначение
main.tsx	Входная точка клиентского приложения, обеспечивающая монтирование виртуального дерева React в DOM-структуру браузера.
App.tsx	Корневой программный компонент, инкапсулирующий декларативное описание глобальных маршрутов (Routing) веб-платформы.
PageNotFound.tsx	Компонент визуализации интерфейсной ошибки с кодом 404 при попытке обращения к несуществующему адресу.
Login.tsx	Интерфейсный модуль, реализующий интерактивную форму аутентификации пользователей и модераторов в системе.
Register.tsx	Компонент формы регистрации, собирающий первичные метаданные для генерации новой учетной записи клиента.
MainPage.tsx	Центральный экран маркетплейса, агрегирующий динамическую витрину туров, сайдбар фильтрации и навигационные ссылки.
TourPage.tsx	Компонент детального отображения целевой путевки, содержащий галерею медиафайлов, условия и блок отзывов.
NewBooking.tsx	Интерфейсный элемент, обеспечивающий отправку транзакционных заявок на бронирование выбранного тура.
Categories.tsx	Компонент, отвечающий за вывод полного листинга доступных направлений и тематических категорий отдыха на витрине.
AdminCategories.tsx	Панель управления категориями, предоставляющая администратору функционал добавления и удаления объектов.
Providers.tsx	Модуль отображения партнерской сети верифицированных компаний-туроператоров для публичных пользователей.
AdminProviders.tsx	Специализированный интерфейс CRUD-модификации и верификации новых поставщиков туристических услуг.
AdminTours.tsx	Административный компонент для интерактивного управления каталогом (добавление, редактирование параметров, скрытие карточек туров).
UserDashboard.tsx	Личный кабинет клиента, организующий вывод персональных данных и истории совершенных поездок.
MyBookings.tsx	Компонент визуализации статусов текущих заказов пользователя («Ожидает», «Подтверждено», «Отклонено»).
AdminBookings.tsx	Консоль модератора, реализующая функционал сквозного мониторинга входящих бронирований и оперативного изменения их статусов.
ReviewForm.tsx	Интерактивный интерфейсный блок для отправки текстовых рецензий и выставления оценок-рейтингов после поездки.

Асинхронный сетевой обмен данными между реактивным клиентским интерфейсом и серверным модулем Express обеспечивается библиотекой Axios. Интеграция этого HTTP-клиента в программную среду маркетплейса обусловлена возможностью гибкого конфигурирования перехватчиков (interceptors). Эти механизмы автоматизируют подстановку авторизационных JWT-токенов в заголовки каждого исходящего запроса. Кроме того, библиотека

предоставляет лаконичный синтаксис для централизованной обработки исключений и сетевых ошибок. Такой подход полностью изолирует уровень отправки запросов от визуальной логики TypeScript-компонентов пользовательского интерфейса.

3.4 Реализация интерфейса веб – платформы

Из пункта 2.3 были учтены все требования для реализации интерфейса и были созданы следующие страницы:

- «Авторизация»;
- «Регистрация»;
- «Главная страница»;
- «Страница тура»;
- «Профиль»;
- «Мои бронирования»;
- «Админ-панель».

Взаимодействие с веб – платформой туристического маркетплейса опирается на трехуровневую ролевую модель, которая была спроектирована в пункте 1.3. В нее входят: гость (неавторизованный пользователь), клиент и администратор. Уровень доступа определяет видимость элементов интерфейса. Модель безопасности блокирует попытки обойти лимиты транзакций или открыть защищенные страницы без нужных прав.

Первичная сессия неавторизованного пользователя (гостя) всегда начинается с загрузки главной страницы, выполняющей роль интерактивной витрины маркетплейса. Данная страница предоставляет полный доступ к каталогу туристических продуктов и их описанию, туроператорам и отзывам, однако полностью блокирует транзакционный функционал. Оформить бронирование или оставить отзыв без подтверждения личности невозможно. Для проведения операций система запрашивает обязательный вход в аккаунт или прохождение регистрации.

Несмотря на защитные ограничения, гость может свободно использовать аналитические инструменты платформы. Интерактивный сайдбар с фильтрами позволяет мгновенно отсекаать нерелевантные предложения, фильтруя туры по тематическим категориям или заданному ценовому лимиту. Для точечного и быстрого подбора путевок в шапку фильтрации внедрена система текстового поиска, обрабатывающая ключевые маркеры и названия локаций в режиме реального времени.

Главная страница при первичной сессии гостя представлена на рисунке 19.

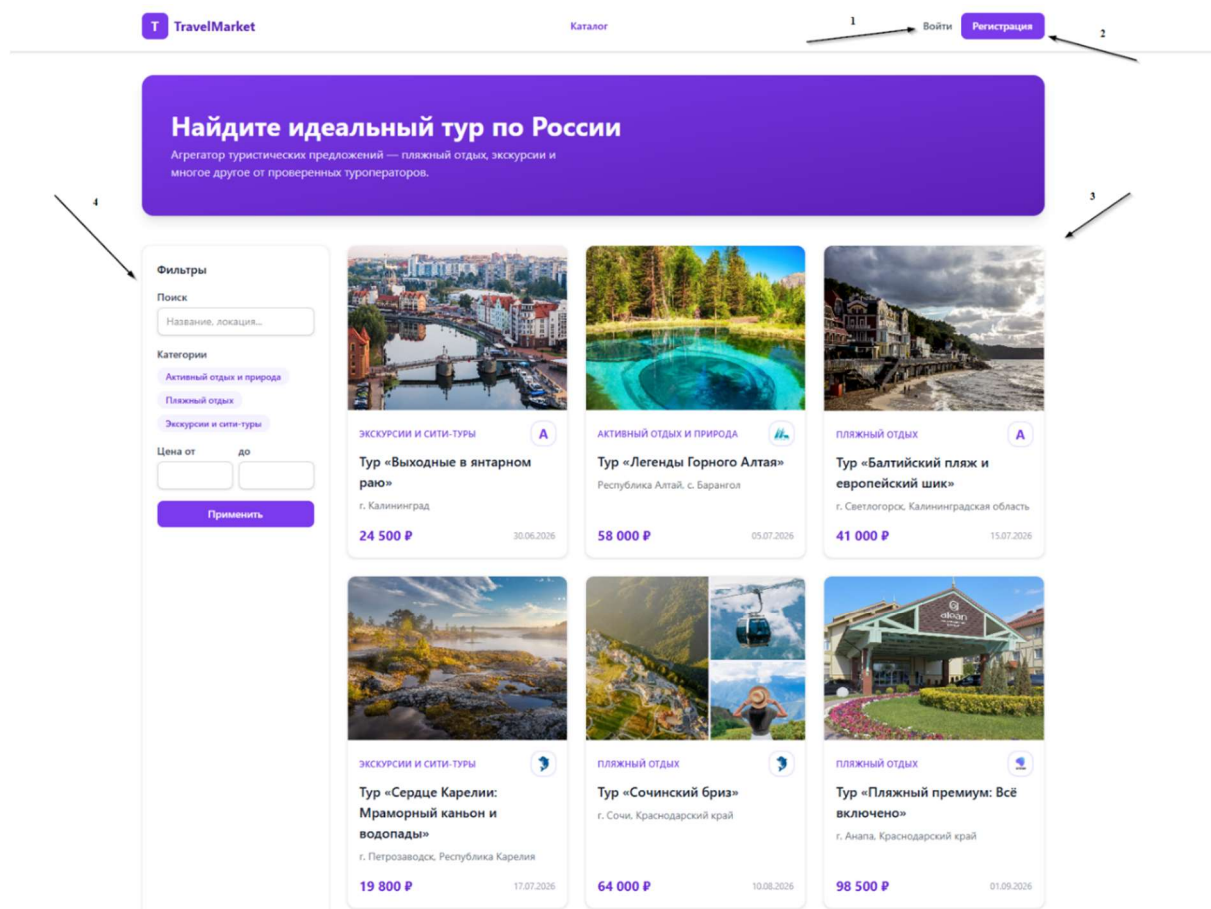


Рисунок 19 – Главная страница при сессии гостя

Рассмотрим основные элементы главной страницы, представленные на рисунке 19:

- 1 – переход на страницу авторизации, кнопка «Войти»;
- 2 – переход на страницу регистрации, кнопка «Регистрация»;
- 3 – каталог туров;
- 4 – фильтрация туров.

Страница конкретного тура при сессии гостя представлена на рисунке 20.

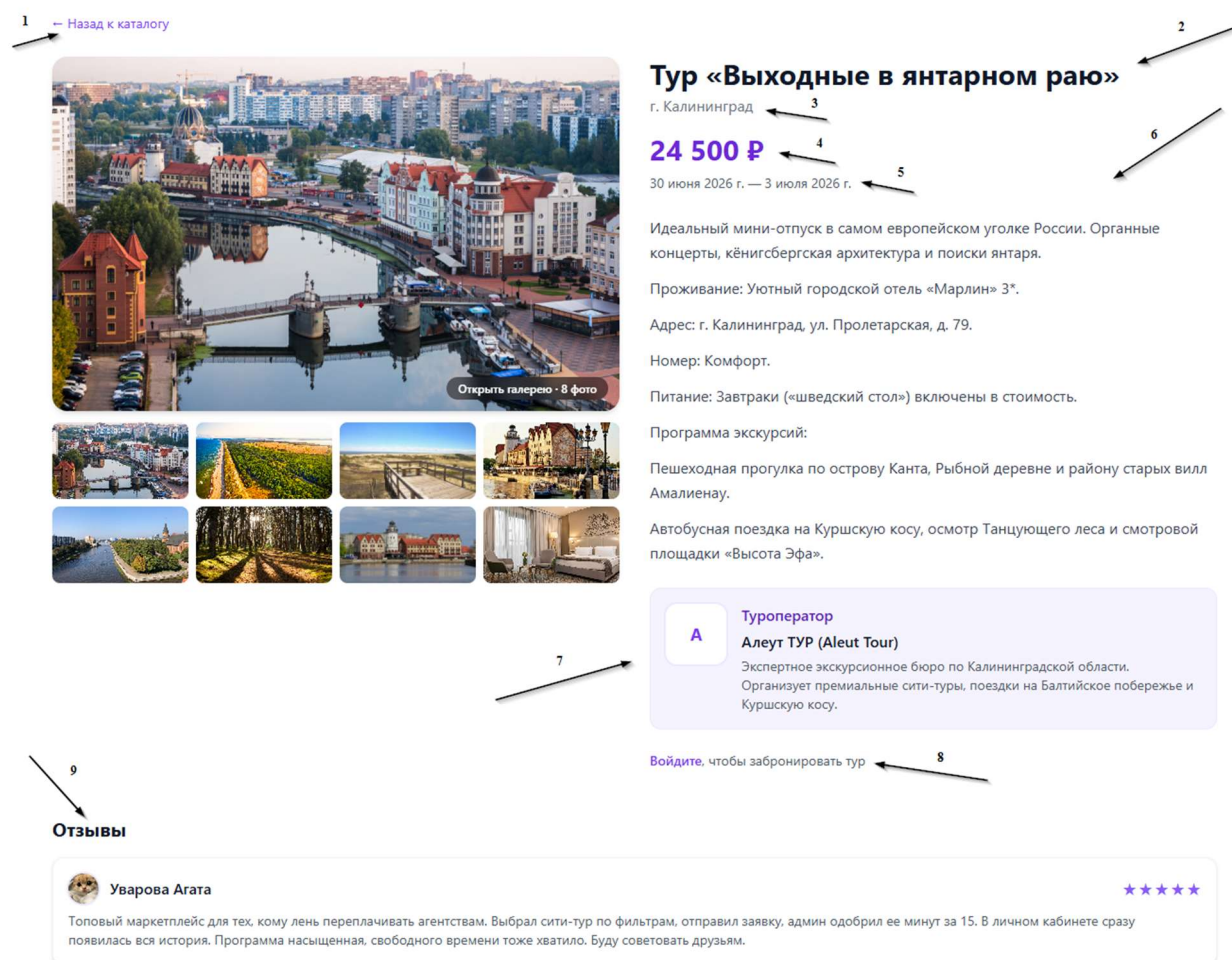


Рисунок 20 – Страница тура при сессии гостя

Рассмотрим основные элементы страницы тура при сессии гостя, представленные на рисунке 20:

- 1 – переход на главную страницу, кнопка «Назад к каталогу»;
- 2 – название тура;
- 3 – местонахождение тура;
- 4 – цена тура;
- 5 – даты проведения тура;
- 6 – описание тура;
- 7 – туроператор;
- 8 – переход на страницу авторизации, кнопка «Войдите»;
- 9 – отзывы на тур;

Переход анонимного гостя в статус полноправного клиента маркетплейса происходит через строго последовательный двухэтапный сценарий. Первичный шаг в этой цепочке – создание постоянного цифрового профиля в системе. Для этого посетитель использует страницу регистрации, структура которой приведена на рисунке 21.

The screenshot shows a registration form titled "Регистрация". Below the title is a link "Уже есть аккаунт? Войти". The form contains three input fields: "Имя" (Name) with the value "Бакаева Алина", "Email" with the value "alina_b@mail.ru", and "Пароль" (Password) with masked characters ".....". Below these fields is a blue button labeled "Создать аккаунт". Four numbered arrows point to the form elements: arrow 1 points to the Name field, arrow 2 points to the Email field, arrow 3 points to the Password field, and arrow 4 points to the "Создать аккаунт" button.

Рисунок 21 – Страница регистрации

Рассмотрим основные элементы страницы регистрации, представленные на рисунке 21:

- 1 – поле ввода имя пользователя;
- 2 – поле ввода почты пользователя;
- 3 – поле ввода пароля пользователя;
- 4 – регистрация в системе, кнопка «Создать аккаунт».

Если пользователь уже зарегистрирован в системе, тогда для аутентификации, ему нужно перейти на страницу авторизации для верификации учетных данных и инициализации рабочей сессии (рисунок 22).

The screenshot shows a login form titled "Вход". Below the title is a link "Нет аккаунта? Зарегистрироваться". The form contains two input fields: "Email" with the value "alina@mail.ru" and "Пароль" (Password) with masked characters ".....". Below these fields is a blue button labeled "Войти". Three numbered arrows point to the form elements: arrow 1 points to the Email field, arrow 2 points to the Password field, and arrow 3 points to the "Войти" button.

Рисунок 22 – Страница авторизации

Рассмотрим основные элементы страницы авторизации, представленные на рисунке 22:

- 1 – поле ввода почты пользователя;
- 2 – поле ввода пароля пользователя;
- 3 – вход в систему, кнопка «Войти».

Успешная аутентификация переключает графическую оболочку маркетплейса в режим зарегистрированного пользователя. Веб – платформа сразу модифицирует верхнюю панель навигации: убирает кнопки входа и регистрации, размещая вместо них аватар профиля, и прямую ссылку на выход из аккаунта (рисунок 23).



Рисунок 23 – Интерфейсный блок идентификации авторизованного пользователя

Помимо вывода блока идентификации (рисунок 23), приложение динамически обновляет список доступных ссылок в верхней панели навигации, открывая пользователю защищенные разделы системы. Актуальный вид обновленной панели навигации со страницами «Каталог», «Мои брони», «Профиль» зафиксирован на рисунке 24.



Рисунок 24 – Верхняя панель при сессии авторизованного пользователя

Страница «Профиль» выступает центральным узлом управления персональными настройками клиента. Приложение реализует здесь функционал редактирования учетных данных в режиме реального времени, обеспечивая синхронизацию изменений с серверной частью системы. Внешний вид данного раздела представлен на рисунке 25.

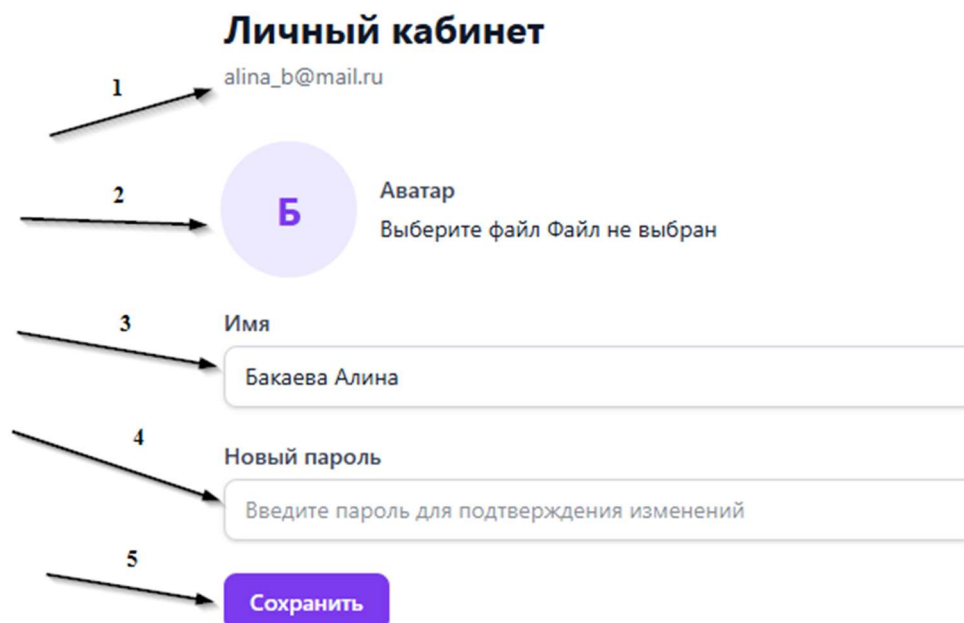


Рисунок 25 – Интерфейс редактирования учетных данных профиля

Рассмотрим основные элементы страницы редактирования профиля, представленные на рисунке 25:

- 1 – почта пользователя;
- 2 – аватар пользователя;
- 3 – поле ввода имя пользователя;
- 4 – поле ввода нового пароля пользователя;
- 5 – сохранение изменений данных пользователя, кнопка «Сохранить»;

При загрузке нового изображения профиля система выполняет его автоматическую подмену в интерфейсе, что визуально подтверждает успешное применение изменений (рисунок 26).



Рисунок 26 – Обновление изображения профиля пользователя

После ввода обновленных параметров и активации команды сохранения, система выводит во всплывающем окне уведомление об успешной обработке запроса (рисунок 27).

A light green rectangular notification box with rounded corners containing the text "Профиль обновлён" in a green sans-serif font.

Рисунок 27 – Уведомление о сохранении внесенных изменений

Взаимодействие с туристическими продуктами на сайте строится на понятных и предсказуемых шагах. Когда клиент находит подходящее предложение в каталоге, он переходит на страницу детального просмотра тура. Для подтверждения своего намерения пользователю достаточно один раз кликнуть по кнопке «Забронировать тур» (рисунок 28).

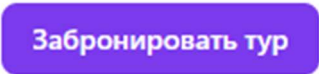
A purple rectangular button with rounded corners and a white border, containing the text "Забронировать тур" in a white sans-serif font.

Рисунок 28 – Кнопка оформления бронирования в карточке тура

При успешном завершении транзакции в верхней части страницы появляется системное сообщение «Заявка на бронирование отправлена!» (рисунок 29). Это уведомление служит подтверждением того, что выбранная поездка успешно зафиксирована в реестре заказов.

A light green rectangular notification box with rounded corners containing the text "Заявка на бронирование отправлена!" in a green sans-serif font.

Рисунок 29 – Уведомление об успешной отправке заявки на бронирование

Далее пользователь может перейти на страницу «Мои бронирования» для контроля актуальных статусов своих заявок. Именно здесь приложение выводит список всех заказов текущего клиента, где у каждой новой записи автоматически проставляется статус «Ожидает» (рисунок 30). Такая отметка наглядно показывает клиенту, что менеджер маркетплейса еще не успел обработать его запрос. На записи также отображается название тура, дата бронирования и цена.

Мои бронирования

Тур «Сочинский бриз»

11.06.2026 · 64 000 Р

Ожидает

Рисунок 30 – Статус заявки в личном кабинете

Клиенту становится доступна опция публикации отзыва. Этот механизм позволяет сформировать объективный рейтинг туристического продукта, основываясь на мнении реальных путешественников.

На рисунке 31 продемонстрирован процесс ввода пользовательского контента. Клиент указывает оценку по пятибалльной шкале и пишет текстовый комментарий.

Оставить отзыв

Оценка

★ ★ ★ ★ ★

Выбрано: 5 из 5

Комментарий

Тур организован на высшем уровне! Программа в Сочи очень насыщенная, всё прошло четко по графику. Большое спасибо организаторам!

Отправить

Рисунок 31 – Форма публикации отзыва тура

Рассмотрим основные элементы формы публикации отзыва, представленные на рисунке 31:

- 1 – пятибалльная оценка;
- 2 – поле ввода комментария пользователя;

3 – публикация отзыва, кнопка «Отправить»;

После клика по кнопке «Отправить» система моментально обрабатывает запрос и выводит уведомление об успехе (рисунок 32).

Отзыв опубликован

Рисунок 32 – Подтверждение успешной публикации отзыва

Обновленный контент подтягивается на страницу тура в секцию отзывов (рисунок 33).

Отзывы

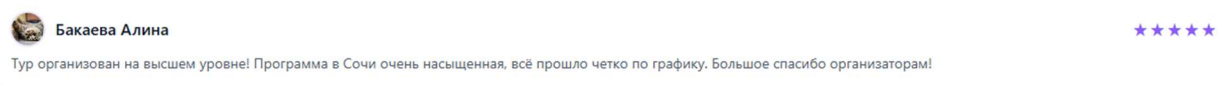


Рисунок 33 – Отображение опубликованного отзыва в карточке тура

Для управления бизнес-процессами маркетплейса предусмотрена панель администратора, доступ к которой получают исключительно авторизованные сотрудники с соответствующими привилегиями. В то время как клиентский интерфейс ориентирован на подбор и бронирование услуг, административная консоль позволяет управлять актуальностью контента и состоянием пользовательских заказов.

В верхней навигационной панели автоматически активируется скрытый ранее раздел «Панель управления» (рисунок 34).

Админ-панель

Рисунок 34 – Админ-панель в навигационной панели при роли администратора

Раздел «Панель управления» обеспечивает доступ к управлению контентом каталога, управлению провайдерами, модерации входящих бронирований и мониторингу отзывов (рисунок 35), что необходимо для поддержания целостности базы данных туристических продуктов.

Админ-панель

Туры

Категории

Провайдеры

Бронирования

Отзывы

Рисунок 34 – Содержимое Админ-панели

Для администрирования контента в системе разработан набор инструментов, позволяющий оперативно управлять базой туров. Процесс работы с данными разбит на несколько этапов.

Создание нового предложения происходит через форму, представленную на рисунке 35. Для внесения правок в уже существующие позиции используется та же форма: при выборе тура система автоматически «подтягивает» в поля ввода актуальную информацию из базы данных, что исключает необходимость ввода всех параметров заново. После внесения изменений и нажатия кнопки «Сохранить» данные в СУБД обновляются.

Рассмотрим основные элементы формы создания и редактирования тура, представленные на рисунке 35:

- 1 – поле ввода названия тура;
- 2 – поле ввода описания тура;
- 3 – поле ввода цены тура на человека;
- 4 – поле ввода локации тура;
- 5 – поле загрузки главного изображения;
- 6 – поле загрузки дополнительных изображений;
- 7 – поле ввода даты начала и окончания тура;
- 8 – поле выбора провайдера;
- 9 – поле выбора категории тура;

10 – сохранение тура в каталог, кнопка «Сохранить»;

11 – отмена сохранения тура, кнопка «Отмена».

Управление турами

The screenshot shows a web form for creating a new tour. The form is titled 'Новый тур' (New tour). It contains several input fields and sections:

- 1** points to the 'Название' (Name) field, which contains 'Тур «Балтийский пляж и европейский шик»'.
- 2** points to the 'Описание' (Description) field, which contains 'Отдых для ценителей мягкого климата, песчаных дюн, крика чаек и освежающего морского бриза в окружении вековых сосен исторического Раушена.'.
- 3** points to the 'Цена' (Price) field, which contains '41000'.
- 4** points to the 'Локация' (Location) field, which contains 'г. Светлогорск, Калининградская область'.
- 5** points to the 'Главное фото *' (Main photo) section, which shows a large image of a beach and a building.
- 6** points to the 'Дополнительные фото' (Additional photos) section, which shows a row of smaller images.
- 7** points to the 'Добавить файлы' (Add files) button in the additional photos section.
- 8** points to the 'Сохранить' (Save) button.
- 9** points to the 'Отмена' (Cancel) button.
- 10** points to the 'Сохранить' (Save) button.
- 11** points to the 'Отмена' (Cancel) button.

Other visible elements include date pickers for 'Дата начала' (Start date) and 'Дата окончания' (End date), a category dropdown set to 'Пляжный отдых' (Beach vacation), and a provider dropdown set to 'Алеут ТУР (Aleut Tour)'.

Рисунок 35 – Форма создания и редактирования тура

Для очистки каталога от неактуальных позиций администратор использует функцию удаления. Чтобы избежать случайных ошибок, каждое действие подтверждается через дополнительное окно. После подтверждения сервер удаляет соответствующую строку из таблицы PostgreSQL, а интерфейс каталога

перестраивается автоматически. Кнопка «Удалить» находится рядом с кнопкой «Изменить» справа от названия карточки тура с ее описанием (рисунок 36).

	Тур «Выходные в янтарном раю» г. Калининград · 24 500 Р	Изменить	Удалить
	Тур «Легенды Горного Алтая» Республика Алтай, с. Барангол · 58 000 Р	Изменить	Удалить
	Тур «Балтийский пляж и европейский шик» г. Светлогорск, Калининградская область · 41 000 Р	Изменить	Удалить
	Тур «Сердце Карелии: Мраморный каньон и водопады» г. Петрозаводск, Республика Карелия · 19 800 Р	Изменить	Удалить
	Тур «Сочинский бриз» г. Сочи, Краснодарский край · 64 000 Р	Изменить	Удалить
	Тур «Пляжный премиум: Всё включено» г. Анапа, Краснодарский край · 98 500 Р	Изменить	Удалить
	Тур «Древние тайны Дагестана» Республика Дагестан (Махачкала / Дербент) · 46 500 Р	Изменить	Удалить

Рисунок 35 – Интерфейс управления списка туров

Для управления категориями туров в административной панели реализован аналогичный механизм, что и для реестра туристических продуктов. Это позволяет администратору поддерживать актуальную структуру каталога без лишних сложностей.

Интерфейс работы с категориями представлен на рисунке 37.

Категории

1 → Название категории 2 → [Добавить](#)

Активный отдых и природа	3 → Изменить	4 → Удалить
Пляжный отдых	Изменить	Удалить
Экскурсии и сити-туры	Изменить	Удалить

5 → (стрелка указывает на таблицу категорий)

Рисунок 37 – Интерфейс управления категориями

Рассмотрим основные элементы интерфейса работы с категориями, представленные на рисунке 37:

- 1 – поле ввода названия категории;
- 2 – добавление категории, кнопка «Добавить»;
- 3 – переход к форме для изменения категории, кнопка «Изменить»;
- 4 – удаление категории тура, кнопка «Удалить»;
- 5 – поле загрузки главного изображения.

Для управления партнерской сетью в админ-панели предусмотрен модуль «Провайдеры» (рисунок 38). Он позволяет централизованно вести базу поставщиков туристических услуг, обеспечивая актуальность информации о них на всех страницах маркетплейса.

Туроператоры

1

Логотип
Выберите файл Файл не выбран

2

Название
Алтай-Актив Тур

3

Описание
Профильный туроператор по Горному Алтаю и Сибири. Специализируется на эко-турах, активном отдыхе, экскурсиях на минивэзах и экспедициях.

Сохранить Отмена

4

5

6

7

8

	Алеан Ведущий национальный туроператор, признанный эксперт в сфере семейного отдыха «Всё включено»...	Изменить	Удалить
	Алеут ТУР (Aleut Tour) Экспертное экскурсионное бюро по Калининградской области. Организует премиальные сити-туры...	Изменить	Удалить
	Алтай-Актив Тур Профильный туроператор по Горному Алтаю и Сибири. Специализируется на эко-турах, активном...	Изменить	Удалить
	Дельфин Крупнейший российский туроператор с 27-летним стажем. Специализируется на внутреннем туризме...	Изменить	Удалить

Рисунок 38 – Интерфейс управления провайдерами

Рассмотрим основные элементы интерфейса работы с провайдерами, представленные на рисунке 38:

- 1 – логотип провайдера;

- 2 – поле ввода названия провайдера;
- 3 – поле ввода описания провайдера;
- 4 – сохранения провайдера, кнопка «Сохранить»;
- 5 – отмена сохранения провайдера, кнопка «Отмена»;
- 6 – список провайдеров;
- 7 – переход к форме для изменения провайдера, кнопка «Изменить»;
- 8 – удаление провайдера, кнопка «Удалить».

Раздел бронирований в административной панели является основным инструментом взаимодействия с клиентами. Интерфейс страницы разработан для оперативной фильтрации и обработки поступающих заявок, что исключает ошибки при ручной работе с данными (рисунок 39).

Централизованное отображение статусов и наличие инструментов быстрого реагирования делают данный модуль эффективным средством для сопровождения полного цикла бронирования – от подачи заявки до подтверждения.

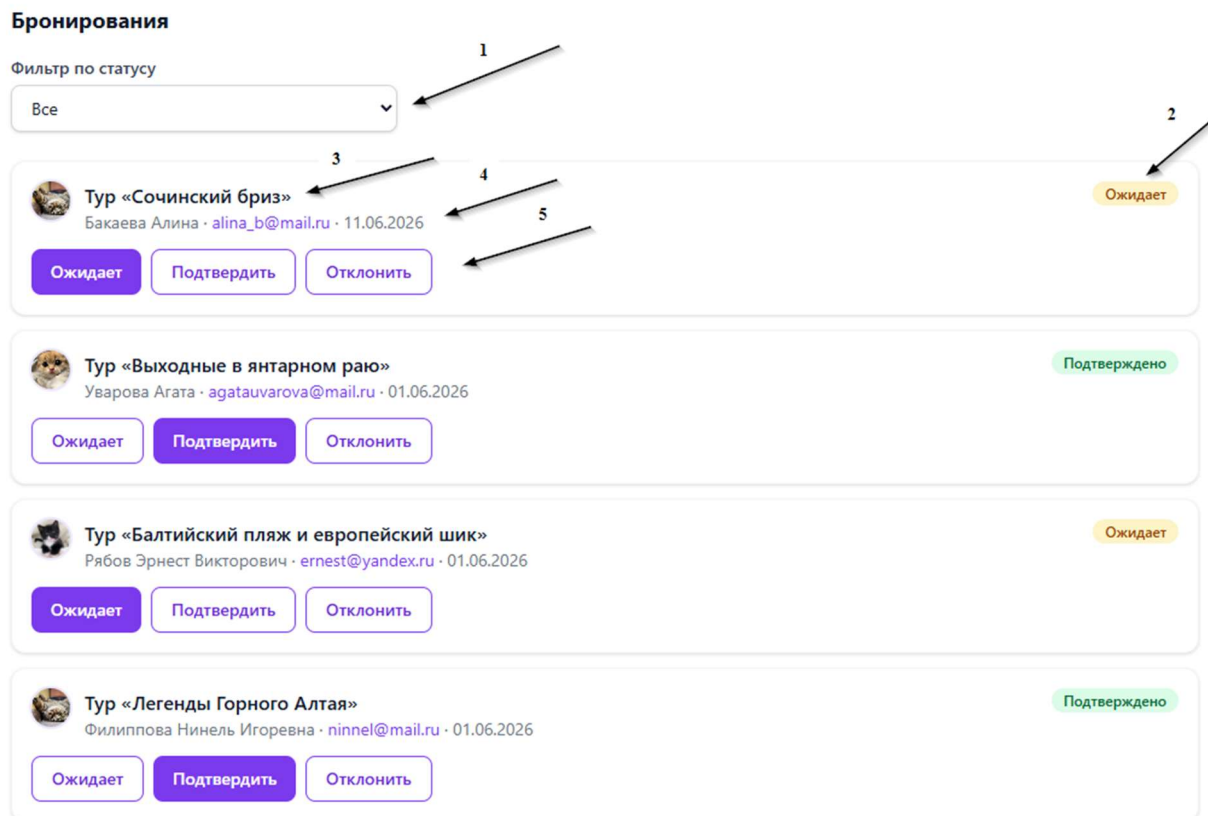


Рисунок 39 – Интерфейс управления бронированием

Рассмотрим основные элементы интерфейса работы с бронированием, представленные на рисунке 39:

- 1 – фильтрация по статусу бронирования;
- 2 – статус бронирования;
- 3 – название тура;
- 4 – информация о клиенте и дата бронирования;
- 5 – кнопки изменения статуса бронирования.

Для контроля качества контента, публикуемого пользователями, в административной панели выделен отдельный раздел «Отзывы» (рисунок 40). Он позволяет просматривать все оставленные комментарии и при необходимости удалять записи, которые не соответствуют правилам площадки.

Подобный инструмент дает возможность следить за актуальностью и достоверностью информации на платформе, исключая спам или некорректные высказывания. Благодаря простоте исполнения, процесс контроля не требует значительных затрат времени, что важно для поддержания репутации маркетплейса.

Модерация отзывов

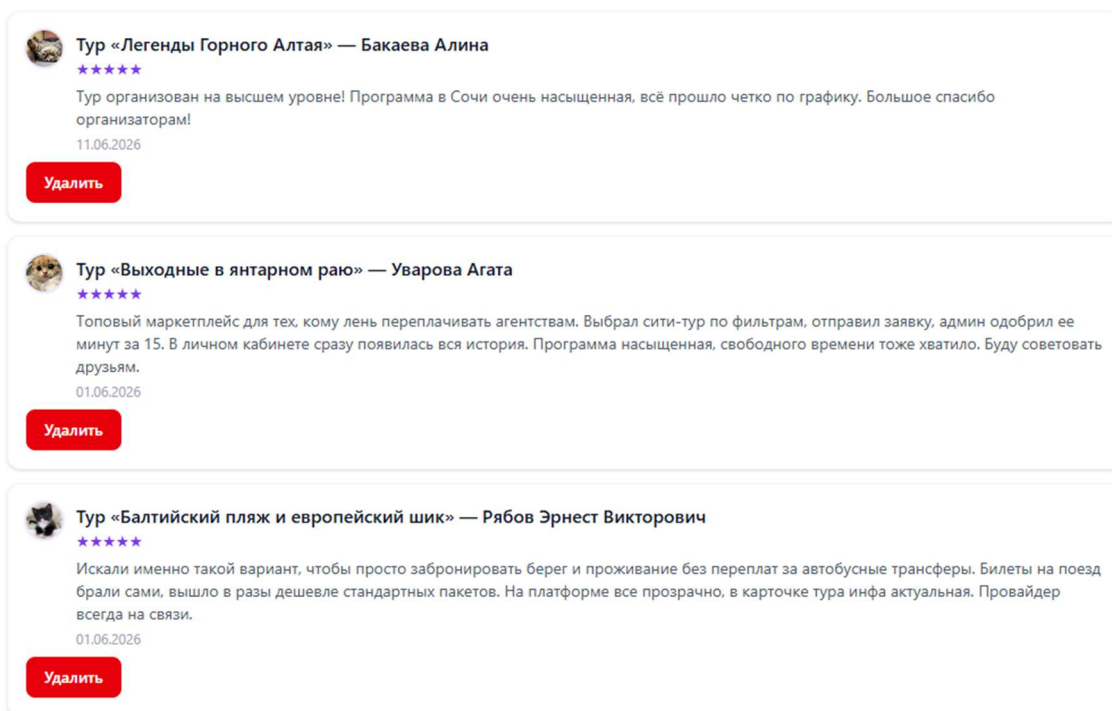


Рисунок 40 – Интерфейс управления отзывами

Разработанная архитектура административной панели и клиентских интерфейсов обеспечивает полный цикл работы с туристическими продуктами: от внесения данных провайдером до публикации отзыва клиентом.

Таким образом, спроектированная функциональность покрывает все заявленные требования к маркетплейсу, создавая надежную основу для дальнейшего масштабирования системы и расширения ассортимента туристических предложений.

4 Контроль качества программного обеспечения

4.1 Обоснования выбора режима тестирования и метрик кода

Тестирование ПО (программное обеспечение) – проверка соответствия между реальным и ожидаемым поведением программы, проводится на наборе тестов, который выбирается некоторым образом [18].

Комплексная проверка системы с применением различных подходов необходима для подтверждения стабильности работы приложения, оценки уровня удобства интерфейса и оперативного выявления программных дефектов. Принято выделять две базовые категории тестирования: функциональное и нефункциональное.

Функциональное тестирование – это методика тестирования программного обеспечения, направленная на проверку соответствия поведения разработанного продукта заданным функциональным требованиям [19].

Нефункциональному тестирование – оценка характеристик системы в специфических условиях эксплуатации. В рамках данной работы особое внимание было уделено кросс-браузерному тестированию. Учитывая специфику проекта, это стало приоритетной задачей, так как стабильное отображение интерфейса во всех популярных браузерах напрямую влияет на восприятие платформы конечным пользователем.

Для углубленного анализа внутренней структуры приложения был проведен аудит метрик программного кода. Использование метрических показателей позволяет объективно оценить техническое состояние продукта, его готовность к масштабированию, а также спрогнозировать дальнейшие трудозатраты на поддержку.

Метрика Холстеда – методика, базирующаяся на математическом подсчете операторов и операндов. Она позволяет формализовать оценку объема программного кода и определить его потенциал для дальнейшего функционального расширения.

Цикломатическая сложность фиксирует количество линейно независимых путей в программных алгоритмах. Данная метрика является индикатором читаемости кода и сложности его отладки, что критически важно для обеспечения долгосрочной поддержки. приложения.

4.2 Тестирование программных средств для веб – платформы туристического маркетплейса

4.2.1 Функциональное тестирование

Для контроля качества был применен метод «черного ящика», позволяющий объективно оценить реакцию интерфейса на действия пользователя и администратора без анализа исходного кода, что подтверждает соответствие системы заявленным требованиям. В качестве ключевых модулей для тестирования были выделены подсистемы бронирований, туристических продуктов и партнерская сеть.

Управление заявками на бронирование реализовано посредством механизма смены состояний, исключающего возможность удаления записи для сохранения истории заказов. Логика переходов представлена на рисунке 41. Сценарий тестирования представлен в таблице 8.

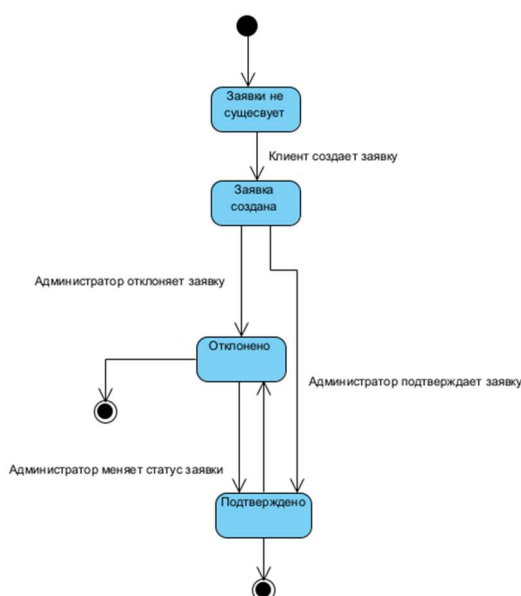


Рисунок 41 – Диаграмма состояний и переходов бронирования

Таблица 8 – Сценарий тестирования модуля бронирований

Назначение	Описание сценария	Результат
Регистрация	Создание новой заявки пользователем	Успешно (рисунок Д.1)
Отображение	Появление заявки на бронирование в «Мои бронирования»	Успешно (рисунок Д.2)
Смена статуса	Переключение состояния (подтвердить, отклонить)	Успешно (рисунок Д.3, рисунок Д.4)

Реестр туров функционирует как справочная подсистема, поддерживающая полный цикл CRUD-операций. Алгоритм управления туристическими продуктами зафиксирован на диаграмме (рисунок 42). Сценарий тестирования карточки тура представлен в таблице 9.

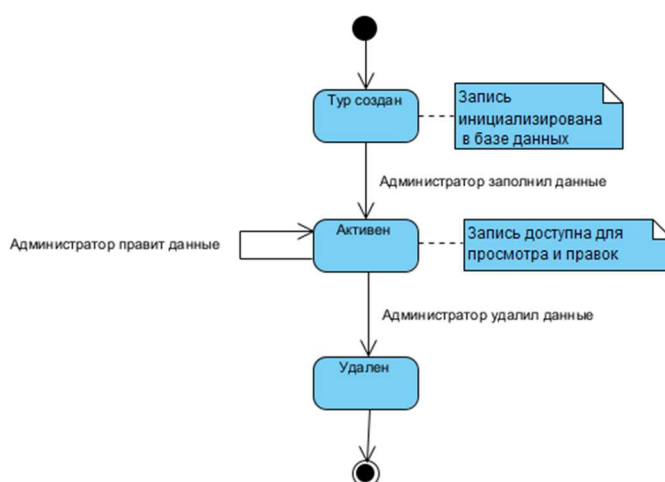


Рисунок 41 – Диаграмма состояний и переходов карточки тура

Таблица 9 – Сценарий тестирования модуля бронирований

Назначение	Описание сценария	Результат
Создание	Создание новой карточки тура, появление в каталоге	Успешно (рисунок Д.5, рисунок Д.6)
Навигация	Переход к главной странице просмотра и поиска туров	Успешно (рисунок Д.7)
Редактирование	Изменение параметров тура администратором	Успешно (рисунок Д.8)

Продолжение таблицы 9

Назначение	Описание сценария	Результат
Удаление	Исключение записи из реестра, получение предупреждения	Успешно (рисунок Д.9, рисунок Д.10)

Реестр провайдеров позволяет централизованно управлять информацией о партнерах. Алгоритм управления провайдерами представлен на диаграмме (рисунок 42). Сценарий тестирования модуля провайдеров представлен в таблице 10.

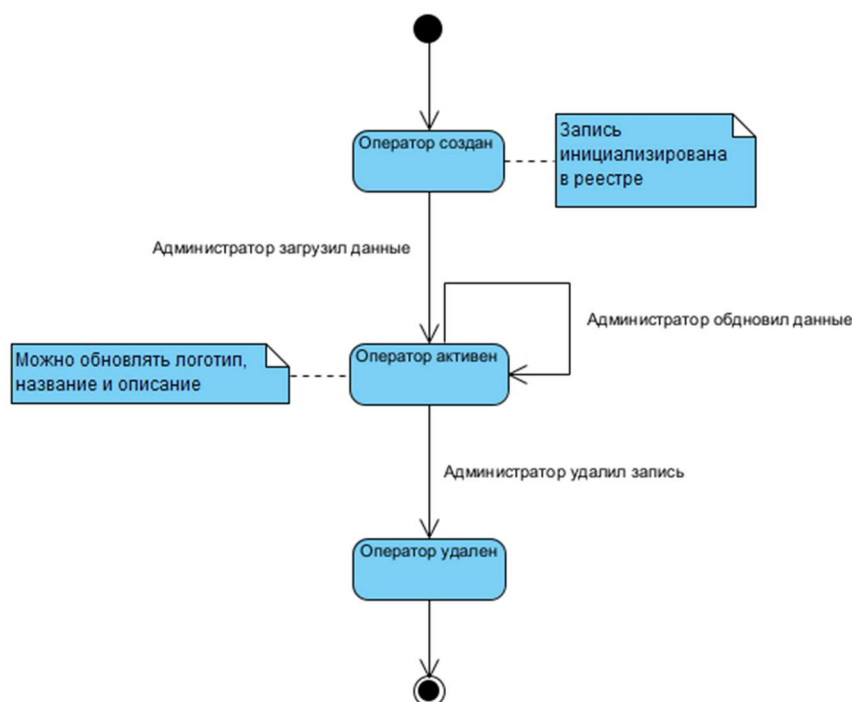


Рисунок 42 – Диаграмма состояний и переходов провайдера

Таблица 10 – Сценарий тестирования модуля бронирований

Назначение	Описание сценария	Результат
Просмотр	Доступ к реестру и списку провайдеров	Успешно (рисунок Д.11)
Создание	Регистрация нового провайдера в базе	Успешно (рисунок Д.12)
Редактирование	Обновление атрибутов компании	Успешно (рисунок Д.13, рисунок Д.14)
Удаление	Удаление провайдера из системы	Успешно (рисунок Д.15)

Данные результаты соответствуют ожидаемым, что свидетельствует о корректной работе веб – платформы туристического маркетплейса.

4.2.2 Кросс-браузерное тестирование

Центральным узлом веб – платформы маркетплейса выступает интерактивный каталог туристических предложений, интегрированный непосредственно в главную страницу сайта. Данное проектное решение направлено на оптимизацию клиентского пути, позволяя пользователю приступить к подбору услуг сразу после загрузки ресурса. Реализация каталога на стеке React требует тщательного контроля за тем, чтобы компоненты интерфейса корректно обрисовывались во всех популярных браузерах: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge и Яндекс.Браузер. Отображение контента может варьироваться в зависимости от движка, на базе которого функционирует браузер:

- Blink – используется в Google Chrome, Microsoft Edge, Opera и Яндекс.Браузере;
- WebKit – обеспечивает работу браузера Safari;
- Gecko – является базовой технологией для Mozilla Firefox.

Для подтверждения высокого качества верстки и стабильности работы React-интерфейса была проведена кросс-браузерная проверка в среде, использующей три различных движка. План проведения данных испытаний представлен в таблице 11, а документальное подтверждение результатов в приложении Е.

Таблица 11 – Матрица кросс-браузерного тестирования каталога туров

Раздел сайта	Браузер	Параметры
Главная (каталог)	Google Chrome	1024 px (рисунок Е1)
Главная (каталог)	Mozilla Firefox	1024 px (рисунок Е2)
Главная (каталог)	Opera	1024 px (рисунок Е3)
Главная (каталог)	Microsoft Edge	1024 px (рисунок Е4)

По итогам верификации подтверждена полная функциональность компонентов каталога в исследуемых браузерах. Интерфейс продемонстрировал необходимую адаптивность, обеспечивая корректное отображение сетки туров и элементов управления независимо от используемого программного обеспечения пользователя.

4.3 Оценка качества программного кода

4.3.1 Метрики Холстеда

Оценка качества программной реализации в рамках разработки веб – платформы была проведена через призму метрик Холстеда. Объектом исследования выступила функция `updateStatus` из модуля администрирования (`AdminBookingsPage.tsx`). Данная функция критически важна для системы, так как она координирует обновление состояний бронирований и обеспечивает актуализацию данных в интерфейсе.

Для расчета показателей был выполнен лексический разбор кода, в ходе которого выделены операторы (конструкции языка) и операнды (переменные и вызовы функций). Результаты подсчета зафиксированы в таблице 12.

Таблица 12 – Операторы и операнды функции `updateStatus`

Операторы	Количество	Операнды	Количество
<code>async</code>	1	<code>updateStatus</code>	1
<code>=></code>	1	<code>id</code>	3
<code>{ }</code>	2	<code>status</code>	3
<code>()</code>	6	<code>bookingsApi</code>	1
<code>try</code>	1	<code>update</code>	1
<code>catch</code>	1	<code>load</code>	1
<code>await</code>	2	<code>err</code>	3
<code>.</code>	3	<code>setError</code>	1
<code>,</code>	1	<code>getErrorMessage</code>	1
<code>:</code>	2	<code>string</code>	1
<code>=</code>	1	<code>BookingStatus</code>	1

Проведенный количественный анализ программного кода позволил оценить архитектурное совершенство модуля `updateStatus`. Итоговые расчеты, представленные в таблице 13, являются математическим подтверждением лаконичности реализации.

Таблица 13 – Результат вычисления метрики Холстеда

Значение	Результат
Словарь программы (n)	40
Длина программы (N)	40
Объем программы (V)	213.1
Сложность программы (D)	2

Полученное значение коэффициента сложности свидетельствует о высокой степени оптимизации логики бронирования. Низкий показатель указывает на то, что алгоритм не перегружен вложенными условными конструкциями и легко поддается рефакторингу. Таким образом, разработанный программный модуль демонстрирует не только функциональную полноту, но и высокий уровень чистоты кода, что является ключевым требованием для обеспечения масштабируемости веб – платформы.

4.3.2 Цикломатическая сложность

В процессе разработки модуля управления туристическими продуктами (`AdminToursPage`) была реализована функция удаления записей, алгоритм работы которой представлен на схеме (рисунок 43). Данная функция представлена в приложении Г.

Взаимодействие с системой начинается с инициации метода `handleDelete`, включающего обязательный этап верификации намерения пользователя. Использование встроенного окна подтверждения предотвращает риск случайного удаления данных. Схема была построена в графической среде Draw.io [20].

Логика обработки запроса включает асинхронное обращение к API-слою (`toursApi.remove`). В случае успешного выполнения транзакции система автоматически выполняет синхронизацию (вызов `load`), обеспечивая актуальное

отображение списка туров в интерфейсе администратора. При возникновении исключительных ситуаций на уровне сетевого взаимодействия, блок обработки ошибок перехватывает исключение, информируя оператора о возникшем сбое. Данная схема обеспечивает консистентность данных и высокую степень отказоустойчивости при администрировании контента.

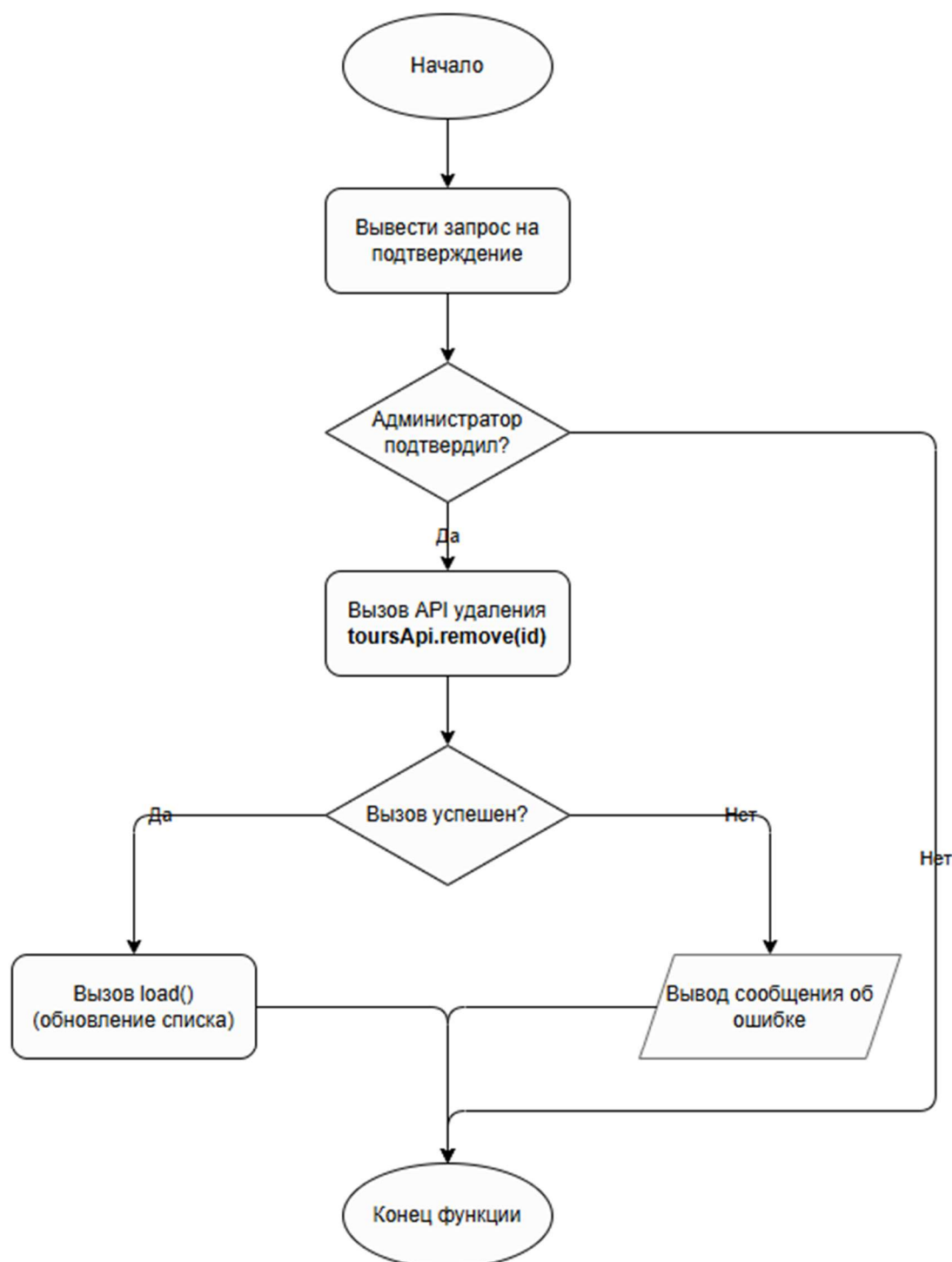


Рисунок 43 – Блок-схема метода handleDelete

Для оценки структурной сложности программного модуля был построен граф потока управления (Control Flow Graph), где узлы представляют базовые блоки кода, а ребра представляют переходы между ними [21]. На рисунке 44 представлены узлы (обозначенные латинскими символами) и направленные ребра, детерминирующие последовательность выполнения инструкций.

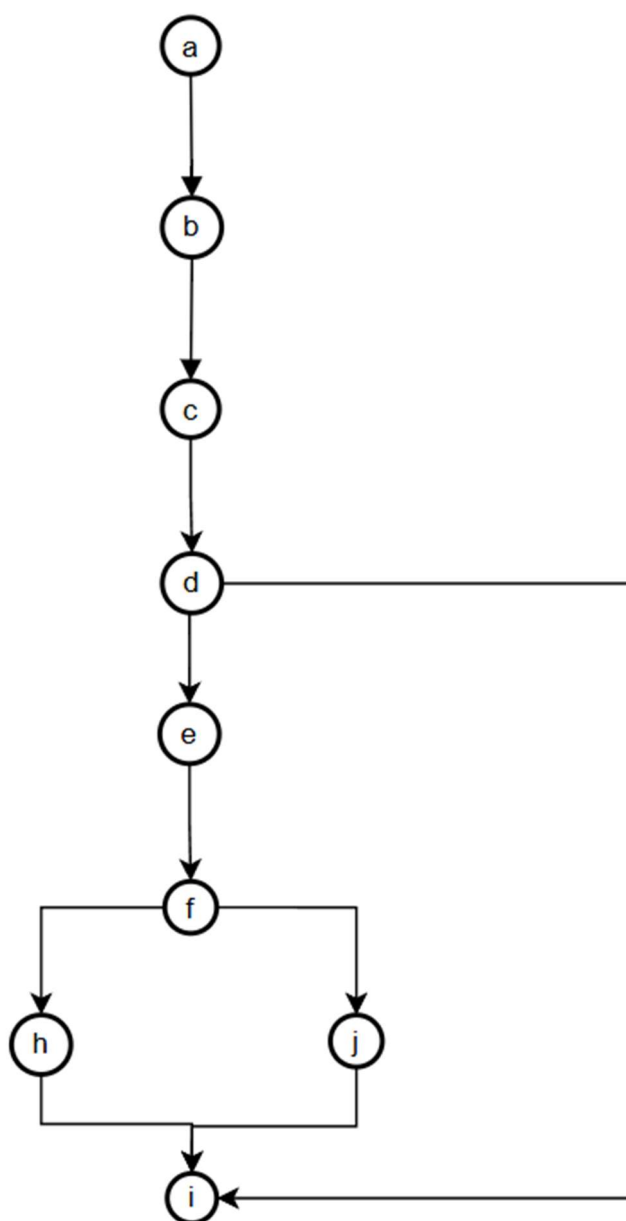


Рисунок 44 – Граф потока управления программного модуля

Анализ количественных характеристик графа позволил вычислить цикломатическую сложность по формуле Маккейба. В исследуемой структуре

зафиксировано 9 узлов и 10 ребер. Принимая во внимание, что граф является связным, расчет сложности выполняется по формуле:

$$V(G) = e - n + 2 \times p,$$

где e – количество ребер в графе потока управления;

n – количество узлов;

p – компонент связности.

Подстановка значений дает результат 3. Полученное значение свидетельствует о высокой степени лаконичности алгоритма. Низкий показатель цикломатической сложности подтверждает оптимальную структуру функции, отсутствие избыточных логических переходов и соответствие кода стандартам простоты поддержки и сопровождения системы веб – платформы туристического маркетплейса.

Заключение

В ходе выполнения выпускной квалификационной работы была спроектирована и реализована веб – платформа туристического маркетплейса, предназначенная для автоматизации процессов бронирования туристических туров и управления административным контентом. Проведен обзор приложений аналогичного назначения и выполнен сравнительный анализ. Выявлены и функциональные и нефункциональные требования к разработке программного обеспечения. Созданы UML-диаграммы вариантов использования и спецификации наиболее значимых бизнес-процессов.

Проведенный анализ предметной области позволил выделить ключевые требования к системе, на основе которых была сформирована высокоуровневая архитектура приложения. Разработка велась с применением современных инструментов статической типизации: использование TypeScript в связке с библиотекой React позволило минимизировать количество ошибок на этапе компиляции и обеспечить строгую типизацию данных на всех уровнях взаимодействия компонентов. Проектирование модели данных было реализовано с применением реляционной СУБД, что обеспечивает высокую целостность и масштабируемость системы. Серверная часть приложения построена на базе среды выполнения Node.js, что гарантирует эффективную обработку запросов.

Экономическое обоснование проекта подтвердило его целесообразность: общая стоимость разработки составила 125 623,89 руб. при совокупных трудозатратах в 279 человеко-часов. Все этапы календарного плана работ, реализованного в период с 8 апреля по 4 июня 2026 года, были соблюдены.

Качество программного продукта верифицировано комплексом функционального и кросс-браузерного тестирования. Метрический анализ кода, включая вычисление метрик Холстеда и цикломатической сложности Маккейба, доказал архитектурную сбалансированность модулей. Строгая типизация, внедренная посредством TypeScript, значительно упростила поддержку кода и повысила его сопровождаемость.

Результатом работы является готовое к эксплуатации веб – приложение, обеспечивающее эффективное взаимодействие между администратором туристических услуг и конечным пользователем. Реализованный функционал управления базой туров и мониторинга статусов бронирований полностью соответствует поставленным целям, а использование передовых технологий веб-разработки обеспечивает соответствие системы современным стандартам индустрии.

Список использованных источников

1. Кущева, Н. Б., Чистяков, М. С. Цифровая трансформация индустрии туризма // Сервис в России и за рубежом. / Н.Б. Кущева, М.С. Чистяков – 2019. – URL: <https://cyberleninka.ru/article/n/tsifrovaya-transformatsiya-v-industrii-turizma> (дата обращения: 01.06.2026).
2. Tourvisor: [Электронный ресурс]. URL: <https://tourvisor.ru/> (дата обращения: 02.06.2026).
3. Travelata: [Электронный ресурс]. URL: <https://travelata.ru/> (дата обращения: 02.06.2026).
4. Диаграмма вариантов использования: учебно-методическое пособие по дисциплине «Современные информационные технологии». – Пенза: Изд-во Пензенского государственного университета. – 14 с.
5. PostgreSQL Wiki: // PostgreSQL Wiki [Электронный ресурс]. URL: https://wiki.postgresql.org/wiki/Main_Page/ru (дата обращения: 03.06.2026).
6. REST API // SKILLFACTORY [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/rest-api/> (дата обращения: 03.06.2026).
7. Построение диаграмм // VISUAL-PARADIGM [Электронный ресурс]. URL: <https://online.visual-paradigm.com/ru/diagrams/> (дата обращения: 03.05.2025).
8. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. –102 с.
9. Катышева, М.А., Кирюхин, Ю.Г., Маслов, В.А. Основы проектной деятельности: Лабораторный практикум / М.А. Катышева, Ю.Г. Кирюхин, В.А. Маслов. – Пенза: Изд-во Пензенского государственного университета, 2022. – 21 с.
10. Создаем диаграмму Ганта // UNISENDER [Электронный ресурс]. URL: <https://www.unisender.com/ru/blog/chto-takoe-diagramma-ganta-i-kak-ee-sdelat/> (Дата обращения: 04.06.2026).

11. Урок по диаграммам развертывания // LUCIDCHART [Электронный ресурс]. URL: <https://www.lucidchart.com/pages/ru/uml-deployment-diagram> (дата обращения: 05.06.2026).
12. Рудь В. Концептуальное, логическое и физическое моделирование данных // MARCUS-AURELIUS [Электронный ресурс]. URL: https://marcus-aurelius.ru/articles/Data_modeling.html (дата обращения 05.06.2026).
13. React Reference: документация библиотеки React // React.dev [Электронный ресурс]. URL: <https://react.dev/reference/react> (дата обращения: 06.06.2026).
14. Использование диаграммы классов UML при проектировании и документировании программного обеспечения // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/572234/> (дата обращения: 06.06.2026).
15. Физическая модель данных // IFMO [Электронный ресурс]. URL: https://neerc.ifmo.ru/wiki/index.php?title=Физическая_модель_базы_данных (дата обращения 07.06.2026).
16. Репозиторий кода TravelMarket// GITHUB. [Электронный ресурс]. URL: <https://github.com/Alina130602/travelmarket> (дата обращения: 10.06.2026).
17. Хавелака, Д. Разработка веб-приложений с помощью Node.js и Express / Д. Хавелака. – Санкт-Петербург, 2021. – 336 с.
18. Теория тестирования ПО просто и понятно // Habr [Электронный ресурс] URL: <https://habr.com/ru/articles/587620/> (дата обращения 10.06.2026)
19. Обзорная статья о видах тестирования IT-продуктов // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/otus/articles/720664/> (дата обращения 10.06.2026)
20. Draw.io // SKILLFACTORY MEDIA [Электронный ресурс]. URL: <https://blog.skillfactory.ru/draw-io/> (дата обращения: 11.06.2025).
21. Код в картинках: визуализация кода // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/742798/> (дата обращения: 11.06.2026).

Приложение А

(справочное)

Глоссарий

БД (База данных) – это имеющая название совокупность данных, которая отражает состояние объектов и их отношений в рассматриваемой предметной области.

СУБД (Система управления базами данных) – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

REST (Representational State Transfer) – это архитектурная концепция, определяющая набор ограничений для построения масштабируемых распределенных систем. Основными принципами REST является ресурсно-ориентированная архитектура, использование стандартных HTTP-методов для манипуляции ресурсами, передача данных без сохранения состояния на сервере, унифицированный интерфейс доступа.

API (Application Programming Interface) – набор готовых классов, функций, процедур, констант и структур, предоставляемых приложением или операционной системой для использования во внешних программных продуктах.

TypeScript – это надстройка над JavaScript, разработанная Microsoft. Главная особенность языка – статическая типизация, которая позволяет находить большинство ошибок еще на этапе написания кода, а не после его запуска в браузере.

React – это JavaScript-библиотека для создания пользовательских интерфейсов.

PostgreSQL – это бесплатная СУБД с открытым исходным кодом. С помощью PostgreSQL можно создавать, хранить базы данных и работать с данными с помощью запросов на языке SQL.

SQL – это декларативный язык программирования, применяющийся для работы с данными в контексте реляционных баз данных.

Фреймворк – программная платформа, задающая архитектурные принципы и общие правила разработки приложений в определенной предметной области.

Веб-браузер – прикладное программное обеспечение для поиска и просмотра web-страниц, а также управления web-приложениями;

Веб – платформа – это техническая база, которая превращает идею в работающий сайт. Она определяет, как вы будете управлять контентом, настраивать дизайн, подключать платежные системы или собирать заявки. Если объяснять, что такое платформа в интернете, простыми словами, можно сравнить ее с каркасом здания.

MVC – это архитектурный паттерн, предписывающий разделение программного кода на три взаимосвязанных компонента: модель данных, представление для визуализации информации и контроллер для обработки пользовательских запросов.

CRUD-операции – это сокращение от четырех основных операций, используемых при работе с данными: Create (создать), Read (прочитать), Update (обновить), Delete (удалить).

Приложение Б

(обязательное)

Атрибуты сущности базы данных

Таблица Б.1 – Атрибуты сущности «users»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность Null	Роль	Пример
id	ID пользователя	text	—	00000000001–99999999999	нет	PK	uuid...
fullName	ФИО	text	—	a-Я a-Z	нет	—	Иванов И.И.
email	Эл. почта	text	—	a-Я a-Z 0–9 ,;./()[]	нет	—	user@mail.ru
password	Пароль	text	—	a-Я a-Z 0–9	нет	—	*****
avatar_url	Ссылка на аватар	text	—	a-Я a-Z 0–9 ,;./()[]	да	—	http://img.com/a
role	Роль	role	—	a-Я	нет	—	CLIENT
created_at	Дата создания	timestamp	—	0–9	нет	—	2026-06-01
updated_at	Дата обновления	timestamp	—	0–9	нет	—	2026-06-15

Таблица Б.2 – Атрибуты сущности «providers»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность Null	Роль	Пример
id	ID провайдера	text	—	00000000001–99999999999	нет	PK	uuid...
name	Название	text	—	a-Я a-Z	нет	—	TravelAgency
description	Описание	text	—	a-Я a-Z 0–9 ,;./()[]	нет	—	Компания...
logo_url	Ссылка на лого	text	—	a-Я a-Z 0–9 ,;./()[]	да	—	http://img.com/l

Таблица Б.3 – Атрибуты сущности «tours»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность Null	Роль	Пример
id	ID тура	text	–	00000000001–99999999999	нет	PK	uuid...
title	Название	text	–	a-Я a-Z 0–9 ,;./()[]	нет	–	Тур в горы
description	Описание	text	–	a-Я a-Z 0–9 ,;./()[]	нет	–	Описание...
location	Локация	text	–	a-Я a-Z	нет	–	Сочи
price	Стоимость	numeric	–	0–9	нет	–	50000
start_date	Дата начала	timestamp	–	0–9	нет	–	2026-07-01
end_date	Дата окончания	timestamp	–	0–9	нет	–	2026-07-10
main_image	Главное фото	text	–	a-Я a-Z 0–9 ,;./()[]	нет	–	http://img.com/1
images	Массив фото	text[]	–	a-Я a-Z 0–9 ,;./()[]	нет	–	{url1, url2}
provider_id	ID провайдера	text	–	00000000001–99999999999	нет	FK	uuid...
category_id	ID категории	text	–	00000000001–99999999999	нет	FK	uuid...

Таблица Б.4 – Атрибуты сущности «categories»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность Null	Роль	Пример
id	ID категории	text	–	00000000001–99999999999	нет	PK	uuid...
title	Название	text	–	a-Я a-Z	нет	–	Пляжный отдых

Таблица Б.5 – Атрибуты сущности «bookings»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность Null	Роль	Пример
id	ID брони	text	—	00000000001–99999999999	нет	PK	uuid...
tour_id	ID тура	text	—	00000000001–99999999999	нет	FK	uuid...
user_id	ID пользователя	text	—	00000000001–99999999999	нет	FK	uuid...
status	Статус	text	—	а-Я	нет	—	ACTIVE
created_at	Дата создания	timestamp	—	0–9	нет	—	2026-06-15

Таблица Б.5 – Атрибуты сущности «reviews»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность Null	Роль	Пример
id	ID отзыва	text	—	00000000001–99999999999	нет	PK	uuid...
rating	Оценка	int4	—	1–5	нет	—	5
comment	Комментарий	text	—	а-Я а-Z 0–9 ,;:/()[]	да	—	Отлично!
tour_id	ID тура	text	—	00000000001–99999999999	нет	FK	uuid...
user_id	ID пользователя	text	—	00000000001–99999999999	нет	FK	uuid...
created_at	Дата создания	timestamp	—	0–9	нет	—	2026-06-20

Приложение В

(обязательное)

Листинг серверной части приложения

Листинг В.1 – Файл auth.controller.ts

```
import { Body, Controller, Post } from '@nestjs/common';
import { ApiOperation, ApiTags } from '@nestjs/swagger';
import { AuthService } from '../auth.service';
import { LoginDto } from '../dto/login.dto';
import { RegisterDto } from '../dto/register.dto';

@ApiTags('auth')
@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('login')
  @ApiOperation({ summary: 'Вход в систему' })
  async login(@Body() dto: LoginDto) {
    return await this.authService.login(dto);
  }

  @Post('register')
  @ApiOperation({ summary: 'Регистрация пользователя' })
  async register(@Body() dto: RegisterDto) {
    return await this.authService.register(dto);
  }
}
```

Листинг В.2 – Файл bookings.controller.ts

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  Query,
  UseGuards,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOperation, ApiTags } from '@nestjs/swagger';
import { BookingsService } from '../bookings.service';
import { CreateBookingDto } from '../dto/create-booking.dto';
import { UpdateBookingDto } from '../dto/update-booking.dto';
import { CurrentUser } from 'src/auth/decorators/current-user.decorator';
import { ActiveUser } from 'src/auth/interfaces/active-user.interface';
import { FilterBookingDto } from '../dto/filter-booking.dto';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';
import { Roles } from 'src/auth/decorators/roles.decorator';
import { RolesGuard } from 'src/auth/guards/roles.guard';
import { Role } from 'src/generated/enums';
```

```

@ApiTags('bookings')
@ApiBearerAuth()
@Controller('bookings')
export class BookingsController {
    constructor(private readonly bookingsService: BookingsService)
    {}

    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @Post()
    @ApiOperation({ summary: 'Создать бронирование (ADMIN)' })
    create(@Body() createBookingDto: CreateBookingDto) {
        return this.bookingsService.create(createBookingDto);
    }

    @UseGuards(JwtAuthGuard)
    @Post('tour/:tourId')
    @ApiOperation({ summary: 'Забронировать тур' })
    bookTour(@Param('tourId') tourId: string, @CurrentUser() user:
ActiveUser) {
        return this.bookingsService.bookTour(user.userId, tourId);
    }

    @UseGuards(JwtAuthGuard)
    @Get('my')
    @ApiOperation({ summary: 'Мои бронирования' })
    myBooks(@CurrentUser() user: ActiveUser) {
        return this.bookingsService.booksByUser(user.userId);
    }

    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @Get('filter')
    @ApiOperation({ summary: 'Фильтр бронирований (ADMIN)' })
    filter(@Query() filter: FilterBookingDto) {
        return this.bookingsService.findAllWithFilter(filter);
    }

    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @Get()
    @ApiOperation({ summary: 'Список бронирований (ADMIN)' })
    findAll() {
        return this.bookingsService.findAll();
    }

    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @Get('/:id')
    @ApiOperation({ summary: 'Получить бронирование по id (ADMIN)'
})
    findOne(@Param('id') id: string) {
        return this.bookingsService.findOne(id);
    }
}

```

```

    }

    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @Patch('/:id')
    @ApiOperation({ summary: 'Обновить бронирование (ADMIN)' })
    update(@Param('id') id: string, @Body() updateBookingDto:
UpdateBookingDto) {
        return this.bookingsService.update(id, updateBookingDto);
    }

    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @Delete('/:id')
    @ApiOperation({ summary: 'Удалить бронирование (ADMIN)' })
    remove(@Param('id') id: string) {
        return this.bookingsService.remove(id);
    }
}

```

Листинг В.3 – Файл catrgories.controller.ts

```

import {
    Controller,
    Get,
    Post,
    Body,
    Patch,
    Param,
    Delete,
    UseGuards,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOperation, ApiTags } from
'@nestjs/swagger';
import { CategoriesService } from '../categories.service';
import { CreateCategoryDto } from '../dto/create-category.dto';
import { UpdateCategoryDto } from '../dto/update-category.dto';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';
import { RolesGuard } from 'src/auth/guards/roles.guard';
import { Roles } from 'src/auth/decorators/roles.decorator';
import { Role } from 'src/generated/enums';

@ApiTags('categories')
@Controller('categories')
export class CategoriesController {
    constructor(private readonly categoriesService:
CategoriesService) {}

    @Post()
    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @ApiBearerAuth()
    @ApiOperation({ summary: 'Создать категорию (ADMIN)' })
    create(@Body() createCategoryDto: CreateCategoryDto) {

```

```

    return this.categoriesService.create(createCategoryDto);
  }

  @Get()
  @ApiOperation({ summary: 'Список категорий' })
  findAll() {
    return this.categoriesService.findAll();
  }

  @Get('/:id')
  @ApiOperation({ summary: 'Получить категорию по id' })
  findOne(@Param('id') id: string) {
    return this.categoriesService.findOne(id);
  }

  @Patch('/:id')
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(Role.ADMIN)
  @ApiBearerAuth()
  @ApiOperation({ summary: 'Обновить категорию (ADMIN)' })
  update(
    @Param('id') id: string,
    @Body() updateCategoryDto: UpdateCategoryDto,
  ) {
    return this.categoriesService.update(id, updateCategoryDto);
  }

  @Delete('/:id')
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(Role.ADMIN)
  @ApiBearerAuth()
  @ApiOperation({ summary: 'Удалить категорию (ADMIN)' })
  remove(@Param('id') id: string) {
    return this.categoriesService.remove(id);
  }
}

```

Листинг В.4 – Файл providers.controller.ts

```

import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  UseGuards,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOperation, ApiTags } from
  '@nestjs/swagger';
import { ProvidersService } from '../providers.service';
import { CreateProviderDto } from '../dto/create-provider.dto';
import { UpdateProviderDto } from '../dto/update-provider.dto';

```

```

import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';
import { RolesGuard } from 'src/auth/guards/roles.guard';
import { Roles } from 'src/auth/decorators/roles.decorator';
import { Role } from 'src/generated/enums';

@ApiTags('providers')
@Controller('providers')
export class ProvidersController {
    constructor(private readonly providersService: ProvidersService)
    {}

    @Post()
    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @ApiBearerAuth()
    @ApiOperation({ summary: 'Создать провайдера (ADMIN)' })
    create(@Body() createProviderDto: CreateProviderDto) {
        return this.providersService.create(createProviderDto);
    }

    @Get()
    @ApiOperation({ summary: 'Список провайдеров' })
    findAll() {
        return this.providersService.findAll();
    }

    @Get('/:id')
    @ApiOperation({ summary: 'Получить провайдера по id' })
    findOne(@Param('id') id: string) {
        return this.providersService.findOne(id);
    }

    @Patch('/:id')
    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @ApiBearerAuth()
    @ApiOperation({ summary: 'Обновить провайдера (ADMIN)' })
    update(
        @Param('id') id: string,
        @Body() updateProviderDto: UpdateProviderDto,
    ) {
        return this.providersService.update(id, updateProviderDto);
    }

    @Delete('/:id')
    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @ApiBearerAuth()
    @ApiOperation({ summary: 'Удалить провайдера (ADMIN)' })
    remove(@Param('id') id: string) {
        return this.providersService.remove(id);
    }
}

```

Листинг В.5 – Файл reviews.controller.ts

```

import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  UseGuards,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOperation, ApiTags } from
 '@nestjs/swagger';
import { ReviewsService } from '../reviews.service';
import { CreateReviewDto } from '../dto/create-review.dto';
import { UpdateReviewDto } from '../dto/update-review.dto';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';
import { RolesGuard } from 'src/auth/guards/roles.guard';
import { Roles } from 'src/auth/decorators/roles.decorator';
import { WriteReviewDto } from '../dto/write-review.dto';
import { CurrentUser } from 'src/auth/decorators/current-
user.decorator';
import { ActiveUser } from 'src/auth/interfaces/active-
user.interface';
import { Role } from 'src/generated/enums';

@ApiTags('reviews')
@Controller('reviews')
export class ReviewsController {
  constructor(private readonly reviewsService: ReviewsService) {}

  @Post('write')
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(Role.ADMIN, Role.USER)
  @ApiBearerAuth()
  @ApiOperation({ summary: 'Написать отзыв на тур' })
  writeReview(@Body() dto: WriteReviewDto, @CurrentUser() user:
ActiveUser) {
    return this.reviewsService.writeReview(user.userId, dto);
  }

  @Get('tour/:tourId/:order')
  @ApiOperation({ summary: 'Отзывы тура (order: asc | desc)' })
  findByTour(
    @Param('tourId') tourId: string,
    @Param('order') order: 'asc' | 'desc',
  ) {
    return this.reviewsService.findByTour(tourId, order);
  }

  @Get()
  @UseGuards(JwtAuthGuard, RolesGuard)

```

```

@Roles(Role.ADMIN)
@ApiBearerAuth()
@ApiOperation({ summary: 'Список отзывов (ADMIN)' })
findAll() {
    return this.reviewsService.findAll();
}

@Post()
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(Role.ADMIN)
@ApiBearerAuth()
@ApiOperation({ summary: 'Создать отзыв (ADMIN)' })
create(@Body() createReviewDto: CreateReviewDto) {
    return this.reviewsService.create(createReviewDto);
}

@Get('/:id')
@ApiOperation({ summary: 'Получить отзыв по id' })
findOne(@Param('id') id: string) {
    return this.reviewsService.findOne(id);
}

@Patch('/:id')
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(Role.ADMIN)
@ApiBearerAuth()
@ApiOperation({ summary: 'Обновить отзыв (ADMIN)' })
update(@Param('id') id: string, @Body() updateReviewDto:
UpdateReviewDto) {
    return this.reviewsService.update(id, updateReviewDto);
}

@Delete('/:id')
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(Role.ADMIN)
@ApiBearerAuth()
@ApiOperation({ summary: 'Удалить отзыв (ADMIN)' })
remove(@Param('id') id: string) {
    return this.reviewsService.remove(id);
}
}

```

Листинг В.6 – Файл tours.controller.ts

```

import {
    Controller,
    Get,
    Post,
    Body,
    Patch,
    Param,
    Delete,
    Query,
    UseGuards,

```

```

} from '@nestjs/common';
import { ApiBearerAuth, ApiOperation, ApiTags } from
 '@nestjs/swagger';
import { ToursService } from '../tours.service';
import { CreateTourDto } from '../dto/create-tour.dto';
import { UpdateTourDto } from '../dto/update-tour.dto';
import { FilterTourDto } from '../dto/filter-tour.dto';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';
import { RolesGuard } from 'src/auth/guards/roles.guard';
import { Roles } from 'src/auth/decorators/roles.decorator';
import { Role } from 'src/generated/enums';

@ApiTags('tours')
@Controller('tours')
export class ToursController {
  constructor(private readonly toursService: ToursService) {}

  @Post()
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(Role.ADMIN)
  @ApiBearerAuth()
  @ApiOperation({ summary: 'Создать тур (ADMIN)' })
  create(@Body() createTourDto: CreateTourDto) {
    return this.toursService.create(createTourDto);
  }

  @Get()
  @ApiOperation({ summary: 'Список туров' })
  findAll() {
    return this.toursService.findAll();
  }

  @Get('search/:searchTerm')
  @ApiOperation({ summary: 'Поиск туров с фильтрами' })
  async search(@Query() dto: FilterTourDto, @Param('searchTerm')
term: string) {
    return await this.toursService.findWithFilter(term, dto);
  }

  @Get('/:id')
  @ApiOperation({ summary: 'Получить тур по id' })
  findOne(@Param('id') id: string) {
    return this.toursService.findOne(id);
  }

  @Patch('/:id')
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(Role.ADMIN)
  @ApiBearerAuth()
  @ApiOperation({ summary: 'Обновить тур (ADMIN)' })
  update(@Param('id') id: string, @Body() updateTourDto:
UpdateTourDto) {
    return this.toursService.update(id, updateTourDto);
  }

```

```

    }

    @Delete('/:id')
    @UseGuards(JwtAuthGuard, RolesGuard)
    @Roles(Role.ADMIN)
    @ApiBearerAuth()
    @ApiOperation({ summary: 'Удалить тип (ADMIN)' })
    remove(@Param('id') id: string) {
        return this.toursService.remove(id);
    }
}

```

Листинг В.7 – Файл user.controller.ts

```

import {
    Controller,
    Get,
    Post,
    Body,
    Patch,
    Param,
    Delete,
    UseGuards,
    ForbiddenException,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOperation, ApiTags } from
'@nestjs/swagger';
import { UsersService } from '../users.service';
import { CreateUserDto } from '../dto/create-user.dto';
import { UpdateUserDto } from '../dto/update-user.dto';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';
import { Roles } from 'src/auth/decorators/roles.decorator';
import { EditProfileDto } from '../dto/edit-profile.dto';
import { CurrentUser } from 'src/auth/decorators/current-
user.decorator';
import { ActiveUser } from 'src/auth/interfaces/active-
user.interface';
import { RolesGuard } from 'src/auth/guards/roles.guard';
import { Role } from 'src/generated/enums';

@ApiTags('users')
@ApiBearerAuth()
@UseGuards(JwtAuthGuard, RolesGuard)
@Controller('users')
export class UsersController {
    constructor(private readonly usersService: UsersService) {}

    @Post()
    @Roles(Role.ADMIN)
    @ApiOperation({ summary: 'Создать пользователя (ADMIN)' })
    create(@Body() createUserDto: CreateUserDto) {
        return this.usersService.create(createUserDto);
    }
}

```

```

@Get()
@Roles(Role.ADMIN)
@ApiOperation({ summary: 'Список пользователей (ADMIN)' })
findAll() {
    return this.userService.findAll();
}

@Patch('profile')
@ApiOperation({ summary: 'Редактировать свой профиль' })
async editProfile(
    @Body() dto: EditProfileDto,
    @CurrentUser() user: ActiveUser,
) {
    return await this.userService.editProfile(user.userId, dto);
}

@Get('/:id')
@Roles(Role.ADMIN, Role.USER)
@ApiOperation({ summary: 'Получить пользователя по id' })
findOne(@Param('id') id: string, @CurrentUser() user:
ActiveUser) {
    if (user.role !== Role.ADMIN && user.userId !== id) {
        throw new ForbiddenException('Можно просматривать только
свой профиль');
    }
    return this.userService.findOne(id);
}

@Patch('/:id')
@Roles(Role.ADMIN)
@ApiOperation({ summary: 'Обновить пользователя (ADMIN)' })
update(@Param('id') id: string, @Body() updateUserDto:
UpdateUserDto) {
    return this.userService.update(id, updateUserDto);
}

@Delete('/:id')
@Roles(Role.ADMIN)
@ApiOperation({ summary: 'Удалить пользователя (ADMIN)' })
remove(@Param('id') id: string) {
    return this.userService.remove(id);
}
}
}

```

Приложение Г

(обязательное)

Листинг клиентской части приложения

Листинг Г.1 – Файл HomePage.tsx

```
import { useCallback, useEffect, useState } from 'react';
import { categoriesApi, getErrorMessage, toursApi } from
'../lib/api';
import type { Category, Tour } from '../types';
import { TourCard } from '../components/TourCard';
import { LoadingSpinner } from '../components/LoadingSpinner';
import { Input } from '../components/ui/Input';
import { Button } from '../components/ui/Button';

export function HomePage() {
  const [tours, setTours] = useState<Tour[]>([]);
  const [categories, setCategories] = useState<Category[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [search, setSearch] = useState('');
  const [selectedCategories, setSelectedCategories] =
useState<string[]>([]);
  const [minPrice, setMinPrice] = useState('');
  const [maxPrice, setMaxPrice] = useState('');

  useEffect(() => {
    categoriesApi
      .list()
      .then(({ data }) => setCategories(data))
      .catch(() => {});
  }, []);

  const loadTours = useCallback(async () => {
    setLoading(true);
    setError('');
    try {
      const term = search.trim();
      const hasFilter =
        selectedCategories.length > 0 || minPrice || maxPrice;
      const filter = {
        ...(selectedCategories.length && { categoryIds:
selectedCategories }),
        ...(minPrice && { minPrice }),
        ...(maxPrice && { maxPrice }),
      };

      if (term) {
        const { data } = await toursApi.search(term, hasFilter ?
filter : undefined);
        setTours(data);
      } else if (hasFilter) {
        const { data } = await toursApi.list();
        setTours(
          data.filter((t) => {
            if (
              selectedCategories.length &&
```

```

        !selectedCategories.includes(t.categoryId)
      ) {
        return false;
      }
      const price = parseFloat(t.price);
      if (minPrice && price < parseFloat(minPrice)) return
false;
      if (maxPrice && price > parseFloat(maxPrice)) return
false;
      return true;
    })),
  );
} else {
  const { data } = await toursApi.list();
  setTours(data);
}
} catch (e) {
  setError(getErrorMessage(e));
} finally {
  setLoading(false);
}
}, [search, selectedCategories, minPrice, maxPrice]);

useEffect(() => {
  loadTours();
}, [loadTours]);

const toggleCategory = (id: string) => {
  setSelectedCategories((prev) =>
    prev.includes(id) ? prev.filter((c) => c !== id) : [...prev,
id],
  );
};

```

```

<div className="grid grid-cols-2 gap-2">
  <Input
    label="Цена от"
    type="number"
    min={0}
    value={minPrice}
    onChange={(e) => setMinPrice(e.target.value)}
  />
  <Input
    label="до"
    type="number"
    min={0}
    value={maxPrice}
    onChange={(e) => setMaxPrice(e.target.value)}
  />
</div>

<Button className="w-full" onClick={loadTours}>
  Применить

```

```

        </Button>
      </aside>

      <div className="flex-1">
        {error && (
          <p className="mb-4 rounded-lg bg-red-50 px-4 py-3
text-sm text-red-700">
            {error}
          </p>
        )}
        {loading ? (
          <LoadingSpinner />
        ) : tours.length === 0 ? (
          <p className="py-12 text-center text-gray-500">Типы не
найденны</p>
        ) : (
          <div className="grid gap-6 sm:grid-cols-2 xl:grid-
cols-3">
            {tours.map((tour) => (
              <TourCard key={tour.id} tour={tour} />
            ))}
          </div>
        )}
      </div>
    </div>
  </div>
);
}

```

Листинг Г.2 – Файл MyBookingPage.tsx

```

import { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { bookingsApi, getErrorMessage, toursApi } from
'../lib/api';
import type { Booking, Tour } from '../types';
import { LoadingSpinner } from '../components/LoadingSpinner';
import { StatusBadge } from '../components/StatusBadge';
import { formatDateShort, formatPrice } from '../lib/format';

type BookingWithTour = Booking & { tour?: Tour };

export function MyBookingsPage() {
  const [bookings, setBookings] = useState<BookingWithTour[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  useEffect(() => {
    bookingsApi
      .my()
      .then(async ({ data }) => {
        const withTours = await Promise.all(
          data.map(async (b) => {
            try {

```

```

        const { data: tour } = await toursApi.get(b.tourId);
        return { ...b, tour };
      } catch {
        return b;
      }
    })),
  );
  setBookings(withTours);
})
.catch((e) => setError(getErrorMessage(e)))
.finally(() => setLoading(false));
}, []);

if (loading) return <LoadingSpinner />;

return (
  <div className="mx-auto max-w-3xl px-4 py-8 sm:px-6">
    <h1 className="text-2xl font-bold text-gray-900">Мои
    бронирования</h1>

    {error && (
      <p className="mt-4 rounded-lg bg-red-50 px-4 py-2 text-sm
text-red-700">
        {error}
      </p>
    )}

    {bookings.length === 0 ? (
      <p className="mt-8 text-center text-gray-500">
        У вас пока нет бронирований.{' '}
        <Link to="/" className="text-brand-600 hover:underline">
          Выбрать тип
        </Link>
      </p>
    ) : (
      <ul className="mt-6 space-y-4">
        {bookings.map((b) => (
          <li
            key={b.id}
            className="flex flex-col gap-2 rounded-xl border
border-gray-100 bg-white p-5 shadow-sm sm:flex-row sm:items-center
sm:justify-between"
          >
            <div>
              {b.tour ? (
                <Link
                  to={` /tours/${b.tour.id}`}
                  className="font-semibold text-brand-700
hover:underline"
                >
                  {b.tour.title}
                </Link>
              ) : (

```

```

        <span className="text-gray-600">Typ
#{b.tourId.slice(0, 8)}</span>
    ))
    <p className="text-sm text-gray-500">
      {formatDateShort(b.createdAt)}
      {b.tour && ` · ${formatPrice(b.tour.price)} `}
    </p>
  </div>
  <StatusBadge status={b.status} />
</li>
  )})
</ul>
  )}
</div>
);
}

```

Листинг Г.3 – Файл AdminBookingsPage.tsx

```

import { useCallback, useEffect, useState } from 'react';
import { bookingsApi, getErrorMessage } from '../lib/api';
import type { Booking, BookingStatus } from '../types';
import { LoadingSpinner } from '../components/LoadingSpinner';
import { StatusBadge } from '../components/StatusBadge';
import { Select } from '../components/ui/Select';
import { Button } from '../components/ui/Button';
import { formatDateShort } from '../lib/format';
import { UserAvatar } from '../components/UserAvatar';

export function AdminBookingsPage() {
  const [bookings, setBookings] = useState<Booking[]>([]);
  const [statusFilter, setStatusFilter] = useState<BookingStatus |
''>('');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  const load = useCallback(async () => {
    setLoading(true);
    setError('');
    try {
      const { data } = statusFilter
        ? await bookingsApi.filter({ status: statusFilter })
        : await bookingsApi.list();
      setBookings(data);
    } catch (e) {
      setError(getErrorMessage(e));
    } finally {
      setLoading(false);
    }
  }, [statusFilter]);

  useEffect(() => {
    load();
  }, [load]);
}

```

```

const updateStatus = async (id: string, status: BookingStatus)
=> {
  try {
    await bookingsApi.update(id, { status });
    await load();
  } catch (err) {
    setError(getErrorMessage(err));
  }
};

if (loading) return <LoadingSpinner />;

return (
  <div>
    <h2 className="text-lg font-bold text-gray-
900">Бронирования</h2>

    <div className="mt-4 max-w-xs">
      <Select
        label="Фильтр по статусу"
        value={statusFilter}
        onChange={(e) =>
          setStatusFilter(e.target.value as BookingStatus | '')
        }
      >
        <option value="">Все</option>
        <option value="PENDING">Ожидает</option>
        <option value="APPROVED">Подтверждено</option>
        <option value="REJECTED">Отклонено</option>
      </Select>
    </div>

    {error && (
      <p className="mt-2 rounded-lg bg-red-50 px-3 py-2 text-sm
text-red-700">
        {error}
      </p>
    )}

    <ul className="mt-6 space-y-3">
      {bookings.map((b) => (
        <li
          key={b.id}
          className="rounded-xl border border-gray-100 bg-white
p-4 shadow-sm"
        >
          <div className="flex flex-wrap items-start justify-
between gap-2">
            <div className="flex gap-3">
              <UserAvatar
                src={b.user?.avatarUrl}
                name={b.user?.fullName ?? 'Пользователь'}

```

```

        size="sm"
      />
    <div>
      <p className="font-medium">
        {b.tour?.title ?? `Typ ${b.tourId.slice(0,
8)}` }`
      </p>

      { /* Обновленная строка с выводом Имени, Email и
Даты */ }

      <div className="text-sm text-gray-500">
        <span>{b.user?.fullName ?? b.userId}</span>
        {b.user?.email && (
          <>
            { ' . ' }
            <a
              href={`mailto:${b.user.email}`}
              className="text-brand-600
hover:underline"
              onClick={ (e) => e.stopPropagation() }
            >
              {b.user.email}
            </a>
          </>
        ) }
        { ' . ' }
        <span>{formatDateShort(b.createdAt)}</span>
      </div>

    </div>
  </div>
  <StatusBadge status={b.status} />
</div>
<div className="mt-3 flex flex-wrap gap-2">
  {[('PENDING', 'APPROVED', 'REJECTED') as
BookingStatus[]).map(
    (s) => (
      <Button
        key={s}
        variant={b.status === s ? 'primary' :
'secondary'}
        onClick={() => updateStatus(b.id, s)}
      >
        {s === 'PENDING'
          ? 'Ожидает'
          : s === 'APPROVED'
            ? 'Подтвердить'
            : 'Отклонить'}
      </Button>
    ),
  ) }
</div>
</li>

```

```

    ))}
  </ul>
</div>
);
}

```

Листинг Г.4 – Файл AdminToursPage.tsx

```

import { useCallback, useEffect, useState } from 'react';
import {
  categoriesApi,
  getErrorMessage,
  providersApi,
  toursApi,
} from '../../lib/api';
import type { Category, Provider, Tour } from '../../types';
import { formatPrice } from '../../lib/format';
import { tourDateFromApi, tourDateToApi } from '../../lib/dates';
import { resolveMediaUrl } from '../../lib/media';
import { ImageUploadField } from
  '../../components/ImageUploadField';
import { ImageGalleryUpload } from
  '../../components/ImageGalleryUpload';
import { LoadingSpinner } from '../../components/LoadingSpinner';
import { Button } from '../../components/ui/Button';
import { Input } from '../../components/ui/Input';
import { Textarea } from '../../components/ui/Textarea';
import { Select } from '../../components/ui/Select';

const emptyForm = {
  title: '',
  description: '',
  price: '',
  location: '',
  mainImage: '',
  extraImages: [] as string[],
  startDate: '',
  endDate: '',
  categoryId: '',
  providerId: '',
};

export function AdminToursPage() {
  const [tours, setTours] = useState<Tour[]>([]);
  const [categories, setCategories] = useState<Category[]>([]);
  const [providers, setProviders] = useState<Provider[]>([]);
  const [loading, setLoading] = useState(true);
  const [form, setForm] = useState(emptyForm);
  const [editingId, setEditingId] = useState<string | null>(null);
  const [error, setError] = useState('');
  const [saving, setSaving] = useState(false);

  const load = useCallback(async () => {
    setLoading(true);

```

```

    try {
      const [t, c, p] = await Promise.all([
        toursApi.list(),
        categoriesApi.list(),
        providersApi.list(),
      ]);
      setTours(t.data);
      setCategories(c.data);
      setProviders(p.data);
    } catch (e) {
      setError(getErrorMessage(e));
    } finally {
      setLoading(false);
    }
  }, []);

  useEffect(() => {
    load();
  }, [load]);

  const resetForm = () => {
    setForm(emptyForm);
    setEditingId(null);
  };

  const startEdit = (tour: Tour) => {
    setEditingId(tour.id);
    setForm({
      title: tour.title,
      description: tour.description,
      price: String(tour.price),
      location: tour.location,
      mainImage: tour.mainImage,
      extraImages: tour.images.filter((url) => url !==
tour.mainImage),
      startDate: tourDateFromApi(tour.startDate),
      endDate: tourDateFromApi(tour.endDate),
      categoryId: tour.categoryId,
      providerId: tour.providerId,
    });
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!form.mainImage) {
      setError('Загрузите главное фото тура');
      return;
    }
    if (form.endDate < form.startDate) {
      setError('Дата окончания не может быть раньше даты начала');
      return;
    }
  }

```

```

    const images =
      form.extraImages.length > 0 ? form.extraImages :
[form.mainImage];

    setSaving(true);
    setError('');
    const payload = {
      title: form.title,
      description: form.description,
      price: form.price,
      location: form.location,
      mainImage: form.mainImage,
      images,
      startDate: tourDateToApi(form.startDate),
      endDate: tourDateToApi(form.endDate),
      categoryId: form.categoryId,
      providerId: form.providerId,
    };
    try {
      if (editingId) {
        await toursApi.update(editingId, payload);
      } else {
        await toursApi.create(payload);
      }
      resetForm();
      await load();
    } catch (err) {
      setError(getErrorMessage(err));
    } finally {
      setSaving(false);
    }
  };

  const handleDelete = async (id: string) => {
    if (!confirm('Удалить тур?')) return;
    try {
      await toursApi.remove(id);
      await load();
    } catch (err) {
      setError(getErrorMessage(err));
    }
  };

  if (loading) return <LoadingSpinner />;

  return (
    <div>
      <h2 className="text-lg font-bold text-gray-900">Управление
турами</h2>

      {error && (
        <p className="mt-2 rounded-lg bg-red-50 px-3 py-2 text-sm
text-red-700">

```

```

        {error}
      </p>
    ) }

    <form
      onSubmit={handleSubmit}
      className="mt-4 space-y-3 rounded-xl border border-brand-
100 bg-brand-50/50 p-4"
    >
      <h3 className="font-medium">
        {editingId ? 'Редактировать тур' : 'Новый тур'}
      </h3>
      <Input
        label="Название"
        value={form.title}
        onChange={(e) => setForm({ ...form, title:
e.target.value })}
        required
      />
      <Textarea
        label="Описание"
        rows={3}
        value={form.description}
        onChange={(e) => setForm({ ...form, description:
e.target.value })}
        required
      />
      <div className="grid gap-3 sm:grid-cols-2">
        <Input
          label="Цена"
          value={form.price}
          onChange={(e) => setForm({ ...form, price:
e.target.value })}
          required
        />
        <Input
          label="Локация"
          value={form.location}
          onChange={(e) => setForm({ ...form, location:
e.target.value })}
          required
        />
      </div>

      <ImageUploadField
        label="Главное фото"
        value={form.mainImage}
        onChange={(mainImage) => setForm({ ...form, mainImage
}})
        required
      />

      <ImageGalleryUpload

```

```

        label="Дополнительные фото"
        values={form.extraImages}
        onChange={({extraImages}) => setForm({ ...form,
extraImages })}
      />

      <div className="grid gap-3 sm:grid-cols-2">
        <div className="space-y-1">
          <Input
            label="Дата начала"
            type="date"
            value={form.startDate}
            onChange={(e) => setForm({ ...form, startDate:
e.target.value })}
            required
          />
          <p className="text-xs text-gray-500">
            На сервер: ISO 8601 (UTC), напр. 2025-06-
15T00:00:00.000Z
          </p>
        </div>
        <div className="space-y-1">
          <Input
            label="Дата окончания"
            type="date"
            value={form.endDate}
            onChange={(e) => setForm({ ...form, endDate:
e.target.value })}
            required
            min={form.startDate || undefined}
          />
        </div>
      </div>
      <div className="grid gap-3 sm:grid-cols-2">
        <Select
          label="Категория"
          value={form.categoryId}
          onChange={(e) => setForm({ ...form, categoryId:
e.target.value })}
          required
        >
          <option value="">Выберите</option>
          {categories.map((c) => (
            <option key={c.id} value={c.id}>
              {c.title}
            </option>
          ))}
        </Select>
        <Select
          label="Провайдер"
          value={form.providerId}
          onChange={(e) => setForm({ ...form, providerId:
e.target.value })}

```

```

        required
      >
        <option value="">Выберите</option>
        {providers.map((p) => (
          <option key={p.id} value={p.id}>
            {p.name}
          </option>
        ))}
      </Select>
    </div>
    <div className="flex gap-2">
      <Button type="submit" loading={saving}>
        {editingId ? 'Сохранить' : 'Создать'}
      </Button>
      {editingId && (
        <Button type="button" variant="ghost"
onClick={resetForm}>
          Отмена
        </Button>
      )}
    </div>
  </form>
</div>
    <div>
      <p className="font-medium">{t.title}</p>
      <p className="text-sm text-gray-500">
        {t.location} · {formatPrice(t.price)}
      </p>
    </div>
  </div>
  <div className="flex gap-2">
    <Button variant="secondary" onClick={() =>
startEdit(t)}>
      Изменить
    </Button>
    <Button variant="danger" onClick={() =>
handleDelete(t.id)}>
      Удалить
    </Button>
  </div>
</li>
  )}
</ul>
</div>
);
}

```

Приложение Д

(рекомендуемое)

Результаты функционального тестирования

Заявка на бронирование отправлена!

Рисунок Д.1 – Уведомление об успешной отправке заявки

Мои бронирования

Тур «Сочинский бриз»
11.06.2026 · 64 000 ₽

Ожидает

Рисунок Д.2 – Появление заявки в личном кабинете

Бронирования

Фильтр по статусу

Все



Тур «Сочинский бриз»

Бакаева Алина · alina_b@mail.ru · 11.06.2026

Отклонено

Ожидает

Подтвердить

Отклонить

Рисунок Д.3 – Смена статуса бронирования «Отклонить»

Бронирования

Фильтр по статусу

Все



Тур «Сочинский бриз»

Бакаева Алина · alina_b@mail.ru · 11.06.2026

Подтверждено

Ожидает

Подтвердить

Отклонить

Рисунок Д.4 – Смена статуса бронирования «Подтвердить»

Управление турами

Новый тур

Название

Тур «Легенды Алтайских гор: конный поход»

Описание

Захватывающее семидневное путешествие по нетронутым тропам Алтая. Мы пройдем через горные перевалы, увидим бирюзовые озера и остановимся в юртах под звездным небом. Программа включает обучение верховой езде и вечерние посиделки у костра.

Цена

65000

Локация

Республика Алтай, Усть-Коксинский район

Главное фото *



Выбрать файл

или кликните сюда и нажмите Ctrl+V

Дополнительные фото

Добавить файлы

несколько файлов или Ctrl+V (можно несколько картинок)

Дата начала

01.08.2026

Дата окончания

07.08.2026

На сервер: ISO 8601 (UTC), напр. 2025-06-15T00:00:00.000Z

Категория

Активный отдых и природа

Провайдер

Алтай-Актив Тур

Создать

Рисунок Д.5 – Создание новой карточки тура

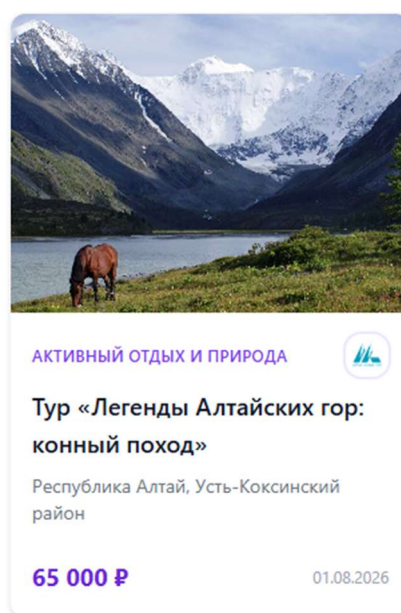


Рисунок Д.6 – Новая карточка тура в каталоге

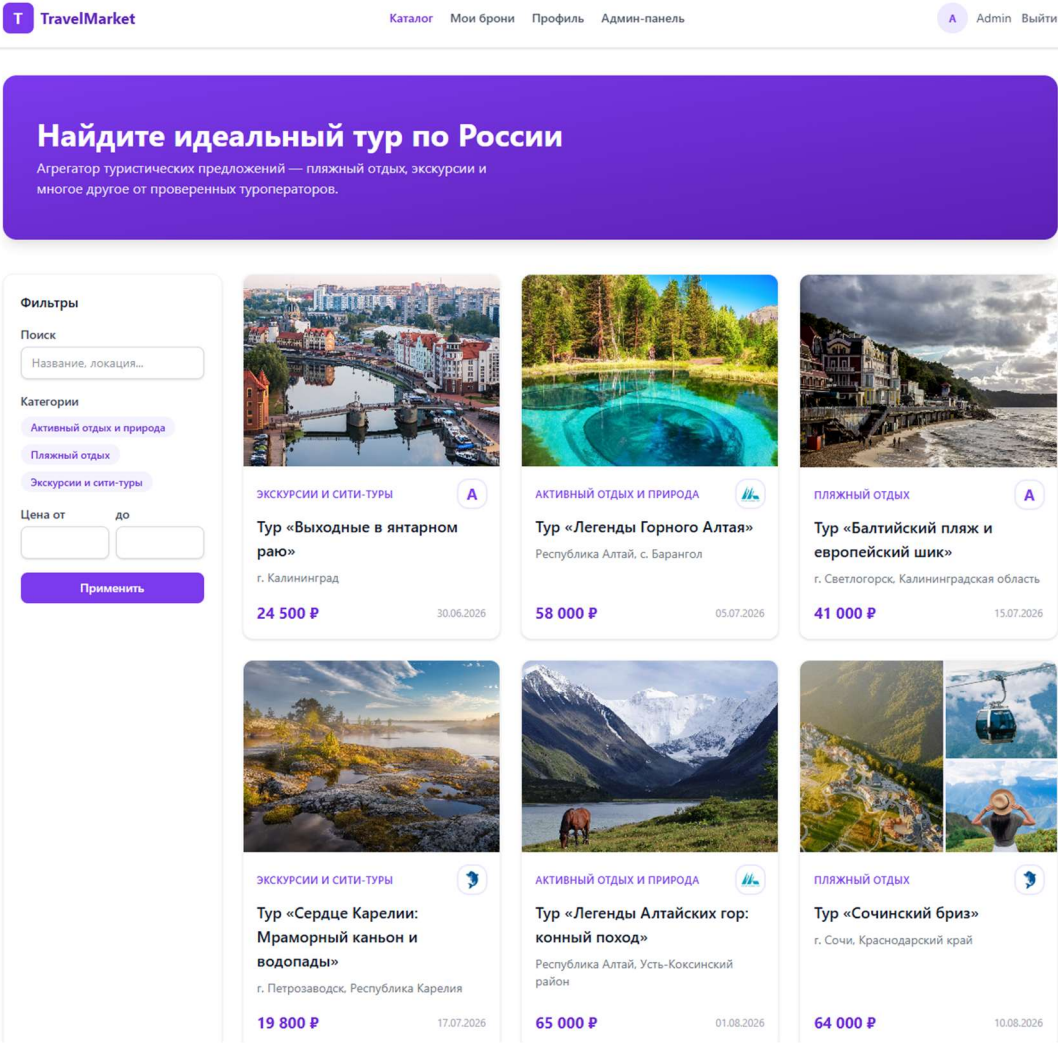


Рисунок Д.7 – Переход к главной странице

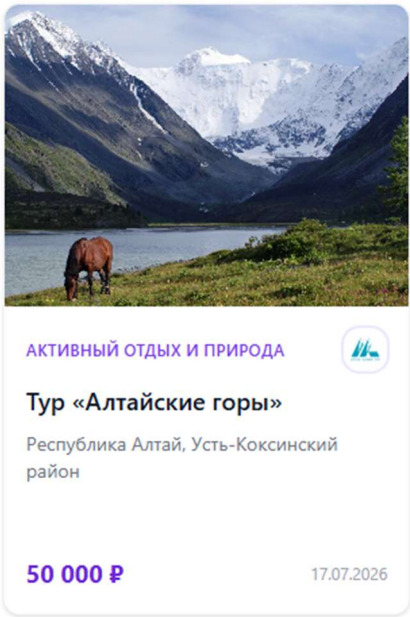


Рисунок Д.8 – Измененная карточка тура

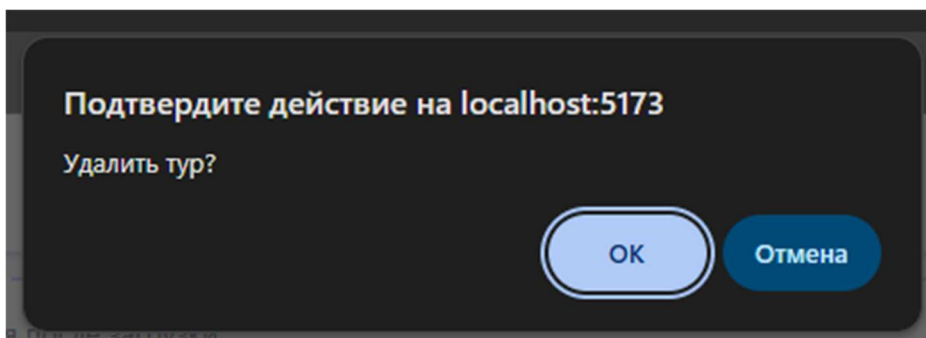


Рисунок Д.9 – Предупреждение перед удалением карточки тура

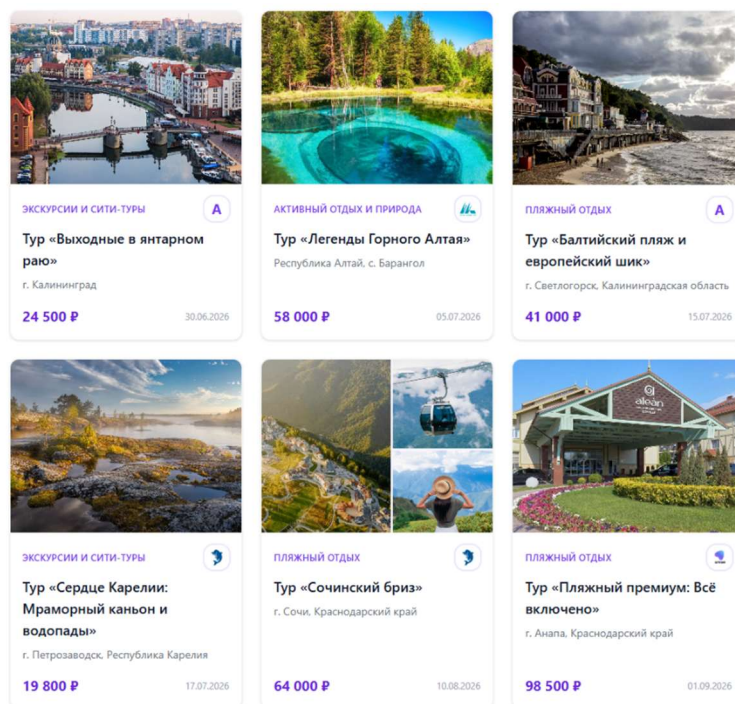


Рисунок Д.10 – Каталог после удаления карточки

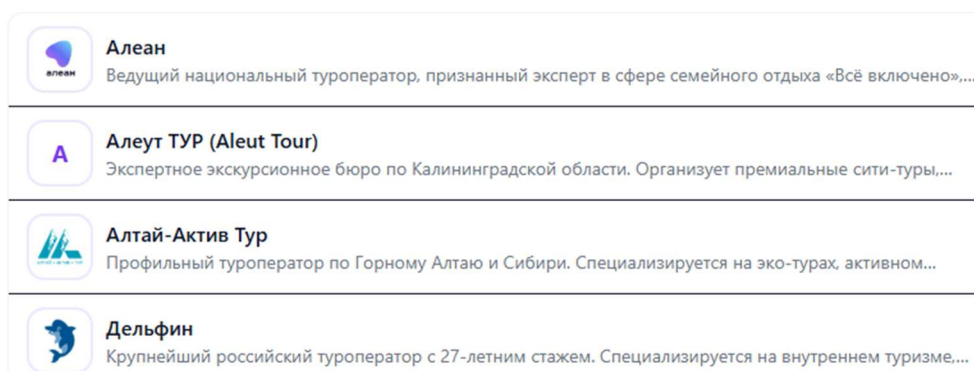


Рисунок Д.11 – Список провайдеров

Туроператоры

В

Логотип
Выберите файл Файл не выбран

Название
Вершины Мира

Описание
Туроператор, специализирующийся на организации экологического туризма, восхождений и экспедиций в труднодоступные регионы. Команда профессиональных гидов с 10-летним стажем работы в горах.

Добавить

Рисунок Д.12 – Изменение туроператора

	Алеан Ведущий национальный туроператор, признанный эксперт в сфере семейного отдыха «Всё включено»,...
	Алеут ТУР (Aleut Tour) Экспертное экскурсионное бюро по Калининградской области. Организует премиальные сити-туры,...
	Алтай-Актив Тур Профильный туроператор по Горному Алтаю и Сибири. Специализируется на эко-турах, активном...
	Вершины Мира Туроператор, специализирующийся на организации экологического туризма, восхождений и экспедиц...
	Дельфин Крупнейший российский туроператор с 27-летним стажем. Специализируется на внутреннем туризме,...

Рисунок Д.13 – Список провайдеров с новым туроператором

	Горные Вершины Туроператор, специализирующийся на организации экологического туризма, восхождений и экспедиц...
---	---

Рисунок Д.14 – Измененный туроператор

	Алеан Ведущий национальный туроператор, признанный эксперт в сфере семейного отдыха «Всё включено»,...
	Алеут ТУР (Aleut Tour) Экспертное экскурсионное бюро по Калининградской области. Организует премиальные сити-туры,...
	Алтай-Актив Тур Профильный туроператор по Горному Алтаю и Сибири. Специализируется на эко-турах, активном...
	Дельфин Крупнейший российский туроператор с 27-летним стажем. Специализируется на внутреннем туризме,...

Рисунок Д.15 – Список провайдеров после удаления

Приложение Е

(рекомендуемое)

Результаты кросс-браузерного тестирования

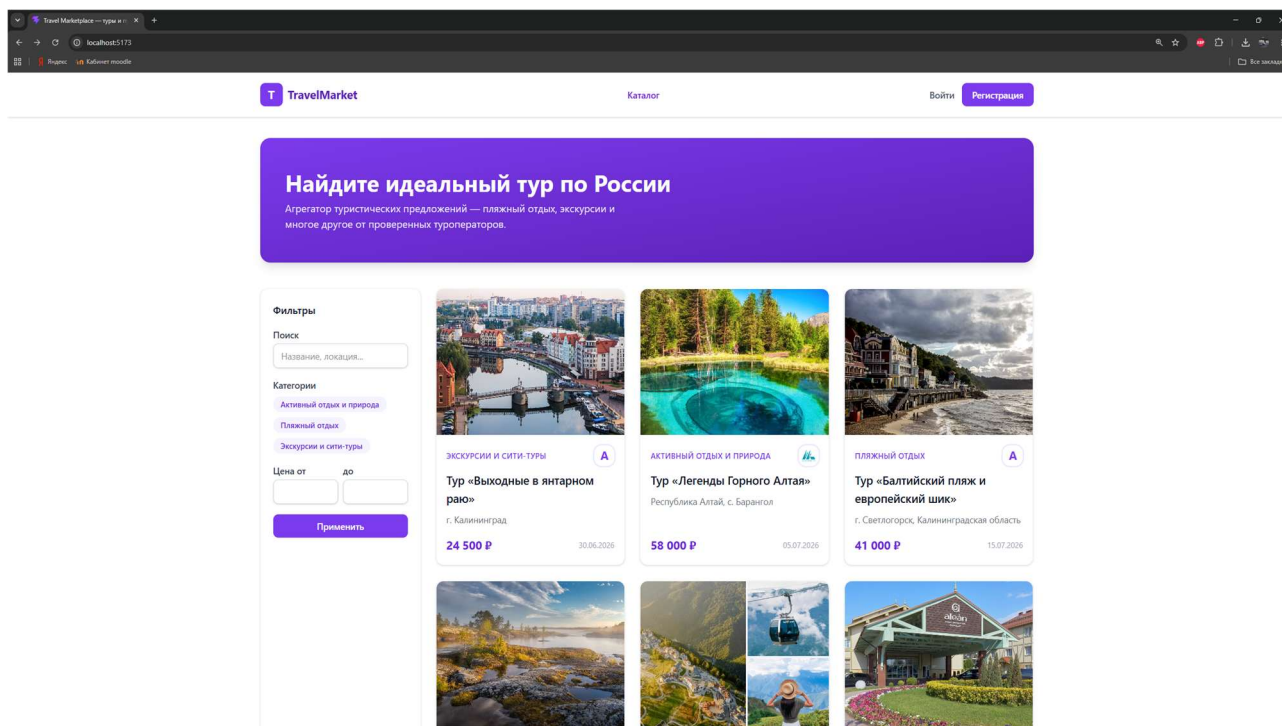


Рисунок Е.1 – Главная страница (каталог) в Google Chrome

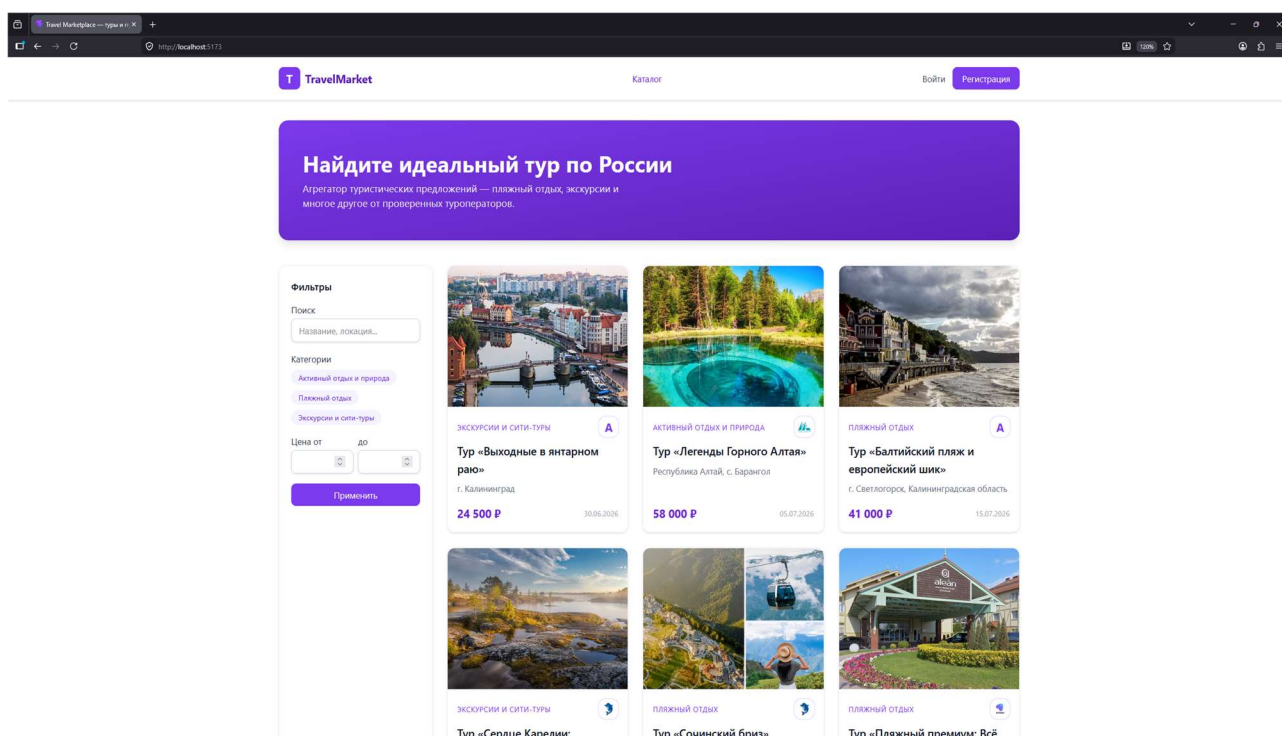


Рисунок Е.2 – Главная страница (каталог) в Mozilla Firefox

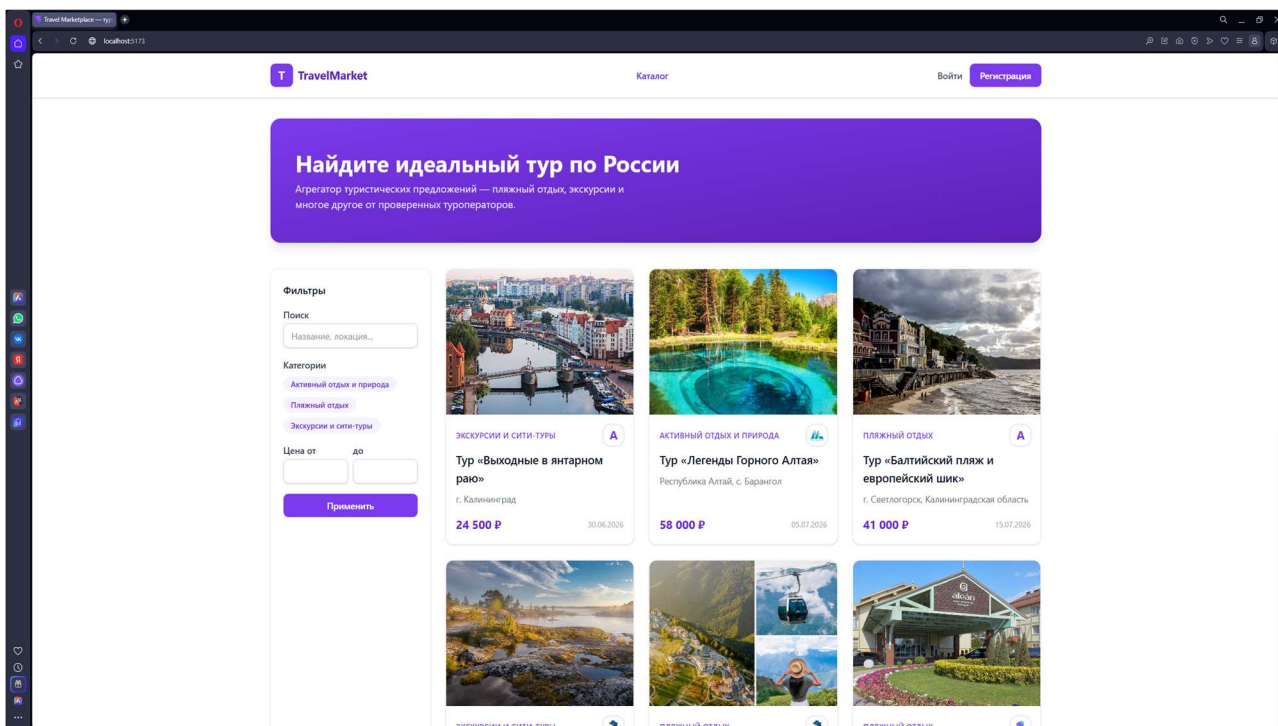


Рисунок Е.3 – Главная страница (каталог) в Opera

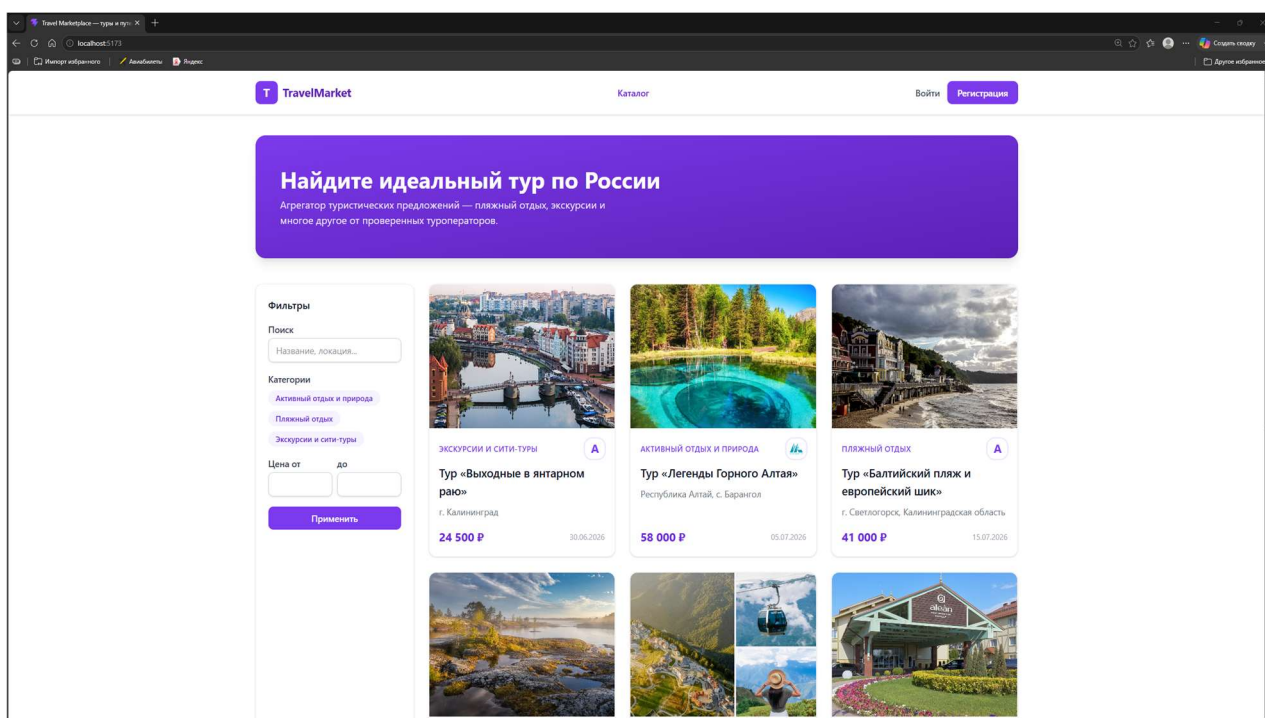


Рисунок Е.4 – Главная страница (каталог) в Microsoft Edge