

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

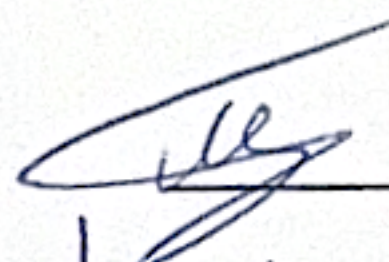
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

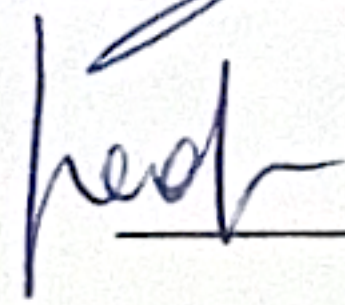
на тему

**СОЗДАНИЕ ВЕБ-ПЛАТФОРМЫ ПРИЕМА И
ОБРАБОТКИ ЗАКАЗОВ КОНТРАГЕНТОВ**

Студент

 25.05.26 Перкин Павел Олегович
(подпись, дата) (ФИО полностью)

Руководитель

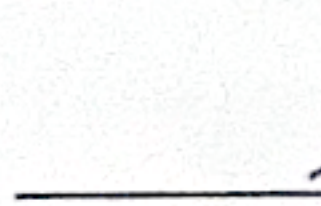
 25.05.26 Леонова Т. Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер

 25.05.26 Бычков А. С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

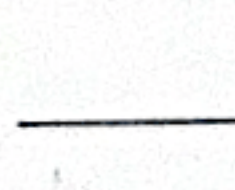
Заведующий кафедрой

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

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

08 июня 2026 г. № 19)

Секретарь ГЭК

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

Пенза 2026

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Перкин Павел Олегович гр. 22ВВП2 факультета вычислительной техники
2. Тема работы: «Создание веб-платформы приема и обработки заказов контрагентов»

Тема утверждена приказом ПГУ № 347/о от "07" апреля 2026 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Леонова Т. Ю.

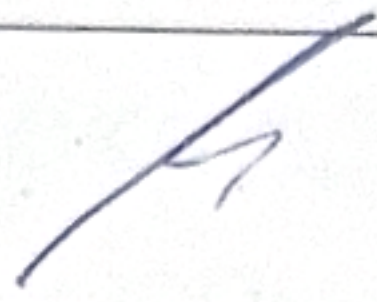
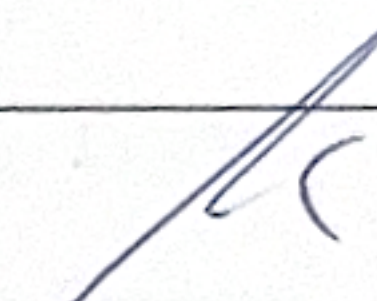
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать веб-платформу для автоматизации процессов приема и обработки заказов контрагентов службы доставки «FreeRoll». Разработку клиентской части выполнить на языке JavaScript с использованием библиотеки React, серверную часть на языке C# с использованием фреймворка ASP.NET Core Web API. Использовать встраиваемую СУБД SQLite для хранения данных. Система должна обеспечивать корректное оформление заказов, управление товарным каталогом, учет промо-акций, а также формирование аналитической отчетности о деятельности службы доставки в структурированном виде.

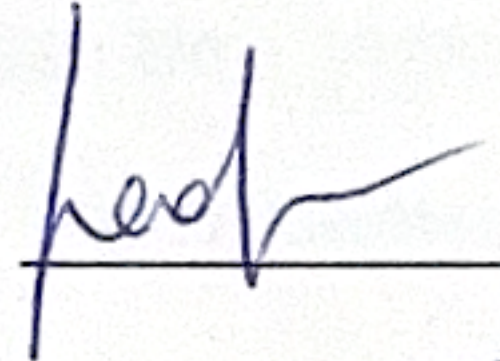
5. Перечень подлежащих разработке вопросов

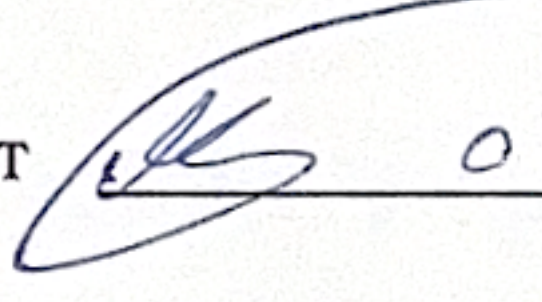
Требуется провести анализ предметной области и изучить существующие программные решения в сфере автоматизации систем доставки продукции. Обосновать выбор технологического стека и инструментов разработки. Спроектировать общую архитектуру системы и определить логику взаимодействия клиентского и серверного приложений. Спроектировать структуру реляционной базы данных. Разработать программную реализацию платформы, включающую пользовательский интерфейс, логику обработки данных и систему управления заказами. Выполнить тестирование и отладку разработанного программного обеспечения.

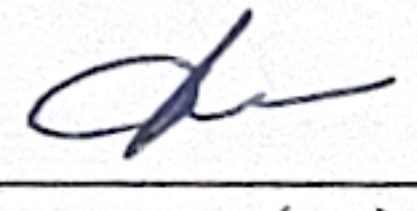
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	45	07.04.26 – 13.04.26	
2	Анализ темы, аналогов, постановка задачи	24	13.04.26 – 17.04.26	
3	Выбор и настройка инструментов разработки	50	17.04.26 – 24.04.26	
4	Проектирование архитектуры	30	24.04.26 – 28.04.26	
5	Разработка серверной и клиентской частей веб-платформы, тестирование системы	151	28.04.26 – 17.05.26	
6	Оформление пояснительной записки	24	17.05.26 – 21.05.26	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  07.04.26 Леонова Т. Ю.
(подпись, дата)

Задание к исполнению принял студент  07.04.26 Перкин П. О.
(подпись, дата)

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

02.06.2020

Реферат

Бакалаврская работа содержит 151 лист, 47 рисунков, 2 таблицы, 10 источников, 5 приложений.

ВЕБ-ПЛАТФОРМА, REACT, REDUX, C#, ASP.NET CORE, ДОСТАВКА ЕДЫ, REST API, БАЗА ДАННЫХ, SQLITE, АВТОМАТИЗАЦИЯ БИЗНЕС-ПРОЦЕССОВ, АДМИНИСТРАТИВНАЯ ПАНЕЛЬ

Цель работы – разработка и внедрение масштабируемой веб-платформы для автоматизации процессов приема, обработки заказов и управления товарным каталогом в службе доставки «FreeRoll».

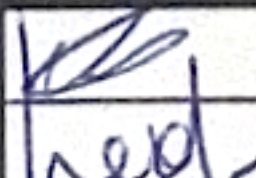
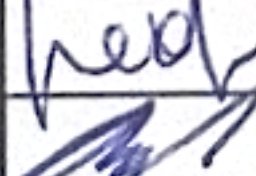
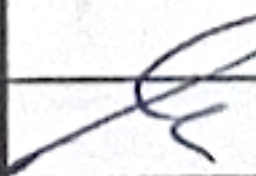
· Объектами разработки являются:

– клиентское веб-приложение, реализованное с использованием библиотеки React;

– серверная часть системы на языке C# (фреймворк ASP.NET Core Web API), обеспечивающая обработку бизнес-логики и взаимодействие с пользователями;

– реляционная база данных под управлением СУБД SQLite для хранения информации о пользователях, ассортименте продукции, транзакциях и истории заказов.

· В результате проделанной работы разработана программная платформа, позволяющая автоматизировать полный цикл жизненного цикла заказа: от выбора позиции в каталоге до доставки. Система включает личный кабинет пользователя, административную панель управления с воронкой статусов заказов, функционал работы с акциями и аналитический модуль для мониторинга эффективности бизнеса.

					ПГУ 1.090301.44.001 ПЗ			
Изм.	Лист	№ докум.	Подпис	Дата	Создание веб-платформы приема и обработки заказов контрагентов Пояснительная записка	Лит.	Лист	Листов
Разраб.		Перкин		25.09.26				
Провер.		Леонова		26.09.26			2	83
Н. Контр.		Бычков А.С.		26.09.26		ФВТ, каф. ВТ гр. 22ВВП2		
Утверд.		Митрохин		25.09.26				

Содержание

Перечень сокращений и обозначений	6
Введение.....	8
1 Анализ предметной области	9
1.1 Назначение и постановка задачи	9
1.2 Обзор аналогов	9
1.2.1 Прямые аналоги	9
1.2.2 Косвенные аналоги.....	10
1.2.3 Альтернативные подходы на базе конструкторов и CMS	12
1.3 Технические требования к системе.....	13
2 Архитектура системы и выбор инструментов разработки	15
2.1 Архитектура системы	15
2.1.1 Принципы декомпозиции и организации архитектуры клиентского уровня	15
2.1.2 Организация межуровневого взаимодействия.....	17
2.1.3 Серверный уровень и подсистема управления данными	18
2.2 Выбор инструментов разработки	19
2.2.1 Серверная часть: ASP.NET Core и язык C#	19
2.2.2 Клиентская часть: React и экосистема фронтенда.....	20
2.2.3 Система управления базами данных: SQLite	21
2.2.4 Инструменты контроля версий и организации процесса	22
2.3 Установка и настройка рабочих сред	22
2.3.2 Установка и настройка WebStorm и клиентского окружения	28
2.3.3 Установка утилиты для администрирования базы данных.....	34
3 Реализация и тестирование системы.....	36
3.1 Структура БД.....	36
3.2 Разработка серверной части.....	40
3.2.1 Подсистема аутентификации и управления профилями	41

3.2.2	Модуль маршрутизации и обработки заказов	43
3.2.3	Модуль аналитики и формирования статистики	45
3.2.4	Подсистема работы с медиафайлами.....	47
3.2.5	Управление маркетингом и обратной связью	49
3.3	Разработка клиентского веб-приложения	49
3.3.1	Точка входа и глобальная инициализация приложения	50
3.3.2	Декларативный роутинг и древовидная структура маршрутов	51
3.3.3	Глобальная компоновка и сквозные элементы раскладки	52
3.3.4	Сетевой уровень и клиент взаимодействия с API.....	54
3.3.5	Централизованное управление состоянием	56
3.3.6	Механизм асинхронных действий и побочных эффектов	58
3.3.7	Управление состоянием корзины и персистентность данных	58
3.3.8	Конфигурация жизненного цикла заказа	59
3.3.9	Базовые визуальные компоненты предметной области	61
3.3.10	Интерактивные инструменты обратной связи и модерации контента .	62
3.3.11	Бизнес-логика корзины: применение промо-акций и управление доставкой.....	63
3.3.12	Событийно-ориентированная архитектура административных модальных окон	64
3.3.13	Композиция клиентской витрины и жестовая навигация	67
3.3.14	Рабочее место оператора: маршрутизация и приоритизация заказов ...	68
3.3.15	Аналитический дашборд и визуализация бизнес-метрик.....	70
3.4	Тестирование системы.....	72
4	Оценка экономической эффективности проекта	77
4.1	Методика расчета затрат на разработку.....	77
4.2	Анализ операционных выгод	77
4.3	Расчет окупаемости	78
5	Безопасность жизнедеятельности	79
5.1	Правила безопасности при использовании ПК.....	79

5.2 Требования производственной санитарии и пожарной безопасности	80
Заключение	82
Список использованных источников	83
Приложение А – UML-диаграммы	84
Приложение Б – Схема базы данных	90
Приложение В – Листинги программы.....	92
Приложение Г – Презентация	143
Приложение Д – Справка о внедрении результатов ВКР.....	150

Перечень сокращений и обозначений

В работе используются следующие обозначения и сокращения

БД – база данных

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

API – (англ. Application Programming Interface) – программный интерфейс приложения

ASP.NET – (англ. Active Server Pages .NET) – фреймворк для разработки веб-приложений

CMS – (англ. Content Management System) – система управления содержимым (контентом) сайта

CORS – (англ. Cross-Origin Resource Sharing) – механизм совместного использования ресурсов разных источников

CRUD – (англ. Create, Read, Update, Delete) – операции создания, чтения, обновления и удаления данных

CSS – (англ. Cascading Style Sheets) – каскадные таблицы стилей

DOM – (англ. Document Object Model) – объектная модель документа

DRY – (англ. Don't Repeat Yourself) – принцип проектирования, направленный на избежание дублирования информации

DTO – (англ. Data Transfer Object) – объект передачи данных

EF Core – (англ. Entity Framework Core) – объектно-реляционная система сопоставления (ORM)

FSD – (англ. Feature-Sliced Design) – методология архитектурной организации проекта

GUID – (англ. Globally Unique Identifier) – статистически уникальный 128-битный идентификатор

HTTP – (англ. HyperText Transfer Protocol) – протокол передачи гипертекста

					ПГУ 1.090301.44.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

IDE – (англ. Integrated Development Environment) – интегрированная среда разработки

JSON – (англ. JavaScript Object Notation) – текстовый формат обмена данными

KPI – (англ. Key Performance Indicators) – ключевые показатели эффективности

MVC – (англ. Model-View-Controller) – архитектурный шаблон разделения данных, логики и представления

ORM – (англ. Object-Relational Mapping) – технология объектно-реляционного отображения

REST – (англ. Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения

SaaS – (англ. Software as a Service) – бизнес-модель продажи и использования программного обеспечения по подписке

SOLID – (англ. SOLID) – акроним пяти основных принципов объектно-ориентированного программирования и проектирования

SPA – (англ. Single Page Application) – одностраничное веб-приложение

SQL – (англ. Structured Query Language) – язык структурированных запросов

SVG – (англ. Scalable Vector Graphics) – язык масштабируемой векторной графики

UI – (англ. User Interface) – пользовательский интерфейс

UX – (англ. User Experience) – пользовательский опыт

Введение

В условиях цифровой экономики бизнесу требуются гибкие системы автоматизации заказов для прозрачного и оперативного обмена данными. Существующие готовые решения часто отличаются высокой стоимостью и избыточностью функционала. Разработка индивидуальной платформы на базе современных веб-технологий позволяет создать масштабируемое решение, точно адаптированное под специфические задачи предприятия, что и определяет актуальность данной работы.

Целью выпускной работы является проектирование и разработка веб-платформы для автоматизации процессов приема, учета и обработки заказов контрагентов службы доставки «FreeRoll».

Объектом исследования выступают процессы автоматизации приема, обработки и управления заказами в сфере доставки готовой продукции.

Предметом исследования являются программные средства, архитектурные шаблоны и методы разработки масштабируемой веб-платформы с использованием современных веб-технологий.

Для достижения поставленной цели необходимо решить следующие задачи:

- Проанализировать предметную область и существующие аналоги систем доставки продукции.
- Обосновать выбор стека технологий и настроить среды разработки.
- Спроектировать структуру реляционной базы данных платформы.
- Разработать серверную часть для обработки бизнес-логики и управления данными.
- Разработать клиентское веб-приложение и интерфейс администратора.
- Провести комплексное тестирование разработанного программного обеспечения.

1 Анализ предметной области

1.1 Назначение и постановка задачи

Основным назначением данной работы является разработка масштабируемой веб-платформы для автоматизации процессов приема, обработки заказов и управления товарным каталогом службы доставки «FreeRoll». Комплекс предназначен для обеспечения взаимодействия между пользователями и администрацией, автоматизации формирования заказов, управления статусами доставки и ведения структурированной истории операций.

В рамках данной работы ставится задача спроектировать, реализовать и протестировать систему, состоящую из клиентского веб-приложения, серверной части API и базы данных для хранения информации. Функциональные возможности системы для различных ролей пользователей представлены на диаграмме вариантов использования в Приложении А.1.

1.2 Обзор аналогов

Для определения места разрабатываемой веб-платформы в ряду существующих решений и для формирования требований к ней был проведен анализ актуальных аналогов на рынке автоматизации предприятий общественного питания и доставки.

1.2.1 Прямые аналоги

В качестве прямых аналогов разрабатываемой платформы выступают профессиональные облачные SaaS-системы автоматизации ресторанов и служб доставки. Основными игроками на этом рынке являются такие программные комплексы, как iiko, r_keeper и Saby (СБИС Presto). Эти платформы решают ту же фундаментальную задачу – прием, обработка и передача заказов, ведение каталога и аналитика продаж, – но ориентированы на крупный сетевой бизнес.

Данные системы представляют собой мощные многомодульные платформы, способные интегрироваться с бухгалтерским учетом, складскими программами и

системами налоговой отчетности. Они широко применяются в индустрии HoReCa для сквозного контроля всех этапов работы заведения.

Пример такой системы представлен на рисунке 1.

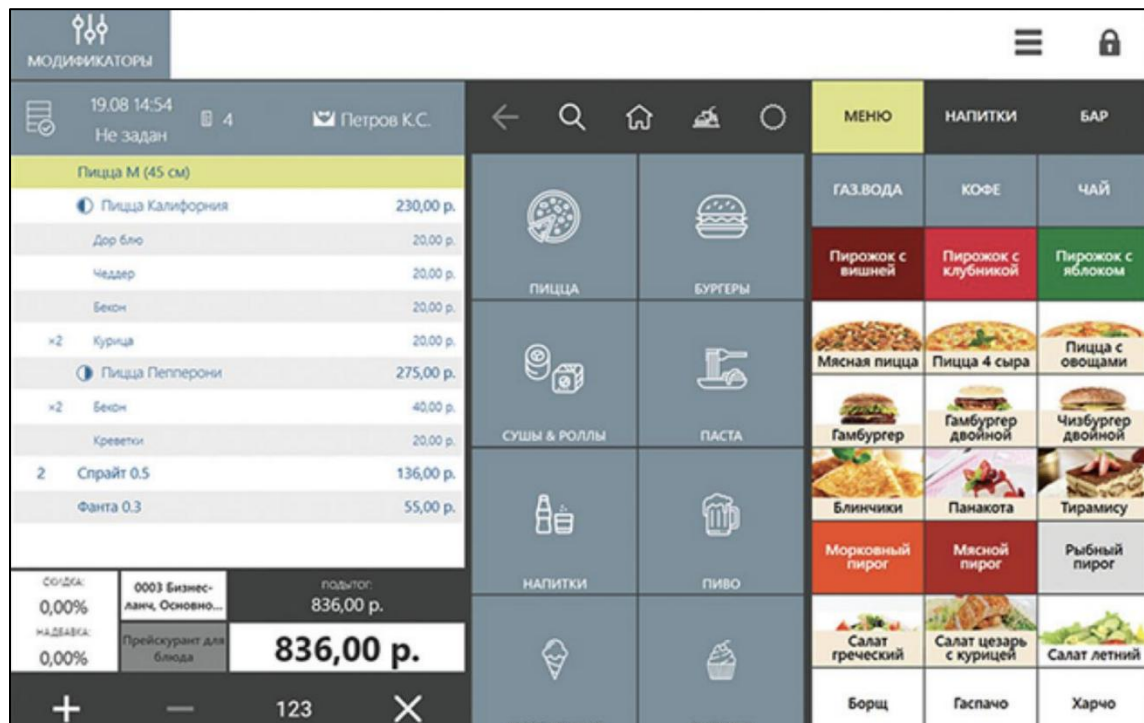


Рисунок 1 – Интерфейс промышленной системы автоматизации ресторана

Ключевым отличием данных систем от разрабатываемого решения является их высокая стоимость владения (регулярная абонентская плата за каждый модуль), избыточность функционала для локальных служб доставки, а также сложность внедрения. Настройка таких комплексов требует привлечения сертифицированных специалистов, а закрытая архитектура не позволяет владельцам бизнеса оперативно и дешево внедрять уникальные функции (например, специфическую логику начисления подарков или кастомизированную воронку статусов заказов).

1.2.2 Косвенные аналоги

К косвенным аналогам можно отнести агрегаторы доставки готовой еды (например, «Яндекс.Еда», «Маркет Деливери», «Купер»). Эти платформы предоставляют заведениям готовый интерфейс для размещения меню и приема

заказов от огромной базы пользователей.

Главным преимуществом агрегаторов является отсутствие необходимости разрабатывать собственную IT-инфраструктуру: ресторан получает готовое мобильное и веб-приложение. Однако этот подход имеет ряд критических недостатков для бизнеса. Пример такого агрегатора доставки представлен на рисунке 2.

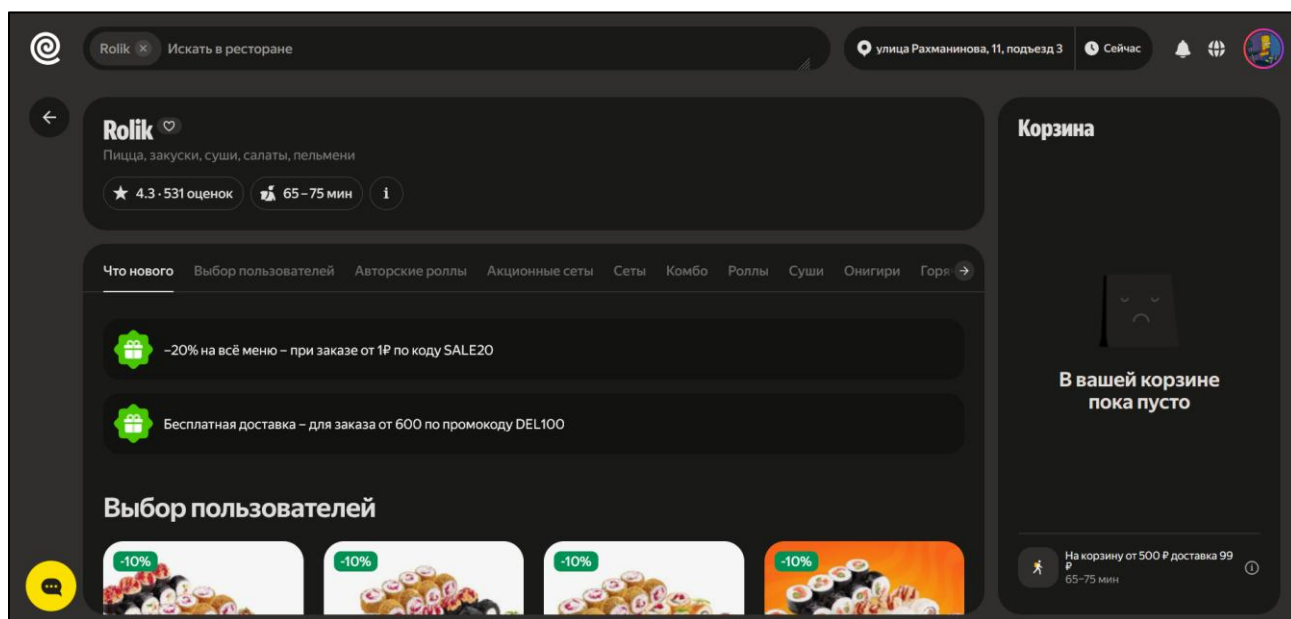


Рисунок 2 – Интерфейс приложения агрегатора доставки «Яндекс.Еда»

Во-первых, агрегаторы взимают высокую комиссионную плату (вплоть до 30-35% с каждого заказа), что делает их использование экономически невыгодным при формировании собственной базы лояльных клиентов. Во-вторых, заведение не имеет полного доступа к данным своих покупателей, что исключает возможность ведения независимой маркетинговой аналитики и рассылок. В-третьих, интерфейс жестко стандартизирован: служба доставки не может выделиться визуально или внедрить собственную сложную систему промокодов.

Таким образом, агрегаторы, решая задачу генерации потока заказов, не могут рассматриваться как полноценная замена собственной платформе управления, так как лишают бизнес независимости и контроля над данными.

1.2.3 Альтернативные подходы на базе конструкторов и CMS

Наиболее близкой по подходу к быстрой разработке является категория проектов, реализуемых на базе конструкторов сайтов (Tilda, Wix) или универсальных систем управления контентом (CMS) с плагинами для электронной коммерции (например, WordPress + WooCommerce).

Существует большое количество готовых шаблонов, позволяющих в короткие сроки запустить сайт-витрину с функцией корзины. В типичной архитектуре такого проекта заказы падают на электронную почту администратора или в подключенную Telegram-группу, либо отображаются во внутренней панели CMS. Пример конструктора сайта представлен на рисунке 3.

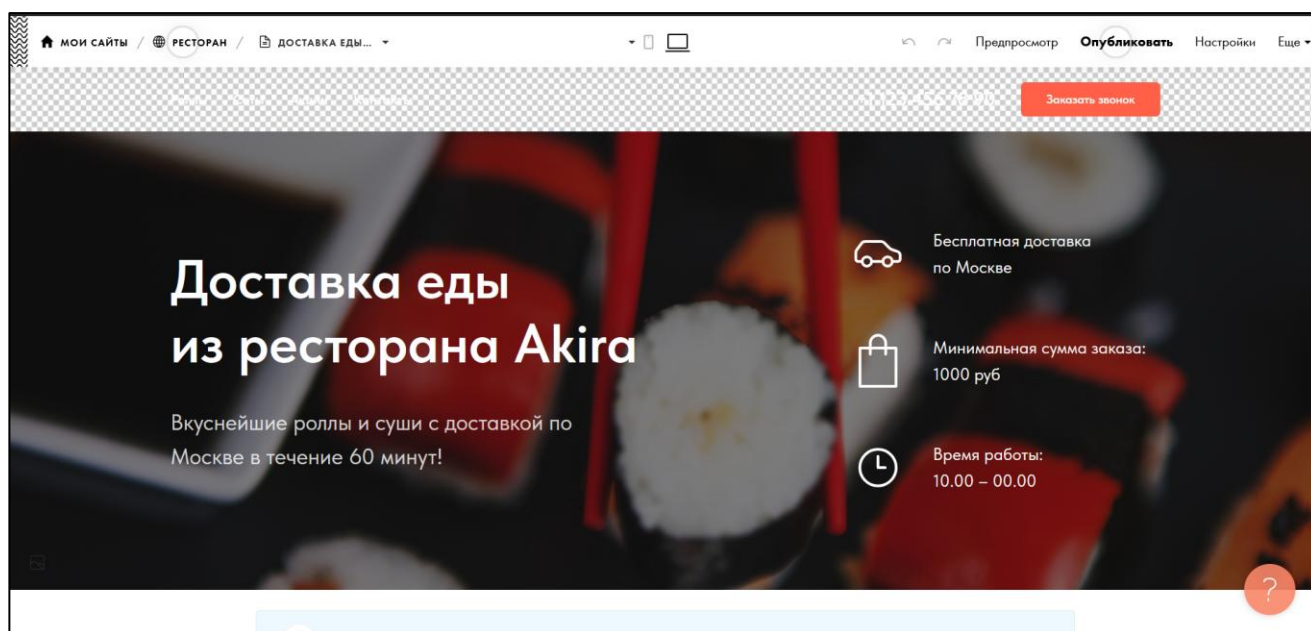


Рисунок 3 – Интерфейс настройки магазина в конструкторе сайтов

Основным преимуществом данного подхода является его доступность и низкий порог входа. Наличие готовых блоков позволяет быстро создать рабочий прототип сайта.

Однако, несмотря на популярность, подход на базе CMS и конструкторов имеет ряд принципиальных ограничений. Подобные системы не спроектированы специально для высоконагруженных служб доставки. Отсутствует возможность

тонкой настройки жизненного цикла заказа (динамическое изменение статусов «на кухне», «в пути» в реальном времени), а производительность монолитных CMS быстро падает при увеличении базы товаров и пользователей. Интеграция сложной бизнес-логики (например, автоматическое применение скидок в зависимости от дня недели, времени суток и состава корзины без учета дополнений) требует написания множества нестандартных скриптов, что сводит на нет преимущество использования готового конструктора.

В отличие от этого, разработка собственной платформы с использованием современного стека веб-технологий (React для интерактивного интерфейса и ASP.NET Core для серверной логики) позволяет создать масштабируемую архитектуру. Модульное разделение клиентской и серверной частей обеспечивает высокую скорость отклика, предсказуемость поведения системы и неограниченные возможности для дальнейшего расширения функционала под конкретные бизнес-процессы службы доставки. Это является ключевым обоснованием выбора пути индивидуальной разработки программного обеспечения.

1.3 Технические требования к системе

Серверная часть платформы должна функционировать под управлением операционных систем семейства Linux или Windows с установленной средой выполнения .NET. Для обеспечения стабильной работы серверного приложения и обработки множественных запросов требуются минимальные аппаратные ресурсы: процессор с тактовой частотой не ниже 2 ГГц и не менее 2 ГБ оперативной памяти. Также необходимо наличие не менее 1 ГБ свободного дискового пространства для размещения исполняемых файлов платформы, хранения файлов базы данных SQLite и пользовательских медиафайлов.

Клиентская часть представляет собой кроссплатформенное веб-приложение и не требует установки специализированного программного обеспечения на устройство пользователя. Для корректного отображения и работы интерфейса

необходимо любое пользовательское устройство (персональный компьютер, планшет или мобильный телефон) с установленным современным веб-браузером (Google Chrome, Safari, Mozilla Firefox или Microsoft Edge), поддержкой исполнения сценариев JavaScript и стабильным подключением к сети Интернет.

					ПГУ 1.090301.44.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

2 Архитектура системы и выбор инструментов разработки

2.1 Архитектура системы

Проектируемая веб-платформа базируется на классической трехзвенной клиент-серверной архитектуре, разделяющей программный комплекс на три изолированных вычислительных уровня: уровень представления (клиентская часть), уровень бизнес-логики (серверная часть) и уровень хранения данных (реляционная база данных). Подобное системное разделение ответственности позволяет минимизировать взаимную связность компонентов, обеспечивает независимость процессов разработки и развертывания каждого элемента, а также закладывает фундамент для горизонтального масштабирования платформы при возрастании интенсивности пользовательских запросов. Взаимодействие между уровнями строго регламентировано программными протоколами и интерфейсами, что исключает прямое несанкционированное обращение клиентского уровня к хранилищу данных, повышая общую безопасность и отказоустойчивость системы. Общая схема физического распределения и топология взаимодействия узлов комплекса приведена на диаграмме развертывания в Приложении А.5.

2.1.1 Принципы декомпозиции и организации архитектуры клиентского уровня

Клиентский уровень программного комплекса спроектирован как одностраничное веб-приложение (Single Page Application). Для преодоления проблемы неконтролируемого роста связности кодовой базы и обеспечения высокой степени модульности в качестве структурного стандарта была выбрана и внедрена архитектурная методология Feature-Sliced Design (FSD) [1]. В основе данного подхода лежит концепция послойного распределения модулей в соответствии с их степенью абстракции и функциональным назначением в предметной области.

Декомпозиция клиентской части по методологии FSD осуществляется по вертикали и горизонтали, образуя фиксированную структуру, состоящую из слоев

(Layers), слайсов (Slices) и сегментов (Segments). Иерархия слоев строго регламентирует правила импорта: программные модули, расположенные на вышележащих уровнях, имеют право импортировать код только из нижележащих слоев. Импорт в обратном направлении строго запрещен. Это позволяет полностью исключить возникновение циклических зависимостей и гарантирует, что изменения в коде высокоуровневых интерфейсов не нарушат стабильность базовой логики приложения.

В рамках разработанной системы выделяются следующие шесть архитектурных слоев представления данных:

1) слой приложения (App): выполняет функцию инициализирующего ядра системы. Здесь сосредоточены глобальные настройки платформы, конфигурация централизованного хранилища состояний, декларативный роутинг страниц и подключение глобальных стилей оформления интерфейса;

2) слой страниц (Pages): содержит композиционные модули, представляющие собой полноценные экраны веб-платформы. Модули данного слоя лишены собственной изолированной логики и собираются из нижележащих интерфейсных блоков;

3) слой виджетов (Widgets): объединяет компоненты в крупные, самостоятельные блоки интерфейса, несущие законченный визуальный и функциональный смысл на страницах приложения;

4) слой функций (Features): реализует конкретные действия пользователей или интерактивные сценарии, приносящие непосредственную бизнес-ценность. Модули функций инкапсулируют в себе обработку пользовательских событий, формы ввода и отправку асинхронных сетевых запросов к серверу;

5) слой сущностей (Entities): описывает бизнес-сущности предметной области, их внутреннюю структуру данных, базовые визуальные карточки и логику синхронизации их состояния в глобальном хранилище приложения;

6) слой общих ресурсов (Shared): нижний уровень абстракции, содержащий универсальные переиспользуемые модули, полностью лишенные привязки к

специфике предметной области. Сюда вынесены базовые UI-компоненты, низкоуровневый API-клиент и системные утилиты.

Для обеспечения инкапсуляции каждый слайс внутри слоев предоставляет строго ограниченный внешний интерфейс (Public API) через единую точку экспорта. Взаимодействие между компонентами клиентской части строится на принципах реактивного управления состоянием, при котором изменения в хранилище данных автоматически вызывают направленное обновление соответствующих элементов пользовательского интерфейса.

2.1.2 Организация межуровневого взаимодействия

Связующим звеном между независимыми уровнями представления и бизнес-логики выступает программный интерфейс приложения (API), спроектированный в соответствии с архитектурными принципами REST (Representational State Transfer). Межуровневое взаимодействие организуется поверх прикладного протокола HTTP/HTTPS с использованием асинхронной модели обмена данными.

Клиентская часть не хранит состояние сессии на сервере (Stateless-подход), что позволяет изолировать серверный уровень от контекста конкретного пользователя и существенно повышает общую производительность системы. Передача информации осуществляется в стандартизированном текстовом формате JSON (JavaScript Object Notation).

Архитектура взаимодействия строится на отправке типизированных запросов со стороны клиента:

- запросы на получение информации транслируются через методы GET и обрабатываются сервером без изменения внутреннего состояния системы;
- модификация данных, создание новых записей или удаление объектов реализуются посредством направленной отправки HTTP-методов POST, PUT, PATCH и DELETE.

Для оптимизации трафика и разграничения ответственности на стыке систем используется паттерн проектирования DTO (Data Transfer Object). Это позволяет

передавать между клиентом и сервером только те подмножества данных, которые необходимы для выполнения конкретной операции, скрывая избыточные системные поля базы данных и защищая внутреннюю структуру таблиц от внешнего анализа.

2.1.3 Серверный уровень и подсистема управления данными

Серверная часть платформы реализует функции приема запросов, авторизации пользователей, выполнения вычислительных алгоритмов и координации доступа к хранилищу данных. Логика серверного уровня построена на базе фреймворка ASP.NET Core Web API [2], реализующего паттерн проектирования MVC (Model-View-Controller) в части разделения контроллеров обработки API.

При поступлении сетевого пакета серверный уровень выполняет маршрутизацию и направляет данные на соответствующий программный контроллер. Внутренняя архитектура серверной части разделена на логические слои:

- слой контроллеров (API Layer): отвечает за прием HTTP-запросов, десериализацию входных JSON-пакетов, первичную валидацию параметров и формирование HTTP-ответов;

- слой бизнес-логики (Service/Business Layer): инкапсулирует в себе алгоритмы обработки заказов, вычисление стоимостных показателей с учетом динамических ограничений промо-акций и проведение криптографических процедур;

- слой доступа к данным (Data Access Layer): обеспечивает безопасное взаимодействие с нижележащей СУБД.

Безопасность учетных записей пользователей обеспечивается непосредственно на серверном уровне. При регистрации и аутентификации применяется алгоритм одностороннего криптографического хэширования BCrypt, что исключает хранение паролей в открытом виде и защищает систему от компрометации данных в случае несанкционированного доступа к файлам

системы. Предполагаемый алгоритм работы модулей обработки данных подробно описывается UML-диаграммой деятельности, которая представлена в Приложении А.4.

Нижним звеном архитектурного комплекса является подсистема долговременного хранения информации под управлением реляционной СУБД SQLite. Архитектурное сопряжение серверного кода с базой данных реализовано на основе ORM-системы Entity Framework Core. Использование ORM позволяет абстрагировать логику приложения от прямых SQL-инструкций, транслируя операции над объектными моделями в оптимизированные запросы к базе данных.

Для защиты от уязвимостей класса SQL-инъекций EF Core автоматически применяет механизм параметризации всех входящих аргументов в генерируемых SQL-запросах. Физический файл базы данных располагается в изолированной директории на стороне сервера, что минимизирует время отклика системы при выполнении транзакций ввода-вывода и обеспечивает целостность хранимой структуры данных.

2.2 Выбор инструментов разработки

Для реализации программного комплекса был выбран стек технологий, обеспечивающий высокую производительность, масштабируемость и простоту сопровождения. Выбор основывался на принципах кроссплатформенности, зрелости экосистемы и эффективности инструментов.

2.2.1 Серверная часть: ASP.NET Core и язык C#

В качестве платформы для разработки серверного API выбран фреймворк ASP.NET Core. Это высокопроизводительная, модульная платформа с открытым исходным кодом, позволяющая создавать масштабируемые веб-службы.

Язык C# выбран за строго типизированный синтаксис [3], развитые механизмы асинхронного программирования и глубокую интеграцию с библиотеками .NET. Это позволяет реализовывать сложную бизнес-логику с высокой степенью надежности.

Производительность ASP.NET Core стабильно занимает лидирующие позиции в тестах производительности [4], что критично для обработки потока заказов в реальном времени.

Среда разработки Visual Studio 2022 представлена на рисунке 4.

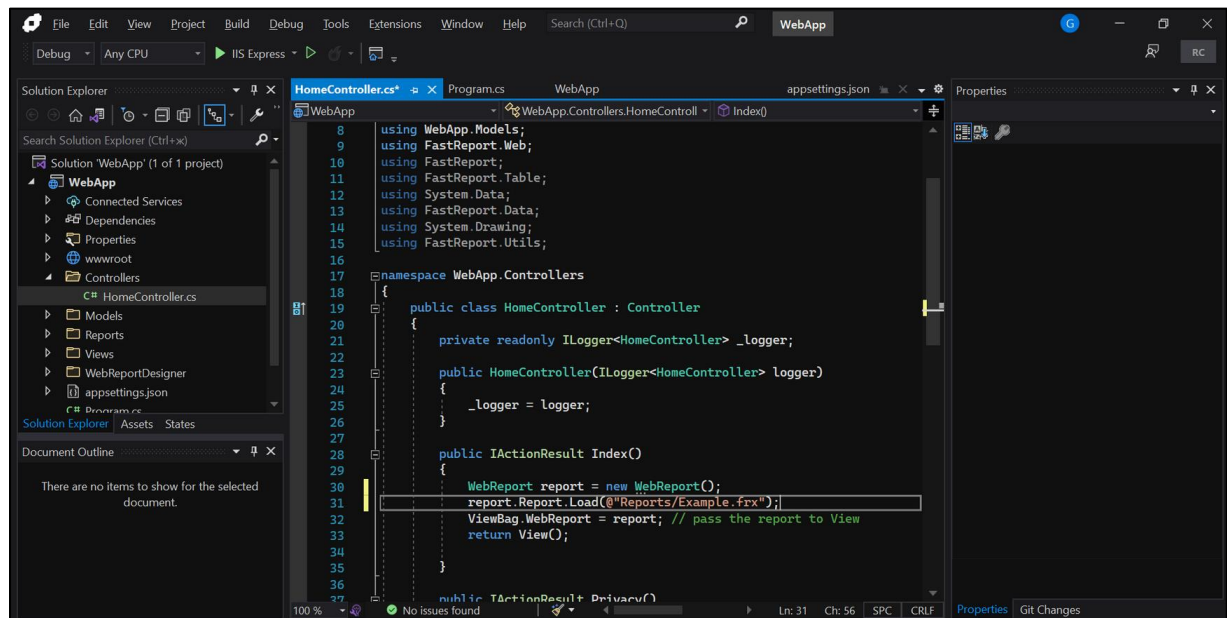


Рисунок 4 – Среда разработки Visual Studio 2022

2.2.2 Клиентская часть: React и экосистема фронтенда

Интерфейс пользователя реализован как одностраничное приложение (SPA) с использованием библиотеки React [5]. Выбор React обусловлен его компонентной моделью [6], которая идеально сочетается с архитектурой Feature-Sliced Design (FSD).

Для фронтенд-части выбрана IDE WebStorm от JetBrains. Ее интеллектуальные инструменты анализа кода, встроенный терминал и мощный отладчик для JavaScript [7] значительно ускоряют написание сложных интерфейсов и поддержку чистоты кода.

Использование Vite в качестве сборщика обеспечивает мгновенный "горячий" перезапуск модулей (HMR), что критически важно при разработке интерактивных элементов интерфейса.

Среда разработки WebStorm показана на рисунке 5.

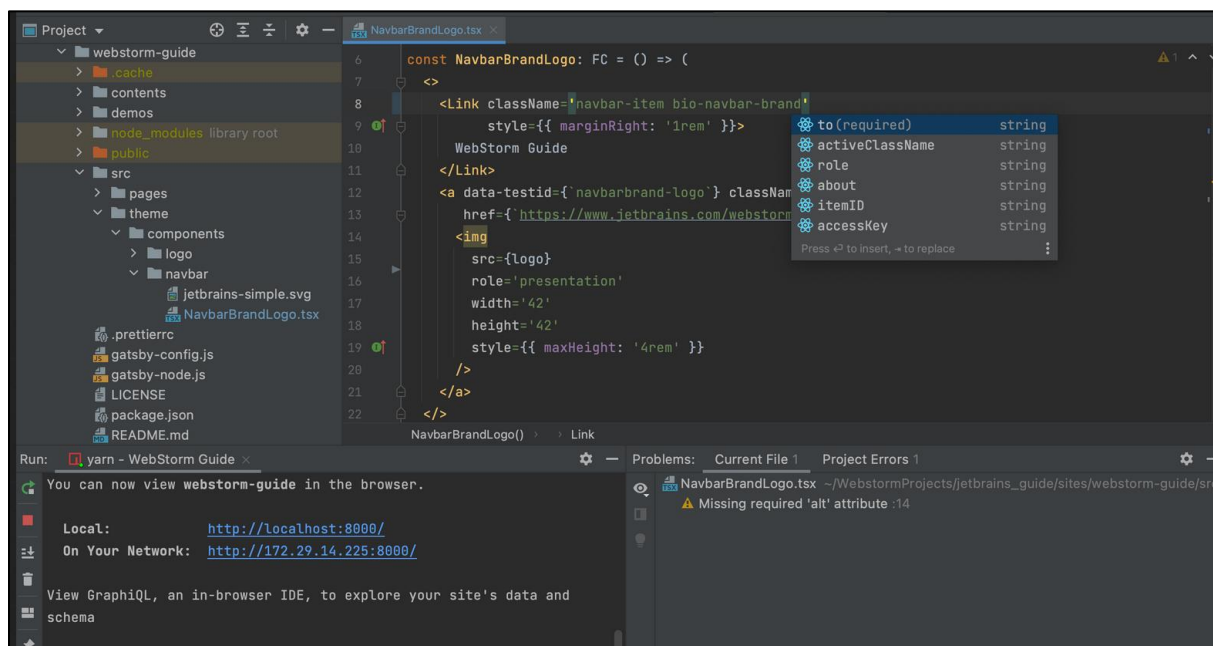


Рисунок 5 – Среда разработки WebStorm

2.2.3 Система управления базами данных: SQLite

Для хранения данных, полученных и обработанных платформой (заказы, профили, товары), выбрана встраиваемая СУБД SQLite [9].

Преимущества: в отличие от тяжелых клиент-серверных СУБД, SQLite не требует развертывания отдельного сервера. База данных хранится в одном файле, что упрощает резервное копирование и перенос данных.

Для работы с БД используется ORM-система Entity Framework Core. Она позволяет манипулировать данными через объекты C# без написания ручных SQL-запросов, обеспечивая безопасность от SQL-инъекций и повышая скорость разработки.

Просмотр структуры БД в DB Browser for SQLite показаны на рисунке 6.

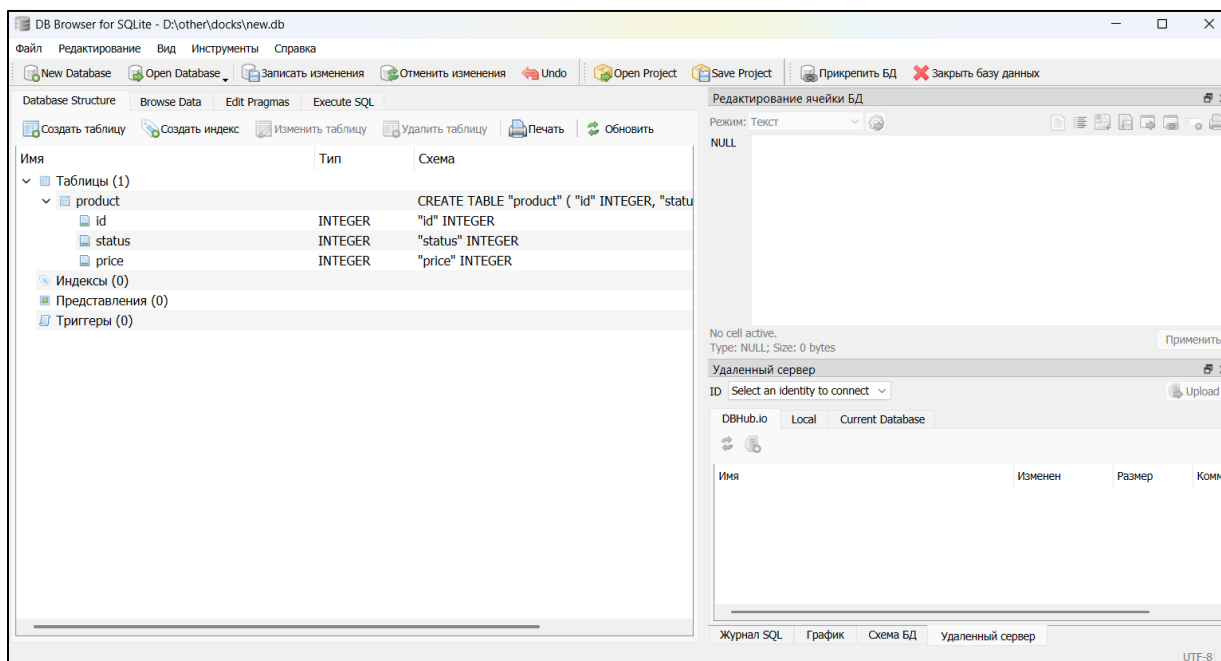


Рисунок 6 – Просмотр структуры БД в DB Browser for SQLite

2.2.4 Инструменты контроля версий и организации процесса

Для обеспечения целостности проекта и организации командной разработки используется система контроля версий Git. Она позволяет отслеживать все изменения в коде, возвращаться к предыдущим версиям и разделять разработку функционала на независимые ветки (feature-branches). В качестве централизованного хранилища используется платформа GitHub, что обеспечивает доступность кода и возможность автоматизации тестирования (CI/CD).

2.3 Установка и настройка рабочих сред

Для реализации программной части веб-платформы требуется предварительная установка и настройка интегрированных сред разработки как для серверного, так и для клиентского уровней приложения, а также подготовка инструментария для работы с базой данных.

2.3.1 Установка и настройка Visual Studio 2022 и серверного окружения

Для разработки серверной логики на базе фреймворка ASP.NET Core была выбрана интегрированная среда Microsoft Visual Studio 2022. Для начала

установки необходимо скачать дистрибутив Visual Studio Installer с официального сайта корпорации Microsoft. Подробный процесс базовой установки среды представлен на рисунках 7 – 9.

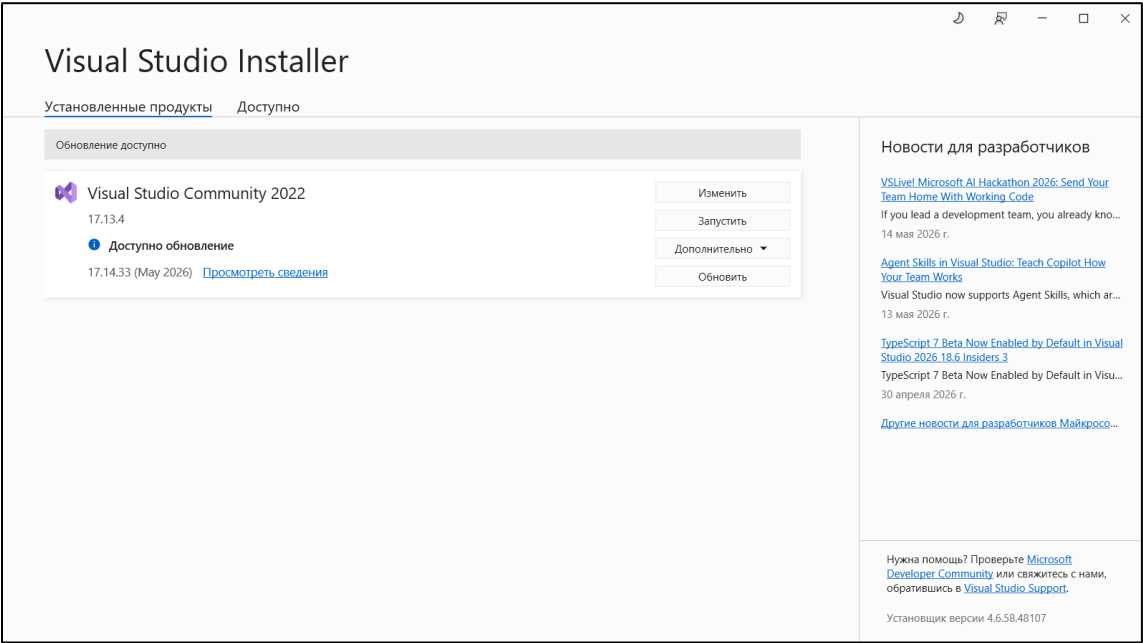


Рисунок 7 – Начальное окно мастера установки Visual Studio

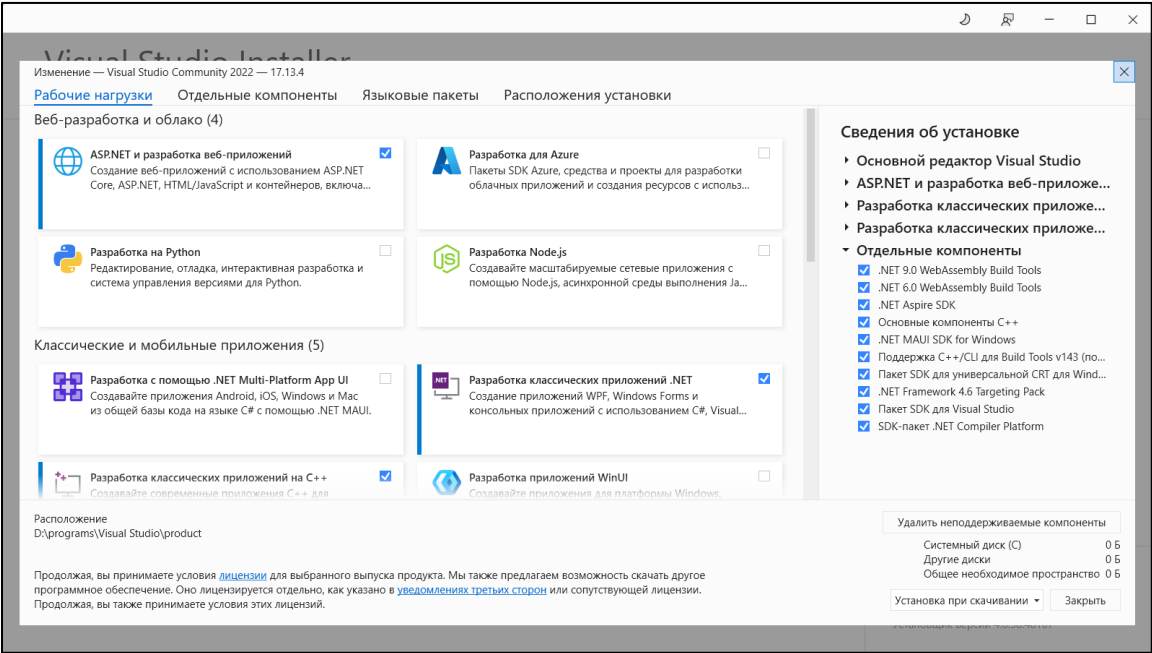


Рисунок 8 – Выбор необходимых компонентов для установки

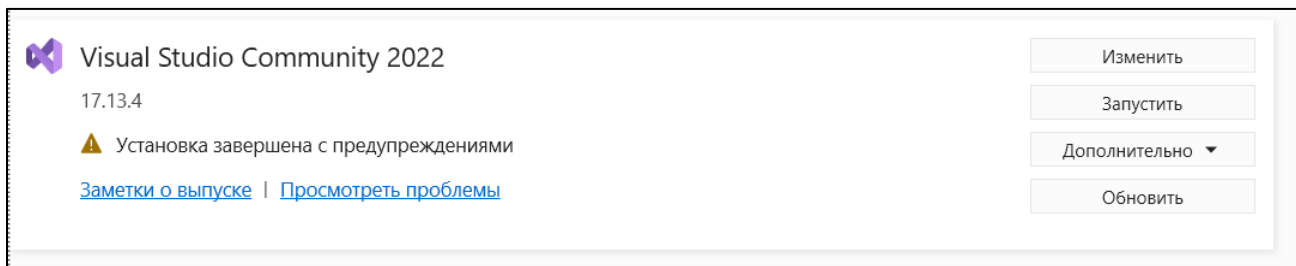


Рисунок 9 – Уведомление о завершении установки

После завершения инсталляции Visual Studio 2022 необходимо перейти к созданию и первичной настройке серверного проекта. Подробный процесс генерации базового шаблона API представлен на рисунках 10 – 14.

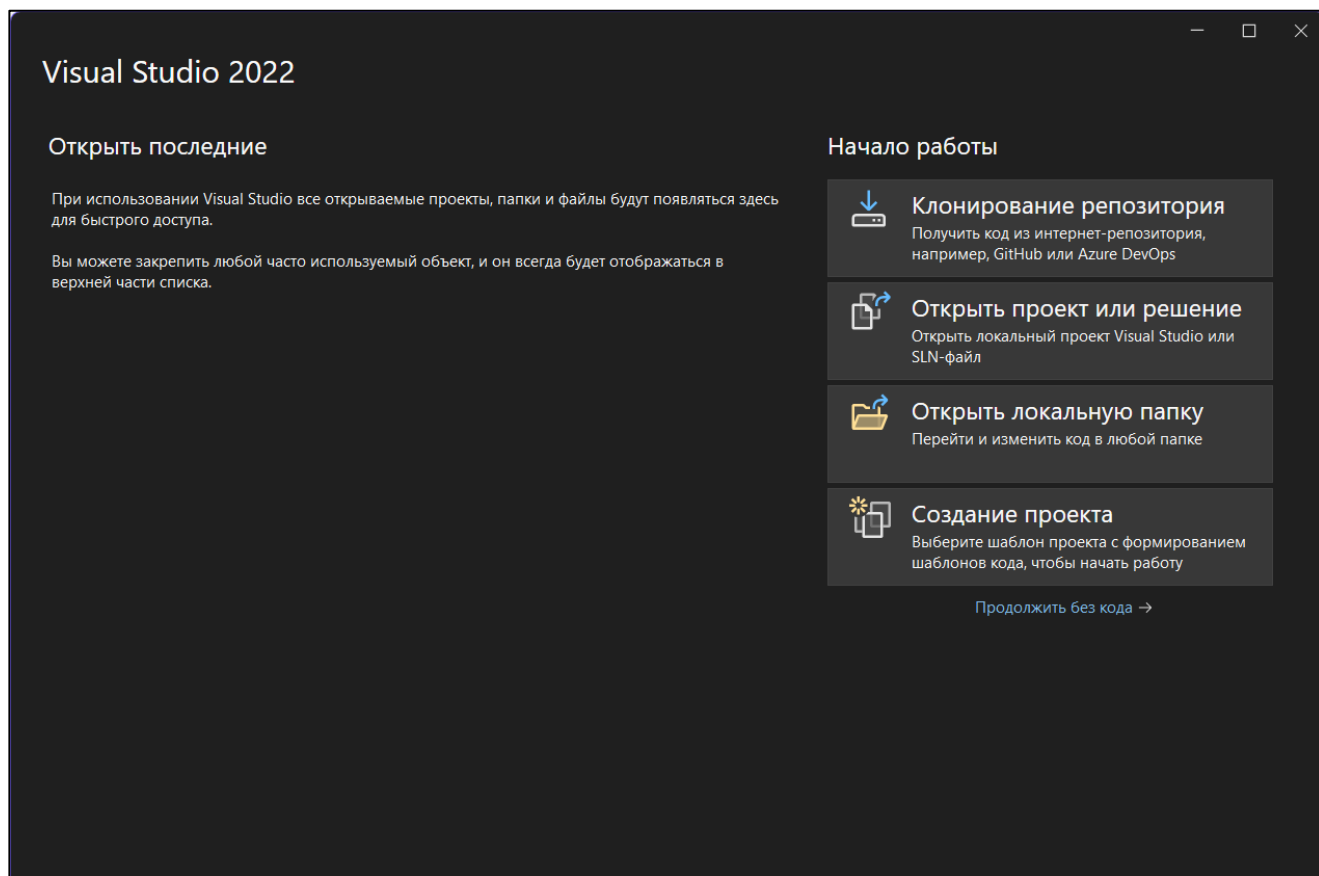


Рисунок 10 – Начальное окно Visual Studio 2022

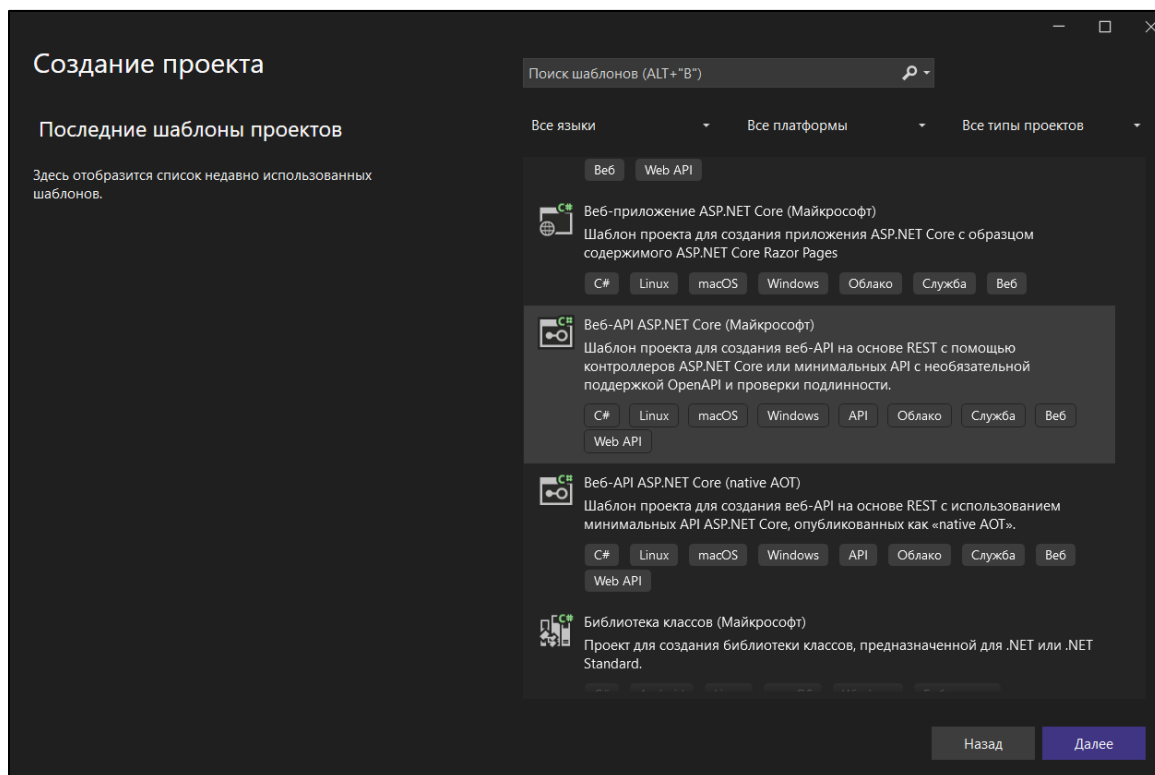


Рисунок 11 – Выбор шаблона для серверного проекта

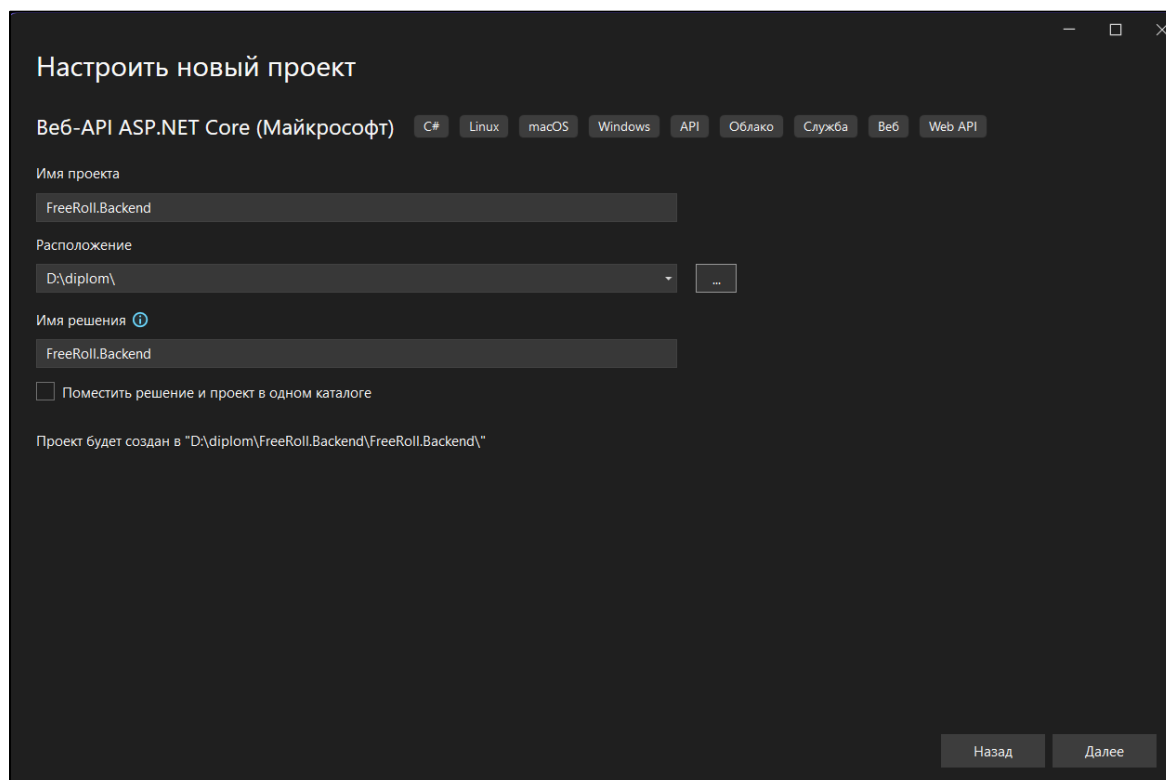


Рисунок 12 – Указание расположения и названия проекта

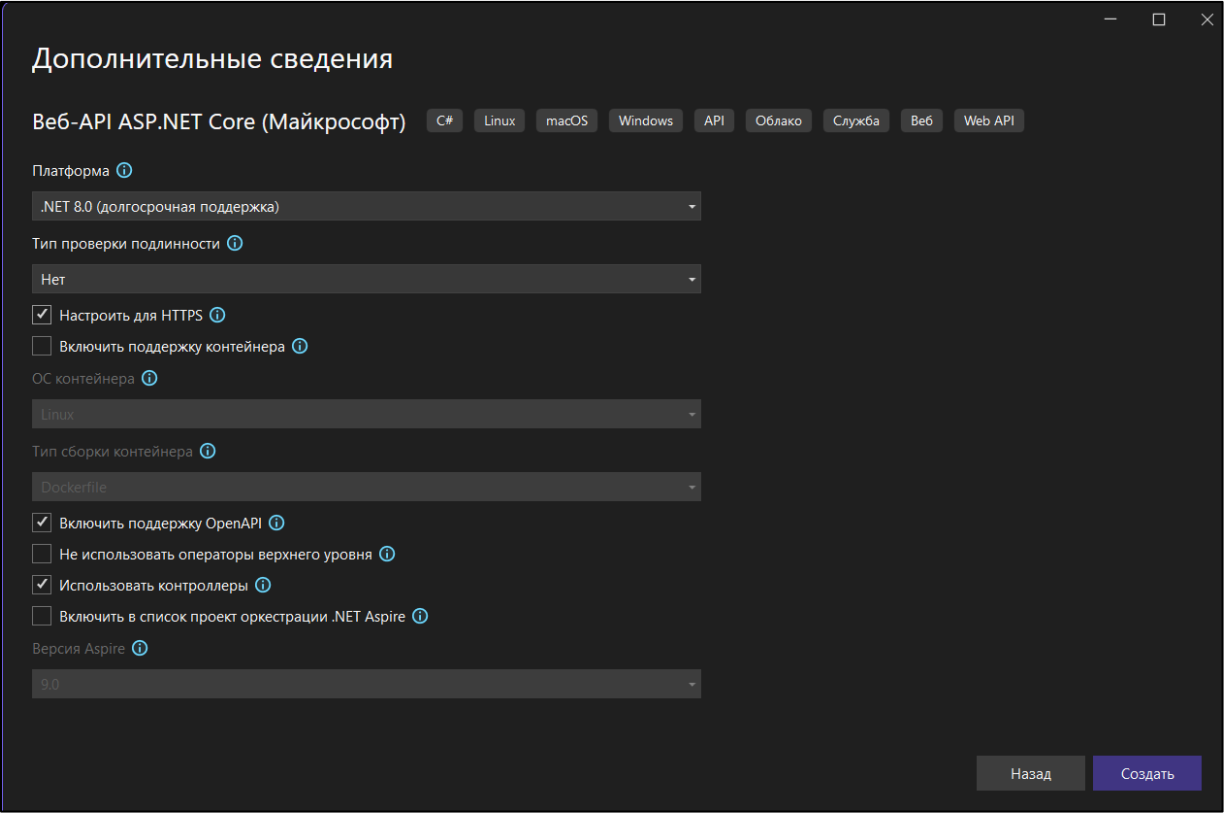


Рисунок 13 – Настройка дополнительных параметров платформы

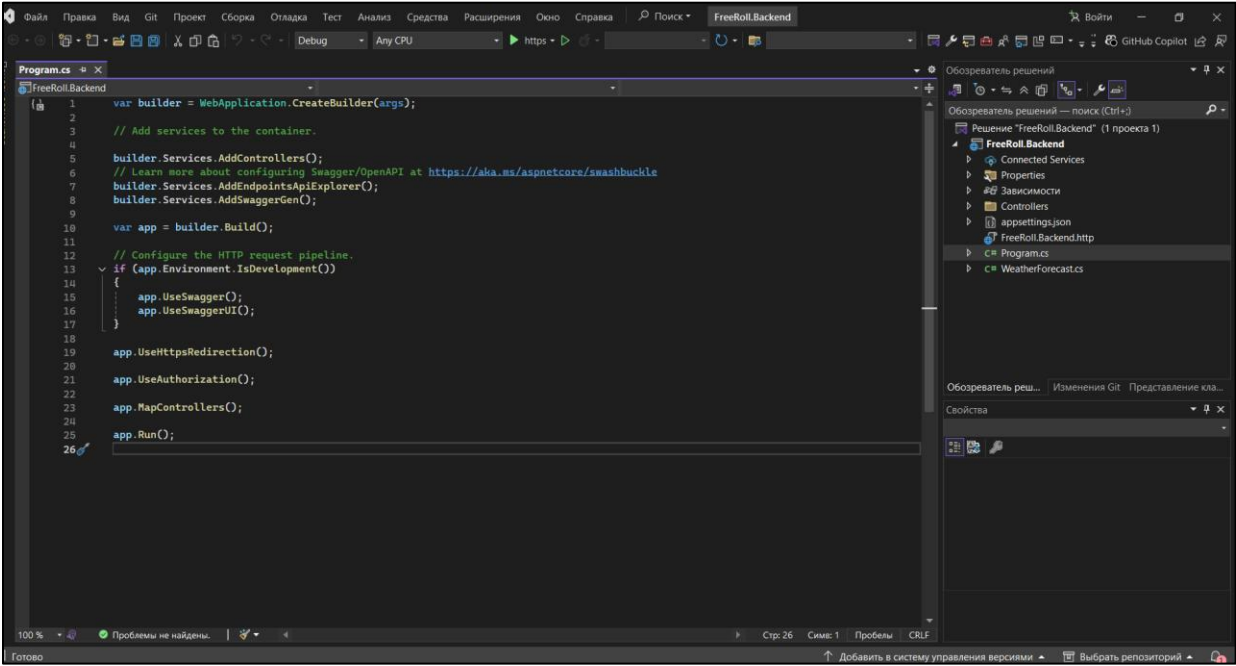


Рисунок 14 – Сгенерированная структура серверного проекта

Для обеспечения работы с данными в проекте используется встраиваемая СУБД SQLite. Поскольку базовый шаблон ASP.NET Core не содержит встроенных драйверов для этой СУБД, необходимо установить пакеты объектно-реляционного отображения (ORM) Entity Framework Core через встроенный диспетчер пакетов NuGet. Подробная настройка взаимодействия с базой данных представлена на рисунках 15 – 17.

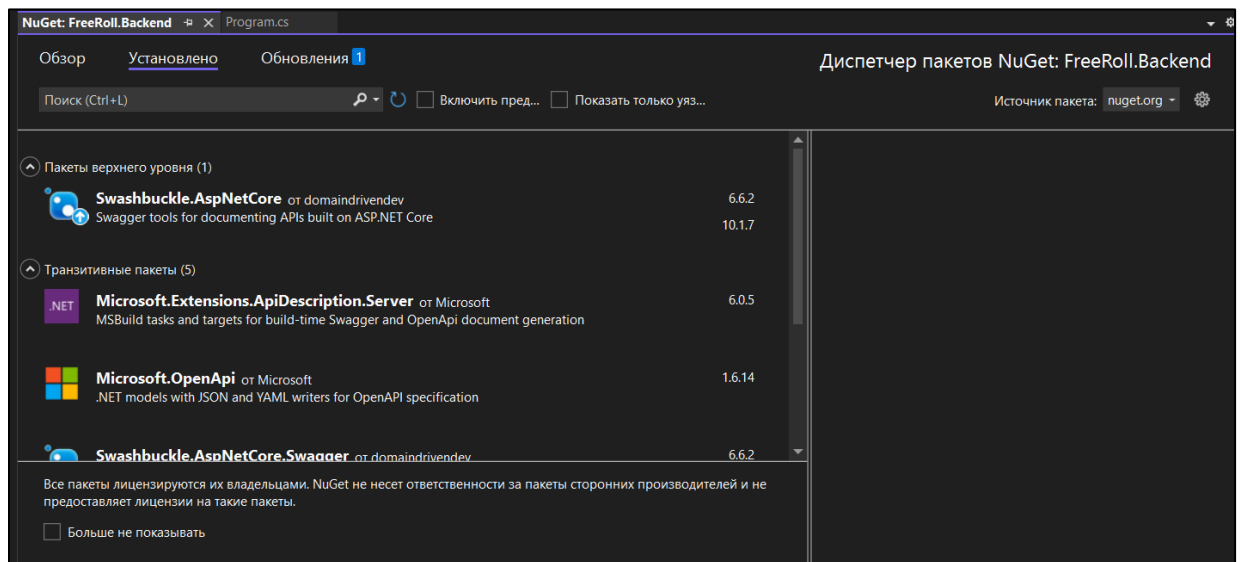


Рисунок 15 – Окно диспетчера пакетов NuGet

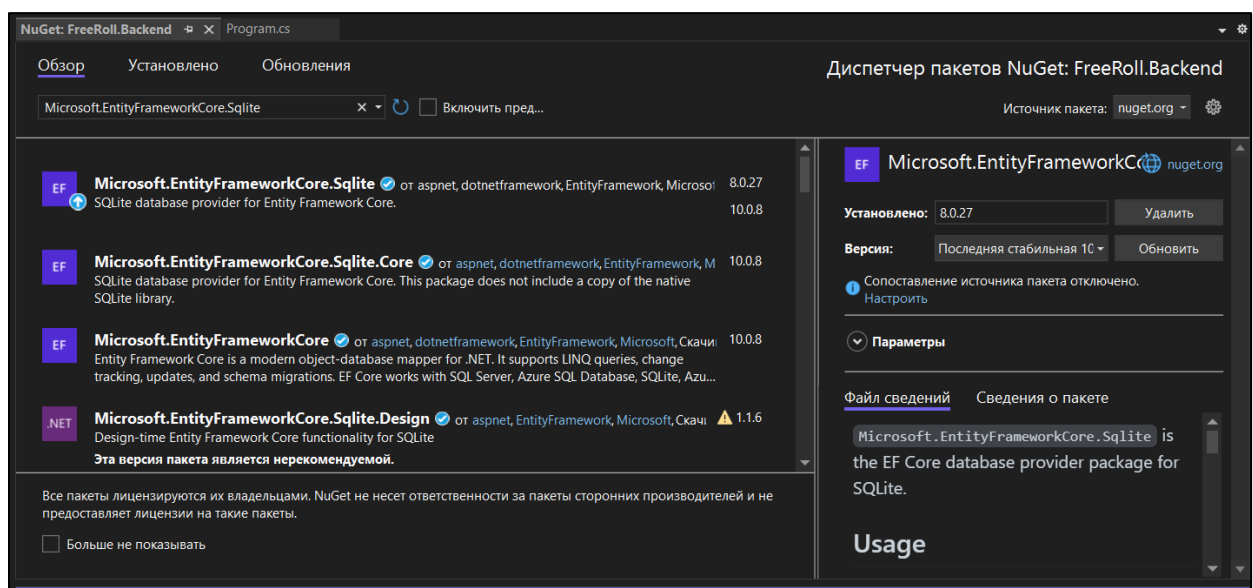


Рисунок 16 – Установка пакета для работы с SQLite

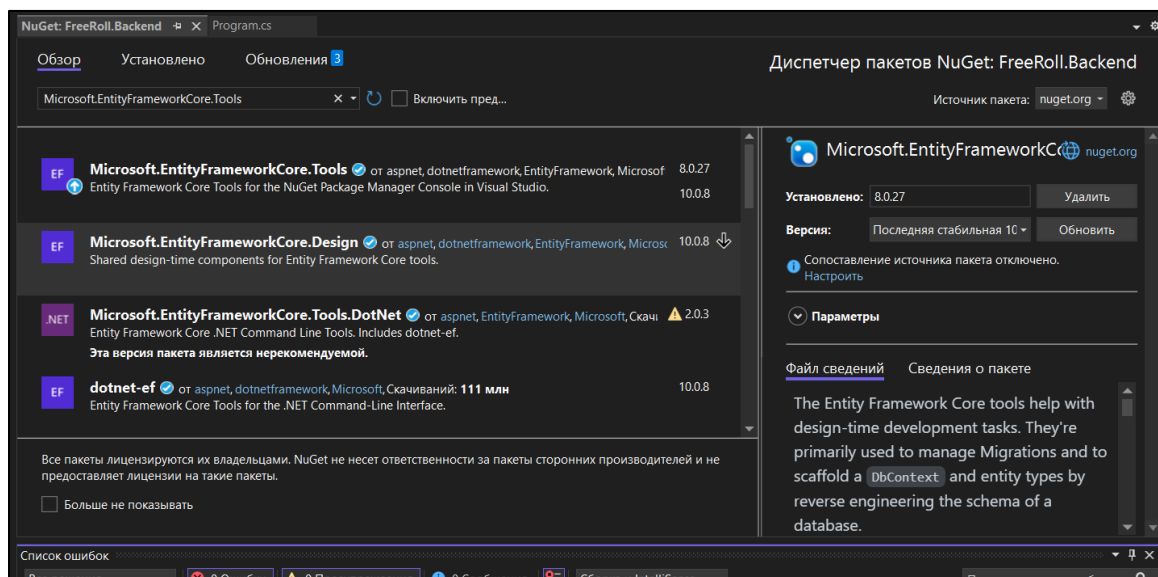


Рисунок 17 – Установка инструментария для работы с миграциями базы данных

2.3.2 Установка и настройка WebStorm и клиентского окружения

Для разработки клиентской части системы на библиотеке React требуется среда выполнения JavaScript. В качестве такой среды выступает платформа Node.js. С официального сайта был загружен LTS-дистрибутив Node.js. Процесс установки платформы представлен на рисунках 18 – 22.

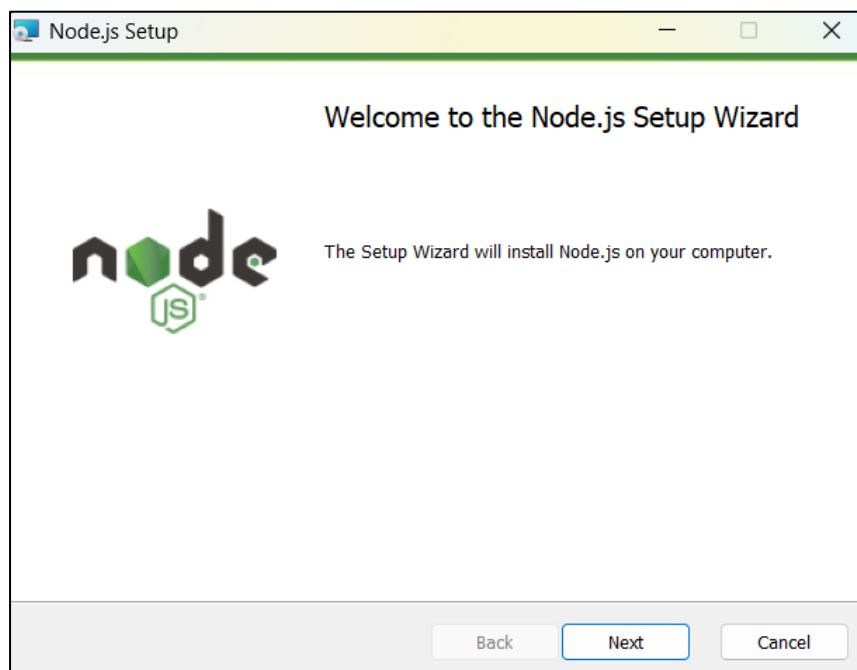


Рисунок 18 – Окно приветствия мастера установки Node.js

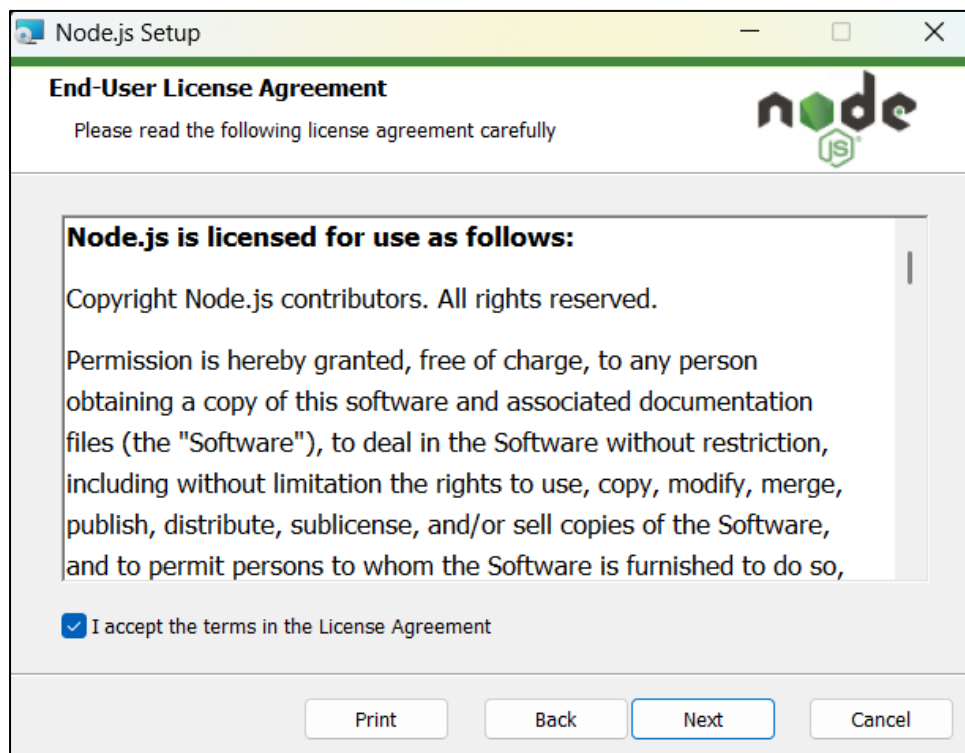


Рисунок 19 – Принятие условий лицензионного соглашения

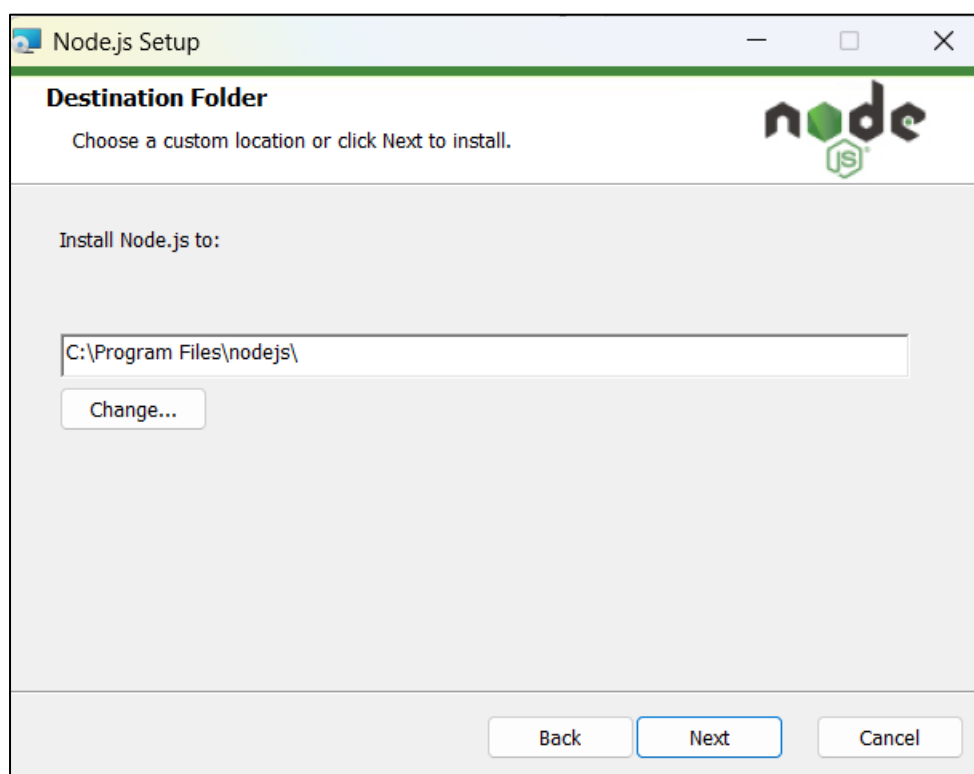


Рисунок 20 – Выбор расположения для установки

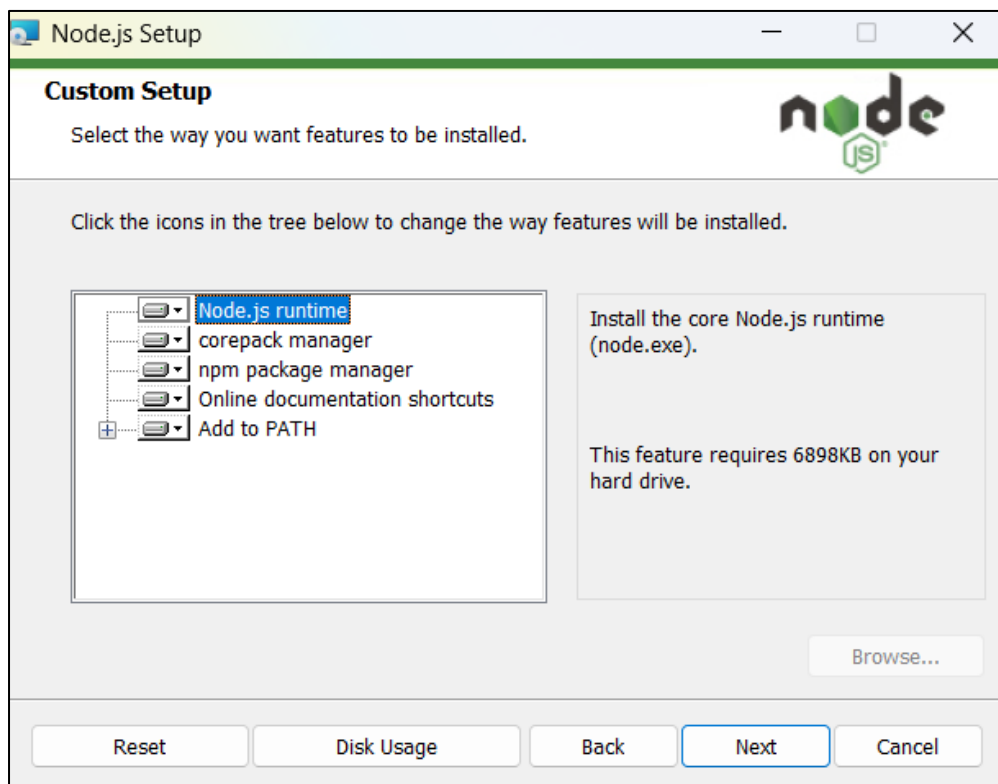


Рисунок 21 – Выбор компонентов среды выполнения

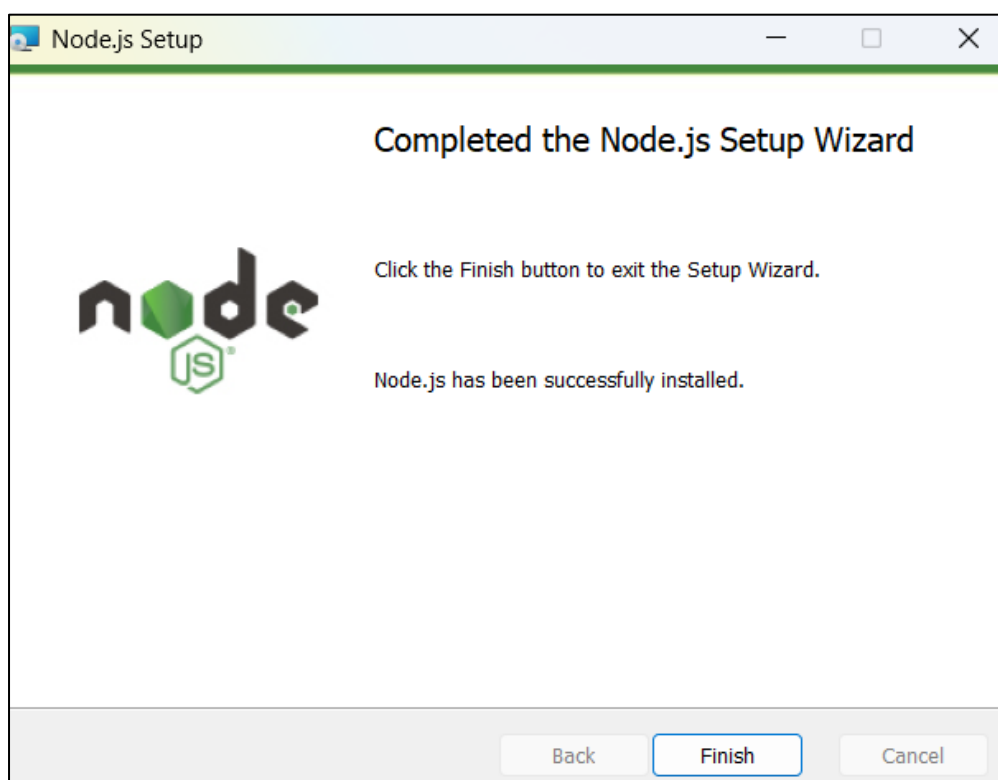


Рисунок 22 – Завершение установки платформы Node.js

В качестве интегрированной среды для написания фронтенд-кода была выбрана среда WebStorm от компании JetBrains, предоставляющая передовые инструменты для анализа кода на JavaScript. После загрузки и стандартной установки IDE необходимо инициализировать проект клиентской части. В качестве сборщика проекта был выбран инструмент Vite. Процесс установки и создания клиентского приложения представлен на рисунках 23 – 29.

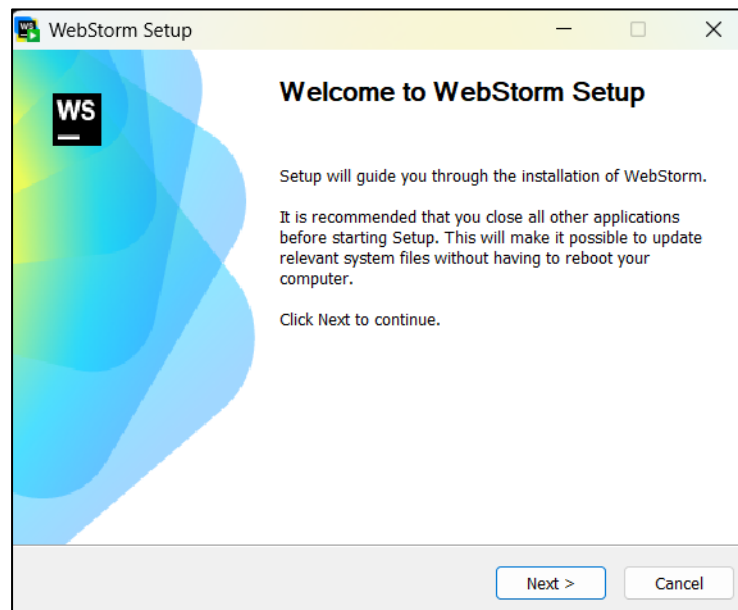


Рисунок 23 – Окно приветствия мастера установки WebStorm

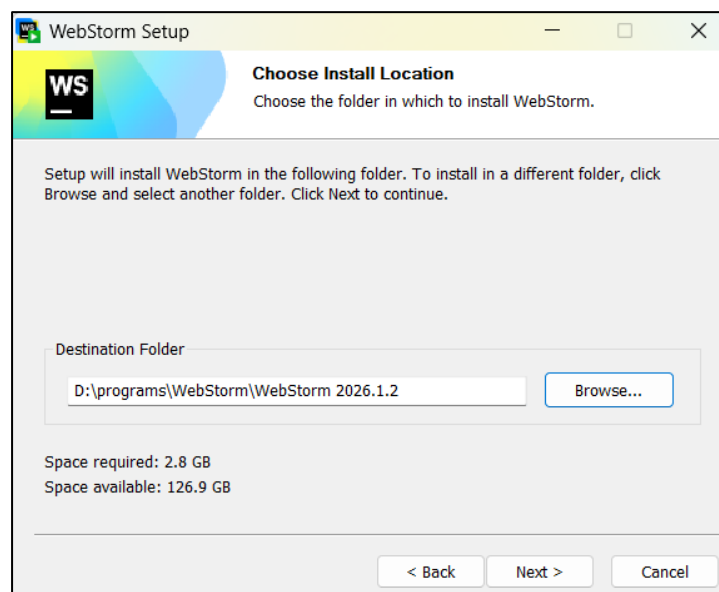


Рисунок 24 – Выбор расположения для установки

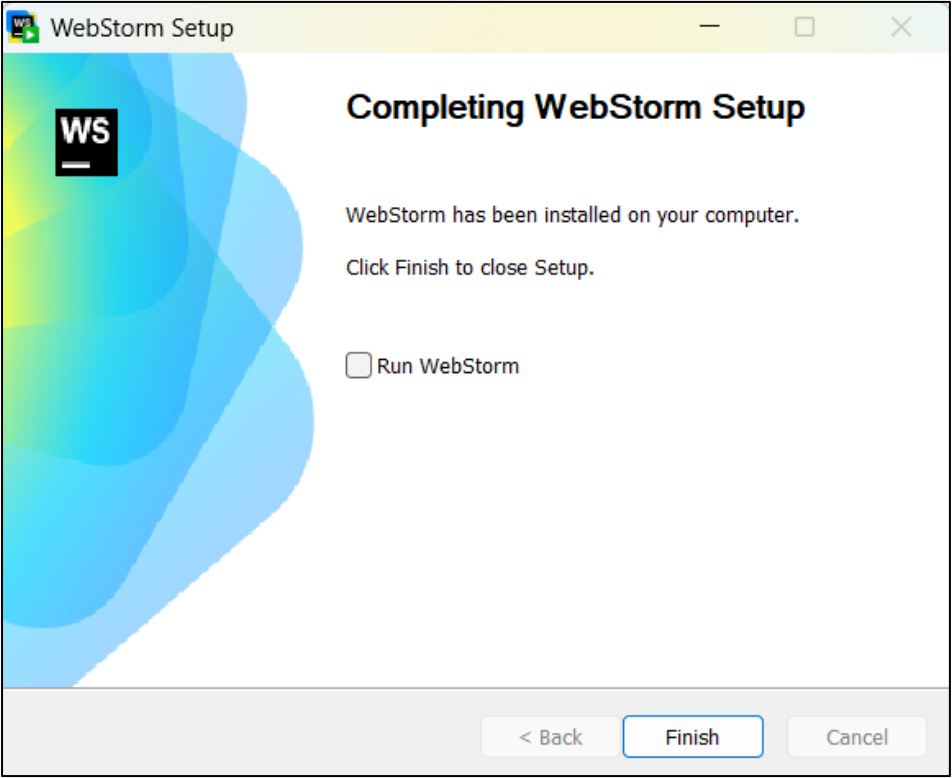


Рисунок 25 – Завершение установки платформы WebStorm

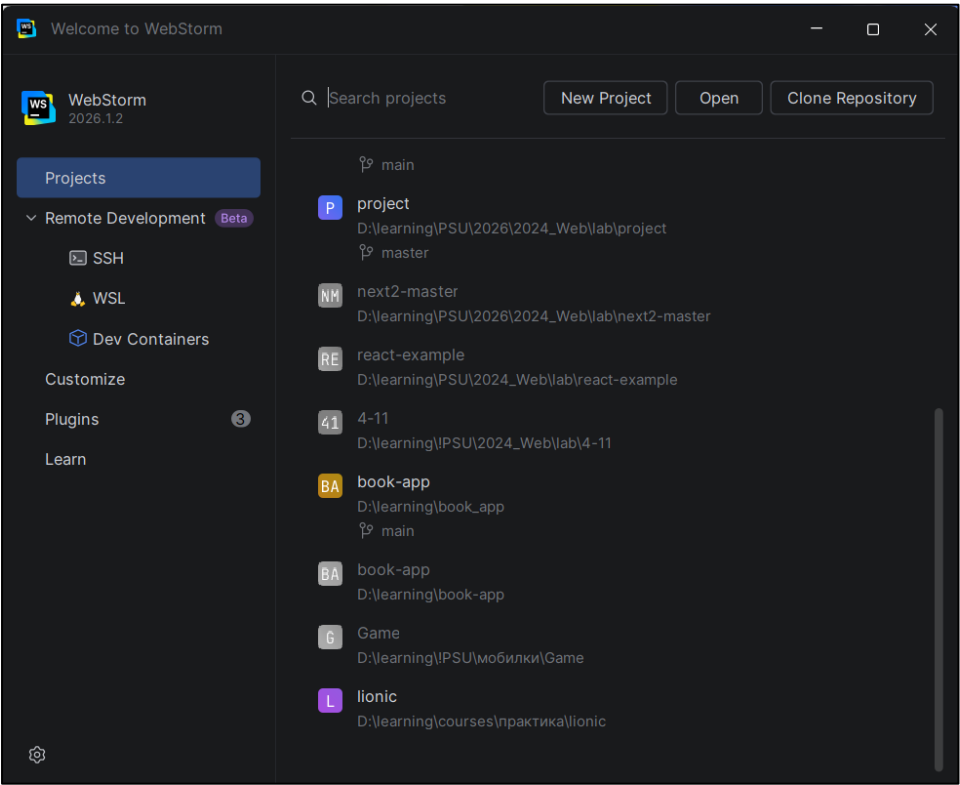


Рисунок 26 – Начальное окно среды разработки WebStorm

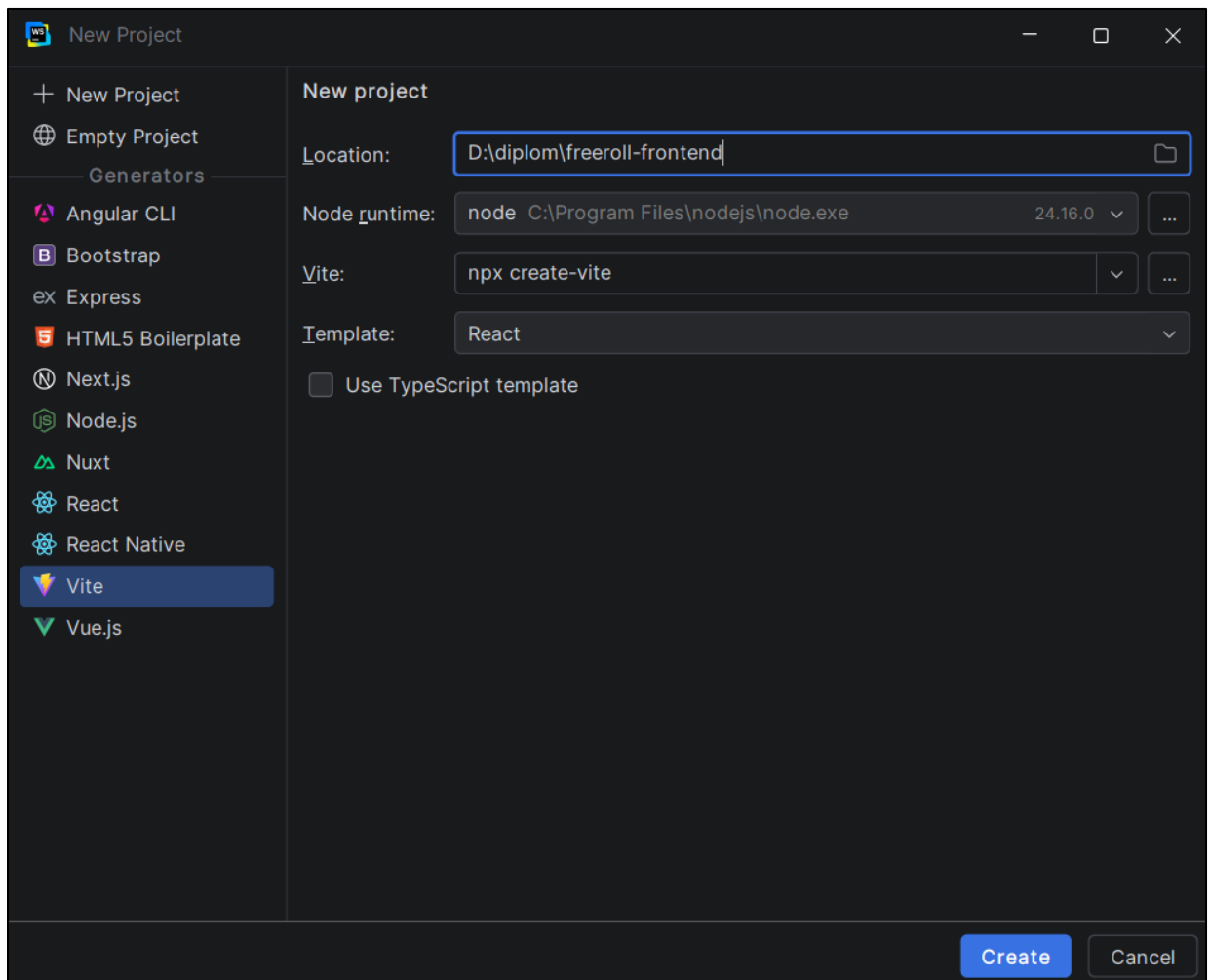


Рисунок 27 – Конфигурация параметров клиентского проекта

```
PS D:\diplom\freeroll-frontend> npm install

added 135 packages, and audited 136 packages in 15s

31 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 28 – Загрузка зависимостей проекта через пакетный менеджер

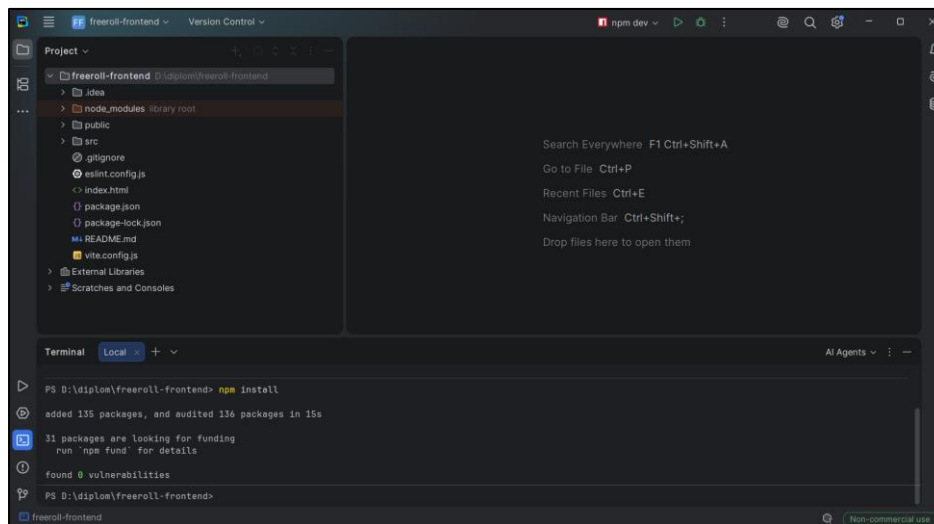


Рисунок 29 – Готовая структура клиентского веб-приложения

2.3.3 Установка утилиты для администрирования базы данных

Поскольку СУБД SQLite является встраиваемой и хранит данные в локальном файле, для визуального контроля содержимого таблиц, выполнения тестовых SQL-запросов и отладки связей между сущностями требуется специализированное программное обеспечение. В качестве такого инструмента была выбрана свободно распространяемая утилита DB Browser for SQLite. Процесс установки представлен на рисунках 30 – 32.

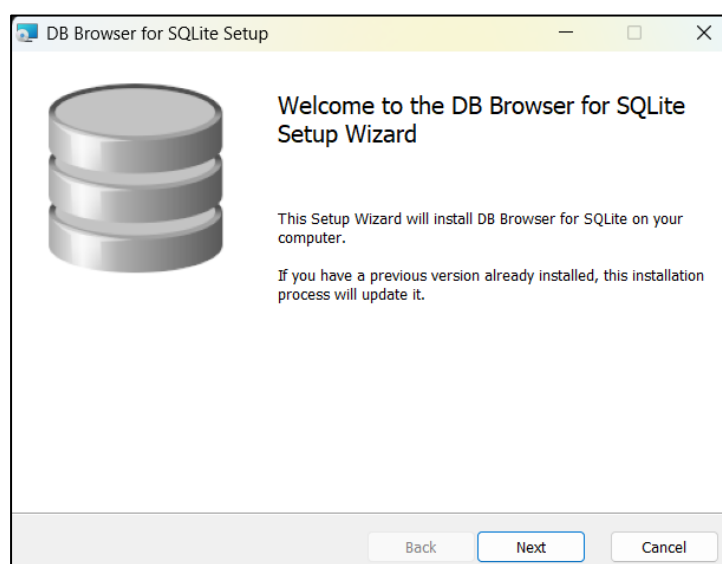


Рисунок 30 – Начальное окно мастера установки DB Browser for SQLite

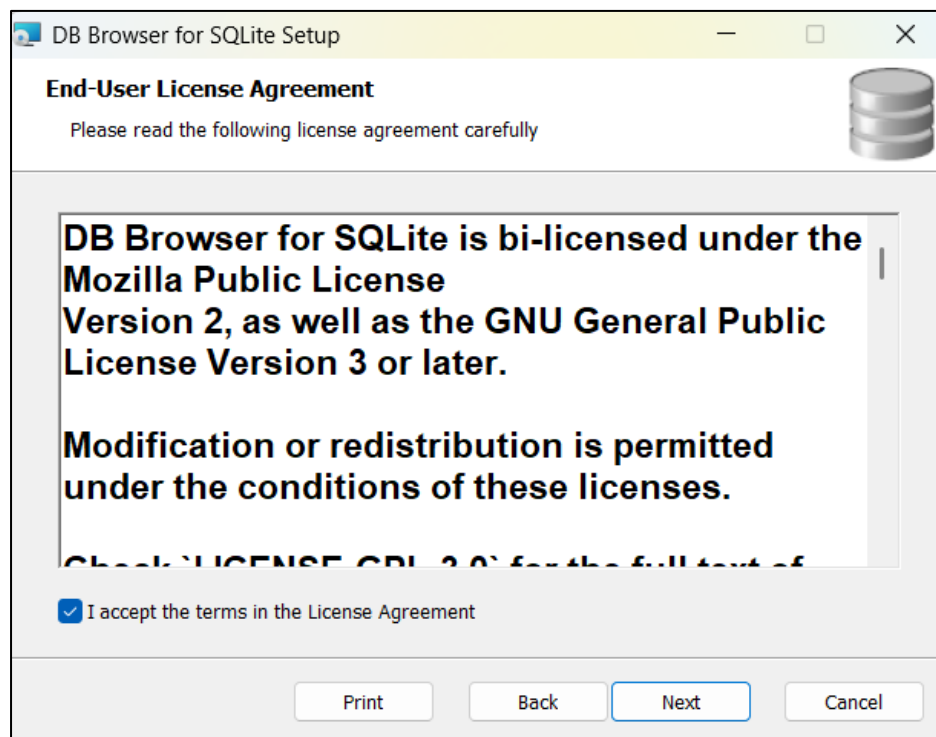


Рисунок 31 – Принятие условий лицензионного соглашения

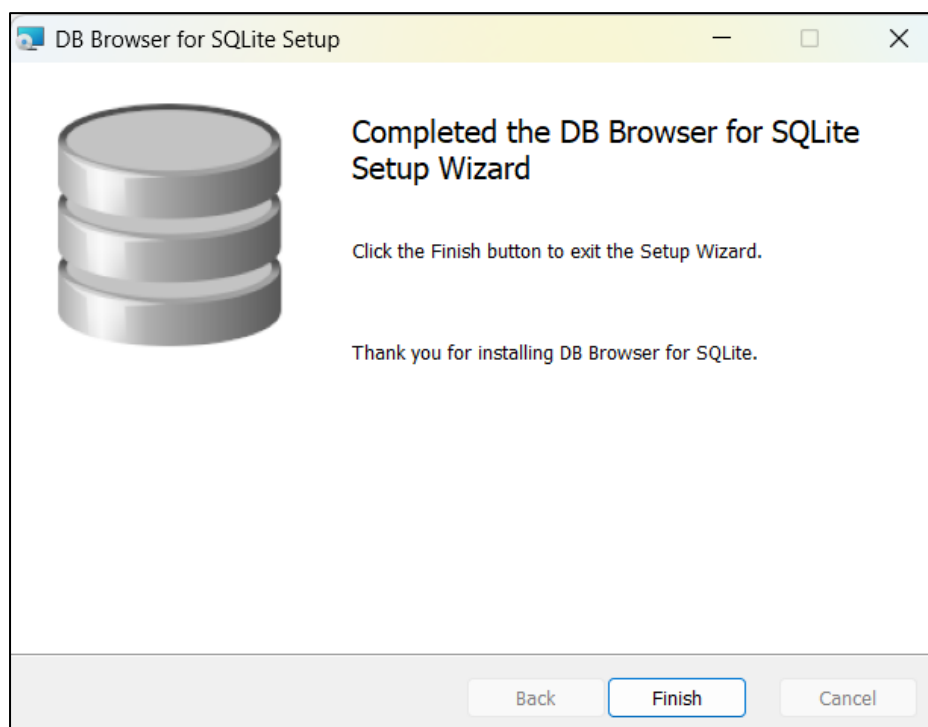


Рисунок 32 – Завершение установки графической утилиты

3 Реализация и тестирование системы

3.1 Структура БД

Вся информация о товарном каталоге, учетных записях пользователей, оформленных заказах и системных настройках сохраняется в локальной реляционной базе данных под управлением встроенной СУБД SQLite. Структура базы данных спроектирована с учетом требований нормализации, что позволяет минимизировать избыточность хранимой информации и обеспечить ссылочную целостность при выполнении транзакций.

Взаимодействие серверного приложения с базой данных осуществляется не через прямые SQL-запросы, а с использованием технологии объектно-реляционного отображения (ORM) Entity Framework Core. Данный подход позволяет оперировать записями в таблицах как строго типизированными объектами языка C#, а также обеспечивает автоматическую генерацию безопасных SQL-инструкций, защищенных от атак типа SQL-инъекций.

Центральным элементом подсистемы хранения выступает класс контекста данных `AppDbContext`, наследуемый от базового класса `DbContext`. В нем определены следующие основные таблицы (наборы данных `DbSet`):

- `Products` – таблица хранения номенклатуры товарного каталога;
- `Orders` – таблица для фиксации оформленных заказов клиентов и их статусов;
- `OrderItems` – связующая таблица, хранящая детализированный состав каждого конкретного заказа (выбранные позиции, количество, цена на момент покупки);
- `Users` – таблица учетных записей пользователей, содержащая идентификационные данные и криптографические хэши паролей;
- `Categories` – справочник категорий меню для логической группировки товаров;

- Promos – таблица активных промо-акций, скидок и условий их применения;
- Settings – хранилище динамических глобальных настроек веб-платформы;
- Reviews – таблица пользовательских отзывов к заказам.

Для обеспечения строгой ссылочной целостности на уровне СУБД используется механизм Fluent API. В частности, при инициализации модели переопределяется метод `OnModelCreating`, где жестко задается связь между строкой чека (`OrderItem`) и ссылкой на базовый товар (`Product`) через внешний ключ. Фрагмент кода, описывающий инициализацию контекста базы данных и настройку связей, представлен ниже.

```
using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Models;

namespace FreeRoll.Backend.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options) { }

        public DbSet<Product> Products { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<OrderItem> OrderItems { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Promo> Promos { get; set; }
        public DbSet<AppSetting> Settings { get; set; }
        public DbSet<Review> Reviews { get; set; }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<OrderItem>()
                .HasOne(i => i.Product)
                .WithMany()
                .HasForeignKey(i => i.ProductId);
        }
    }
}
```

Наиболее комплексной и критически важной для бизнес-логики структурой обладает таблица заказов Orders. Она аккумулирует в себе контактную информацию покупателя, финансовые показатели и подробную историю изменения состояний заказа, что необходимо для построения аналитических отчетов. В таблице Orders определены следующие ключевые поля:

- Id – первичный ключ, уникальный целочисленный идентификатор заказа;
- CustomerName и Phone – строковые поля для хранения контактных данных клиента;
- Address – текстовое поле, содержащее физический адрес доставки;
- TotalPrice – числовое поле для фиксации итоговой стоимости сформированной корзины;
- PaymentMethod – строковое поле, определяющее выбранный способ оплаты (по умолчанию безналичный расчет);
- DeliveryNotice – дополнительная текстовая информация о желаемом времени доставки;
- Status – поле для хранения текущего этапа жизненного цикла заказа. По умолчанию принимает значение «New».

Особое внимание при проектировании структуры БД было уделено реализации системы отслеживания стадий выполнения заказа. Для этого в модель Order внедрен набор временных меток (timestamps), допускающих значения NULL: CreatedAt (момент оформления), CookingAt (передача на кухню), DeliveringAt (передача курьеру) и CompletedAt (успешное завершение). Наличие этих полей позволяет серверной части вычислять точное время, затраченное на каждый этап обслуживания, и формировать статистику эффективности работы заведения.

Детализированный состав каждого заказа хранится в подчиненной таблице OrderItems. Каждая запись в этой таблице содержит внешний ключ OrderId, идентификатор купленного товара ProductId и его количество Count. Важным архитектурным решением является фиксация поля Price (стоимость позиции на

момент покупки) непосредственно в строке чека. Это гарантирует, что историческая стоимость оформленных заказов останется неизменной даже в случае последующего изменения цен в основном товарном каталоге.

Для безопасного приема данных от клиентского веб-приложения используется паттерн Data Transfer Object (DTO). Структуры OrderCreateDto и OrderItemCreateDto служат входными контрактами API, фильтруя избыточные данные и защищая внутреннюю модель БД от некорректной модификации со стороны пользователя. Фрагмент кода, описывающий базовые модели заказа и строки чека, представлен ниже.

```
using System;
using System.Collections.Generic;

namespace FreeRoll.Backend.Models
{
    public class Order
    {
        public int Id { get; set; }
        public string CustomerName { get; set; } = string.Empty;
        public string Phone { get; set; } = string.Empty;
        public string Address { get; set; } = string.Empty;
        public decimal TotalPrice { get; set; }

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public DateTime? CookingAt { get; set; }
        public DateTime? DeliveringAt { get; set; }
        public DateTime? CompletedAt { get; set; }

        public string Status { get; set; } = "New";
        public string PaymentMethod { get; set; } = "card";

        public string DeliveryNotice { get; set; } = " Как можно быстрее";

        public List<OrderItem> Items { get; set; } = new();
    }

    public class OrderItem
    {
        public int Id { get; set; }
        public int OrderId { get; set; }
        public int ProductId { get; set; }
        public Product? Product { get; set; }
        public int Count { get; set; }
    }
}
```

```

        public decimal Price { get; set; }
    }

    public class OrderCreatedto
    {
        public string CustomerName { get; set; } = string.Empty;
        public string Phone { get; set; } = string.Empty;
        public string Address { get; set; } = string.Empty;
        public decimal TotalPrice { get; set; }
        public string PaymentMethod { get; set; } = "card";
        public string DeliveryNotice { get; set; } = " Как можно
быстрее";

        public List<OrderItemCreatedto> Items { get; set; } = new();
    }

    public class OrderItemCreatedto
    {
        public int ProductId { get; set; }
        public int Count { get; set; }
        public decimal Price { get; set; }
    }
}

```

Для наглядного представления архитектуры подсистемы хранения данных была спроектирована диаграмма «сущность-связь». Она демонстрирует логические отношения между таблицами, первичные ключи (PK) и направления внешних ключей (FK). Графическое представление структуры используемой БД представлено в Приложении Б.1.

3.2 Разработка серверной части

Серверная часть платформы «FreeRoll» спроектирована в соответствии с архитектурным стилем REST (Representational State Transfer) и реализована на базе высокопроизводительного кроссплатформенного фреймворка ASP.NET Core Web API. Главной задачей серверного уровня является прием и маршрутизация HTTP-запросов от клиентского веб-приложения, выполнение заложенных алгоритмов бизнес-логики, взаимодействие с базой данных SQLite и формирование стандартизированных JSON-ответов.

Архитектура бэкенда построена на принципе внедрения зависимостей (Dependency Injection, DI). В точке входа приложения (конфигурационный файл

Program.cs) осуществляется централизованная регистрация всех сервисов: контекста базы данных AppDbContext, служб авторизации, политик CORS (Cross-Origin Resource Sharing) для обеспечения безопасного междоменного взаимодействия с React-клиентом, а также автоматического генератора документации API (Swagger).

Логика обработки запросов строго декомпозирована и вынесена в специализированные классы-контроллеры, унаследованные от базового класса ControllerBase.

3.2.1 Подсистема аутентификации и управления профилями

Модуль аутентификации отвечает за безопасную регистрацию новых клиентов, проверку учетных данных и выдачу сессионных прав доступа. В целях обеспечения информационной безопасности пароли пользователей никогда не хранятся в базе данных в открытом виде.

При обработке запроса на регистрацию контроллер извлекает переданные пользователем данные, после чего применяет криптографическую хеш-функцию. Для хеширования используется современный алгоритм BCrypt, который программно замедляет процесс генерации хеша, делая атаки методом прямого перебора (brute-force) вычислительно нецелесообразными. Только после успешного хеширования объект User сохраняется в базу данных.

При попытке авторизации система извлекает пользователя по логину, вычисляет хеш от введенного пароля и сравнивает его с эталонным значением в БД. При успешном совпадении сервер возвращает клиенту статус 200 ОК и данные профиля. Процесс клиент-серверного взаимодействия при авторизации проиллюстрирован на UML-диаграмме последовательности в Приложении А.2. Фрагмент кода, реализующий безопасную проверку пароля и обработку авторизации, представлен ниже.

```
[HttpPost("login")]
public async Task<IActionResult> Login([FromBody] User loginData)
{
    var adminExists = await _context.Users.AnyAsync(u => u.IsAdmin);
```

```

        if (!adminExists && loginData.Login == "admin" &&
loginData.Password == "admin")
        {
            var newAdmin = new User
            {
                Login = "admin",
                Password = BCrypt.Net.BCrypt.HashPassword("admin"),
                Name = "Администратор",
                IsAdmin = true
            };
            _context.Users.Add(newAdmin);
            await _context.SaveChangesAsync();
        }

        var user = await _context.Users
            .Include(u => u.Addresses)
            .FirstOrDefaultAsync(u => u.Login == loginData.Login);

        if (user == null ||
!BCrypt.Net.BCrypt.Verify(loginData.Password, user.Password))
        {
            return Unauthorized(new { message = "Неверный логин или
пароль!" });
        }

        user.Password = "";
        return Ok(new { message = "Успешный вход!", user });
    }

```

Дополнительно в рамках данного контроллера реализована логика управления адресной книгой клиента. Интерфейс тестирования API аутентификации представлен на рисунке 33.

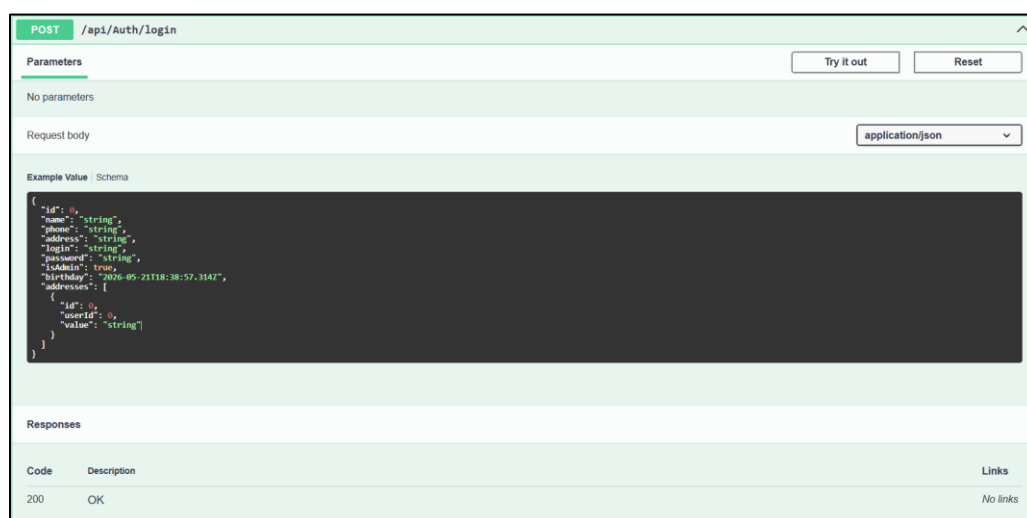


Рисунок 33 – Интерфейс тестирования API аутентификации (Swagger)

3.2.2 Модуль маршрутизации и обработки заказов

Ядром бизнес-логики платформы выступает контроллер `OrdersController`. Он отвечает за полный жизненный цикл заказа: от момента первичного формирования корзины клиентом до передачи готовых блюд курьеру и получения итоговой статистической сводки администратором.

Для защиты базы данных от некорректно сформированных запросов прием данных о новом заказе осуществляется исключительно через объекты передачи данных (DTO). Структура `OrderCreateDto` валидируется сервером, после чего алгоритм осуществляет маппинг (преобразование) входящих данных во внутреннюю модель `Order`. При этом серверная часть автоматически устанавливает текущую временную метку оформления заказа (`CreatedAt`) и начальный статус «New». Детали заказа (`OrderItems`) переносятся в БД с жесткой фиксацией текущей стоимости (`Price`), чтобы защитить историю покупок от будущих изменений цен в каталоге. Подробная схема взаимодействия компонентов при оформлении заказа представлена в Приложении А.3.

Особую важность в данном модуле представляет логика управления "Воронкой статусов". Для обеспечения прозрачной аналитики контроллер содержит специализированные методы для администратора (например, `PATCH /api/orders/{id}/status`). При изменении статуса заказа алгоритм не только обновляет строковое поле `Status`, но и автоматически фиксирует время прохождения этапа с помощью встроенных в модель свойств, допускающих значение `Null`.

Фрагмент кода, демонстрирующий логику автоматической фиксации временных меток при движении заказа по производственной воронке, представлен ниже.

```
[HttpPatch("{id}/status")]
public async Task<IActionResult> UpdateStatus(int id, [FromBody]
string newStatus)
{
    var order = await _context.Orders.FindAsync(id);
    if (order == null) return NotFound(new { message = "Заказ не
найден" });
```

```

order.Status = newStatus;

var now = DateTime.SpecifyKind(DateTime.UtcNow,
DateTimeKind.Utc);

if (newStatus == "Cooking" && order.CookingAt == null)
    order.CookingAt = now;

if (newStatus == "Delivering" && order.DeliveringAt == null)
    order.DeliveringAt = now;

if (newStatus == "Completed" && order.CompletedAt == null)
    order.CompletedAt = now;

await _context.SaveChangesAsync();
return Ok(new { message = "Статус заказа обновлен!", status =
order.Status });
}

```

Такой архитектурный подход позволяет в дальнейшем агрегировать данные на стороне сервера и предоставлять руководителю службы доставки детализированные отчеты: вычислять среднее время приготовления (CookingAt минус CreatedAt), среднее время доставки курьером (CompletedAt минус DeliveringAt) и определять проблемы в производственных процессах. Процесс программного тестирования эндпоинта создания заказа представлен на рисунке 34.

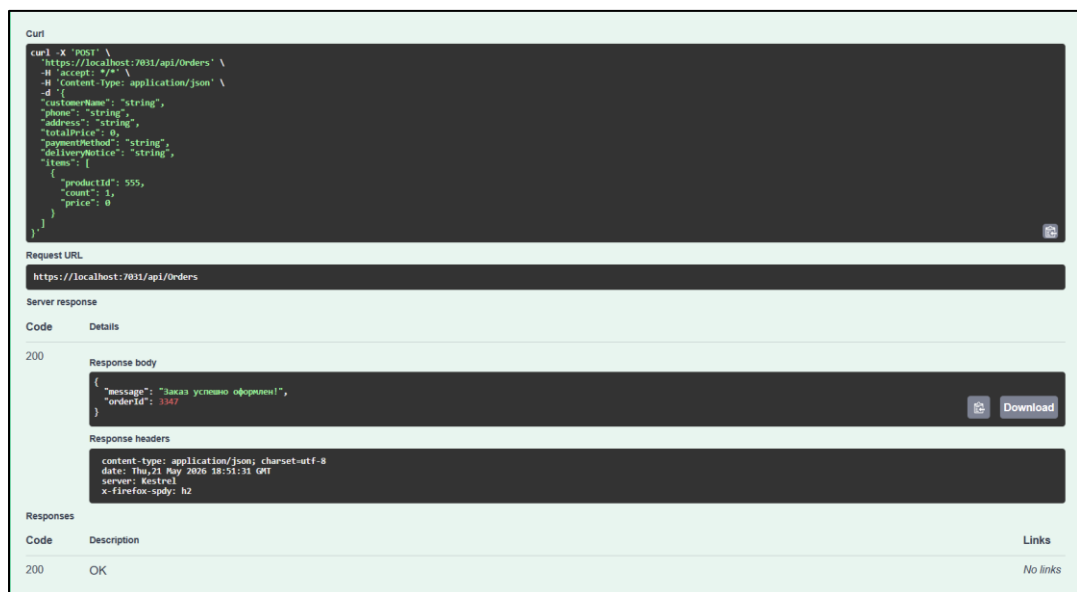


Рисунок 34 – Процесс программного тестирования эндпоинта создания заказа

3.2.3 Модуль аналитики и формирования статистики

Для обеспечения руководства службы доставки инструментами контроля и принятия управленческих решений, в состав OrdersController интегрирован специализированный аналитический эндпоинт. Данный модуль выполняет агрегацию исторических данных о продажах и формирует сводную статистику для административной панели в режиме реального времени.

Алгоритм генерации статистики базируется на механизмах языка интегрированных запросов (LINQ) поверх объектной модели Entity Framework Core. Для исключения искажений финансовых показателей выборка данных производится исключительно по заказам, успешно завершившим свой жизненный цикл (статус Completed), в рамках заданного временного диапазона.

Вычислительное ядро модуля производит расчет следующих ключевых бизнес-метрик:

- финансовые показатели: общая выручка (totalRevenue) и динамика продаж по дням (revenueByDay);
- товарная аналитика: определение Топ-5 наиболее популярных позиций меню на основе количества проданных единиц;
- временной анализ спроса (тепловые карты): группировка заказов по часам суток и дням недели. Это позволяет выявлять пиковые нагрузки на кухню и оптимизировать графики выхода персонала;
- анализ производственной эффективности: вычисление среднего времени, затрачиваемого на приготовление блюда (разница между DeliveringAt и CookingAt), а также среднего времени работы курьерской службы (разница между CompletedAt и DeliveringAt).

Фрагмент кода, реализующий выборку данных и расчет производственных метрик для дашборда, представлен ниже.

```
[HttpGet("stats")]  
public async Task<IActionResult> GetStats([FromQuery] DateTime? startDate, [FromQuery] DateTime? endDate)  
{
```

```

var end = endDate ?? DateTime.UtcNow;
var start = startDate ?? end.AddDays(-7);

var completedOrders = await _context.Orders
    .Include(o => o.Items)
    .ThenInclude(i => i.Product)
    .Where(o => o.Status == "Completed" && o.CreatedAt >=
start && o.CreatedAt <= end)
    .ToListAsync();

var totalRevenue = completedOrders.Sum(o => o.TotalPrice);
var totalOrders = completedOrders.Count;

var revenueByDay = completedOrders
    .GroupBy(o => o.CreatedAt.Date)
    .Select(g => new
    {
        date = g.Key.ToString("dd.MM"),
        revenue = g.Sum(o => o.TotalPrice)
    })
    .OrderBy(x => x.date)
    .ToList();

var topProducts = completedOrders
    .SelectMany(o => o.Items)
    .GroupBy(i => i.Product != null ? i.Product.Name :
"Удаленный товар")
    .Select(g => new
    {
        name = g.Key,
        count = g.Sum(i => i.Count)
    })
    .OrderByDescending(x => x.count)
    .Take(5)
    .ToList();

var ordersByHour = completedOrders
    .GroupBy(o => o.CreatedAt.ToLocalTime().Hour)
    .Select(g => new { hour = $"{g.Key:D2}:00", count =
g.Count() })
    .OrderBy(x => int.Parse(x.hour.Split(':')[0]))
    .ToList();

var daysOfWeekRus = new[] { "Бс", "Пн", "Вт", "Ср", "Чт",
"Пт", "Сб" };
var ordersByDayOfWeek = completedOrders
    .GroupBy(o => (int)o.CreatedAt.ToLocalTime().DayOfWeek)
    .Select(g => new { day = daysOfWeekRus[g.Key], count =
g.Count(), dayIndex = g.Key })
    .OrderBy(x => (x.dayIndex + 6) % 7)
    .Select(x => new { day = x.day, count = x.count })

```



```

        .ToList();

        var kitchenTimes = completedOrders
            .Where(o => o.CookingAt.HasValue &&
                o.DeliveringAt.HasValue)
            .Select(o => (o.DeliveringAt.Value -
                o.CookingAt.Value).TotalMinutes)
            .ToList();

        var avgCookingTime = kitchenTimes.Any() ?
            Math.Round(kitchenTimes.Average()) : 0;

        var deliveryTimes = completedOrders
            .Where(o => o.DeliveringAt.HasValue &&
                o.CompletedAt.HasValue)
            .Select(o => (o.CompletedAt.Value -
                o.DeliveringAt.Value).TotalMinutes)
            .ToList();

        var avgDeliveryTime = deliveryTimes.Any() ?
            Math.Round(deliveryTimes.Average()) : 0;

        return Ok(new
        {
            totalRevenue,
            totalOrders,
            revenueByDay,
            topProducts,
            ordersByHour,
            ordersByDayOfWeek,
            avgCookingTime,
            avgDeliveryTime
        });
    }
}

```

3.2.4 Подсистема работы с медиафайлами

Для обеспечения платформы графическим контентом (фотографии блюд, баннеры акций, аватары) разработана подсистема загрузки и маршрутизации медиафайлов UploadController. Данный контроллер принимает бинарные потоки данных по протоколу HTTP с использованием формата multipart/form-data.

Архитектурной особенностью модуля является динамическое управление структурой каталогов. Для предотвращения хаотичного скопления файлов в одной директории контроллер принимает строковый параметр folder, определяющий целевую подпапку (например, /uploads/products/ или /uploads/promos/). При

отсутствии указанной директории на сервере алгоритм автоматически создает ее средствами класса `Directory`.

Для оптимизации поисковой выдачи (SEO) и удобства администрирования реализована поддержка человеко-понятных URL (ЧПУ). Контроллер позволяет клиентскому приложению передавать желаемое имя файла (`fileName`). В случае отсутствия пользовательского имени система применяет фолбэк-механизм, генерируя уникальный криптографический идентификатор (GUID), что полностью исключает риск перезаписи существующих файлов с одинаковыми именами.

```
[HttpPost]
[Consumes("multipart/form-data")]
public async Task<IActionResult> UploadImage(IFormFile file,
[FromForm] string folder = "common", [FromForm] string fileName =
null)
{
    if (file == null || file.Length == 0)
        return BadRequest(new { message = "Файл не выбран" });

    try
    {
        var wwwroot = _env.WebRootPath ??
Path.Combine(Directory.GetCurrentDirectory(), "wwwroot");

        var targetFolder = Path.Combine(wwwroot, "uploads",
folder);

        if (!Directory.Exists(targetFolder))
        {
            Directory.CreateDirectory(targetFolder);
        }

        var finalFileName = string.IsNullOrEmpty(fileName)
            ? Guid.NewGuid().ToString() +
Path.GetExtension(file.FileName)
            : fileName;

        var filePath = Path.Combine(targetFolder, finalFileName);

        using (var stream = new FileStream(filePath,
FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }
    }
}
```



```

        return Ok(new { url =
$"https://localhost:7031/uploads/{folder}/{finalFileName}" });
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = ex.Message });
    }
}

```

3.2.5 Управление маркетингом и обратной связью

Помимо обработки заказов, серверная часть реализует сложные бизнес-правила для работы с клиентами. Контроллер PromosController содержит логику строгой валидации применяемых купонов. При проверке промокода сервер анализирует временные ограничения (ValidFrom, ValidTo), диапазоны разрешенных сумм заказа (MinOrderSum, MaxOrderSum), а также тип поощрения: начисление фиксированной скидки, процентной скидки или автоматическое добавление подарочного товара (GiftProductId) в корзину пользователя.

Контроллер отзывов (ReviewsController) реализует механизм административной премодерации. Любой отзыв, поступающий от клиентского приложения, сохраняется в базу данных со статусом IsApproved = false. Серверный API блокирует передачу таких записей в публичную выдачу до тех пор, пока администратор платформы не изменит статус флага на истинный, что защищает платформу от спама и ненормативной лексики.

3.3 Разработка клиентского веб-приложения

Клиентский уровень программного комплекса «FreeRoll» функционирует в среде веб-браузера конечного пользователя и спроектирован как одностраничное веб-приложение (Single Page Application, SPA). Применение данной архитектурной концепции позволяет минимизировать сетевой трафик за счет исключения перезагрузки страниц с сервера при навигации: интерфейс перерисовывается динамически на стороне клиента, обеспечивая бесшовный пользовательский опыт, сопоставимый с нативными приложениями. Кодовая база

фронтенд-части строго структурирована по методологии Feature-Sliced Design (FSD).

3.3.1 Точка входа и глобальная инициализация приложения

Процесс развертывания и жизненного цикла клиентской части берет свое начало в файле точки входа `./src/main.jsx`. На данном этапе задействуются механизмы библиотеки `react-dom/client` для интеграции виртуального дерева компонентов в реальную структуру DOM (Document Object Model) веб-страницы.

Синтаксическая конструкция использует метод `createRoot`, который выполняет поиск корневого HTML-элемента с идентификатором `root` и инициализирует в нем рендеринг. Для повышения надежности программного продукта вся иерархия компонентов оборачивается в системный мета-компонент `<StrictMode>`. Данный инструмент не компилируется в финальный бандл, однако на этапе разработки выполняет строгие проверки: выявляет небезопасные жизненные циклы, предупреждает об использовании устаревшего API и осуществляет двойной рендеринг для обнаружения скрытых побочных эффектов (side-effects) в функциональных компонентах.

Через данную точку входа в иерархию внедряется главный объединяющий компонент `<App />`, расположенный в директории `src/app/App.jsx`. Его ключевое назначение - агрегация и каскадное внедрение глобальных контекстных провайдеров (Providers), необходимых для функционирования нижележащих архитектурных слоев. Листинг программного модуля инициализации верхнего уровня представлен ниже.

```
import { RouterProvider } from 'react-router-dom';
import { Provider } from 'react-redux';
import { store } from './store/store';
import { router } from './providers/router';
import './styles/global.css';

export const App = () => {
  return (
    <Provider store={store}>
      <RouterProvider router={router} />
    </Provider>
  );
};
```



```
);  
};
```

Внедрение провайдеров происходит в строгой последовательности, определяющей иерархию доступа к данным:

- `<Provider store={store}>` – компонент высшего порядка (Higher-Order Component) из библиотеки `react-redux`. Он внедряет контекст централизованного хранилища состояний `Redux Toolkit` во все дочерние узлы, позволяя любому интерфейсному компоненту системы подписываться на изменения моделей данных;

- `<RouterProvider router={router}>` – управляющий компонент из пакета `react-router-dom`. Он активирует декларативный движок маршрутизации, связывая текущий URL-адрес адресной строки браузера с соответствующим компонентом страницы.

3.3.2 Декларативный роутинг и древовидная структура маршрутов

Управление переходами между экранами платформы вынесено в изолированный программный модуль `src/app/providers/router.jsx`. Для построения системы навигации задействовано современное API маршрутизатора - метод `createBrowserRouter`. Данный подход использует нативный `History API` браузера, что позволяет сохранять корректную историю переходов и обеспечивает поддержку кнопок навигации «Вперед» и «Назад» в браузере пользователя.

Конфигурация роутера описывается в виде древовидного массива объектов. На корневом уровне (`path: '/'`) определяется базовый компонент раскладки интерфейса `<BaseLayout />`. Все остальные маршруты регистрируются как его дочерние элементы (`children`), что позволяет использовать концепцию вложенных маршрутов (Nested Routing).

Система осуществляет сопоставление URL со следующими конечными точками:

- `/` – главная страница каталога товаров (`HomePage`);
- `/cart` – страница оформления и валидации заказа (`CartPage`);
- `/promos` – маркетинговая страница действующих акций (`PromosPage`);

- /profile – личный кабинет пользователя и история покупок (ProfilePage);
- /reviews – публичная лента отзывов клиентов (ReviewsPage);
- /admin/orders – панель управления очередями заказов для персонала (AdminOrdersPage);
- /admin/stats – модуль аналитики и графиков для руководства (AdminStatsPage).

Для визуализации архитектуры переходов и иерархического распределения прав доступа (клиентская часть и защищенная панель администратора HoReCa) в рамках проектной документации спроектирована граф-схема. Графическая структура декларативной карты маршрутов представлена на рисунке 35.

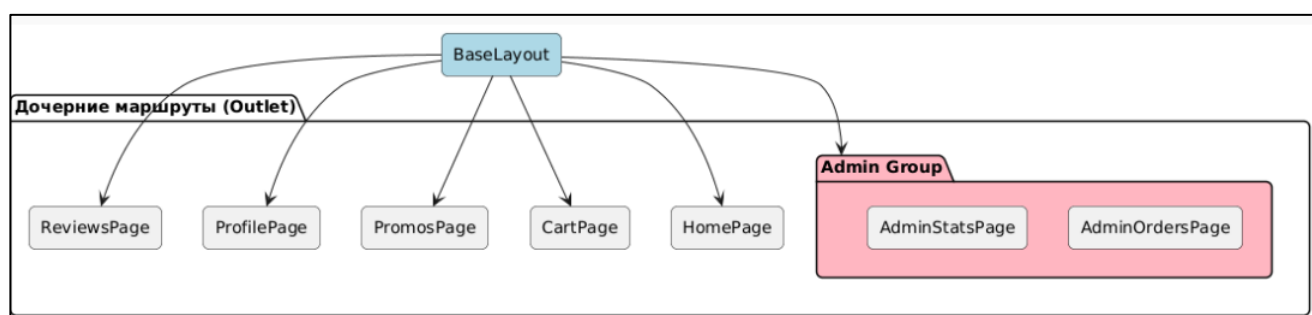


Рисунок 35 – Структурная схема декларативной маршрутизации веб-платформы

3.3.3 Глобальная компоновка и сквозные элементы раскладки

Компонент `<BaseLayout />` (src/app/layouts/BaseLayout.jsx) выступает в роли единого визуального каркаса приложения. Вместо полной замены интерфейса при переходе по ссылкам, BaseLayout сохраняет неизменными глобальные элементы, перерисовывая только внутреннее содержимое.

Для динамической подстановки компонентов страниц задействован специализированный слот `<Outlet />` из пакета react-router-dom. Когда пользователь переходит по адресу /cart, роутер определяет, что дочерним элементом является CartPage, и монтирует его ровно на место компонента Outlet.

Помимо центральной контентной зоны, в структуру каркаса интегрированы сквозные модули и утилиты автоматизации:

– `<ScrollToTop />` - системный компонент, отслеживающий смену URL-адреса и мгновенно сбрасывающий координату вертикальной прокрутки окна в нулевое положение (`window.scrollTo(0, 0)`), предотвращая сохранение позиции скrolла предыдущего экрана;

– `<Header />` и `<Footer />` - высокоуровневые виджеты шапки и подвала сайта, содержащие элементы сквозной навигации и личные данные авторизованного профиля;

– `<ScrollToTopButton />` - плавающая кнопка плавного возврата к началу страницы, активирующаяся при превышении порога прокрутки в 300 пикселей;

– `<AdminFloatingButton />` и `<AdminModalManager />` - инжектируемые инструменты глобального контекста администратора, обеспечивающие вызов модальных окон создания номенклатурных единиц из любой точки интерфейса.

Код компонента `BaseLayout.jsx` представлен ниже.

```
import { Outlet } from 'react-router-dom';

import { Header } from '@widgets/header';
import { Footer } from '@widgets/footer';
import { AdminFloatingButton, AdminModalManager } from
'@/features/admin-tools';

import { ScrollToTopButton } from '@shared/ui/ScrollToTopButton';
import { ScrollToTop } from '@shared/lib/ScrollToTop';

export const BaseLayout = () => {
  return (
    <div className="min-h-screen flex flex-col">
      {/* Утилита скролла при переходе по страницам */}
      <ScrollToTop />

      <Header />

      <main className="flex-1 container mx-auto px-4 py-8">
        <Outlet />
      </main>

      <Footer />

      <ScrollToTopButton />

      {/* Глобальные инструменты админа */}
      <AdminFloatingButton />
    </div>
  );
}
```

```

        <AdminModalManager />
    </div>
  );
};

```

3.3.4 Сетевой уровень и клиент взаимодействия с API

Взаимодействие клиентского веб-приложения с серверной частью (Backend) осуществляется по протоколу HTTP. Вместо прямого вызова встроенной функции браузера `fetch` в каждом компоненте, архитектура проекта подразумевает использование инкапсулированного сетевого модуля `ApiClient.js`, расположенного в слое общих ресурсов (`src/shared/api`).

Создание единого класса-обертки (модуля) для сетевых запросов решает сразу несколько архитектурных задач:

- Соблюдение принципа DRY (Don't Repeat Yourself): Избавление от дублирования кода настройки заголовков (Headers), парсинга JSON и обработки сетевых ошибок.
- Безопасность и конфигурация окружения: Базовый URL сервера (`BASE_URL`) не "захардкожен" в коде, а динамически подтягивается из переменных окружения Vite (`import.meta.env.VITE_API_URL`). Это позволяет безопасно разделять адреса для локальной разработки и рабочей среды.
- Централизованная обработка статусов: В случае возврата сервером кодов ошибок, клиент автоматически генерирует исключение (Exception) с читаемым текстом, извлеченным из тела ответа.

Модуль реализует полный набор методов REST API: `get`, `post`, `put`, `patch`, `delete`, а также специализированный метод `upload` для передачи бинарных данных (изображений) с использованием объекта `FormData`. Учтена специфика обработки HTTP-статуса 204 No Content (часто возвращаемого при удалении или обновлении), чтобы предотвратить падение клиента при попытке распарсить пустое тело ответа.

Фрагмент кода, демонстрирующий реализацию методов в сетевом клиенте, представлен ниже.


```

const BASE_URL = import.meta.env.VITE_API_URL;

export const apiClient = {
  async get(endpoint) {
    const res = await fetch(`${BASE_URL}${endpoint}`);
    if (!res.ok) throw new Error(`Ошибка GET ${endpoint}`);
    return res.json();
  },

  async post(endpoint, body) {
    const res = await fetch(`${BASE_URL}${endpoint}`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(body)
    });
    if (!res.ok) {
      const err = await res.json().catch(() => ({}));
      throw new Error(err.message || `Ошибка POST
${endpoint}`);
    }
    return res.json();
  },

  async upload(endpoint, formData) {
    const res = await fetch(`${BASE_URL}${endpoint}`, {
      method: 'POST',
      body: formData
    });
    if (!res.ok) throw new Error(`Ошибка UPLOAD ${endpoint}`);
    return res.json();
  },

  async put(endpoint, body) {
    const res = await fetch(`${BASE_URL}${endpoint}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(body)
    });
    if (!res.ok) {
      const err = await res.json().catch(() => ({}));
      throw new Error(err.message || `Ошибка PUT
${endpoint}`);
    }

    const text = await res.text();
    return text ? JSON.parse(text) : {};
  },

  async patch(endpoint, body) {
    const res = await fetch(`${BASE_URL}${endpoint}`, {
      method: 'PATCH',

```

					ПГУ 1.090301.44.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		55

```

        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(body)
    });
    if (!res.ok) throw new Error(`Ошибка PATCH ${endpoint}`);
    return res.json();
},

async delete(endpoint) {
    const res = await fetch(`${BASE_URL}${endpoint}`, {
        method: 'DELETE'
    });
    if (!res.ok) throw new Error(`Ошибка DELETE ${endpoint}`);

    const text = await res.text();
    return text ? JSON.parse(text) : {};
}
};

```

3.3.5 Централизованное управление состоянием

Для управления глобальным состоянием веб-платформы, к которому требуется доступ из множества независимых компонентов (например, подсчет товаров в шапке сайта, оформление заказа в корзине, проверка прав администратора), была выбрана библиотека Redux Toolkit (RTK) [8]. Использование RTK избавляет от необходимости пробрасывать свойства через множество уровней дерева компонентов (Prop Drilling) и предоставляет строгую однонаправленную архитектуру потока данных (Flux).

Инициализация хранилища (Store) производится в модуле `src/app/store/store.js`. Функция `configureStore` автоматически настраивает системные инструменты разработчика (Redux DevTools) и подключает промежуточное программное обеспечение (Middleware) для работы с асинхронными действиями (Thunks).

Глобальное состояние логически декомпозировано на три независимых домена (слайса/slices):

- `cart` - состояние корзины покупок (содержит добавленные товары и итоговую сумму);
- `products` - кэшированный товарный каталог меню;

– user - состояние текущей сессии (информация об авторизованном пользователе и его адресной книге).

```
import { configureStore } from '@reduxjs/toolkit';
import cartReducer from '@entities/cart/model/cartSlice';
import productReducer from '@entities/product/model/productSlice';
import userReducer from '@entities/user/model/userSlice.js'

export const store = configureStore({
  reducer: {
    cart: cartReducer, // Подключаем нашу корзину
    products: productReducer, // <-- Добавляем в глобальное хранилище
    user: userReducer,
  },
});
```

Визуализация дерева глобального хранилища в инструментах разработчика представлена на рисунке 36.

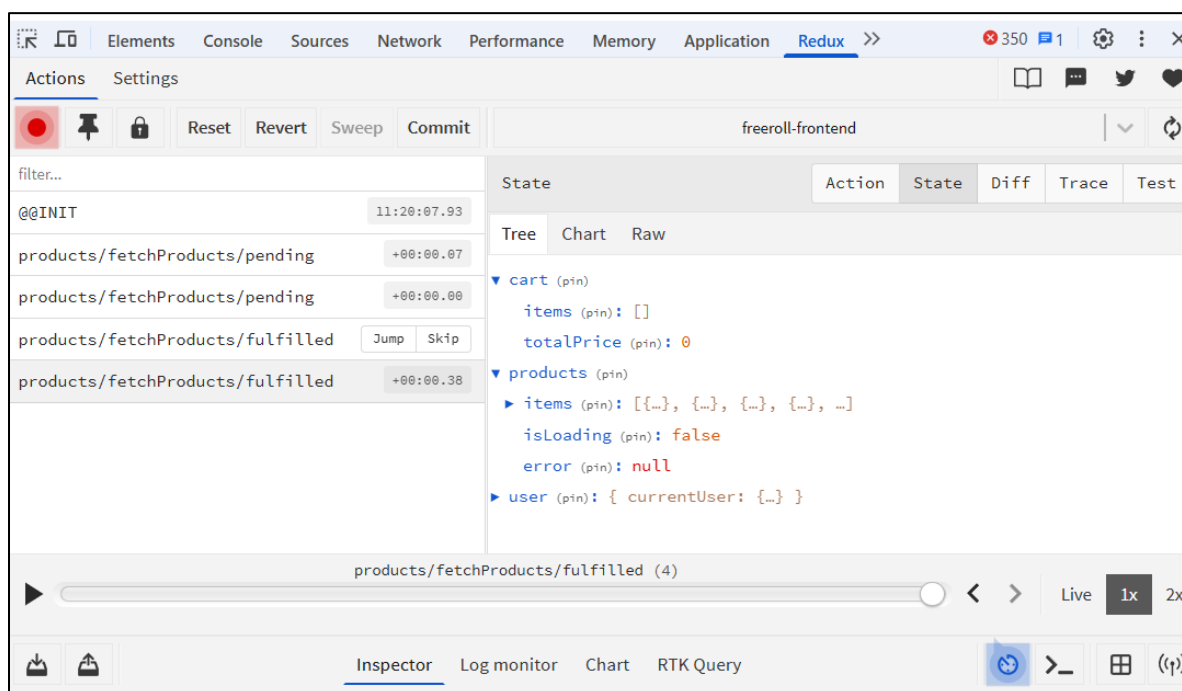


Рисунок 36 – Визуализация дерева глобального хранилища в инструментах разработчика

3.3.6 Механизм асинхронных действий и побочных эффектов

Связующим звеном между сетевым клиентом (`apiClient`) и синхронным хранилищем Redux выступает механизм асинхронных действий `createAsyncThunk`. Данная функция-утилита генерирует жизненный цикл запроса, автоматически отправляя (`dispatch`) в хранилище экшны в зависимости от состояния сети: `pending` (ожидание), `fulfilled` (успешное выполнение) или `rejected` (ошибка).

Ярким примером интеграции сетевого уровня и состояния является модель каталога блюд `src/entities/product/model/productSlice.js`. Метод `fetchProducts` инициирует GET-запрос к серверу. Пока данные загружаются, редьюсер устанавливает флаг `isLoading = true`, что позволяет интерфейсу отображать индикаторы загрузки (спиннеры). При успешном ответе данные помещаются в массив `items`, а флаг загрузки снимается.

Такой подход инкапсулирует всю сложную логику работы с сетью внутри слоя Entities, оставляя визуальные React-компоненты максимально чистыми - они только вызывают экшны и реактивно подписываются на изменения через хук `useSelector`.

3.3.7 Управление состоянием корзины и персистентность данных

Одним из ключевых модулей слоя сущностей является подсистема управления корзиной покупок, реализованная в файле `src/entities/cart/model/cartSlice.js`. Поскольку корзина должна быть доступна на любой странице приложения (для отображения счетчика в шапке сайта и расчета итоговой суммы), ее состояние хранится в глобальном хранилище Redux.

Критически важным требованием к современным интернет-магазинам является сохранность выбранных товаров при случайном обновлении страницы или закрытии вкладки браузера. Для реализации персистентности (сохранения) данных алгоритм интегрирован с локальным хранилищем браузера - `localStorage`. При инициализации приложения функция `loadCartFromStorage` извлекает сохраненный JSON-объект, а при любом изменении состава корзины функция

saveCartToStorage синхронизирует актуальное состояние Redux с памятью браузера.

Редьюсер cartSlice инкапсулирует в себе строгие математические алгоритмы изменения состояния:

- addToCart - проверяет наличие товара в массиве. Если товар уже существует, инкрементирует его счетчик (count), иначе добавляет новый объект. Затем пересчитывает общую сумму totalPrice;

- decrementCount - уменьшает количество товара. Если счетчик достигает единицы, функция удаляет товар из массива корзины;

- removeItem и clearCart - обеспечивают полное удаление позиций и обнуление счетчиков.

Фрагмент кода, демонстрирующий логику добавления товара с синхронизацией в локальное хранилище, представлен ниже.

```
const cartSlice = createSlice({
  name: 'cart',
  initialState: loadCartFromStorage(),
  reducers: {
    addToCart: (state, action) => {
      const product = action.payload;
      const existingItem = state.items.find(item => item.id
=== product.id);

      if (existingItem) {
        existingItem.count += 1;
      } else {
        state.items.push({ ...product, count: 1 });
      }
      state.totalPrice += product.price;
      saveCartToStorage(state);
    },
```

3.3.8 Конфигурация жизненного цикла заказа

Для обеспечения единообразного отображения статусов заказа по всему приложению (как в личном кабинете клиента, так и в панели администратора), бизнес-логика воронки статусов вынесена в отдельный конфигурационный модуль src/entities/order/model/orderPipeline.js.

Данный модуль реализует шаблон проектирования «Конечный автомат» (State Machine) на минимальном уровне. Объект ORDER_PIPELINE выполняет роль словаря, жестко связывающего технические статусы сервера (New, Cooking, Delivering, Completed, Cancelled) с их визуальным представлением: цветовыми константами TailwindCSS [10], читаемыми текстовыми метками, иконками и, что наиболее важно, указателем на следующий логический шаг в воронке (next).

Использование централизованной конфигурации исключает появление «магических строк» (magic strings) в компонентах интерфейса и позволяет мгновенно изменять цвета или названия этапов доставки без необходимости рефакторинга десятков файлов. Код модуля orderPipeline.js представлен ниже.

```
export const ORDER_PIPELINE = {
  'New': {
    label: 'Оформлен',
    next: 'Cooking',
    btnText: 'Взять в готовку',
    color: 'bg-[#0099ff] hover:bg-[#007acc]',
    badge: 'bg-blue-100 text-blue-600',
    borderColor: 'border-blue-200'
  },
  'Cooking': {
    label: 'На кухне',
    next: 'Delivering',
    btnText: 'Передать курьеру',
    color: 'bg-amber-500 hover:bg-amber-600',
    badge: 'bg-amber-100 text-amber-700',
    borderColor: 'border-amber-200'
  },
  'Delivering': {
    label: 'В пути',
    next: 'Completed',
    btnText: 'Заказ доставлен',
    color: 'bg-purple-500 hover:bg-purple-600',
    badge: 'bg-purple-100 text-purple-700',
    borderColor: 'border-purple-200'
  },
  'Completed': {
    label: 'Доставлен',
    next: null,
    btnText: '',
    color: 'bg-green-500',
    badge: 'bg-green-100 text-green-600',
    borderColor: 'border-slate-100'
  },
  'Cancelled': {
```



```

    label: 'Отменен',
    next: null,
    btnText: '',
    color: 'bg-red-500',
    badge: 'bg-red-100 text-red-600',
    borderColor: 'border-slate-100'
  }
};

export const getOrderStage = (status) => ORDER_PIPELINE[status] ||
ORDER_PIPELINE['New'];
export const isOrderDone = (status) => status === 'Completed' ||
status === 'Cancelled';

```

Визуализация жизненного цикла заказа в интерфейсе клиента представлена на рисунке 37.

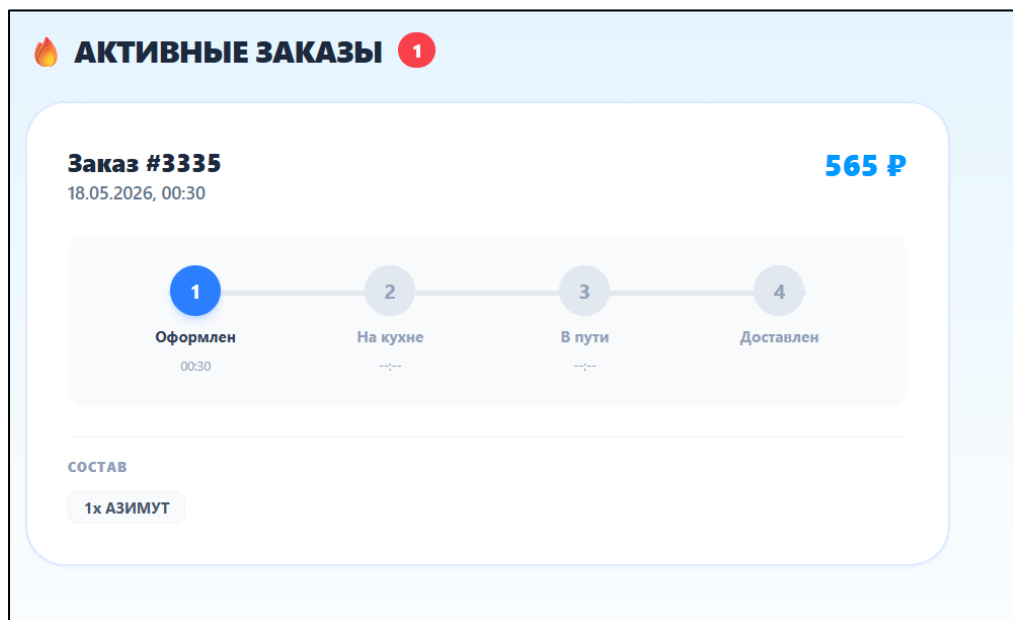


Рисунок 37 – Визуализация жизненного цикла заказа в интерфейсе клиента

3.3.9 Базовые визуальные компоненты предметной области

Слой Entities содержит компоненты представления, которые занимаются исключительно рендерингом переданных в них данных (Props). Основными компонентами данного слоя являются ProductCard.jsx (карточка товара), ClientOrderCard.jsx (карточка заказа для истории) и PromoCard.jsx (карточка акции).

Архитектурной особенностью этих компонентов является широкое использование паттерна «Слоты» (React Render Props / Children). Например, компонент ProductCard не содержит в себе логики удаления товара или открытия модальных окон администратора, потому что это нарушило бы принцип единственной ответственности (Single Responsibility Principle). Вместо этого карточка принимает пропс adminControls, куда вышестоящие компоненты могут "прокинуть" необходимые кнопки управления.

Такой подход обеспечивает слабую связность кода (Low Coupling): карточка товара остается универсальной и может переиспользоваться на главной странице, в результатах поиска или в блоке рекомендаций корзины без изменений в собственном исходном коде. Визуальный интерфейс сущности товарного каталога представлен на рисунке 38.

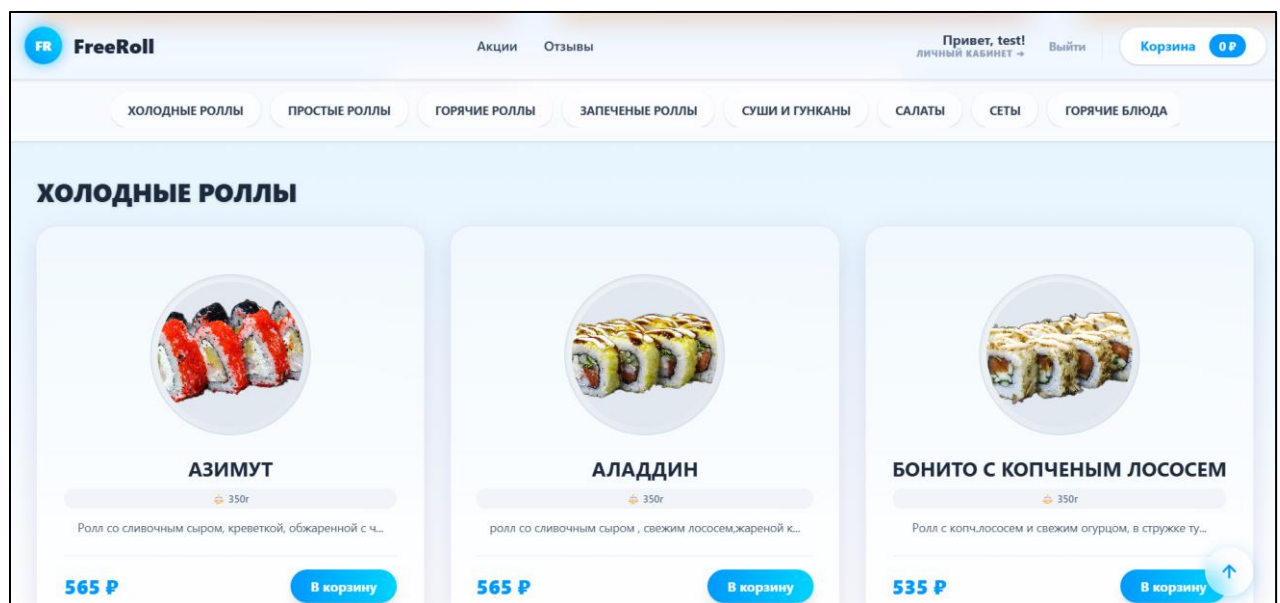


Рисунок 38 – Визуальный интерфейс сущности товарного каталога

3.3.10 Интерактивные инструменты обратной связи и модерации контента

Платформа предоставляет пользователям возможность оставлять отзывы о качестве обслуживания. Эта функциональность вынесена в модуль src/features/create-review. Компонент ReviewForm.jsx реализует сложную

составную форму, включающую текстовое поле, интерактивный рейтинг в виде звезд и загрузчик графических файлов. Для обеспечения уникальности имен сохраняемых изображений на сервере клиентская часть автоматически генерирует название файла на основе имени пользователя и текущей временной метки (Date.now()).

Связанным функциональным блоком выступает инструмент администратора ReviewModerationPanel.jsx. В связи с бизнес-требованием о премодерации отзывов (наличие флага IsApproved = false в базе данных), данный модуль отображает скрытую очередь публикаций. Администратор имеет возможность одним кликом отправить PATCH-запрос для одобрения отзыва или DELETE-запрос для его удаления, что немедленно обновляет состояние клиентского интерфейса без перезагрузки страницы.

3.3.11 Бизнес-логика корзины: применение промо-акций и управление доставкой

Процесс оформления заказа дополнен микро-функциями, повышающими гибкость платформы:

- ApplyPromo.jsx: модуль, отвечающий за прием пользовательских промокодов. Он захватывает текстовый ввод и передает его в вышестоящий контроллер для валидации через API, отображая сообщения об ошибке или успешной активации скидки;

- DeliveryTimeSelector.jsx: реализует логику переключения между немедленной доставкой («Как можно быстрее») и отложенным заказом («Ко времени»). При выборе второго варианта динамически активируется выпадающий список (select) с доступными часовыми интервалами;

- AddressManager.jsx: модуль управления адресной книгой. Позволяет пользователю добавлять новые локации в свою учетную запись или переключаться между существующими адресами прямо на этапе оформления корзины.

3.3.12 Событийно-ориентированная архитектура административных модальных окон

Для управления номенклатурой меню, категориями и акциями в системе реализован богатый административный инструментарий (src/features/admin-tools). Архитектурной проблемой сложных веб-приложений является вызов модальных окон из глубоко вложенных компонентов интерфейса. Традиционный подход с передачей callback-функций (Prop Drilling) приводит к сильной связанности кода.

В рамках данного проекта была успешно применена событийно-ориентированная архитектура (Event-Driven Architecture) с использованием нативного браузерного API CustomEvent.

Глобальный компонент-оркестратор AdminModalManager.jsx, размещенный на верхнем уровне приложения (BaseLayout), подписывается на системное событие admin-modal-open. Любая кнопка редактирования в любой точке интерфейса (например, внутри карточки товара) может отправить это событие, прикрепив к нему полезную нагрузку (detail: { type: 'product', data: product }). Менеджер перехватывает событие и динамически рендерит нужное модальное окно с предзаполненными данными.

Фрагмент кода, демонстрирующий логику оркестрации модальных окон через пользовательские события, представлен ниже.

```
import { useState, useEffect } from 'react';
import { useDispatch } from 'react-redux';
import { fetchProducts } from
'@/entities/product/model/productSlice';

import { AdminProductModal } from './AdminProductModal';
import { AdminRulesModal } from './AdminRulesModal';
import { AdminPromoModal } from './AdminPromoModal';
import { AdminCategoryModal } from './AdminCategoryModal';

export const AdminModalManager = () => {
  const dispatch = useDispatch();
  const [config, setConfig] = useState({
    isOpen: false,
    type: null,
    data: null
  });
};
```

```

useEffect(() => {
  const handleOpen = (e) => {
    setConfig({
      isOpen: true,
      type: e.detail.type,
      data: e.detail.data || null
    });
  };

  window.addEventListener('admin-modal-open', handleOpen);
  return () => window.removeEventListener('admin-modal-open',
handleOpen);
}, []);

const close = () => setConfig({ ...config, isOpen: false });

const handleSuccess = () => {
  close();

  if (config.type === 'product') {
    dispatch(fetchProducts());
  } else {
    window.dispatchEvent(new CustomEvent('admin-data-
updated', { detail: { type: config.type } }));
    window.location.reload();
  }
};

if (!config.isOpen) return null;

return (
  <>
    {config.type === 'product' && (
      <AdminProductModal
        isOpen={config.isOpen}
        onClose={close}
        editProduct={config.data}
        onSuccess={handleSuccess}
      />
    )}

    {config.type === 'promo' && (
      <AdminPromoModal
        isOpen={config.isOpen}
        onClose={close}
        editPromo={config.data}
        onSuccess={handleSuccess}
      />
    )}

    {config.type === 'rules' && (

```

```

        <AdminRulesModal
            isOpen={config.isOpen}
            onClose={close}
            currentRules={config.data}
            onSuccess={handleSuccess}
        />
    ) }

    {config.type === 'category' && (
        <AdminCategoryModal
            isOpen={config.isOpen}
            onClose={close}
            editCategory={config.data}
            onSuccess={handleSuccess}
        />
    ) }
</>
);
};

```

Такой паттерн проектирования делает компоненты карточек товаров независимыми от бизнес-логики редактирования базы данных, что полностью соответствует принципам SOLID. Административная панель управления пользовательским контентом представлена на рисунке 39.

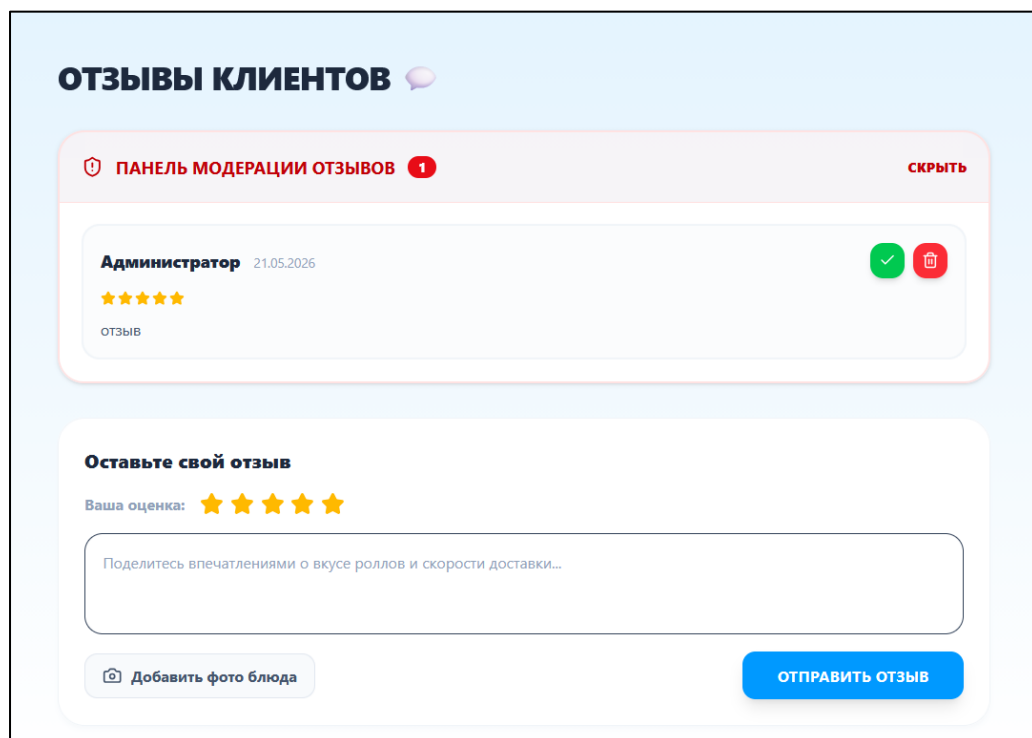


Рисунок 39 – Административная панель управления пользовательским контентом

3.3.13 Композиция клиентской витрины и жестовая навигация

Главная страница приложения (HomePage.jsx) является ключевым экраном взаимодействия с клиентом. На этапе инициализации страницы пользовательский хук useHomeData осуществляет параллельную загрузку дерева категорий, каталога товаров и списка активных рекламных баннеров из хранилища Redux и API-клиента.

Интерфейс страницы визуально разделен на несколько независимых виджетов:

- CategoryNav: липкая (sticky) панель навигации по товарным категориям;
- PromoCarousel: интерактивная карусель рекламных акций;
- ProductList: динамически генерируемые сетки карточек товаров, отфильтрованные по принадлежности к конкретной категории и строке пользовательского поиска.

Композиция главной страницы клиентского приложения представлена на рисунке 40.

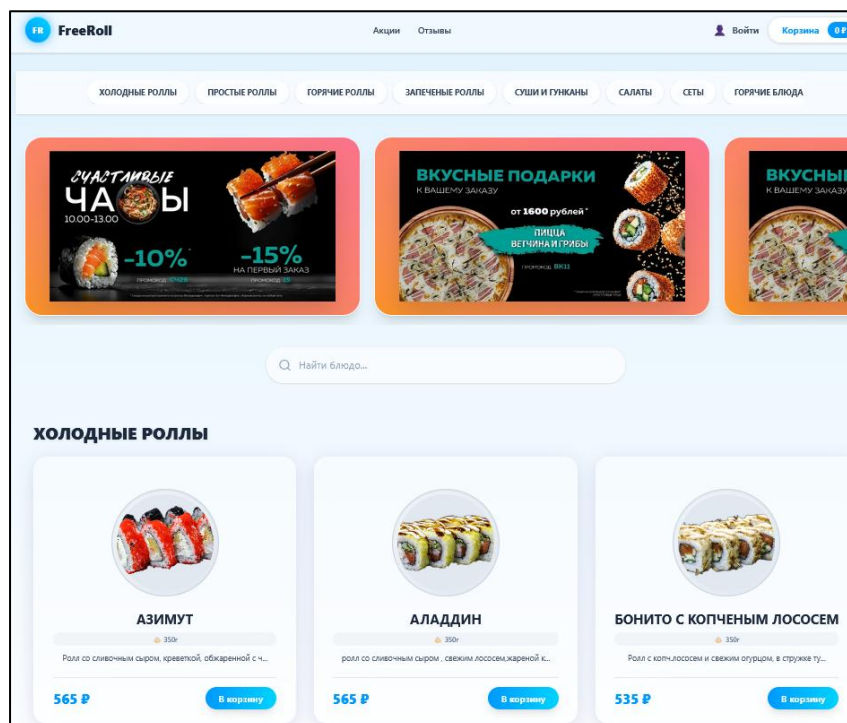


Рисунок 40 – Композиция главной страницы клиентского приложения

3.3.14 Рабочее место оператора: маршрутизация и приоритизация заказов

Для нужд персонала службы доставки спроектирован защищенный экран `AdminOrdersPage.jsx`. Интерфейс страницы организован по принципу Канбан-доски (Kanban), адаптированной под вертикальную прокрутку на мобильных устройствах.

Вычислительное ядро страницы инкапсулировано в хуке `useAdminOrders.js`. При получении массива заказов от сервера алгоритм использует хук мемоизации `useМемо` для предотвращения ресурсоемких перерасчетов при каждом рендеринге интерфейса. Внутри хука происходит распределение единого массива данных на три логические очереди:

- `asapOrders` - живая очередь текущих заказов («Как можно быстрее»);
- `scheduledOrders` - отложенные заказы («Ко времени»), требующие отдельного контроля;
- `completedOrders` - архивные заказы (выполненные или отмененные).

Для корректной отрисовки живой очереди применяется взвешенная сортировка: сначала заказы выстраиваются в соответствии с приоритетом их текущего статуса (новые заказы находятся выше), а при совпадении статусов сортируются по временной метке оформления. Фрагмент кода, реализующий алгоритм взвешенной сортировки и распределения очередей, представлен ниже.

```
const changeOrderStatus = async (orderId, newStatus) => {
  try {
    await apiClient.patch(`/orders/${orderId}/status`,
      newStatus);

    setOrders(prev => prev.map(o => {
      if (o.id === orderId) {
        const updatedOrder = { ...o, status: newStatus };
        if (newStatus === 'Cooking') updatedOrder.cookingAt
          = new Date().toISOString();
        if (newStatus === 'Delivering')
          updatedOrder.deliveringAt = new Date().toISOString();
        if (newStatus === 'Completed')
          updatedOrder.completedAt = new Date().toISOString();
        return updatedOrder;
      }
      return o;
    }
  )
}
```

```

    ));
    window.dispatchEvent(new CustomEvent('order-status-
updated'));
    } catch (err) {
        alert("Не удалось обновить статус");
    }
};

const { asapOrders, scheduledOrders, completedOrders } = useMemo(()
=> {
    const statusWeight = { 'New': 1, 'Cooking': 2, 'Delivering': 3,
'Completed': 4, 'Cancelled': 5 };
    const sortPipeline = (a, b) => {
        if (statusWeight[a.status] !== statusWeight[b.status])
return statusWeight[a.status] - statusWeight[b.status];
        return new Date(a.createdAt) - new Date(b.createdAt);
    };

    const isScheduled = (notice) => notice && notice.includes('Ко
времени');

    return {
        asapOrders: orders.filter(o => o.status !== 'Completed' &&
o.status !== 'Cancelled' &&
!isScheduled(o.deliveryNotice)).sort(sortPipeline),
        scheduledOrders: orders.filter(o => o.status !== 'Completed'
&& o.status !== 'Cancelled' &&
isScheduled(o.deliveryNotice)).sort(sortPipeline),
        completedOrders: orders.filter(o => o.status === 'Completed'
|| o.status === 'Cancelled').sort((a, b) => new Date(b.createdAt) -
new Date(a.createdAt))
    };
}, [orders]);

```

Визуализация очередей реализована через переиспользуемый виджет OrdersAccordion.jsx (шторка/аккордеон). Каждый аккордеон поддерживает динамическое раскрытие списка, автоматический подсчет элементов внутри секции и применение цветового кодирования (Theme) в зависимости от типа содержимого. Панель управления производственными очередями представлена на рисунке 41.

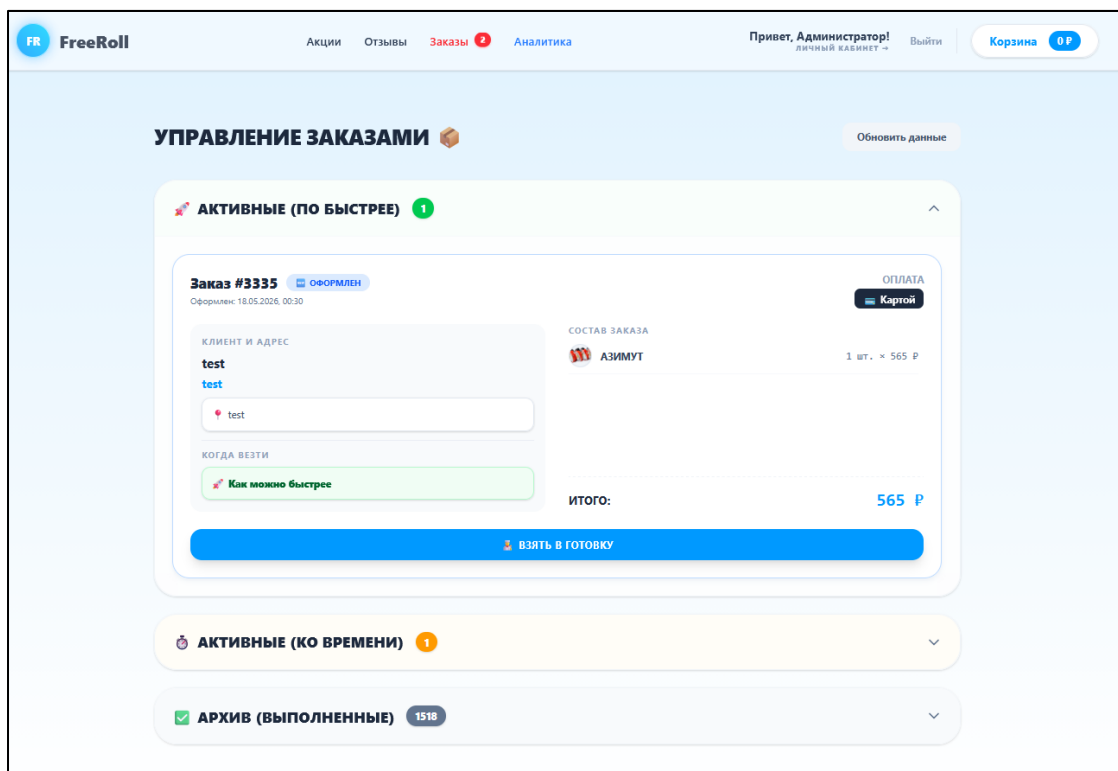


Рисунок 41 – Панель управления производственными очередями

3.3.15 Аналитический дашборд и визуализация бизнес-метрик

Завершающим модулем административного интерфейса является страница AdminStatsPage.jsx. Ее основное назначение предоставление руководителю предприятия наглядной сводки об эффективности работы (KPI).

Страница интегрирована с виджетом календарного фильтра StatsFilter.jsx, который позволяет динамически изменять период анализа (неделя, месяц, год) с последующей отправкой запроса к аналитическому эндпоинту серверной части.

Для визуализации статистических данных (динамика выручки, пиковые часы нагрузки, популярность категорий меню) используется сторонний графический пакет recharts. Данная библиотека базируется на компонентном подходе React и формирует интерактивные диаграммы в векторном формате SVG, обеспечивающем идеальную четкость отрисовки на экранах с любым разрешением.

Внутри виджета DashboardCharts.jsx применяются компоненты AreaChart (график с заливкой области) и BarChart (гистограмма). Для обеспечения стилистического единства системы стандартные всплывающие подсказки графиков переопределены пользовательским компонентом GlassTooltip.jsx, который реализует эффект матового стекла и фирменное форматирование числовых значений. Аналитический дашборд руководителя платформы представлен на рисунке 42.



Рисунок 42 – Аналитический дашборд руководителя платформы

3.4 Тестирование системы

В начале процесса тестирования необходимо сформировать структурированный план проверок. Это позволит последовательно оценить корректность работы всех узлов клиент-серверной архитектуры веб-платформы. В рамках испытаний выделены четыре основных этапа. На первом этапе проверяется надежность системы валидации пользовательского ввода на стороне клиентского интерфейса. На втором этапе тестируется способность клиентского приложения устанавливать связь с серверным API и обрабатывать HTTP-ответы. Третий этап включает проверку алгоритмов бизнес-логики: формирование корзины, применение промокодов и управление жизненным циклом заказа. Четвертый этап направлен на подтверждение целостности сохраненной информации в локальной реляционной базе данных. План тестирования представлен в таблице 1.

Таблица 1 – План тестирования

Описание проверки	Ожидаемый результат
Валидация пользовательского ввода	Блокировка некорректных данных, вывод сообщений об ошибке
Клиент-серверное взаимодействие	Успешная маршрутизация запросов и получение корректных JSON-ответов
Функциональное тестирование бизнес-логики	Успешное оформление заказа и смена статусов в панели администратора
Проверка подсистемы хранения данных	Корректная запись всех деталей заказа и профилей в таблицы БД

На начальной стадии запуска клиентского веб-приложения проверяется устойчивость программы к некорректным действиям пользователя согласно первому пункту плана. Алгоритм ввода параметров (например, при регистрации или оформлении заказа) оснащен программными фильтрами. Приложение блокирует отправку данных при несовпадении паролей, незаполненных обязательных полях или попытке ввести недопустимые значения. Реакция приложения на некорректный ввод с выводом предупреждающего сообщения представлена на рисунке 43.

The screenshot shows a registration form with the following elements:

- Buttons: "Вход" (Login) and "Регистрация" (Registration).
- Error message: "Пароли не совпадают!" (Passwords do not match!).
- Input fields:
 - Username: "Павел"
 - Phone number: "8800553535"
 - Address: "ул. Пушкина д. 10"
 - Birth date: "2004-05-07"
 - First name: "pavel"
 - Password: masked with dots
 - Repeat password: masked with dots
- Submit button: "СОЗДАТЬ АККАУНТ" (Create Account).

Рисунок 43 – Проверка валидации ввода и обработки ошибок

Второй этап тестирования посвящен проверке сетевого взаимодействия между React-клиентом и ASP.NET-сервером. При инициализации запроса (например, получение каталога товаров) система должна успешно установить HTTP-соединение, а сервер - обработать запрос и вернуть статус 200 ОК с полезной нагрузкой. Подтверждение успешного обмена пакетами данных представлено на рисунке 44.

The screenshot shows the Network tab in Chrome DevTools with the following data:

Статус	Метод	Домен	Файл	Инициатор	Тип	Передано	Размер	0 мс
200	GET	localhost:7031	active	apiClient.js:4 (fetch)	json	656 кБ	6,37 кБ	
200	GET	localhost:7031	products	apiClient.js:4 (fetch)	json	112,22 кБ	112,03 кБ	
200	GET	localhost:7031	categories	apiClient.js:4 (fetch)	json	670 Б	482 Б	
200	GET	localhost:7031	active	apiClient.js:4 (fetch)	json	656 кБ	6,37 кБ	
200	GET	localhost:7031	d5b80068-3263-4027-9b8b-a3d6c468a63c.jpg	img	jpeg	кешировано	137,06 кБ	
200	GET	localhost:7031	135f8d01-32fa-49fa-a841-a0d7235039de.jpg	img	jpeg	кешировано	184,73 кБ	
200	GET	localhost:7031	kukuku.jpg	img	jpeg	кешировано	184,73 кБ	

Рисунок 44 – Подтверждение установки связи и получения данных от API

На третьем этапе проверяется функциональность ключевых бизнес-процессов платформы. Производится симуляция действий клиента: сборка товаров в корзину, применение скидочного промокода и отправка заказа. Затем проверяется реакция административной панели - появление нового заказа в живой очереди и успешное перемещение карточки заказа по воронке статусов (из состояния «Оформлен» в «На кухне»). Результат выполнения комплексного сценария представлен на рисунке 45.

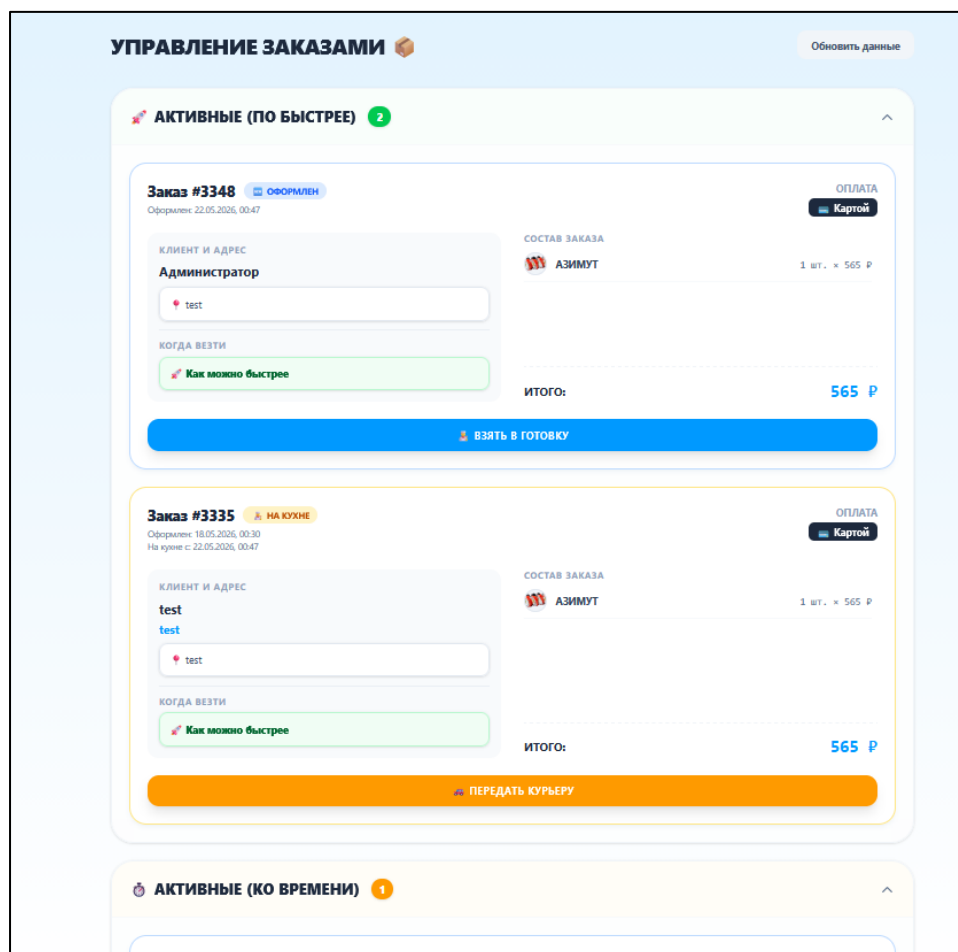


Рисунок 45 – Выполнение бизнес-сценария оформления и маршрутизации заказа

Завершающим этапом тестирования является проверка работы подсистемы долговременного хранения. После успешного выполнения операций на сервере Entity Framework Core генерирует и обновляет физический файл базы данных.

Наличие созданного файла БД freeroll.db в корневой директории серверного проекта представлено на рисунке 46.

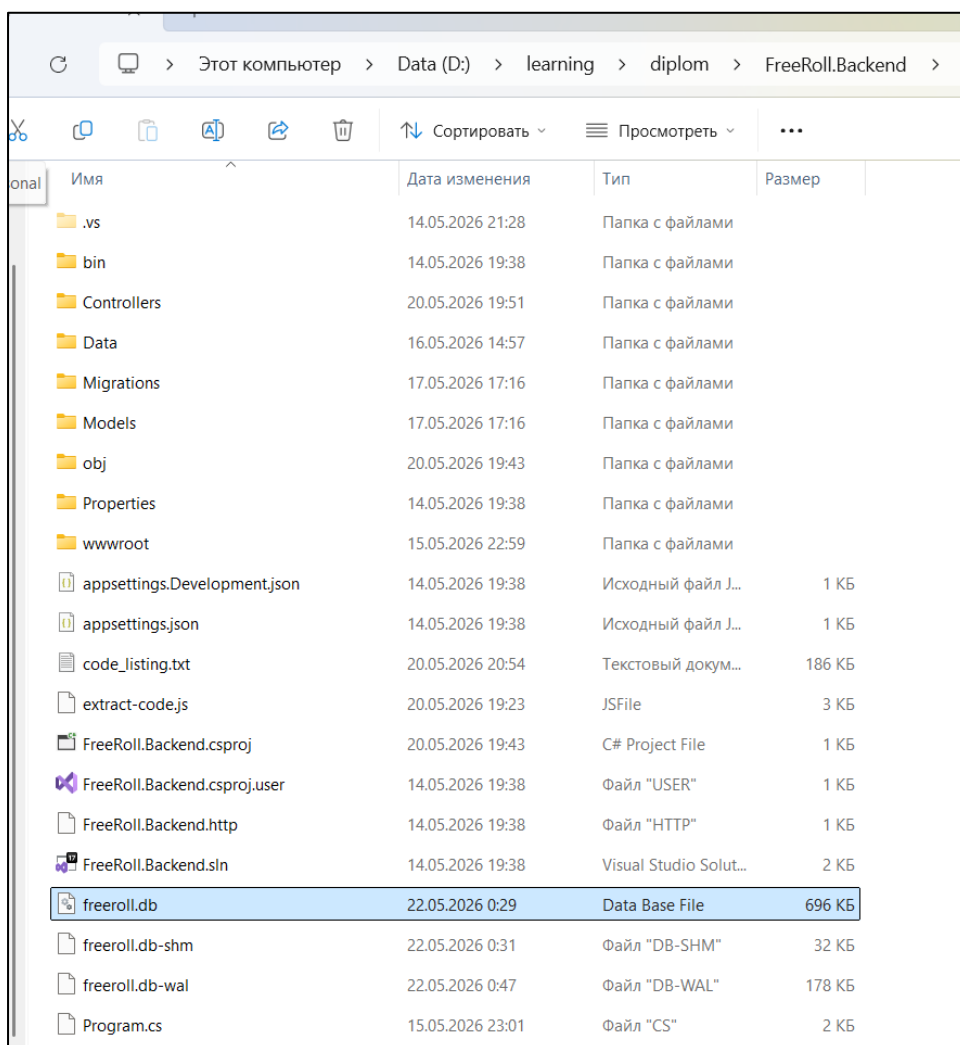


Рисунок 46 – Директория проекта с файлом БД

Для проверки целостности сохраненной информации применяется специализированное ПО - DB Browser for SQLite. Анализ БД подтверждает строгое соответствие типов данных и записанных значений (состав чека, итоговая цена, временные метки статусов) результатам, отображаемым в клиентском интерфейсе. Содержимое сформированной таблицы заказов с результатами работы системы представлено на рисунке 47.

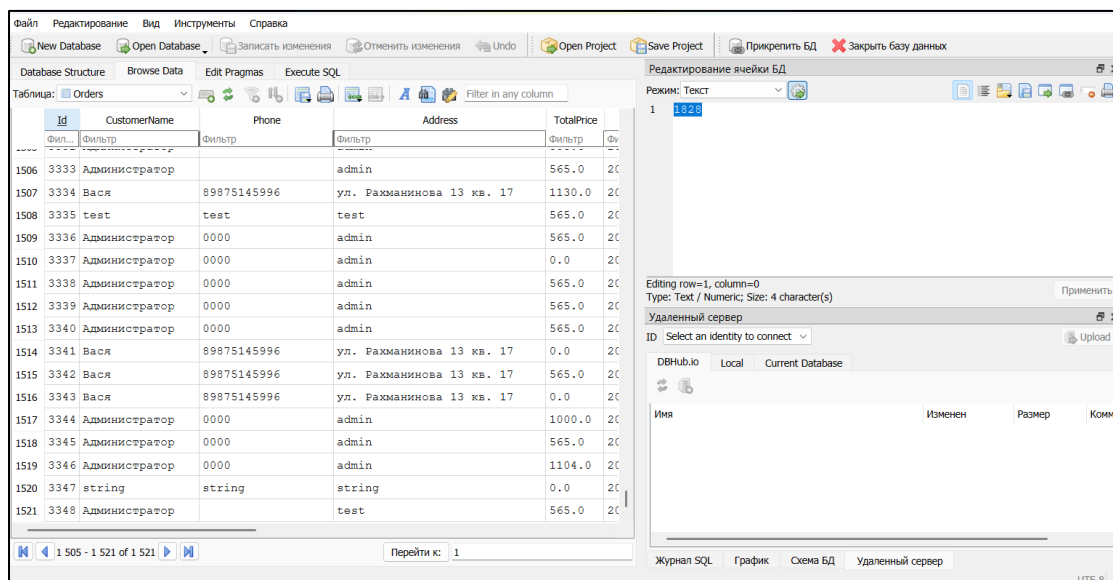


Рисунок 47 – Содержимое БД

Анализ полученных данных подтверждает полную работоспособность цепочки передачи информации и корректность алгоритмов маршрутизации. Приложение успешно выполняет обработку клиентских запросов, вычисляет финансовые показатели и безопасно распределяет данные. Результаты проведенных испытаний представлены в таблице 2.

Таблица 2 – Результаты тестирования

Описание проверки	Ожидаемый результат	Фактический результат	Статус
Валидация пользовательского ввода	Блокировка некорректных данных, вывод ошибок	Некорректный ввод успешно заблокирован интерфейсом	Успешно
Клиент-серверное взаимодействие	Успешная маршрутизация запросов	Соединение установлено, данные получены в формате JSON	Успешно
Функциональное тестирование бизнес-логики	Оформление заказа и смена статусов	Заказ оформлен, статус успешно изменен в панели администратора	Успешно
Проверка подсистемы хранения данных	Корректная запись заказов в БД	Информация успешно и безопасно сохранена в таблицах SQLite	Успешно

4 Оценка экономической эффективности проекта

Оценка экономической эффективности разработанного программного комплекса проводилась путем сравнения процессов "как есть" (ручная обработка заказов) и "как будет" (автоматизированная веб-платформа FreeRoll).

4.1 Методика расчета затрат на разработку

Затраты на реализацию проекта включают:

- Трудозатраты: Общее время разработки составило около 300 часов (изучение, проектирование, кодирование, тестирование). При средней ставке разработчика затраты на разработку составили порядка 150 000 руб.;
- Технические затраты: Использование open-source технологий (React, ASP.NET Core, SQLite) позволило исключить расходы на приобретение лицензионного ПО.

4.2 Анализ операционных выгод

Внедрение системы позволяет снизить эксплуатационные расходы за счет:

- сокращения времени обработки заказа: Переход от телефонных/бумажных заявок к цифровому интерфейсу сократил время обработки одного заказа оператором с 5-7 минут до 1-2 минут. При среднем потоке в 50 заказов в день экономия составляет порядка 3 часов рабочего времени ежедневно;
- снижения количества ошибок: Автоматизация формирования чека и интеграция базы товаров исключают ошибки "человеческого фактора" (неверная цена, пропуск позиции), что снижает издержки на исправление заказов и возвраты на 15%;
- аналитического модуля: Возможность отслеживания пиковых нагрузок и популярности товаров позволяет оптимизировать закупки ингредиентов, что сокращает объем списаний продуктов на 5-8% в месяц.

					ПГУ 1.090301.44.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

4.3 Расчет окупаемости

На основе полученных данных был рассчитан экономический эффект: годовая экономия (на ФОТ операторов и минимизации брака): 185 177 руб. Срок окупаемости проекта: рассчитывается по формуле

$$T_{\text{ок}} = \frac{C}{E},$$

где C - затраты на разработку, E - годовая экономия.

$$T_{\text{ок}} = \frac{150\,000}{185\,177} \approx 0,81 \text{ года (около 10 месяцев).}$$

Таким образом, разработанная система является экономически целесообразной, так как срок окупаемости проекта составляет менее одного года, что является высоким показателем для программных решений в сфере малого и среднего бизнеса.

Практическое применение результатов работы и фактическое подтверждение достигнутого экономического эффекта от внедрения разработанной системы в деятельность ИП Кравченко А. В. зафиксировано в справке о внедрении в Приложении Д.1. Данный документ удостоверяет успешную эксплуатацию программного комплекса в реальных производственных условиях, подтверждает соответствие системы заявленным функциональным требованиям и практическую значимость внедренных автоматизированных решений

5 Безопасность жизнедеятельности

При разработке и эксплуатации программного комплекса необходимо соблюдать комплекс мер, направленных на обеспечение безопасности пользователя, создание комфортных условий труда и сохранение работоспособности оборудования.

5.1 Правила безопасности при использовании ПК

В соответствии с ГОСТ 12.1.019-2017 электробезопасность при работе с вычислительной техникой обеспечивается путем обязательного защитного заземления оборудования. Питание системного блока и периферийных устройств осуществляется от сети переменного тока напряжением 220 В. Подключение всех компонентов системы должно производиться через розетки с заземляющим контактом, соединенным с внешним контуром заземления.

Требования к организации рабочего места с ПК регламентируются единым стандартом ГОСТ Р 50923-96. Согласно данному нормативному документу, рабочее пространство должно быть организовано с учетом минимизации статического напряжения мышц шейно-плечевой области и спины. Экран монитора следует располагать на оптимальном расстоянии от 600 до 700 мм от глаз пользователя. Освещенность рабочей поверхности стола должна быть равномерной и составлять от 300 до 500 лк, при этом оборудование необходимо размещать перпендикулярно источникам естественного света для исключения появления мешающих бликов.

При эксплуатации техники категорически запрещается использовать соединительные кабели с видимыми повреждениями изоляции или неисправные разъемы. Не допускается ограничение доступа воздуха к вентиляционным отверстиям системного блока и монитора. Соблюдение установленных нормативных правил позволяет обеспечить безопасные условия труда и сохранить

работоспособность технических средств в процессе разработки программного обеспечения или эксплуатации готовой платформы.

5.2 Требования производственной санитарии и пожарной безопасности

Поскольку разрабатываемая веб-платформа представляет собой программный продукт, основными производственными факторами, негативно влияющими на разработчика или оператора системы, являются зрительное перенапряжение и гиподинамия. В соответствии с актуальными санитарно-эпидемиологическими требованиями (СанПиН 1.2.3685-21), для предотвращения переутомления зрительного анализатора и развития профессиональных заболеваний, необходимо соблюдать строго регламентированный режим труда и отдыха.

Рекомендуется делать регламентированные перерывы продолжительностью 10-15 минут через каждый час непрерывной работы за экраном монитора при написании программного кода. Во время перерывов целесообразно выполнять комплексы гимнастических упражнений для глаз и снятия мышечного напряжения с опорно-двигательного аппарата.

Дополнительным критически важным аспектом является обеспечение пожарной безопасности на рабочем месте и в помещениях размещения серверного оборудования, на котором развернут бэкенд платформы. Согласно правилам противопожарного режима, помещения, оснащенные вычислительной техникой, относятся к категории помещений с повышенным риском возгорания из-за наличия большого количества электрокабелей и оборудования, работающего в непрерывном режиме.

Для предотвращения пожароопасных ситуаций категорически запрещается оставлять включенное в сеть оборудование без присмотра по окончании рабочего дня (за исключением выделенных серверов, предназначенных для круглосуточной работы). Запрещается размещать вблизи системных блоков

					ПГУ 1.090301.44.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

легковоспламеняющиеся материалы, жидкости и перекрывать контуры воздушного охлаждения серверов. Рабочие помещения должны быть в обязательном порядке оборудованы автоматическими системами пожарной сигнализации и первичными средствами пожаротушения (углекислотными электробезопасными огнетушителями класса ОУ, безопасными для тушения электроустановок под напряжением). Строгое соблюдение описанных правил гарантирует безопасность жизнедеятельности и предотвращает риск возникновения нештатных ситуаций на рабочем месте ИТ-специалиста.

					ПГУ 1.090301.44.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения выпускной квалификационной работы была достигнута поставленная цель: спроектирована и успешно реализована веб-платформа для автоматизации процессов приема, учета и обработки заказов службы доставки «FreeRoll». Все задачи, поставленные на этапе проектирования, выполнены в полном объеме.

На основе проведенного анализа предметной области и существующих аналогов был выбран и настроен оптимальный стек современных веб-технологий, позволивший создать гибкое и масштабируемое решение.

Спроектирована логическая структура реляционной базы данных и успешно реализована серверная часть на платформе ASP.NET Core (язык C#). Встроенная СУБД SQLite обеспечивает надежное хранение информации, а разработанный Backend эффективно решает задачи маршрутизации REST-запросов, криптографической защиты профилей и выполнения сложной бизнес-логики, включая расчет стоимости корзины и валидацию промокодов.

Разработано клиентское веб-приложение на базе библиотеки React, объединяющее интуитивно понятную витрину для конечных покупателей и многофункциональный интерфейс для административного персонала. Использование архитектурной методологии Feature-Sliced Design (FSD) обеспечило высокую модульность фронтенда, что позволяет оперативно адаптировать платформу под специфические задачи предприятия без риска нарушения работы существующего кода.

Проведенное комплексное тестирование подтвердило высокую надежность алгоритмов валидации пользовательского ввода, безопасность механизмов авторизации и корректность межуровневого обмена информацией. Разработанный программный комплекс представляет собой удобный, практичный и полностью готовый к внедрению IT-продукт, способный эффективно автоматизировать реальные бизнес-процессы предприятия.

Список использованных источников

1. Feature-Sliced Design: Architectural methodology for frontend projects [Электронный ресурс]. - URL: <https://feature-sliced.design/> (дата обращения 05.05.2026).
2. Официальная документация ASP.NET Core [Электронный ресурс]. - URL: <https://learn.microsoft.com/ru-ru/aspnet/core/> (дата обращения 11.04.2026).
3. Троелсен, Э. Язык программирования C# 9 и платформа .NET 5: основные принципы и практики программирования / Э. Троелсен, Ф. Джепикс. - Москва : Диалектика, 2022. - 1328 с. - ISBN 978-5-907458-15-4.
4. Фримен, А. ASP.NET Core в действии / А. Фримен. - Санкт-Петербург : Питер, 2022. - 1008 с. - ISBN 978-5-4461-1773-7.
5. React: The library for web and native user interfaces [Электронный ресурс]. - URL: <https://react.dev/> (дата обращения 19.04.2026).
6. Макдоннелл, М. Разработка одностраничных веб-приложений с использованием React и Redux / М. Макдоннелл. - Москва : ДМК Пресс, 2020. - 400 с. - ISBN 978-5-97060-843-2.
7. Флэнаган, Д. JavaScript. Подробное руководство. 7-е изд. / Д. Флэнаган. - Москва: Диалектика, 2021. - 720 с. - ISBN 978-5-907203-72-3.
8. Redux Toolkit: Official Web Site [Электронный ресурс]. - URL: <https://redux-toolkit.js.org/> (дата обращения 02.05.2026).
9. SQLite Official Documentation [Электронный ресурс]. - URL: <https://www.sqlite.org/docs.html> (дата обращения 13.05.2026).
10. Tailwind CSS Documentation [Электронный ресурс]. - URL: <https://tailwindcss.com/docs> (дата обращения 15.05.2026).

Приложение А

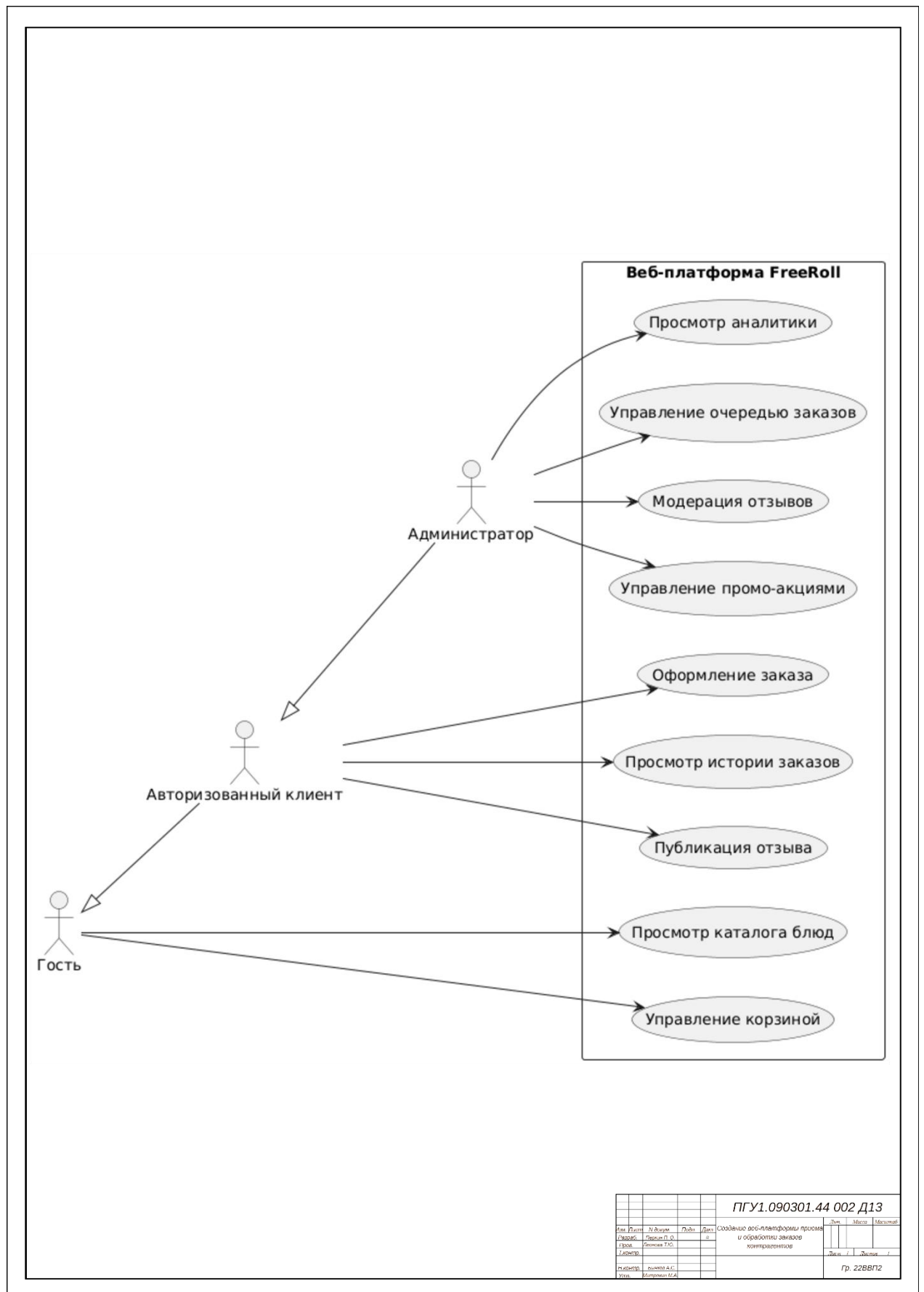
(обязательное)

Создание веб-платформы приема и обработки заказов контрагентов

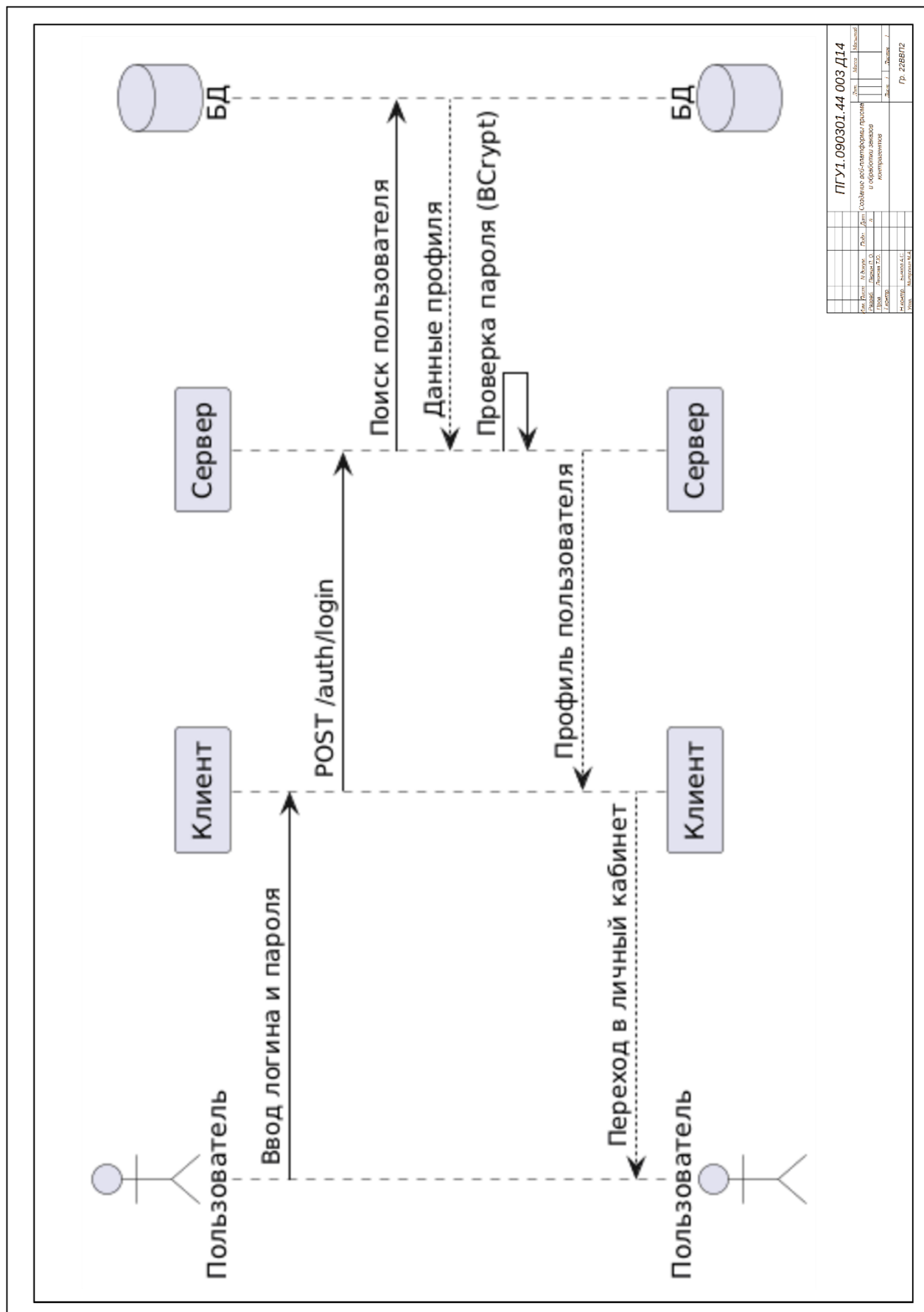
UML-диаграммы

					ПГУ 1.090301.44.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

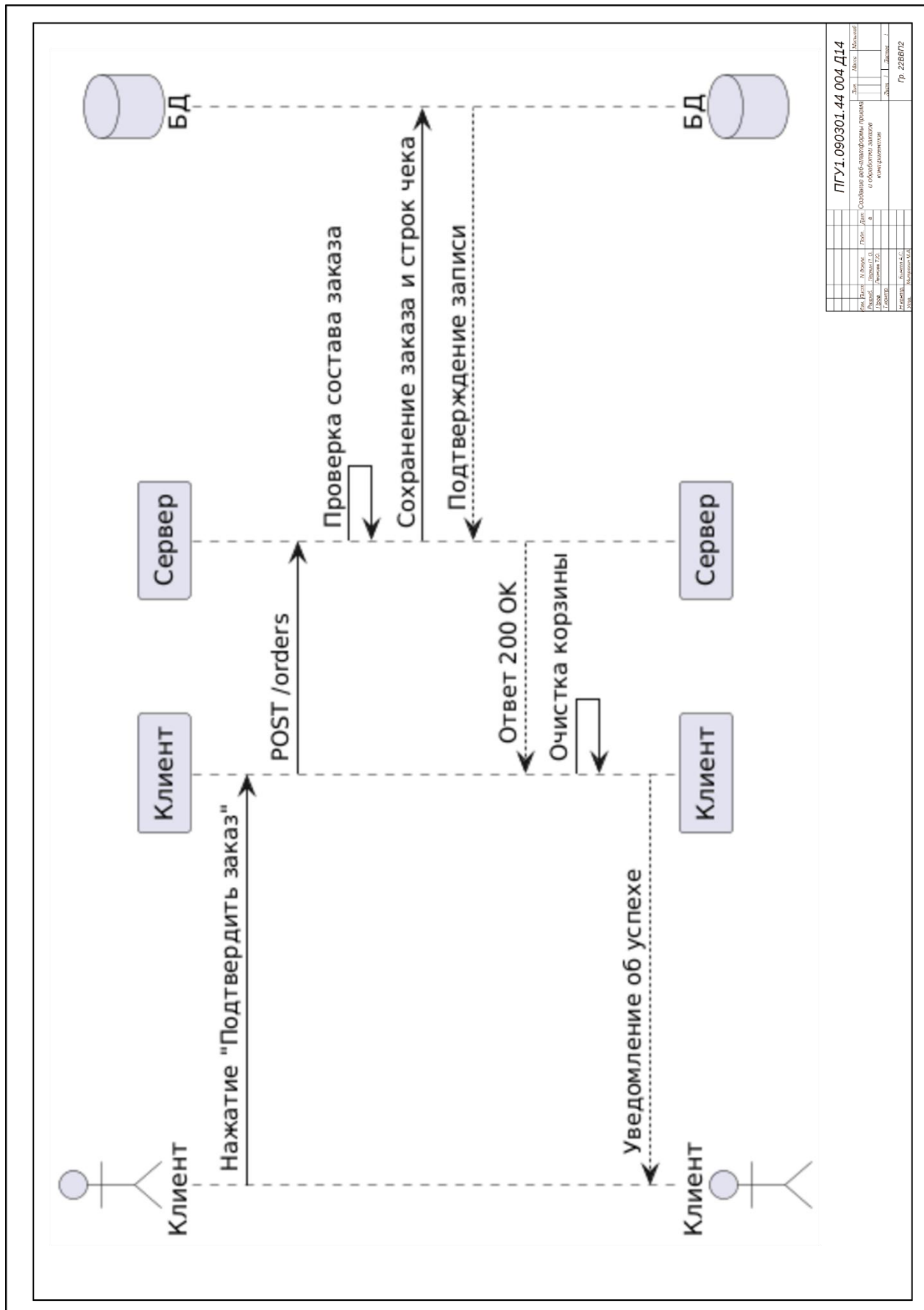
UML-диаграмма прецедентов



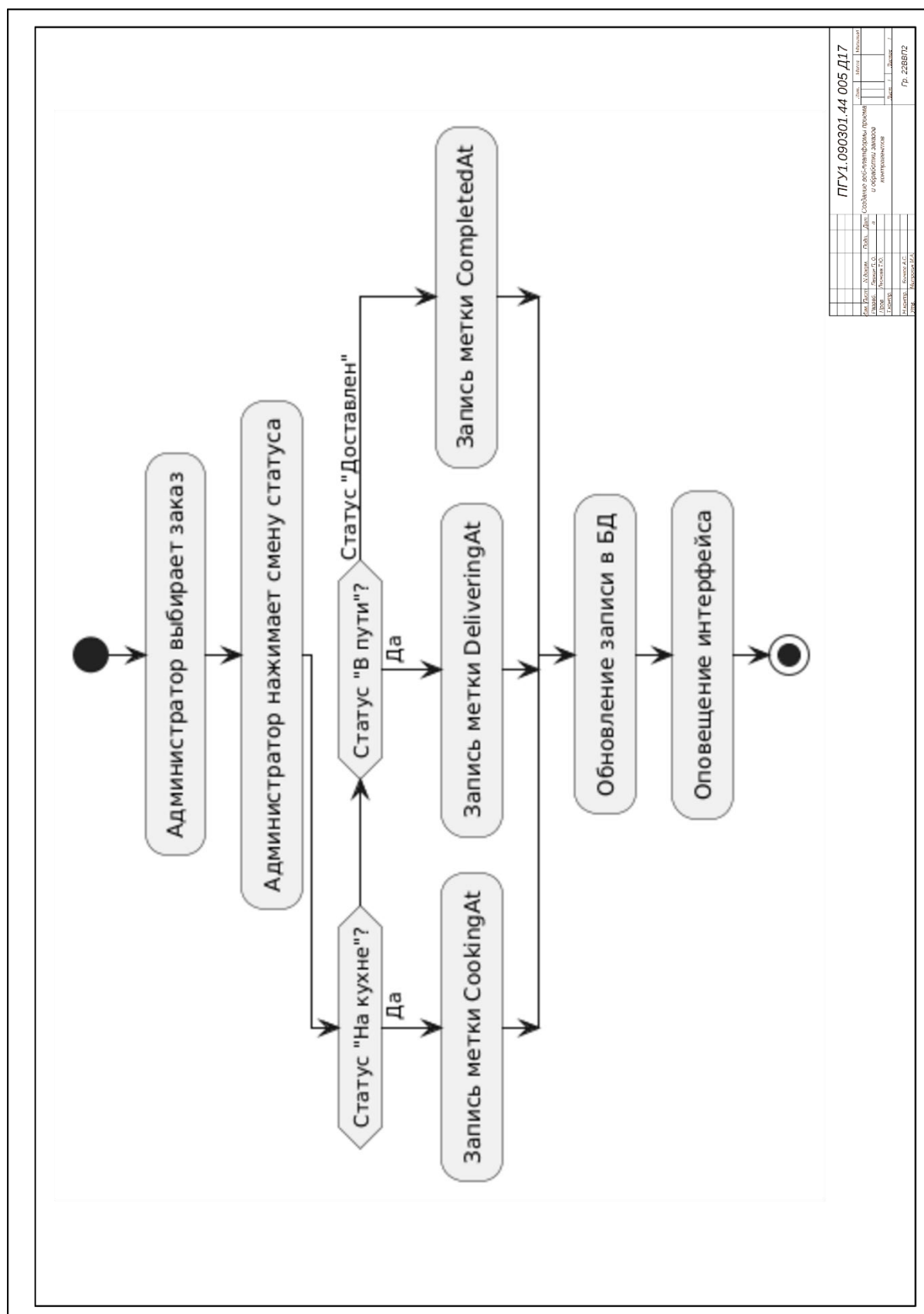
UML-диаграмма последовательности: авторизация



UML-диаграмма последовательности: оформление заказа



UML-диаграмма деятельности: обработка статуса заказа



ПГУ 1.090301.44 005 Д17									
Имя	В.И.	И.И.	И.И.	И.И.	И.И.	И.И.	И.И.	И.И.	И.И.
Фамилия	И.И.	И.И.	И.И.	И.И.	И.И.	И.И.	И.И.	И.И.	И.И.
Пол	М.	Ж.	М.	Ж.	М.	Ж.	М.	Ж.	М.
Дата	...	...	...	...	...	...	...	...	...
Время	...	...	...	...	...	...	...	...	...
Место	...	...	...	...	...	...	...	...	...
Страна	...	...	...	...	...	...	...	...	...
Город	...	...	...	...	...	...	...	...	...
Улица	...	...	...	...	...	...	...	...	...
Дом	...	...	...	...	...	...	...	...	...
Квартира	...	...	...	...	...	...	...	...	...
Телефон	...	...	...	...	...	...	...	...	...
Почта	...	...	...	...	...	...	...	...	...
Сайт	...	...	...	...	...	...	...	...	...
Гр	228672								

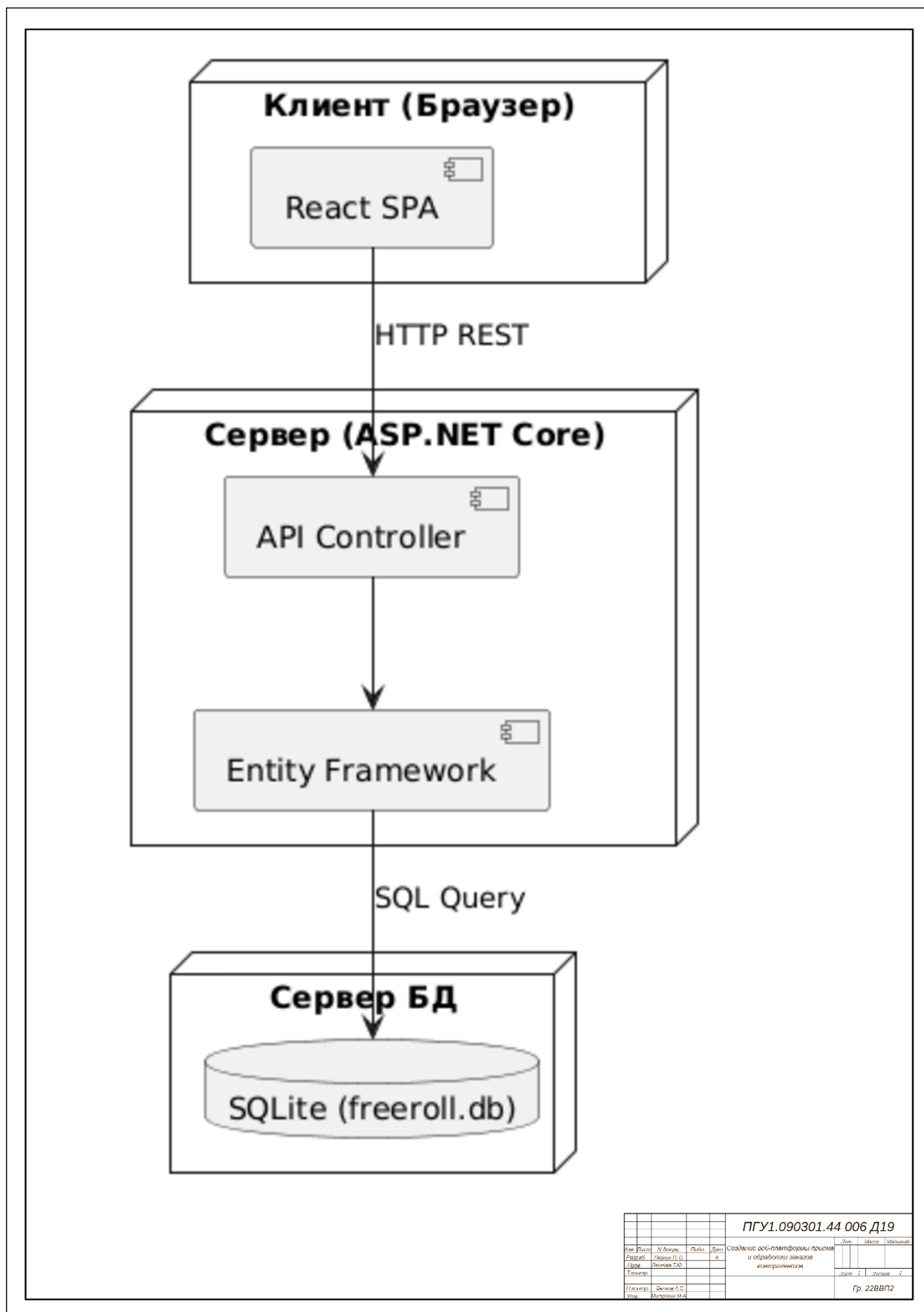
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.44.001 ПЗ

Лист

88

UML-диаграмма развертывания

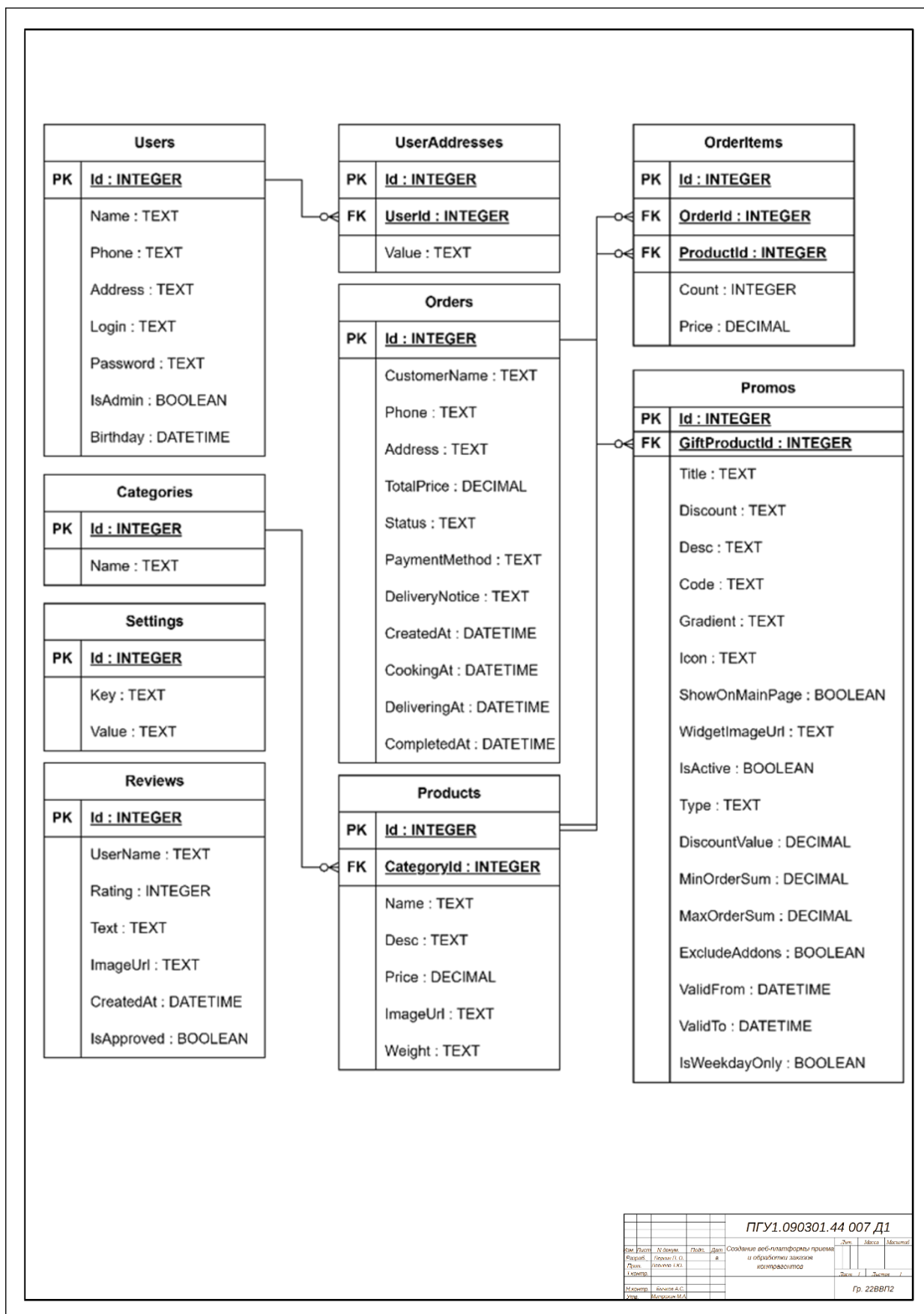


Приложение Б
(обязательное)

Создание веб-платформы приема и обработки заказов контрагентов

Схема базы данных

Логическая модель данных



ПГУ1.090301.44 007 Д1									
Создано	Разработано	Проверено	Утверждено	Дата	Дет.	Месс.	Месс.	Месс.	Месс.
Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор
Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор	Инициатор
Гр. 22ВВП2									

Приложение В
(обязательное)

Создание веб-платформы приема и обработки заказов контрагентов

Листинги программы

					ПГУ 1.090301.44.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

Листинг серверной части проекта

Файл Program.cs

```
using FreeRoll.Backend.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.FileProviders;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowReactApp", policy =>
    {
        policy.WithOrigins("http://localhost:5173")
            .AllowAnyHeader()
            .AllowAnyMethod();
    });
});

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite("Data Source=freeroll.db"));

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

var wwwrootPath = Path.Combine(Directory.GetCurrentDirectory(),
    "wwwroot");
var uploadsPath = Path.Combine(wwwrootPath, "uploads");

if (!Directory.Exists(wwwrootPath))
    Directory.CreateDirectory(wwwrootPath);
if (!Directory.Exists(uploadsPath))
    Directory.CreateDirectory(uploadsPath);

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseStaticFiles();

app.UseRouting();

app.UseCors("AllowReactApp");

app.UseHttpsRedirection();
```

					ПГУ 1.090301.44.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

```
app.UseAuthorization();
```

```
app.MapControllers();
```

```
app.Run();
```

Файл AppSetting.cs

```
namespace FreeRoll.Backend.Models
{
    public class AppSetting
    {
        public int Id { get; set; }
        public string Key { get; set; } = string.Empty;
        public string Value { get; set; } = string.Empty;
    }
}
```

Файл Category.cs

```
namespace FreeRoll.Backend.Models
{
    public class Category
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
    }
}
```

Файл Order.cs

```
using System;
using System.Collections.Generic;

namespace FreeRoll.Backend.Models
{
    public class Order
    {
        public int Id { get; set; }
        public string CustomerName { get; set; } = string.Empty;
        public string Phone { get; set; } = string.Empty;
        public string Address { get; set; } = string.Empty;
        public decimal TotalPrice { get; set; }

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public DateTime? CookingAt { get; set; }
        public DateTime? DeliveringAt { get; set; }
        public DateTime? CompletedAt { get; set; }

        public string Status { get; set; } = "New";
        public string PaymentMethod { get; set; } = "card";
    }
}
```



```

        public string DeliveryNotice { get; set; } = " Как можно
быстрее";

        public List<OrderItem> Items { get; set; } = new();
    }

    public class OrderItem
    {
        public int Id { get; set; }
        public int OrderId { get; set; }
        public int ProductId { get; set; }
        public Product? Product { get; set; }
        public int Count { get; set; }
        public decimal Price { get; set; }
    }

    public class OrderCreatedDto
    {
        public string CustomerName { get; set; } = string.Empty;
        public string Phone { get; set; } = string.Empty;
        public string Address { get; set; } = string.Empty;
        public decimal TotalPrice { get; set; }
        public string PaymentMethod { get; set; } = "card";
        public string DeliveryNotice { get; set; } = " Как можно
быстрее";

        public List<OrderItemCreateDto> Items { get; set; } = new();
    }

    public class OrderItemCreateDto
    {
        public int ProductId { get; set; }
        public int Count { get; set; }
        public decimal Price { get; set; }
    }
}

```

Файл Product.cs

```

namespace FreeRoll.Backend.Models
{
    public class Product
    {
        public int Id { get; set; }
        public int CategoryId { get; set; }
        public string Name { get; set; } = string.Empty;
        public string Desc { get; set; } = string.Empty;
        public decimal Price { get; set; }
        public string? ImageUrl { get; set; }
        public string? Weight { get; set; }
    }
}

```

Файл Promo.cs

```
using System;

namespace FreeRoll.Backend.Models
{
    public class Promo
    {
        public int Id { get; set; }

        public string Title { get; set; } = string.Empty;
        public string Discount { get; set; } = string.Empty;
        public string Desc { get; set; } = string.Empty;
        public string Code { get; set; } = string.Empty;
        public string Gradient { get; set; } = string.Empty;
        public string Icon { get; set; } = string.Empty;
        public bool ShowOnMainPage { get; set; } = false;
        public string WidgetImageUrl { get; set; } = string.Empty;

        public bool IsActive { get; set; } = true;

        public string Type { get; set; } = "discount";
        public decimal DiscountValue { get; set; } = 0;
        public int? GiftProductId { get; set; }

        public decimal MinOrderSum { get; set; } = 0;

        public decimal MaxOrderSum { get; set; } = 0;

        public bool ExcludeAddons { get; set; } = false;

        public DateTime? ValidFrom { get; set; }
        public DateTime? ValidTo { get; set; }

        public TimeSpan? StartTime { get; set; }
        public TimeSpan? EndTime { get; set; }
        public bool IsWeekdayOnly { get; set; } = false;
    }
}
```

Файл Review.cs

```
namespace FreeRoll.Backend.Models
{
    public class Review
    {
        public int Id { get; set; }
        public string UserName { get; set; } = string.Empty;
        public int Rating { get; set; }
        public string Text { get; set; } = string.Empty;
        public string? ImageUrl { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    }
}
```

```

        public bool IsApproved { get; set; } = false;
    }
}

```

Файл User.cs

```

using System;
using System.Collections.Generic;
using System.Text.Json.Serialization;

namespace FreeRoll.Backend.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        public string Phone { get; set; } = string.Empty;

        public string Address { get; set; } = string.Empty;

        public string Login { get; set; } = string.Empty;
        public string Password { get; set; } = string.Empty;

        public bool IsAdmin { get; set; } = false;
        public DateTime? Birthday { get; set; }

        public List<UserAddress> Addresses { get; set; } = new();
    }

    public class UserAddress
    {
        public int Id { get; set; }
        public int UserId { get; set; }

        [JsonIgnore]
        public User? User { get; set; }

        public string Value { get; set; } = string.Empty;
    }

    public class AddressDto
    {
        public string Address { get; set; } = string.Empty;
    }
}

```

Файл AppDbContext.cs

```

using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Models;

namespace FreeRoll.Backend.Data
{
    public class AppDbContext : DbContext

```

```

    {
        public AppDbContext (DbContextOptions<AppDbContext> options)
: base(options) { }

        public DbSet<Product> Products { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<OrderItem> OrderItems { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Promo> Promos { get; set; }
        public DbSet<AppSetting> Settings { get; set; }
        public DbSet<Review> Reviews { get; set; }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<OrderItem>()
                .HasOne(i => i.Product)
                .WithMany()
                .HasForeignKey(i => i.ProductId);
        }
    }
}

```

Файл AuthController.cs

```

using FreeRoll.Backend.Data;
using FreeRoll.Backend.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;
using BCrypt.Net;

namespace FreeRoll.Backend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly AppDbContext _context;

        public AuthController(AppDbContext context)
        {
            _context = context;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] User
user)
        {

```



```

        if (await _context.Users.AnyAsync(u => u.Login ==
user.Login))
        {
            return BadRequest(new { message = "Пользователь с
таким логином уже существует!" });
        }

        if (!string.IsNullOrEmpty(user.Address))
        {
            user.Addresses.Add(new UserAddress { Value =
user.Address });
        }

        user.Password =
BCrypt.Net.BCrypt.HashPassword(user.Password);

        _context.Users.Add(user);
        await _context.SaveChangesAsync();

        user.Password = "";
        return Ok(new { message = "Регистрация успешна!", user
});
    }

    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] User
loginData)
    {
        var adminExists = await _context.Users.AnyAsync(u =>
u.IsAdmin);
        if (!adminExists && loginData.Login == "admin" &&
loginData.Password == "admin")
        {
            var newAdmin = new User
            {
                Login = "admin",
                Password =
BCrypt.Net.BCrypt.HashPassword("admin"),
                Name = "Администратор",
                IsAdmin = true
            };
            _context.Users.Add(newAdmin);
            await _context.SaveChangesAsync();
        }

        var user = await _context.Users
.Include(u => u.Addresses)
.FirstOrDefaultAsync(u => u.Login ==
loginData.Login);

```

```

        if (user == null ||
!BCrypt.Net.BCrypt.Verify(loginData.Password, user.Password))
        {
            return Unauthorized(new { message = "Неверный логин
или пароль!" });
        }

        user.Password = "";
        return Ok(new { message = "Успешный вход!", user });
    }

[HttpPost("{id}/addresses")]
public async Task<IActionResult> AddAddress(int id,
[FromBody] AddressDto dto)
{
    var user = await _context.Users.Include(u =>
u.Addresses).FirstOrDefaultAsync(u => u.Id == id);
    if (user == null) return NotFound(new { message =
"Пользователь не найден" });

    if (!string.IsNullOrEmpty(dto.Address))
    {
        user.Addresses.Add(new UserAddress { Value =
dto.Address });
        await _context.SaveChangesAsync();
    }

    user.Password = "";
    return Ok(new { message = "Адрес добавлен", user });
}

[HttpDelete("{userId}/addresses/{addressId}")]
public async Task<IActionResult> DeleteAddress(int userId,
int addressId)
{
    var user = await _context.Users.Include(u =>
u.Addresses).FirstOrDefaultAsync(u => u.Id == userId);
    if (user == null) return NotFound(new { message =
"Пользователь не найден" });

    var address = user.Addresses.FirstOrDefault(a => a.Id ==
addressId);
    if (address != null)
    {
        _context.Set<UserAddress>().Remove(address);
        await _context.SaveChangesAsync();
    }

    user.Password = "";

```

```

        return Ok(new { message = "Адрес удален", user });
    }
}

```

Файл CategoriesController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Data;
using FreeRoll.Backend.Models;

namespace FreeRoll.Backend.Controllers;

[ApiController]
[Route("api/[controller]")]
public class CategoriesController : ControllerBase
{
    private readonly AppDbContext _context;

    public CategoriesController(AppDbContext context) { _context = context; }

    [HttpGet]
    public async Task<ActionResult<IEnumerable<Category>>>
    GetCategories()
        => await _context.Categories.ToListAsync();

    [HttpPost]
    public async Task<IActionResult> CreateCategory(Category
category)
    {
        _context.Categories.Add(category);
        await _context.SaveChangesAsync();
        return Ok(category);
    }

    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateCategory(int id, Category
category)
    {
        if (id != category.Id) return BadRequest(new { message = "ID
не совпадают" });

        var existingCat = await _context.Categories.FindAsync(id);
        if (existingCat == null) return NotFound(new { message =
"Категория не найдена" });

        existingCat.Name = category.Name;

        await _context.SaveChangesAsync();
        return NoContent();
    }
}

```

					ПГУ 1.090301.44.001 ПЗ	Лист
						101
Изм.	Лист	№ докум.	Подпись	Дата		

```

[HttpDelete("{id}")]
public async Task<IActionResult> DeleteCategory(int id)
{
    var cat = await _context.Categories.FindAsync(id);
    if (cat == null) return NotFound();

    var hasProducts = await _context.Products.AnyAsync(p =>
p.CategoryId == id);
    if (hasProducts)
    {
        return BadRequest(new { message = "Нельзя удалить
категорию! Сначала перенесите или удалите товары внутри нее." });
    }

    _context.Categories.Remove(cat);
    await _context.SaveChangesAsync();
    return NoContent();
}
}

```

Файл OrdersController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Data;
using FreeRoll.Backend.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FreeRoll.Backend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class OrdersController : ControllerBase
    {
        private readonly AppDbContext _context;

        public OrdersController(AppDbContext context)
        {
            _context = context;
        }

        [HttpPost]
        public async Task<IActionResult> CreateOrder(OrderCreatedDto
request)
        {
            if (request.Items == null || !request.Items.Any())
            {
                return BadRequest(new { message = "Нельзя отправить
пустой заказ" });
            }
        }
    }
}

```

					ПГУ 1.090301.44.001 ПЗ	Лист
						102
Изм.	Лист	№ докум.	Подпись	Дата		


```

var order = new Order
{
    CustomerName = request.CustomerName,
    Phone = request.Phone,
    Address = request.Address,
    TotalPrice = request.TotalPrice,
    PaymentMethod = request.PaymentMethod,
    DeliveryNotice = request.DeliveryNotice,
    CreatedAt = DateTime.SpecifyKind(DateTime.UtcNow,
DateTimeKind.Utc),
    Status = "New"
};

foreach (var item in request.Items)
{
    var productExists = await
_context.Products.AnyAsync(p => p.Id == item.ProductId);
    if (!productExists)
    {
        return BadRequest(new { message = "Товар с
указанным ProductId не найден в базе данных меню!" });
    }

    order.Items.Add(new OrderItem
    {
        ProductId = item.ProductId,
        Count = item.Count,
        Price = item.Price
    });
}

_context.Orders.Add(order);
await _context.SaveChangesAsync();

return Ok(new { message = "Заказ успешно оформлен!",
orderId = order.Id });
}

[HttpGet]
public async Task<ActionResult<IEnumerable<Order>>>
GetOrders()
{
    var orders = await _context.Orders
        .Include(o => o.Items)
        .ThenInclude(x =>
x.Product)
        .OrderByDescending(o =>
o.CreatedAt)
        .ToListAsync();

    return Ok(orders);
}

```

```

    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Order>> GetOrder(int id)
    {
        var order = await _context.Orders
                                .Include(o => o.Items)
                                .ThenInclude(x =>
x.Product)
                                .FirstOrDefaultAsync(o => o.Id
== id);

        if (order == null)
        {
            return NotFound(new { message = "Заказ не найден"
});
        }

        return Ok(order);
    }

    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteOrder(int id)
    {
        var order = await _context.Orders.FindAsync(id);
        if (order == null) return NotFound(new { message =
"Заказ не найден" });

        _context.Orders.Remove(order);
        await _context.SaveChangesAsync();
        return NoContent();
    }

    [HttpPatch("{id}/status")]
    public async Task<IActionResult> UpdateStatus(int id,
[FromBody] string newStatus)
    {
        var order = await _context.Orders.FindAsync(id);
        if (order == null) return NotFound(new { message =
"Заказ не найден" });

        order.Status = newStatus;

        var now = DateTime.SpecifyKind(DateTime.UtcNow,
DateTimeKind.Utc);

        if (newStatus == "Cooking" && order.CookingAt == null)
            order.CookingAt = now;

        if (newStatus == "Delivering" && order.DeliveringAt ==
null)

```

					ПГУ 1.090301.44.001 ПЗ	Лист
						104
Изм.	Лист	№ докум.	Подпись	Дата		

```

        order.DeliveringAt = now;

        if (newStatus == "Completed" && order.CompletedAt ==
null)
            order.CompletedAt = now;

        await _context.SaveChangesAsync();
        return Ok(new { message = "Статус заказа обновлен!",
status = order.Status });
    }

    [HttpDelete("clear-all")]
    public async Task<IActionResult> ClearAllOrders()
    {
        var items = await _context.OrderItems.ToListAsync();
        var orders = await _context.Orders.ToListAsync();

        _context.OrderItems.RemoveRange(items);
        _context.Orders.RemoveRange(orders);
        await _context.SaveChangesAsync();

        return Ok(new { message = "История заказов полностью
очищена!" });
    }

    [HttpGet("stats")]
    public async Task<IActionResult> GetStats([FromQuery]
DateTime? startDate, [FromQuery] DateTime? endDate)
    {
        var end = endDate ?? DateTime.UtcNow;
        var start = startDate ?? end.AddDays(-7);

        var completedOrders = await _context.Orders
            .Include(o => o.Items)
            .ThenInclude(i => i.Product)
            .Where(o => o.Status == "Completed" && o.CreatedAt
>= start && o.CreatedAt <= end)
            .ToListAsync();

        var totalRevenue = completedOrders.Sum(o =>
o.TotalPrice);
        var totalOrders = completedOrders.Count;

        var revenueByDay = completedOrders
            .GroupBy(o => o.CreatedAt.Date)
            .Select(g => new
            {
                date = g.Key.ToString("dd.MM"),
                revenue = g.Sum(o => o.TotalPrice)
            });
    }

```

```

    })
    .OrderBy(x => x.date)
    .ToList();

    var topProducts = completedOrders
        .SelectMany(o => o.Items)
        .GroupBy(i => i.Product != null ? i.Product.Name :
"Удаленный товар")
        .Select(g => new
        {
            name = g.Key,
            count = g.Sum(i => i.Count)
        })
        .OrderByDescending(x => x.count)
        .Take(5)
        .ToList();

    var ordersByHour = completedOrders
        .GroupBy(o => o.CreatedAt.ToLocalTime().Hour)
        .Select(g => new { hour = $"{g.Key:D2}:00", count =
g.Count() })
        .OrderBy(x => int.Parse(x.hour.Split(':')[0]))
        .ToList();

    var daysOfWeekRus = new[] { "Бс", "Пн", "Вт", "Ср",
"Чт", "Пт", "Сб" };
    var ordersByDayOfWeek = completedOrders
        .GroupBy(o =>
(int)o.CreatedAt.ToLocalTime().DayOfWeek)
        .Select(g => new { day = daysOfWeekRus[g.Key], count
= g.Count(), dayIndex = g.Key })
        .OrderBy(x => (x.dayIndex + 6) % 7)
        .Select(x => new { day = x.day, count = x.count })
        .ToList();

    var kitchenTimes = completedOrders
        .Where(o => o.CookingAt.HasValue &&
o.DeliveringAt.HasValue)
        .Select(o => (o.DeliveringAt.Value -
o.CookingAt.Value).TotalMinutes)
        .ToList();
    var avgCookingTime = kitchenTimes.Any() ?
Math.Round(kitchenTimes.Average()) : 0;

    var deliveryTimes = completedOrders
        .Where(o => o.DeliveringAt.HasValue &&
o.CompletedAt.HasValue)
        .Select(o => (o.CompletedAt.Value -
o.DeliveringAt.Value).TotalMinutes)
        .ToList();

```



```

        var avgDeliveryTime = deliveryTimes.Any() ?
Math.Round(deliveryTimes.Average()) : 0;

        return Ok(new
        {
            totalRevenue,
            totalOrders,
            revenueByDay,
            topProducts,
            ordersByHour,
            ordersByDayOfWeek,
            avgCookingTime,
            avgDeliveryTime
        });
    }
}
}

```

Файл ProductsController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Data;
using FreeRoll.Backend.Models;

namespace FreeRoll.Backend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductsController : ControllerBase
    {
        private readonly AppDbContext _context;

        public ProductsController(AppDbContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Product>>>
GetProducts()
        {
            return await _context.Products.ToListAsync();
        }

        [HttpPost]
        public async Task<ActionResult<Product>> AddProduct(Product
product)
        {
            _context.Products.Add(product);
        }
    }
}

```

```

        await _context.SaveChangesAsync();

        return Ok(product);
    }

    [HttpPost("bulk")]
    public async Task<IActionResult>
AddProductsBulk(List<Product> products)
    {
        _context.Products.AddRange(products);

        await _context.SaveChangesAsync();

        return Ok(new { message = $"Успешно добавлено
{products.Count} товаров в базу!" });
    }

    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateProduct(int id,
Product updatedProduct)
    {
        if (id != updatedProduct.Id)
        {
            return BadRequest("ID товара не совпадает");
        }

        _context.Entry(updatedProduct).State =
EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!_context.Products.Any(p => p.Id == id))
            {
                return NotFound("Товар не найден");
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }
}

```

```

[HttpDelete("{id}")]
public async Task<IActionResult> DeleteProduct(int id)
{
    var product = await _context.Products.FindAsync(id);
    if (product == null)
    {
        return NotFound("Товар не найден");
    }

    _context.Products.Remove(product);
    await _context.SaveChangesAsync();

    return NoContent();
}

[HttpDelete("clear-all")]
public async Task<IActionResult> ClearAllProducts()
{
    var allProducts = _context.Products.ToList();
    _context.Products.RemoveRange(allProducts);
    await _context.SaveChangesAsync();
    return Ok(new { message = "Меню полностью очищено!" });
}
}
}

```

Файл PromosController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Data;
using FreeRoll.Backend.Models;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FreeRoll.Backend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class PromosController : ControllerBase
    {
        private readonly AppDbContext _context;
        public PromosController(AppDbContext context) { _context = context; }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Promo>>>
        GetPromos()
        {
            return await _context.Promos.ToListAsync();
        }
    }
}

```

```

    }

    [HttpGet("active")]
    public async Task<ActionResult<IEnumerable<Promo>>>
GetActivePromos()
    {
        return await _context.Promos.Where(p =>
p.IsActive).ToListAsync();
    }

    [HttpPost]
    public async Task<ActionResult> CreatePromo(Promo promo)
    {

        if (!string.IsNullOrEmpty(promo.Code))
        {
            promo.Code = promo.Code.ToUpper().Trim();
        }

        _context.Promos.Add(promo);
        await _context.SaveChangesAsync();
        return Ok(promo);
    }

    [HttpDelete("{id}")]
    public async Task<ActionResult> DeletePromo(int id)
    {
        var promo = await _context.Promos.FindAsync(id);
        if (promo == null) return NotFound();
        _context.Promos.Remove(promo);
        await _context.SaveChangesAsync();
        return NoContent();
    }

    [HttpPut("{id}")]
    public async Task<ActionResult> PutPromo(int id, Promo
promo)
    {
        if (id != promo.Id) return BadRequest();

        if (!string.IsNullOrEmpty(promo.Code))
        {
            promo.Code = promo.Code.ToUpper().Trim();
        }

        _context.Entry(promo).State = EntityState.Modified;
        await _context.SaveChangesAsync();
        return NoContent();
    }
}

```


Файл ReviewsController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Data;
using FreeRoll.Backend.Models;

namespace FreeRoll.Backend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ReviewsController : ControllerBase
    {
        private readonly AppDbContext _context;

        public ReviewsController(AppDbContext context) { _context = context; }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Review>>>
        GetApprovedReviews()
        {
            return await _context.Reviews
                                .Where(r => r.IsApproved)
                                .OrderByDescending(r =>
r.CreatedAt)
                                .ToListAsync();
        }

        [HttpGet("moderation")]
        public async Task<ActionResult<IEnumerable<Review>>>
        GetUnapprovedReviews()
        {
            return await _context.Reviews
                                .Where(r => !r.IsApproved)
                                .OrderByDescending(r =>
r.CreatedAt)
                                .ToListAsync();
        }

        [HttpPost]
        public async Task<IActionResult> CreateReview(Review review)
        {
            if (review.Rating < 1 || review.Rating > 5) return
BadRequest();

            review.CreatedAt = DateTime.UtcNow;
            review.IsApproved = false;

            _context.Reviews.Add(review);
        }
    }
}
```

```

        await _context.SaveChangesAsync();
        return Ok(new { message = "Отзыв отправлен на
модерацию!" });
    }

    [HttpPatch("{id}/approve")]
    public async Task<IActionResult> ApproveReview(int id)
    {
        var review = await _context.Reviews.FindAsync(id);
        if (review == null) return NotFound();

        review.IsApproved = true;
        await _context.SaveChangesAsync();
        return Ok(new { message = "Отзыв успешно опубликован!"
    });
    }

    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteReview(int id)
    {
        var review = await _context.Reviews.FindAsync(id);
        if (review == null) return NotFound();

        _context.Reviews.Remove(review);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}

```

Файл SettingsController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using FreeRoll.Backend.Data;
using FreeRoll.Backend.Models;

namespace FreeRoll.Backend.Controllers;

[ApiController]
[Route("api/[controller]")]
public class SettingsController : ControllerBase
{
    private readonly AppDbContext _context;
    public SettingsController(AppDbContext context) { _context =
context; }

    [HttpGet("{key}")]
    public async Task<IActionResult> GetSetting(string key)
    {

```

```

        var setting = await _context.Settings.FirstOrDefaultAsync(s
=> s.Key == key);
        return Ok(setting);
    }

    [HttpPost]
    public async Task<IActionResult> UpdateSetting([FromBody]
AppSetting input)
    {
        var setting = await _context.Settings.FirstOrDefaultAsync(s
=> s.Key == input.Key);

        if (setting == null)
        {
            _context.Settings.Add(input);
        }
        else
        {
            setting.Value = input.Value;
        }

        await _context.SaveChangesAsync();
        return Ok(setting);
    }
}

```

Файл UploadController.cs

```

using Microsoft.AspNetCore.Mvc;

namespace FreeRoll.Backend.Controllers;

[ApiController]
[Route("api/[controller]")]
public class UploadController : ControllerBase
{
    private readonly IWebHostEnvironment _env;

    public UploadController(IWebHostEnvironment env)
    {
        _env = env;
    }

    [HttpPost]
    [Consumes("multipart/form-data")]
    public async Task<IActionResult> UploadImage(IFormFile file,
[FromForm] string folder = "common", [FromForm] string fileName =
null)
    {
        if (file == null || file.Length == 0)
            return BadRequest(new { message = "Файл не выбран" });

        try

```

```

        {
            var wwwroot = _env.WebRootPath ??
Path.Combine(Directory.GetCurrentDirectory(), "wwwroot");

            var targetFolder = Path.Combine(wwwroot, "uploads",
folder);

            if (!Directory.Exists(targetFolder))
            {
                Directory.CreateDirectory(targetFolder);
            }

            var finalFileName = string.IsNullOrEmpty(fileName)
                ? Guid.NewGuid().ToString() +
Path.GetExtension(file.FileName)
                : fileName;

            var filePath = Path.Combine(targetFolder,
finalFileName);

            using (var stream = new FileStream(filePath,
FileMode.Create))
            {
                await file.CopyToAsync(stream);
            }

            return Ok(new { url =
$"https://localhost:7031/uploads/{folder}/{finalFileName}" });
        }
        catch (Exception ex)
        {
            return StatusCode(500, new { message = ex.Message });
        }
    }
}

```

					ПГУ 1.090301.44.001 ПЗ	Лист
						114
Изм.	Лист	№ докум.	Подпись	Дата		

Листинг клиентской части проекта

Файл index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Freeroll</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Файл App.jsx

```
import { RouterProvider } from 'react-router-dom';
import { Provider } from 'react-redux';
import { store } from './store/store';
import { router } from './providers/router';
import './styles/global.css';

export const App = () => {
  return (
    <Provider store={store}>
      <RouterProvider router={router} />
    </Provider>
  );
};
```

Файл router.jsx

```
import { createBrowserRouter } from 'react-router-dom';
import { BaseLayout } from '@app/layouts/BaseLayout';
import { HomePage } from '@pages/home';
import { CartPage } from '@pages/cart';
import { PromosPage } from '@pages/promo';
import { ReviewsPage } from '@pages/reviews';
import { AdminOrdersPage } from '@pages/admin-orders';
import { ProfilePage } from '@pages/profile';
import { AdminStatsPage } from '@pages/admin-stats';

export const router = createBrowserRouter([
  {
    path: '/',
    element: <BaseLayout />,
    children: [
      { path: '/', element: <HomePage /> },
      { path: '/cart', element: <CartPage /> },
    ],
  },
]);
```

					ПГУ 1.090301.44.001 ПЗ	Лист
						115
Изм.	Лист	№ докум.	Подпись	Дата		

```

        { path: '/promos', element: <PromosPage /> },
        { path: '/profile', element: <ProfilePage /> },
        { path: '/reviews', element: <ReviewsPage /> },
        { path: '/admin/orders', element: <AdminOrdersPage /> },
        { path: '/admin/stats', element: <AdminStatsPage /> },
    ],
  },
]);

```

Файл BaseLayout.jsx

```

import { Outlet } from 'react-router-dom';

import { Header } from '@widgets/header';
import { Footer } from '@widgets/footer';
import { AdminFloatingButton, AdminModalManager } from
  '@features/admin-tools';

import { ScrollToTopButton } from '@shared/ui/ScrollToTopButton';
import { ScrollToTop } from '@shared/lib/ScrollToTop';

export const BaseLayout = () => {
  return (
    <div className="min-h-screen flex flex-col">
      </* Утилита скролла при переходе по страницам */>
      <ScrollToTop />

      <Header />

      <main className="flex-1 container mx-auto px-4 py-8">
        <Outlet />
      </main>

      <Footer />

      <ScrollToTopButton />

      </* Глобальные инструменты админа */>
      <AdminFloatingButton />
      <AdminModalManager />
    </div>
  );
};

```

Файл store.js

```

import { configureStore } from '@reduxjs/toolkit';
import cartReducer from '@entities/cart/model/cartSlice';
import productReducer from '@entities/product/model/productSlice';
import userReducer from '@entities/user/model/userSlice.js'

export const store = configureStore({
  reducer: {
    cart: cartReducer,

```

```

        products: productReducer,
        user: userReducer,
    },
  ));

```

Файл HomePage.jsx

```

import { useState } from 'react';
import { Search } from 'lucide-react';
import { ProductList } from '@widgets/productList';
import { useHomeData } from '../lib/useHomeData';
import { CategoryNav } from '@widgets/category-nav';
import { PromoCarousel } from '@widgets/promo-carousel';

export const HomePage = () => {
  const [searchQuery, setSearchQuery] = useState('');

  const { categories, promos, products, currentUser, isLoading,
    error } = useHomeData();

  return (
    <div className="flex flex-col gap-10 animate-in fade-in
duration-300">
      {/* 1. Навигация по категориям */}
      <CategoryNav
        categories={categories}
        isLoading={isLoading}
        currentUser={currentUser}
      />

      {/* 2. Карусель баннеров акций */}
      <PromoCarousel promos={promos} />

      {/* 3. Строка живого поиска */}
      <div className="relative max-w-2xl mx-auto w-full px-4">
        <input
          type="text"
          placeholder="Найти блюдо..."
          value={searchQuery}
          onChange={(e) => setSearchQuery(e.target.value)}
          className="w-full bg-white/60 backdrop-blur-sm
border border-white/80 rounded-full py-4 pl-14 pr-6 text-lg text-
slate-700 placeholder-slate-400 focus:outline-none focus:ring-2
focus:ring-[#0099ff] shadow-sm transition-all"
        />
        <Search className="absolute left-9 top-1/2 -
translate-y-1/2 text-slate-400" size={24} />
      </div>

      {/* 4. Рендер списков товаров по категориям */}
      <div className="px-4">
        {!isLoading && !error && categories.map(category =>

```

```

{
    const categoryProducts = products.filter(p =>
        p.categoryId === category.id &&
        p.name.toLowerCase().includes(searchQuery.trim().toLowerCase())
    );

    if (categoryProducts.length === 0 &&
        !currentUser?.isAdmin) return null;

    return (
        <ProductList key={category.id}
        id={category.id} title={category.name} products={categoryProducts}
        />
    );
    })}
</div>
</div>
);
};

```

Файл useHomeData.js

```

import { useState, useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchProducts } from
'@/entities/product/model/productSlice';
import { apiClient } from '@shared/api/apiClient';

export const useHomeData = () => {
    const dispatch = useDispatch();
    const [categories, setCategories] = useState([]);
    const [promos, setPromos] = useState([]);
    const [isCategoriesLoading, setIsCategoriesLoading] =
    useState(true);

    const { items: products, isLoading: isProductsLoading, error } =
    useSelector((state) => state.products);
    const currentUser = useSelector((state) =>
    state.user?.currentUser);

    useEffect(() => {
        dispatch(fetchProducts());

        apiClient.get('/categories')
            .then(data => {
                setCategories(data);
                setIsCategoriesLoading(false);
            })
            .catch(err => console.error("Ошибка категорий:", err));

        apiClient.get('/promos/active')
            .then(data => {

```



```

        if (Array.isArray(data)) {
            setPromos(data.filter(p => p.showOnMainPage &&
p.widgetImageUrl));
        }
    })
    .catch(err => console.error("Ошибка виджетов акций:",
err));
    }, [dispatch]);

    return {
        categories,
        promos,
        products,
        currentUser,
        isLoading: isCategoriesLoading || isProductsLoading,
        error
    };
};

```

Файл PromosPage.jsx

```

import { useSelector } from 'react-redux';
import { AdminPromoControls } from '@/features/admin-tools';
import { PromoCard } from '@/entities/promo';
import { useCopyToClipboard } from
'@/shared/lib/useCopyToClipboard';
import { usePromos } from '../lib/usePromos';

export const PromosPage = () => {
    const currentUser = useSelector((state) =>
state.user?.currentUser);
    const isAdmin = currentUser?.isAdmin;

    const { promos, rules, isLoading, handleEditRules } =
usePromos(isAdmin);
    const { copiedId, copy } = useCopyToClipboard();

    if (isLoading) return (
        <div className="flex flex-col items-center justify-center
py-32 gap-4">
            <div className="w-12 h-12 border-4 border-[#0099ff]
border-t-transparent rounded-full animate-spin"></div>
            <p className="font-bold text-slate-400">Загружаем
магию...</p>
        </div>
    );

    return (
        <div className="max-w-7xl mx-auto px-4 py-12 select-none
animate-in fade-in duration-300">
            <div className="text-center mb-16">
                <h1 className="text-4xl md:text-6xl font-black text-

```

```

slate-800 mb-6 tracking-tight">
    Наши Акции <span className="text-
[#0099ff]"></span>
    </h1>
    <p className="text-lg md:text-xl text-slate-500 max-
w-2xl mx-auto leading-relaxed">
        Выберите лучшее предложение, копируйте промокод
и наслаждайтесь любимыми роллами с выгодой!
    </p>
</div>

    {/* СЕТКА АКЦИЙ */}
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-
cols-3 gap-8 mb-20">
        {promos.map((promo) => (
            <PromoCard
                key={promo.id}
                promo={promo}
                isCopied={copiedId === promo.id}
                onCopy={copy}
            >
                {/* Кнопки админа прокидываем слотом внутрь
карточки */}
                {isAdmin && (
                    <div className="absolute top-4 right-4
z-20 opacity-0 group-hover:opacity-100 transition-opacity">
                        <AdminPromoControls promo={promo} />
                    </div>
                )}
            </PromoCard>
        )})}
    </div>

    {/* БЛОК ПРАВИЛ */}
    <div className="bg-slate-900 rounded-[40px] p-8 md:p-16
text-slate-300 shadow-2xl relative overflow-hidden border border-
slate-800">
        <div className="absolute -top-10 -right-10 opacity-5
text-[200px] pointer-events-none font-black text-white">!</div>

        <div className="flex items-center justify-between
mb-10 relative z-10">
            <h3 className="text-3xl md:text-4xl font-black
text-white tracking-tight">
                <span className="text-
[#0099ff]">ВАЖНЫЕ</span> ПРАВИЛА
            </h3>

            {isAdmin && (
                <button
                    onClick={handleEditRules}

```

```

        className="flex items-center gap-2 bg-
white/10 hover:bg-[#0099ff] px-6 py-3 rounded-2xl text-white font-
bold transition-all active:scale-95 border border-white/10"
        >
            <span className="hidden
sm:inline">Редактировать</span>
            </button>
        </div>

        <ul className="grid grid-cols-1 md:grid-cols-2 gap-8
relative z-10">
            {rules.length > 0 ? rules.map((rule, idx) => (
                <li key={idx} className="flex items-start
gap-5 group">
                    <span className="flex-shrink-0 w-10 h-10
rounded-xl bg-[#0099ff]/20 text-[#0099ff] flex items-center justify-
center font-black text-lg group-hover:bg-[#0099ff] group-hover:text-
white transition-colors">
                        {idx + 1}
                    </span>
                    <p className="text-lg leading-relaxed
text-slate-300 group-hover:text-white transition-colors font-
medium">
                        {rule}
                    </p>
                </li>
            )) : (
                <p className="text-slate-500 italic font-
bold">Правила еще не установлены...</p>
            )}
        </ul>
    </div>
</div>
);
};

```

Файл usePromos.js

```

import { useState, useEffect } from 'react';
import { apiClient } from '@shared/api/apiClient';

export const usePromos = (isAdmin) => {
    const [promos, setPromos] = useState([]);
    const [rules, setRules] = useState([]);
    const [isLoading, setIsLoading] = useState(true);

    useEffect(() => {
        const loadData = async () => {
            setIsLoading(true);
            try {
                const promosEndpoint = isAdmin ? '/promos' :
'/promos/active';

```

					ПГУ 1.090301.44.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		121

```

        const [promosData, rulesData] = await Promise.all([
            apiClient.get(promosEndpoint),
            apiClient.get('/settings/Rules')
        ]);

        if (Array.isArray(promosData))
setPromos(promosData);

        if (rulesData?.value) {
            setRules(rulesData.value.split('\n').filter(r =>
r.trim() !== ''));
        }
    } catch (err) {
        console.error("Ошибка загрузки данных страницы
акций:", err);
    } finally {
        setIsLoading(false);
    }
};

    loadData();
}, [isAdmin]);

const handleEditRules = () => {
    window.dispatchEvent(new CustomEvent('admin-modal-open', {
        detail: { type: 'rules', data: rules }
    }));
};

return { promos, rules, isLoading, handleEditRules };
};

```

Файл ProfilePage.jsx

```

import { useSelector, useDispatch } from 'react-redux';
import { Navigate, Link } from 'react-router-dom';
import { logoutUser } from '@/entities/user/model/userSlice';
import { useUserOrders } from '../lib/useUserOrders';
import { AddressManager } from '@/features/manage-addresses';
import { ClientOrderCard } from
'@/entities/order/ui/ClientOrderCard';
import { ArchivedOrderCard } from
'@/entities/order/ui/ArchivedOrderCard';

export const ProfilePage = () => {
    const dispatch = useDispatch();
    const currentUser = useSelector((state) =>
state.user?.currentUser);
    const { activeOrders, archivedOrders, isLoading } =
useUserOrders(currentUser?.phone);

    if (!currentUser) return <Navigate to="/" replace />;

```

```

    return (
      <div className="max-w-6xl mx-auto p-4 flex flex-col lg:flex-
row gap-10 mt-8 select-none">

        { /* ЛЕВАЯ ЧАСТЬ: ДАННЫЕ ЮЗЕРА */ }
        <div className="w-full lg:w-[350px] flex flex-col gap-
6">
          <div className="bg-white p-8 rounded-[32px] shadow-
sm border border-slate-100 text-center relative overflow-hidden">
            <div className="absolute top-0 left-0 right-0 h-
4 bg-gradient-to-r from-[#0099ff] to-[#00d4ff]"></div>
            <div className="w-24 h-24 bg-blue-50 border-4
border-white shadow-md rounded-full mx-auto flex items-center
justify-center text-4xl mb-4 mt-2">

              </div>
              <h2 className="text-2xl font-black text-slate-
800 mb-1">{currentUser.name}</h2>
              <p className="text-[#0099ff] font-bold mb-
6">{currentUser.phone}</p>

              <div className="text-left bg-slate-50 p-4
rounded-2xl flex flex-col gap-3">
                <div>
                  <span className="text-[10px] font-black
text-slate-400 uppercase tracking-widest block">Логин</span>
                  <span className="font-bold text-slate-
700">{currentUser.login}</span>
                </div>
                <div>
                  <span className="text-[10px] font-black
text-slate-400 uppercase tracking-widest block">Дата рождения</span>
                  {currentUser.birthday ? (
                    <span className="font-bold text-
slate-700">
                      {new
Date(currentUser.birthday).toLocaleDateString('ru-RU')}
                    </span>
                  ) : (
                    <span className="font-bold text-
amber-500 text-sm">Не указана</span>
                  ) }
                </div>
              </div>

              { /* ФИЧА УПРАВЛЕНИЯ АДРЕСАМИ */ }
              <AddressManager currentUser={currentUser} />

              <button
                onClick={() => dispatch(logoutUser())}

```



```

        className="w-full mt-6 bg-slate-100
hover:bg-red-50 text-red-500 font-bold py-3 rounded-xl transition-
colors"
    >
        Выйти из аккаунта
    </button>
</div>
</div>

{/* ПРАВАЯ ЧАСТЬ: ЗАКАЗЫ */}
<div className="flex-1 flex flex-col gap-8">
    <div>
        <h3 className="text-2xl font-black text-slate-
800 uppercase tracking-tight mb-6 flex items-center gap-3">
            АКТИВНЫЕ ЗАКАЗЫ
            {activeOrders.length > 0 && <span
className="bg-red-500 text-white text-sm px-3 py-1 rounded-full
animate-pulse">{activeOrders.length}</span>}
        </h3>

        {isLoading ? (
            <div className="text-slate-400 font-
bold">Ищем ваши заказы...</div>
        ) : activeOrders.length === 0 ? (
            <div className="bg-white p-8 rounded-[32px]
border border-slate-100 shadow-sm text-center">
                <span className="text-4xl block mb-
2"></span>

                <p className="text-slate-500 font-
medium">У вас нет активных заказов.</p>
                <Link to="/" className="text-[#0099ff]
font-bold mt-2 inline-block hover:underline">Перейти в меню</Link>
            </div>
        ) : (
            <div className="flex flex-col gap-6">
                {activeOrders.map(order => (
                    <ClientOrderCard key={order.id}
order={order} />
                ))}
            </div>
        )}
    </div>

    {archivedOrders.length > 0 && (
        <div className="mt-8">
            <h3 className="text-xl font-black text-
slate-400 uppercase tracking-tight mb-4">История заказов</h3>
            <div className="flex flex-col gap-4">
                {archivedOrders.map(order => (
                    <ArchivedOrderCard key={order.id}
order={order} />
                ))}
            </div>
        </div>
    )}

```

```

        ))}
      </div>
    </div>
  )}
</div>

</div>
);
};

```

Файл useUserOrders.js

```

import { useState, useEffect } from 'react';
import { apiClient } from '@shared/api/apiClient';

export const useUserOrders = (phone) => {
  const [activeOrders, setActiveOrders] = useState([]);
  const [archivedOrders, setArchivedOrders] = useState([]);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    if (!phone) {
      setIsLoading(false);
      return;
    }

    const fetchOrders = async () => {
      setIsLoading(true);
      try {
        const data = await apiClient.get('/orders');

        if (Array.isArray(data)) {
          const userOrders = data.filter(o => o.phone ===
phone);
          setActiveOrders(userOrders.filter(o => o.status
!== 'Completed' && o.status !== 'Cancelled'));
          setArchivedOrders(userOrders.filter(o =>
o.status === 'Completed' || o.status === 'Cancelled'));
        }
      } catch (err) {
        console.error("Ошибка загрузки истории заказов:",
err);
      } finally {
        setIsLoading(false);
      }
    };

    fetchOrders();
  }, [phone]);

  return { activeOrders, archivedOrders, isLoading };
};

```

Файл CartItemsList.jsx

```
import { AddToCartButton } from '@features/cart/add-to-cart';

export const CartItemsList = ({ items, showStub, onClear,
onSelectProduct }) => {
  return (
    <div className="flex-1 bg-white p-4 sm:p-6 md:p-8 rounded-
[32px] shadow-sm border border-slate-100 h-fit">
      <div className="flex justify-between items-center mb-6
sm:mb-8">
        <h2 className="text-xl sm:text-2xl font-extrabold
text-slate-800">Ваш заказ</h2>
        <button onClick={onClear} className="text-sm
font-semibold text-slate-400 hover:text-red-500 transition-colors">
          ОЧИСТИТЬ ВСЁ
        </button>
      </div>

      <div className="flex flex-col gap-4 sm:gap-6">
        {showStub ? (
          <div className="flex flex-col items-center text-
center py-10">
            <div className="text-green-600 font-
extrabold text-2xl mb-2"> Тестовый заказ успешно оформлен!</div>
            <p className="text-slate-500">Администратор
уже видит его в панели управления.</p>
          </div>
        ) : (
          items.map((item, idx) => (
            <div key={item.id + '-' + idx}
className="flex flex-col sm:flex-row sm:items-center justify-between
border-b border-slate-100 pb-4 sm:pb-6 gap-3 sm:gap-4 animate-in
fade-in">
              <div className="flex items-center gap-3
sm:gap-4 flex-1 cursor-pointer hover:opacity-80" onClick={() =>
item.price > 0 && onSelectProduct(item)}>
                <div className="w-14 h-14 sm:w-16
sm:h-16 flex-shrink-0 rounded-full bg-slate-50 p-1 border border-
slate-200">
                  <img src={item.imageUrl}
alt={item.name} className="w-full h-full object-cover rounded-full"
/>
                </div>
                <div>
                  <span className="font-bold text-
slate-700 text-base sm:text-lg leading-tight
block">{item.name}</span>
                  {item.price === 0 && <span
className="text-xs bg-red-500 text-white font-black px-2 py-0.5
```

```

rounded-md uppercase tracking-wider mt-1 inline-block animate-
pulse">Подарок </span>}
        </div>
      </div>
      <div className="flex items-center
justify-between sm:justify-end gap-4 w-full sm:w-auto">
        <span className="font-black text-
[#0099ff] text-lg whitespace-nowrap">{item.price === 0 ? '0 ₺' :
`$${item.price * item.count} ₺`</span>
        {item.price > 0 ? <AddToCartButton
product={item} /> : <span className="text-slate-400 font-bold text-
sm px-4 py-2 bg-slate-50 rounded-xl border border-
dashed">Добавлено</span>}
      </div>
    </div>
  ))
  })
</div>
</div>
);
};

```

Файл CheckoutSummary.jsx

```

import { MapPin } from 'lucide-react';
import { AddressManager } from '@features/manage-addresses';
import { DeliveryTimeSelector } from '@features/delivery-time';
import { ApplyPromo } from '@features/apply-promo';

export const CheckoutSummary = ({
  showStub, currentUser,
  onOpenAuth, isSending, onSubmit,
  deliveryProps, promoProps,
  addressProps, priceProps, paymentProps
}) => {
  return (
    <div className="w-full lg:w-[420px] bg-white p-6 md:p-8
rounded-[32px] shadow-sm border border-slate-100 h-fit flex flex-col
gap-6">
      <h3 className="text-xl font-black text-slate-800
uppercase tracking-wide">Оформление</h3>

      {showStub ? (
        <div className="flex flex-col gap-3 mt-2 animate-in
zoom-in duration-300">
          <p className="text-sm font-bold text-slate-600
text-center mb-2">Настоящий заказ можно сделать тут:</p>
          <a href="https://freeroll13.ru/" target="_blank"
rel="noreferrer" className="w-full flex items-center justify-center
gap-2 bg-gradient-to-r from-[#0099ff] to-[#00d4ff] text-white font-
bold py-3 px-4 rounded-xl shadow-md transition-all active:scale-95">
Официальный сайт</a>
          <a href="https://vk.com/freeroll13"

```

```

target="_blank" rel="noreferrer" className="w-full flex items-center
justify-center gap-2 bg-[#4680C2] text-white font-bold py-3 px-4
rounded-xl shadow-md transition-all active:scale-95"> Группа
ВКонтакте</a>

        <button onClick={() =>
addressProps.setShowStub(false)} className="mt-4 text-sm font-
semibold text-slate-400 hover:text-slate-600 text-center w-
full">Вернуться к корзине</button>
        </div>
        ) : currentUser ? (
        <>
            <div className="bg-blue-50/40 p-5 rounded-2xl
border border-blue-100 flex flex-col gap-3">
                <div className="flex items-center gap-2
text-slate-700 font-extrabold text-sm uppercase tracking-wide">
                    <MapPin size={16} className="text-
[#0099ff]" />
                    <span>Куда везти?</span>
                </div>
                <AddressManager currentUser={currentUser}
selectedAddress={addressProps.selectedAddress}
onSelect={addressProps.setSelectedAddress} hideTitle={true} />
                </div>

                <DeliveryTimeSelector {...deliveryProps} />
                <ApplyPromo {...promoProps} />
            </>
        ) : (
            <div className="text-center bg-slate-50 p-6 rounded-
2xl border border-slate-100">
                <p className="font-bold mb-4">Войдите, чтобы
оформить заказ</p>
                <button onClick={onOpenAuth} className="w-full
bg-slate-800 text-white font-bold py-3 rounded-xl">Войти</button>
            </div>
        )
    }

    { /* Блок суммы */ }
    <div className="bg-slate-50/40 p-4 rounded-2xl border
border-slate-100 flex flex-col gap-3">
        <div className="flex justify-between items-center
text-sm text-slate-400 font-bold">
            <span>Базовая сумма:</span>
            <span className="line-through font-
mono">{priceProps.baseTotalPrice} ₽</span>
        </div>
        {priceProps.appliedPromo && <div className="text-xs
font-bold text-green-600 bg-green-50 px-3 py-1.5 rounded-lg border
border-green-100">{priceProps.appliedPromo.name}</div>}
        <div className="flex justify-between items-center
pt-3 border-t border-slate-100">

```



```

        <span className="text-slate-800 font-black text-
lg">К ОПЛАТЕ:</span>
        <span className="text-3xl font-black text-
[#0099ff] font-mono">{priceProps.finalPrice} Р</span>
      </div>
    </div>

    {/* Кнопки подтверждения */}
    {!showStub && currentUser && (
      <div className="flex flex-col gap-5">
        <div className="flex gap-2">
          <button
            onClick={() =>
paymentProps.setPaymentMethod('cash')}
            className={`flex-1 py-3 rounded-xl font-
bold text-sm transition-all border ${paymentProps.paymentMethod ===
'cash' ? 'bg-slate-800 text-white shadow-md' : 'bg-white text-slate-
600'}`}
          >
            Наличными
          </button>
          <button
            onClick={() =>
paymentProps.setPaymentMethod('card')}
            // ВОТ ТУТ ДОБАВИЛИ paymentProps.
            className={`flex-1 py-3 rounded-xl font-
bold text-sm transition-all border ${paymentProps.paymentMethod ===
'card' ? 'bg-slate-800 text-white shadow-md' : 'bg-white text-slate-
600'}`}
          >
            Картой
          </button>
        </div>
        <button onClick={onSubmit} disabled={isSending}
className="w-full bg-[#0099ff] text-white font-black text-lg py-4
rounded-2xl shadow-lg transition-all active:scale-95
disabled:opacity-50">
          {isSending ? 'Отправка...' : 'Подтвердить
заказ'}
        </button>
      </div>
    )}
  </div>
);
};

```

Файл CartPage.jsx

```

import { useState, useEffect } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { clearCart } from '@/entities/cart/model/cartSlice';
import { ProductModal } from '@/entities/product';
import { AuthModal } from '@/features/auth';

```

					ПГУ 1.090301.44.001 ПЗ	Лист
						129
Изм.	Лист	№ докум.	Подпись	Дата		

```

import { useCartCheckout } from '../lib/useCartCheckout';
import { CartItemsList } from './CartItemsList';
import { CheckoutSummary } from './CheckoutSummary';

export const CartPage = () => {
  const dispatch = useDispatch();
  const { items, totalPrice: baseTotalPrice } =
useSelector((state) => state.cart);
  const currentUser = useSelector((state) =>
state.user?.currentUser);

  const [paymentMethod, setPaymentMethod] = useState('card');
  const [selectedProduct, setSelectedProduct] = useState(null);
  const [isAuthModalOpen, setIsAuthModalOpen] = useState(false);
  const [deliveryType, setDeliveryType] = useState('asap');
  const [deliveryTimeSlot, setDeliveryTimeSlot] = useState('10:00
- 11:00');
  const [selectedAddress, setSelectedAddress] = useState('');

  const displayAddresses = currentUser?.addresses?.length > 0 ?
currentUser.addresses : (currentUser?.address ? [{ id: 0, value:
currentUser.address }] : []);

  useEffect(() => {
    if (currentUser && displayAddresses.length > 0 &&
!selectedAddress) {
      setSelectedAddress(displayAddresses[0].value);
    }
  }, [currentUser, displayAddresses]);

  const {
    isSending, showStub, setShowStub, appliedPromo,
setAppliedPromo,
    promoError, validateAndApplyPromo, finalPrice, finalItems,
handleCheckout
  } = useCartCheckout(items, baseTotalPrice, currentUser);

  const submitOrder = () => {
    if (!selectedAddress) return alert("Пожалуйста, выберите
адрес доставки!");
    handleCheckout({
      customerName: currentUser.name, phone:
currentUser.phone, address: selectedAddress,
      paymentMethod, totalPrice: finalPrice,
      deliveryNotice: deliveryType === 'scheduled' ? ` Ко
времени: ${deliveryTimeSlot}` : ' Как можно быстрее',
      items: finalItems.map(i => ({ productId: i.id, count:
i.count, price: i.price })))
    });
  };
};

```

					ПГУ 1.090301.44.001 ПЗ	Лист
						130
Изм.	Лист	№ докум.	Подпись	Дата		

```

    if (items.length === 0 && !showStub) {
      return (
        <div className="text-center py-32 flex flex-col items-center">
          <span className="text-6xl mb-4">🛒</span>
          <h2 className="text-2xl font-bold text-slate-700">Ваша корзина пуста</h2>
        </div>
      );
    }

    return (
      <div className="max-w-6xl mx-auto p-4 flex flex-col lg:flex-row gap-10 mt-8 select-none">
        <CartItemListComponent
          items={finalItems} showStub={showStub}
          onSelectProduct={setSelectedProduct}
          onClear={() => { dispatch(clearCart());
setAppliedPromo(null); }}
        />

        <CheckoutSummary
          showStub={showStub} currentUser={currentUser}
          isSending={isSending}
          onOpenAuth={() => setIsAuthModalOpen(true)}
          onSubmit={submitOrder}
          addressProps={{ selectedAddress, setSelectedAddress,
setShowStub }}
          deliveryProps={{ deliveryType, setDeliveryType,
deliveryTimeSlot, setDeliveryTimeSlot }}
          promoProps={{ onApply: validateAndApplyPromo,
appliedPromo, promoError }}
          priceProps={{ baseTotalPrice, finalPrice,
appliedPromo }}
          paymentProps={{ paymentMethod, setPaymentMethod }}
        />

        <ProductModal product={selectedProduct} onClose={() =>
setSelectedProduct(null)} />
        <AuthModal isOpen={isAuthModalOpen} onClose={() =>
setIsAuthModalOpen(false)} />
      </div>
    );
  };
};

```

Файл useCartCheckout.jsx

```

import { useState, useEffect } from 'react';
import { useDispatch } from 'react-redux';
import { clearCart } from '@/entities/cart/model/cartSlice';
import { apiClient } from '@shared/api/apiClient';

```

```

export const useCartCheckout = (items, baseTotalPrice, currentUser)
=> {
  const dispatch = useDispatch();
  const [isSending, setIsSending] = useState(false);
  const [showStub, setShowStub] = useState(false);

  const [activePromos, setActivePromos] = useState([]);
  const [appliedPromo, setAppliedPromo] = useState(null);
  const [promoError, setPromoError] = useState('');

  useEffect(() => {
    apiClient.get('/promos/active')
      .then(data => setActivePromos(data))
      .catch(err => console.error("Ошибка загрузки акций:",
err));
  }, []);

  useEffect(() => {
    if (currentUser?.birthday) {
      const today = new Date();
      const birthDate = new Date(currentUser.birthday);
      if (today.getDate() === birthDate.getDate() &&
today.getMonth() === birthDate.getMonth()) {
        setAppliedPromo({
          type: 'discount', value: 0.15, name: ' Скидка
15% в День Рождения!', excludeAddons: true
        });
        return;
      }
    }
    if (appliedPromo?.code)
validateAndApplyPromo(appliedPromo.code, true);
  }, [baseTotalPrice, currentUser, activePromos]);

  const validateAndApplyPromo = async (codeToValidate,
isAutoReplay = false) => {
    const code = codeToValidate.trim().toUpperCase();
    setPromoError('');
    if (!code) return;

    const promo = activePromos.find(p => p.code?.toUpperCase()
=== code);
    if (!promo) {
      if (!isAutoReplay) setPromoError('Такого промокода
нет!');
      if (isAutoReplay) setAppliedPromo(null);
      return;
    }
  }

```

					ПГУ 1.090301.44.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		132

```

        const now = new Date();
        if (promo.minOrderSum > 0 && baseTotalPrice <
promo.minOrderSum) return setPromoError(`Сумма от
${promo.minOrderSum} ₸!`);
        if (promo.maxOrderSum > 0 && baseTotalPrice >
promo.maxOrderSum) return setPromoError(`Сумма до
${promo.maxOrderSum} ₸!`);
        if (promo.validTo && now > new Date(promo.validTo)) return
setPromoError('Срок истёк!');

        if (promo.type === 'discount') {
            setAppliedPromo({
                code, type: 'discount', value: promo.discountValue,
name: ` ${promo.title}`, excludeAddons: promo.excludeAddons
            });
        } else if (promo.type === 'gift' && promo.giftProductId) {
            try {
                const giftProduct = await
apiClient.get(`/products/${promo.giftProductId}`);
                setAppliedPromo({
                    code, type: 'gift', name: ` ПОДАРОК:
${giftProduct.name}!`,
                    giftItem: { ...giftProduct, count: 1, price: 0 }
                });
            } catch (err) {
                if (!isAutoReplay) setPromoError('Ошибка загрузки
подарка. ');
            }
        }
    };

    let finalPrice = baseTotalPrice;
    let finalItems = [...items];

    if (appliedPromo) {
        if (appliedPromo.type === 'discount') {
            if (appliedPromo.excludeAddons) {
                const eligibleSum = items.reduce((sum, item) =>
item.categoryId !== 11 ? sum + (item.price * item.count) : sum, 0);
                const addonsSum = items.reduce((sum, item) =>
item.categoryId === 11 ? sum + (item.price * item.count) : sum, 0);
                finalPrice = Math.round(eligibleSum * (1 -
appliedPromo.value) + addonsSum);
            } else {
                finalPrice = Math.round(baseTotalPrice * (1 -
appliedPromo.value));
            }
        } else if (appliedPromo.type === 'gift') {
            finalItems.push(appliedPromo.giftItem);
        }
    }

```



```

    }

    const handleCheckout = async (orderData) => {
      setIsSending(true);
      try {
        await apiClient.post('/orders', orderData);
        dispatch(clearCart());
        setShowStub(true);
      } catch (error) {
        alert(`Ошибка: ${error.message}`);
      } finally {
        setIsSending(false);
      }
    };

    return {
      isSending, showStub, setShowStub,
      appliedPromo, setAppliedPromo, promoError,
      validateAndApplyPromo,
      finalPrice, finalItems, handleCheckout
    };
  };
};

```

Файл ReviewsPage.jsx

```

import { useSelector } from 'react-redux';
import { Trash2 } from 'lucide-react';

import { useReviews } from '../lib/useReviews';
import { ReviewForm } from '@/features/create-review';
import { ReviewModerationPanel } from '@/widgets/review-moderation-panel';
import { ReviewCard } from '@/shared/ui/ReviewCard';

export const ReviewsPage = () => {
  const currentUser = useSelector((state) =>
state.user?.currentUser);
  const isAdmin = currentUser?.isAdmin;

  const {
    reviews,
    moderationReviews,
    isLoading,
    refresh,
    approveReview,
    deleteReview
  } = useReviews(isAdmin);

  return (
    <div className="max-w-4xl mx-auto p-4 mt-8 flex flex-col gap-8 select-none animate-in fade-in duration-300">
      <h1 className="text-3xl font-black text-slate-800

```

```

uppercase tracking-tight">Отзывы клиентов 🗨️</h1>

    {/* ВЕДЖЕТ МОДЕРАЦИИ ДЛЯ АДМИНА */}
    {isAdmin && (
      <ReviewModerationPanel
        moderationReviews={moderationReviews}
        onApprove={approveReview}
        onDelete={deleteReview}
      />
    )}

    {/* ФИЧА ФОРМЫ ОТЗЫВА */}
    <ReviewForm currentUser={currentUser}
onReviewCreated={refresh} />

    {/* ЛЕНТА ОТЗЫВОВ */}
    <div className="flex flex-col gap-6">
      <h3 className="font-extrabold text-slate-400 text-xs
uppercase tracking-widest ml-1">Лента отзывов</h3>

      {isLoading ? (
        <div className="text-center text-slate-400 font-
bold py-6">Загрузка отзывов...</div>
      ) : reviews.length === 0 ? (
        <div className="text-center text-slate-400 py-
6">Отзывов пока нет. Будьте первыми! </div>
      ) : (
        <div className="grid gap-6">
          {reviews.map((rev) => (
            <ReviewCard key={rev.id} {...rev}>
              {/* Используем слот children, чтобы
подкинуть кнопку удаления только админу */}
              {isAdmin && (
                <button
                  onClick={() =>
deleteReview(rev.id)}
                  className="text-xs text-red-
400 hover:text-red-600 font-bold self-start mt-2 flex items-center
gap-1 transition-colors"
                >
                  <Trash2 size={12} /> Удалить
                </button>
              )}
            </ReviewCard>
          )}}
        </div>
      )}
    </div>
  )}
</div>
</div>

```

```
);  
};
```

Файл useReviews.js

```
import { useState, useEffect, useCallback } from 'react';  
import { apiClient } from '@shared/api/apiClient';  
  
export const useReviews = (isAdmin) => {  
  const [reviews, setReviews] = useState([]);  
  const [moderationReviews, setModerationReviews] = useState([]);  
  const [isLoading, setIsLoading] = useState(true);  
  
  const fetchReviews = useCallback(async () => {  
    try {  
      const data = await apiClient.get('/reviews');  
      if (Array.isArray(data)) setReviews(data);  
  
      if (isAdmin) {  
        const modData = await  
apiClient.get('/reviews/moderation');  
        if (Array.isArray(modData))  
setModerationReviews(modData);  
      }  
    } catch (err) {  
      console.error("Ошибка загрузки отзывов:", err);  
    } finally {  
      setIsLoading(false);  
    }  
  }, [isAdmin]);  
  
  useEffect(() => {  
    fetchReviews();  
  }, [fetchReviews]);  
  
  const approveReview = async (id) => {  
    try {  
      await apiClient.patch(`/reviews/${id}/approve`);  
      fetchReviews();  
    } catch (err) {  
      console.error(err);  
    }  
  };  
  
  const deleteReview = async (id) => {  
    if (!window.confirm("Удалить этот отзыв?")) return;  
    try {  
      await apiClient.delete(`/reviews/${id}`);  
      fetchReviews();  
    } catch (err) {  
      console.error(err);  
    }  
  };  
};
```

```

    return { reviews, moderationReviews, isLoading, refresh:
fetchReviews, approveReview, deleteReview };
};

```

Файл AdminOrdersPage.jsx

```

import { useSelector } from 'react-redux';
import { Navigate } from 'react-router-dom';
import { useAdminOrders } from '../lib/useAdminOrders';
import { OrdersAccordion } from '@widgets/orders-
accordion/ui/OrdersAccordion';

export const AdminOrdersPage = () => {
    const currentUser = useSelector((state) =>
state.user?.currentUser);
    const isAdmin = currentUser?.isAdmin;

    const {
        isLoading,
        fetchOrders,
        changeOrderStatus,
        asapOrders,
        scheduledOrders,
        completedOrders
    } = useAdminOrders(isAdmin);

    if (!isAdmin) return <Navigate to="/" replace />;

    return (
        <div className="max-w-6xl mx-auto p-4 select-none animate-in
fade-in duration-300">
            {/* ШАПКА */}
            <div className="flex justify-between items-center mb-10
mt-6">
                <h1 className="text-3xl font-black text-slate-800
uppercase tracking-tight">Управление заказами </h1>
                <button
                    onClick={fetchOrders}
                    className="bg-slate-100 hover:bg-slate-200 text-
slate-600 px-5 py-2.5 rounded-xl font-bold transition-all
active:scale-95 text-sm"
                >
                    Обновить данные
                </button>
            </div>

            {/* СПИСОК ШТОРОК */}
            {isLoading ? (
                <div className="text-center py-20 text-slate-400
font-bold">Загрузка списка заказов...</div>
            ) : (!asapOrders.length && !scheduledOrders.length &&
!completedOrders.length) ? (

```

```

        <div className="text-center py-20 text-slate-400">Заказов пока нет</div>
      ) : (
        <div className="flex flex-col gap-6">
          <OrdersAccordion
            title=" Активные (По быстрее) "
            orders={asapOrders}
            onChangeStatus={changeOrderStatus}
            defaultOpen={true}
            theme="green"
            emptyMessage="Живая очередь пуста! "
          />

          <OrdersAccordion
            title=" Активные (Ко времени) "
            orders={scheduledOrders}
            onChangeStatus={changeOrderStatus}
            defaultOpen={true}
            theme="amber"
            emptyMessage="Предзаказов пока нет! "
          />

          <OrdersAccordion
            title=" Архив (Выполненные) "
            orders={completedOrders}
            onChangeStatus={changeOrderStatus}
            defaultOpen={false}
            theme="slate"
            emptyMessage="Архив пуст"
          />
        </div>
      )}
    </div>
  );
};

```

Файл useAdminOrders.js

```

import { useState, useEffect, useMemo } from 'react';
import { apiClient } from '@shared/api/apiClient';

export const useAdminOrders = (isAdmin) => {
  const [orders, setOrders] = useState([]);
  const [isLoading, setIsLoading] = useState(true);

  const fetchOrders = async () => {
    setIsLoading(true);
    try {
      const data = await apiClient.get('/orders');
      setOrders(data);
      window.dispatchEvent(new CustomEvent('order-status-
updated'));
    } catch (err) {

```



```

        console.error("Ошибка загрузки заказов:", err);
    } finally {
        setIsLoading(false);
    }
};

useEffect(() => {
    if (isAdmin) fetchOrders();
}, [isAdmin]);

const changeOrderStatus = async (orderId, newStatus) => {
    try {
        await apiClient.patch(`/orders/${orderId}/status`,
newStatus);

        setOrders(prev => prev.map(o => {
            if (o.id === orderId) {
                const updatedOrder = { ...o, status: newStatus
};

                if (newStatus === 'Cooking')
updatedOrder.cookingAt = new Date().toISOString();
                if (newStatus === 'Delivering')
updatedOrder.deliveringAt = new Date().toISOString();
                if (newStatus === 'Completed')
updatedOrder.completedAt = new Date().toISOString();
                return updatedOrder;
            }
            return o;
        }));
        window.dispatchEvent(new CustomEvent('order-status-
updated'));
    } catch (err) {
        alert("Не удалось обновить статус");
    }
};

const { asapOrders, scheduledOrders, completedOrders } =
useMemo(() => {
    const statusWeight = { 'New': 1, 'Cooking': 2, 'Delivering':
3, 'Completed': 4, 'Cancelled': 5 };
    const sortPipeline = (a, b) => {
        if (statusWeight[a.status] !== statusWeight[b.status])
return statusWeight[a.status] - statusWeight[b.status];
        return new Date(a.createdAt) - new Date(b.createdAt);
    };

    const isScheduled = (notice) => notice &&
notice.includes('Ко времени');

    return {
        asapOrders: orders.filter(o => o.status !== 'Completed'

```

```

    && o.status !== 'Cancelled' &&
    !isScheduled(o.deliveryNotice)).sort(sortPipeline),
        scheduledOrders: orders.filter(o => o.status !==
    'Completed' && o.status !== 'Cancelled' &&
    isScheduled(o.deliveryNotice)).sort(sortPipeline),
        completedOrders: orders.filter(o => o.status ===
    'Completed' || o.status === 'Cancelled').sort((a, b) => new
    Date(b.createdAt) - new Date(a.createdAt))
    };
    }, [orders]);

    return { isLoading, fetchOrders, changeOrderStatus, asapOrders,
    scheduledOrders, completedOrders };
};

```

Файл AdminStatsPage.jsx

```

import { useSelector } from 'react-redux';
import { Navigate } from 'react-router-dom';
import { TrendingUp, PackageCheck, Activity, ChefHat, Truck } from
    'lucide-react';

import { useAdminStats } from '../lib/useAdminStats';
import { StatsFilter } from '@/widgets/admin-stats-filter';
import { DashboardCharts } from '@/widgets/dashboard-charts';
import { KpiCard } from '@/shared/ui/KpiCard';

export const AdminStatsPage = () => {
    const currentUser = useSelector((state) =>
    state.user?.currentUser);
    const isAdmin = currentUser?.isAdmin;

    const {
        stats, isLoading,
        startDate, setStartDate, endDate, setEndDate, setPeriod
    } = useAdminStats(isAdmin);

    if (!isAdmin) return <Navigate to="/" replace />;

    return (
        <div className="max-w-7xl mx-auto p-4 sm:p-8 select-none">

            <div className="flex flex-col lg:flex-row justify-
            between items-start lg:items-end mb-8 gap-4">
                <h1 className="text-4xl font-black text-slate-800
                uppercase tracking-tight">
                    Аналитика <span className="text-
                [#0099ff]">Бизнеса</span>
                </h1>

                <StatsFilter
                    startDate={startDate}

```

```

setStartDate={setStartDate}
                endDate={endDate} setEndDate={setEndDate}
                setPeriod={setPeriod}
            />
        </div>

        {isLoading || !stats ? (
            <div className="text-center py-32 text-slate-400
font-bold text-xl">Загружаем данные... </div>
        ) : (
            <>
                {/* РЯД 1: ФИНАНСЫ (Большие карточки) */}
                <div className="grid grid-cols-1 md:grid-cols-3
gap-6 mb-6">

                    <KpiCard
                        title="Выручка"

value={`\${stats.totalRevenue.toLocaleString('ru-RU')} ₽`}
                        icon={TrendingUp} theme="blue"
                    />
                    <KpiCard
                        title="Доставлено"
value={`\${stats.totalOrders} шт.`}
                        icon={PackageCheck} theme="green"
                    />
                    <KpiCard
                        title="Средний чек"
value={`\${stats.totalOrders > 0 ?
Math.round(stats.totalRevenue /
stats.totalOrders).toLocaleString('ru-RU') : 0} ₽`}
                        icon={Activity} theme="amber"
                    />
                </div>

                {/* РЯД 2: СКОРОСТЬ (Малые карточки) */}
                <div className="grid grid-cols-1 md:grid-cols-2
gap-6 mb-8">

                    <KpiCard
                        title="Среднее время на кухне"
value={stats.avgCookingTime > 0 ?
`\${stats.avgCookingTime} мин` : 'Нет данных'}
                        icon={ChefHat} theme="slate" isSmall
                    />
                    <KpiCard
                        title="Среднее время доставки курьером"
value={stats.avgDeliveryTime > 0 ?
`\${stats.avgDeliveryTime} мин` : 'Нет данных'}
                        icon={Truck} theme="slate" isSmall
                    />
                </div>
            </>
        )
    }
}

```

```

        { /* РЯД 3: ГРАФИКИ */ }
        <DashboardCharts stats={stats} />
    </>
    })
</div>
);
};

```

Файл useAdminStats.js

```

import { useState, useEffect } from 'react';
import { apiClient } from '@shared/api/apiClient';
export const useAdminStats = (isAdmin) => {
    const [stats, setStats] = useState(null);
    const [isLoading, setIsLoading] = useState(true);

    const [startDate, setStartDate] = useState(() => {
        const d = new Date(); d.setDate(d.getDate() - 7); return
d.toISOString().split('T')[0];
    });
    const [endDate, setEndDate] = useState(() => new
Date().toISOString().split('T')[0]);

    useEffect(() => {
        if (!isAdmin) return;

        const fetchStats = async () => {
            setIsLoading(true);
            try {
                const data = await
apiClient.get(`/orders/stats?startDate=${startDate}T00:00:00Z&endDat
e=${endDate}T23:59:59Z`);
                setStats(data);
            } catch (err) {
                console.error("Ошибка загрузки статистики:", err);
            } finally {
                setIsLoading(false);
            }
        };
        fetchStats();
    }, [isAdmin, startDate, endDate]);

    const setPeriod = (days) => {
        const end = new Date();
        const start = new Date();
        start.setDate(end.getDate() - days);
        setEndDate(end.toISOString().split('T')[0]);
        setStartDate(start.toISOString().split('T')[0]);
    };
    return { stats, isLoading, startDate, setStartDate, endDate,
setEndDate, setPeriod };
};

```

Приложение Г
(обязательное)

Создание веб-платформы приема и обработки заказов контрагентов

Презентация

					ПГУ 1.090301.44.001 ПЗ	Лист
						143
Изм.	Лист	№ докум.	Подпись	Дата		

Создание веб-платформы приема и обработки заказов контрагентов

Автор работы:

студент гр.22ВВП2 Перкин Павел Олегович

Научный руководитель:

к.п.н., доцент кафедры ВТ Леонова Татьяна Юрьевна

Пензенский государственный университет

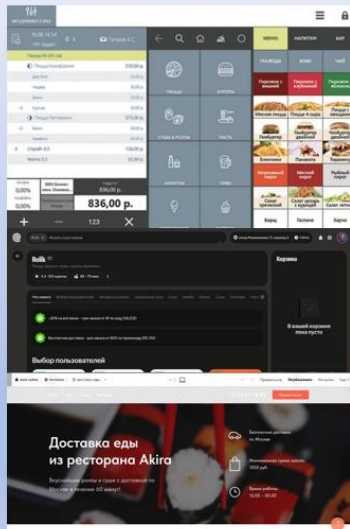
Цель и задачи

Цель работы – проектирование и разработка веб-платформы для автоматизации процессов приема, учета и обработки заказов контрагентов службы доставки «FreeRoll».

Для достижения цели были решены следующие задачи:

1. Проанализированы предметная область и существующие аналоги систем доставки продукции.
2. Обоснованы выбор стека технологий и настройка среды разработки.
3. Спроектирована структура реляционной базы данных платформы.
4. Разработана серверную часть для обработки бизнес-логики и управления данными.
5. Разработано клиентское веб-приложение и интерфейс администратора.
6. Проведено комплексное тестирование разработанного программного обеспечения.

Обзор аналогов



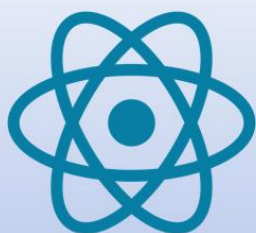
Сравнение собственной веб-платформы с существующими решениями (iiko, агрегаторы, конструкторы) выявило их ключевые ограничения для проекта «FreeRoll»:

Зависимость от сторонних площадок: риск потери клиентской базы и жесткая привязка к алгоритмам ранжирования.

Функциональная избыточность: плата за неиспользуемые модули складского и бухгалтерского учета в тяжелых POS-системах.

Статичность процессов: невозможность внедрения кастомной системы промокодов и подарков без дорогостоящей доработки стороннего ПО.

Выбор стека технологий



– обеспечивает мгновенную перерисовку интерфейса без перезагрузки страницы.



– высокопроизводительный кроссплатформенный фреймворк

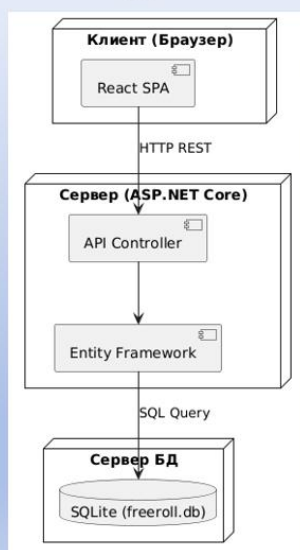


– реляционная СУБД, совместимая с [ASP.NET Core](#).



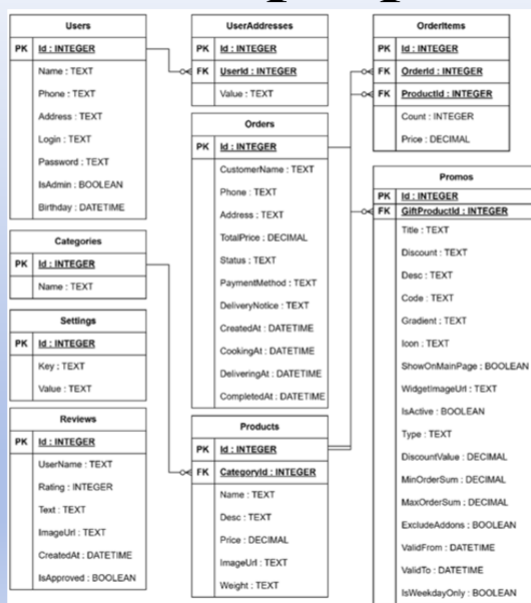
– инструмент эффективного контроля сквозного состояния приложения и предсказуемого потока данных.

Архитектура системы

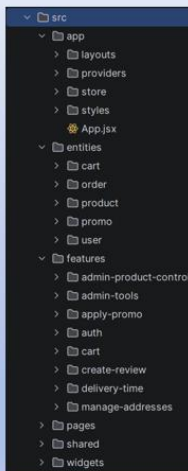


- **Классическая трёхзвенная архитектура:** клиент, сервер приложений и база данных.
- **Асинхронное взаимодействие компонентов** по стандарту REST API через формат JSON.
- **Безопасное управление данными** через ORM-слой и встраиваемую СУБД SQLite.

Реализация серверной части

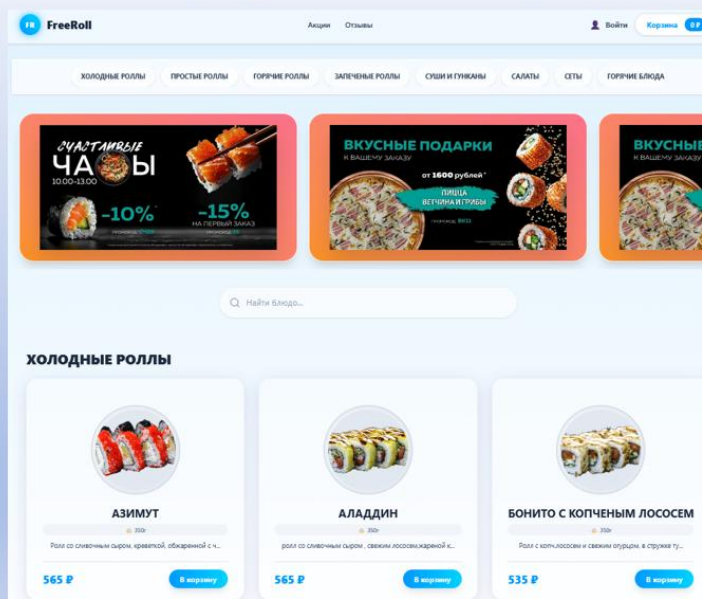


Фронтенд-архитектура: Feature-Sliced Design



App — Инициализация ядра.
↑
Pages — Экраны приложения.
↑
Widgets — Сложные блоки.
↑
Features — Действия пользователя.
↑
Entities — Бизнес-сущности.
↑
Shared — Общие ресурсы.

Интерфейс витрины



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.44.001 ПЗ

Лист

147

Бизнес-логика корзины

Ваш заказ

Очистить всё

АЗИМУТ 565 P 1

АЛАДДИН 565 P 1

ОФОРМЛЕНИЕ

КУДА ВЕЗТИ?

test

+ Добавить новый адрес

ВРЕМЯ ДОСТАВКИ:

На быстрее На время

ЕСТЬ ПРОМОКОД?

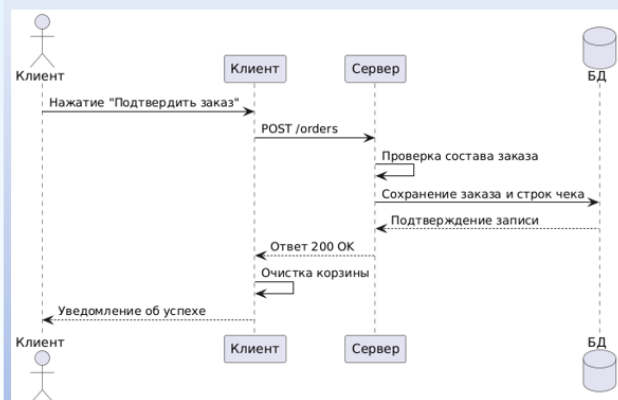
НАПРИМЕР: CUDS Применить

Базовая сумма: 1130 P

КОПЛАТЕ: 1130 P

Наличными Картой

Подтвердить заказ



Реализация панели администратора

УПРАВЛЕНИЕ ЗАКАЗАМИ

Обновить данные

АКТИВНЫЕ (ПО БЫСТРЕЕ) 1

Заказ #3349 **ОФОРМЛЕН**

Оформлен: 22.05.2024, 09:35

КЛИЕНТ И АДРЕС

test

test

КУДА ВЕЗТИ

Как можно быстрее

СОСТАВ ЗАКАЗА

АЛАДДИН

1 шт. × 565 P

ИТОГО: 565 P

ВВЕСТИ В ГОТОВКУ



Экономическая эффективность

Капитальные вложения (рыночная стоимость аналога) – 150 000 руб.

Прямая экономия фонда оплаты труда за счёт автоматизации – 120 000 руб./год.

Косвенный эффект (прирост чистой прибыли от дополнительных клиентов) – 65 000 руб./год.

Суммарный годовой эффект – 185 000 руб.

Срок окупаемости – 0,81 года (около 10 месяцев).

Спасибо за внимание!

					ПГУ 1.090301.44.001 ПЗ	Лист
						149
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Д
(обязательное)

Создание веб-платформы приема и обработки заказов контрагентов

Справка о внедрении результатов ВКР

					ПГУ 1.090301.44.001 ПЗ	Лист
						150
Изм.	Лист	№ докум.	Подпись	Дата		

Справка о внедрении результатов выпускной квалификационной работы

СПРАВКА о внедрении результатов выпускной квалификационной работы Перкина Павла Олеговича для организации рабочего процесса в ИП Кравченко А. В.

Настоящим подтверждается, что выпускная квалификационная работа студента группы 22ВВП2 Перкина Павла Олеговича на тему «Создание веб-платформы приема и обработки заказов контрагентов» была внедрена в производственную деятельность ИП Кравченко А. В.

В ходе реализации проекта Перкиным П. О. была спроектирована и внедрена веб-платформа, обеспечивающая автоматизацию полного цикла обработки заказов. Система, основанная на использовании фреймворка ASP.NET Core и клиентской части на библиотеке React, позволила перевести ручные операции по обработке входящих заявок в цифровой формат, что значительно повысило прозрачность бизнес-процессов и оперативность обслуживания контрагентов.

Внедрение разработанного программного комплекса позволило решить следующие задачи:

1. Оптимизация процесса сбора заказов за счет создания единой базы данных и автоматизированного товарного каталога.
2. Сокращение времени обработки одного заказа с 5–7 минут до 1–2 минут благодаря использованию воронки статусов и административной панели управления.
3. Повышение точности учета финансовых показателей и снижение количества ошибок при формировании отчетности.

Экономический эффект от внедрения платформы составил 185 000 рублей в год. Данная сумма складывается из следующих показателей:

- 120 000 рублей в год — экономия фонда оплаты труда за счет автоматизации процессов приема заказов (сокращение нагрузки на операторов и минимизация ручного ввода данных).
- 65 000 рублей в год — прирост прибыли, достигнутый за счет внедрения аналитического модуля и системы автоматизированного применения промо-акций, что способствовало увеличению среднего чека и росту клиентской лояльности.

Программное обеспечение демонстрирует высокую стабильность, соответствие заявленным техническим требованиям и может быть рекомендовано к дальнейшей эксплуатации и масштабированию.

Индивидуальный предприниматель
Кравченко Андрей Владимирович
ОГРНИП 318132600028003
ИНН 132709614944



A handwritten signature in blue ink, appearing to be 'А.В. Кравченко', written over a faint grid background.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.44.001 ПЗ

Лист

151