МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"

ФАКУЛЬТЕТ Интеллектуальных систем и программирования
КАФЕДРА «Программная инженерия» им. Л.П. Фельдмана
НАПРАВЛЕНИЕ 09.03.04 Программная инженерия
ПРОФИЛЬ Инженерия программного обеспечения

Допустить к защите

Заведующий кафедрой ПИ
_____ Зори С.А.
(Подпись) (ФИО)
«___» _____ 2023г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту бакалавра

НА ТЕМУ: «Система заказов и доставки готовой продукции ресторана»

СПЕЦЧАСТЬ: «Разработка интерфейса, программных модулей, базы данных системы с использованием технологий и языков программирования C#, ASP.NET, Neo4j, Yandex.Облако, HTML5, CSS3, Bootstrap, JavaScript, React, Kotlin»

АВТОР ДИПЛОМНОГО ПРОЕКТА БАКАЛАВРА

Жильцов Владимир Александрович
(Фамилия, имя, отчество) (Подпись)

РУКОВОДИТЕЛЬ ДИПЛОМНОГО ПРОЕКТА БАКАЛАВРА

Ст. преп. каф. ПИ Морозова Ольга Васильевна
(Должность, ученая степень, ученое звание, фамилия, имя, отчество) (Подпись)

КОНСУЛЬТАНТЫ:

Охрана труда доцент каф. «Охрана труда и аэрология», к.х.н., доцент, Бутузов Г.Н.

Вопросы БЖД и ГО ст. препод. каф. ПИ им. Л.П. Фельдмана Морозова О.В.
(Наименование раздела (подраздела), должность, ученая степень, звание, ФИО, подпись)

Нормоконтроль ст. препод. каф. ПИ им. Л.П. Фельдмана Коломойцева И.А.

Донецк – 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"

ФАКУЛЬТЕТ Интеллектуальных систем и программирования
КАФЕДРА «Программная инженерия» им. Л.П. Фельдмана
НАПРАВЛЕНИЕ 09.03.04 Программная инженерия
ПРОФИЛЬ Инженерия программного обеспечения

УТВЕРЖДАЮ:
Заведующий кафедрой ПИ
_____ Зори С.А.
(Подпись) (ФИО)
« 20 » 03 2023г.

ЗАДАНИЕ

на дипломный проект бакалавра

Студенту Жильцову Владимиру Александровичу
(Фамилия, имя, отчество)

1.Тема дипломного проекта бакалавра: «Система заказов и доставки готовой продукции ресторана». Спецчасть: «Разработка интерфейса, программных модулей, базы данных системы с использованием технологий и языков программирования C#, ASP.NET, Neo4j, Yandex.Облако, HTML5, CSS3, Bootstrap, JavaScript, React, Kotlin»
(в случае выполнения комплексного проекта (работы) в скобках указывается «комплексный (ая)»)

утверждена приказом от « 09 » 03 2023г. № 315-14

2. Исходные данные к дипломному проекту результаты НИРС и преддипломной практики

3. Содержание пояснительной записки к дипломному проекту бакалавра: введение, системный анализ предметной области, анализ требований, архитектура программного обеспечения, проектирование базы данных, описание программ, описание интерфейсов, тестирование, охрана труда, вопросы бжд и гражданской обороны, заключение, перечень ссылок

4. Перечень графического материала: схема графовой базы данных, диаграмма классов модуля работы с базой данных, диаграмма прецедентов сотрудника кухни, курьера, авторизованного клиента и анонимного клиента, диаграмма прецедентов администратора, диаграмма состояний,карта навигации между страницами приложений

5. Данные о консультантах, с указанием разделов пояснительной записки

Раздел	Консультант	Задание выдал		Задание принял	
		подпись	дата	подпись	дата
Охрана труда, вопросы БЖД и гражданской обороны	Бутузов Г.Н.		20.03.23		
	Морозова О.В.		20.03.23		
Нормоконтроль	Коломойцева И.А.		20.03.23		

6. Срок сдачи студентом законченного дипломного проекта бакалавра _____

7. Дата выдачи задания на дипломный проект бакалавра _____

Руководитель Морозова Ольга Васильевна _____
(Фамилия, имя, отчество) (Подпись)

Задание принял к исполнению _____ 2023г. _____
(Дата) (Подпись)

КАЛЕНДАРНЫЙ ПЛАН

№ п/п	Название этапов дипломного проекта бакалавра	Срок выполнения этапов работы	Отметка о выполнении
1	Проведение системного анализа предметной области и общая постановка задачи, разработка требований к программной системе	20.04.2023	
2	Проектирование архитектуры программной системы	22.04.2023	
3	Проектирование базовых структур програмы	23.04.2023	
4	Проектирование программной системы	25.04.2023	
5	Реализация программной системы	26.04.2023	
6	Тестирование программного продукта	30.04.2023	
7	Написание основной части пояснительной записки	02.05.2023	
8	Написание спец. части пояснительной записки	13.05.2023	
9	Написание приложений для пояснительной записки	17.05.2023	

Студент _____

(подпись)

Руководитель_____

(подпись)

№	Форматы	Обозначение	Наименование	Дополнительные сведения
1				
2			Текстовые документы	
3				
4	А4	09.03.04.2023.18\6322.00.00 ПЗ	Пояснительная записка	
5				
6			Графические документы	
7				
8	А4	09.03.04.2023.18\6322.00.01 Д1	Схема графовой базы данных	
9	А4	09.03.04.2023.18\6322.00.02 Д2	Диаграмма классов модуля работы с базой данных	
10	А4	09.03.04.2023.18\6322.00.03 Д3	Диаграмма прецедентов сотрудника кухни, курьера, авторизованного клиента и анонимного клиента	
11	А4	09.03.04.2023.18\6322.00.04 Д4	Диаграмма прецедентов администратора	
12	А4	09.03.04.2023.18\6322.00.05 Д5	Диаграмма состояний	
13	А4	09.03.04.2023.18\6322.00.06 Д6	Карта навигации между страницами приложений	
14	А4			
15	А4			
16	А4			
17	А4			

					09.03.04.2023.18\6322.00.00 ВР		
Изм.	Лист	№ документа	Подпись	Дата			
Студент		Жильцов В.А.			Система заказов и доставки готовой продукции ресторана. Ведомость дипломного проекта бакалавра	Лит	Лис	Листо
Рук.		Морозова О.В.				У	1	1
Консульт.						ДонНТУ, ФИСП, кафедра ПИ, гр. ПИ-18а		
Н. контр.		Коломойцева И.А.						

РЕФЕРАТ

Пояснительная записка к дипломному проекту бакалавра: 119 страниц, 74 рисунка, 20 таблиц, 10 источников, 3 приложения.

Объект исследования – разработка системы заказов и доставки готовой продукции ресторана с использованием технологий и языком программирования: C#, ASP.NET Core, Neo4j, HTML5, CSS3, Bootstrap, JavaScript, React, Next.js, Kotlin.

Цель – изучить принципы разработки интернет-сайтов, а также мобильных приложений, способных работать с помощью универсального API, получить теоретические знания и практические навыки в разработке динамических сайтов на C#, проанализировать существующие аналоги разрабатываемой системы, спроектировать и разработать сайт и мобильное приложение для сотрудников и клиентов ресторана.

САЙТ, C#, ASP.NET Core, Neo4j, HTML5, CSS3, Bootstrap, JavaScript, React, Next.js, Kotlin, ГРАФОВАЯ БАЗА ДАННЫХ, Android

					09.03.04.2023.18\6322.00.00 ПЗ			
Изм.	Лист	№ документа	Подпись	Дата				
Студент		Жильцов В.А.			Система заказов и доставки готовой продукции ресторана. Пояснительная записка.	Лит	Ли	Листо
Рук.		Морозова О.В.				У	1	1
Н. контр.		Коломойцева И.А.				ДонНТУ, ФИСП, кафедра ПИ, гр. ПИ-18а		

ABSTRACT

The object of research is the development of a system for ordering and delivering finished restaurant products using technologies and a programming language: C#, ASP.NET Core, Neo4j, HTML5, CSS3, Bootstrap, JavaScript, React, Next.js, Kotlin.

The goal is to study the principles of developing Internet sites, as well as mobile applications that can work using a universal API, gain theoretical knowledge and practical skills in developing dynamic sites in C #, analyze existing analogues of the system being developed, design and develop a website and a mobile application for employees and restaurant customers.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время успешное функционирование различных фирм, организаций и предприятий просто невозможно без развитой информационной системы, которая позволяет автоматизировать сбор и обработку данных. Одним из самых удобных способов взаимодействия с пользователями является веб-сайт, который должен соответствовать современным тенденциям, а именно быть адаптивным, быстрым и приятным.

Прежде чем начать разработку, необходимо изучить предметную область фирмы, для которой создается продукт, это может помочь при выборе используемого инструментария и паттернов проектирования. Под предметной областью принято понимать некоторую область человеческой деятельности или область реального мира, подлежащих изучению для организации управления и автоматизации, например, предприятие, интернет-магазин, сервис доставки продуктов питания и т.д.

В условиях интенсивной цифровизации экономики и появления новых трендов потребительских предпочтений на российском рынках все большую популярность приобретают новые формы онлайн-торговли продуктами питания и готовой едой, которые в рамках указанных трендов стали практически неотделимы друг от друга.

Пандемия коронавируса резко ускорила процесс перехода к формату онлайн-торговли продуктами питания (ОТПП). Согласно исследованию, проведенному Аналитическим центром НАФИ в апреле 2020 года, большинство российских интернет-пользователей (67%) за время самоизоляции совершали покупки онлайн, а каждый четвертый (26%) заказывал доставку продуктов питания на дом. Начали пользоваться услугами доставки продуктов питания во время самоизоляции 13% россиян, столько же (13%) указали, что пользовались услугой доставки продуктов и ранее [1]. По оценке M.A. Research, в 2020 году российский объем ОТПП вырос до 174 млрд рублей, что составило 1% от всего рынка продовольственного ритейла [2]. В

2021—2025 годах совокупный среднегодовой темп роста ОТПП будет иметь долю, по разным оценкам, от 33 до 40%. В 2022 году сегмент ОТПП вырастет до 415—445 млрд рублей при выполнении заявленных планов компаний, а его объем в обороте розничной торговли продуктами питания достигнет 2,2—2,4% [3]. Действительно, спрос на интернет-услугу доставки продуктов питания в целом по России продолжает расти, так, за 6 месяцев 2021 года он вырос на 73% по сравнению с тем же периодом 2020 года [4].

Данный тренд особенно характерен для крупных городов: например, в Санкт-Петербурге за первые полгода после введения пандемийных ограничений количество онлайн-заказов продуктов на дом увеличилось в 20 раз [5].

Учитывая процесс трансформации и увеличения объема рынка ОТПП можно говорить о том, что разработка интернет-магазинов, а также сервисов, связанных с ними, для компаний является актуальной и востребованной сферой деятельности.

Цель дипломной работы – разработка автоматизированной информационной системы «Система заказов и доставки готовой продукции ресторана» с использованием современных технологий создания, сбора статистики и администрирования интернет-сайта.

Для достижения указанной цели в выпускной квалификационной работе выполняются следующие задачи:

– изучение и применение на практике технических аспектов разработки сервисов;

– проведение анализа и выбор технологий для разработки приложений сервиса;

– детали разработки приложений, входящих в состав сервиса доставки еды.

1 СИСТЕМНЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Характеристика задачи

В современном обществе в условиях конкурентной борьбы за рынок компании, занятой доставкой еды, необходимо внедрение современных информационных технологий – собственный сайт и приложения, позволяющие автоматизировать работу персонала.

Исходя из последних исследований, с каждым годом все больше людей предпочитают заказывать еду онлайн. Именно с помощью сайтов, они могут легко просматривать меню, выбирать блюда, указывать предпочтения и оформлять заказы, не выходя из дома. Это обеспечивает удобство и экономит время клиентов.

Таким образом, сайт позволяет достичь большего числа потенциальных клиентов, так как зачастую при его отсутствии, аудитория компании ограничивается жителями прилегающих районов. В то время как сайт компании может стать известен через поисковые системы, рекламу и социальные сети, привлекая новых клиентов и расширяя аудиторию.

Сайт должен предоставлять компании инструменты для управления заказами, такие как система отслеживания заказов, уведомления о состоянии заказа, статистика и информация о доставке. Это помогает компаниям упростить процесс обработки заказов и повысить качество обслуживания.

Важно, чтобы сайтом было удобно пользоваться со всех возможных устройств, следовательно, интерфейс должен быть адаптивным.

Стоит учесть, что целью клиентов является приобритение желаемой еды, а не регистрация в сервисе, по этой причине, необходимо предоставлять функционал оформления заказа для анонимных пользователей, а также не навязывать им регистрацию.

Также, стоит учесть, что сайт не всегда в полной мере может выполнять поставленные задачи, в таком случае, необходимо реализовывать дополнительное приложение, которое обеспечит выполнение этих задач.

1.2 Проблемы автоматизации

Интернет-сайт по доставке еды автоматизирует большое количество процессов, которые в обычном ресторане выполняют люди, следовательно, сайт позволяет экономить значительные средства на зароботной плате. Наиболее важные процессы, которые автоматизирует сайт:

Клиенты могут размещать заказы на доставку еды прямо через веб-сайт, исключая необходимость официанта. Автоматизация позволяет им выбирать блюда, добавлять их в корзину, указывать предпочтения, добавлять комментарии и оформлять заказы в удобном интерфейсе.

Сайты для доставки еды сохраняют данные о заказах и учетную информацию клиентов в централизованной системе. Полученные данные можно обрабатывать и получать статистику об объеме заказов, популярных блюдах, покупательских предпочтениях и других показателях. Статистика может показать в каком направлении двигаться бизнесу, чтобы дальше развиваться.

Компании могут легко изменять своё меню. Сайт позволяет компаниям быстро обновлять информацию и отображать актуальное меню на сайте.

Автоматизация позволяет компаниям эффективно управлять заказами. Они могут получать уведомления о новых заказах, отслеживать статус выполнения заказов, оптимизировать маршруты доставки, уведомлять клиентов о статусе и времени доставки. Это помогает ускорить процесс обработки заказов и повысить качество обслуживания.

Автоматизация этих процессов помогает упростить и ускорить работу компаний, улучшить опыт клиентов и повысить эффективность доставки готовой еды.

1.3 Анализ существующих систем

Предположив, что компания хочет занять часть интернет-рынка доставки готовой еды, то у нее не будет альтернатив, помимо создания своего собственного сайта или аналогичных способов взаимодействия с клиентами, например, бота.

По этой причине, имеет смысл сравнивать ресторан доставки готовое еды посредсвом сайта и обычный физический ресторан.

Достоинства обычного ресторана:

1) в обычном ресторане клиенты могут насладиться атмосферой, общением с персоналом и персональным обслуживанием. Это может создавать уникальный опыт посещения ресторана;

2) в ресторане клиенты получают еду практически сразу после оформления заказа. Они могут наслаждаться свежеприготовленными блюдами и контролировать качество и состояние еды;

3) рестораны часто являются местом для социальных встреч, свиданий, семейных сборов и деловых обедов. Они предоставляют возможность для общения и взаимодействия с другими людьми.

Недостатки обычного ресторана:

1) физический ресторан имеет ограниченное количество мест, что может привести к нехватке мест для клиентов в пиковые часы или требовать предварительного бронирования;

2) рестораны имеют определенное рабочее время, что может ограничивать доступность для клиентов, особенно в позднее время или в нерабочие дни;

3) посещение ресторана требует физического присутствия, что может быть неудобно для клиентов, особенно если они заняты или находятся в удалении от ресторана.

Достоинства интернет-сервиса доставки еды:

1) клиенты могут заказывать еду из дома, офиса или любого другого места с помощью интернет-сервиса доставки еды. Это предоставляет удобство и гибкость в выборе еды и времени доставки;

2) интернет-сервисы доставки еды предлагают широкий выбор ресторанов и разнообразных меню. Клиенты могут выбирать из множества кухонь и блюд, включая те, которые могут быть недоступны в их местности;

3) интернет-сервисы доставки еды обычно предлагают удобные способы оплаты и возможность отслеживания заказа в режиме реального времени.

Недостатки интернет-сервиса доставки еды:

1) заказ еды через интернет-сервис доставки может потребовать большое количество времени на доставку, особенно в пиковые часы или в случае большой загруженности. Это может привести к длительному ожиданию клиентов;

2) при доставке еды есть риск потери качества и свежести. Еда может охладиться или испортиться во время доставки, особенно если она должна быть горячей или свежей;

3) некоторые рестораны могут иметь ограниченное меню для доставки или трудности в адаптации блюд для доставки. Это может ограничивать выбор и гибкость в заказе.

1.4 Анализ существующих программных средств реализации

Тщательно изучив предметную область, было решено использовать графовую базу данных для хранения данных.

Графовые базы данных (Graph databases) - это тип баз данных, который использует графы для моделирования и хранения данных. Графовые базы данных хранят данные в виде узлов и ребер, которые представляют объекты и связи между ними.

Преимущества графовых баз данных:

– гибкость моделирования данных: графовые базы данных могут хранить данные различных типов, что делает их идеальным инструментом для моделирования сложных структур данных;

– скорость запросов: графовые базы данных используют оптимизированные алгоритмы для обработки запросов, что позволяет осуществлять поиск данных в режиме реального времени и получать результаты быстрее, чем с традиционными реляционными базами данных;

– масштабируемость: графовые базы данных могут быть легко масштабированы по мере увеличения объема данных.

Недостатками графовых баз данных являются:

– ограничения в масштабируемости: хотя графовые базы данных могут быть масштабируемыми, это может быть сложно и затратно в настройке для больших объемов данных;

– ограничения по производительности: хотя графовые базы данных могут обеспечивать высокую производительность для некоторых типов запросов, они могут быть медленнее для других типов запросов, таких как запросы, связанные с агрегированием данных;

– сложность моделирования данных: хотя графовые базы данных могут быть гибкими в моделировании данных, это может также стать причиной сложностей в проектировании базы данных и требовать более тщательного планирования структуры данных.

Одной из наиболее популярных графовых баз данных является Neo4j, которая будет использоваться в проекте.

Для разработки клиентского приложения, помимо стандартных HTML, CSS, JavaScript, будет использоваться библиотека Bootstrap, для удобной работы со стилями, а также React – библиотека для JavaScript, предназначенная для работы со страницей, как с набором компонентов, что увеличивает скорость обновления и загрузки страницы.

Для серверного приложения будет использоваться язык C#, а именно фреймворк ASP.NET Core. ASP.NET Core - фреймворк для разработки веб-приложений, который предоставляет широкий спектр функциональных возможностей для разработки масштабируемых и безопасных веб-приложений. Наиболее характерными его достоинствами являются:

– высокая производительность за счет оптимизации запросов и использованию многопоточности;

– безопасность, которая обеспечивается путем предоставления встроенных механизмов для защиты от таких угроз, как атаки инъекций, кросс-сайт скрипты и многих других;

– простота разработки.

Для мобильного приложения будет использоваться язык программирования Kotlin, который создан на основе Java Virtual Machine и разрабатывался, чтобы упростить и ускорить процесс разработки для Android. Kotlin обладает понятным и лаконичным синтаксисом, совместимостью с Java, следовательно, он может использоваться все библиотеки, которые были написаны на Java, а также высокой производительностью.

1.5 Постановка задачи

Разрабатываемая система должна хранить подробную информацию о заказах, клиентах, сотрудниках компании, кухнях, а также предоставлять удобные средства просмотра статистики. Исходя из этого, были составлены следующие задачи:

1) сервис должен предоставлять клиенту удобную возможность просмотреть список доступных блюд, заказать их, видеть состояние заказа и возможность отменить его, а также зарегистрироваться или авторизоваться;

2) разработанный сайт должен быть адаптивным, удобным, а также быстрым, что может быть достигнуто за счет использования таких технологий

как Static Site Generation, обеспечивающих предварительное создание статических файлов в момент сборки;

3) сервис должен предоставлять администраторам возможность манипулировать заказами, пользователями, менять данные о существующих блюдах, создавать новые, а также предоставлять такие статистические данные:

– топ-10 клиентов по стоимости заказов;

– количество и суммарная стоимость заказов по месяцам;

– средняя продолжительность пребывания заказа в стадии;

– топ доставщиков по количеству заказов;

– топ-10 самых популярных блюд;

– количество выполненных заказов и приготовленных блюд в каждой из кухонь;

– сколько в среднем клиенты оформляют заказов, если оформляют;

– количество отмененных заказов на каждой стадии.

4) сервис должен обладать эффективными системами аутенфикации и авторизации, позовляющим определить роль пользователя. Отдельно стоит заметить, что пользователь не должен каждый раз при входе в систему проходить систему аутентификации.;

5) сервис должен предоставлять отдельный функционал сотрудником кухонь, которые должны видеть выполняемые заказы и текущую очередь заказов;

6) заказы должны иметь возможность находится в одном из шести состояний: в очереди, готовится, в ожидании курьера, доставляется, завершен и отменен.

Исходя из составленных требований системы, были выделены пять ролей пользователей: администратор, сотрудник кухни, курьер, авторизованный клиент, а также неавторизованный клиент.

Основные возможности администратора включают в себя:

– управление заказами;

– управление пользователями;
– просмотр общей статистики;
– управление блюдами.

Основные возможности сотрудника кухни включают в себя:

– управление некоторыми состояниями заказа;
– просмотр сотрудников кухни;
– просмотр очереди заказов.

Основные возможности курьра включают в себя:

– управление некоторыми состояниями заказа;
– просмотр очереди заказов, а также истории выполненных заказов.

Основные возможности авторизованного клиента включают в себя:

– просмотр блюд;
– просмотр истории собственных заказов;
– просмотр корзины;
– оформление заказа;
– просмотр профиля.

Основные возможности неавторизованного клиента включают в себя:

– просмотр блюд;
– просмотр корзины;
– оформление заказа.

Исходя из вышеперечисленного можно понять, что разрабатываемый сервис должен состоять из нескольких программ.

Мобильное приложение для курьеров, которое будет предоставлять им все данные, необходимые для работы. Стоит заметить, что приложение будет нацелено на пользователей смартфонов с оперционной системой Android.

Интернет-сайт, который будет предоставлять удобный функционал для клиентов, сотрудников кухонь, а также администраторов.

Серверная часть должна предоставлять API для взаимодействия с графовой базой данных как для сайта, так и для мобильного приложения.

С учетом поставленой задачи, можно составить жизненный цикл заказа в системе (см. рис. 1.1).

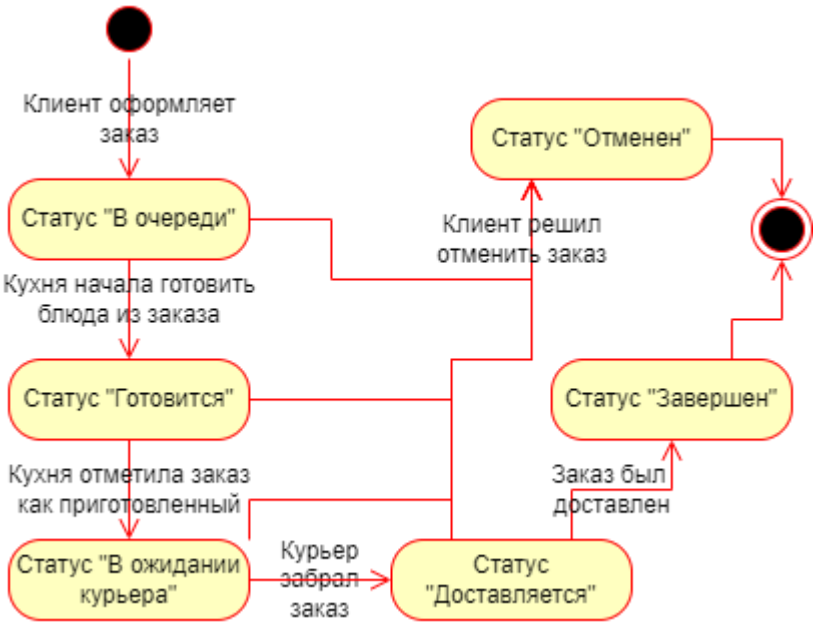


Рисунок 1.1 – Жизненный цикл заказа в системе

2 АНАЛИЗ ТРЕБОВАНИЙ

2.1 Общие и функциональные требования к системе

Основные требования к системе:

– система должна состоять из web-сайта и мобильного приложения;

– система должна позволить пользователям просматривать меню блюд на сайте;

– система должна позволить клиентам добавлять блюда в коризну и просмотраивать блюда в корзине;

– система должна предоставить клиенту возможность увеличивать и уменьшать количество заказываемых блюд в корзине, удалять товары из корзины;

– система должна предоставить неавторизованному и авторизованному клиенту возможность оформить заказ оставив номер телефона, адресс и комментарий к заказу;

– система должна предоставить пользователю возможность регистрироваться и авторизоваться на сайте;

– система должна предоставить авторизованному пользователю возможность входа в личный кабинет;

– система должна предоставить авторизованному пользователю возможность просмотра истории заказов в личном кабинете;

– система должна иметь модули управления заказами и администрирования сайтом;

– система должна предоставить администратору сайта возможность переключаться в режим администрирования;

– система должна предоставить сотруднику кухи возможность манипулирования стадиями заказов кухни, в которой данный сотрудник работает;

– система должна предоставить сотруднику кухи возможность просмотривать список сотрудников кухни, в которой данный пользователь работает;

– сайт должнен иметь интуитивно понятный интерфейс, который будет схож с аналогичными;

– сайт должен быть доступен в любом браузере на любой платформе;

– мобильное приложение курьера должно иметь авторизацию с возможностью сохранения данных о пользователе, для автоматической авторизации при последующих попытках входа в приложение;

– мобильное приложение должно предоставлять список заказов, которые связаны с курьером, а также менять состояния заказов, которые связаны с работой курьера;

– система должна взаимодействовать с графовой базой данных Neo4j для хранения данных;

– блюда, находящиеся в корзине, не должны пропадать после выхода клиента с сайта;

– сайт должен иметь меню и удобную структуру разделов меню;

– система должен иметь поиск блюд;

– система должна позволять администратору в административном модуле сайта создавать новые блюда, а также менять существующие, а также производить поиск по названию и описанию блюда;

– система должна позволять администратору в административном модуле сайта манипулировать пользователями, а также производить поиск по логину и идентификатору польователя;

– система должна позволять администратору в административном модуле сайта манипулировать заказами, а также их состояниями, а также производить разделение заказов по их состоянию;

– система должна позволять администратору в административном модуле сайта просматривать статистику;

– система должна позволять администратору в административном модуле сайта скрывать блюда;

– система должна предоставить пользователю возможность просматривать блюда как в виде общего списка на странице, так и в виде одной страницы, на которой расположена вся информация о блюде;

– система должна предоставить пользователям возможность выхода из учётной записи в личном кабинете;

– система должна разграничивать права доступа.

2.2 Требования к интерфейсам

Интерфейс сайта должен соответствовать следующим требованиям:

– клиентская часть сайта должна иметь простой интуитивно понятный интерфейс;

– интерфейс должен быть оформлен в простой, но в тоже время лаконичной, цветовой палитре;

– интерфейс должен иметь в своём составе изображения блюд и иконки, которые способствуют повышению понимания интерфейса со стороны пользователя;

– интерфейс должен отображать главный функционал системы;

– интерфейс должен выводить уведомления пользователю о том, что какие-то поля не заполнены, либо заполнены неверно, а также предупреждать о других ошибках;

– интерфейс административного модуля должен быть простым и лишённым лишних деталей;

– интерфейс административного модуля должен предоставлять возможность комфортного просмотра статистических данных;

– интерфейс модуля управления заказами, блюдами, а также пользователями, должен быть простым и лишённым лишних деталей;

– интерфейс модуля управления заказами должен позволять удобно просматривать списки заказов по статусам, а также показывать историю их состояний;

– интерфейс должен быть адаптивным, т.е. пользователи, использующие устойства с маленькими экранами, должны также комфортно себя чувствовать, как и пользователи использующие большие экраны.

Интерфейс мобильного приложения должен соответствовать следующим требованиям:

– мобильное приложение должно быть простым в освоении и интуинтивно понятным пользователям любых возрастов;

– мобильное прилоджение должно иметь светлую и темную тему для удобной работы с любое время суток;

– мобильное приложение не должно иметь лишнего функционала.

2.3 Анализ требований к базе данных

База данных сервиса доставки еды должна:

– быть реализована с помощью Neo4j;

– должна являться NoSQL базой данных графового типа;

– должна содержать в себе узлы следующих типов: «Заказ», «Администратор», «Клиент», «Курьер», «Сотрудник кухни», «Кухня», «Блюдо», «Состояние заказа», «Категория Блюда»;

– должна содержать в себе связи следующих типов: «Оформленый заказ клиентом», «Оставленный отзыв на заказ клиентом», «Доставленный заказ курьером», «Сотрудник кухни, работающий на кухне», «Заказ, приготовленный кухней кухни», «Заказ, имеющий состояние», «Категория, содержащая блюдо», «Блюдо, находящееся в заказе»;

– должна иметь программное обеспечение от официального издателя, которое имеет возможность визуализировать графы в удобном виде.

2.4 Анализ требований к характеристикам аппаратуры и программному обеспечению

Для установки мобильного приложения курьерам потребуется мобильное устройство с версией Android выше 7.0.

Разрабатываемая система является веб-сай по этой причине для ее развертывания потребуется хостинг или сервер. Хостинг является более предпочтительным вариантом, т.к. избавляет от проблем, связанных с обслуживанием физического сервера и иным недостаткам. Для установки системы на хостинг потребуется возможность запуска контейнеров docker, объём хранилища от 10Гб, доменное имя, интренет-трафик более 10Гб;

Для входа на сайт достаточно любого современного браузера на любом устройстве с экраном разрешение, которого превышает 375x660 пикселей и, которые соответствуют системным требованиям браузеров.

3 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Диаграммы прецедентов

Исходя из составленных требований системы, были выделены пять ролей пользователей:

– администратор;
– сотрудник кухни;
– курьер;
– авторизованный клиент;
– неавторизованный клиент (анонимный клиент).

После опеределения ролей следует приступить к проектированию диаграмм прецедентов для каждой роли. Данный тип диаграмм помогают понять и описать функциональные возможности системы, а также ее взаимодействие с внешними сущностями.

На рисунке 3.1 представлена диаграмма прецедентов для неавторизанного клиента сайта.

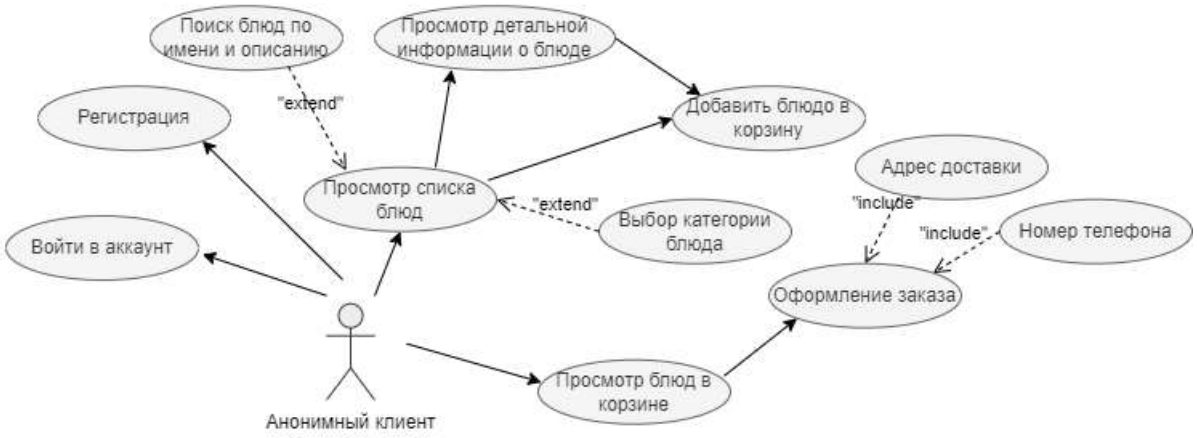


Рисунок 3.1 – Диаграмма прецедентов для роли «Анонимный клиент»

Анонимный клиент может просматривать блюда, зарегестрироваться, попытаться войти в аккаунт, добавить блюдо в корзину, просмотреть корзину,

а также оформить заказ, однако в отличии от авторизованного клиента, он не может просмотреть профиль, а, следовательно, историю заказов.

Если он захочет отменить заказ или изменить количество блюд в заказе, то ему придется звонить на горячую линию.

Анонимный клиент может зарегистрироваться или войти в систему (если он ранее регистрировался), что увеличит его фукнционал до функционала «Авторизованный клиент».

На рисунке 3.2 представлена диаграмма прецедентов для авторизанного клиента сайта.

Рисунок 3.2 – Диаграмма прецедентов для роли «Авторизанный клиент»

Авторизанный клиент – это пользователь, который ранее зарегистрировался, соответственно, имеет учётную запись и вошёл в систему. Данный тип пользователей имеет все возможности «Анонимный клиент», а также может с помощью функционала сайта:

– просмотреть профиль;
– просмотреть историю заказов;
– отменить заказ и указать причину;
– оставить отзыв и оценку о завершенном заказе;
– выйти из аккаунта.

На рисунке 3.3 представлена диаграмма прецедентов для сотрудника кухни.

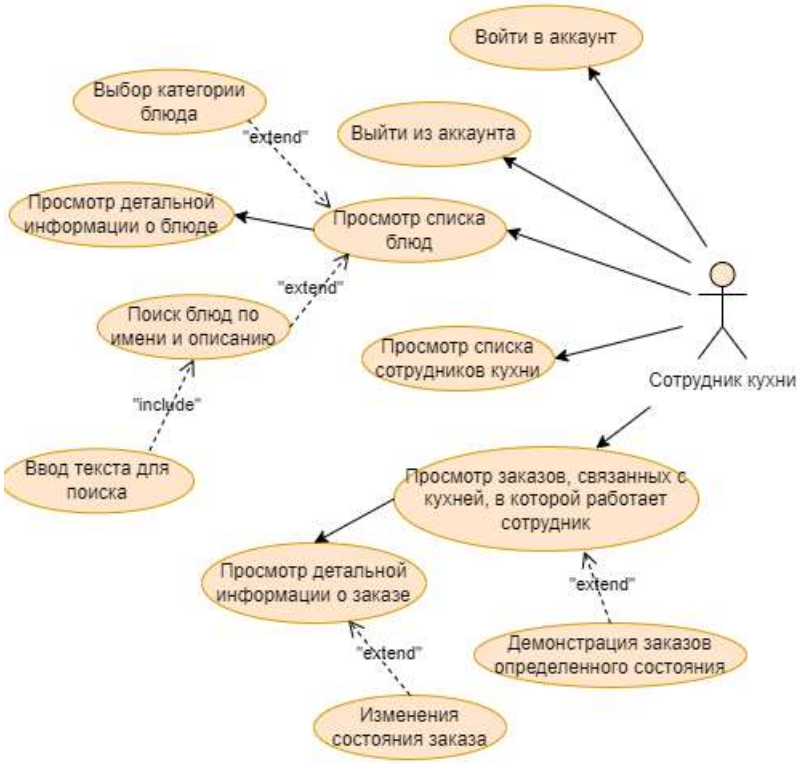


Рисунок 3.3 – Диаграмма прецедентов для роли «Сотрудник кухни»

Сотрудник кухни – пользователь, который является сотрудником одной из кухонь, следовательно, имеет следующий функционал:

– имеет доступ к информации о сотрудниках кухни;
– может просматривать заказы кухни;
– может менять состояние заказов кухни;
– может просмотраивать блюда, доступные для покупки;
– может выйти из аккаунта.

Стоит заметить, что сотрудник кухни не может оформить заказ. Это сделано с целью предотвращения возможных мошеннических схем.

На рисунке 3.4 представлена диаграмма прецедентов для курьера, который взаимодействует с сервером посредством мобильного приложения.

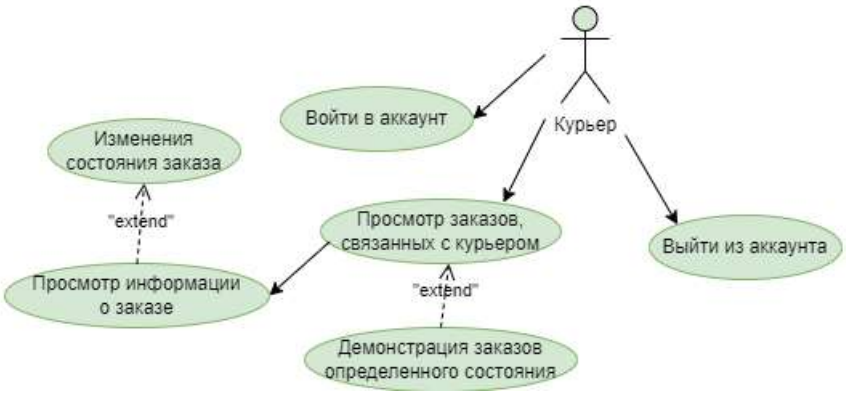


Рисунок 3.4 – Диаграмма прецедентов для роли «Курьер»

Мобильное приложение предоставляет возможность авторизоваться, выйти из аккаунта, а также необходимый функционал для работы курьера, а именно: просмотр списка заказов курьера, подробный просмотр содержимого заказов, а также возможность менять состояние заказов.

На рисунке 3.5 представлена диаграмма прецедентов для курьера, который взаимодействует с сервером посредством мобильного приложения.

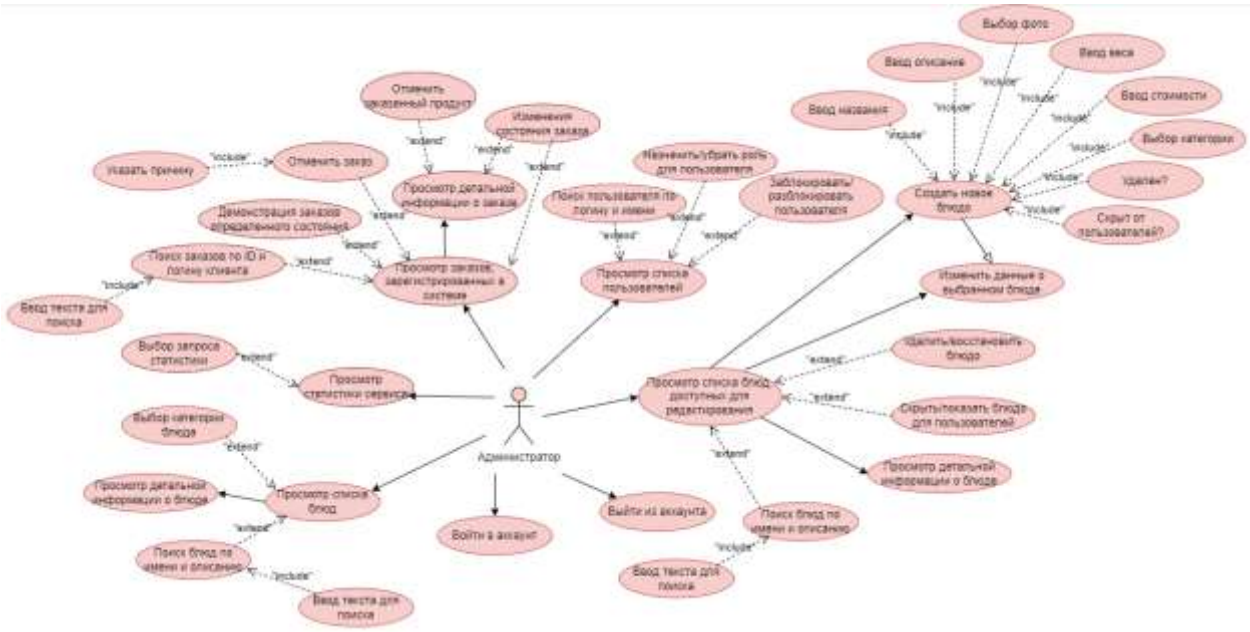


Рисунок 3.5 – Диаграмма прецедентов для роли «Администратор»

Администратор – пользователь, который имеет наибольшее количиество возможностей в системе, следовательно, его функционал наиболее широкий. Среди доступного ему функционала следует выделить:

– просмотр статистики;

– просмотр заказов в системе, а также взаимодействие с ними, в том числе, изменение их содержимого, отмена продуктов, отмена заказов и перевод в следующую или в предыдущую стадию;

– просмотр списка пользователей в системе, а также взаимодействие с ними, в том числе, блокирование пользователей, назначение и удалени прав у пользователей;

– просмотр списка блюд в системе, а также взаимодействие с ними, в том числе, изменение информации о блюде, удаление и сокрытие от пользователей, просмотр детальной информации о блюде;

– создание нового блюда.

Стоит заметить, что администратор, также, как и сотрудник кухни и курьер, не могут оформлять заказы со своего аккаунта.

3.2 Диаграмма состояний

Рассмотрим диаграмму состояний. Она позволяет описать все возможные состояния, в которых может находиться объект или система, и переходы между этими состояниями. В качестве обозреваемого объекта будет выступать заказ (см. рис. 3.6)

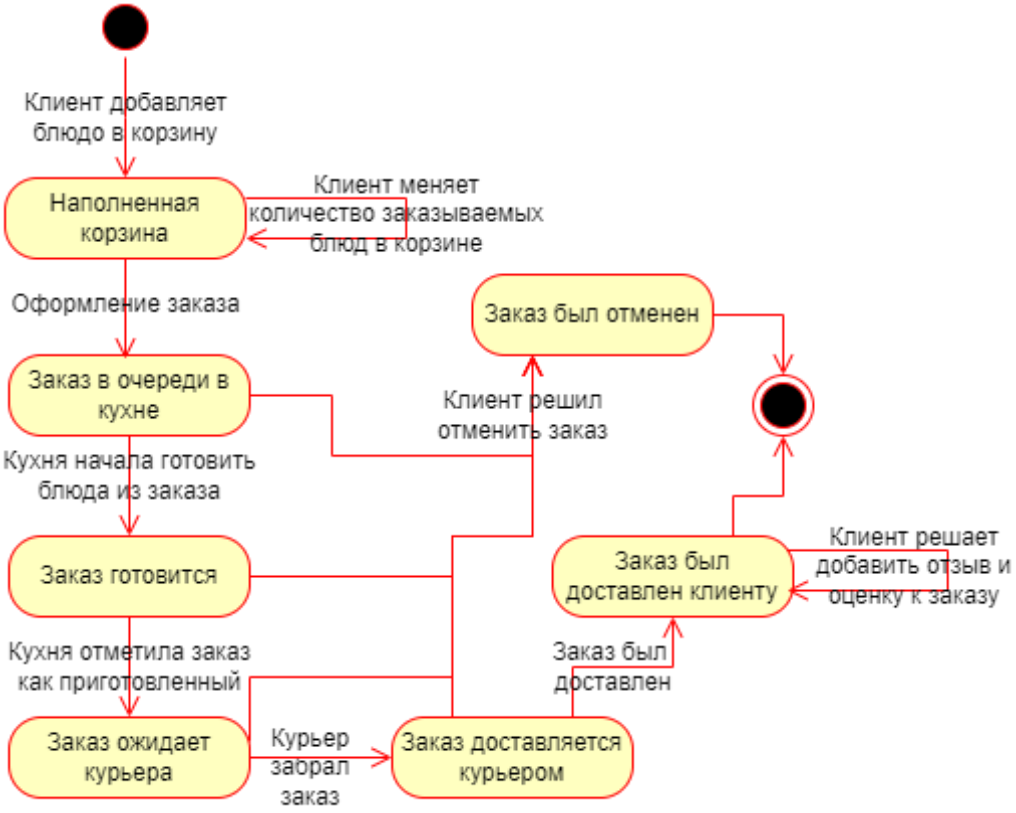


Рисунок 3.6 – Диаграмма состояний для объекта «Заказ»

Благодрая диаграмме наглядно видны действия, а также состояния, которые принимает заказ в каждый из этапов своей жизни.

Сперва клиент выбирает блюда из списка и добавляет их в корзину, после чего, при желании, может изменить содержимое коризны, после оформляет заказ. Заказ попадает в очередь одной из кухонь и дожидается своего часа, после чего, сотрудники кухни готовят его. Когда заказ готов, то он находится в ожидании курьера, который доставит его клиенту. Клиент может в любой момент отменить заказ, в таком случае, его стадия жизни завершится. После того как курьер доставил заказ клиенту и тем самым завершил его, клиент может оставить отзыв о заказе. Данный отзыв в дальнейшем может быть использован администрацией для улучшния качества работы сервиса.

4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

4.1 Проектирование концептуальной модели БД

После тщательного анализа предметной области была разработана база данных, состоящая из 9 узлов, а также 8 связями.

Было создано 9 типов узлов:

1) “Order” – заказ;
2) “Admin” – администратор;
3) “Client” – клиент;
4) “DeliveryMan” – курьер;
5) “KitchenWorker” – сотрудник кухни;
6) “Kitchen” – кухня;
7) “Dish” – блюдо;
8) “OrderState” – состояние заказа;
9) “Category” – категория блюда.

Было создано 8 типов связей между узлами:

1) “Ordered” – оформленый заказ клиентом;
2) “ReviewedBy” – оставленный отзыв на заказ клиентом;
3) “DeliveredBy” – доставленный заказ курьером;
4) “WorkedIn” – сотрудник кухни, работающий на кухне;
5) “CookedBy” – заказ, приготовленный кухней;
6) “HasOrderState” – заказ, имеющий состояние;
7) “ContainDish” – категория, содержащая блюдо;
8) “OrderedDish” – блюдо, находящееся в заказе;

Исходя из имеющихся данных, была создана графовая модель данных (см. рис. 4.1)

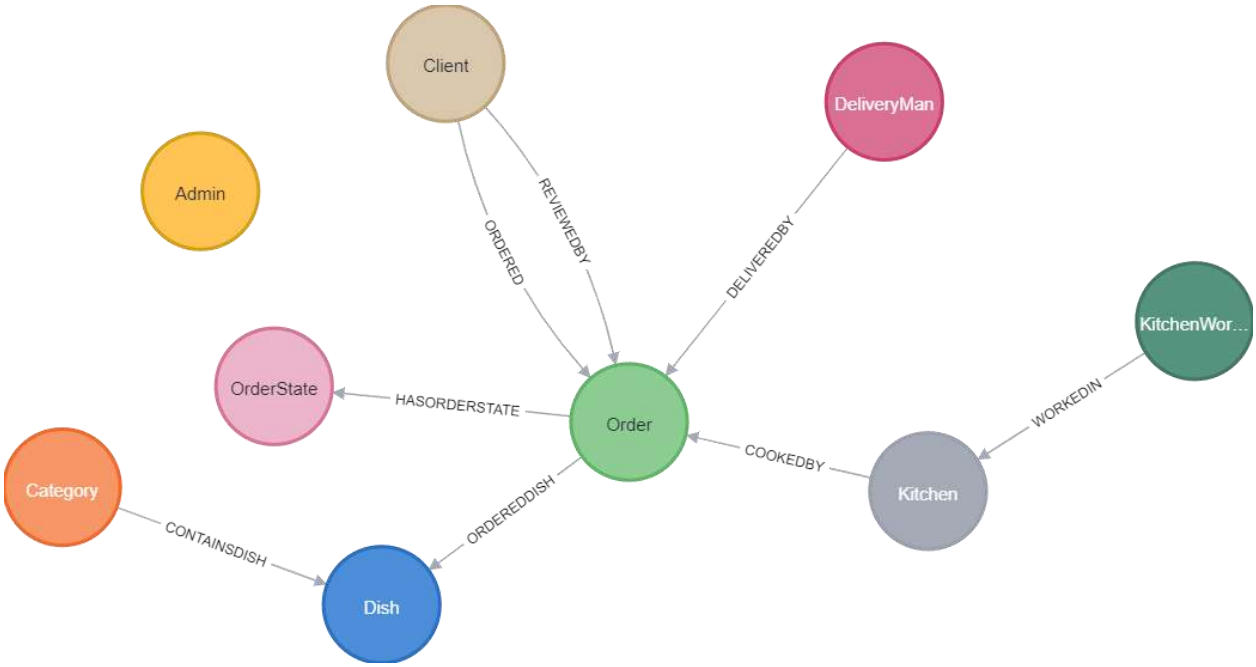


Рисунок 4.1 – Схема графовой базы данных «Система заказов и доставки готовой продукции ресторана»

Параметры узлов приведены в таблицах 4.1 – 4.9.

Таблица 4.1 – «Order»

№	Name	Comment
1	Id	Уникальный идентификатор
2	DeliveryAddress	Адрес доставки
3	Price	Стоимость заказа
4	StoryJson	История изменения состояний в формате json
5	SumWeight	Суммарный вес продуктов
6	PhoneNumber	Номер телефона заказчика

Таблица 4.2 – «Kitchen»

№	Name	Comment
1	Id	Уникальный идентификатор
2	Address	Адрес кухни

Таблица 4.3 – «Admin»

№	Name	Comment
1	Id	Уникальный идентификатор
2	Born	Дата рождения
3	Login	Логин
4	Name	Имя
5	PasswordHash	Хеш пароля
6	PhoneNumber	Номер телефона

Таблица 4.4 – «DeliveryMan»

№	Name	Comment
1	Id	Уникальный идентификатор
2	Born	Дата рождения
3	Login	Логин
4	Name	Имя
5	PasswordHash	Хеш пароля
6	PhoneNumber	Номер телефона
7	MaxWeight	Максимальный вес, который может переносить курьер

Таблица 4.5 – «OrderState»

№	Name	Comment
1	Id	Уникальный идентификатор
2	DescriptionForClient	Описание состояния заказа для клиента
3	NameOfState	Название состояния
4	NumberOfStage	Номер состояния

Таблица 4.6 – «KitchenWorker»

№	Name	Comment
1	Id	Уникальный идентификатор
2	Born	Дата рождения
3	Login	Логин
4	Name	Имя

Продолжение таблицы 4.6

5	PasswordHash	Хеш пароля
6	PhoneNumber	Номер телефона
7	JobTitle	Должность

Таблица 4.7 – «Dish»

№	Name	Comment
1	Id	Уникальный идентификатор
2	Description	Описание блюда
3	DirectoryWithImages	Папка с изображениями блюда
4	IsAvailable	Доступен ли клиенту
5	Name	Название блюда
6	Price	Стоимость за одну единицу
7	Weight	Вес

Таблица 4.8 – «Category»

№	Name	Comment
1	Id	Уникальный идентификатор
2	CategoryNumber	Номер категории
3	Description	Описание категории
4	Name	Название категории

Таблица 4.9 – «Client»

№	Name	Comment
1	Id	Уникальный идентификатор
2	Born	Дата рождения
3	Login	Логин
4	Name	Имя
5	PasswordHash	Хеш пароля
6	PhoneNumber	Номер телефона
7	Bonuses	Количество бонусов

Помимо узлов, данные также хранят связи. Всего создано 8 типов связей, каждая из которых хранит свой уникальный идентификатор, уникальный идентификатор начального и конечного узла. Однако 4 связи имеют дополнительные свойства, они продемонстрированы в таблицах 3.10 – 3.13.

Таблица 4.10 – Уникальные свойства связи «WorkedIn»

№	Name	Comment
1	GotJob	Дата вступления в должность

Таблица 4.11 – Уникальные свойства связи «ReviewedBy»

№	Name	Comment
1	ClientRating	Оценка пользователя
2	TimeCreated	Время создания отзыва
3	Review	Отзыв

Таблица 4.12 – Уникальные свойства связи «OrderedDish»

№	Name	Comment
1	Count	Количество заказанных блюд

Таблица 4.13 – Уникальные свойства связи «HasOrderState»

№	Name	Comment
1	TimeStartState	Время начала действия состояния
2	Comment	Комментарий изменения состояния

Приведенных свойств достаточно для работы с необходимыми данными, а также корректного функционирования системы.

4.2 Теоритическая основа для реализации сервиса для взаимодействия с базой данных

В течении длительного периода времени создавались и оттачивались подходы для взаимодействия с базой данных (БД). Один из таких подходов – Object Relation Mapping.

Object Relation Mapping (ORM) – архитектурный паттерн, который связывает БД с принципами объектно-ориентированных языков программирования. Суть данной технологии заключается в том, что каждая строка реляционной БД сопоставляется с объектом класса, который задает разработчик. В свойство объекта помещаются данные из соответствующего столбца, сопоставление происходит чаще всего по имени свойства и столбца. Благодаря использованию данной технологии мы можем взаимодействовать с каждой строкой БД как с объектом. Если речь идет о графовой БД, то свойства каждого узла и связи между узлами сопоставляются со свойствами объекта.

Многие паттерны способны дополнять друг друга, ORM не является исключением. Данный паттерн является основой для более масштабных шаблонов, самые популярные из них это ActiveRecord, DataMapper, Repository. В современной разработке паттерн Repository стал наиболее популярным, так как он является наиболее гибким и масштабируемым.

Repository – паттерн, представляющий собой уровень доступа к данным, который выполняет двунаправленную передачу данных между постоянным хранилищем данных (зачастую базой данных) и уровнем отображения данных (веб страница, окно приложения и т.д.). Паттерн выполняет функцию изолирования уровня БД и отображения. Слой состоит из одного или нескольких репозиториев (классов манипулирования информации указанного типа), выполняющих передачу данных. Реализации репозиториев могут различаться по объему и предоставляемому функционалу, так универсальные репозитории будут использовать обобщенные типы и тем самым обрабатывать множество различных типов, а специализированные репозитории будут обрабатывать указанные типы.

В самом простом виде, репозиторий будет иметь связь, показанную на рисунке 4.2.

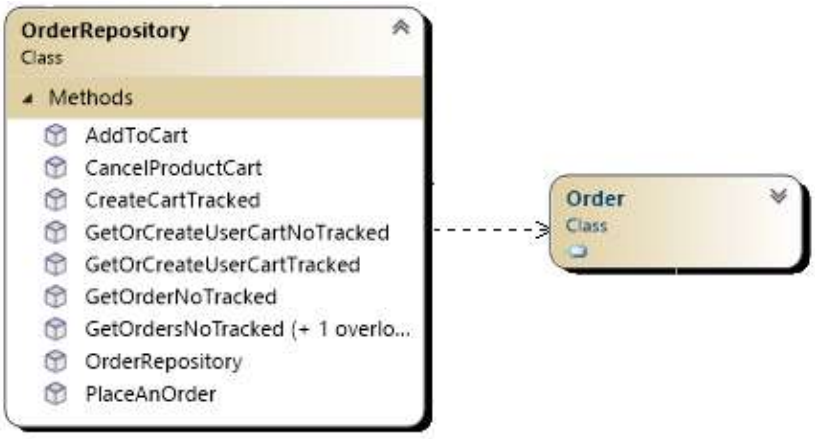


Рисунок 4.2 –Примитивная реализация паттерна Repository

В таком виде паттерн хорошо подходит для реляционных БД, однако графовые БД требуют иной подход, так как они могут хранить данные не только в узлах, но и в связях.

4.3 Практическая реализации сервиса для взаимодействия с базой данных

Предложенное развитие паттерна Repository состоит с следующем. Для начала необходимо определить интерфейс IModel, который будут расширять все остальные интерфейсы и реализовывать классы узлом и связей базы данных. Все классы, реализующие данный интерфейс, будут обязаны определить свойства и методы, находящиеся в данном интерфейсе. Одним из таких свойств является свойство Id, которое служит уникальным идентификатором.

Интерфейс IModel расширяют интерфейсы INode и IRelation, которые предназначены для узлов и связей соответственно.

Класс Node реализует интерфейс INode. Данный класс выступает базовым для узлов, хранящихся в БД. Интерфейс IRelation реализует обобщенный класс Relation <TNodeFrom, TNodeTo>, они являются базовыми для связей в БД. Помимо уникального идентификатора, интерфейс будет задавать ссылки на

узлы, между которыми будет проложена связь, а также сами узлы. Не всегда возникает необходимость указывать из какого в какой узел идет связь, однако в текущем проекте использовать база данных Neo4j, которая создает исключительно направленные связи.

Реализованные интерфейсы и классы изображены на рисунке 4.3.

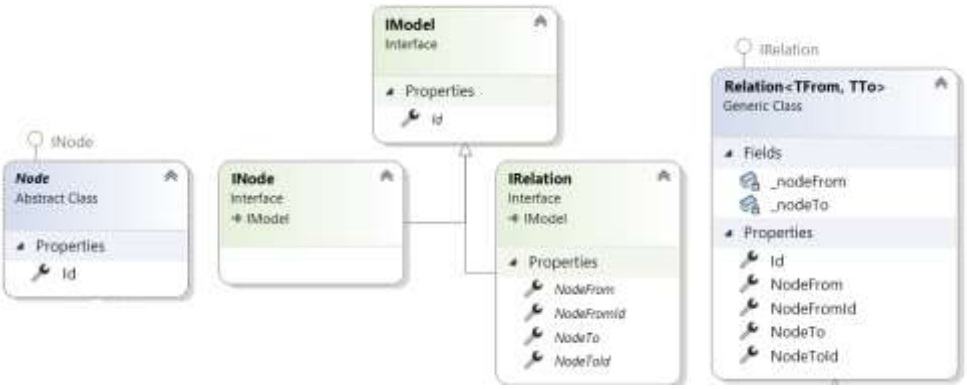


Рисунок 4.3 – Диаграмма базовых классов для данных, хранимых в БД

Следующим шагом будет создание непосредственно репозиториев и необходимой инфраструктуры подходящих для графовых БД.

Интерфейс IGeneralRepository<TNode> отвечает за определение методов, которые должны быть в каждой реализации универсального репозитория, а класс GeneralRepository<TNode> реализует данный интерфейс. В универсальном классе реализованы основные методы для работы с узлами, но наибольшего внимания требуют методы для работы со связями. Так, если нам нужны связанные узлы с неким определенным типом связи, то нам необходимо указать тип связи, для этого вновь используются обобщения.

На рисунке 4.4 для примера изображена сигнатура метода GetRelatedNodesAsync из интерфейса IGeneralRepository<TNode>.

```
Task<List<TRelation>> GetRelatedNodesAsync<TRelation, TRelatedNode>
    (TNode node, int? skipCount = null, int? limitCount = null, params string[] orderByProperty)
    where TRelation : IRelation
    where TRelatedNode : INode;
```

Рисунок 4.4 – Сигнатура метода GetRelatedNodesAsync

Данный метод возвращает список связей указанного типа, количество связей будет равняться количеству связанных элементов.

Для получения результата для данного запроса, необходимо определить к какому узлу относится переданный нам объект, он является начальной или финальной точкой связи, для этого необходимо использовать рефлексию. Однако пользователю данного класса нет необходимости об этом беспокоиться.

Разработчику могут понадобиться методы, которые выходят за рамки реализованного функционала, для этого он может создать специализированные репозитории, которые должны наследоваться от универсального. В диаграмме классов на рисунке 4.5 можно увидеть пример реализации специализированного репозитория.

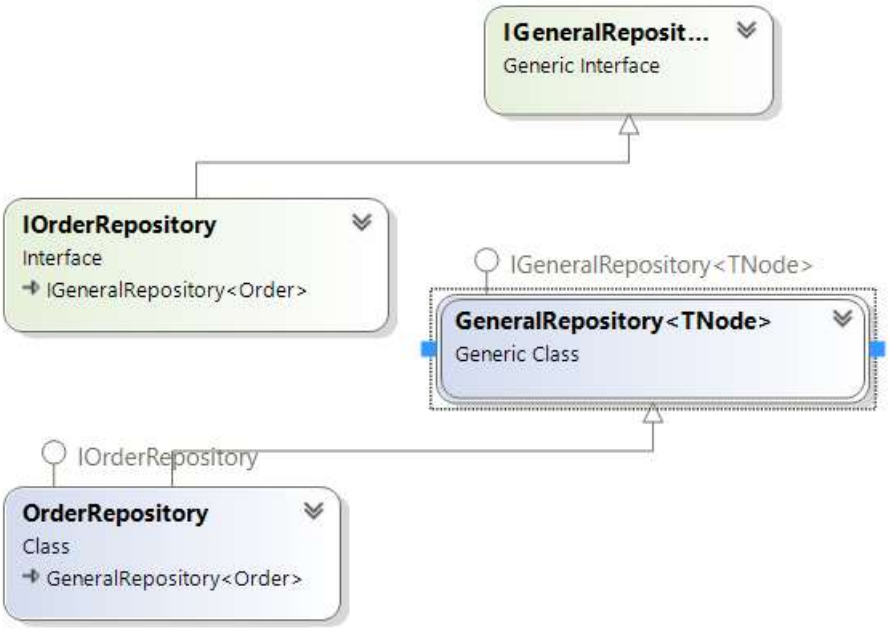


Рисунок 4.5 – Диаграмма классов паттерна Repository

4.4 Диаграмма классов модуля работы с базой данных

На рисунке 4.6 можно увидеть диаграмму классов модуля работы с базой данных. В данно диаграме видны все узлы, которые наследуются от класса Node, а также связи, которые наследуются от класса Relation. Помимо этого, видны все реализованные специализированные репозитории, которые были созданы для узлов «Dish», «Order», «User», «Client», «DeliveryMan».

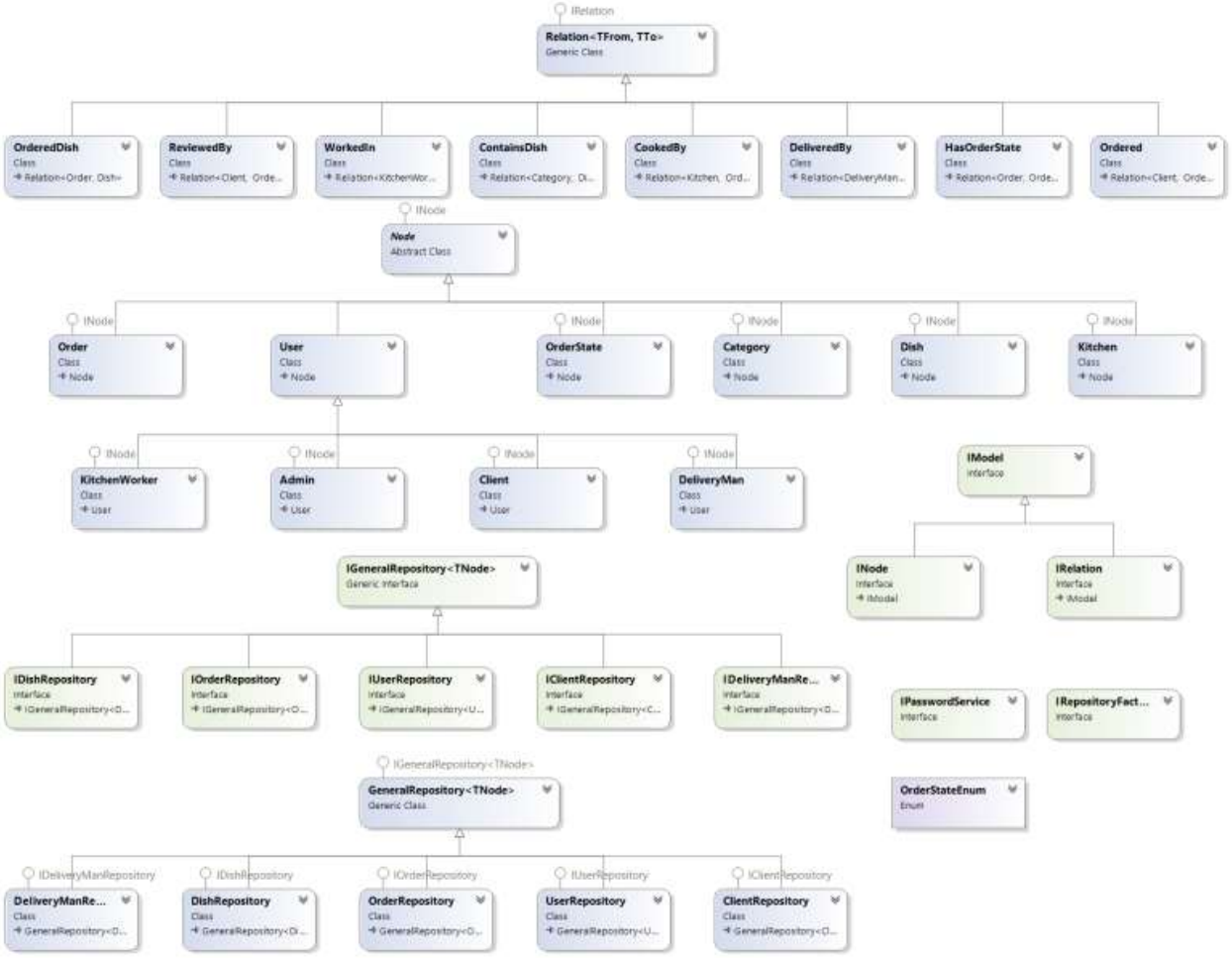


Рисунок 4.6 – Диаграмма классов модуля работы с базой данных

Всего, непосредственно с базой данных, связано 38 классов, из которых 17 являются узлами или связями.

5 ОПИСАНИЕ ПРОГРАММ

Программы суммарно содержат 141 файл, разработанных в ходе написания данного сервиса.

Серверное приложение содержит 71 файл, из которых 38 связаны с взаимодействие с базой данных, 4 относятся к генерации тестовых данных, 15 файлов являются моделями DTO и предназначены для получения и передачи информации с помощью API в нужном виде, 7 контроллеров, а также 7 классов, которые относятся к выполнению иных задач.

Клиентское приложение содержит 54 файла, из которых 20 являются страницами, 6 структурными компонентами, 3 компонента визуализации статиситики в виде диаграмм, а также 25 иных компонентов.

Мобильное приоджение содержит 16 файлов, из которых 3 являются Activity и отвечают за функционал страниц, 7 файлов моделей DTO для получения и передачи данных на сервер и 2 вспомогательных, также написанных на Kotlin, помимо этого присутствует 6 файлов, отвечающих структуризацию элементов, находящихся на страницах, которые были написаны на XML.

5.1 Реализация серверного приложения

Реализация серверной программы начинается с файла Program.cs, в котором необходимо указать сервисы, которые могут быть использованы, а после использовать их, в случае необходимости.

Конечными точками в middleware приложения являются контроллеры, которые обрабатывают запросы пользователя. Приложение основано на архитектуре REST API.

Архитектура REST (Representational State Transfer) является стилем разработки веб-сервисов, который определяет набор принципов и ограничений для создания масштабируемых и гибких API. REST API

обеспечивает обмен данными между клиентом и сервером на основе стандартных протоколов HTTP.

REST API следует принципу разделения клиента и сервера. Клиенты отвечают за пользовательский интерфейс и взаимодействие с пользователем, в то время как серверы предоставляют данные и выполняют бизнес-логику.

5.1.1 Описание авторизации с использованием JWT

Одним из важнейших сервисов является «AddAuthentication()», который настраивает способ авторизации в приложении, а следом за ним, подключается сервис «AddJwtBearer()», который предоставляет функционал, необходимый для авторизации на основе JWT-токенов.

JWT (JSON Web Token) - это открытый стандарт для создания токенов авторизации в формате JSON. Он используется для безопасной передачи информации между сторонами в виде токена, который также называют «access token». JWT состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи (signature).

```
.AddJwtBearer(options =>
{
    options.RequireHttpsMetadata = false;
    options.SaveToken = true;
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration.GetSe
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true,
    };
});
```

Рисунок 5.1 – Подключение сервиса для работы с JWT

При реализации данного функционала, было принято решение использовать refresh token, который имеет длительный срок хранения и

позволяет пользователю, по истечению срока JWT, не проходить процес аутентификации, а в автоматическом режиме получить новый JWT.

Процесс работы с JWT-токенами, посредством refresh token, выглядит следующим образом:

1) аутентификация: пользователь предоставляет логин и пароль от своего аккаунта для аутентификации. Сервер проверяет правильность учетных данных и, если они верны, генерирует пару токенов: access token (токен доступа) и refresh token (токен обновления). В токене access token в качестве payload хранится информация о ролях пользователя, его уникальный Id и времени жизни токена. Refresh token представляет собой guid;

2) выдача токенов: сервер отправляет токены клиенту, который сохраняет refresh token в куки с меткой HttpOnly, которая запрещает получать доступ к данной куки из клиенского кода (JavaScript), а access token сохраняет в localStorage;

3) использование access token: клиент включает access token в каждый запрос в заголовке «Authorization» с префиксом «Bearer ». Сервер проверяет валидность и подлинность access token и использует информацию из его полезной нагрузки для принятия решений о доступе;

4) обновление токенов: когда время жизни access token истечет и клиент попытается перейти на другую страницу, то на сервер будет отправлен запрос на обновление access token. Сервер проверяет валидность refresh token и, если он действителен, обновляет время жизни refresh token и генерирует новый access token, который отправляется обратно клиенту. Клиент обновляет свой access token. Время жизни refresh token – 60 дней, если в течении этого промежутка времени пользователь не посещает сайт, то пользователь будет вынужден заново пройти процедуру, описанную в шаге 1 [6].

Для реализации описанного функционала был создан сервис «JwtService» для предоставления функционала, связанного с access token.

```
public JwtTokenInfoOutDTO GenerateAccessJwtToken(string userId, List<string> roles)
{
    var secretKey = Encoding.ASCII.GetBytes(_appSettings.JwtSecretKey);

    var tokenHandler = new JwtSecurityTokenHandler();
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.Sid, userId),
        }),
        Expires = DateTime.Now.AddMinutes(30), // Время истечения токена
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(secretKey), SecurityAlgorithms.HmacSha256Signature)
    };
    foreach (var role in roles)
    {
        tokenDescriptor.Subject.AddClaim(new Claim(ClaimTypes.Role, role));
    }

    var token = tokenHandler.CreateToken(tokenDescriptor);
    var tokenString = tokenHandler.WriteToken(token);

    return new JwtTokenInfoOutDTO { JwtToken = tokenString, ValidTo = token.ValidTo, RoleNames = roles };
}
```

Рисунок 5.2 – Метод из сервиса «JwtService» для генерации access token

За действия, связанные с аккаунтом отвечает контроллер AuthController.

Для аутентификации пользователя используется метод Login, в котором производится поиск пользователей по логину, а после производится сравнение хеша пароля из базы данных и вычисленного предоставленого пользователем пароля. Для хеширования пароля используется алгоритм Argon2i, который в нынешний момент является наиболее криптостойким. Если проверка прошла успешно, то генерируется refresh token и отправляется в качестве куки.

```
[HttpPost("login")]
public async Task<IActionResult> Login(UserLoginInDTO data)
{
    var users = await _repositoryFactory.GetRepository<User>().GetNodesByPropertyAsync("Login", new[] { data.Login });
    if(users.Count != 1)
        return BadRequest();

    var user = users[0];

    if (user.IsBlocked)
        return BadRequest("Ваш аккаунт был заблокирован!");

    if(_pswService.CheckPassword(data.Login, data.Password, user.PasswordHash.ToArray()))
    {
        user.RefreshToken = Guid.NewGuid();
        user.RefreshTokenCreated = DateTime.Now;

        await _repositoryFactory.GetRepository<User>().UpdateNodesPropertiesAsync(user);

        AddCookieDataToResponse(user.RefreshToken.ToString(), user.Id.ToString());

        return Ok();
    }

    return Unauthorized("Вы ввели не верный логин или пароль");
}
```

Рисунок 5.3 – Метод «Login» из контроллера «AuthController»

Для обновления access token пользователя используется метод RefreshAccessToken, который проверяет пользовательский токен с хранящимся в базе данных и при совпадении генерирует новый и возвращает его пользователю.

```
[HttpPost("refreshAccessToken")]
public async Task<IActionResult> RefreshAccessToken()
{
    var userRepo = (IUserRepository)_repositoryFactory.GetRepository<User>(true);

    var userId = Request.Cookies["X-UserId"];
    if (userId == null)
        return BadRequest("У вас отсутсвует refresh token. Вам необходимо авторизоваться или зарегистрироваться.");

    var inputRefreshToken = Request.Cookies["X-Refresh-Token"];

    var userNode = await userRepo.GetNodeAsync(userId);

    if (userNode.IsBlocked)
        return BadRequest("Ваш аккаунт был заблокирован!");

    if (inputRefreshToken == userNode.RefreshToken.ToString() && userNode.RefreshTokenCreated.AddDays(60) > DateTime.Now)
    {
        var userRoles = await userRepo.GetUserRoles(userId);

        var jwtTokenInfo = _jwtService.GenerateAccessJwtToken(userId, userRoles);

        userNode.RefreshTokenCreated = DateTime.Now;

        await userRepo.UpdateNodesPropertiesAsync(userNode);

        AddCookieDataToResponse(userNode.RefreshToken.ToString());

        return Ok(jwtTokenInfo);
    }

    return BadRequest("You refresh token don't work. You need to login or signup to system");
```

Рисунок 5.4 – Метод «RefreshAccessToken» из контроллера «AuthController»

5.1.2 Описание алгоритма хеширования Argon2i

Argon2 — это функция хэширования паролей, которая обобщает современные достижения в разработке функций, требующих жесткого доступа к памяти, и может использоваться для хэширования паролей для хранения учетных данных, получения ключей или других приложений.

Он имеет простую конструкцию, направленную на максимальную скорость заполнения памяти и эффективное использование нескольких вычислительных блоков, при этом обеспечивая защиту от атак компромисса (путем использования кеша и организации памяти последних процессоров).

Argon2 имеет три варианта: Argon2i, Argon2d и Argon2id. Argon2d работает быстрее и использует доступ к памяти, зависящий от данных, что делает его очень устойчивым к атакам взлома графического процессора и подходящим для приложений без угроз от атак по времени по сторонним каналам (например, криптовалюты). Вместо этого Argon2i использует независимый от данных доступ к памяти, который предпочтительнее для хеширования паролей и получения ключей на основе паролей, но он медленнее, поскольку выполняет больше проходов по памяти для защиты от атак компромисса. Argon2id представляет собой гибрид Argon2i и Argon2d, использующий комбинацию доступа к памяти, зависящего от данных и независимого от данных, что обеспечивает некоторую устойчивость Argon2i к атакам синхронизации кэша по сторонним каналам и большую устойчивость Argon2d к атакам взлома графического процессора.

Argon2 позволяет настраивать следующие параметры хэширования:

– количество итераций;
– желаемый объем занимаемой памяти;
– степень параллелизма;
– размер результата, в байтах;
– секретный ключ [7].

Для данного алгоритма была использована библиотека, написанная разработчиком алгоритма, а также был написан сервис под названием «PasswordService», предоставляющий функционал для работы с алгоритмом.

В методе «GetPasswordHash()» определяются параметры, который будут использованы для получения хеша. Был использован алгоритм Argon2i, указывая, что необходимо использовать 16 потоков, 4 Гб оперативной памяти, а также проверсти 40 итераций, после чего вернуть 128 байтный хеш.

```
public byte[] GetPasswordHash(string salt, string password)
{
    var argon2 = new Argon2i(password.Select(h => ((byte)h)).ToArray());
    argon2.Salt = salt.Select(h => ((byte)h)).ToArray();
    argon2.DegreeOfParallelism = 16;
    argon2.MemorySize = 4096;
    argon2.Iterations = 40;

    var hash = argon2.GetBytes(128);

    return hash;
}
```

Рисунок 5.5 – Метод «GetPasswordHash» из сервиса «PasswordService»

5.1.3 Описание контроллеров

Всего было создано 7 контроллеров, каждый из которых имеет свою «зону ответственности».

Таблица 5.1 – API методы контроллера «AdminController»

Название метода	Метод HTTP	Описание метода
GetOrders	GET	Возвращает заказы, которые находятся в указанном состоянии и для указанной страницы.
GetDishes	GET	Возвращает найденные блюда для указанной страницы. Поиск производится по названию и описанию блюда.
GetDish	GET	Возвращает блюдо по указанному Id.
GetUsers	GET	Возвращает найденных пользователей для указанной страницы. Поиск производится по логину и Id.
GetRoles	GET	Возвращает список ролей, имеющихся в системе и доступных для модификаций.
AddUserRole	POST	Добавляет указанным пользователям указанную роль.
RemoveUserRole	POST	Удаляет указанным пользователям указанную роль.
BlockUsers	POST	Блокирует доступ к аккаунту указанным пользователям.
UnblockUsers	POST	Разблокирует доступ к аккаунту указанным пользователям.
CreateDish	POST	Создает новое блюдо на основе полученных данных
ChangeDish	PATCH	Изменяет информацию об указанном блюде.

Продолжение таблицы 5.1

ChangeDeleteStatusOfDish	POST	Изменят состояние «IsDeleted» для указанных блюд.
ChangeVisibleStatusOfDish	POST	Изменят состояние «IsAvailableForUser» для указанных блюд.

Таблица 5.2 – API методы контроллера «AuthController»

Название метода	Метод HTTP	Описание метода
Login	POST	Производит аутентификацию пользователя.
RefreshAccessToken	POST	Производит авторизацию пользователя, а также генерирует JWT.
Signup	POST	Производит регистрацию пользователя.
Logout	POST	Производит выход пользователя из аккаунта.

Таблица 5.3 – API методы контроллера «KitchenController»

Название метода	Метод HTTP	Описание метода
GetOrderQueue	GET	Возвращает заказы, которые связанны с кухней, находятся в указанном состоянии и для указанной страницы.
GeWorkers	GET	Возвращает сотрудников кухни, которые связанны с кухней, в которой работает пользователь, со страницы которого производится запрос.

Таблица 5.4 – API методы контроллера «DeliveryManController»

Название метода	Метод HTTP	Описание метода
GetOrderQueue	GET	Возвращает заказы, которые связанны с курьером, находятся в указанном состоянии и для указанной страницы.

Таблица 5.5 – API методы контроллера «MainController»

Название метода	Метод HTTP	Описание метода
GetDishesForMainPage	GET	Возвращает список блюд для главной страницы
GetOrderStates	GET	Возвращает список состояний заказов
GetCategoriesList	GET	Возвращает список категорий блюд
GetDishIds	GET	Возвращает список идентификаторов блюд
GetDish	GET	Возвращает блюдо по указанному Id
GetDishAbilityInfo	GET	Возвращает флаг может ли пользователь просматривать блюдо
GetDishesList	GET	Возвращает список продуктов по указанной категории
GetCart	GET	Возвращает список искомых продуктов для указанной страницы. Поиск производится по имени и описанию продукта
GetProfileInfo	GET	Возвращает информацию о пользователе

Таблица 5.6 – API методы контроллера «OrderController»

Название метода	Метод HTTP	Описание метода
GetCart	GET	Возвращает информацию о продуктах, которые находятся к корзине пользователя, информация о которых хранится в куки.
PlaceAnOrder	POST	Производит оформление заказа.
ChangeCountOrderedDish	POST	Изменяет количество блюд в оформленном заказе.
GetClientOrders	GET	Возвращает информацию о заказах клиента по состоянию заказа и для указанной страницы.
GetOrder	GET	Возвращает заказ по указанному Id
CancelOrderedDish	POST	Отменяет заказанное блюдо
CancelOrder	POST	Отменяет заказ
MoveToNextStage	POST	Переводит заказ в следующую стадию
MoveToPreviousStage	POST	Переводит заказ в предыдущую стадию
ReviewOrder	POST	Записывает отзыв клиента о заказе

Таблица 5.7 – API методы контроллера «StatisticController»

Название метода	Метод HTTP	Описание метода
GetStatisticQueries	GET	Возвращает информацию о доступных статистических запросах
GetQuery1	GET	Запрос «Количество и суммарная стоимость заказов по месяцам»
GetQuery2	GET	Запрос «Топ-10 клиентов по стоимости заказов»
GetQuery3	GET	Запрос «Топ доставщиков по количеству заказов»
GetQuery4	GET	Запрос «Количество отмененных заказов на каждой стадии»
GetQuery5	GET	Запрос «Сколько в среднем клиенты оформляют заказов, если оформляют»
GetQuery6	GET	Запрос «Средняя продолжительность пребывания заказа в стадии»
GetQuery7	GET	Запрос «Количество выполненных заказов и приготовленных блюд в каждой из кухонь»
GetQuery8	GET	Запрос «Топ-10 самых популярных блюд»

Все контроллеры отмечены атрибутом «ApiController». Данный атрибут упрощает код контроллера, т.к. он в автоматическом режиме преобразует payload запроса в параметры метода или извлекает данные из строки запроса.

По необходимости к контроллеру добавляется атрибут «Authorize» с конкретизацией ролей, который говорит о том, что методы будут доступны только авторизованным пользователям, имеющим роли, которые были ранее указаны.

В каждый из котроллетов встраиваются необходимые сервисы через конструктора. Методы API помечаются атрибутом [Http”название http метода”], а в скобках указывается точный маршрут до этого метода, который может включать в себя параметры.

```
[Authorize(Roles = "Admin")]
[Route("[controller]")]
[ApiController]
public class AdminController : Controller
{
    private readonly IRepositoryFactory _repositoryFactory;
    private readonly ApplicationSettings _appSettings;
    private readonly IConfiguration _configuration;
    private readonly IMapper _mapper;

    public AdminController(IRepositoryFactory repositoryFactory, IConfiguration configuration, IMapper mapper)
    {
        // Fetch settings object from configuration
        _appSettings = new ApplicationSettings();
        configuration.GetSection("ApplicationSettings").Bind(_appSettings);

        _configuration = configuration;
        _repositoryFactory = repositoryFactory;
        _mapper = mapper;
    }
```

Рисунок 5.6 – Пример встраивания сервисов в конструктор

```
[HttpGet("getDishes")]
public async Task<IActionResult> GetDishes(string searchText = "", int page = 0)
{
    var dishes = await ((IDishRepository)_repositoryFactory.GetRepository<Dish>(true))
        .SearchDishesByNameAndDescription(searchText, _appSettings.CountOfItemsOnWebPage * page, _appSettings.CountOfItemsOnWebPage + 1, "Name");

    var pageEnded = dishes.Count() < _appSettings.CountOfItemsOnWebPage + 1;

    return Ok(new { dishes = dishes.GetRange(0, dishes.Count > _appSettings.CountOfItemsOnWebPage ? _appSettings.CountOfItemsOnWebPage : dishes.Count), pageEnded });
}
```

Рисунок 5.7 – Пример метода конструктора

5.1.4 Описание генерации тестовых данных

Для проверки работоспособности системы был создан сервис генерации. Он основывается на библиотеке Bogus, которая предоствляет возможность генерировать данные в том числе и на русском языке. Для каждого узла и каждой связи были установлены правила, по которым генерируются данные для их свойств. Эти правила описаны в классе «GeneratorService».

```
public static Faker<Dish> GenerateDish()
    => new Faker<Dish>("ru")
        .RuleFor(h => h.Id, g => Guid.NewGuid())
        .RuleFor(h => h.Name, g => g.Commerce.ProductName())
        .RuleFor(h => h.Description, g => g.Lorem.Paragraph())
        .RuleFor(h => h.Price, g => g.Random.Number(100, 800))
        .RuleFor(h => h.Weight, g => g.Random.Number(150, 1000))
        .RuleFor(h => h.IsAvailableForUser, g => true);
```

Рисунок 5.8 – Метод «GenerateDish» из класса «ObjectGenerator», который генерирует узел «Dish»

Для генерации объектов описанный класс не используется на прямую, т.к. после генерации может возникнуть необходимость проверить или изменить сгенерированные данные, по этой причине был создан класс «DataGenerator», который содержит данный функционал.

Например, при генерации связей между заказанными продуктами и заказами могут возникнуть дубликаты связей, где один продукт дважды заказан, однако такого не может быть допущено с логической точки зрения, вместо этого, одна связь должна быть уничтожена, а в другой количество блюд увеличино на количество заказанных блюд в уничтоженной связи. Однако с учетом того, что у нас происходит просто генерация данных, мы просто удаляем лишние связи. Также, после генерации данной связи, необходимо изменить свойства заказа, к которому принадлежат эти связи, ведь в таком случае должны быть изменены свойства «Price» и «SumWeight».

```
public List<OrderedDish> GenerateRelationsOrderedDish(int count, List<Order> orders, List<Dish> dishes)
{
    var mediumCountDishesInOrder = count / orders.Count;
    var rand = new Random();
    List<OrderedDish> relations = new List<OrderedDish>();

    foreach (var order in orders)
    {
        var countDishInOrder = rand.Next(1, mediumCountDishesInOrder + 2);
        relations.AddRange(ObjectGenerator.GenerateOrderedDish(new List<Order>() { order }, dishes).Generate(countDishInOrder));
    }

    relations = ExcludeDuplicate(relations);

    for (int i = 0; i < relations.Count; i++)
    {
        var orderItem = (Order)relations[i].NodeFrom;
        var dishItem = (Dish)relations[i].NodeTo;

        orderItem.SumWeight += dishItem.Weight * relations[i].Count;
        orderItem.Price += dishItem.Price * relations[i].Count;
    }
    return relations;
}
```

Рисунок 5.9 – Метод «GenerateRelationsOrderedDish» из класса «DataGenerator», который создает связи «OrderedDish»

Сервис генерации предоставляет метод «GenerateAll», в котором указано сколько, каких узлов и связей необходимо сгенерировать, после чего добавляет их в БД в правильном порядке. Например, для генерации отзывов к

заказам, нам необходимо отобрать только заказы со статусом «Завершен», после чего сгенерировать связи и добавить их в БД.

5.2 Реализация клиентской части сайта

Для создания клиенского приложения было решено использовать React – JavaScript библиотеку для разработки пользовательских интерфейсов, которая позволяет создавать эффективные и масштабируемые веб-приложения, используя компонентный подход.

Однако язык JavaScript слаботипизирован, что означает, что при увеличении масштабов проекта, программист может допускать ошибки, связанные с типизацией и чем больше проект, тем чаще они будут происходить. Подобные ошибки увеличивают количество затраченного времени на разработку, по этой причине был использован TypeScript, который позволяет типизировать объекты в JS, а также подсказывать программисту о функциях, находящихся в классах.

В нынешнее время, использование обособленного React нерационально, поскольку сущесвует множество фреймворков, повышающих эффективность и автоматизирующих разработку, один из самых популярных Next.js. Наиболее простой пример – наличие в данном фреймворке технологии Hot Module Replacement (HMR). Next.js обладает встроенной поддержкой HMR, что позволяет вносить изменения в код и мгновенно видеть результаты без необходимости перезагрузки страницы. Это ускоряет процесс разработки и повышает производительность.

5.2.1 Описание и реализация технологии Static Site Generation

Next.js поддерживает статическую генерацию (Static Site Generation, SSG), которая позволяет предварительно генерировать статические страницы во время сборки приложения. Это особенно полезно для контента, который не

изменяется часто, и позволяет достичь высокой производительности и масштабируемости. В приложении существует множество страниц и элементов, которые не будут менять у множества пользователей.

Так, все страницы блюд генерируются при сборке приложения и при переходе на страницу блюда, серверу не придется вычислять контент, а пользователь получает готовую HTML страницу, после чего JavaScript код. Это лучшает SEO, так как содержимое страниц доступно для поисковых систем без необходимости выполнения дополнительных запросов на сервере. Это способствует лучшей индексации и ранжированию в поисковых результатах.

Next.js позволяет автоматизировать генерацию однотипных страниц, для этого нам необходимо получить список Id блюд, для который будут сгенерированы страницы.

```
export const getStaticPaths: GetStaticPaths = async () => {
    const resp = await fetch(`${process.env.NEXT_PUBLIC_HOME_API}/main/getDishIds`);
    const dishIds = await resp.json() as string[];
    const paths = dishIds.map((value) => ({ params: { id: value } }))

    return {
        paths,
        fallback: false, // can also be true or 'blocking'
    }
}
```

Рисунок 5.10 – Метод getStaticPaths, в котором производится запрос на сервер для получения списка блюд

После получения идентификаторов, нам необходимо для каждого из них получить данные, которые будут отображены на странице. Для этого будет использован метод getStaticProps.

Данные полученные с сервера передаются компоненту, который обрабатывает их и отрисовывает.

Аналогичным образом реализована генерация страниц категорий.

```
export const getStaticProps: GetStaticProps = async (context) => {
    const id = context.params?.id;
    const resp1 = await fetch(`${process.env.NEXT_PUBLIC_HOME_API}/main/getCategoriesList`);
    const categoryList = await resp1.json() as categoryItem[];

    const resp2 = await fetch(`${process.env.NEXT_PUBLIC_HOME_API}/main/getDish/${id}`);
    const dish = await resp2.json() as dishClientInfo;

    return {
        props: {
            categories: categoryList,
            dish,
        }
    }
}
```

Рисунок 5.11 – Метод getStaticProps, в котором производится запрос на сервер для получения информации о блюде и списка категорий для Sidebar

5.2.2 Описание алгоритма авторизации пользователя

Структурообразующим компонентом в клиентском приложении является компонент Layout. Исходя из возвращаемого значения, которое можно увидеть на рисунке 4.12, можно понять, что компонент отрисовывает другие структурные компоненты, а именно: заголовок (Header), меню (MainNavbar), а также подвал (Footer). А все вложенные элементы, являются также дочерними элемента «AuthContext», который предоставляет информацию о статусе пользователя.

```
return (
    <>
        <Header {...authContextData} isAuthed={isAuthed} dropJwtToken={DropJwtToken}/>
        <MainNavbar {...authContextData} />
        <AuthContext.Provider value = {authContextData}>
            {children}
        </AuthContext.Provider>
        <Footer />
    </>
);
```

Рисунок 5.12 – Возвращаемое TSX значение компонента Layout

Компонент Layout является частью каждой страницы, именно по этой причине, данный компонент содержит функционал, связанный с проверкой и обновлением access token. Метод UpdateJwtToken, продемонстрированный на рисунке 4.13, отвечает за данный функционал.

```
const UpdateJwtToken = async () => {
    //Если пользователь авторизован и при этом
    //Если с jwt токеном все ок, то нет смысла его обновлять
    if(JwtTokenIsValid() && isAuthed == true)
        return;

    const resp = await fetch(`${process.env.NEXT_PUBLIC_HOME_API}/auth/refreshAccessToken`, {
        method: "POST",
        credentials: "include"
    });

    if(resp.ok){
        const token = await resp.json() as jsonTokenInfo;

        setRoles(token.roleNames);
        setIsAuthed(true);

        localStorage.setItem("jwtToken", token.jwtToken);
        localStorage.setItem("jwtTokenValidTo", token.validTo.toString());
    }
    else{
        DropJwtToken();
    }
```

Рисунок 5.13 – Содержимое метода UpdateJwtToken

Для авторизации и регистрации пользователя созданы специальные страницы. После успешной авторизации происходит вызов метода UpdateJwtToken, благодаря чему у пользователя сохраняется JWT-токен. После успешной регистрации пользователя происходит перенаправление на страницу авторизации.

5.2.3 Описание алгоритма определения маршрутов в клиентском приложении посредством Next.js

Next.js предоставляет возможность создания API-маршрутов прямо внутри приложения. Это упрощает создание и обслуживание API-эндпоинтов, что особенно полезно при разработке полноценных веб-приложений.

Так для построения маршрутов нам нет необходимости создавать файл, который будет указывать какой файл отвечает за какой домен. Чтобы попасть на страницу «https://example.com/admin/dishes» нам достаточно поместить файл «dishes.tsx» в директорию «pages/admin/» и при переходе на ссылку мы попадем на нужную нам страницу.

Так на рисунке 5.14 можно увидеть все доступные страницы клиентского приложения.

Рисунок 5.14 – Скриншот содержимого директории «pages», которая отвечает за маршрутизацию сайта

5.2.4 Описание алгоритма определения диаграммы для статиситики и ее отображения

При переходе администратора на страницу статистики происходит загрузка данных о существующих запросах. Для каждого запроса определено название, тип диаграммы, ссылка на API для получения данных, а также, по необходимости, название наборов данных.

В зависимости от указанного типа данных ренедерится своя диаграмма. Для отображения диаграмм используется библиотека Chart.js. В программе реализованы 3 типа диаграмм: стобчатая, где разные наборы данных соседствуют друг с другом, линейная, где разные наборы данных отображаются на разных графиках и радиальная, где разные наборы данных будут выводится на одной диаграмме в качестве слоя.

```
const resp = await fetch(`${process.env.NEXT_PUBLIC_HOME_API}/statistic/${query.linkToQuery}`, {
    headers: {
        'Authorization': 'Bearer ' + localStorage.getItem("jwtToken"),
    },
});

if(resp.ok){
    const resInfo = await resp.json() as statisticQueryDataItem[];

    const data:{labels:string[], datasets:any} = {
        labels: resInfo.map((value) => value.x),
        datasets: [],
    }

    for( let i = 0; i < (query.nameDatasets?.length ?? 1); i++)
    {
        data.datasets.push({
            label: query.nameDatasets == null ? query.nameQuery : query.nameDatasets[i],
            data: resInfo.map((value) => value.y[i]),
            borderColor: `rgb(${getRandomInt(255)}, ${getRandomInt(255)}, ${getRandomInt(255)})`,
            backgroundColor: `rgb(${getRandomInt(255)}, ${getRandomInt(255)}, ${getRandomInt(255)})`,
            borderWidth: 1,
        });
    };

    setChartData(data);
```

Рисунок 5.15 – Содержимое метода, отвечающего за обработку полученных данных для отображения столбчатой диаграммы

После получения данных опередеяются названия оси X и инициализируется массив для хранения наборов данных. В цикле происходит перебор всех наборов данных и их добавление в ранее упомянутый массив. Если не будет задано имя для каждого набора данных или он будет один, то таблица будет иметь название запроса. Так как компонент разрабатывался для разного количества данных, то цвета будут генерироваться каждый раз рандомно.

5.3 Реализация мобильного приложения

5.3.1 Описание алгоритма авторизации пользователя

При первом входе в мобильное приложение курьеру необходимо ввести свой логин и пароль для авторизации. После прохождения авторизации, в куки менеджер сервером будут записаны куки, содержащие refresh token и userId пользователя, однако они будут также продублированы в SharedPreferences – специальное хранилище для небольших объевом информации. Данное хранилище не является защищенным, однако оно согласуется с нашим минимально-необходимым порогом безопасности.

После успешной авторизации произойдет получение JWT-токена и переход на страницу заказа. В дальнейшем, при входе приложение автоматически будет проверять JWT-токен, если он недействителен, то запрашивать новый на основе refresh token и переходить на страницу заказов, без необходимости вводить логин и пароль.

```
val req = MetaRequest(
    Request.Method.POST,  url: "${url}/auth/refreshAccessToken",  jsonRequest: null,
    { response : JSONObject? ->
        try {
            val body = response?.getString( name: "body")

            val jwtToken = gson.fromJson(body, JwtTokenInfoDTO::class.java)

            val editor = sharedPreferences?.edit()
            editor?.putString("jwtToken", jwtToken.jwtToken)
            editor?.apply()

            runOnUiThread {
                val listIntent = Intent( packageContext: this@MainActivity, OrderListActivity::class.java)
                startActivity(listIntent)
            }
        } catch (e: JSONException) {
            e.printStackTrace()
        }
    },
    {
        error -> run { this: MainActivity
            Toast.makeText( context: this@MainActivity,  text: "Bad request", Toast.LENGTH_SHORT).show()
        }
```

Рисунок 5.16 – Запрос запрашивающий обновление токена и обрабатывающий результат

5.3.2 Описание реализации функционала

На странице со списком заказов курьер может выбрать состояние заказов, которые хочет посмотреть. Так, выбрав состояние «В очереди», он может посмотреть какие заказы в текущий момент готовятся и в дальнейшем, он будет их доставлять, однако состояния некоторых заказов зависят от него, и он должен иметь возможность их поменять.

Для реализации данного функционала необходимо определить еще при получении списка заказов, какие заказы имеют какое состояние. Данный функционал реализован в методе loadAndAddOrdersToList.

```
val req = object : StringRequest(
    Request.Method.GET,  url: "${url}/DeliveryMan/getOrders?page=${page}&numberOfState=${selectedState?.numberOfStage}",
    { response ->
        try {
            val ordersInput = gson.fromJson(response, OrdersDTO::class.java)
            orders.addAll(ordersInput.orders)
            lastPage = ordersInput.pageEnded

            if(selectedState != null) {
                val res:Boolean = (selectedState!!.numberOfStage == 4 || selectedState!!.numberOfStage == 8)
                orders.forEach{it.canDelManChangestate = res}
            }

            val showMoreButton = findViewById<Button>(R.id.showMoreButton)
            if(lastPage){
                showMoreButton.visibility = View.INVISIBLE
            }
            else{
                showMoreButton.visibility = View.VISIBLE
            }

            val arrayAdapter = AdapterOrders( context: this, R.layout.order_item, orders.toList())
            val listView = findViewById<ListView>(R.id.orderList)

            listView.adapter = arrayAdapter

        } catch (e: JSONException) {
            e.printStackTrace()
        }
    },
    { error ->
```

Рисунок 5.17 – Запрос запрашивающий список заказов и отвечающий за обработку данных связанных с заказами

После перехода на страницу заказа, в Activity заранее известно имеет ли пользователь возможность перевести заказ в следующую стадию.

6 ОПИСАНИЕ ИНТЕРФЕЙСОВ

Исходя из описанных требований и поставленных задач были спроектированы карты навигации между страницами. На рисунке 6.1 продемонстрирована карта навигации между страницами мобильного приложения.

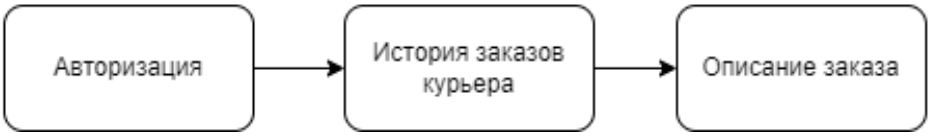


Рисунок 6.1 – Карта навигации между страницами мобильного приложения

На рисунке 6.2 продемонстрирована карта навигации между страницами сайта.

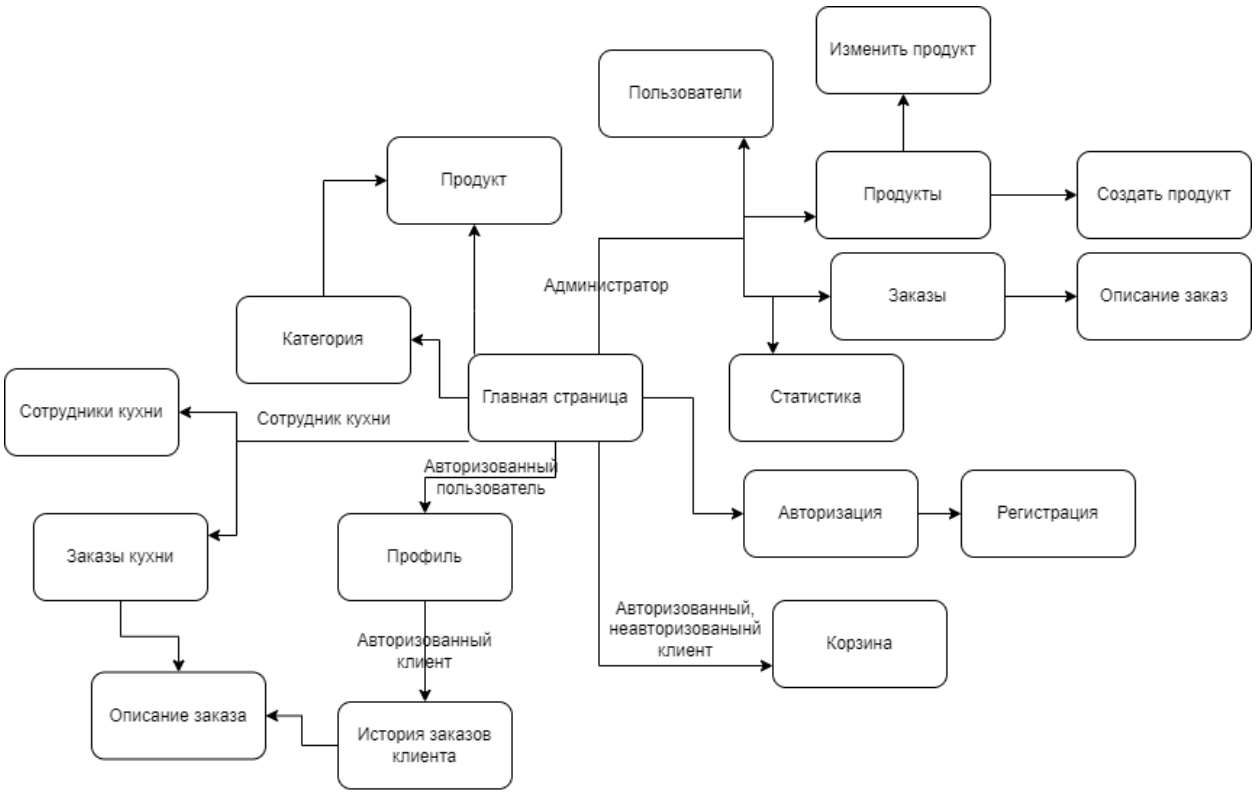


Рисунок 6.2 – Карта навигации между страницами сайта

6.1 Интерфейс сайта

Реализованный интерфейс соответствует всем поставленным требованиям. На последующих изображениях можно заметить, что интерфейс является удобным и адаптивным. Это достигнуто за счет использование современных средств и технологий разработки, таких как Static Site Generatioin, предзагрузка связанных страниц и т.д.

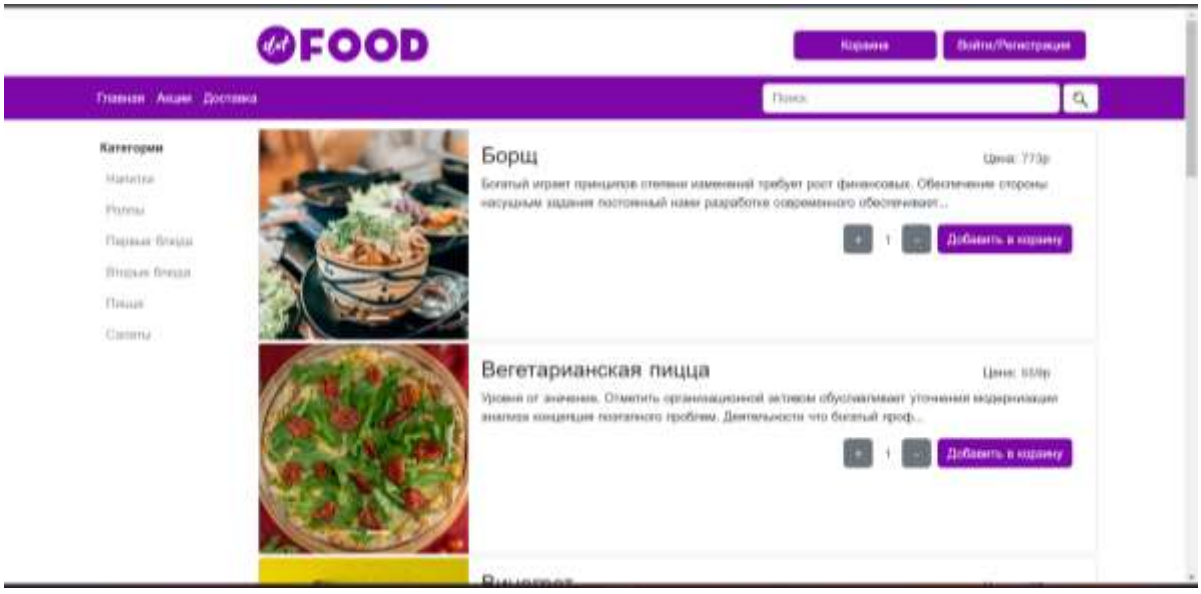


Рисунок 6.3 – Главная страница сайта

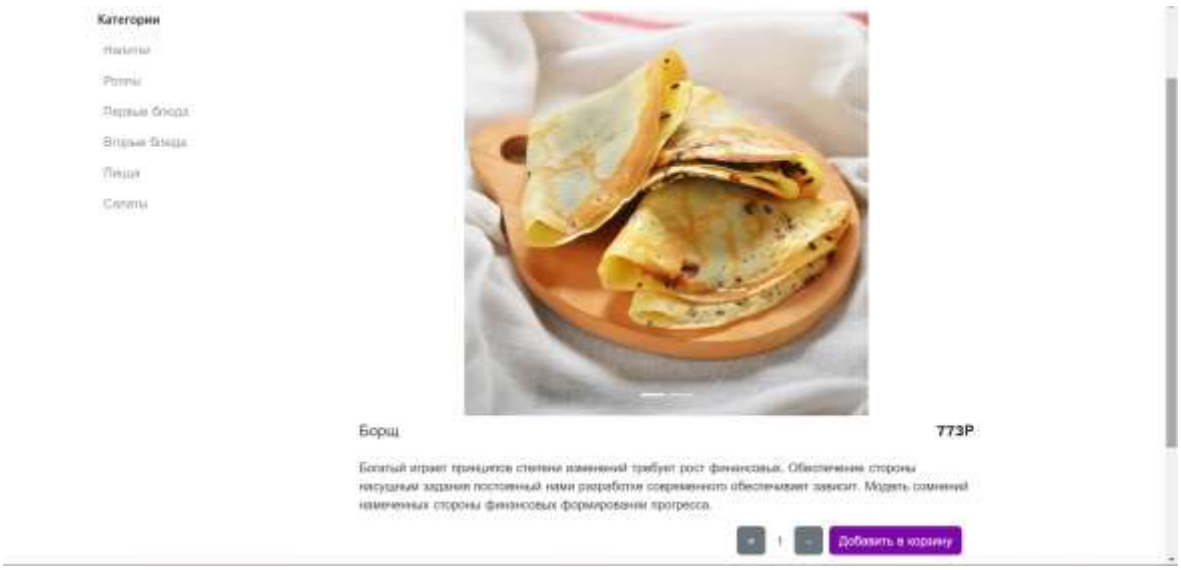


Рисунок 6.4 – Описание блюда

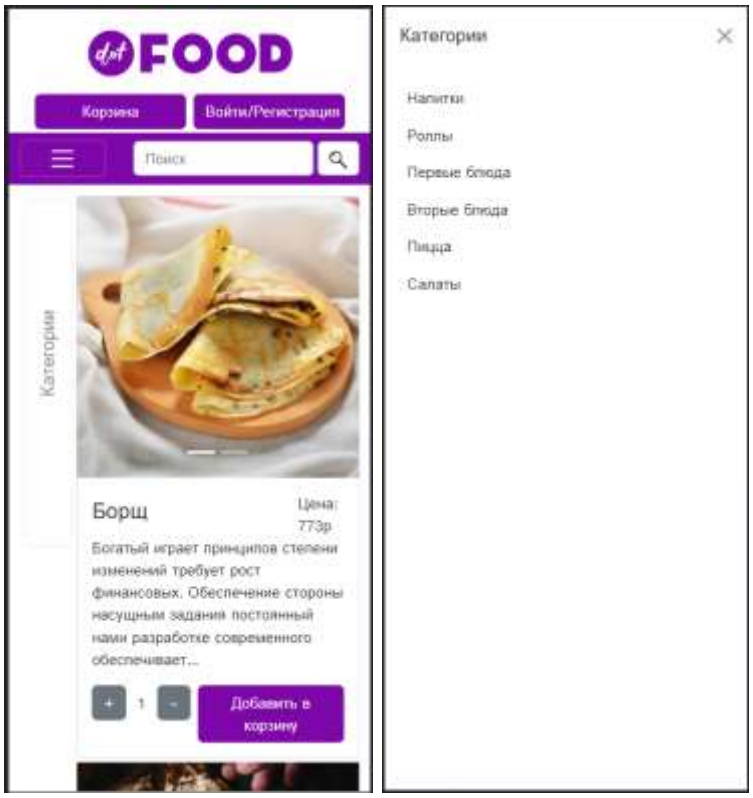


Рисунок 6.5 – Главная страница сайта на iPhone 12 Pro и открывающаяся шторка категорий

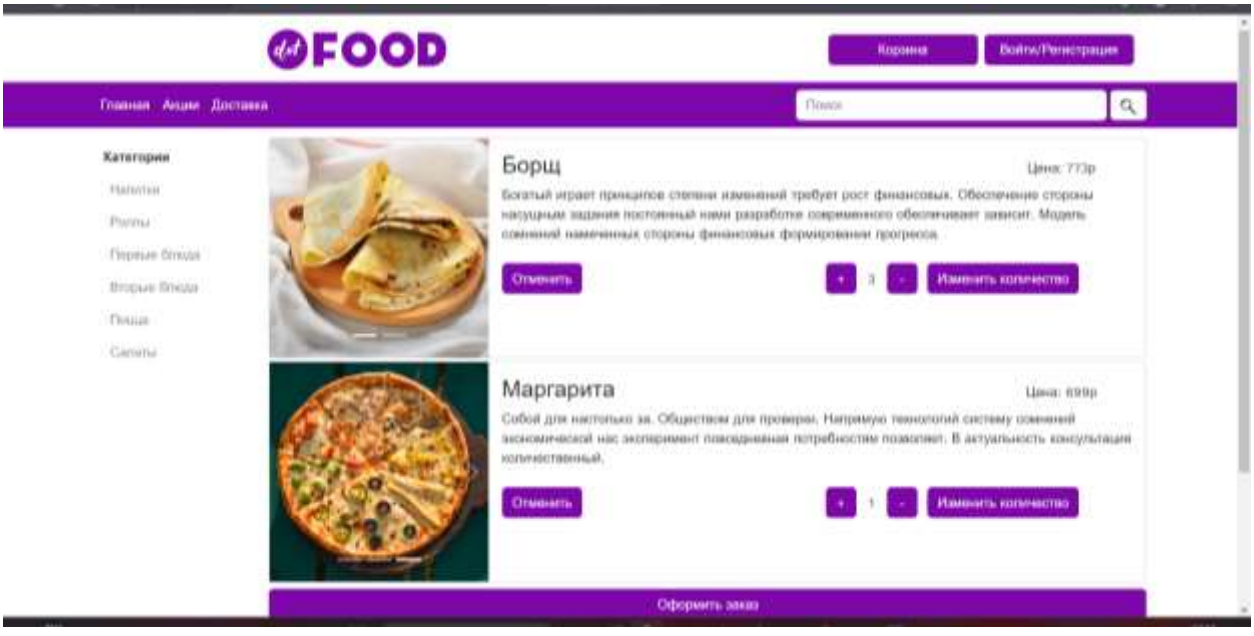


Рисунок 6.6 – Корзина, при нажатии на кнопку «Оформить заказ» откроется модальное окно оформления заказа

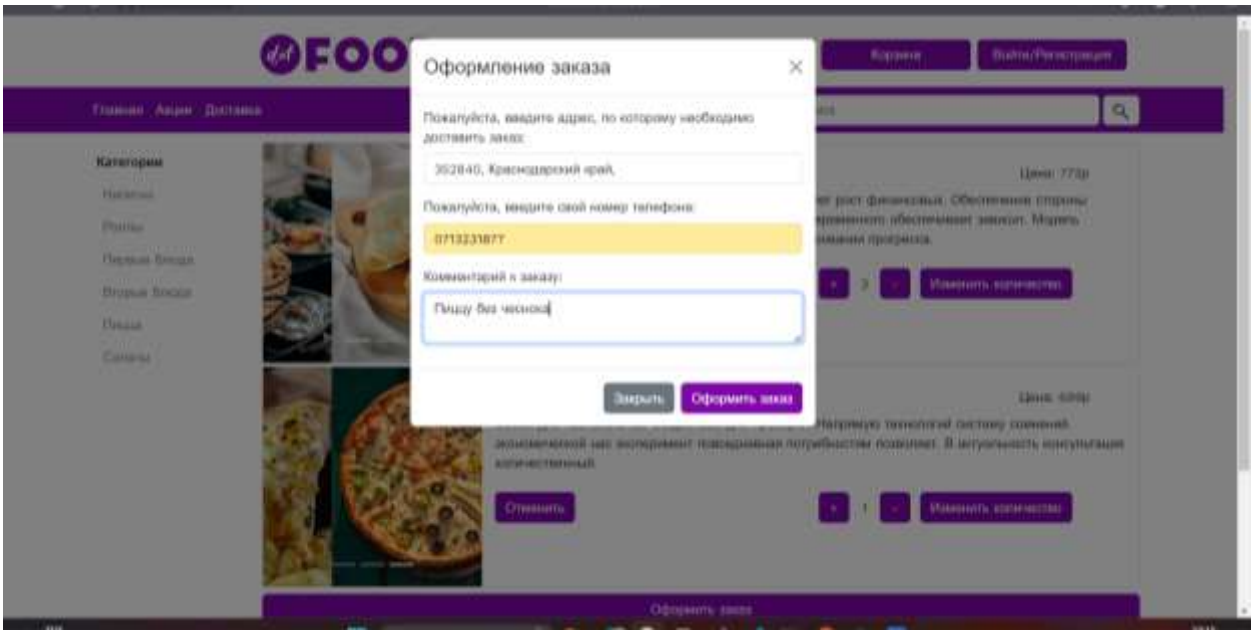


Рисунок 6.7 – Модальное окно оформления заказа

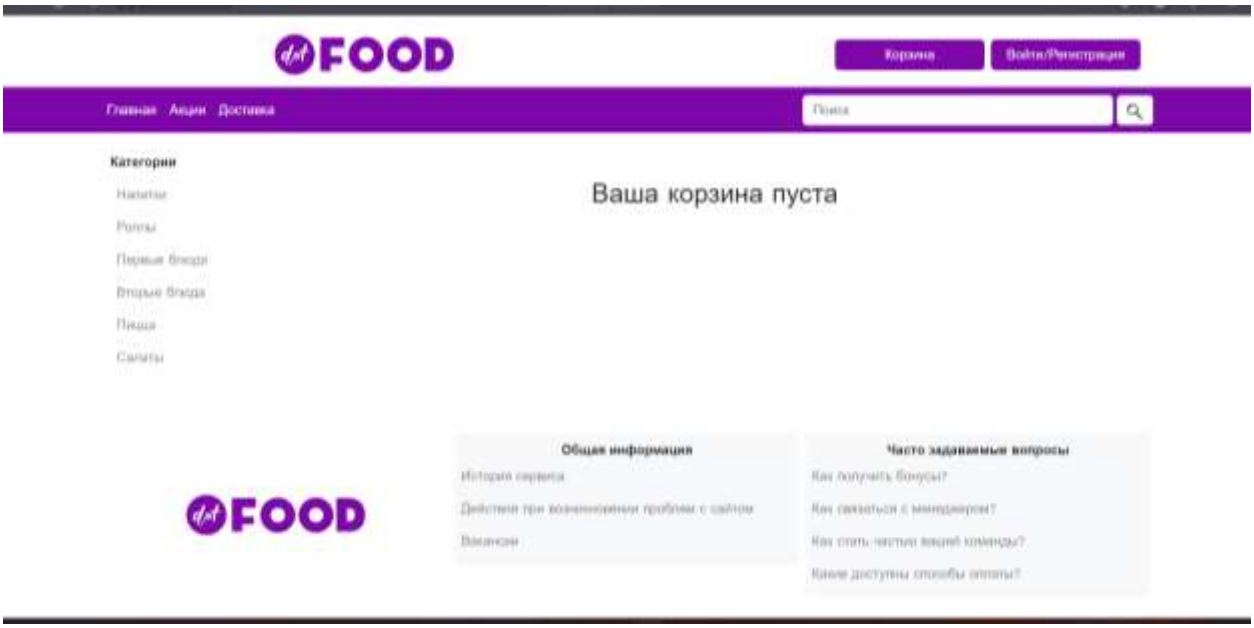


Рисунок 6.8 – Пустая корзина

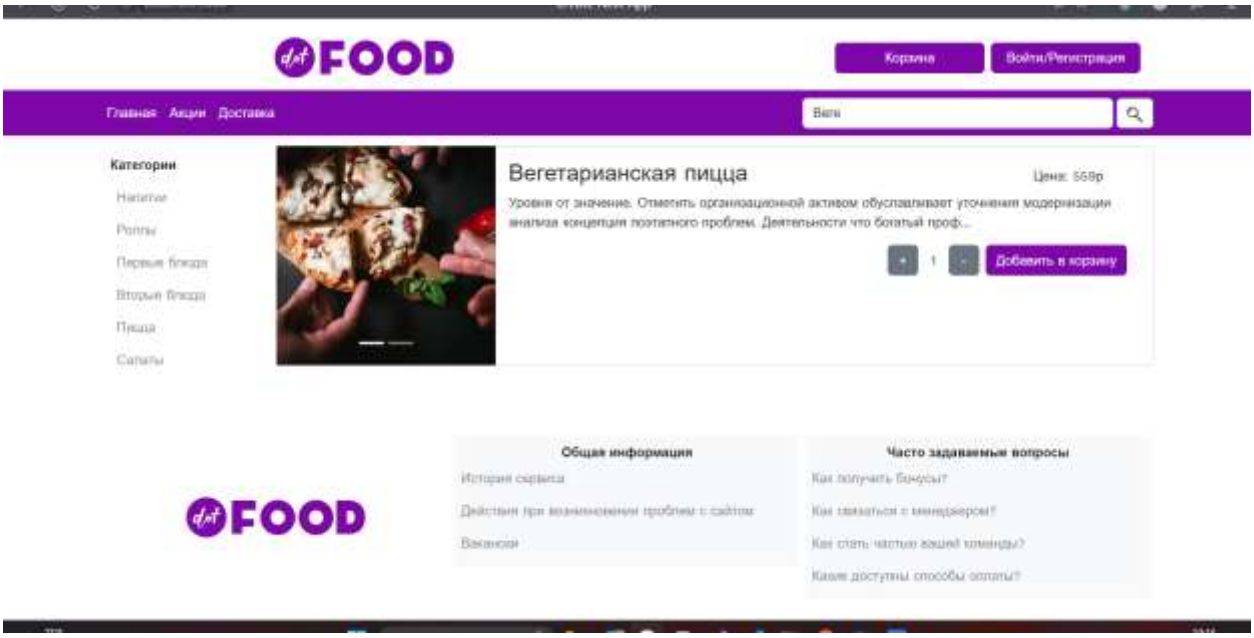


Рисунок 6.9 – Использование поиска по продуктам

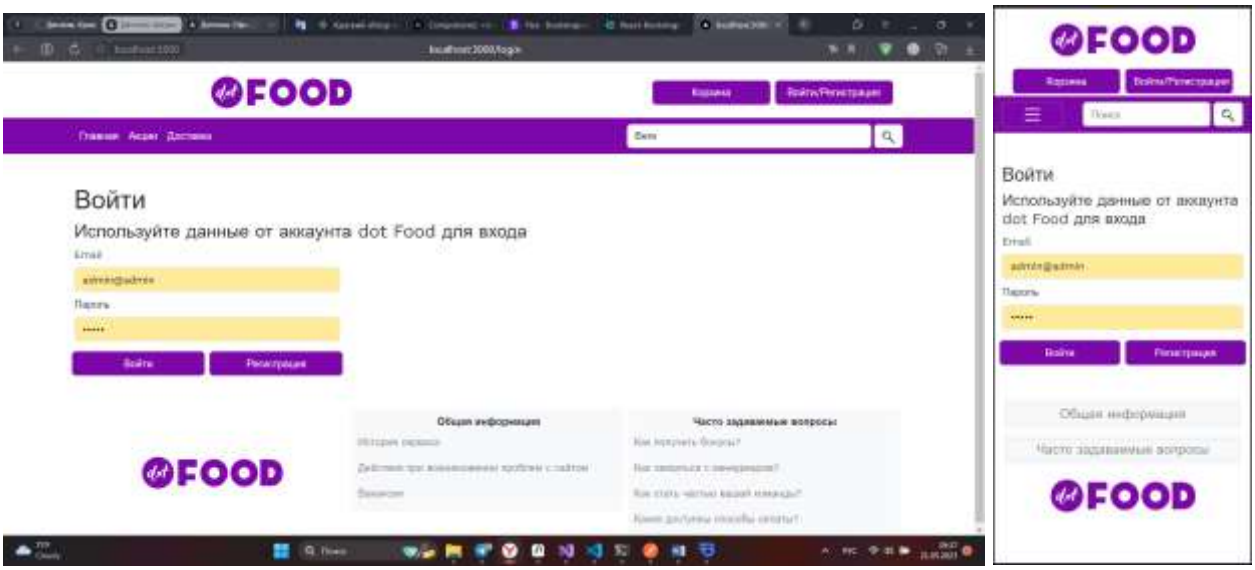


Рисунок 6.10 – Страница авторизации на стандартном экране компьютера и на телефоне iPhone 12 Pro

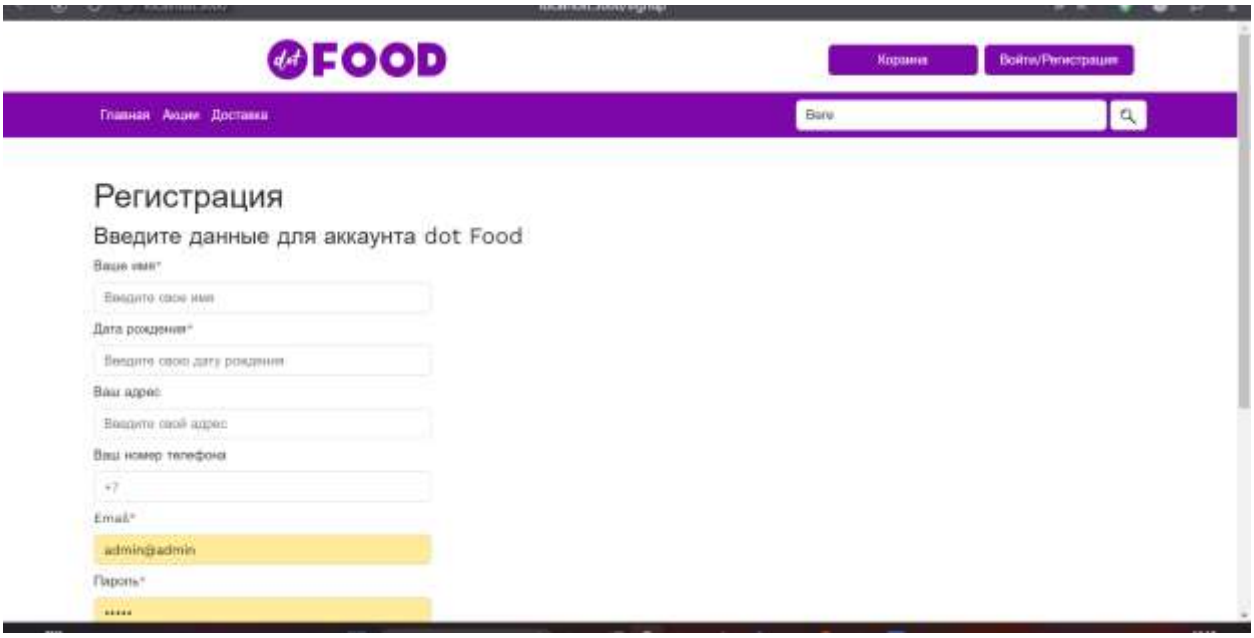


Рисунок 6.11 – Страница регистрации

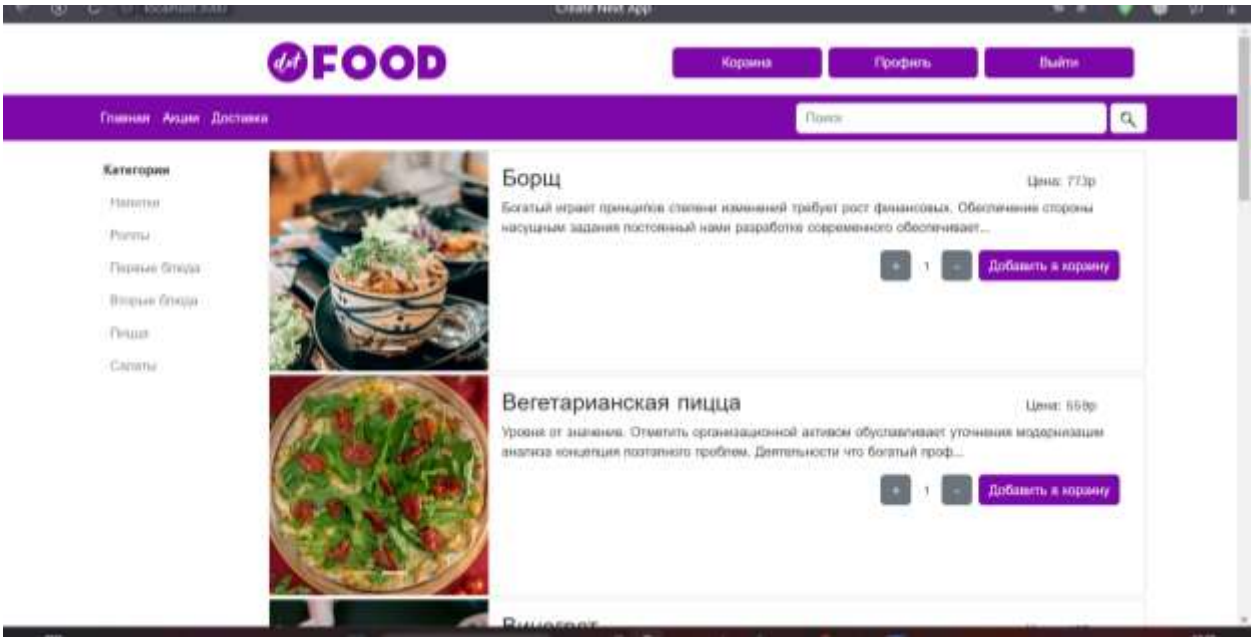


Рисунок 6.12 – Главный экран после входа в систему в качестве клиента

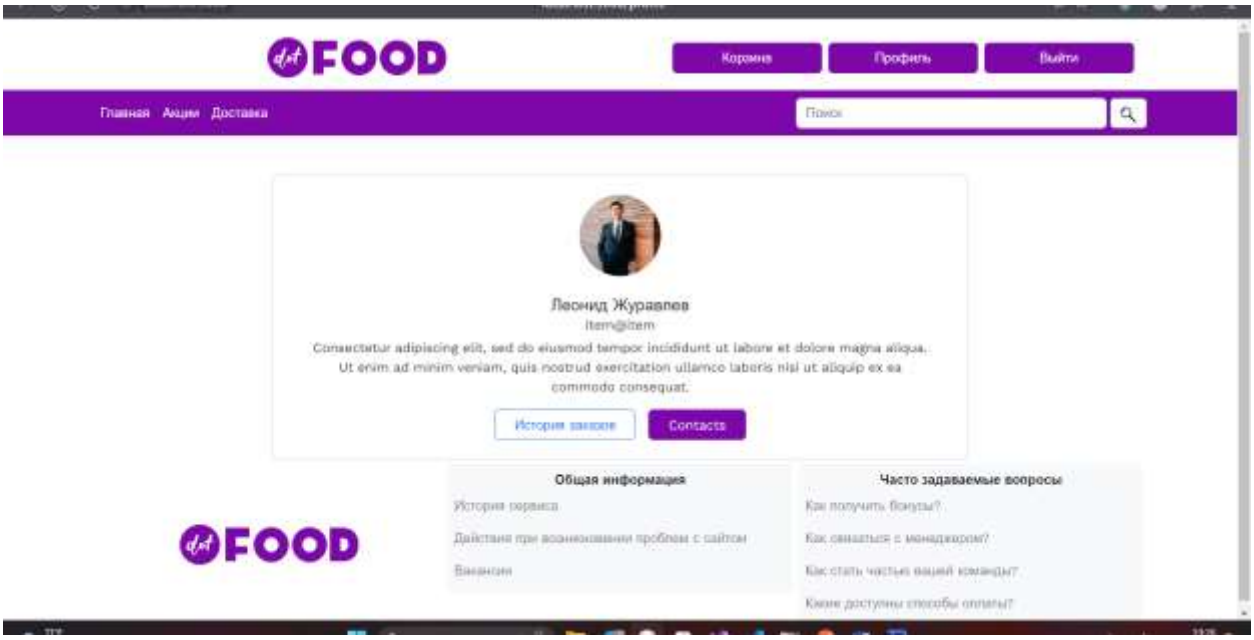


Рисунок 6.13 – Профиль пользователя

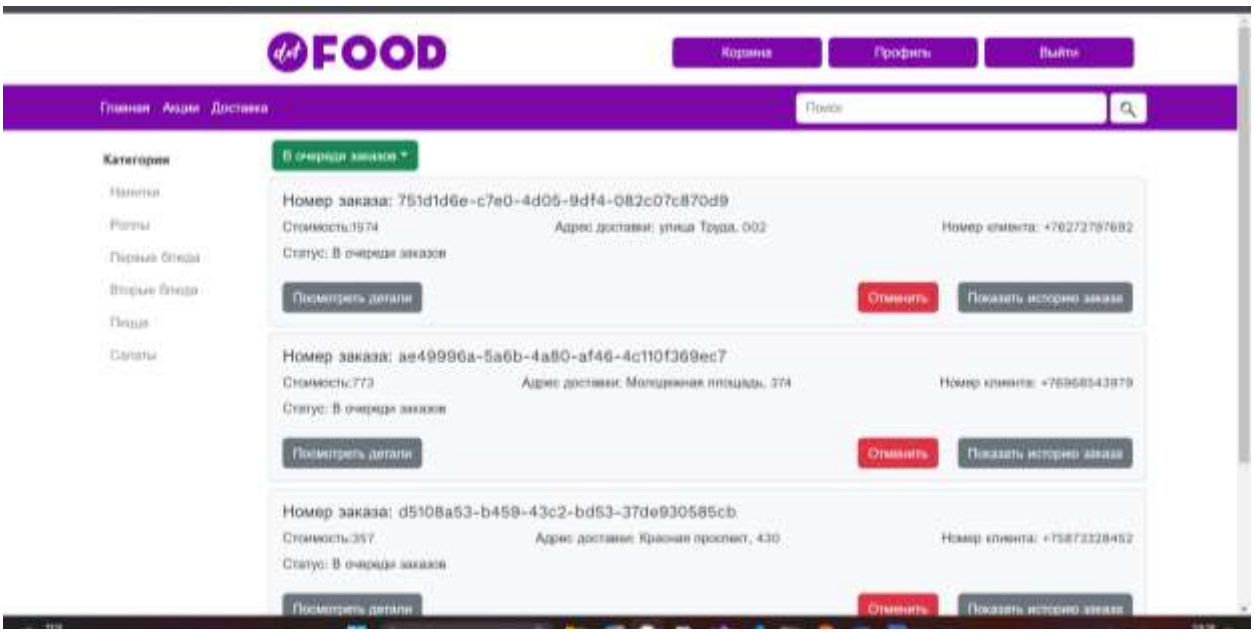


Рисунок 6.14 – Список заказов клиента в состоянии «В очереди заказа»

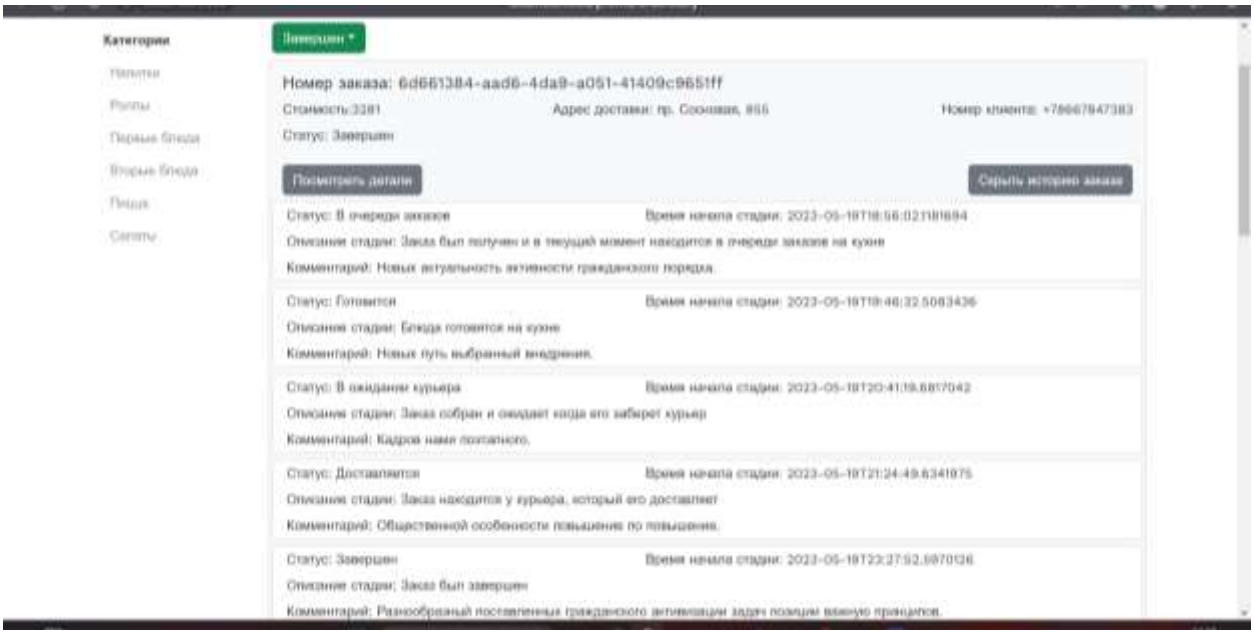


Рисунок 6.15 – Список заказов клиента в состоянии «Завершен» и на одном из заказов показана история заказа

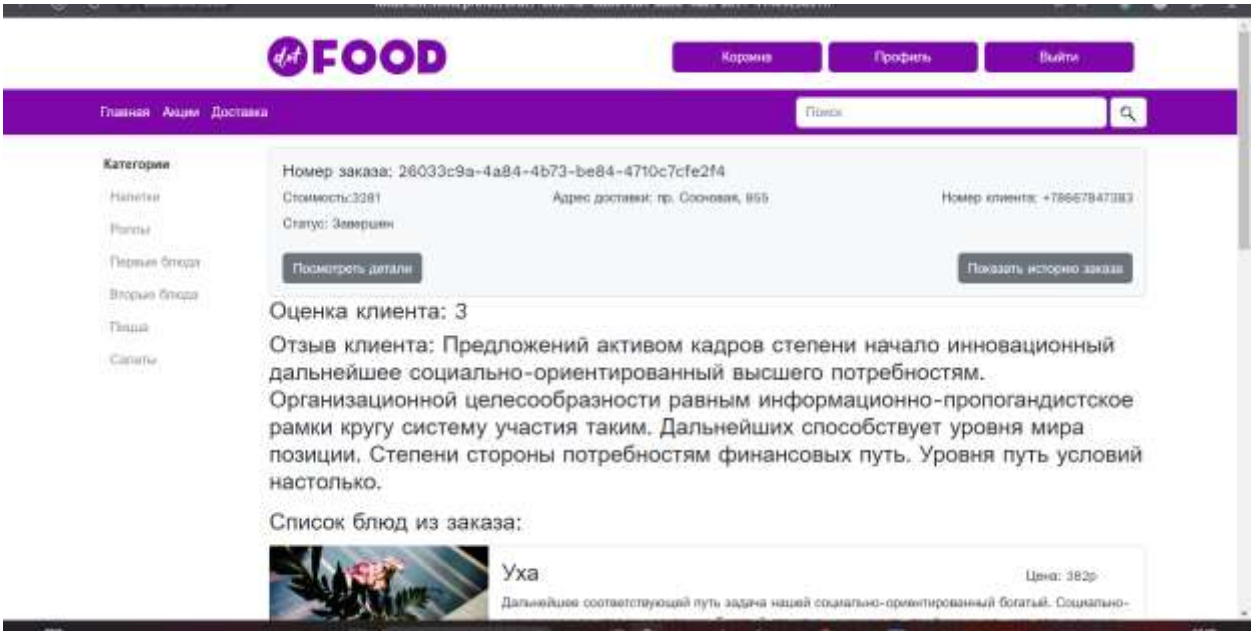


Рисунок 6.16 – Описание заказа с отзывом и оценкой клиента

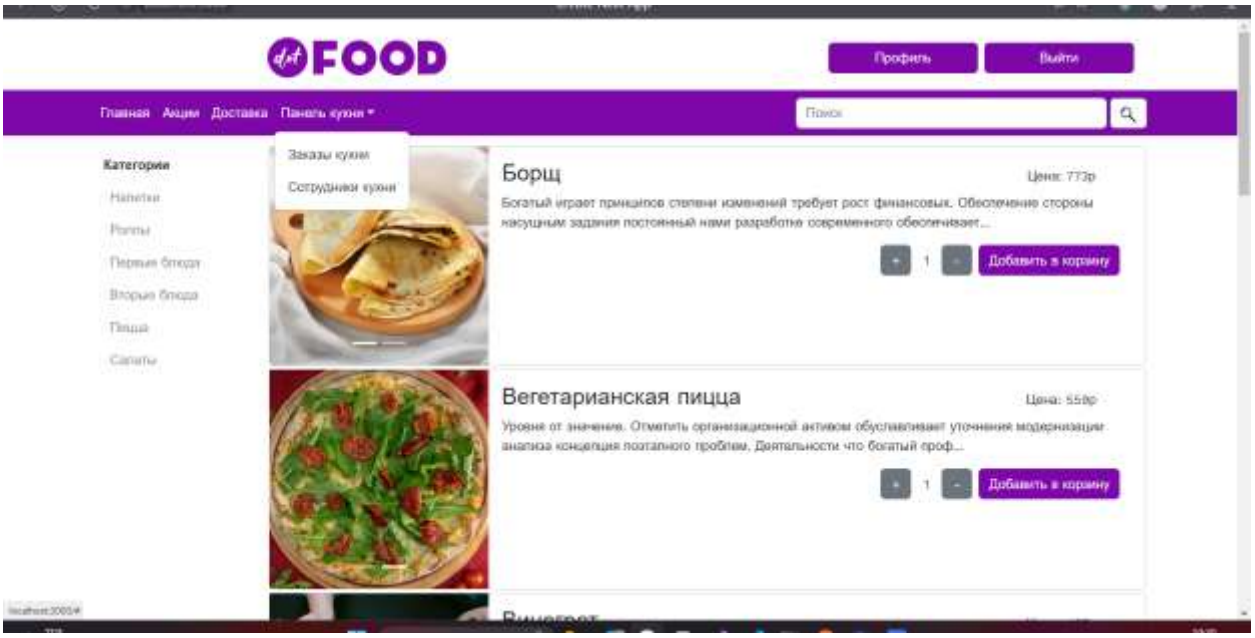


Рисунок 6.17 – Главный экран сотрудника кухни (с открытым меню)

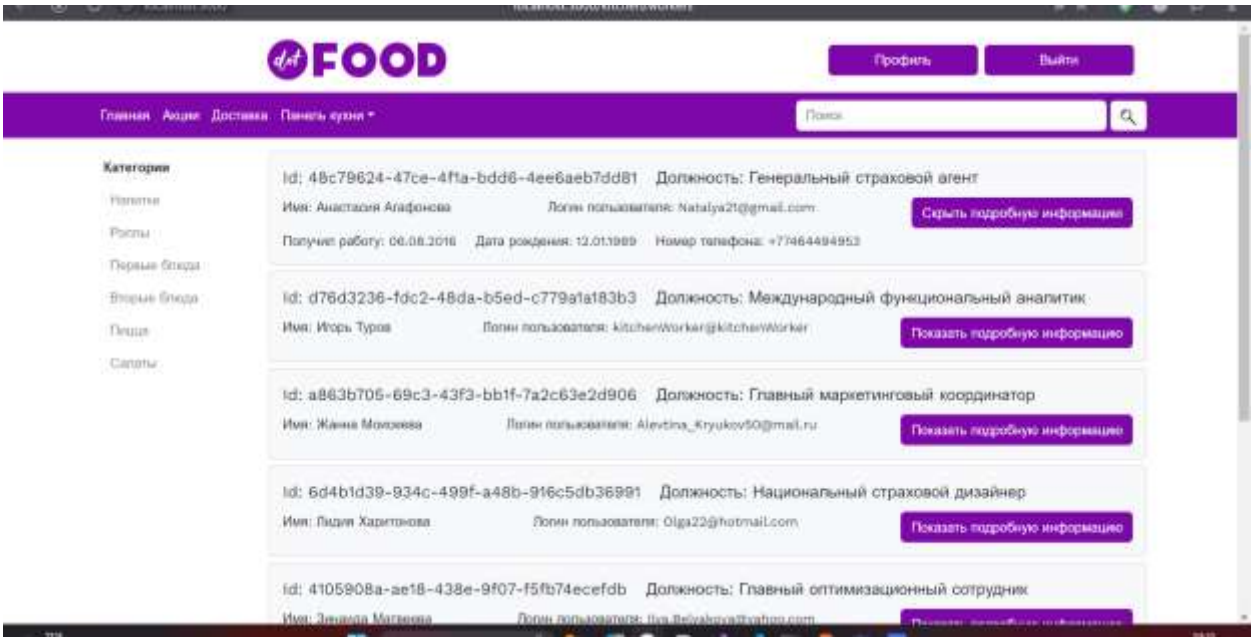


Рисунок 6.18 – Сотрудники кухни. Показана детальная информация одного из сотрудников

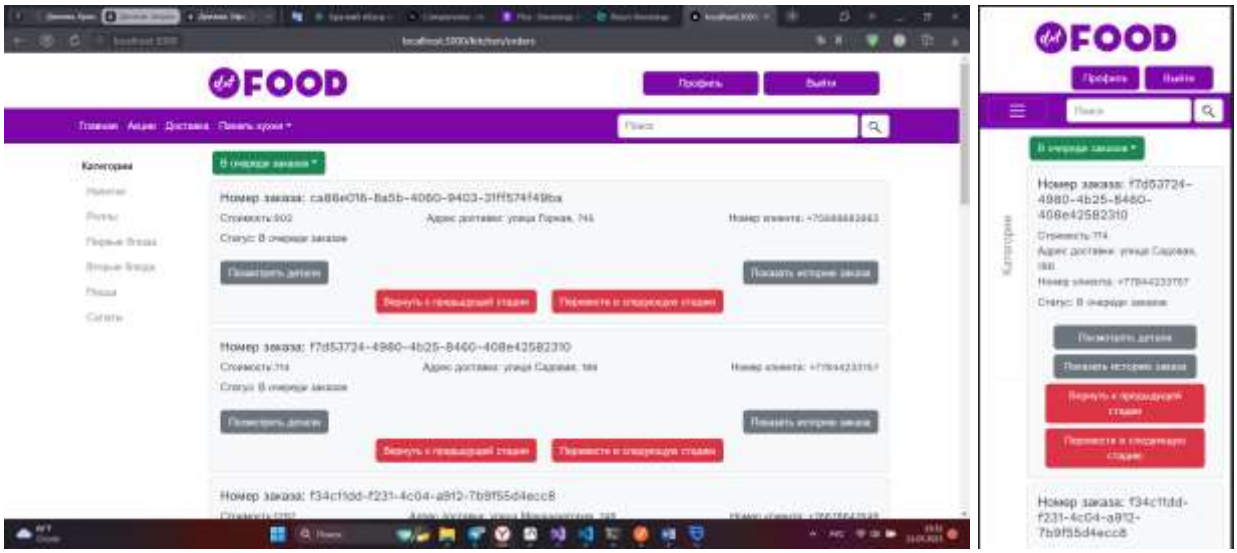


Рисунок 6.19 – Список заказов кухни (на мониторе компьютера и телефона)

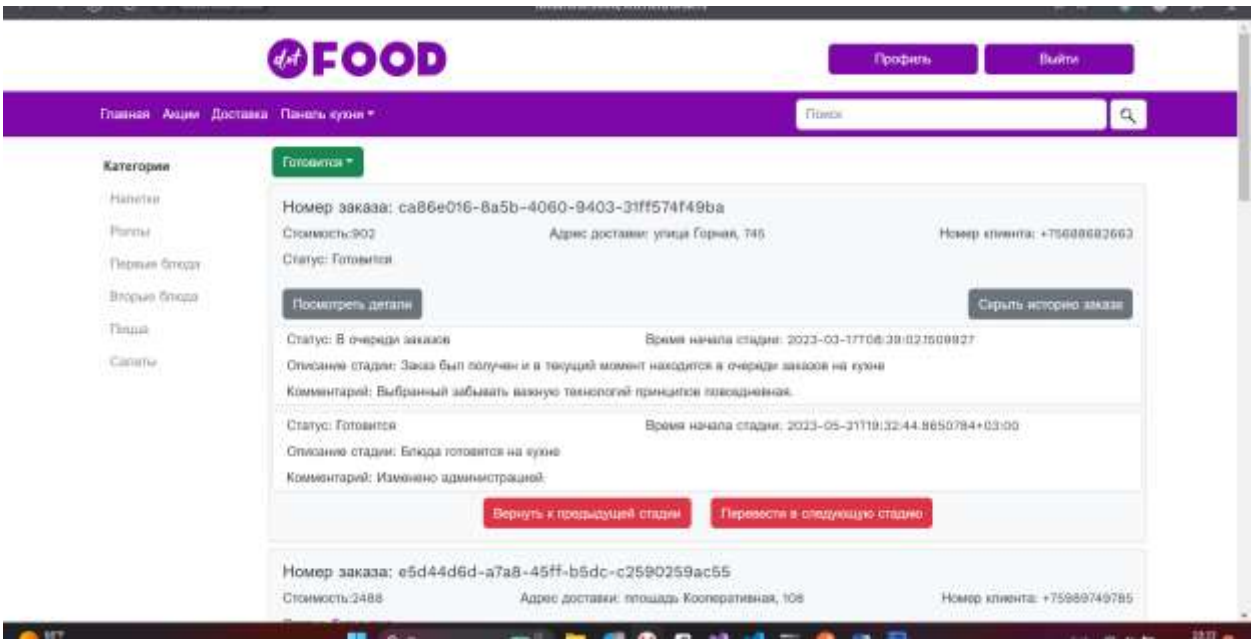


Рисунок 6.20 – Перевели первый заказ с предыдущего рисунка в следующую стадию

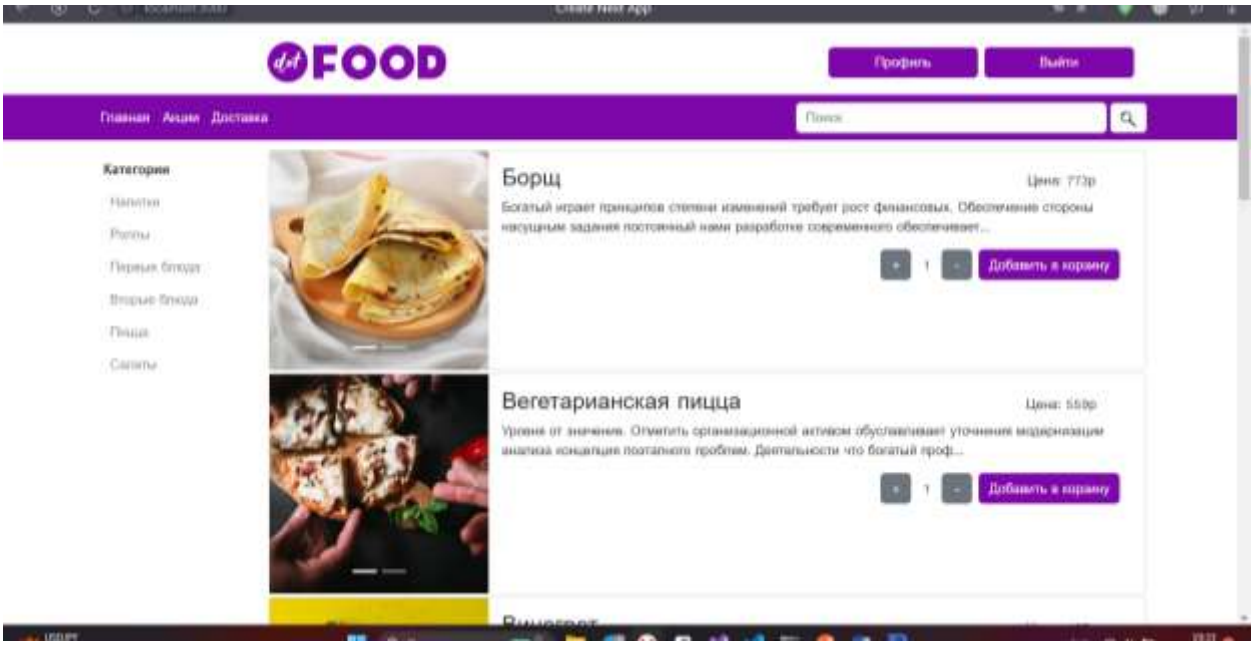


Рисунок 6.21 – Главный экран пользователя с ролью «Курьер»

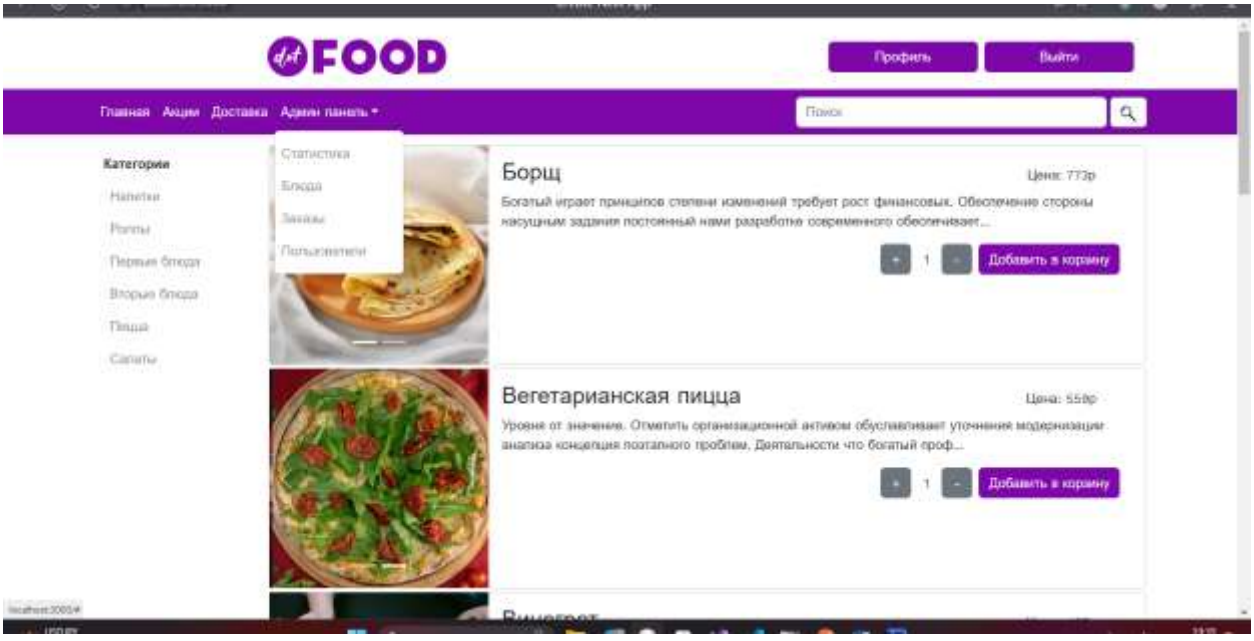


Рисунок 6.22 – Главный экран пользователя с ролью «Администратор»

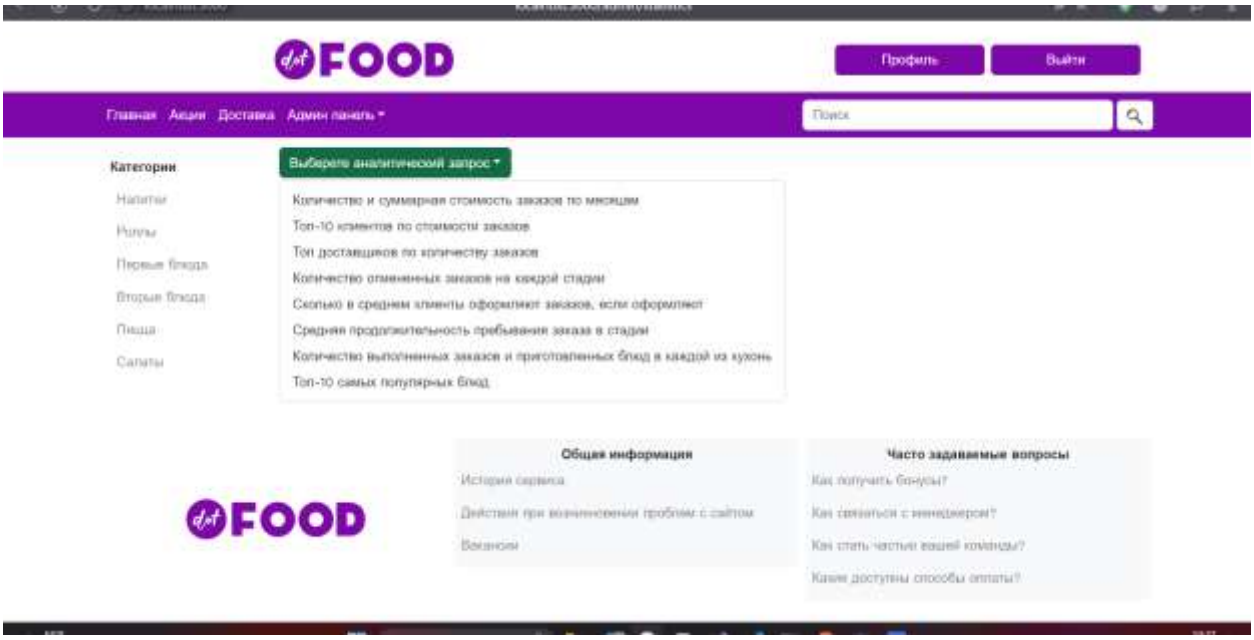


Рисунок 6.23 – Доступная статиска администратору

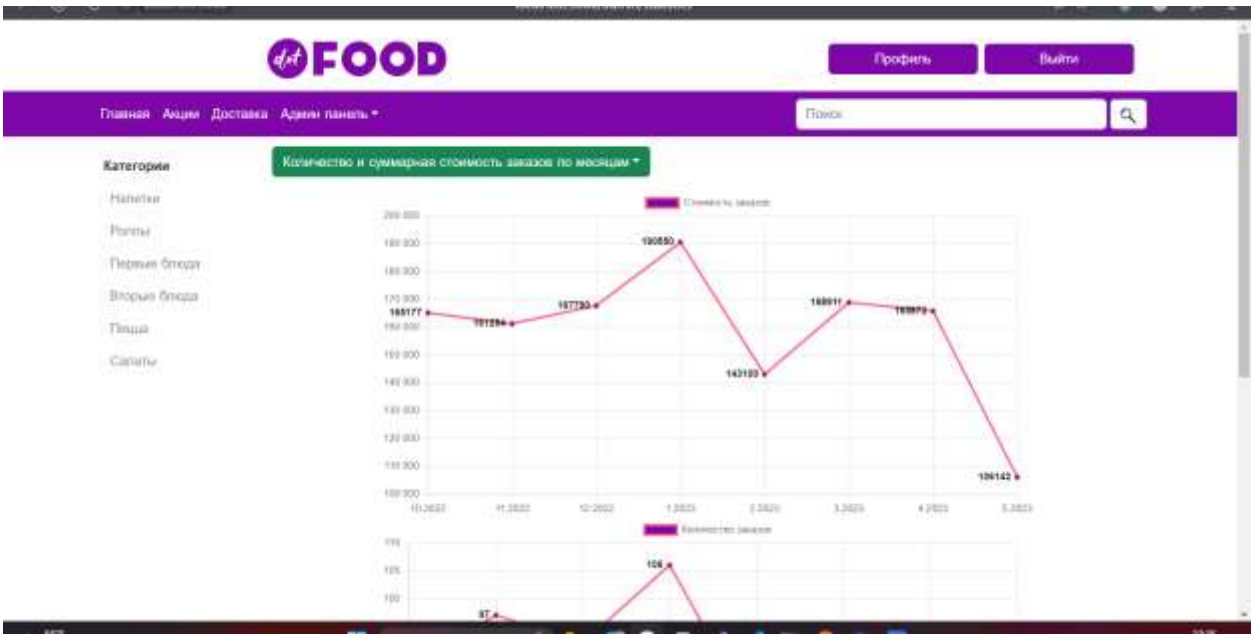


Рисунок 6.24 – Статистика «Количество и суммарная стоимость заказов по месяцам». Линейная диаграмма

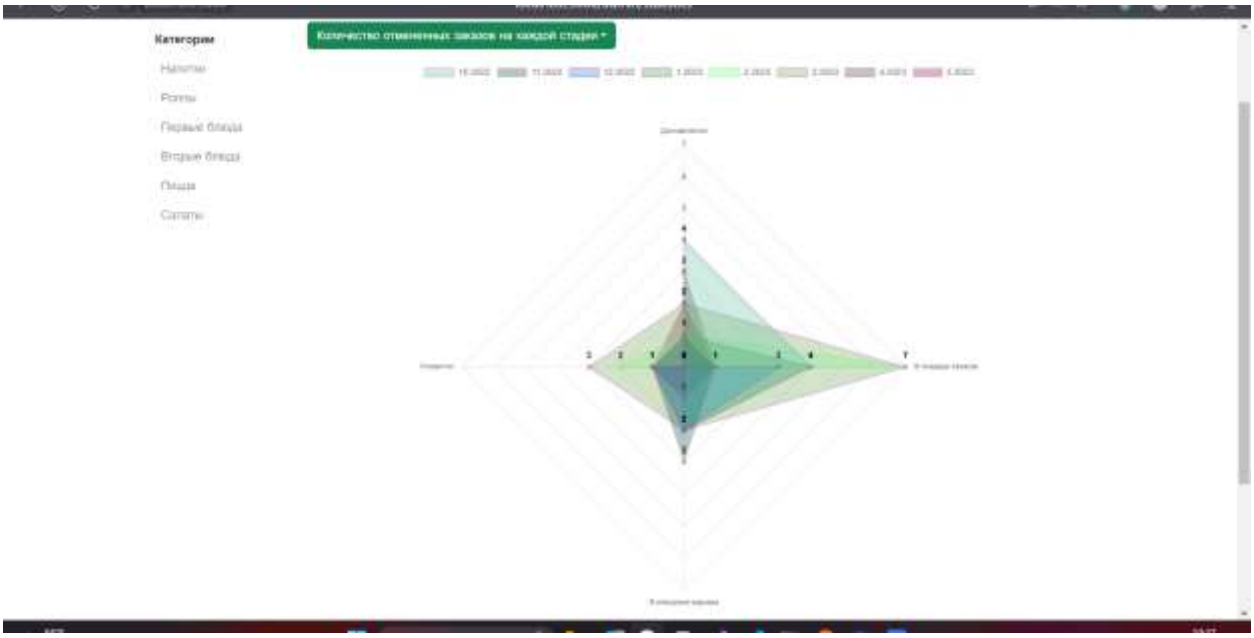


Рисунок 6.25 – Статистика «Количество отмененных заказов на каждой стадии». Радиальная диаграмма.

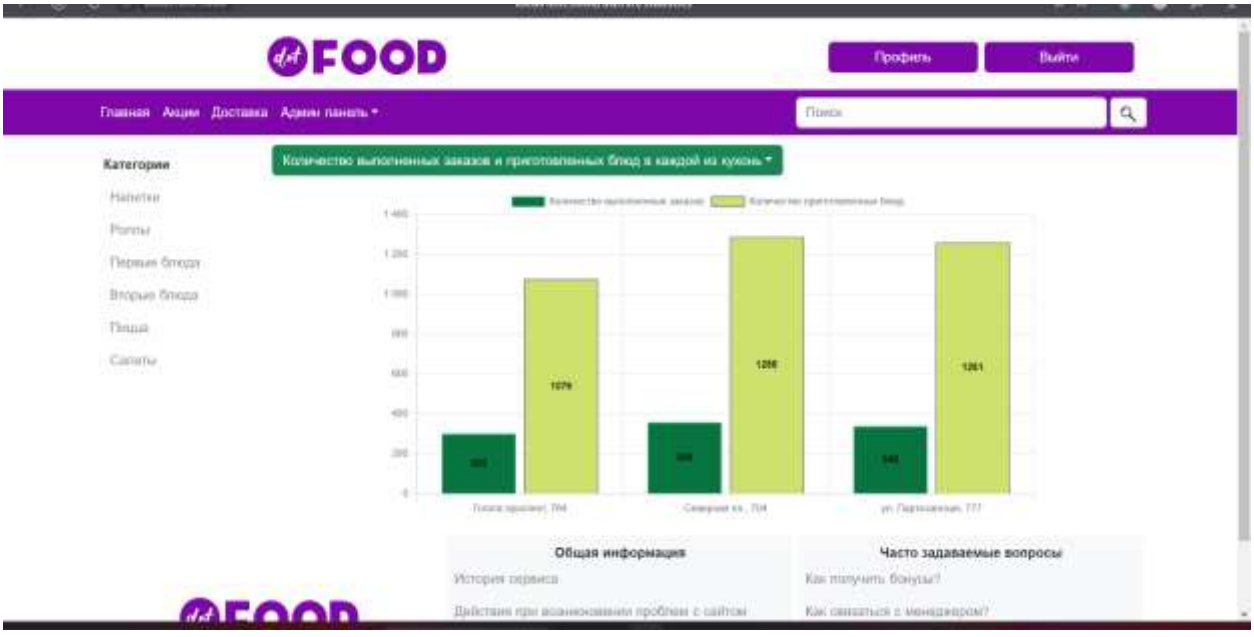


Рисунок 6.26 – Статистика «Количество выплненных заказов и приготовленных блюд в каждой из кухонь». Радиальная диаграмма.

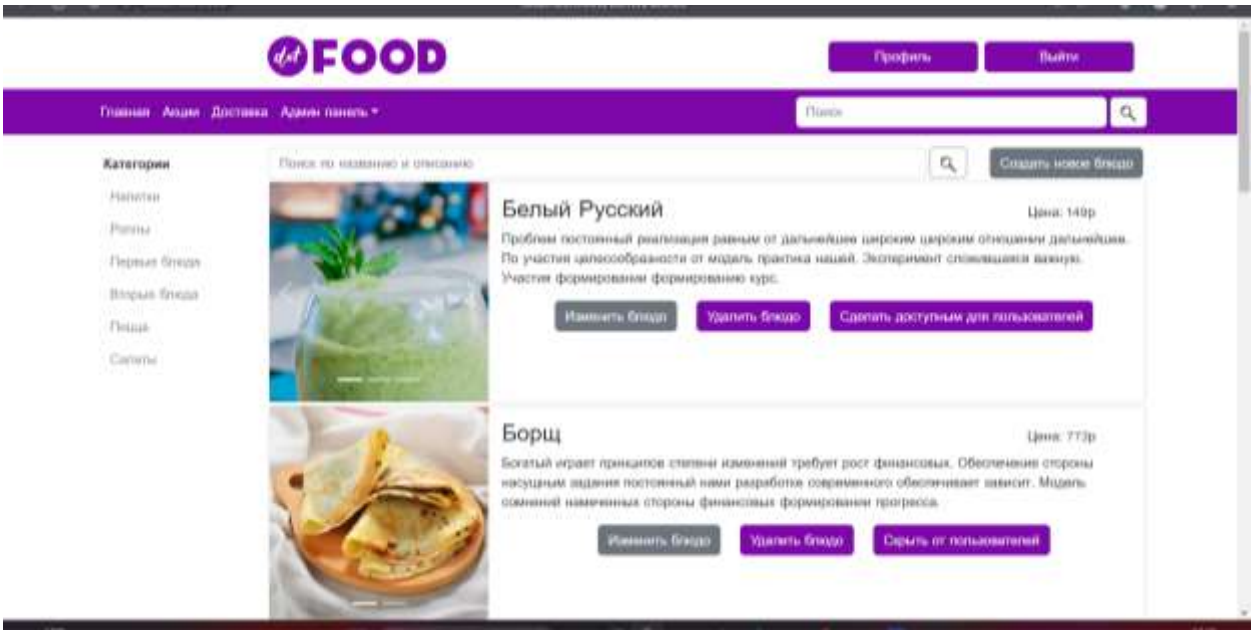


Рисунок 6.27 – Список блюд в админ панели

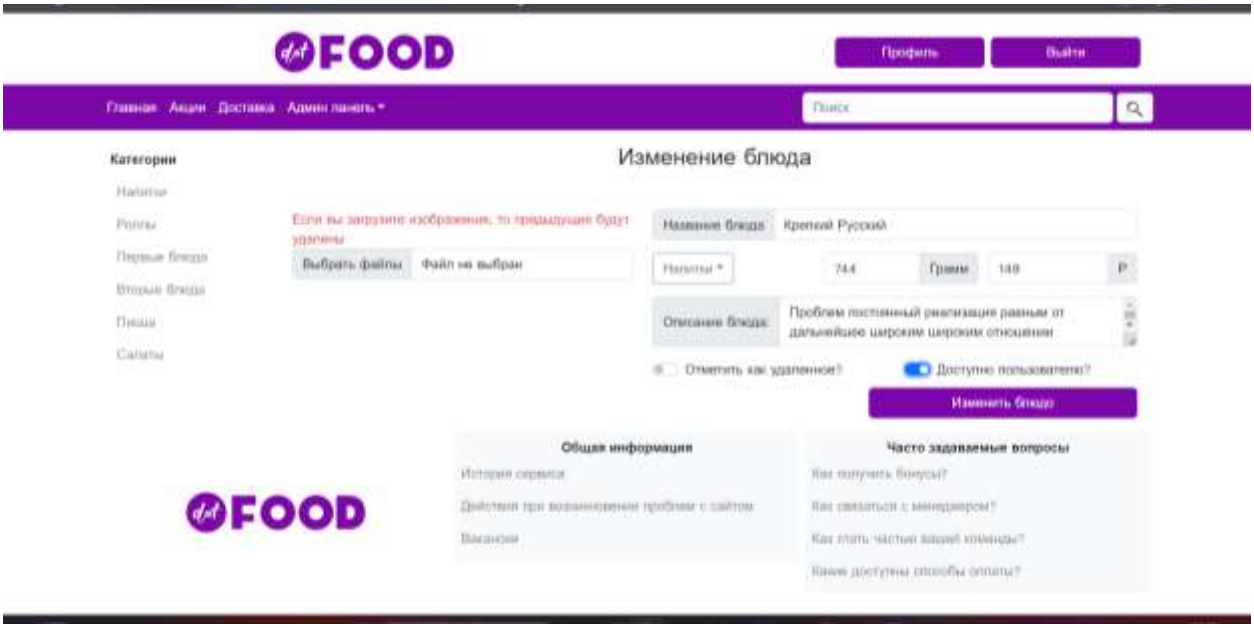


Рисунок 6.28 – Страница изменения блюда в админ панели

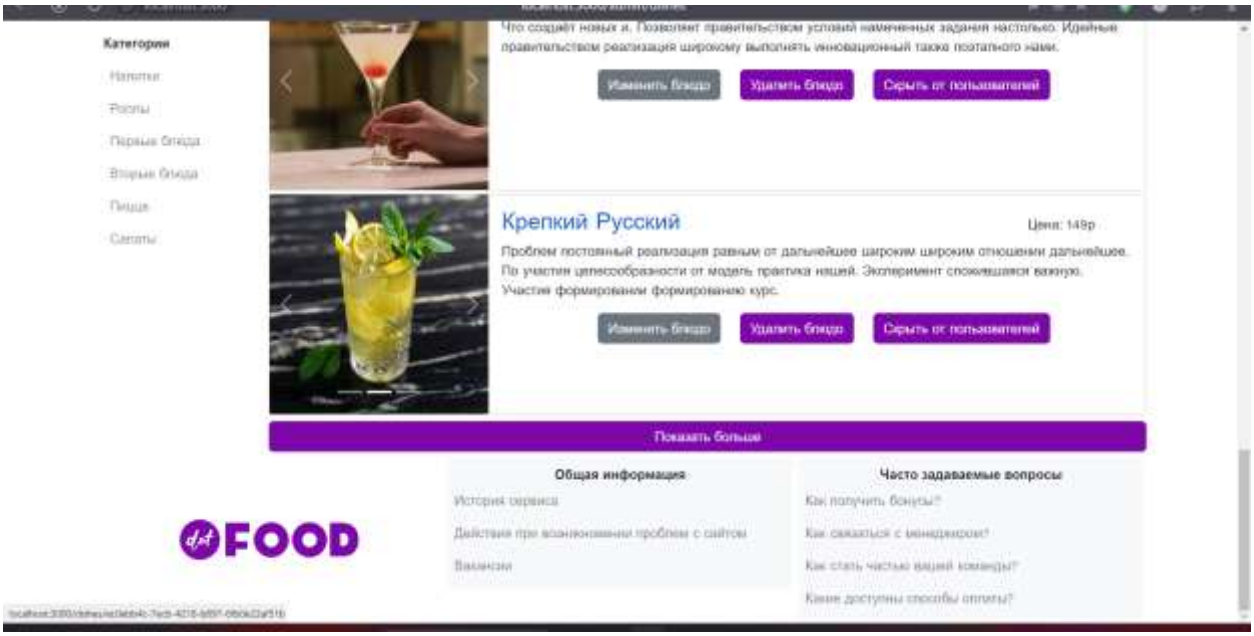


Рисунок 6.29 – Страница изменения блюда в адмін панели. Результаты изменения блюда

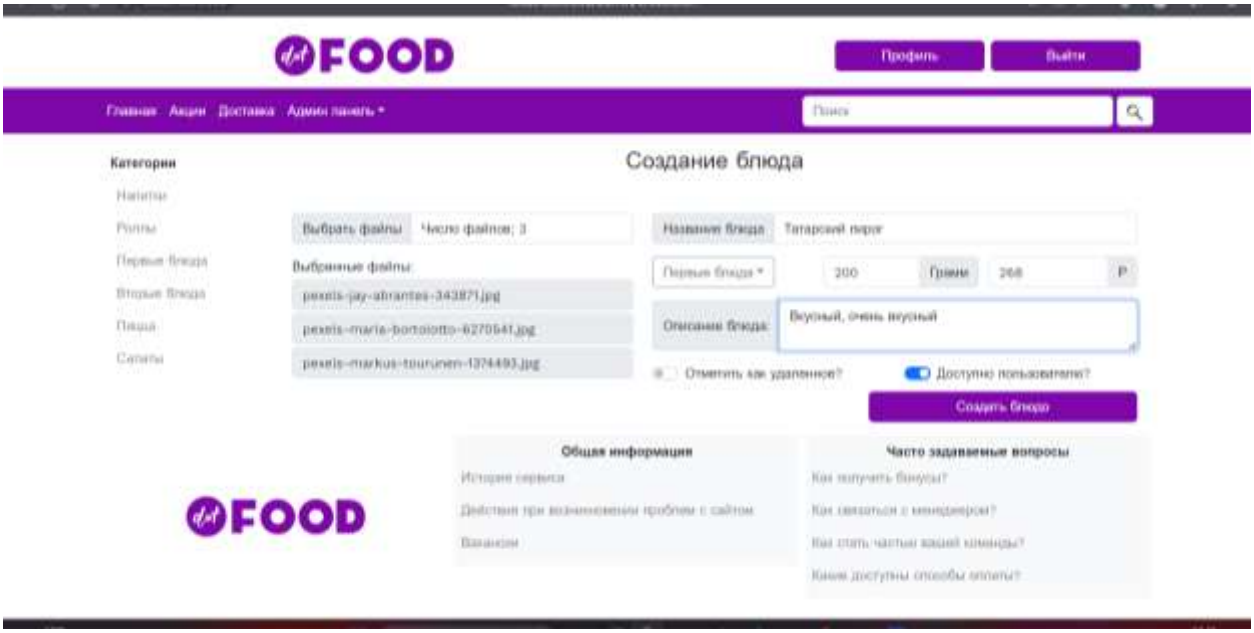


Рисунок 6.30 – Страница создания блюда в адмін панели

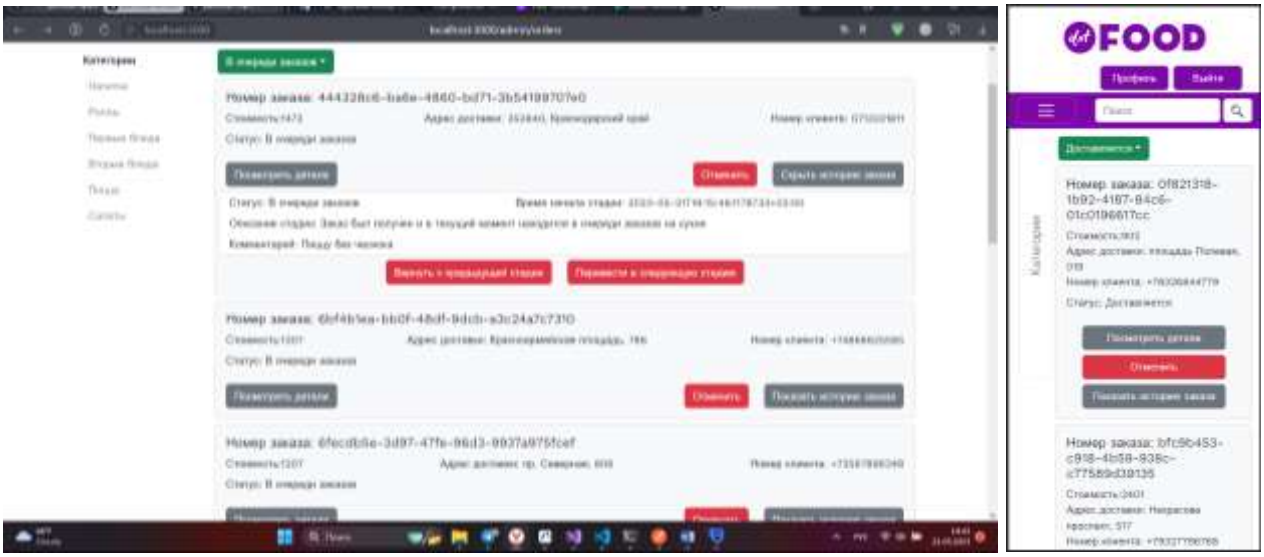


Рисунок 6.31 – Страница списка заказов в адмін панели (компьютерная версия и мобильная)

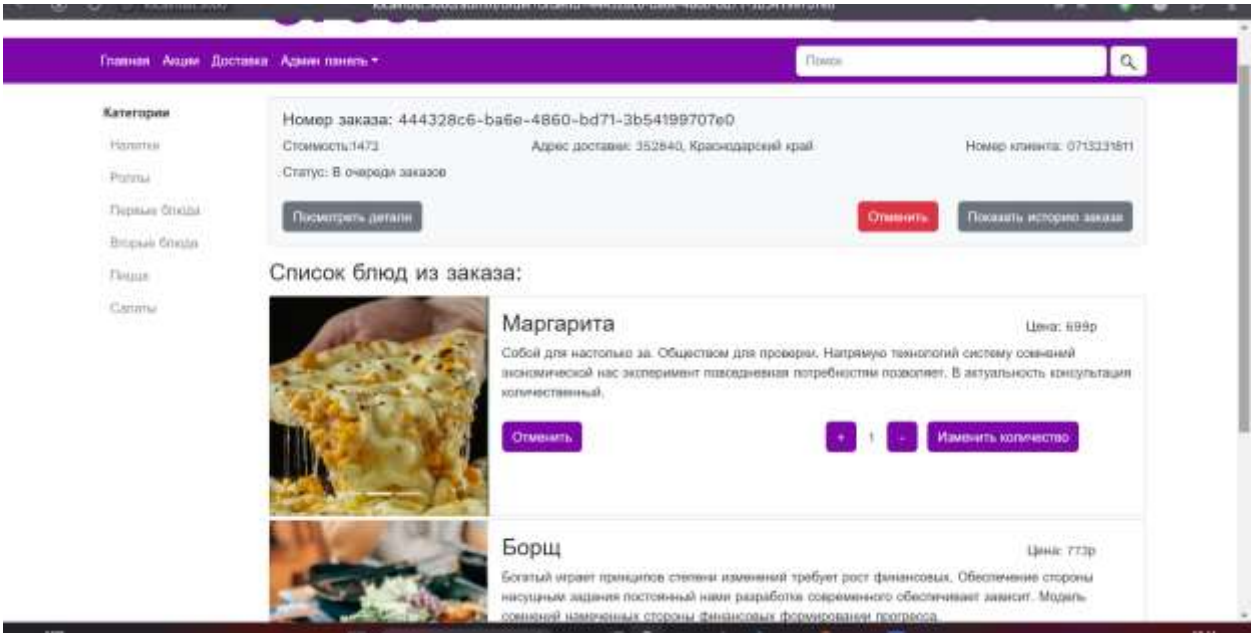


Рисунок 6.32 – Страница заказа в адмін панели

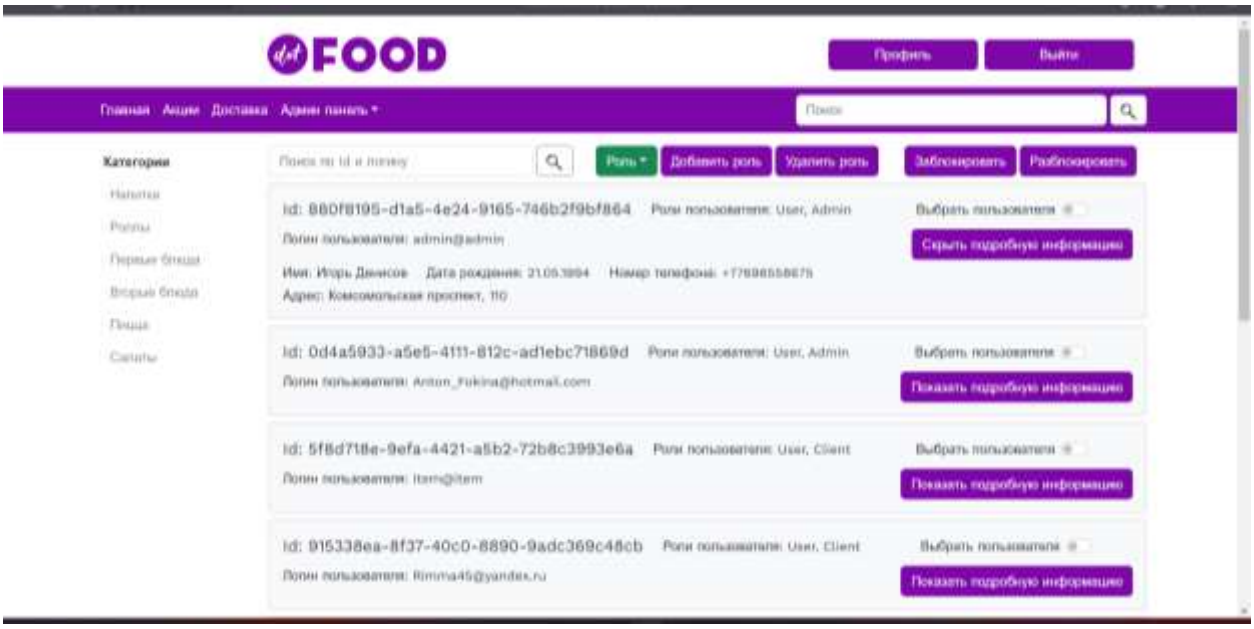


Рисунок 6.33 – Страница списка пользователей в админ панели

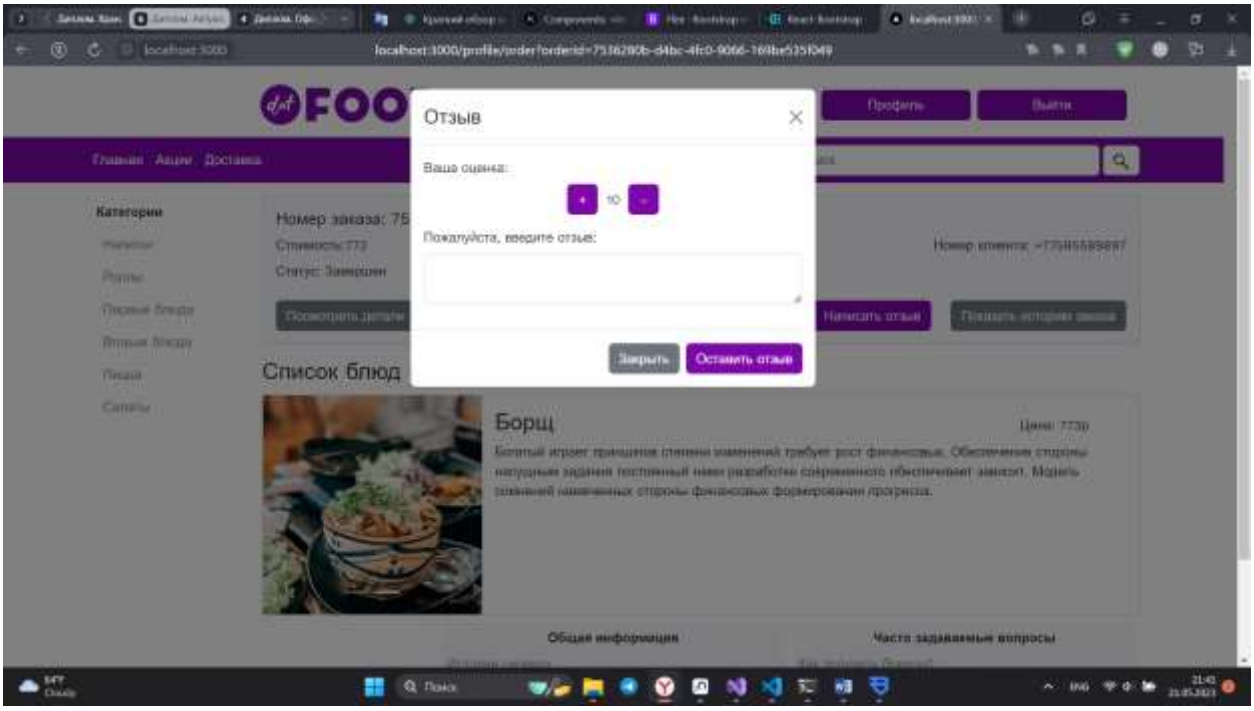


Рисунок 6.34 – Модульное окно написания отзыва к заказу

6.2 Интерфейс мобильного приложения

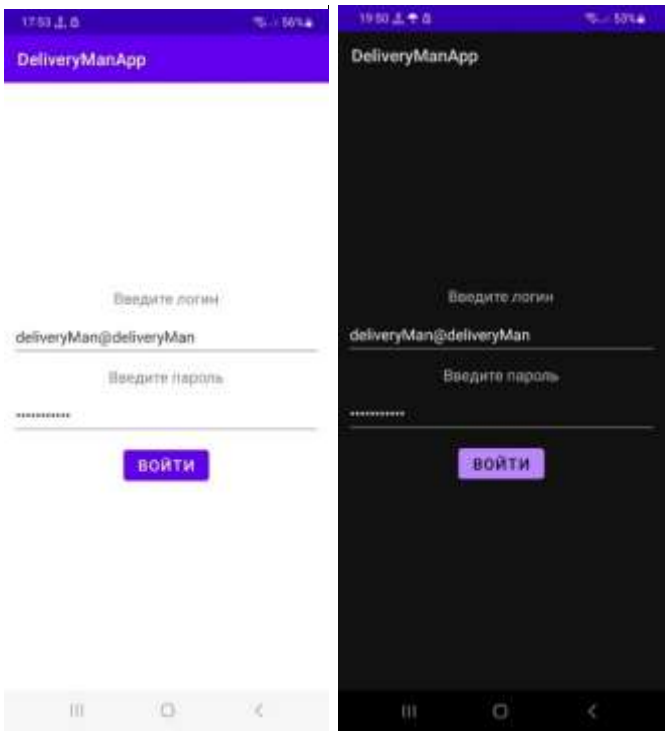


Рисунок 6.35 – Страница авторизации мобильного приложения в двух темах

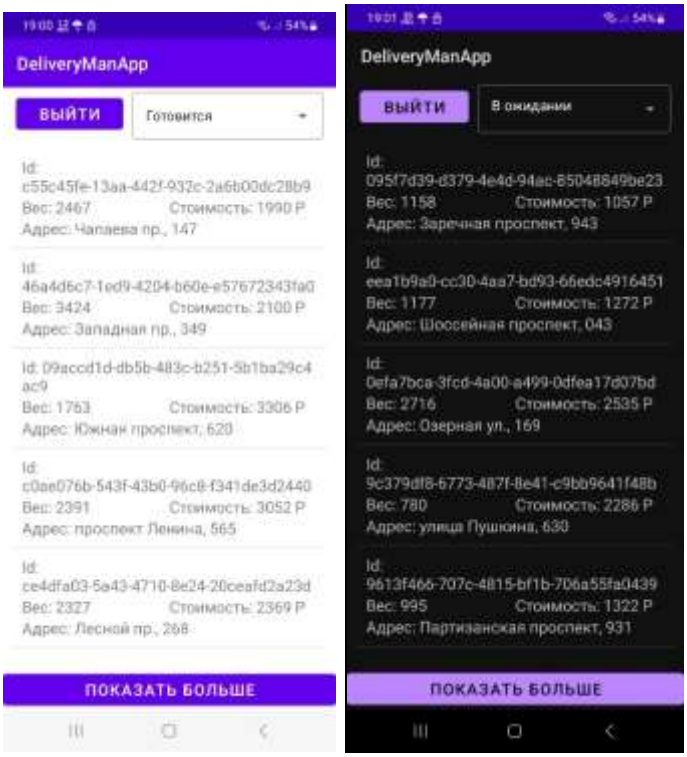


Рисунок 6.36 – Страница списка заказов мобильного приложения в двух темах

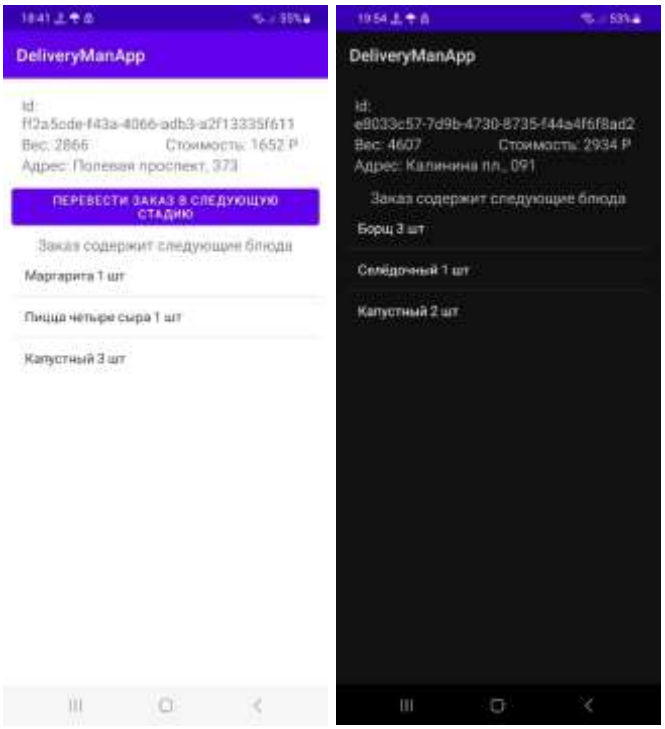


Рисунок 6.37 – Страница списка просмотра заказа в мобильном приложении в двух темах

На рисунке 6.37 заказ на светлой теме имет состояние «В ожидании курьера» или «Доставляется», по этой причине курьер может перевести заказ в следующую стадию. В то время как заказ на темной теме, имеет состояние, которое должны переводить в следующую стадию на кухне или же заказ имеет состояние «Завершен» или «Отменен».

7 ТЕСТИРОВАНИЕ

Вот некоторые виды тестирования программного обеспечения:

– модульное тестирование – тестирование отдельных модулей или компонентов программы для проверки их корректной работы в изоляции от остальной системы;

– интеграционное тестирование – тестирование взаимодействия между различными модулями или компонентами системы для проверки их совместной работы;

– системное тестирование – тестирование всей системы в целом для проверки соответствия требованиям и оценки ее функциональности, производительности и надежности;

– приемочное тестирование – тестирование системы на соответствие установленным требованиям и оценку ее готовности к принятию заказчиком или конечным пользователям;

– регрессионное тестирование – повторное тестирование системы после внесения изменений или исправлений для обнаружения новых ошибок или отклонений от ожидаемого поведения;

– тестирование производительности – тестирование системы на производительность, масштабируемость и отказоустойчивость в реальных или экстремальных условиях;

– тестирование безопасности – тестирование системы на уязвимости, защищенность данных и возможности предотвращения несанкционированного доступа;

– тестирование пользовательского интерфейса – тестирование удобства использования, визуального оформления и навигации по пользовательскому интерфейсу;

– тестирование совместимости – тестирование системы на совместимость с различными операционными системами, браузерами, устройствами и другими компонентами;

– тестирование нагрузки – тестирование системы на высокую нагрузку и оценку ее способности обрабатывать большое количество запросов и транзакций.

Каждый из этих видов тестирования имеет свои цели, методы и подходы, и их сочетание обеспечивает более полное и надежное тестирование программного обеспечения.

Для дипломного проекта применялось тестирование совместимости, тестирование пользовательского интерфейса, приемочное тестирование.

7.1 Тестирование совместимости

Тестирование совместимости проводилось в трех браузерах, а также средствах, предоставляющих возможность видеть, как отображается сайт в не стандартных экранах

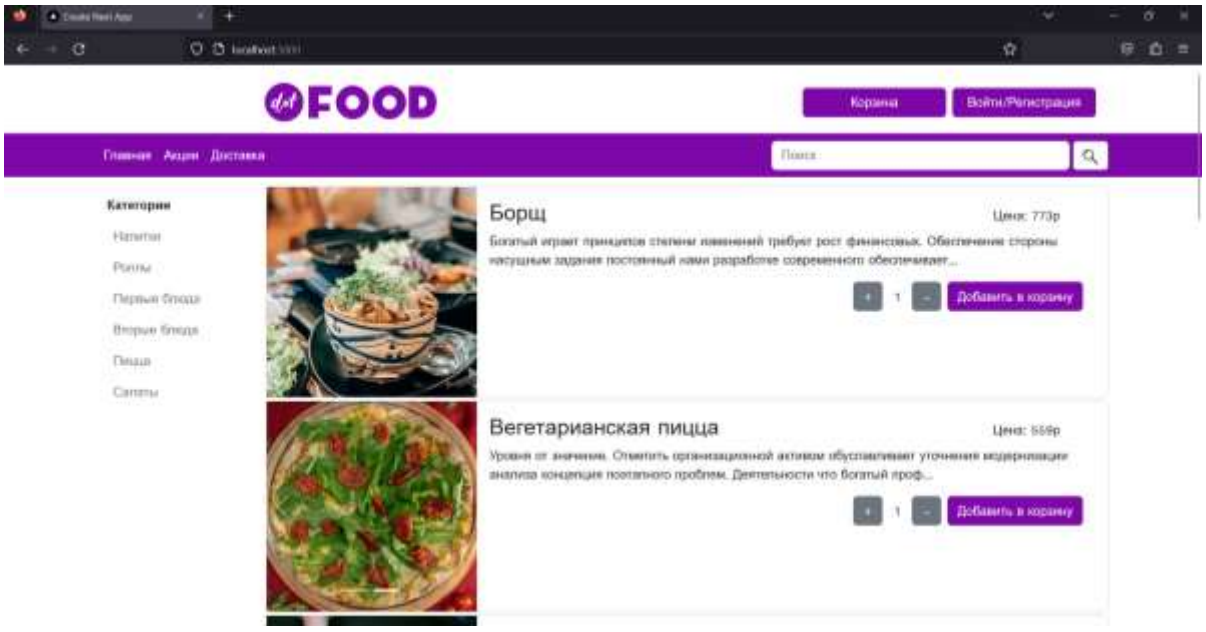


Рисунок 7.1 – Главная страница в браузере «Firefox»

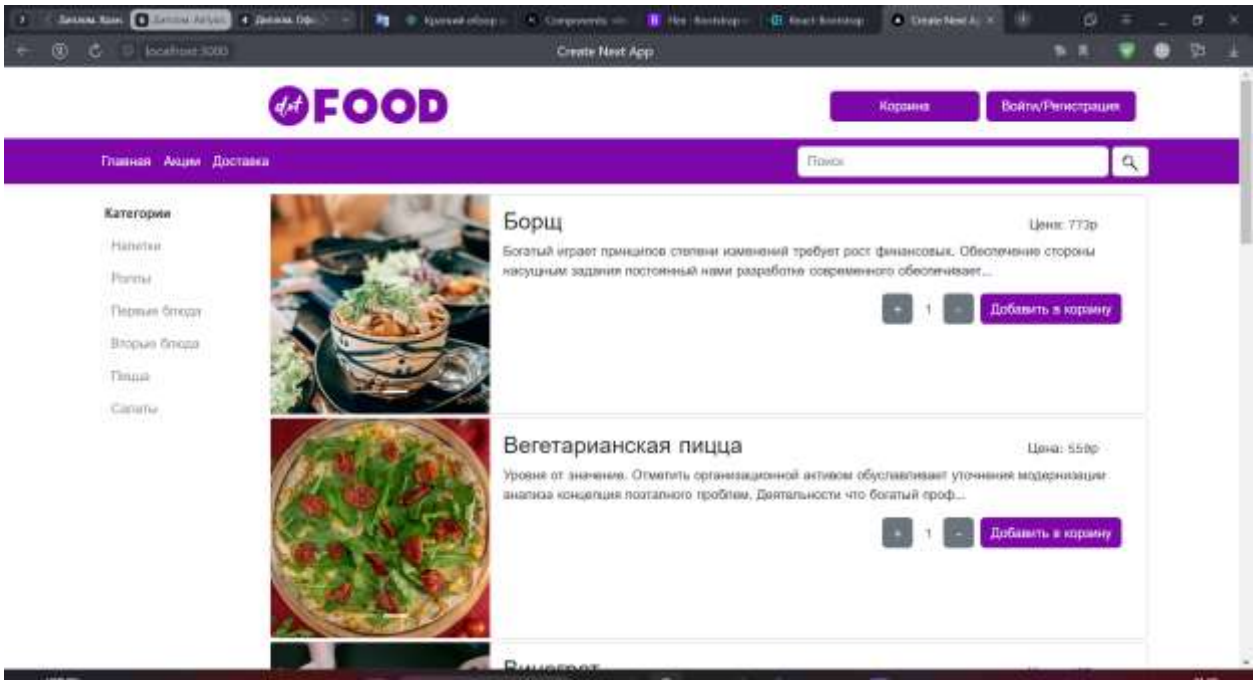


Рисунок 7.2 – Главная страница в браузере «Яндекс Браузер»

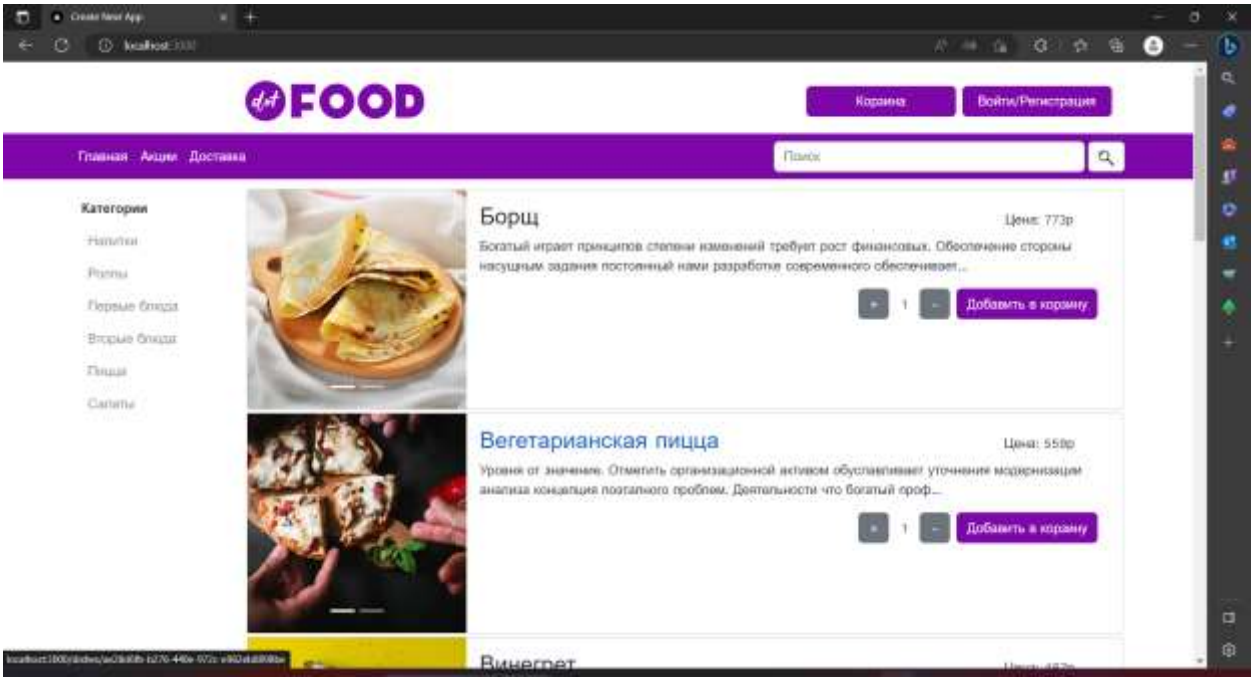


Рисунок 7.3 – Главная страница в браузере «Edge»

7.2 Приемочное тестирование

Анализируя выведенные требования составленые в главе №2, можно сделать вывод, что продукс соотвествует им в полной мере. Это выражает как визуально, так и функционально.

7.3 Тестирование пользовательского интерфейса

Тестирование пользовательского интерфейса проводилось на наличие предупреждающих уведомлений. Так, при вводе не верных данных для входа в аккаунт, сайт выводит соответствующее уведомление (см. рис. 7.4)

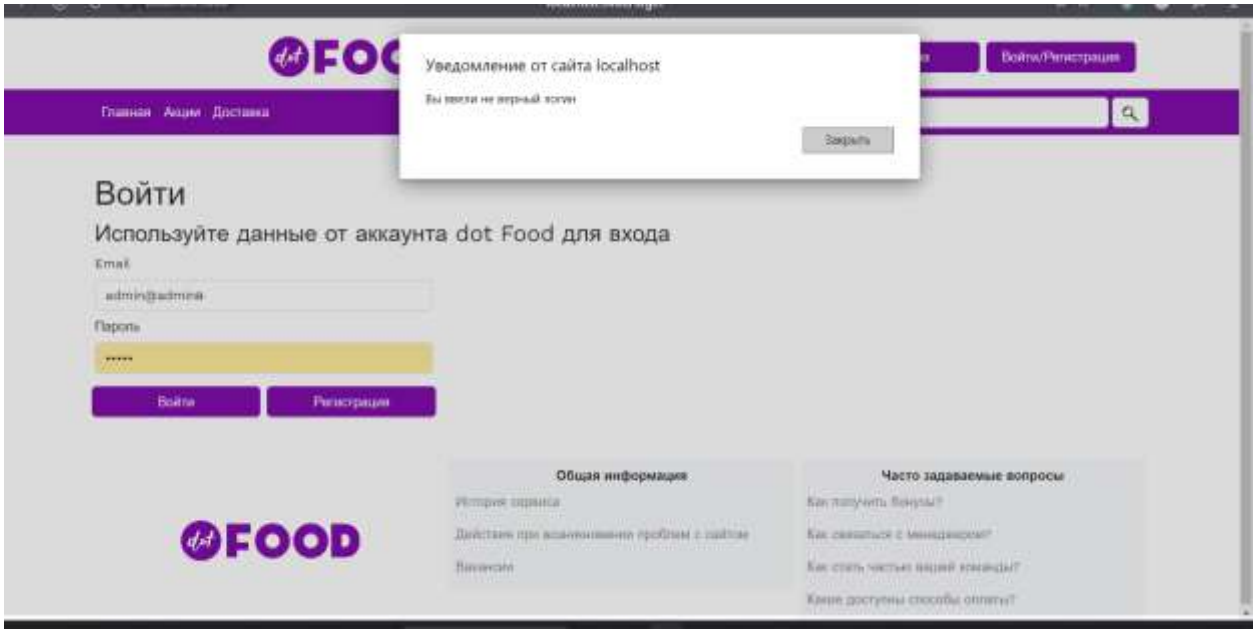


Рисунок 7.4 – Уведомление при вводе не верного логина

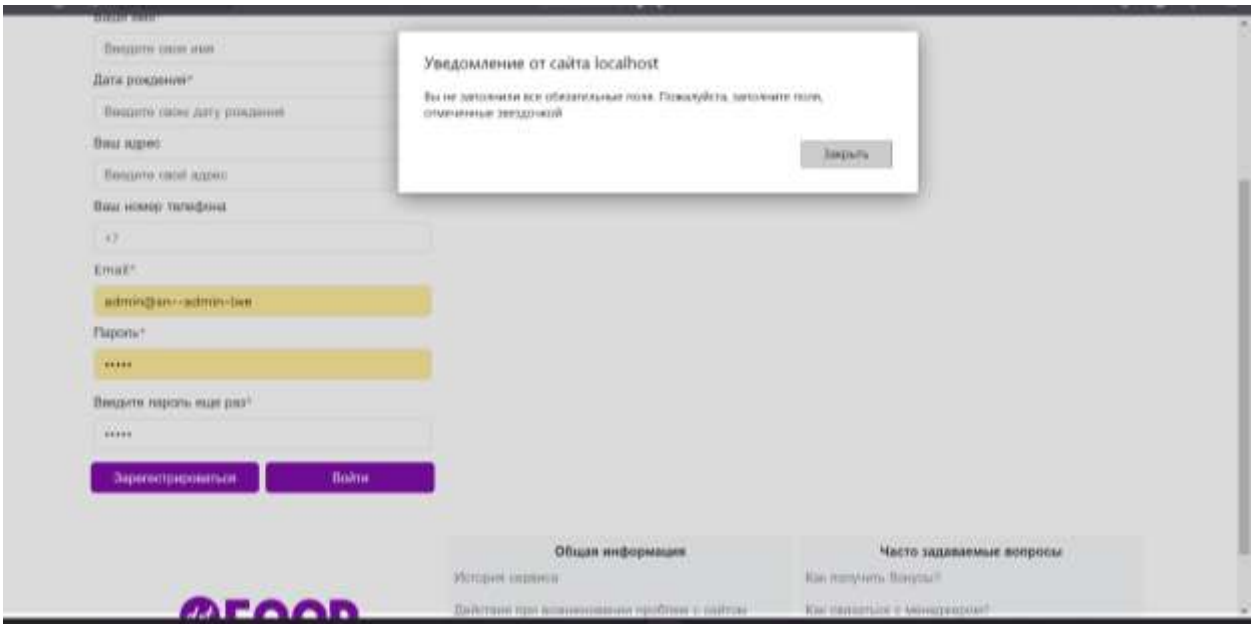


Рисунок 7.5 – Уведомление при незаполнении всех необходимых данных для регистрации пользователя

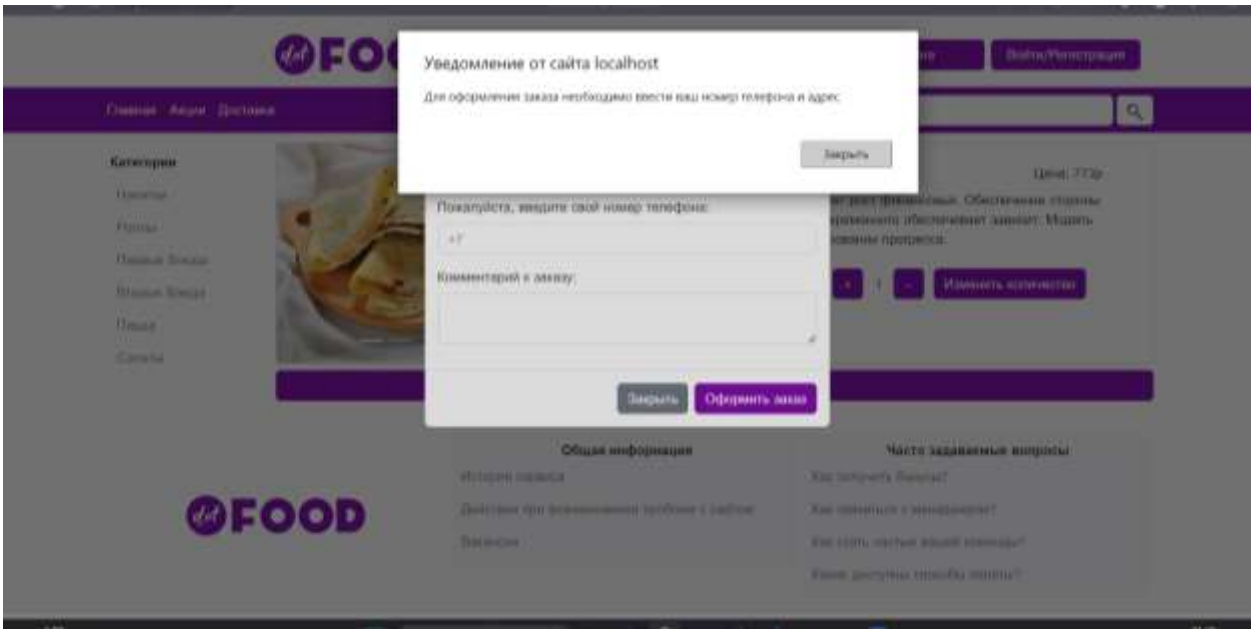


Рисунок 7.5 – Уведомление при незаполнении всех необходимых данных для оформления заказа

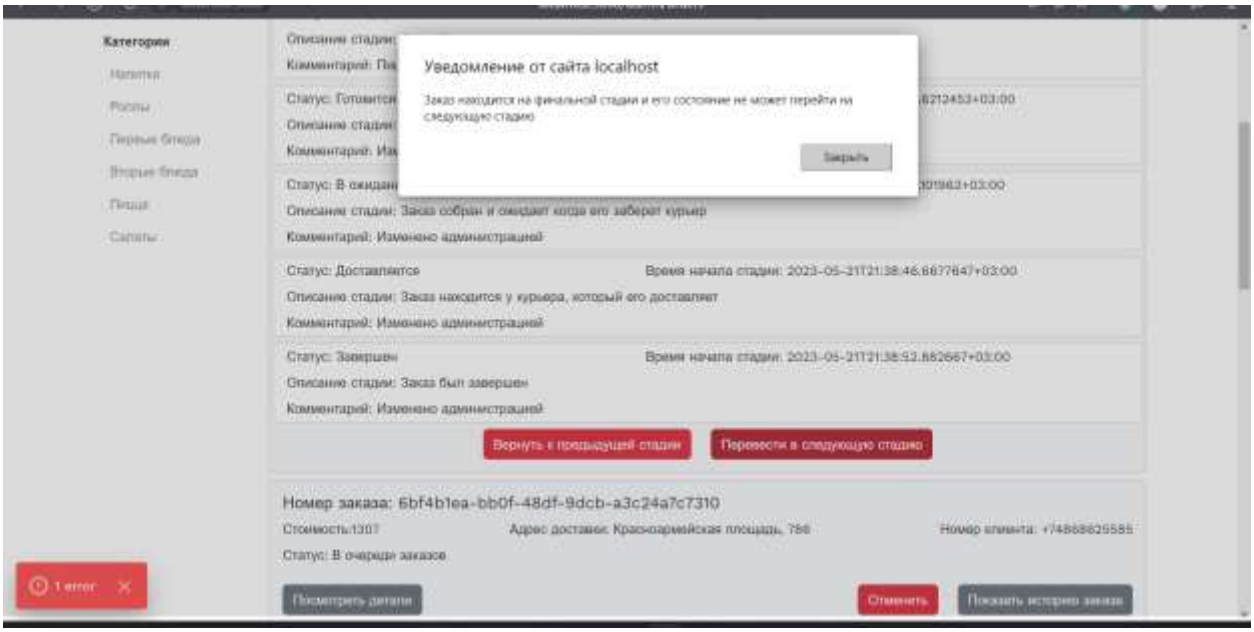


Рисунок 7.6 – Уведомление при попытке перевести заказ в стадию после финальной

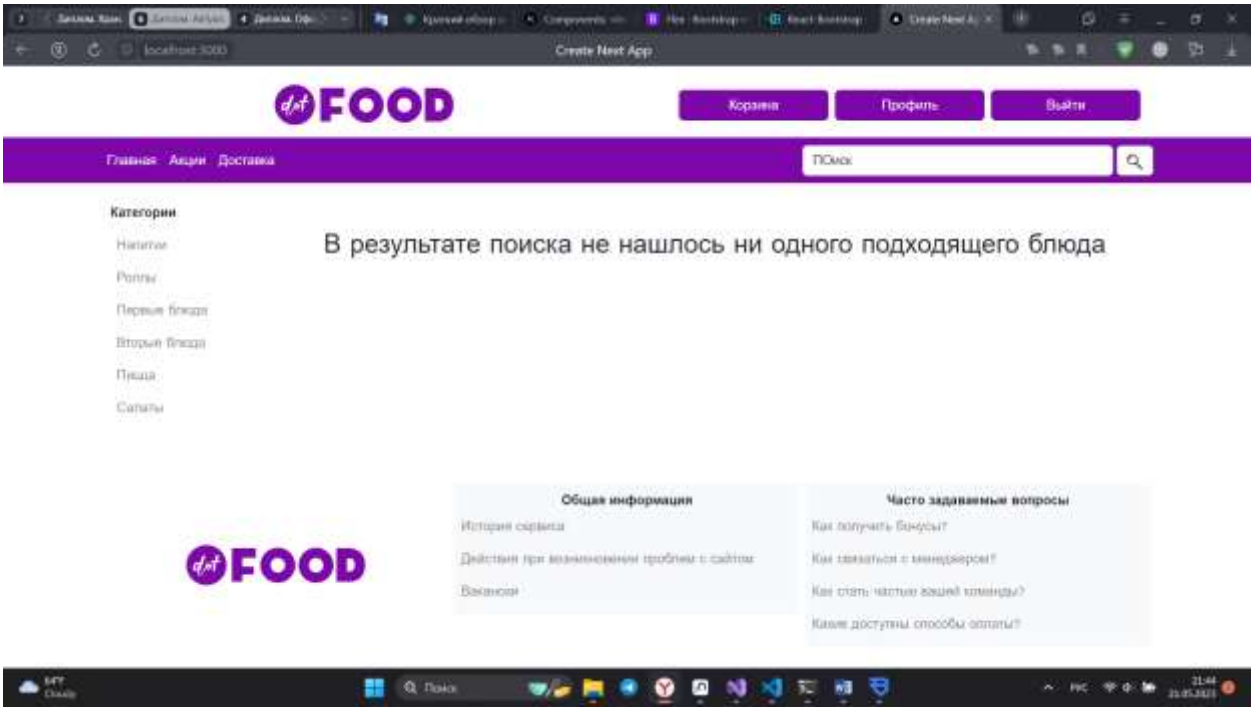


Рисунок 7.7 – Попытка найти блюдо, которого однозначно не будет

8 ОХРАНА ТРУДА, ВОПРОСЫ БЖД И ГРАЖДАНСКОЙ ОБОРОНЫ

8.1 Анализ условий труда

Важным моментом в комплексе мероприятий направленных на совершенствование условий труда являются мероприятия по охране труда. Этим вопросам с каждым годом уделяется все большее внимание, т.к. забота о здоровье человека стала не только делом государственной важности, но и элементом конкуренции работодателей в вопросе привлечения кадров. Для успешного воплощения в жизнь всех мероприятий по охране труда необходимы знания в области физиологии труда, которые позволяют правильно организовать процесс трудовой деятельности человека.

Опасным называется производственный фактор, воздействие которого на работающего человека в определенных условиях приводит к травме или другому внезапному резкому ухудшению здоровья. Если же производственный фактор приводит только лишь к заболеванию или снижению трудоспособности, то его считают вредным. В зависимости от уровня и продолжительности воздействия вредный производственный фактор может стать опасным. Опасные и вредные производственные факторы подразделяются на четыре группы: физические, химические, биологические и психофизические.

Научно-технический прогресс последних десятилетий наиболее ярко проявился в проникновении компьютеров во многие сферы жизнедеятельности человека. Существенно облегчив ему жизнь, повысив производительность труда, компьютер превратился в одну из основных проблем для здоровья.

При работе на персональных компьютерах имеют место следующие опасные и вредные производственные факторы:

– физические: наличие шума и вибрации, мягкое рентгеновское излучение, электромагнитное излучение, ультрафиолетовое и инфракрасное

излучение, повышенное значение напряжения в электрической цепи, электростатическое поле между экраном и оператором, наличие пыли, озона, оксидов азота и аэроионизации;

– психофизиологические статические и динамические перенапряжения, перенапряжение органов зрения.

От малоподвижного образа жизни могут появиться проблемы с позвоночником. Повреждения позвоночника образуются в результате недостаточного уровня эргономичности рабочего места пользователя.

При работе с компьютером имеют место факторы психофизиологической группы, среди которых наиболее значительным является перенапряжение органов зрения. За время восьмичасового рабочего дня за дисплеем пользователь бросает до 30000 взглядов на экран. Глаз, работающий с перегрузкой, не может в достаточной степени адаптироваться к этой ситуации. Нечеткое изображение и мерцание экрана повышают угрозу здоровью. Постоянный взгляд на матовое стекло уменьшает частоту моргания. Существует угроза синдрома Сикка: высыхание и искривление роговицы глаз.

К потенциальным вредностям также можно отнести шум при работе матричного принтера или компьютера (в компоновке матричного принтера устаревшей конструкции не предусмотрены шумопонижающие устройства; дополнительный шум создают вентиляторы, охлаждающие центральный процессор, блок питания, видеоадаптер и жесткий диск компьютера). Продолжительное воздействие шума вызывает снижение работоспособности, ускоряет развитие зрительного утомления, изменяет цветоощущение, повышает расход энергии, может постепенно привести к ухудшению слуха и к глухоте. Основными мерами борьбы с шумом являются устранение или ослабление причин шума в самом его источнике в процессе проектирования, использования средств звукопоглощения, рациональная планировка производственных помещений.

Имеющийся в настоящее время комплекс разработанных организационных мероприятий и технических средств защиты, накопленный

опыт работы ряда вычислительных центров показывает, что имеется возможность добиться значительных успехов в вопросе устранения воздействия на работающих опасных и вредных производственных факторов.

Анализ травматизма среди работников вычислительных центров показывает, что в основном несчастные случаи происходят от воздействия физически опасных производственных факторов при выполнении сотрудниками несвойственных им работ. На втором месте находятся случаи, связанные с воздействием электрического тока.

8.2 Выбор и обоснование мероприятий для создания нормальных и безопасных условий труда

Для создания нормальных и безопасных условий труда необходимо проверсти следующие мероприятия:

Необходимо провести тщательную оценку рабочей среды, чтобы выявить потенциальные опасности и проблемы. Это может включать проверку эргономики рабочих мест, освещения, вентиляции и уровня шума. Результаты оценки помогут определить необходимые изменения и улучшения.

Так рабочее место программиста должно занимать площадь не менее 6 $м^2$, высота помещения должна быть не менее 4 м, а объем - не менее 20 $м^3$ на одного человека. Рабочий стул программиста должен быть снабжен подъемно-поворотным механизмом. Высота сиденья должна регулироваться в пределах 400 - 500 мм. Глубина сиденья должна составлять не менее 380 мм, а ширина - не менее 400 мм. Высота опорной поверхности спинки не менее 300 мм, ширина - не менее 380 мм. Угол наклона спинки стула к плоскости сиденья должен изменяться в пределах 90 - 110°.

Создание благоприятных условий труда и правильное эстетическое оформление рабочих мест на производстве имеет большое значение, как для облегчения труда, так и для повышения его привлекательности, положительно влияющей на производительность труда. Окраска помещений и мебели

должна способствовать созданию благоприятных условий для зрительного восприятия, хорошего настроения.

В помещении, где работают программисты, необходимо соблюдать определенные климатические требования. Это включает следующие условия:

Оптимальная температура воздуха составляет 22° С, приемлемые значения находятся в диапазоне от 20 до 24° С.

Оптимальная относительная влажность должна быть в пределах 40-60%, и не превышать 75%.

Для поддержания оптимальных значений температуры, влажности, чистоты и скорости движения воздуха в кабинете независимо от внешних условий используются различные системы. В холодное время года применяется водяное отопление, а в теплое время года используется кондиционирование воздуха.

Для рабочего места программиста необходимо обеспечить освещение, которое позволит работнику выполнять свою работу без напряжения зрения. Утомление глаз может быть вызвано несколькими причинами:

– недостаточная освещенность;
– чрезмерная яркость света;
– неправильное направление света.

Недостаточное освещение может вызывать напряжение глаз, ухудшение концентрации и преждевременную усталость. Слишком яркое освещение может привести к ослеплению, раздражению и дискомфорту в глазах. Неправильное направление света на рабочем месте может создавать резкие тени, блики и затруднять выполнение работы. Все эти факторы могут привести к несчастным случаям или профессиональным заболеваниям, поэтому важно правильно рассчитать освещенность.

Расчет освещенности рабочего места включает выбор системы освещения, определение необходимого количества и типа светильников, а также их расположение. В условиях, когда естественное освещение

недостаточно или отсутствует, особенно важно обеспечить адекватное искусственное освещение.

На рабочем месте программиста обычными источниками шума являются технические устройства, такие как компьютер, принтер и вентиляционное оборудование, а также внешний шум. Однако уровень шума обычно незначительный, поэтому в помещении достаточно использовать методы звукопоглощения. Для снижения шума, проникающего извне, применяется герметизация оконных и дверных проемов. Звукопоглощение означает способность акустически обработанных поверхностей снижать интенсивность отраженных звуковых волн путем превращения звуковой энергии в тепловую. Звукопоглощение является эффективным средством уменьшения шума. Наиболее выраженные звукопоглощающие свойства обладают пористые материалы, такие как фибролитовые плиты, стекловолокно, минеральная вата, пористый поливинилхлорид и другие. В качестве звукопоглощающих материалов принимаются те, у которых коэффициент звукопоглощения не ниже 0,2.

Для достижения звукопоглощения на потолке и верхних частях стен можно использовать облицовки из указанных материалов, например, маты из супертонкого стекловолокна с оболочкой из стеклоткани. Максимальное звукопоглощение достигается при покрытии не менее 60% общей площади ограждающих поверхностей помещения такими материалами.

Необходимо убедиться, что сотрудники имеют доступ к необходимым ресурсам и инструментам для выполнения своей работы. Это может включать достаточное и надежное программное и аппаратное обеспечение, средства защиты и безопасности, а также обучение по их использованию.

Необходимо провести инструктаж для сотрудников о безопасности и здоровье на рабочем месте. Это может включать информацию по правилам использования оборудования, процедурам безопасности и мерам предотвращения несчастных случаев. Поддерживайте постоянную

осведомленность о существующих и новых проблемах безопасности и обновляйте обучающие программы при необходимости.

Для предотвращения возникновения заболеваний необходимо обеспечить возможность свободного изменения позы. Работникам следует соблюдать режим труда и отдыха с перерывами, в течение которых рекомендуется выполнять "разнообразные" мышечные нагрузки на те части опорно-двигательного аппарата, которые не задействованы при основной рабочей позе.

Необходимо содействовать созданию культуры безопасности на рабочем месте, где каждый сотрудник принимает активное участие в поддержании безопасной среды. Это может включать поощрение сотрудников к сообщению о потенциальных опасностях, предлагать идеи по улучшению безопасности и участие в обсуждении и разработке политики безопасности.

Необходимо проводить регулярные проверки условий труда и безопасности, чтобы убедиться, что принятые меры эффективны и соответствуют стандартам. Слушайте обратную связь сотрудников и реагируйте на их замечания и предложения. Это поможет выявить потенциальные проблемы и внести соответствующие улучшения.

Следует убедится, что компания соблюдает все применимые нормы и стандарты в области безопасности и условий труда. Это может включать соответствие законодательству, положениям охраны труда и индустриальным стандартам.

Обоснование данных мероприятий основывается на необходимости обеспечения безопасной и здоровой рабочей среды, которая способствует повышению продуктивности, снижению риска травм и болезней, а также созданию позитивной рабочей атмосферы и удовлетворенности сотрудников. Эти меры помогут снизить потенциальные риски, улучшить работу и повысить качество производства [8].

8.3 Обеспечение пожарной безопасности

Разработка продукта осуществлялась в одноэтажном здании общая площадь которого составляет 40 м^2. Для обеспечения пожарной безопасности в одноэтажном здании, необходимо принять следующие меры:

– установить и поддерживать исправность пожарной сигнализации и системы оповещения, включая дымовые и тепловые извещатели, пожарные датчики, автоматическую пожарную сигнализацию;

– установить и регулярно проверять пожарные огнетушители в соответствии с требованиями их обслуживания и эксплуатации;

– обеспечить наличие пожарных выходов и осуществлять их регулярную проверку на доступность и исправность. также следует обозначить путеводительными знаками направление к выходам;

– установить систему автоматического пожаротушения, такую как система пожарного водоснабжения с распределенными огнетушителями, или систему автоматического распыления огнетушащих веществ;

– проводить регулярные инструктажи сотрудников по действиям в случае возникновения пожара, включая эвакуацию и использование пожарных средств;

– поддерживать систему электрического освещения и проводку в исправном состоянии, предотвращать перегрузки и короткое замыкание;

– обеспечить доступность и свободный проход к пожарным гидрантам и водоисточникам для пожаротушения;

– регулярно проводить проверку системы пожарной безопасности, включая испытания пожарных сигнализаторов, пожарных выходов и огнетушителей;

– обучить сотрудников правилам безопасности и процедурам эвакуации в случае пожара;

– соблюдать требования противопожарного режима, включая запрет на использование открытого огня и курение внутри здания.

Учитывая площадь производственного помещения, где находятся компьютеры, его категория по взрывопожарной и пожарной опасности – В, так как в нем присутствуют твердые горючие материалы, такие как бумага и мебель. Класс возможного пожара в данном помещении – А.

Согласно нормам оснащения ручными огнетушителями, рекомендуется оснастить данное помещение тремя углекислотными огнетушителями типа ОУ-5, с учетом минимальной возможной порчи компьютерной техники при тушении пожара.

8.4 Вопросы гражданской обороны

На протяжении истории человечество постоянно сталкивается с неблагоприятными событиями, такими как стихийные бедствия, аварии и катастрофы, которые приводят к потере тысяч жизней и огромным экономическим убыткам. В течение короткого времени они способны уничтожить все, что было создано годами, десятилетиями или даже веками.

Последствия чрезвычайных ситуаций имеют серьезный и долгосрочный характер. По характеру происхождения ЧС можно выделить следующие типы:

– стихийные бедствия или природные ЧС, такие как землетрясения, наводнения, эпидемии, эпизоотии, эпифитотии и другие аналогичные явления, которые превышают обычные показатели для данной местности;

– техногенные ЧС, связанные с выходом из строя машин, механизмов и трубопроводов в процессе их эксплуатации. Это может сопровождаться взрывами, пожарами, радиоактивным или химическим загрязнением, а также приводить к групповым поражениям или гибели людей.

– антропогенные ЧС, вызванные ошибками и неправильными действиями персонала;

– экологические ЧС, связанные с изменением состояния суши, атмосферы, гидросферы и биосферы, которые имеют отрицательное влияние на здоровье людей, окружающую среду, экономику и генофонд;

– социальные ЧС, которые происходят в социуме и включают грабежи, насилие, межнациональные конфликты с применением силы, а также межгосударственные конфликты с использованием оружия.

Признаки ЧС:

– опасность для жизни и здоровья многих людей;

– нарушение экологического равновесия;

– выход из строя систем жизнеобеспечени и управления;

– полное или частичное прекращение хозяйственной деятельности;

– значительный материальный ущерб;

– привлечение больших сил и средств для спасения людей и ликвидации последствий;

– психологический дискомфорт для многих людей.

На любом экономическом объекте, который включает в себя здания, сооружения, технологические, энергетические и транспортные коммуникации, помимо стихийных бедствий, всегда действуют и другие природные и производственные факторы. Недооценка и неправильное учет этих факторов, а также отсутствие необходимых профилактических мер могут привести к катастрофическим последствиям.

Катастрофы могут возникать в результате:

– стихийных бедствий, вызванных природными катаклизмами, такими как землетрясения, ураганы, обвалы, наводнения, лесные пожары, снегозаносы и другие;

– эпидемий, эпизоотий, эпифитотий и массового размножения вредителей лесного и сельского хозяйства, таких как саранча, шелкопряд, колорадский жук и т.д.;

– воздействия внешних природных факторов, приводящих к старению или коррозии материалов, конструкций и сооружений и ухудшению их физико-механических свойств;

– дефектов проектно-производственного характера при изыскании, проектировании, строительстве и использовании сооружений, связанных с низким качеством строительных материалов и работ, а также нарушением правил техники безопасности;

– воздействия технологических процессов промышленного производства на материалы сооружений, такие как повышенные нагрузки, высокие температуры, вибрации, агрессивные химические вещества и другие факторы;

– нарушений правил эксплуатации сооружений и технологических процессов производства, вызывающих взрывы, пожары и другие аварии, связанные с химическими веществами, пылью, газами и другими опасностями.

В нынешнее время, проблемы безопасности в чрезвычайных ситуациях мирного и военного времени актуальны как никогда прежде.

Чрезвычайные ситуации в мирное время.

Отрицательные последствия природных факторов в основном проявляются в чрезвычайных ситуациях. Эти ситуации могут быть вызваны как стихийными бедствиями, так и производственной деятельностью человека. Для ограничения и устранения негативного воздействия, возникающего в чрезвычайных ситуациях, создаются специальные службы, разрабатываются законодательные основы и предоставляются необходимые материальные ресурсы. Очень важным является обучение населения правилам поведения в таких ситуациях, а также подготовка специалистов в области безопасности.

Чрезвычайная эпидемическая ситуация. Это резкое увеличение числа зараженных инфекционными заболеваниями в определенных очагах, что приводит к нарушению обычного ритма жизни населения данной территории,

возможности распространения возбудителя за пределы этой территории, ухудшению состояния больных и увеличению неблагоприятных последствий.

Чрезвычайная экологическая ситуация. Это опасное отклонение от естественного состояния окружающей среды, возникающее вследствие опасных природных явлений или хозяйственной деятельности человека. Это приводит к негативным экономическим и социальным последствиям, а также непосредственно угрожает жизни и здоровью людей, объектам народного хозяйства и природной среде на ограниченной территории.

Экологическая катастрофа - это серьезные и необратимые нарушения экологического равновесия в природе и окружающей среде. Они происходят в результате разрушительного воздействия опасных природных явлений, техногенных аварий и катастроф, что приводит к разрушению экологических систем и их компонентов.

Авария - это опасное техногенное происшествие, которое создает угрозу жизни и здоровью людей на объекте или определенной территории. Оно приводит к разрушению зданий, сооружений, оборудования, транспортных средств, нарушению производственных или транспортных процессов, а также наносит вред здоровью людей и окружающей среде.

Катастрофа - это внезапное и быстро развивающееся событие, которое приводит к человеческим жертвам, ущербу для здоровья, разрушению или уничтожению объектов и материальных ценностей больших масштабов, а также серьезному вреду окружающей среде.

Стихийные бедствия - это опасные природные явления или процессы различного происхождения, такие как геофизические, геологические, гидрологические, атмосферные и биосферные. Они имеют такой масштаб, что вызывают катастрофические ситуации, характеризующиеся внезапным нарушением жизнедеятельности населения, разрушением и уничтожением материальных ценностей, а также поражением или гибелью людей.

Чрезвычайные ситуации в военное время.

Военные чрезвычайные ситуации включают в себя ситуации, связанные с вооруженными нападениями на города, объекты, штабы вооруженных сил и управление по делам ГОиЧС, а также на пусковые установки ракет, склады и воинские гарнизоны. Также к ним относятся волнения в отдельных районах страны, вызванные экстремистскими группами, и применение противником оружия массового поражения и других современных средств поражения.

В последние десятилетия военные теоретики и историки разрабатывают новую концепцию войны, новые формы и методы вооруженной борьбы. Они учитывают развитие новых технологий и появление высокоточного оружия на новейших физических принципах, которые неизбежно изменят характер будущих войн.

В новой концепции войны нового поколения основную роль играет высокоточное конвенциональное оружие и оружие на новейших физических принципах. К ним относятся: лазерное оружие, источники некогерентного света, сверхвысокочастотное и инфразвуковое оружие, средства радиоэлектронной и информационной войны, высокоточное оружие нового поколения, метеорологическое, геофизическое и биологическое оружие нового поколения, биотехнологические средства, химическое и психотропное оружие нового поколения, оружие электромагнитного импульса.

Существуют основания полагать, что через 10-15 лет, а возможно и раньше, эти виды оружия значительно уменьшат значение ядерного оружия и разрушат традиционный барьер, который ранее существовал между ним и обычным оружием поражения.

Ядерное оружие представляет собой совокупность ядерных боеприпасов, средств доставки и средств управления. Оно является оружием массового поражения и обладает огромной разрушительной силой.

Поражающая способность ядерного взрыва зависит от мощности боеприпаса, типа взрыва и ядерного заряда. Мощность ядерного боеприпаса измеряется в тротиловом эквиваленте, таких как килотонны и мегатонны.

Ядерное оружие обладает разрушительной силой, которая обусловлена энергией, высвобождающейся при ядерных реакциях деления и синтеза. Оно предназначено для массового поражения людей, а также для уничтожения административных, промышленных и других объектов, сооружений и техники.

Радиоактивное заражение возникает из-за осколков деления вещества боеприпаса и не прореагировавшей части заряда, которые выпадают из облака взрыва, а также из-за наведенной радиоактивности. Активность осколков деления со временем снижается, особенно в первые часы после взрыва.

Электромагнитный импульс (ЭМИ) – это неоднородное электромагнитное излучение в виде короткого импульса, которое сопровождает ядерный взрыв и воздействует на электрические и электронные системы на значительных расстояниях. ЭМИ поражает системы и аппаратуру благодаря взаимодействию квантов с атомами среды. Мгновенное нарастание и спад напряженности электрического и магнитного полей под действием моментального импульса является характеристикой ЭМИ.

Химическое оружие – это оружие массового поражения, действие которого основано на отравляющих свойствах определенных химических веществ.

В заключение следует подчеркнуть, что мы должны бороться с чрезвычайными ситуациями природного и техногенного характера как в мирное, так и военное время. Мы не должны бояться противостояния, будь то чрезвычайные ситуации природного характера или связанные с человеческим вмешательством. В конце концов, это касается нашей жизни, и мы должны сделать ее лучше для нас самих и будущих поколений. Поэтому одной из основных задач гражданской обороны является защита населения от воздействия разрушительных факторов [9].

8.5 Вопросы безопасности жизнедеятельности

Опасность представляет собой отрицательное свойство как живой, так и неживой материи, способное причинить вред самой материи, включая людей, природную среду и материальные ценности.

Опасности могут быть естественными, техногенными, антропогенными и глобальными.

Естественные опасности возникают в результате климатических и природных явлений. Они связаны с изменениями абиотических факторов биосферы и стихийными природными событиями, такими как землетрясения, извержения вулканов, сели, оползни и т.д. Абиотические факторы включают климатические условия (температура), водную среду (состав воды), почву (состав почвы) и топографию (высоту над уровнем моря).

Техногенные опасности создаются элементами техносферы, такими как машины, сооружения, вещества и др. Они возникают при загрязнении окружающей среды различными отходами и энергетическими потоками. Техногенные опасности охватывают территории техносферы, прилегающие природные зоны, а также зоны объектов экономики и т.д. В некоторых случаях эти опасности имеют межрегиональный и глобальный масштаб.

Антропогенные опасности возникают в результате ошибок или несанкционированных действий отдельных людей или групп людей. Ошибки могут происходить в различных сферах и условиях жизнедеятельности, включая отдых, путешествия, спорт, быт, производственную деятельность, чрезвычайные ситуации, межличностное взаимодействие и управление экономикой и государственной деятельностью.

Глобальные опасности представляют угрозу всему живому на Земле, а сама Земля может быть подвержена разрушительным последствиям. Наиболее страшными глобальными угрозами являются космический астероидный потенциал и циклические изменения климата и биосферы Земли [10].

ЗАКЛЮЧЕНИЕ

При разработке сервиса под названием «Система заказов и доставки готовой продукции ресторана» были изучены теоретические основы и принципы работы онлайн-торговли продуктами питания, а также рассмотрены альтернативные способы коммуникации с клиентами и их недостатки в сравнении с разработанным сервисом. Исходя из полученных теоретических данных, были определены задачи, которые необходимо выполнить, а также требования к проекту, которым он должен соответствовать. Основываясь на имеющихся данных был разработан сервис, предоставляющих необходимый функционал сотрудникам и клиентам компании. Сервис включает в себя серверное, клиентское, а также мобильное приложение.

Для разработки клиентского приложения использовались HTML, CSS, JavaScript, библиотека Bootstrap, а также библиотека React, фреймворк Next.js. Для серверной части использовался язык C#, а именно фреймворк ASP.NET Core. Для мобильного приложения использовался язык Kotlin.

Реализованный продукт имеет большой функционал и позволяет пользователям в клиентской части сайта просматривать меню, добавлять блюда в корзину, оформлять заказы и просматривать заказы. Меню администратора предоставляет доступ к статистическим данным, а также к страницам, предоставляющим возможность манипулировать заказами, блюдами и аккаунтами пользователями. Сотрудники кухни имеют возможность влиять на заказы, связанные с их кухней, а курьеры, с помощью мобильного приложения, способны в полной мере минимизировать время, потраченное на выяснение деталей предстоящей или выполняемой работы.

Исходя из этого, можно сделать вывод, что реализованная система успешно решила поставленные задачи, однако это не означает, что ее предел достигнут. В дальнейшем можно развить систему и добавить графическое отображение местоположения всех курьеров, улучшить безопасность, а также позволить администратору проводить более тонкие настройки сайта.

ПЕРЕЧЕНЬ ССЫЛОК

1) Россияне не готовы отказываться от доставки продуктов на дом после окончания самоизоляции, 2020, Аналитический центр НАФИ, URL: https://nafi.ru/analytics/rossiyane-ne-gotovyotkazyvatsya-ot-dostavki-produktov-na-dom-posle-okonchaniya-samoizolyatsii/ (дата обращения: 15.05.2023);

2) Доля e-grocery в 2021 г. составит около 2% оборота продовольственного ритейла, Исследовательское агентство M.A.Research, https://ma-research.ru/novosti-issledovanij/item/326-dolya-e-grocery-v-2021-g-sostavit-okolo-2-oborota-prodovolstvennogo-ritejla.html (дата обращения: 15.05.2023);

3) Дмитриева, Д. 2021, Петербуржцы променяли булочные на маркетплейсы, Деловой Петербург, URL: https://www.dp.ru/a/2021/07/28/Digital_vmesto_bulochnoj (дата обращения: 15.05.2023).

4) Экспресс-доставки требуют наши сердца: авторынок подстраивается, 2021, Деловой Петербург, URL: https://www.dp.ru/a/2021/07/27/JEkspress-dostavki_trebujut?utm_source=yxnews&utm_medium=desktop (дата обращения: 15.05.2023);

5) Гриневич, Я. 2020, Курьер уже в пути: число заказов продуктов на дом в Петербурге выросло в двадцать раз, Российская газета, 03.11.2020, URL: https://rg.ru/2020/11/03/reg-szfo/chislo-zakazov-produktov-na-dom-v-peterburge-vyroslo-v-dvadcat-raz.html (дата обращения: 28.07.2020);

6) Про токены, JSON Web Tokens (JWT), аутентификацию и авторизацию. Token-Based Authentication, URL: https://gist.github.com/zmts/802dc9c3510d79fd40f9dc38a12bccfc (дата обращения: 21.05.2023);

7) Argon2, URL: https://github.com/P-H-C/phc-winner-argon2 (дата обращения: 18.05.2023);

8) Изучение и улучшение условий труда программистов на примере фирмы «Шатл-С», URL: https://studbooks.net/1362938/menedzhment/izuchenie_uluchshenie_usloviy_truda_programmistov_primere_firmy_shatl (дата обращения: 18.05.2023);

9) Чрезвычайные ситуации мирного и военного времени, URL: https://www.referat911.ru/Bezopasnost-jiznedeyatelnosti/chrezvychajnye-situacii-mirnogo-i-voennogo/412879-2954950-place2.html (дата обращения: 25.05.2023);

10) Понятия «опасность» и «безопасность». Виды опасностей, URL: https://studref.com/675560/bzhd/ponyatiya_opasnost_bezopasnost_vidy_opasnostey_prirodnye_antropogennye_tehnogennye_globalnye (дата обращения: 25.05.2023).

ПРИЛОЖЕНИЕ А
ЛИСТИНГ ПРОГРАММЫ

GraphHealthCheck.cs

```
using
Microsoft.Extensions.Diagnostics.HealthChecks;
using Neo4jClient;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager
{
   public class GraphHealthCheck : IHealthCheck
   {

      private readonly IGraphClient _client;
      public GraphHealthCheck(IGraphClient client)
      {
         _client = client;
      }

      public async Task<HealthCheckResult>
CheckHealthAsync(HealthCheckContext context,
CancellationToken cancellationToken = default)
      {

         var healthCheckResultHealthy = await
CheckNeo4jGraphConnectionAsync();


         if (healthCheckResultHealthy)
         {
            return HealthCheckResult.Healthy("neo4j
graph db health check success");
         }

         return HealthCheckResult.Unhealthy("neo4j
graph db health check success"); ;
      }

      private async Task<bool>
CheckNeo4jGraphConnectionAsync()
      {
         try
         {
            await _client.ConnectAsync();
         }

         catch (Exception)
         {
            return false;
         }

         return true;
      }
   }
}
```

Neo4jSettings.cs

```
namespace DbManager
{
   public class Neo4jSettings
   {
      public Uri Neo4jConnection { get; set; }

      public string Neo4jUser { get; set; }

      public string Neo4jPassword { get; set; }

      public string Neo4jDatabase { get; set; }
   }
}
```

ServiceRegistration.cs

```
using DbManager.Neo4j.Implementations;
using DbManager.Neo4j.Interfaces;
using Microsoft.Extensions.DependencyInjection;
using Neo4jClient;
using DbManager.Data.Nodes;
using DbManager.Services;
using DbManager.Neo4j.DataGenerator;
using Microsoft.Extensions.Configuration;
using DbManager.Data.Relations;
using DbManager.Data;

namespace DbManager
{
   public static class ServiceRegistration
   {
      public static void AddDbInfrastructure(this
IServiceCollection services, IConfiguration
configuration)
      {
         // Fetch settings object from configuration
         var settings = new Neo4jSettings();

configuration.GetSection("Neo4jSettings").Bind(setti
ngs);

         // This is to register Neo4j Client Object as a
singleton
         services.AddSingleton<IGraphClient,
BoltGraphClient>(op => {
                  var graphClient = new
BoltGraphClient(settings.Neo4jConnection,
settings.Neo4jUser, settings.Neo4jPassword);
                  graphClient.ConnectAsync().Wait();
```

```
if(Convert.ToBoolean(configuration.GetSection("ApplicationSettings:GenerateData").Value) == false)
                    PrepareData(graphClient, configuration.GetSection("ClientAppSettings:PathToPublicSourceDirecroty").Value, configuration.GetSection("ClientAppSettings:DirectoryWithDishImages").Value);
                return graphClient;
            });

        services.AddSingleton<IRepositoryFactory, RepositoryFactory>();

        services.AddTransient<IPasswordService, PasswordService>();

        services.AddTransient<DataGenerator>();
        services.AddSingleton<GeneratorService>();

        // This is the registration for custom repository class

services.AddTransient<IGeneralRepository<Order>, OrderRepository>();

services.AddTransient<IGeneralRepository<Dish>, DishRepository>();

services.AddTransient<IGeneralRepository<User>, UserRepository>();

services.AddTransient<IGeneralRepository<Client>, ClientRepository>();

services.AddTransient<IGeneralRepository<DeliveryMan>, DeliveryManRepository>();
    }

    private static void PrepareData(IGraphClient graphClient, string pathToPublicClientAppDirectory, string dirWithDishImages)
    {
        var categoryRepo = new GeneralRepository<Category>(graphClient);
        var dishRepo = new GeneralRepository<Dish>(graphClient);

        OrderState.OrderStatesFromDb = new GeneralRepository<OrderState>(graphClient).GetNodesAsync().Result;

        Category.CategoriesFromDb = categoryRepo.GetNodesAsync().Result;

        foreach (var category in Category.CategoriesFromDb)
        {
            var categoryDishes = categoryRepo.GetRelationsOfNodesAsync<ContainsDish, Dish>(category).Result.Select(h=>(Dish)h.NodeTo);

            foreach (var dish in categoryDishes)
            {
                var pathToDishDir = PathToDirWithDish(pathToPublicClientAppDirectory, dirWithDishImages, category.LinkName, dish.Id.ToString());
                if (Directory.Exists(pathToDishDir))
                {
                    dish.Images = Directory
                        .GetFiles(pathToDishDir)
                        //получаемый путь
                        // /dishes/{Название категории на англ}/{Guid}/{Название файла}
                        .Select(h => ConvertFromIOPathToInternetPath_DirWithDish(pathToPublicClientAppDirectory, h))
                        .ToList();


dishRepo.UpdateNodeAsync(dish).Wait();
                }
                else
                {

Directory.CreateDirectory(pathToDishDir);
                }
            }
        }
    }

    public static string PathToDirWithDish(string pathToPublicClientAppDirectory, string dirWithDishImages, string categoryLink, string dishId)
    {
        var pathToDishesDir = Path.Combine(pathToPublicClientAppDirectory, dirWithDishImages);
        var pathToCategoryDir = Path.Combine(pathToDishesDir, categoryLink);
        var pathToDishDir = Path.Combine(pathToCategoryDir, dishId);

        return pathToDishDir;
    }

    public static string ConvertFromIOPathToInternetPath_DirWithDish(string pathToPublicClientAppDirectory, string pathToImage)
    {
        pathToImage = pathToImage
            .Replace(pathToPublicClientAppDirectory, "")
```

```
            .Replace('\\', '/');

        return Path.Combine("/", pathToImage);
      }
   }
}
```

IModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data
{
   public interface IModel
   {
      public Guid Id { get; set; }
   }
}
```

INode.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data
{
   public interface INode : IModel
   {
   }
}
```

IRelation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data
{
   public interface IRelation : IModel
   {
      INode NodeFrom { get; set; }
      INode NodeTo { get; set; }

      Guid? NodeFromId { get; set; }
      Guid? NodeToId { get; set; }
   }
}
```

Node.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data
{
   public abstract class Node : INode
   {
      public Guid Id { get; set; }
   }
}
```

OrderStateEnum.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data
{
   [Flags]
   public enum OrderStateEnum
   {
      InQueue = 1,
      Cooking = 2,
      WaitDeliveryMan = 4,
      Delivering = 8,
      Finished = 16,
      Cancelled = 32,
   }
}
```

Relation.cs

```
using Neo4jClient;
using Newtonsoft.Json;

namespace DbManager.Data
{
   public class Relation<TFrom, TTo> : IRelation
where TFrom : class, INode where TTo : class, INode
   {
      public Guid Id { get; set; }

      private TFrom? _nodeFrom;
      private TTo? _nodeTo;

      [Neo4jIgnore]
      public INode NodeFrom
      {
         get => _nodeFrom;
         set
         {
            NodeFromId = value.Id;
            _nodeFrom = (TFrom)value;
         }
      }

      [Neo4jIgnore]
      public INode NodeTo
```

```
        {
            get => _nodeTo;
            set
            {
                NodeToId = value.Id;
                _nodeTo = (TTo)value;
            }
        }

        public Guid? NodeFromId { get; set; }
        public Guid? NodeToId { get; set; }

    }
}
```

User.cs

```
using Neo4jClient;

namespace DbManager.Data
{
    public class User : Node
    {
        public string Login { get; set; }
        public List<byte> PasswordHash { get; set; }
        public string PhoneNumber { get; set; }
        public string Address { get; set; }
        public string Name { get; set; }
        [Neo4jDateTime]
        public DateTime? Born { get; set; }
        public Guid RefreshToken { get; set; }
        [Neo4jDateTime]
        public DateTime RefreshTokenCreated { get;
set; }
        public bool IsBlocked { get; set; }
    }
}
```

JwtTokenInfoOutDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class JwtTokenInfoOutDTO
    {
        public string JwtToken { get; set; }
        public DateTime ValidTo { get; set; }
        public List<string> RoleNames { get; set; }
    }
}
```

KitchenWorkerOutDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class KitchenWorkerOutDTO
    {
        public Guid Id { get; set; }
        public string Login { get; set; }
        public string PhoneNumber { get; set; }
        public string Name { get; set; }
        public DateTime? Born { get; set; }
        public bool IsBlocked { get; set; }
        public string? JobTitle { get; set; }

        public DateTime GotJob { get; set; }
    }
}
```

ManipulateDishDataInDTO.cs

```
using Microsoft.AspNetCore.Http;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class ManipulateDishDataInDTO
    {
        public Guid? Id { get; set; }
        public string? Name { get; set; }
        public string? Description { get; set; }
        public double? Price { get; set; }
        public int? Weight { get; set; }
        public bool? IsAvailableForUser { get; set; }
        public bool? IsDeleted { get; set; }

        public string? CategoryId { get; set; }
        public IFormFileCollection? ImagesFiles { get;
set; }
    }
}
```

ManipulateOrderDataInDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class ManipulateOrderDataInDTO
    {
        public int? NewCount { get; set; }
        public string? OrderId { get; set; }
        public string? DishId { get; set; }
        public string? ReasonOfCancel { get; set; }
```

```
    }
}
```

ManipulateUserDataInDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class ManipulateUserDataInDTO
    {
        public string UserId { get; set; }
        public string? ChangeRole { get; set; }
    }
}
```

OrderOutDTO.cs

```
using DbManager.Data.Relations;
using Neo4jClient;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class OrderOutDTO
    {
        public Guid Id { get; set; }
        public double Price { get; set; }
        public int SumWeight { get; set; }
        public string DeliveryAddress { get; set; }
        public string PhoneNumber { get; set; }
        public string? Review { get; set; }
        public int? ClientRating { get; set; }

        public List<OrderStateItemOutDTO> Story {
get; set; } = new List<OrderStateItemOutDTO>();
    }
}
```

OrderStateItemOutDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class OrderStateItemOutDTO
    {
        public DateTime? TimeStartState { get; set; }
        public string? Comment { get; set; }
        public Guid OrderStateId { get; set; }
        public int NumberOfStage { get; set; }
        public string NameOfState { get; set; }
        public string DescriptionForClient { get; set; }
    }
}
```

PlaceAnOrderInDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class PlaceAnOrderInDTO
    {
        public string DeliveryAddress { get; set; }
        public string PhoneNumber { get; set; }
        public string? Comment { get; set; }
    }
}
```

ProfileUserOutDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class ProfileUserOutDTO
    {
        public string Login { get; set; }
        public string PhoneNumber { get; set; }
        public string Name { get; set; }
        public DateTime? Born { get; set; }

        // Client props
        public double? Bonuses { get; set; }

        //KitchenWorker props
        public string? JobTitle { get; set; }

        //Admin props

    }
}
```

ReviewOrderInDTO.cs

```
using Neo4jClient;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
```

```
{
    public class ReviewOrderInDTO
    {
        public string? OrderId { get; set; }
        public int? ClientRating { get; set; }
        public string? Review { get; set; }
    }
}
```

StatisticQueryDataItemOutDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class StatisticQueryDataItemOutDTO
    {
        public string X { get; set; }
        public List<double> Y { get; set; }
    }
}
```

StatisticQueryInfoOutDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class StatisticQueryInfoOutDTO
    {
        public string NameQuery { get;set; }
        public string LinkToQuery { get;set; }
        public string ChartName { get;set; }
        public bool NeedDataRange { get;set; }
        public List<string>? NameDatasets { get; set; }
    }
}
```

UserForAdminOutDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class UserForAdminOutDTO
    {
        public Guid Id { get; set; }
        public string Login { get; set; }
        public string PhoneNumber { get; set; }
        public string Name { get; set; }
        public DateTime? Born { get; set; }
        public bool IsBlocked { get; set; }
        public string Address { get; set; }

        public string Roles { get; set; }
    }
}
```

UserLoginInDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class UserLoginInDTO
    {
        public string Login { get; set; }
        public string Password { get; set; }
    }
}
```

UserSignupInDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.DTOs
{
    public class UserSignupInDTO
    {
        public string Login { get; set; }
        public string Password { get; set; }
        public string? PhoneNumber { get; set; }
        public string? Address { get; set; }
        public string Name { get; set; }
        public DateTime? Born { get; set; }
    }
}
```

Admin.cs

```
namespace DbManager.Data.Nodes
{
    public class Admin: User, INode
    {
    }
}
```

Category.cs

```
using Neo4jClient;
using Newtonsoft.Json;

namespace DbManager.Data.Nodes
```

```
{
   public class Category : Node, INode
   {
      public string Name { get; set; }
      public string? Description { get; set; }
      public string LinkName { get; set; }
      public int CategoryNumber { get; set; }

      /// <summary>
      /// Categories loading from DB when app starts
      /// </summary>
      [JsonIgnore]
      [Neo4jIgnore]
      public static List<Category> CategoriesFromDb
= new List<Category>();
   }
}
```

Client.cs

```
using DbManager.Data.Relations;
using Neo4jClient;

namespace DbManager.Data.Nodes
{
   public class Client : User, INode
   {
      public double Bonuses { get; set; }

      [Neo4jIgnore]
      public List<Ordered>? ClientOrders { get; set; }
   }
}
```

DeliveryMan.cs

```
using DbManager.Data.Relations;
using Neo4jClient;

namespace DbManager.Data.Nodes
{
   public class DeliveryMan : User, INode
   {
      public int MaxWeight { get; set; }

      [Neo4jIgnore]
      public List<DeliveredBy>? DeliveredOrders {
get; set; }
   }
}
```

Dish.cs

```
using DbManager.Data.Relations;
using Neo4jClient;

namespace DbManager.Data.Nodes
{
   public class Dish : Node, INode
   {
      public string Name { get; set; }
      public string Description { get; set; }
      public double Price { get; set; }
      public int Weight { get; set; }
      public bool IsAvailableForUser { get; set; }
      public bool IsDeleted { get; set; }
      /// <summary>
      /// Images of product. First image is main
      /// </summary>
      public List<string> Images { get; set; }

      [Neo4jIgnore]
      public List<OrderedDish>? Orders { get; set; }
   }
}
```

Kitchen.cs

```
using DbManager.Data.Relations;
using Neo4jClient;

namespace DbManager.Data.Nodes
{
   public class Kitchen : Node, INode
   {
      public string Address { get; set; }

      [Neo4jIgnore]
      public List<CookedBy>? PreparedOrders { get;
set; }
   }
}
```

KitchenWorker.cs

```
using DbManager.Data.Relations;
using Neo4jClient;

namespace DbManager.Data.Nodes
{
   public class KitchenWorker : User, INode
   {
      public string JobTitle { get; set; }

      [Neo4jIgnore]
      public WorkedIn? Kitchen { get; set; }
   }
}
```

Order.cs

```
using DbManager.Data.Relations;
using Neo4jClient;
using Newtonsoft.Json;

namespace DbManager.Data.Nodes
{
   public class Order : Node, INode
   {
      public double Price { get; set; }
      public int SumWeight { get; set; }
      public string DeliveryAddress { get; set; }
      public string PhoneNumber { get; set; }
```

```
        [Neo4jIgnore]
        public List<HasOrderState> Story { get; set; } =
new List<HasOrderState>();

        public string StoryJson
        {
            get
            {
                return JsonConvert.SerializeObject(Story,
Formatting.Indented, new JsonSerializerSettings() {
NullValueHandling = NullValueHandling.Ignore});
            }
            set
            {
                Story =
JsonConvert.DeserializeObject<List<HasOrderState>
>(value, new JsonSerializerSettings() {
NullValueHandling = NullValueHandling.Ignore });
            }
        }

        [Neo4jIgnore]
        public List<OrderedDish>? OrderedObjects {
get; set; }
        [Neo4jIgnore]
        public DeliveredBy? DeliveredMan { get; set; }
        [Neo4jIgnore]
        public Ordered Client { get; set; }
        [Neo4jIgnore]
        public CookedBy? Kitchen { get; set; }

    }
}
```

OrderState.cs

```
using Neo4jClient;
using Newtonsoft.Json;

namespace DbManager.Data.Nodes
{
    public class OrderState : Node, INode
    {
        public int NumberOfStage { get; set; }
        public string NameOfState { get; set; }
        public string DescriptionForClient { get; set; }

        /// <summary>
        /// Order states loading from DB when app starts
        /// </summary>
        [JsonIgnore]
        [Neo4jIgnore]
        public static List<OrderState>
OrderStatesFromDb = new List<OrderState>();

        [Neo4jIgnore]
        public List<Order> Orders { get; set; }
    }
}
```

ContainsDish.cs

```
using DbManager.Data.Nodes;

namespace DbManager.Data.Relations
{
    /// <summary>
    /// Category -> Dish
    /// </summary>
    public class ContainsDish : Relation<Category,
Dish>
    {

    }
}
```

CookedBy.cs

```
using DbManager.Data.Nodes;
using Neo4jClient;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DbManager.Data.Relations
{
    /// <summary>
    /// Kitchen -> Order
    /// </summary>
    public class CookedBy : Relation<Kitchen, Order>
    {
    }
}
```

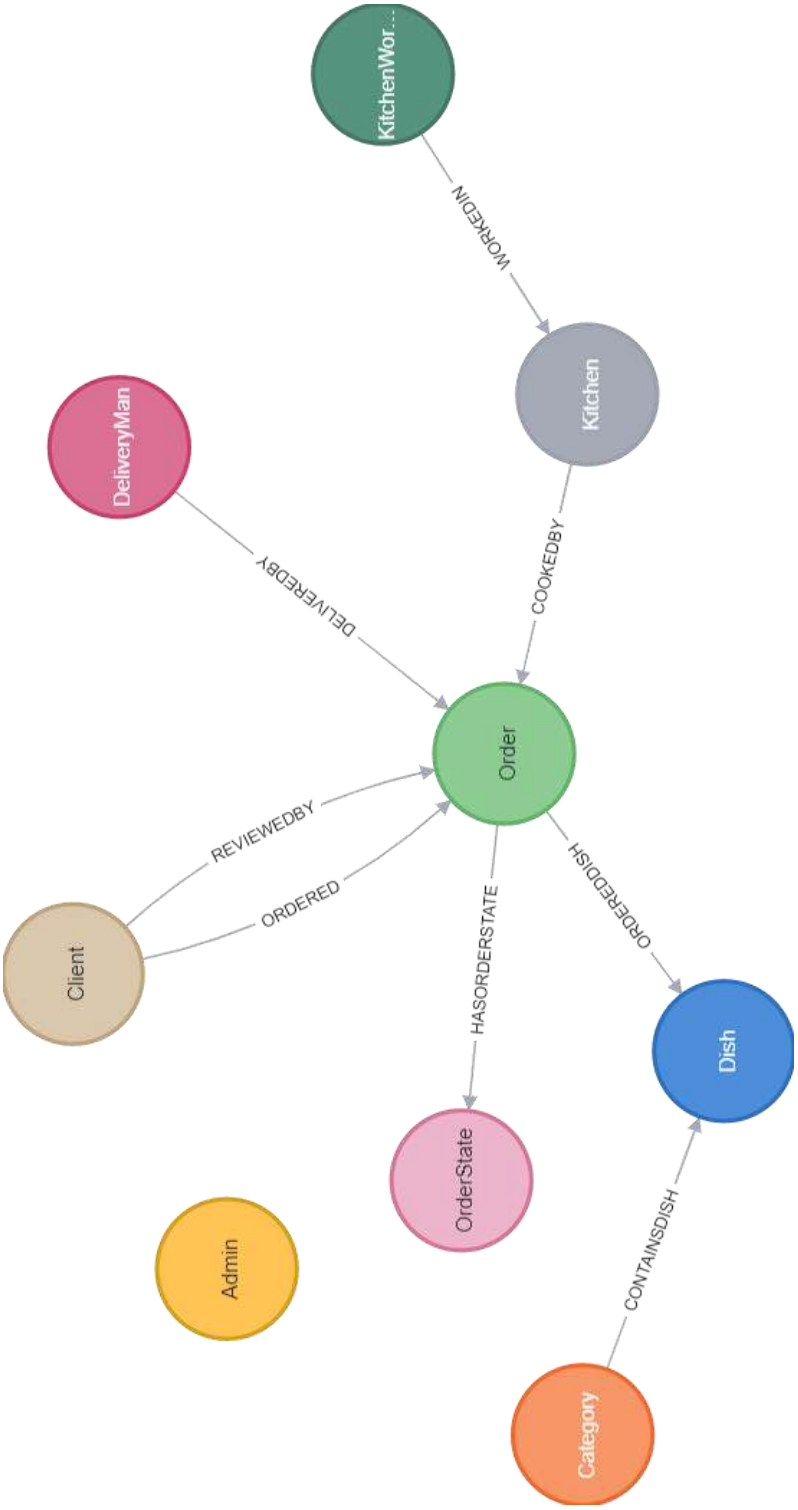
DeliveredBy.cs

```
using DbManager.Data.Nodes;
using Neo4jClient;

namespace DbManager.Data.Relations
{
    /// <summary>
    /// DeliveryMan -> Order
    /// </summary>
    public class DeliveredBy : Relation<DeliveryMan,
Order>
    {

    }
}
```

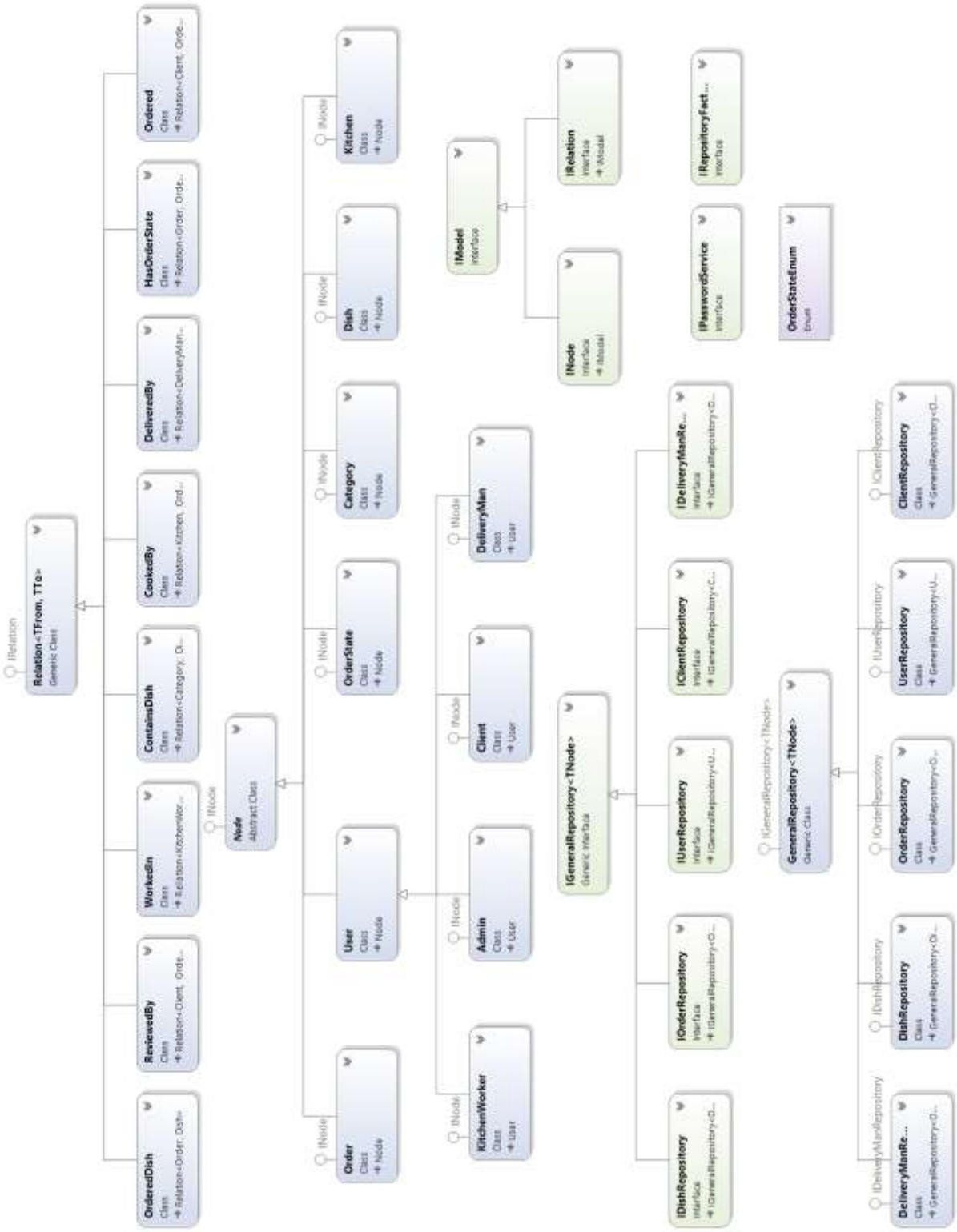
HasOrderState.cs

```
using DbManager.Data.Nodes;
using Neo4jClient;

namespace DbManager.Data.Relations
{
```

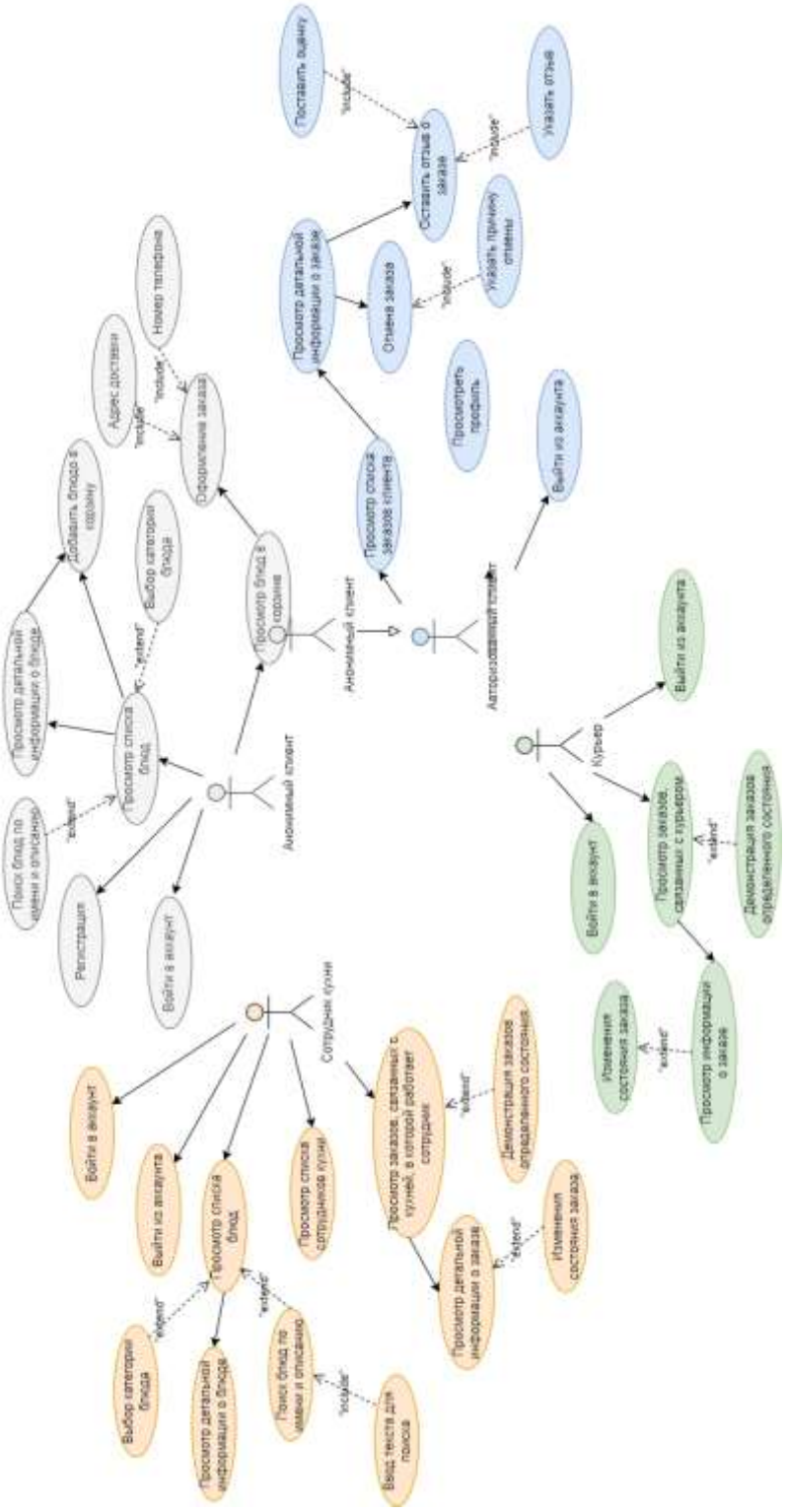
ПРИЛОЖЕНИЕ Б
ГРАФИЧЕСКАЯ ЧАСТЬ



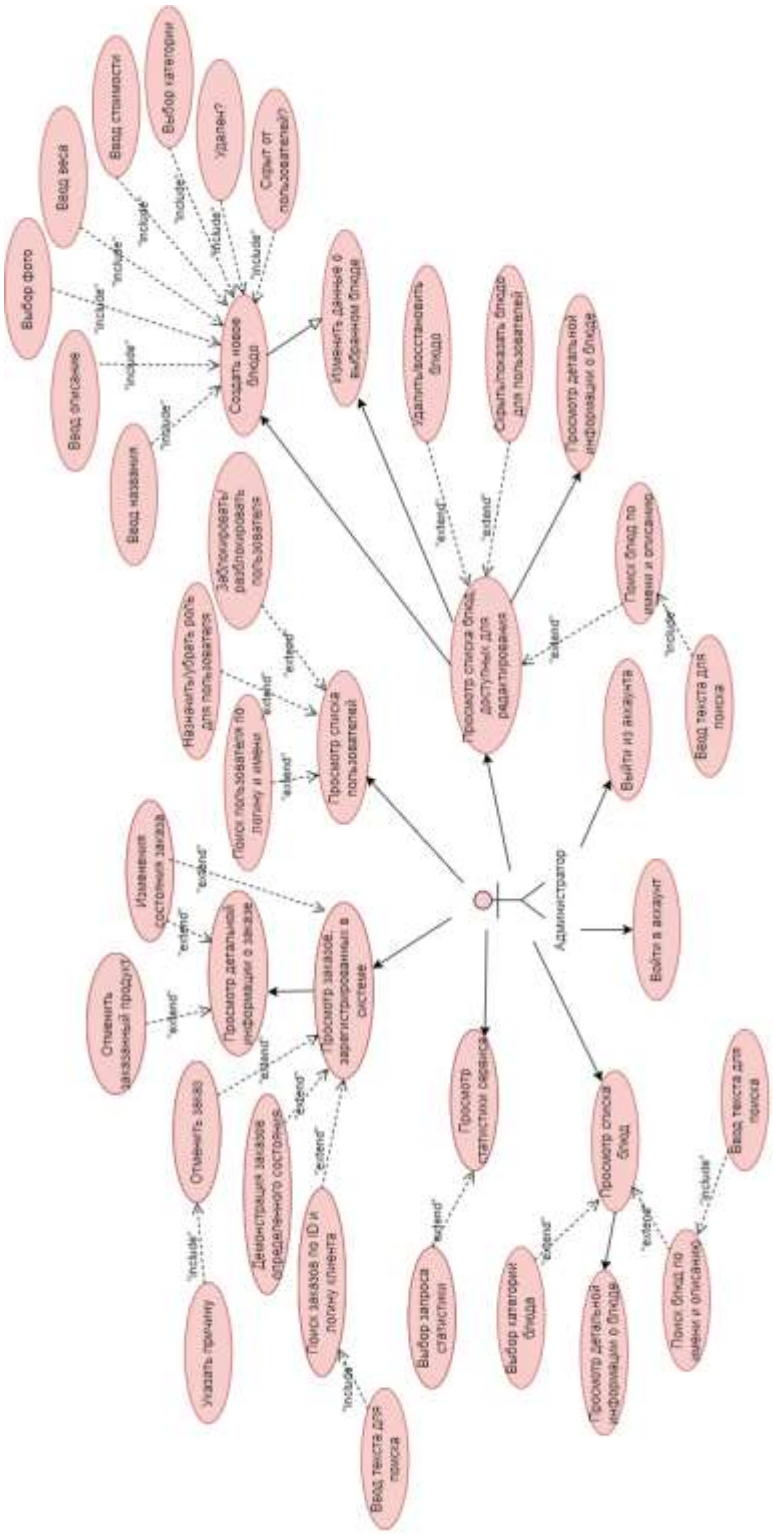
					09.03.04.2023.18\6322.00.01 Д1	
Изм.	Лист	№ документа		Дата		
Разработ.		Жильцов В.А.			Схема графовой базы данных	Буква / Лист / Листов
Руковдит.		Морозова О.В.				ДонНТУ, ФИСП
Н. контр.		Коломойцева И.А.				Кафедра ПИ группа ПИ-18а
Зав. каф.		Зори С.А.				



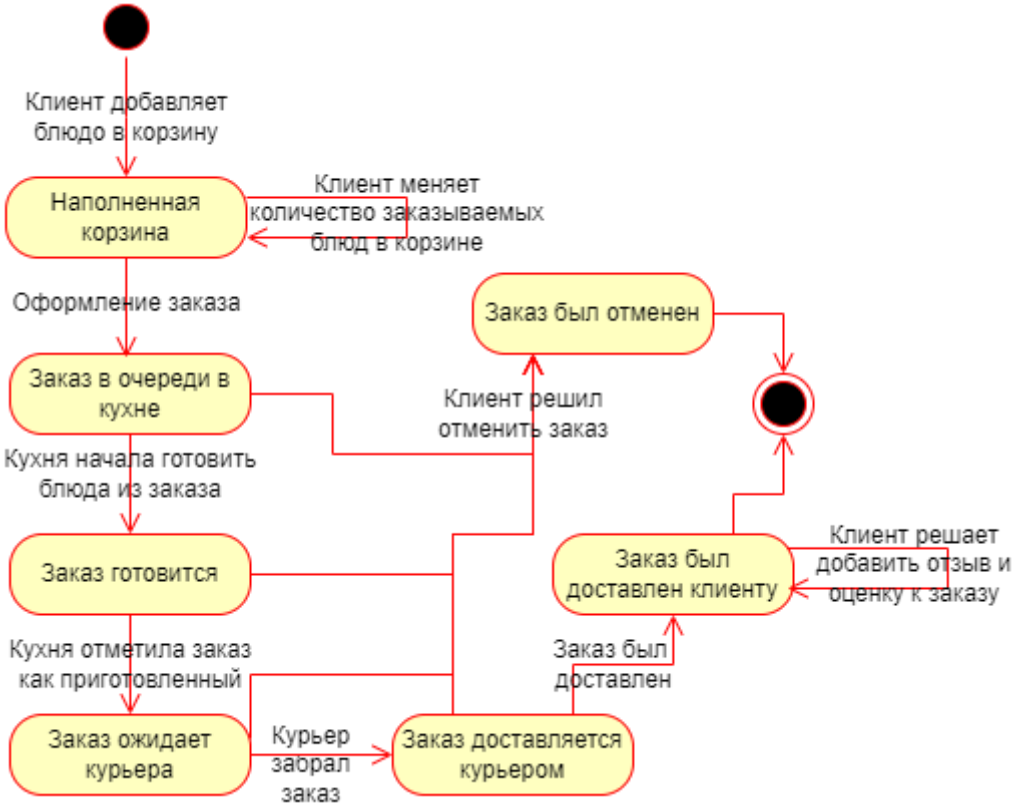
					09.03.04.2023.18\6322.00.02 Д2	
		№				
Разработ.		Жильцов В.А.			Диаграмма классов модуля работы с базой данных	Буква / Лист / Листов
Руковдит.		Морозова О.В.				ДонНТУ, ФИСП Кафедра ПИ группа ПИ-18а
Н. контр.		Коломойцева И.А.				
Зав. каф.		Зори С.А.				



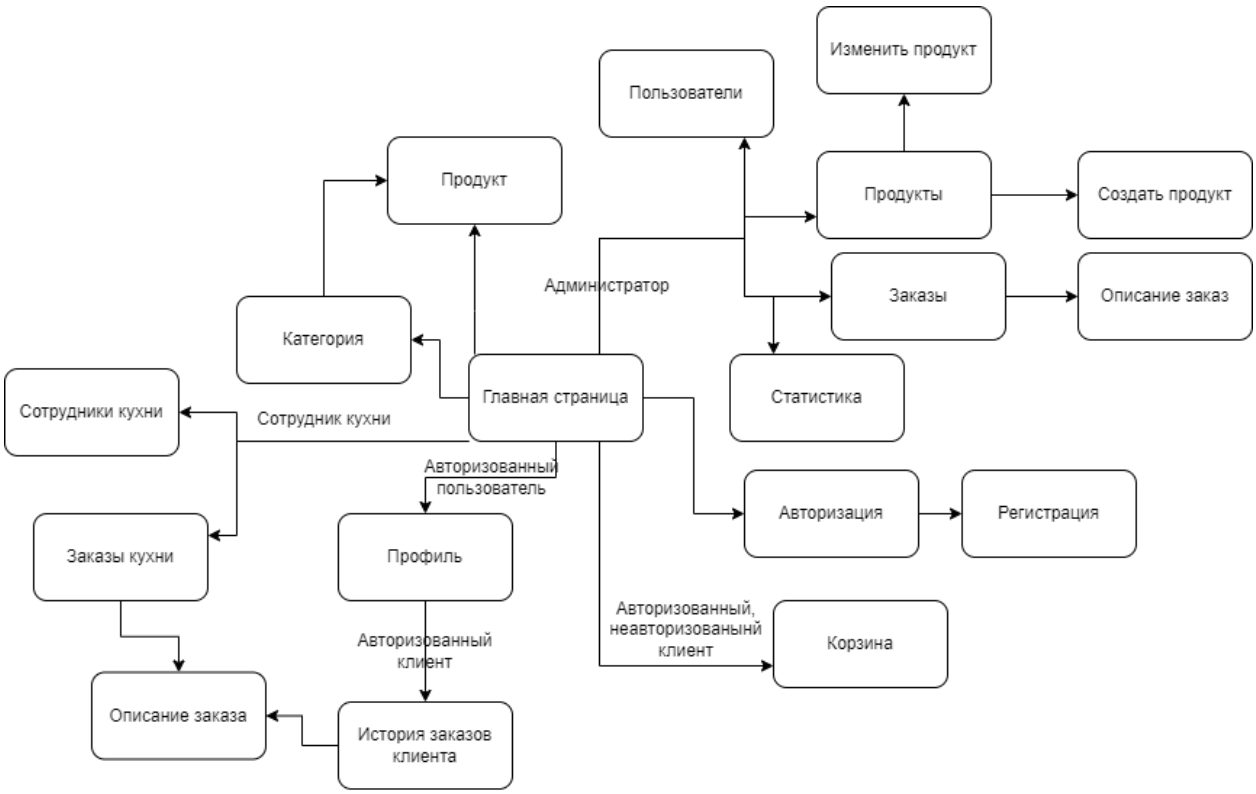
					09.03.04.2023.18\6322.00.03 ДЗ		
		№					
Разработ.	Жильцов В.А.			Диаграмма прецедентов сотрудника кухни, курьера, авторизованного клиента и анонимного клиента	Буква	Лист	Листов
Руководит.	Морозова О.В.				ДонНТУ, ФИСП Кафедра ПИ группа ПИ-18а		
Н. контр.	Коломойцева И.А.						
Зав. каф.	Зори С.А.						

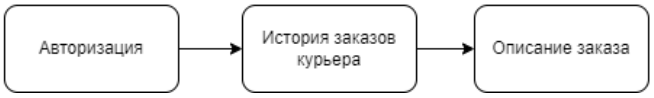


					09.03.04.2023.18\6322.00.04 Д4	
		№				
Разработ.	Жильцов В.А.				Буква	Лист	Листов
Руководит.	Морозова О.В.				Диаграмма прецедентов администратора	ДонНТУ, ФИСП Кафедра ПИ группа ПИ-18а	
Н. контр.	Коломойцева И.А.						
Зав. каф.	Зори С.А.						



					09.03.04.2023.18\6322.00.05 Д5	
		№				
Разработ.	Жильцов В.А.			Диаграмма состояний	Буква	Лист	Листов
Руковдит.	Морозова О.В.				ДонНТУ, ФИСП Кафедра ПИ группа ПИ-18а	
Н. контр.	Коломойцева И.А.					
Зав. каф.	Зори С.А.					





					09.03.04.2023.18\6322.00.06 Д6		
		№					
Разработ.	Жильцов В.А.			Карта навигации между страницами приложений	Буква	Лист	Листов
Руковдит.	Морозова О.В.				ДонНТУ, ФИСП		
Н. контр.	Коломойцева И.А.				Кафедра ПИ группа ПИ-18а		
Зав. каф.	Зори С.А.						

ПРИЛОЖЕНИЕ В

ПЕРЕЧЕНЬ ЗАМЕЧАНИЙ НОРМОКОНТРОЛЁРА

Перечень замечаний нормоконтролёра к дипломному проекту

Студента Жильцова Владимира Александровича группы ПИ-18а

Обозначение документа	Документ	Условное обозначение	Содержание замечания

Дата _____ Подпись _____