Частное учреждение образования

«Колледж бизнеса и права»

«Разработка веб ‒ приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph»

Пояснительная записка

к курсовому проекту по дисциплине

«Конструирование программ и языки программирования»

КП Т.318013.401

Руководитель проекта (А.В. Кривошеина)

Учащийся (А.А. Латушко)

2025

Изм.

Лист

№ докум.

Подпись

Датаа

Лист

3

КП Т.318013.401 Г ALTER TABLE заказы

ADD CONSTRAINT fk\_заказы\_пользователи

FOREIGN KEY (id\_пользователя)

REFERENCES пользователи(Id)

ON DELETE CASCADE;Ч

Разраб.

Латушко А. А.

Провер.

Кривошеина А.В

Реценз.

Н. Контр.

Утверд.

*Разработка веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph»*

Лит.

Листов

КБП

у

СОДЕРЖАНИЕ

[Введение 4](#_Toc200978271)

[1 Описание задачи 6](#_Toc200978272)

[1.1 Анализ предметной области 6](#_Toc200978273)

[1.2 Постановка задачи 7](#_Toc200978274)

[2 Проектирование веб-приложения 9](#_Toc200978275)

[2.1 Проектирование модели 9](#_Toc200978276)

[2.2 Требования к веб-приложению 10](#_Toc200978277)

[2.3 Структура веб-приложения 11](#_Toc200978278)

[2.4 Проектирование макета веб-приложения 12](#_Toc200978279)

[2.5 Программно-технические средства, необходимые для разработки приложения 13](#_Toc200978280)

[2.6 Защита и сохранность данных 15](#_Toc200978281)

[2.7 Организация и ведение информационной базы (модели) 15](#_Toc200978282)

[3 Реализация веб-приложения 22](#_Toc200978283)

[3.1 Описание разделов веб-приложения 22](#_Toc200978284)

[3.2 Разработка административной части приложения 23](#_Toc200978285)

[3.3 Разработка клиентской части приложения 24](#_Toc200978286)

[3.4 Описание используемых функций и процедур 25](#_Toc200978287)

[3.5 Функциональное тестирование 28](#_Toc200978288)

[4 Применение 30](#_Toc200978289)

[4.1 Назначение веб-приложения 30](#_Toc200978290)

[4.2 Программно-аппаратное обеспечение сервера и клиента 30](#_Toc200978291)

[Заключение 32](#_Toc200978292)

[Список информационных источников 33](#_Toc200978293)

[Приложения А Ментальная карта 34](#_Toc200978294)

[Приложение Б Логическая и физическая структура 35](#_Toc200978295)

[Приложение В Макеты страниц веб – приложения 37](#_Toc200978296)

[Приложение Г Код модулей веб – приложения 46](#_Toc200978297)

[Приложение Д Результаты работы веб – приложения 67](#_Toc200978298)

# Введение

Целью проекта на тему «Разработка веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph»» является создание эффективной, надёжной и удобной информационной системы, предназначенной для автоматизации ключевых процессов, связанных с оформлением заказов и оплатой услуг в сети кофеен. Проект направлен на упрощение взаимодействия клиентов с кофейней, повышение качества обслуживания, сокращение времени оформления заказов и централизованное управление меню и заказами через единое веб-приложение.

Разрабатываемое решение обеспечит пользователям интуитивно понятный интерфейс, включающий просмотр и фильтрацию ассортимента напитков и блюд, добавление позиций в заказ, оформление заказа с выбором способов оплаты и вариантов доставки или самовывоза, а также возможность бронирования столиков и отслеживания статуса бронирования. Пользователи также получат доступ к личному кабинету с историей заказов, управлением профилем и бронированиями. Для внутреннего использования приложение поддерживает формирование и экспорт отчётов о продажах, заказах и бронированиях, что упрощает анализ данных и управление бизнес-процессами кофейни.

Актуальность проекта обусловлена необходимостью повышения эффективности и прозрачности работы кофейни, оптимизации бизнес-процессов и внедрения современных цифровых технологий в сферу обслуживания клиентов. В условиях растущей конкуренции и цифровизации сервиса автоматизация формирования заказов и оплаты услуг кофейни «Paragraph» позволит значительно улучшить клиентский сервис, повысить лояльность посетителей и ускорить обработку заказов.

Решение поставленных задач подробно рассмотрено в пояснительной записке, которая состоит из четырёх основных разделов и описывает этапы анализа, проектирования, реализации и внедрения веб-приложения.

В первом разделе «Описание задачи» описываются цели автоматизации, анализируется предметная область кофейни, выявляются основные проблемы текущих процессов обслуживания клиентов, формулируются функциональные и нефункциональные требования к системе, а также рассматриваются существующие аналоги и их недостатки.

Во втором разделе «Проектирование системы» представлена архитектура веб-приложения, структура базы данных PostgreSQL, логика взаимодействия между серверной частью на ASP.NET и клиентским приложением на React, схемы пользовательского интерфейса и особенности обеспечения безопасности данных. Описываются сущности, связи между таблицами, а также обработка пользовательских сценариев.

В третьем разделе «Описание и реализация программного средства» рассматриваются применённые технологии и инструменты разработки (ASP.NET Core, React, PostgreSQL, HTML/CSS, JavaScript). Подробно описываются модули: регистрация и авторизация пользователей, управление меню (напитки, блюда), корзина и оформление заказа, история заказов, интеграция с платёжными системами, а также функциональность формирования и экспорта отчётов.

Четвёртый раздел «Применение» посвящён эксплуатации веб-приложения как для клиентов кофейни, так и для сотрудников компании. Рассматриваются сценарии использования, особенности адаптивного интерфейса для различных устройств (мобильные телефоны, планшеты, ПК), меры по защите персональных данных пользователей и организация поддержки и дальнейшего развития системы.

В заключении анализируются результаты работы, степень выполнения поставленных задач, эффективность внедрения системы в работу кофейни и направления для возможного расширения функциональности.

Список использованных источников включает нормативные документы, справочную литературу и интернет-ресурсы, применённые при проектировании и разработке системы.

В Приложении А представлены ментальная карта

В Приложении Б представлена логическая и физическая структура

В Приложении В представлены макеты страниц веб - приложения

В Приложении Г код модулей веб - приложения

В Приложении Д результаты работы веб - приложения.

Графическая часть содержит диаграмму классов, диаграмму развертывания, диаграмму прецедентов, структуру базы данных PostgreSQL и архитектурную схему веб-приложения «Paragraph».

# Описание задачи

## Анализ предметной области

Темой проекта является разработка веб-приложения «Paragraph», предназначенного для автоматизации процессов формирования заказов и оплаты услуг в сети кофеен. Предметной областью выступает сфера общественного питания и сервисного обслуживания, в частности — организация онлайн-заказа напитков и блюд, бронирования столиков, управления заказами и взаимодействия с клиентами.

Основными объектами предметной области являются:

* меню кофейни с напитками и блюдами, включая описания и изображения;
* категории и фильтры для удобного поиска по ассортименту;
* учетные записи пользователей (личные кабинеты);
* заказы с возможностью отслеживания;
* корзина;
* бронирование столиков с возможностью управления и отслеживания;
* интерфейс администраторов для управления контентом сайта.

Бизнес-процессы, реализуемые в системе, охватывают полный цикл покупки:

пользователь регистрируется или авторизуется, просматривает меню, применяет фильтры по категориям и характеристикам (тип напитка, размер, состав), добавляет позиции в корзину, оформляет заказ с выбором способа оплаты и опций доставки либо самовывоза, а также может забронировать столик с указанием даты и времени. После оформления заказ и бронирование сохраняются в системе и доступны для просмотра в личном кабинете пользователя. Пользователь также может сохранять любимые позиции в избранное для быстрого повторного заказа. Администратор получает доступ к управлению меню, заказами, бронированиями и пользователями через панель администратора.

Необходимость компьютерной обработки обусловлена высокой динамикой заказов и бронирований, большим объёмом данных, а также необходимостью обеспечения быстрой и безопасной обработки информации. Веб-приложение «Paragraph» автоматизирует хранение и обработку заказов и бронирований, взаимодействие с клиентами, а также формирование аналитических отчётов для управления кофейней. Безопасность обеспечивается через систему авторизации, разграничение ролей (пользователь/администратор) и защищённое хранение данных в базе PostgreSQL.

Ключевые задачи, подлежащие реализации:

* проектирование и реализация базы данных для хранения информации о товарах, заказах, пользователях, корзине и избранном;
* реализация системы регистрации и авторизации с защитой пользовательских данных;
* создание API-контроллеров для работы с товарами, корзиной, заказами, авторизацией и админ-панелью;
* разработка адаптивного фронтенда на ReactJS с поддержкой базовых пользовательских сценариев;
* реализация функции экспорта заказов в Word-документы для ведения отчётности;
* интеграция системы с пунктами выдачи и методами оплаты.

Целевую аудиторию составляют:

* посетители и постоянные клиенты сети кофеен «Paragraph»;
* бариста и сотрудники кофейни, обслуживающие заказы;
* менеджеры, управляющие меню, заказами и бронированиями;
* администраторы и контент-менеджеры сайта;
* специалисты, анализирующие эффективность работы кофейни.

На современном рынке общественного питания существует множество сервисов для заказа еды и напитков, однако не все обеспечивают полноценную интеграцию онлайн-заказов с управлением бронированиями и внутренними бизнес-процессами кофейни. Разрабатываемое веб-приложение «Paragraph» направлено на устранение этих ограничений за счёт адаптивной архитектуры, расширенного функционала и модульного API. Система предоставляет пользователям современный удобный интерфейс, интеграцию с внутренними процессами кофейни и возможности для масштабирования и дальнейшего развития.

## Постановка задачи

Основная задача разработки веб-приложения **«**Paragraph**»** заключается в создании современного и удобного инструмента для автоматизации ключевых бизнес-процессов, связанных с обслуживанием клиентов кофейни, формированием заказов, бронированием столов, управлением меню и оплатой услуг.

С помощью разрабатываемого программного обеспечения решаются следующие задачи в рамках предметной области:

* автоматизация отображения и фильтрации товаров. Приложение предоставляет пользователям удобный каталог с напитками, десертами и блюдами, с возможностью фильтрации по категориям, ключевым словам и популярности. Это упрощает выбор и ускоряет процесс оформления заказа;
* формирование и управление корзиной. Пользователи могут добавлять позиции в корзину, редактировать её содержимое, выбирать количество порций, оставлять комментарии к заказу и оформлять его с выбором способа получения (на месте или на вынос);
* личный кабинет и история заказов. Система регистрации и авторизации позволяет каждому пользователю вести собственный профиль, просматривать историю покупок, сохранять настройки и управлять своими предпочтениями;
* бронирование столиков. Система позволяет клиентам заранее забронировать столик, указав дату, время и количество гостей, что повышает удобство и снижает вероятность перегрузки зала;
* администрирование через панель администратора. Для сотрудников и менеджеров предусмотрен интерфейс для управления товарами, заказами, пользователями и категориями, а также возможность выгрузки отчётов по продажам в формате Word;
* безопасность и защита данных. Использование авторизации, разграничения ролей и безопасного хранения информации в базе данных позволяет защитить персональные данные пользователей и обеспечить соответствие современным стандартам безопасности;
* адаптивный и интуитивно понятный интерфейс. Благодаря ReactJS, CSS-модулям и базовой адаптивной вёрстке интерфейс корректно отображается на различных устройствах, обеспечивая удобную навигацию как с компьютеров, так и с мобильных устройств.

На основе анализа бизнес-процессов в сфере общественного питания и работы кофейни были выделены следующие функции, подлежащие автоматизации в рамках веб-приложения:

* каталогизация и фильтрация товаров по характеристикам;
* формирование и оформление заказа с возможностью онлайн-оплаты и выбора точки кофейни;
* регистрация и авторизация пользователей, ведение профилей и персональных данных;
* управление корзиной;
* хранение истории заказов и генерация отчётов (в формате Word);
* управление ассортиментом, заказами и бронированиями через админ-панель;
* интеграция с платёжными модулями;
* обеспечение безопасности пользовательских данных через авторизацию и разграничение ролей.

На рынке присутствуют такие аналоги, как мобильные приложения и сайты сетей Starbucks, KFC, Black Star Coffee и других заведений, предоставляющих функции заказа и оплаты. Однако многие из них ограничены в возможностях кастомизации, не поддерживают экспорт отчётности в Word и не интегрированы с внутренней системой управления заведением. Разработка собственного веб-приложения **«**Paragraph**»** позволяет учесть все особенности внутренней работы кофейни, реализовать уникальные функции и обеспечить удобство как для клиентов, так и для сотрудников.

# Проектирование веб-приложения

## Проектирование модели

Диаграмма прецедентов веб-приложения **«**Paragraph**»** отражает структуру взаимодействия пользователей с системой, включая администратора и покупателя. Она демонстрирует все основные действия, доступные каждому типу пользователя, а также зависимости между ними.

Клиент начинает взаимодействие с системой через регистрацию и авторизацию, после чего получает доступ к основному функционалу: просмотру меню (каталога товаров), фильтрации по категориям (кофе, десерты, блюда), добавлению товара в корзину, оформлению заказа, а также бронированию столика. Добавление товара в корзину встроено в оформление заказа и связано через механизм *include*. Бронирование столика реализовано как расширение (*extend*) основного сценария взаимодействия.

Администратор имеет расширенные полномочия: управление заказами, меню (каталогом товаров) и столиками. Эти действия позволяют администрировать систему, контролировать содержимое и обеспечивать корректную работу веб-приложения.

Диаграмма прецедентов представлена на листе 1 графической части.

Диаграмма классов отображает внутреннюю структуру приложения **«**Paragraph**»** и показывает основные сущности, их свойства и связи между ними. Она необходима для построения базы данных, а также для описания логики работы системы.

Класс Пользователь содержит идентификатор idПользователя, электронную почту, пароль и ФИО. Пользователь ассоциирован с объектом Корзина, что означает, что каждый пользователь может иметь свою собственную корзину. Основные методы: зарегистрироваться() и авторизоваться() — обеспечивают доступ пользователя к системе.

Класс Корзина включает idКорзины, ссылку на пользователя и список товаров (товары: List<Ingredient>), что отражает возможность добавления нескольких товаров. Методы добавить\_товар(товар: Product) и удалить\_товар(товар: Product) позволяют управлять содержимым корзины. Примечание: в классе используется Ingredient, но по диаграмме логичнее заменить его на Product.

Класс Товар содержит поля: idТовара, название, описание, цена и категория. Метод получить\_информацию() возвращает текстовое описание товара. Товар может быть добавлен в корзину и участвовать в заказах.

Класс Заказ содержит поля: idЗаказа, Номер, ссылку на пользователя, дату заказа (дата\_заказа), количество и итоговую цену. Методы: оформить\_заказ() и отменить\_заказ() реализуют действия пользователя при оформлении покупки. Заказ связан с пользователем, от которого он исходит, и товарами, которые были выбраны.

Диаграмма классов представлена на листе 2 графической части.

Диаграмма развёртывания веб-приложения **«**Paragraph**»** иллюстрирует архитектуру системы и распределение компонентов между клиентской, серверной и облачной частями. Она отражает потоки данных, направления обмена информацией и назначение каждого узла.

На стороне клиентского узла расположен веб-браузер с адаптивным интерфейсом, реализованным с помощью React. Через протокол HTTPS осуществляется обмен с сервером.

Серверный узел включает в себя веб-сервер, на котором размещено серверное приложение на ASP.NET Core. Логика сервера взаимодействует с базой данных через SQL-запросы.

База данных развернута на PostgreSQL и содержит таблицы для пользователей, заказов, товаров, категорий и столиков.

Размещение приложения осуществляется через облачную платформу — например, Vercel (для фронтенда) и Railway/Azure (для бэкенда).

Диаграмма развёртывания представлена на листе 3 графической части.

## Требования к веб-приложению

При разработке веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph» необходимо учитывать широкий спектр требований, касающихся стилистического, графического, функционального и технического аспектов. Крайне важно обеспечить единообразие визуального оформления, соответствующего фирменному стилю, и интуитивную структуру пользовательского интерфейса. Интерфейс должен быть минималистичным, удобным и способствующим быстрому восприятию информации.

Графический дизайн веб-приложения должен отличаться визуальной чистотой, современностью и доступностью. Применение качественных изображений, иконок и акцентных элементов повышает визуальную привлекательность и уровень доверия со стороны пользователей. Элементы управления (меню, каталог, фильтры, кнопки) должны быть легко различимы, а логика взаимодействия — интуитивно понятной. Все функциональные блоки должны быть визуально разграничены. Цветовая палитра выдержана в светлых и глубоких сине-зелёных тонах с основным цветом #0f3739, что подчёркивает атмосферу уюта, современности и технологичности кофейни.

Шрифт, применяемый в интерфейсе — Montserrat, как современный, геометричный и легко читаемый. Рекомендуется ограничить палитру до двух-трёх начертаний. Цвет текста должен чётко выделяться на фоне, заголовки должны быть визуально доминирующими, а важные элементы — выделяться жирным начертанием или цветом.

Веб-приложение должно корректно работать во всех современных браузерах (Chrome, Firefox, Safari, Edge) и адаптироваться под устройства разного типа: ПК, планшеты, смартфоны. Это обеспечит максимальное удобство и доступность для всех категорий пользователей вне зависимости от типа устройства.

Контент веб-приложения должен быть актуальным, структурированным и легко воспринимаемым. В каталоге услуг и товаров кофейни информация должна представляться кратко и чётко: название, изображение, цена, метка скидки или новинки. Карточка услуги включает расширенные характеристики и отзывы пользователей. Система управления контентом (CMS) должна позволять легко добавлять, редактировать и удалять товары и услуги, а также управлять текстовыми разделами сайта (профиль, политика конфиденциальности, о компании).

Также важно обеспечить надёжную защиту персональных данных, включая безопасную регистрацию, авторизацию и разграничение прав доступа.

Все ключевые функции должны быть доступны на каждой странице, а информация — отображаться логично и удобно для восприятия. Обязательна реализация функций поиска и фильтрации товаров и услуг, а также управление корзиной, оформлением заказа и историей покупок.

Для разработки веб-приложения «Paragraph» определены следующие требования к визуальному оформлению, обеспечивающие единообразие и привлекательность интерфейса:

Шрифт — Montserrat:

* «Montserrat Light» — для основного текста, придаёт интерфейсу лёгкость и современность;
* «Montserrat Regular» — универсальный стиль для большинства текстов;
* «Montserrat SemiBold/Bold» — используется для заголовков и акцентов.

Рекомендуемые размеры шрифтов для интерфейса:

* заголовки 1 уровня 24px;
* заголовки 2 уровня 20px;
* заголовки 3 уровня 18px;
* основной текст 16px;
* подписи, технический текст: 14px.

Цветовая палитра интерфейса:

* основной текст чёрный #000000, белый #FFFFFF;
* основной фон светло-серый #F5F5F5;
* хедер/меню — глубокий сине-зелёный #0f3739;
* фон карточек товара #FFFFFF, блоков — #E8EAE5;
* акценты и кнопки — насыщенный бирюзово-синий #1ca3a3, тёмно-зелёный #144b4a.

Такая система визуального оформления позволяет поддерживать строгую и понятную структуру интерфейса, а также способствует созданию единого, эстетически привлекательного и уютного облика веб-приложения кофейни «Paragraph».

## Структура веб-приложения

Структура веб-приложения представляет собой упорядоченную схему, показывающую, как различные компоненты и функциональные элементы взаимодействуют между собой, а также каким образом они организованы. Она определяет способ отображения информации пользователям и взаимодействие пользователей с приложением. Структура веб-приложения (ментальная карта) представлена на приложении А на рисунке А.1.

## Проектирование макета веб-приложения

Веб-сайт был спроектирован в онлайн-сервисе Figma, который предоставляет мощные инструменты для прототипирования и проектирования интерфейсов. С помощью Figma были созданы интерактивные элементы, включая кнопки, формы, меню и векторные изображения, что позволило реализовать современный и удобный дизайн.

Сайт использует шрифт Montserrat, который обеспечивает отличную читаемость и поддерживает современный стиль дизайна. Шрифт применяется для всех текстов, включая заголовки, описания и подписи к кнопкам.

Дизайн сайта имеет в основном белый, голубой, светло-фиолетовый цвета. Он был спроектирован таким образом, чтобы удовлетворить интересы пользователей. Это улучшает качество пользовательского опыта.

Размер макета был выбран исходя из размеров окна браузера.

Главная страница сайта имеет размер 1920 пикселей по ширине и 2940 по высоте.

Раздел «Каталог товаров» имеет размер 1920 пикселей по ширине и 2940 по высоте.

Раздел «Корзина (с товарами)» имеет размер 1920 пикселей по ширине и 1080 по высоте.

Раздел «Корзина (пустая)» имеет размер 1920 пикселей по ширине и 1080 по высоте.

Раздел «Оформление заказа» имеет размер 1920 пикселей по ширине и 1080 по высоте.

Раздел «Привязка новой карты» имеет размер 1920 пикселей по ширине и 1080 по высоте.

Раздел «Политика персональных данных» имеет размер 1920 пикселей по ширине и 1080 по высоте.

Форма авторизации имеет размер 1920 пикселей по ширине и 1080 по высоте.

Форма регистрации имеет размер 1920 пикселей по ширине и 1080 по высоте.

Раздел «Профиль» имеет размер 1920 пикселей по ширине и 1080 по высоте.

На приложении В на рисунке В.1 представлен макет главной страницы, на которой размещены баннер, категории товаров и блок акций.

На приложении В на рисунке В.2 представлен макет страницы «Каталог товаров», содержащей фильтры и карточки товаров.

На приложении В на рисунке В.3 представлен макет страницы «Корзина (с товарами)», отображающей список товаров и сумму заказа.

На приложении В на рисунке В.4 представлен макет страницы «Корзина (пустая)», которая отображается, если пользователь ещё не добавил товары.

На приложении В на рисунке В.5 представлен макет страницы «Оформление заказа», содержащей формы ввода адреса и выбора способа оплаты.

На приложении В на рисунке В.6 представлен макет формы авторизации, содержащей поля ввода e-mail и пароля.

На приложении В на рисунке В.7 представлен макет формы регистрации, содержащей поля для заполнения личных данных.

На приложении В на рисунке В.8 представлен макет страницы «Профиль», где отображаются личные данные пользователя.

## Программно-технические средства, необходимые для разработки приложения

Для реализации веб-приложения «Paragraph» используется современный стек технологий, включающий языки программирования, фреймворки, библиотеки и инструменты разработки, которые обеспечивают высокую производительность, надёжность и адаптивность системы. Приложение построено на клиент-серверной архитектуре с поддержкой сетевого взаимодействия, регистрации пользователей, формирования и оплаты заказов в режиме онлайн.

С помощью выбранных технологий решаются следующие задачи в рамках реализации проекта:

* построение серверной логики приложения. Для разработки backend-части используется язык программирования C# с фреймворком ASP.NET Core MVC [5]. Он выбран благодаря своей высокой надёжности, масштабируемости, встроенной поддержке безопасной авторизации и удобной интеграции с базами данных. ASP.NET Core позволяет строить REST API, что идеально подходит для взаимодействия с клиентской частью;
* создание динамичного интерфейса. Для frontend-части применяется библиотека ReactJS на языке JavaScript [9]. React позволяет создавать компонентную структуру, повторно использовать интерфейсные блоки и реализовывать интерактивные элементы. Выбор React обусловлен его популярностью, богатой экосистемой и высокой производительностью при работе с изменяемыми данными;
* реализация структуры и стиля интерфейса. Для вёрстки используется HTML — язык гипертекстовой разметки, задающий структуру контента, а также CSS (в частности, CSS-модули), позволяющие применять адаптивные стили, задавать цветовые схемы, шрифты и элементы дизайна. Такой выбор обеспечивает кроссбраузерность и гибкость в оформлении интерфейса [6];
* взаимодействие между клиентом и сервером. Для отправки HTTP-запросов и получения данных используется библиотека Axios [8]. Она проста в использовании и обеспечивает эффективную работу с API: получение списка товаров, регистрация, оформление заказа и т.д.;
* работа с базой данных. Для хранения данных о пользователях, товарах, заказах и избранном используется PostgreSQL — мощная, надёжная и производительная реляционная СУБД с открытым исходным кодом. Для взаимодействия с базой данных применяется ORM-фреймворк Entity Framework Core, который упрощает доступ к данным и снижает количество прямых SQL-запросов за счёт использования модели сущностей. Такой подход позволяет работать с базой данных на уровне C#-объектов, обеспечивая удобство разработки и масштабируемость системы;
* организация среды разработки. Для клиентской и сервисной части применяется Visual Studio Code — лёгкий, но функциональный редактор, который идеально подходит для React-проектов [11]. Оба инструмента поддерживают Git, расширения и плагины;
* тестирование API и отладка. Для проверки и документирования API-запросов используется Postman [7]. Это приложение позволяет вручную тестировать маршруты, аутентификацию и отправку данных, обеспечивая уверенность в корректной работе backend-сервиса.

Веб-приложение «Paragraph» реализует полноценную сетевую архитектуру. Клиенты взаимодействуют с серверной частью через интернет по защищённому протоколу HTTPS, получая актуальные данные из базы, отправляя заказы и обновляя информацию в реальном времени. Такая архитектура обеспечивает стабильную работу как на стационарных компьютерах, так и на мобильных устройствах, предоставляя пользователям удобный доступ к функциям выбора услуг, корзины, оформления заказа и оплаты.

Выбор технологий обусловлен их надёжностью, широким сообществом, доступностью большого количества готовых решений и возможностью быстрой разработки, масштабирования и поддержки. Это позволяет обеспечить высокую производительность, безопасность и современный пользовательский опыт веб-приложения кофейни «Paragraph».

## Защита и сохранность данных

Веб-приложение «Paragraph» разработано с учётом современных требований безопасности, направленных на защиту персональных данных пользователей и предотвращение несанкционированного доступа.

Для обеспечения конфиденциальности реализована система регистрации и аутентификации, основанная на использовании уникальных логинов и паролей. Пользовательский профиль связан с конкретной сессией и надёжно идентифицируется в базе данных, что исключает возможность доступа к персональным данным со стороны третьих лиц.

Данные о пользователях, заказах, корзине и истории оплат сохраняются в базе данных PostgreSQL, доступ к которой осуществляется через защищённые серверные запросы с использованием API. Вся передача данных между клиентом и сервером происходит по защищённому протоколу HTTPS, что предотвращает перехват или подмену информации.

Для взаимодействия с приложением пользователь предоставляет только минимально необходимые разрешения:

* доступ к интернету — используется для отправки запросов к API при регистрации, авторизации, загрузке товаров, оформлении заказов и обновлении информации в профиле;
* сохранение токенов аутентификации — применяется для обеспечения постоянной сессии и упрощения повторного входа;
* взаимодействие с внешними платёжными системами — используется при выборе оплаты картой.

Серверная часть приложения реализована на ASP.NET Core с применением механизмов авторизации, включая безопасное хэширование паролей и использование JWT-токенов для защиты сессий. Это обеспечивает надёжную защиту данных пользователей, а также позволяет эффективно разграничивать доступ к различным ресурсам и функциям приложения.

Разграничение прав пользователей, шифрование данных при хранении и передаче, а также строгая структура API обеспечивают высокий уровень безопасности пользовательской информации. Архитектура приложения построена с учётом принципов «privacy by design» и «minimum access», что минимизирует риски утечек и несанкционированного доступа.

## Организация и ведение информационной базы (модели)

Диаграмма «сущность-связь» представляет логическую модель базы данных веб-приложения по автоматизации формирования заказа и оплаты услуг кофейни «Paragraph» и отражает основные сущности, их атрибуты и связи между ними. Она используется на этапе проектирования системы для определения структуры хранения данных и логики их взаимодействия.

Сущность «Клиент» содержит атрибуты UserID, ФИО, Email, Логин, Пароль и используется для идентификации при входе в систему и оформления заказов. Клиент может делать заказы и просматривать историю своих покупок. Связь с другими таблицами позволяет отслеживать активность и историю заказов конкретного клиента.

Сущность «Товар» включает атрибуты ProductID, Название, Цена, Категория и представляет собой наименования продукции кофейни (напитки, десерты и т.д.), доступной для заказа.

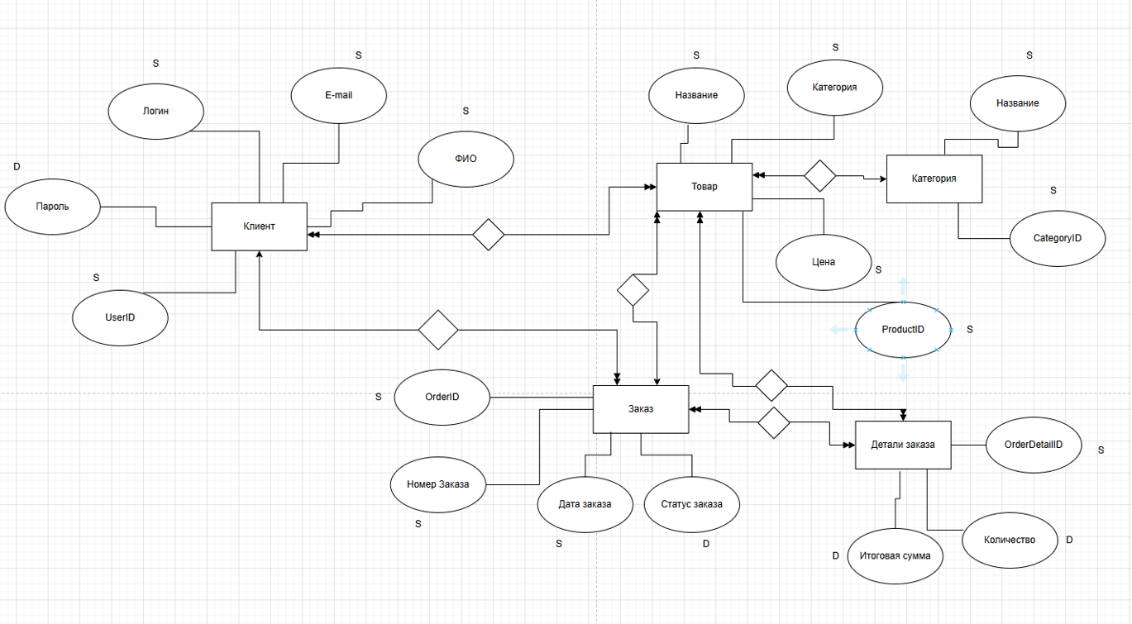
Сущность «Категория» представлена атрибутами CategoryID и Название и используется для группировки товаров по типу (например, кофе, чай, выпечка). Каждый товар принадлежит одной категории, что упрощает навигацию и фильтрацию в меню.

Сущность «Заказ» хранит информацию об оформленных заказах: OrderID, Номер заказа, Дата заказа, Статус заказа. Она фиксирует действия клиента при оформлении заказа и позволяет отслеживать его состояние.

Сущность «Детали заказа» (Order Details) содержит OrderDetailID, Количество и Итоговую сумму, определяя, какие товары входят в конкретный заказ и в каком количестве.

Связи между сущностями построены таким образом, чтобы обеспечить логическую целостность данных и правильную маршрутизацию между заказами, товарами и клиентами. Один клиент может иметь множество заказов, каждый заказ состоит из нескольких товаров, каждый товар привязан к определённой категории. Таким образом, обеспечивается полный цикл оформления и отслеживания заказов в кофейне «Paragraph».

Диаграмма «Сущность-связь» представлена на Рисунке 1.



1. – Диаграмма «Сущность-связь»

Приведённые таблицы базы данных были спроектированы на основании ER-диаграммы и отражают физическую реализацию логической структуры, разработанной в рамках предметной области веб-приложения «Paragraph» — системы автоматизации заказов и оплаты услуг кофейни. Каждая таблица соответствует отдельной сущности, представленной на диаграмме, а связи между таблицами реализованы с помощью внешних ключей, обеспечивающих целостность и согласованность данных.

Такая структура позволяет эффективно реализовать все заявленные функции: управление пользователями и заказами, обработку корзины, отображение каталога, бронирование столов и хранение данных о товарах. Кроме того, база данных обеспечивает надёжную основу для расширения функциональности приложения в будущем.

Для реализации этих функций была разработана следующая структура таблиц:

Таблица «Пользователи» хранит информацию о зарегистрированных пользователях.

Таблица 1 – Структура таблицы «Users»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| id | integer | 4 | Идентификатор пользователя |
| Name | character varying | 100 | Имя пользователя |
| Email | character varying | 100 | Электронная почта |
| PasswordHash | character varying | 255 | Хэш пароля |

Таблица «Продукты» содержит данные о товарах, доступных для покупки.

Таблица 2 – Структура таблицы «Products»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| id | serial | 4 | Идентификатор товара |
| name | text | - | Название товара |
| category | text | - | Категория (например, напитки, еда) |
| price | numeric(10,2) | - | Цена |
| createdat | timestamp without time zone | - | Дата добавления |
| description | text | - | Описание |
| imageurl | character varying(255) | 255 | Ссылка на изображение |
| weight | integer | 4 | Вес (в граммах) |
| ishidden | boolean | - | Признак скрытого товара |

Таблица «Бронирование» используется для отображения забронированных столов пользователем.

Таблица 3 – Структура таблицы «Bookings»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| id | serial | 4 | Идентификатор бронирования |
| userid | integer | 4 | Ссылка на пользователя |
| username | character varying(100) | 100 | Имя клиента |
| useremail | character varying(100) | 100 | Email клиента |
| tableid | integer | 4 | Номер столика |
| capacity | integer | 4 | Кол-во мест |
| bookingdatetime | timestamp without time zone | - | Время бронирования |
| createdat | timestamp without time zone | - | Дата создания брони |

Таблица «Элементы корзины» содержит товары, добавленные в корзину.

Таблица 4 – Структура таблицы «CartItems»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| id | integer | 4 | Идентификатор записи |
| user\_id | integer | 4 | Ссылка на пользователя |
| product\_id | integer | 4 | Ссылка на товар |
| quantity | integer | 4 | Количество товара |

Таблица «Заказы» содержит сведения об оформленных покупках.

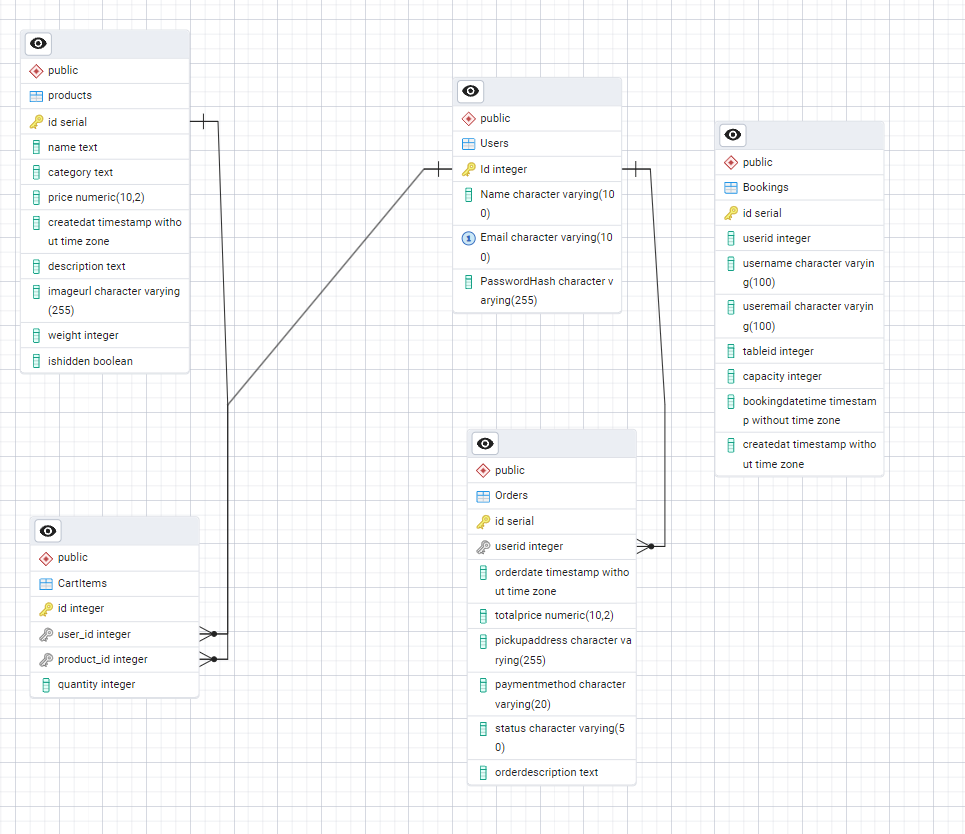
Таблица 5 – Структура таблицы «Orders»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| 1 | 2 | 3 | 4 |
| id | serial | 4 | Идентификатор заказа |
| userId | integer | 4 | Ссылка на пользователя |
| orderdate | timestamp without time zone | - | Дата оформления заказа |
| totalprice | numeric(10,2) | - | Общая сумма заказа |

Продолжение Таблицы 5

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| 1 | 2 | 3 | 4 |
| pickupaddress | character varying(255) | 255 | Адрес самовывоза |
| paymentmethod | character varying(20) | 20 | Способ оплаты |
| status | character varying(50) | 50 | Статус заказа (напр. ожидается, готов) |
| orderdescription | text | - | Комментарий к заказу |

Диаграмма базы данных представлена на рисунке 2



1. – Диаграмма базы данных

# Реализация веб-приложения

## Описание разделов веб-приложения

Главная страница веб-приложения «Paragraph» представляет собой удобный и интуитивно понятный интерфейс, созданный для комфортного взаимодействия пользователей с системой автоматизации заказов и оплаты услуг кофейни. В верхней части страницы расположено навигационное меню с основными разделами: «Главная», «Каталог», «О нас», «Бронь», «Профиль» и «Корзина».

Центральная часть главной страницы содержит яркие баннеры с партнёрами, отзывами и фотогалереей, а также каталог доступных напитков, блюд и десертов. Каждый товар представлен в виде карточки с изображением, названием, ценой и кнопками «Добавить в корзину». Пользователь может навестись на карточку с подробным описанием блюда или напитка, где отображаются состав, характеристики. Для быстрого поиска по меню реализована удобная строка с товаром.

Страница «Корзина» отображает выбранные пользователем позиции, их количество, цену за единицу и итоговую стоимость заказа. Пользователь может изменять количество товаров, удалять позиции и переходить к оформлению заказа. На этапе оформления необходимо указать контактные данные, выбрать способ оплаты (например, онлайн-оплата картой). После успешного оформления пользователю выводится подтверждение заказа с номером и деталями, а также отправляется чек на почту.

Раздел «Профиль» содержит личную информацию пользователя — имя, адрес электронной почты, телефон, а также историю заказов и список брони. Пользователь может редактировать свои данные и выходить из системы. Интерфейс профиля адаптирован для комфортного использования как на мобильных устройствах, так и на десктопах.

Раздел «О нас» содержит информацию о кофейне — название, номер телефона, адрес электронной почты и физический адрес. Также интегрирована карта, которая помогает быстро найти местоположение заведения и проложить маршрут.

Административная часть веб-приложения доступна только после авторизации администратора через защищённую форму входа. В разделе «Управление меню» администратор может добавлять новые позиции, редактировать описание, цену и количество товаров, а также загружать фотографии блюд и напитков.

Редактирование информации о товарах производится на отдельной странице, где администратор может менять название, состав, описание, цену и наличие каждого блюда или напитка, что обеспечивает удобное и оперативное управление ассортиментом кофейни.

## Разработка административной части приложения

Административная часть веб-приложения «Paragraph» представляет собой совокупность логических и физических компонентов, обеспечивающих централизованное управление ассортиментом, заказами, пользователями и отчетностью кофейни.

Логическая структура включает несколько ключевых модулей: модуль авторизации, модуль управления меню, модуль управления заказами, модуль управления пользователями и модуль формирования отчетов. Административная панель служит единой точкой доступа, предоставляя администратору полный контроль над системой.

Модуль авторизации обеспечивает доступ исключительно для администраторов, реализуя проверку прав и контроль безопасности. Модуль управления меню позволяет добавлять, редактировать, удалять и просматривать позиции меню — напитки, блюда, десерты, включая изображения, состав, цены и описание.

Модуль управления заказами предоставляет возможность просматривать все поступившие заказы, изменять их статус и управлять процессом выполнения. Модуль управления пользователями содержит функционал просмотра, редактирования данных зарегистрированных клиентов и мониторинга их активности.

Модуль формирования отчетов предназначен для анализа ключевых показателей работы кофейни: количества заказов, объема продаж, активности пользователей за выбранные периоды. Все модули взаимодействуют через общую базу данных, что обеспечивает целостность бизнес-логики и актуальность данных.

Логическая структура административной части представлена на приложение Б рисунок Б.1.

Физическая структура административной части веб-приложения «Paragraph» включает три основные компоненты: клиентскую, серверную и базу данных. Клиентская часть реализована в виде веб-интерфейса, доступного через браузер администратора, где происходит взаимодействие с системой.

Серверная часть отвечает за обработку запросов, реализацию бизнес-логики по управлению меню, заказами, пользователями и формированию отчетов. Взаимодействие между клиентской и серверной частями осуществляется через REST API по архитектуре клиент-сервер.

Базовая часть — это система управления базами данных, где хранится структурированная информация о меню, заказах, клиентах, статусах заказов и прочей бизнес-логике административной панели.

При необходимости архитектура может быть масштабирована за счёт разделения компонентов, внедрения облачных сервисов и выделения отдельных микросервисов для повышения производительности и отказоустойчивости.

Таким образом, физическая и логическая структуры административной части обеспечивают надежную, масштабируемую и удобную платформу для эффективного управления процессами кофейни «Paragraph».

Логическая структура административной части представлена на приложение Б рисунок Б.2.

## Разработка клиентской части приложения

Клиентская часть веб-приложения «Paragraph» предназначена для обеспечения удобного взаимодействия пользователей с системой кофейни. Логическая структура клиентской части включает несколько взаимосвязанных компонентов, каждый из которых выполняет определённые функции.

Логическая структура веб-приложения состоит из следующих ключевых компонентов. Регистрация и вход обеспечивают доступ пользователей к личному кабинету и сохранение индивидуальных данных. Редактирование профиля позволяет изменять личную информацию и просматривать историю заказов.

Фильтрация используются для удобной навигации по меню кофейни, позволяя быстро находить нужные позиции по категориям, составу или ключевым словам. Просмотр меню предоставляет пользователю доступ ко всем доступным напиткам, блюдам и десертам с возможностью перехода на подробную карточку товара.

Добавление в заказ позволяет сформировать список выбранных позиций, а оформление заказа — завершить покупку, выбрав способ оплаты и доставки (самовывоз или доставка на адрес). Также реализован просмотр заказов с детализацией по каждому из них.

Навигационные элементы (меню,кнопки и маршруты) обеспечивают удобное перемещение по разделам.

Логическая структура клиентской части представлена на приложение Б на рисунке Б.3

Физическая структура клиентской части веб-приложения «Paragraph» представляет собой описание размещения физических компонентов, организованных по поддиректориям в рамках основной директории проекта. Эти компоненты обеспечивают отображение интерфейса, маршрутизацию, обработку пользовательских действий и взаимодействие с серверной частью системы.

Структура включает разделы, отвечающие за навигацию между страницами, подключение к базе данных, отображение данных, а также взаимодействие с моделью приложения. Каждый раздел содержит компоненты, необходимые для реализации отдельных функций клиентской части, таких как просмотр меню, работа с заказом, оформление заказов и управление профилем пользователя.

Физическая структура клиентской части представлена на приложение Б на рисунке Б.4.

## Описание используемых функций и процедур

Для реализации графического интерфейса веб-приложения «Paragraph» использовались технологии C#, ASP.NET Core MVC, React и PostgreSQL. Такой стек обеспечивает современный внешний вид, кроссбраузерную совместимость и надёжное хранение данных. Архитектура построена по принципу клиент–серверного взаимодействия с использованием REST API, что позволяет эффективно масштабировать проект и разделять логику между фронтендом и бэкендом.

Функция в программировании — это отдельный блок кода, выполняющий конкретную задачу и вызываемый из разных частей программы. Использование функций помогает структурировать код, делает его более понятным, удобным для повторного использования и сопровождения. Функции могут принимать входные параметры и возвращать результат своей работы.

Кроссплатформенность и адаптивность обеспечиваются за счёт использования React: все компоненты интерфейса автоматически адаптируются под размер экрана устройства. Для стилизации применяются кастомные CSS-решения и адаптивная вёрстка, что делает сайт удобным для использования как на ПК, так и на мобильных устройствах.

Локальная база данных реализована с использованием PostgreSQL и ORM-библиотеки Entity Framework Core, что обеспечивает надёжную и эффективную работу с таблицами пользователей, товаров, заказов, корзины, изображений, категорий и отзывов.

Таким образом, веб-приложение «Paragraph» представляет собой полноценный интернет-магазин, предлагающий пользователю удобную навигацию, авторизацию, возможность оформления заказов, ведения истории покупок и управления профилем.

Код функции регистрации пользователя представлен ниже:

[HttpPost("register")]

*public* *async* Task<ActionResult<User>> Register([FromBody] UserRegistrationDto registration)

        {

            if (registration == null)

                return BadRequest("Данные для регистрации не были переданы.");

            if (string.IsNullOrWhiteSpace(registration.Name) || string.IsNullOrWhiteSpace(registration.Email) || string.IsNullOrWhiteSpace(registration.Password))

                return BadRequest("Все поля обязательны для заполнения.");

            if (await \_context.Users.AnyAsync(u => u.Email == registration.Email))

                return BadRequest("Пользователь с таким email уже существует.");

*var* emailRegex = new Regex(@"^[^@\s]+@[^@\s]+\.[^@\s]+$");

            if (!emailRegex.IsMatch(registration.Email))

                return BadRequest("Некорректный email.");

            if (registration.Password.Length < 8)

                return BadRequest("Пароль должен содержать минимум 8 символов.");

*var* user = new User(registration.Name, registration.Email, "");

            user.PasswordHash = \_passwordHasher.HashPassword(user, registration.Password);

            \_context.Users.Add(user);

            await \_context.SaveChangesAsync();

            return CreatedAtAction(nameof(Register), new { id = user.Id }, user);

        }

Код функции авторизации пользователя:

[HttpPost("login")]

*public* *async* Task<ActionResult<User>> Login([FromBody] UserLoginDto login)

        {

*var* user = await \_context.Users.FirstOrDefaultAsync(u => u.Email == login.Email);

            if (user == null)

                return Unauthorized("Неверный email или пароль.");

*var* result = \_passwordHasher.VerifyHashedPassword(user, user.PasswordHash, login.Password);

            if (result == PasswordVerificationResult.Failed)

                return Unauthorized("Неверный email или пароль.");

            return Ok(user);

        }

Код функции добавления товара в корзину:

[HttpPost]

*public* *async* Task<IActionResult> AddToCart([FromBody] AddToCartRequest request)

    {

        if (request.Quantity < 1)

            return BadRequest("Количество должно быть не меньше 1.");

*var* cartItem = await \_context.CartItems

            .FirstOrDefaultAsync(ci => ci.UserId == request.UserId && ci.ProductId == request.ProductId);

        if (cartItem == null)

        {

            cartItem = new CartItem

            {

                UserId = request.UserId,

                ProductId = request.ProductId,

                Quantity = request.Quantity

            };

            \_context.CartItems.Add(cartItem);

        }

        else

        {

            cartItem.Quantity += request.Quantity;

            \_context.CartItems.Update(cartItem);

        }

        await \_context.SaveChangesAsync();

        return Ok(cartItem);

    }

Код функции оформления заказа:

[HttpPost]

*public* *async* Task<IActionResult> CreateOrder([FromBody] OrderDto dto)

        {

            if (dto.PaymentMethod == "Онлайн")

            {

                if (string.IsNullOrWhiteSpace(dto.CardNumber) ||

                    string.IsNullOrWhiteSpace(dto.CardExpiry) ||

                    string.IsNullOrWhiteSpace(dto.CardCvv))

                {

                    return BadRequest("Для онлайн-оплаты необходимо указать данные карты.");

                }

            }

*var* order = new Order

            {

                UserId = dto.UserId,

                PickupAddress = dto.PickupAddress,

                PaymentMethod = dto.PaymentMethod,

                OrderDescription = dto.OrderDescription,

                TotalPrice = dto.TotalPrice,

                OrderDate = DateTime.UtcNow,

                Status = dto.Status

            };

            \_context.Orders.Add(order);

            await \_context.SaveChangesAsync();

*// Отправка чека на почту*

*var* user = await \_context.Users.FindAsync(dto.UserId);

            if (user != null && !string.IsNullOrEmpty(user.Email))

            {

*var* subject = "Ваш заказ в Кофейне";

*var* body = $@"

Здравствуйте, {user.Name}!

Ваш заказ успешно оформлен:

📦 Состав заказа:

{dto.OrderDescription}

📍 Адрес самовывоза:

{dto.PickupAddress}

💰 Общая сумма: {dto.TotalPrice} BYN

Спасибо за заказ!";

                try

                {

*var* smtpClient = new SmtpClient("smtp.gmail.com")

                    {

                        Port = 587,

                        Credentials = new NetworkCredential("alexalexey914@gmail.com", "bxwa cgei ymiu zvyu"),

                        EnableSsl = true,

                    };

*var* mail = new MailMessage("alexalexey914@gmail.com", user.Email, subject, body);

                    await smtpClient.SendMailAsync(mail);

                }

                catch (Exception ex)

                {

                    Console.WriteLine("Ошибка отправки email: " + ex.Message);

                }

            }

            return Ok(order);

        }

Код функции редактирования профиля пользователя:

[HttpPost("login")]

*public* *async* Task<ActionResult<User>> Login([FromBody] UserLoginDto login)

        {

*var* user = await \_context.Users.FirstOrDefaultAsync(u => u.Email == login.Email);

            if (user == null)

                return Unauthorized("Неверный email или пароль.");

*var* result = \_passwordHasher.VerifyHashedPassword(user, user.PasswordHash, login.Password);

            if (result == PasswordVerificationResult.Failed)

                return Unauthorized("Неверный email или пароль.");

            return Ok(user);

        }

        [HttpPut("update")]

*public* *async* Task<IActionResult> UpdateUser([FromBody] UpdateUserDto dto)

        {

*var* user = await \_context.Users.FindAsync(dto.Id);

            if (user == null)

                return NotFound("Пользователь не найден");

            user.Name = dto.Name;

            user.Email = dto.Email;

            await \_context.SaveChangesAsync();

            return Ok();

        }

        [HttpPut("change-password")]

*public* *async* Task<IActionResult> ChangePassword([FromBody] ChangePasswordDto dto)

        {

            if (dto == null)

                return BadRequest("Данные не были переданы");

*var* user = await \_context.Users.FindAsync(dto.UserId);

            if (user == null)

                return NotFound("Пользователь не найден");

            user.PasswordHash = \_passwordHasher.HashPassword(user, dto.NewPassword);

            await \_context.SaveChangesAsync();

            return Ok("Пароль успешно обновлён");

        }

## Функциональное тестирование

Функциональное тестирование веб-приложения кофейни «Paragraph» представляет собой процесс проверки программного обеспечения на соответствие заявленным требованиям, при котором оценивается корректность выполнения системой всех предусмотренных функций. Основная задача данного вида тестирования — убедиться, что приложение стабильно и надёжно выполняет все действия, предусмотренные для пользователей, как при корректном, так и при некорректном вводе данных.

Проверка будет осуществляться на основе заранее подготовленных тест-кейсов, каждый из которых описывает определённый сценарий использования, включает входные данные, ожидаемый результат и фактический результат выполнения. Тест-кейсы охватывают ключевые модули приложения: регистрацию и вход пользователей, просмотр меню кофейни, добавление позиций в заказ, оформление заказа с выбором способа оплаты и доставки, управление личным кабинетом. Особое внимание будет уделено проверке обработки некорректных данных, таких как пустые поля, неверный формат контактной информации, ошибки при авторизации и некорректное изменение заказа.

Результаты функционального тестирования будут документированы и дополнены скриншотами пользовательского интерфейса, подтверждающими корректность работы функций или наличие обнаруженных ошибок.

Таблица 11 – Тест - кейсы

|  |  |  |  |
| --- | --- | --- | --- |
| Модуль / Функция | Шаги выполнения | Ожидаемый результат | Фактический результат |
| 1 | 2 | 3 | 4 |
| Регистрация | 1. Открыть сайт 2. Нажать иконку «Личного кабинета» 3. Нажать кнопку «Нет аккаунта? Создайте его» 4. Заполнить данные | Переход на форму «Личный кабинет» | Успешный переход на страницу «Личный кабинет».  Результат представлен на приложении Д на рисунке Д.1 |

Продолжение таблицы

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Авторизация | 1. Нажать иконку «Личного кабинета» 2. Нажать кнопку «Нет аккаунта? Создайте его» 3. Заполнить данные | Переход на форму «Личный кабинет» | Успешный переход на страницу «Личный кабинет».  Результат представлен на приложении Д на рисунке Д.2 |
| Добавление товара в корзину | 1. Выбрать товар 2. Нажать кнопку «В корзину» | Добавление товара в корзину | Успешное добавление товара в корзину  Результат представлен на приложении Д на рисунке Д.3 |
| Оформление заказа | 1. Перейти в корзину 2. Нажать на кнопку оформление заказа 3. Заполнить данные 4. Нажать на кнопку «Подтвердить заказ» | Успешное оформление заказа. | Успешное оформление заказа  Результат представлен на приложении Д на рисунке Д.4 |
| Фильтрация по категории | 1. Открыть каталог товаров 2. . Выбрать категорию (например, «Кофе») 3. Проверить, что отображаются только соответствующие товары | Товары отфильтрованы по выбранной категории | Фильтрация работает корректно Приложение Д, рисунок Д.5 |
| Редактирование профиля | 1. Перейти в личный кабинет 2. Нажать «Редактировать» 3. Изменить данные (имя, email и т.д.) 4. Сохранить изменения | Данные обновлены | Успешное обновление данных Приложение Д, рисунок Д.6 |
| Просмотр заказов | 1. Перейти в личный кабинет 2. Открыть вкладку «Мои заказы» 3. Выбрать любой заказ | Отображается информация по заказу | История заказов отображается корректно Приложение Д, рисунок Д.7 |

# Применение

## Назначение веб-приложения

Данное веб-приложение разработано для автоматизации процессов в кофейне «Paragraph». Оно предоставляет удобные инструменты для управления меню, оформления заказов, бронирования столиков, ведения корзины, а также регистрации и обслуживания пользователей. Система упрощает работу персонала и улучшает взаимодействие с клиентами как в заведении, так и онлайн.

Для установки приложения необходимо следовать инструкциям разработчика. После размещения на сервере требуется выполнить настройку окружения: установить необходимые библиотеки, подключить базу данных PostgreSQL и выполнить начальную миграцию для формирования таблиц.

После завершения установки приложение запускается с помощью соответствующих команд, после чего проводится проверка его корректной работы и готовности к использованию.

Веб-приложение ориентировано на автоматизацию процессов обслуживания и повышение качества клиентского сервиса в кофейне «Paragraph». Оно позволяет посетителям просматривать актуальное меню, добавлять позиции в корзину, оформлять заказы, бронировать столики, а также управлять своим личным кабинетом.

Ограничения области применения включают:

* доступ к оформлению заказов и управлению данными доступен только авторизованным пользователям;
* административный функционал доступен исключительно для сотрудников с соответствующими правами доступа;
* система специализируется на продаже бытовой техники и ориентирована на магазины, работающие по модели электронной коммерции.

## Программно-аппаратное обеспечение сервера и клиента

Разрабатываемое веб-приложение для кофейни **«**Paragraph**»** функционирует на сервере под управлением операционной системы Windows10.

Минимальные системные требования для сервера:

* процессор с четырьмя и более физическими ядрами;
* оперативная память объёмом не менее 4 ГБ;
* свободное место на жёстком диске от 2 ГБ.

Для запуска и корректной работы программного обеспечения необходимо установить следующие компоненты:

* веб-сервер Apache HTTP Server — сервер с открытым исходным кодом, обрабатывающий HTTP-запросы от клиентов и предоставляющий доступ к статическому и динамическому контенту (HTML, CSS, JavaScript). Используется для хостинга клиентской части и API;
* система управления базами данных PostgreSQL — реляционная СУБД с открытым исходным кодом, используемая для хранения информации о товарах (кофе, блюда, десерты), пользователях, заказах, бронированиях столов и других элементах бизнес-логики кофейни. PostgreSQL обеспечивает надёжное хранение данных, поддержку транзакций и высокую производительность при работе с REST API;
* платформа разработки .NET SDK (ASP.NET Core) — набор средств разработки для реализации серверной логики приложения на языке C# с архитектурой MVC. Поддерживает построение REST API и безопасную работу с базой данных;
* среда выполнения JavaScript Node.js — используется для запуска JavaScript-кода на сервере, применяясь в задачах интеграции с внешними сервисами, сборки фронтенда и отправки уведомлений (по необходимости).

Клиентская часть веб-приложения создаётся с использованием следующих инструментов:

* javaScript-библиотека React — используется для создания интерактивного интерфейса пользователя, включая каталог товаров, корзину, оформление заказов, профиль и админ-панель;
* ведактор кода Visual Studio Code — основной инструмент для разработки клиентской части, поддерживающий HTML, CSS, JavaScript и React, а также интеграцию с системами контроля версий;
* веб-браузер (Google Chrome, Firefox, Microsoft Edge и др.) — необходим для запуска клиентской части, предоставляющей доступ к веб-интерфейсу приложения.

Развёртывание базы данных осуществляется с помощью pgAdmin — графического интерфейса для управления PostgreSQL. В pgAdmin создаётся база данных CafeDB, в которую затем добавляются все необходимые таблицы и связи согласно структуре проекта. Серверная часть запускается через Visual Studio, а клиентская — при помощи команды npm start в командной строке Node.js.

После запуска веб-приложение становится доступным локально: API функционирует на http://localhost:5189, клиентская часть — на http://localhost:3000. Клиент взаимодействует с сервером через HTTP-запросы, обрабатываемые ASP.NET Core и SQL Server.

# Заключение

В рамках выполнения проекта на тему **«**Разработка веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph» было создано полнофункциональное веб-приложение, предназначенное для оформления заказов, бронирования столиков и управления ассортиментом товаров. Приложение предоставляет пользователям возможность просматривать меню (кофе, блюда, десерты), фильтровать позиции по категориям, добавлять товары в корзину, оформлять заказы, а также регистрироваться и управлять личным кабинетом.

Для достижения целей проектирования были выполнены следующие задачи:

* разработка веб-приложения для удобного взаимодействия с пользователями и товарными позициями;
* проектирование пользовательских сценариев и ролей (покупатель, администратор);
* создание интерфейсов в React с адаптивной вёрсткой для поддержки мобильных и десктопных устройств;
* реализация серверной части с использованием ASP.NET Core и архитектуры REST API;
* подключение базы данных PostgreSQL и реализация схемы данных с помощью Entity Framework Core;
* описание структуры приложения через диаграммы: прецедентов, классов и развёртывания;
* разработка административной панели для управления товарами, заказами и пользователями;
* проведение тестирования основных функций: регистрация, поиск, оформление заказа, управление каталогом.

Приложение обладает удобным, современным и адаптивным интерфейсом. Пользователь может быстро зарегистрироваться, подобрать нужный товар с помощью фильтрации, добавить его в корзину и оформить заказ, выбрав подходящий способ доставки и оплаты. Благодаря использованию компонентов React интерфейс динамично реагирует на действия пользователя и сохраняет целостность на всех типах устройств.

Ключевыми преимуществами проекта являются модульная архитектура, масштабируемость, безопасность и простота обслуживания. В ходе разработки были закреплены практические навыки по построению клиент-серверных приложений, проектированию базы данных, реализации бизнес-логики и созданию адаптивного фронтенда.

Веб-приложение **«**Paragraph**»** готово к использованию в реальной среде и может применяться как основа для цифровой автоматизации процессов обслуживания в кофейнях и заведениях общественного питания.

# Список информационных источников

1. Багласова, Т.Г. Методические указания по выполнению курсового проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г. Багласова. – Минск : КБП, 2017. – 30 c.
2. Багласова, Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г. Багласова, К.О. Якимович. – Минск : КБП, 2013. – 29 c.
3. Бондарь, А.Г. Microsoft SQL Server 2012 / А.Г. Бондарь. – СПб. : БХВ-Петербург, 2013. – 608 с.
4. Тепляков, С. Паттерны проектирования на платформе .NET / С.Тепляков. – СПб. : Питер, 2015. – 320 с
5. Руководство по программированию на C# [Электронный ресурс]. – Microsoft, 2020. – Режим доступа : http://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/. – Дата доступа : 25.04.2020.
6. HTML и CSS. Полный справочник [Электронный ресурс] / J. Duckett. — Режим доступа: <https://developer.mozilla.org> — Дата доступа: 13.06.2025.
7. Постман. Документация и загрузка [Электронный ресурс]. – Режим доступа: <https://www.postman.com> – Дата доступа: 13.05.2025.
8. Axios: HTTP-библиотека для JavaScript [Электронный ресурс]. – Режим доступа: <https://axios-http.com> – Дата доступа: 13.05.2025.
9. React – библиотека JavaScript [Электронный ресурс]. – Режим доступа: <https://react.dev> – Дата доступа: 13.05.2025.
10. Entity Framework Core: официальная документация [Электронный ресурс]. – Microsoft, 2024. – Режим доступа: <https://learn.microsoft.com/ef/core> – Дата доступа: 13.05.2025.
11. Visual Studio Code – редактор исходного кода [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com> – Дата доступа: 13.05.2025.

Приложения А  
Ментальная карта

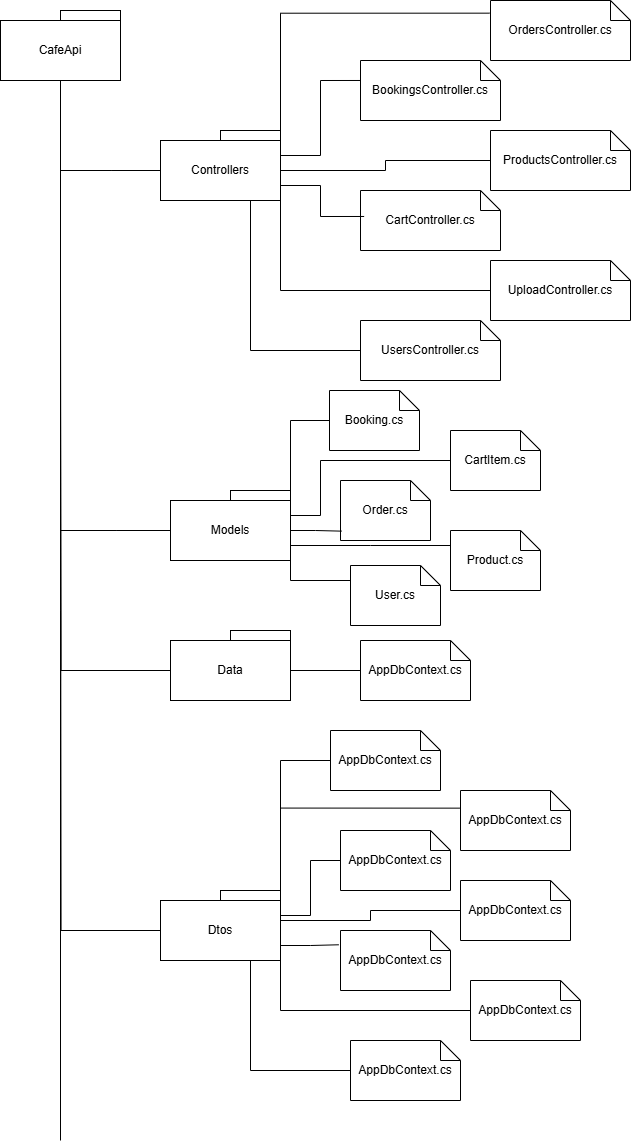


Рисунок А.1 – Ментальная карта

Приложение Б  
Логическая и физическая структура



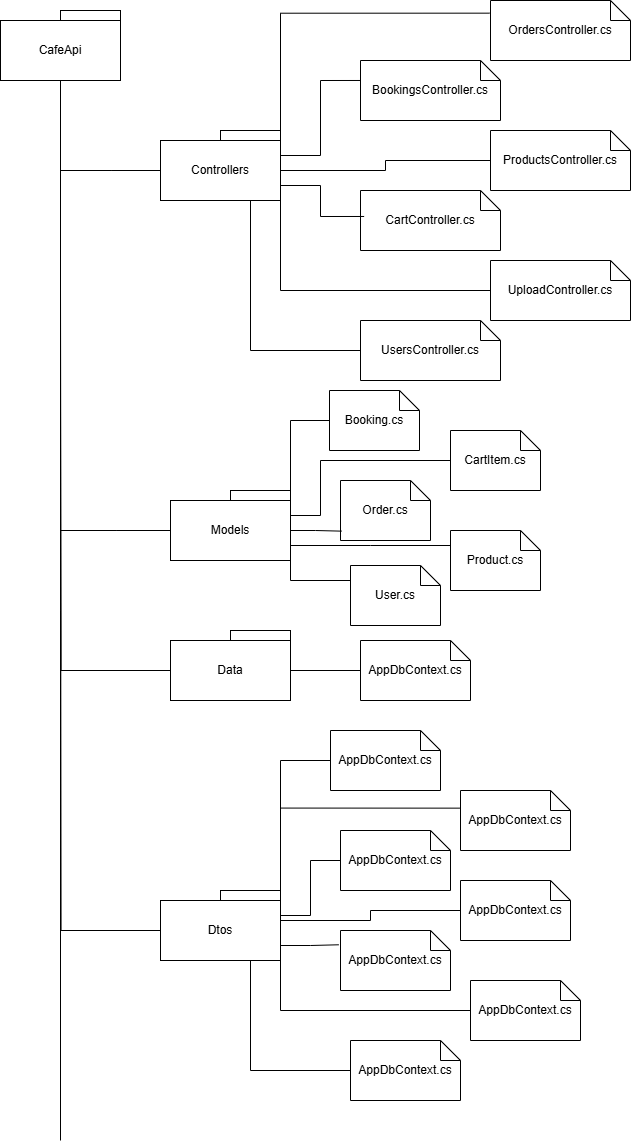
1. Логическая структура административной части веб -приложения



1. Физическая структура административной части веб – приложения

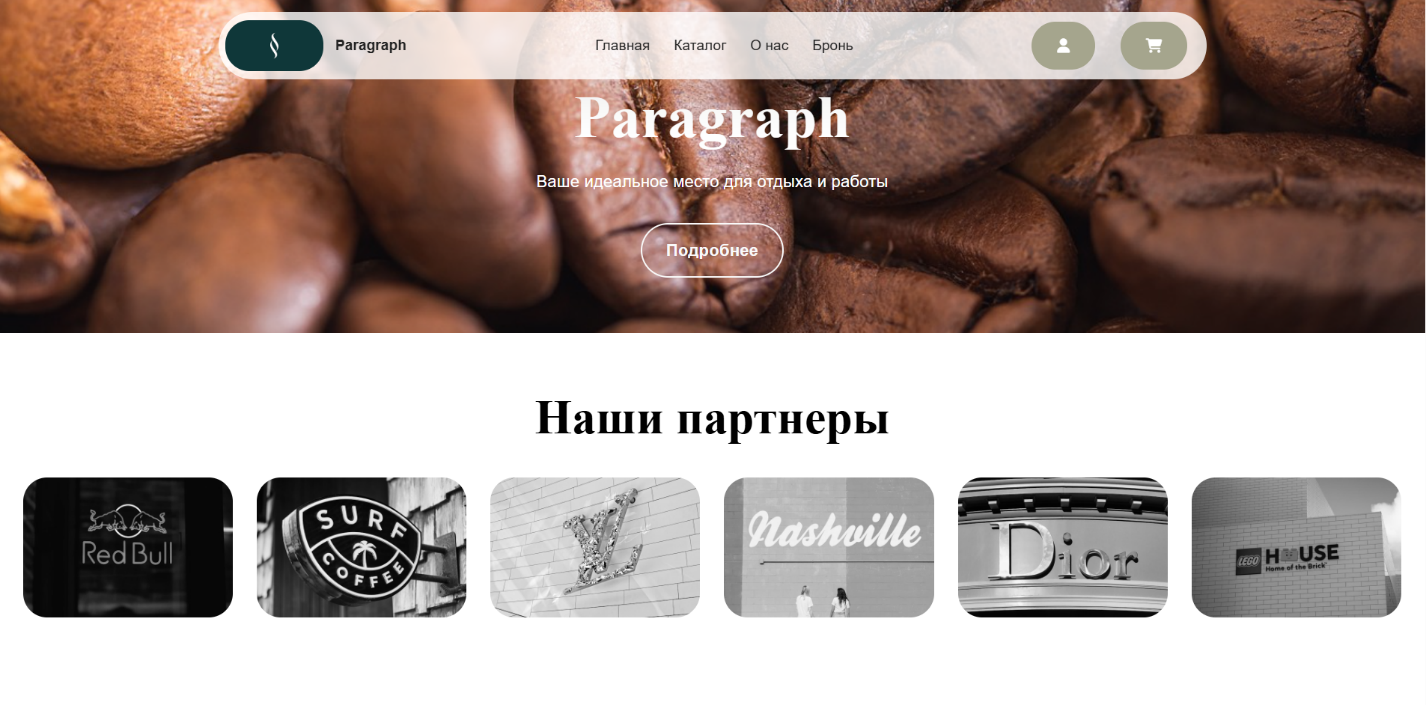


1. Логическая структура клиентской части веб – приложения

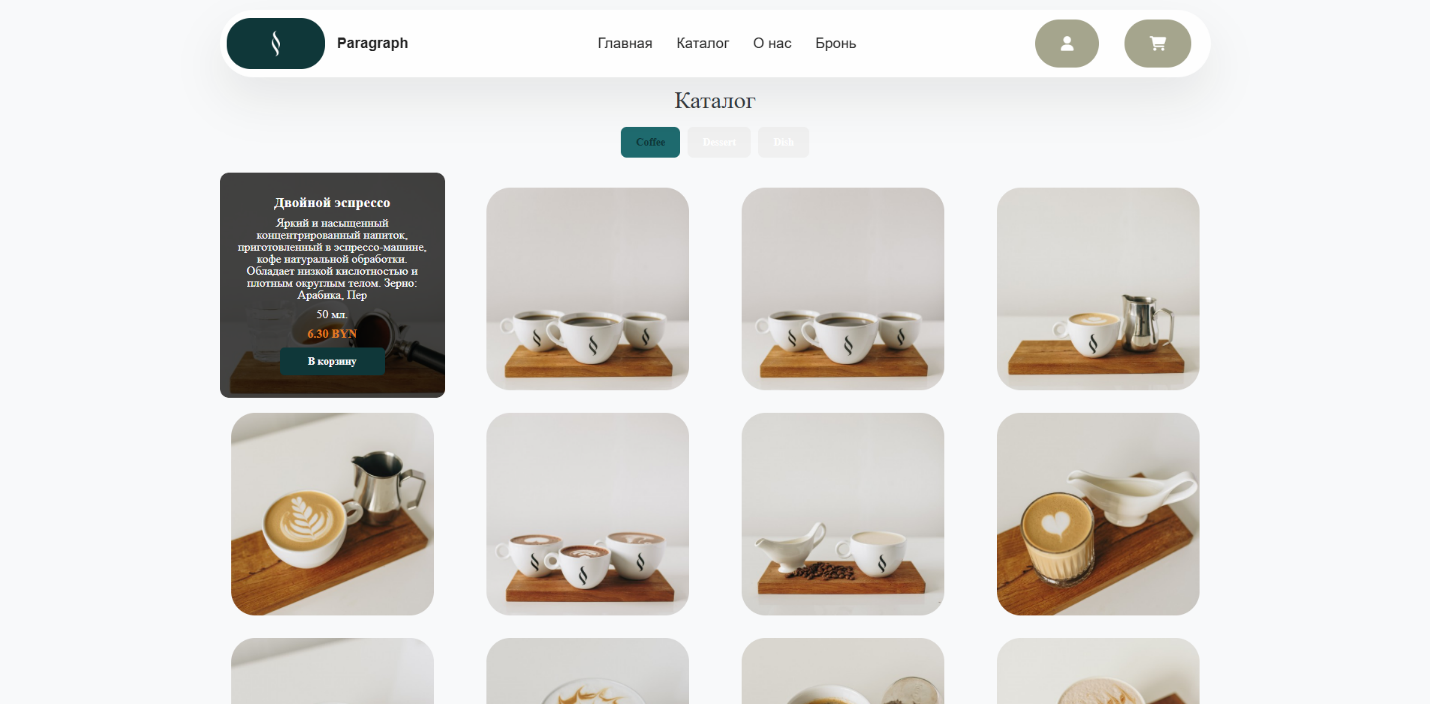


1. Физическая структура клиентской части веб – приложения

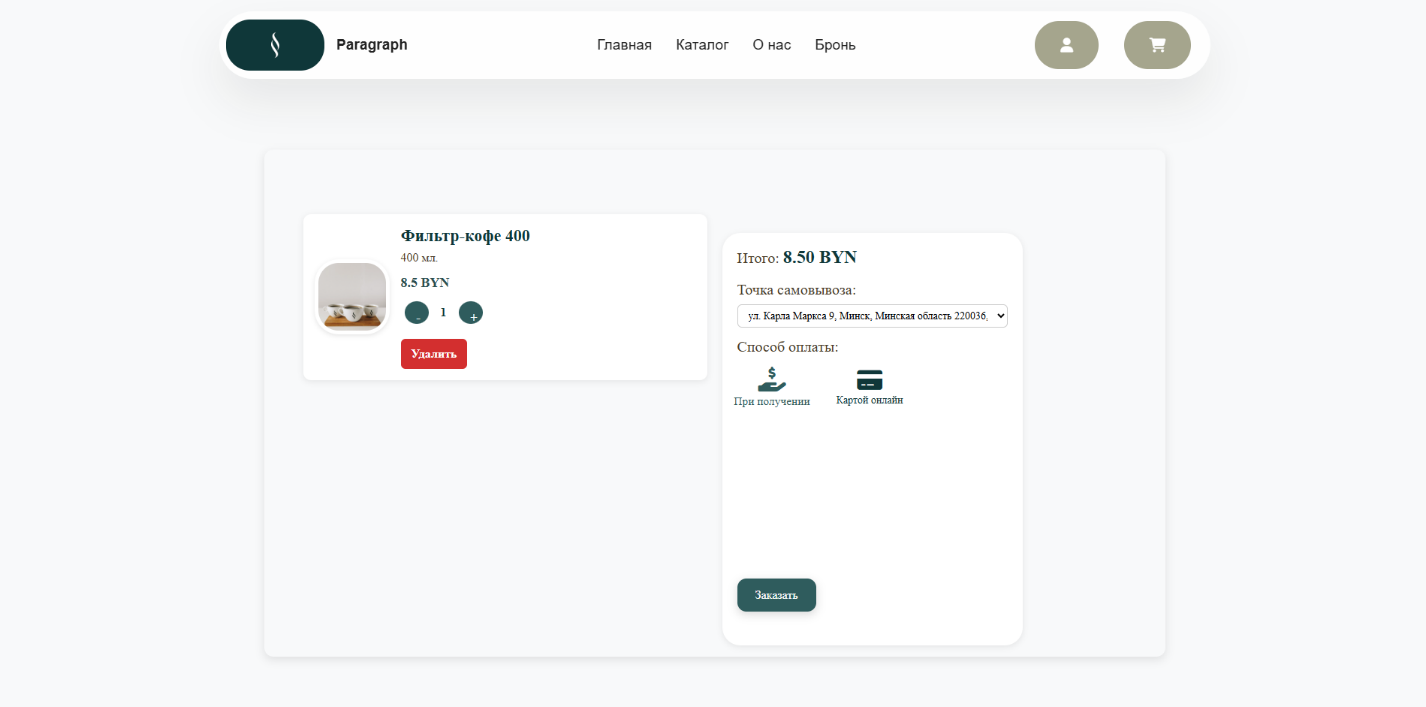
Приложение В  
Макеты страниц веб – приложения



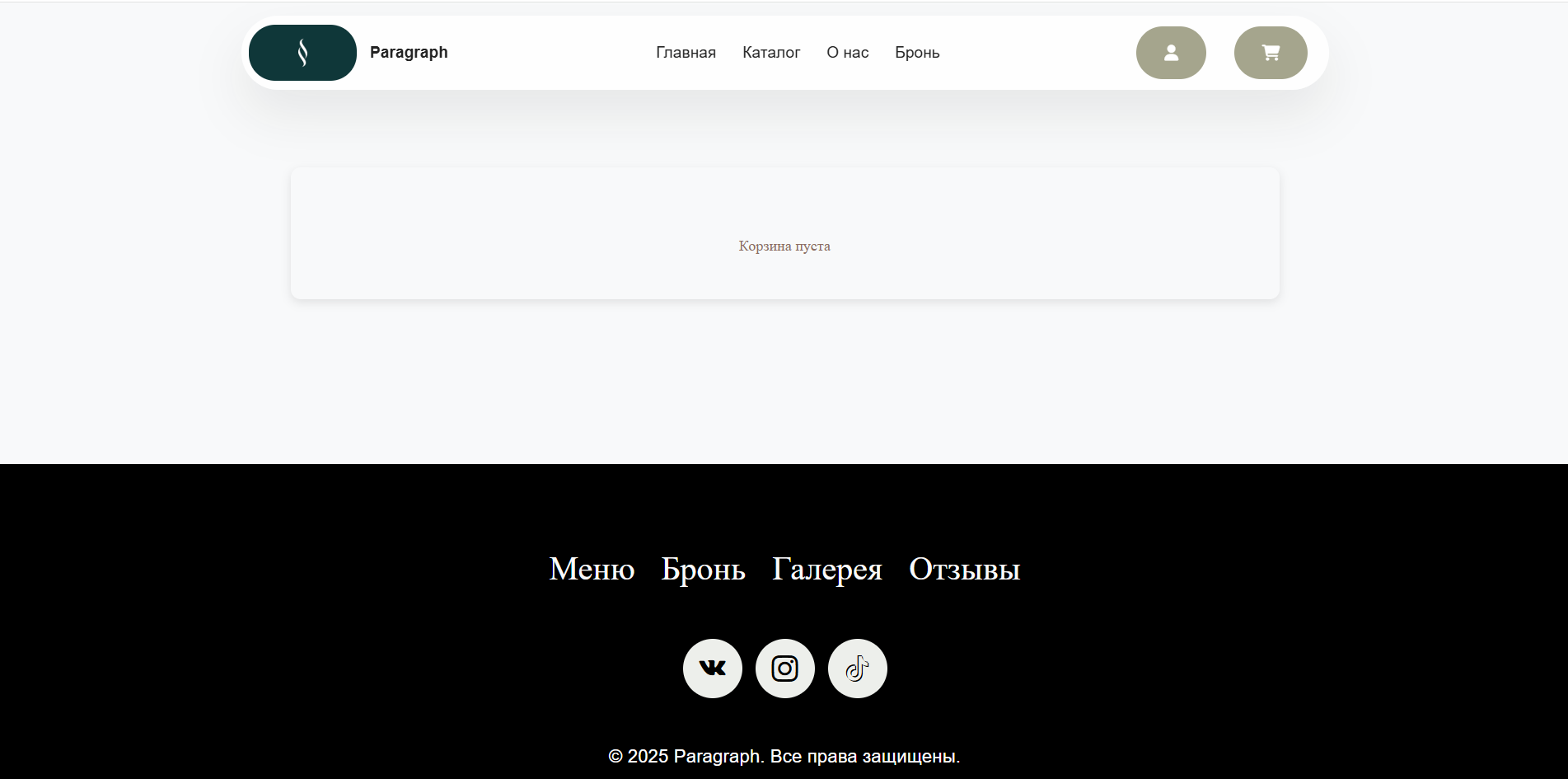
1. Макет страницы «Главная



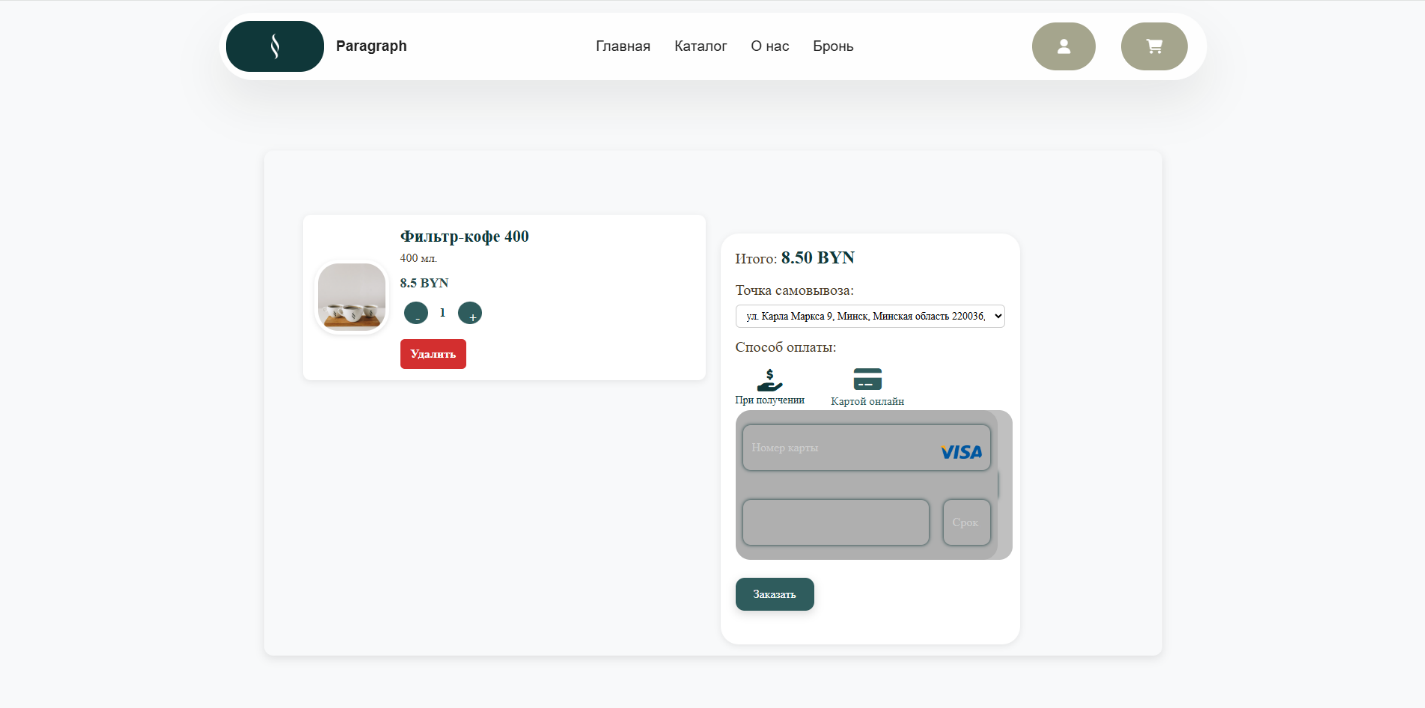
1. Макет страницы «Каталог товаров»



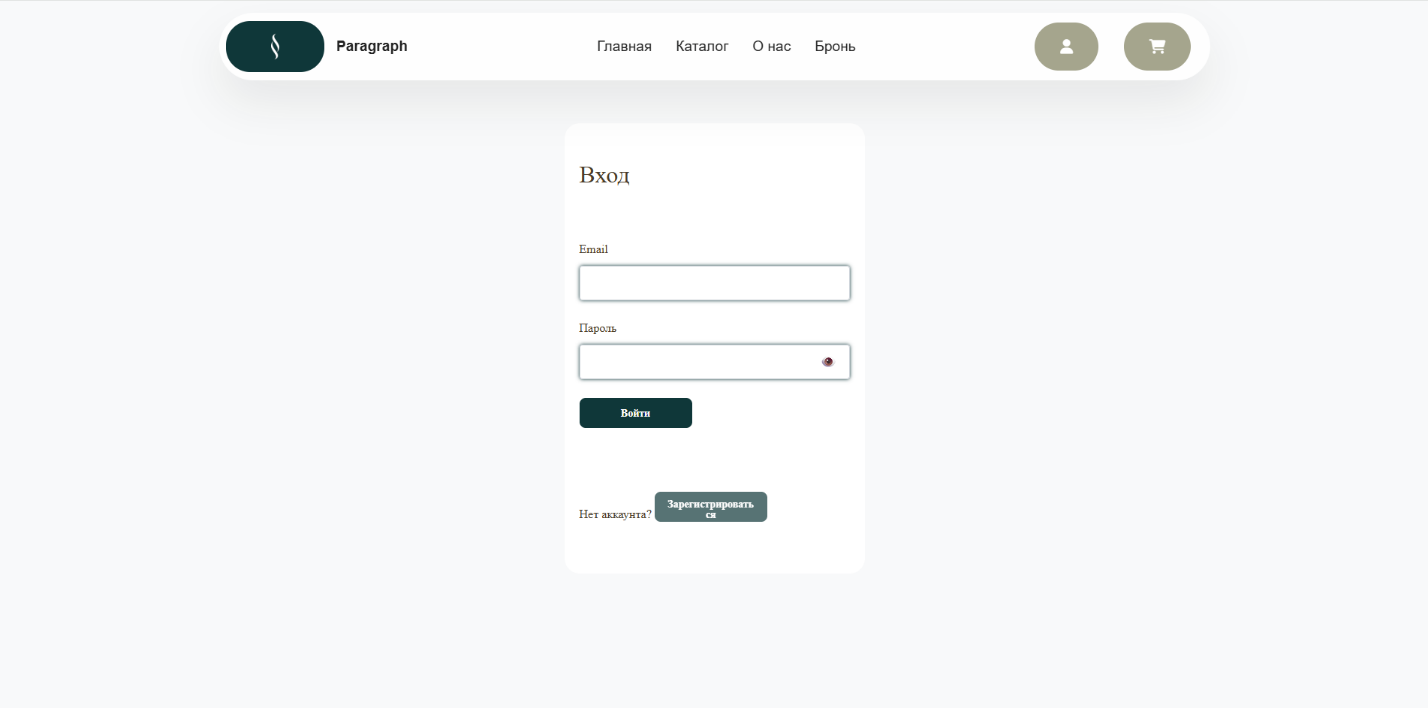
1. Макет страницы «Корзина(с товарами)»



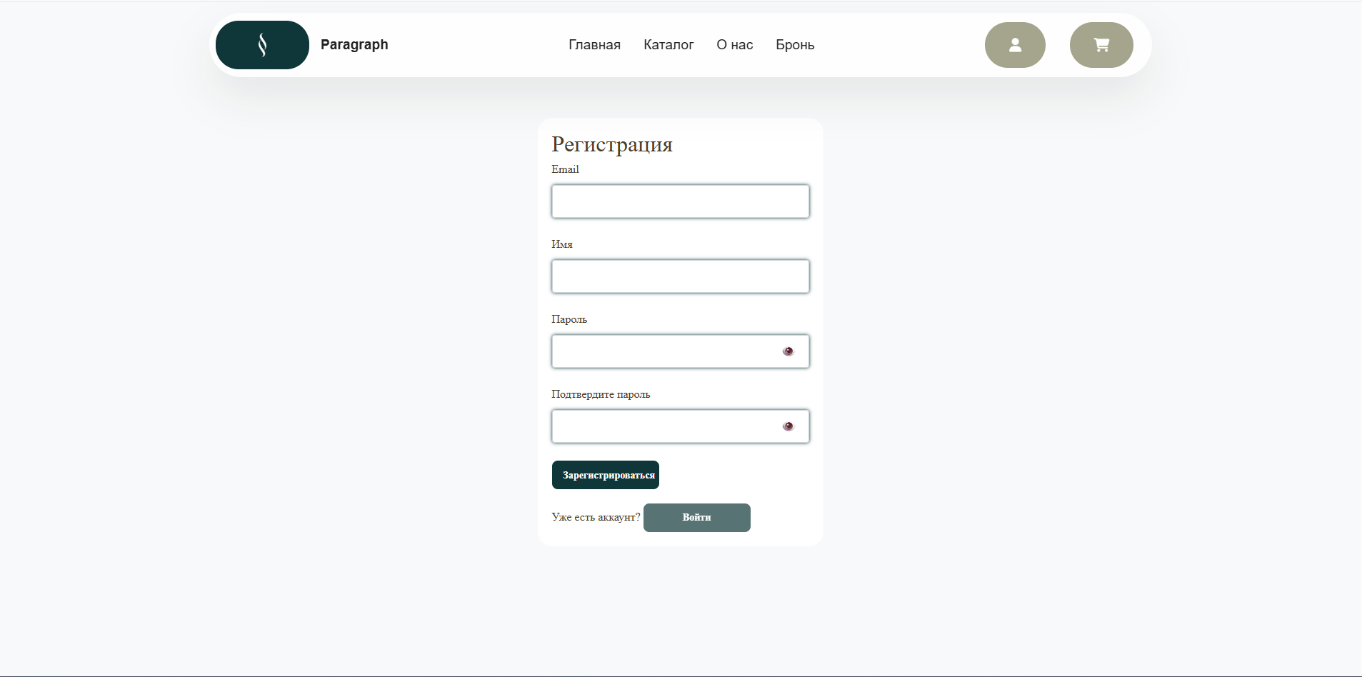
1. Макет страницы «Корзина (пустая)



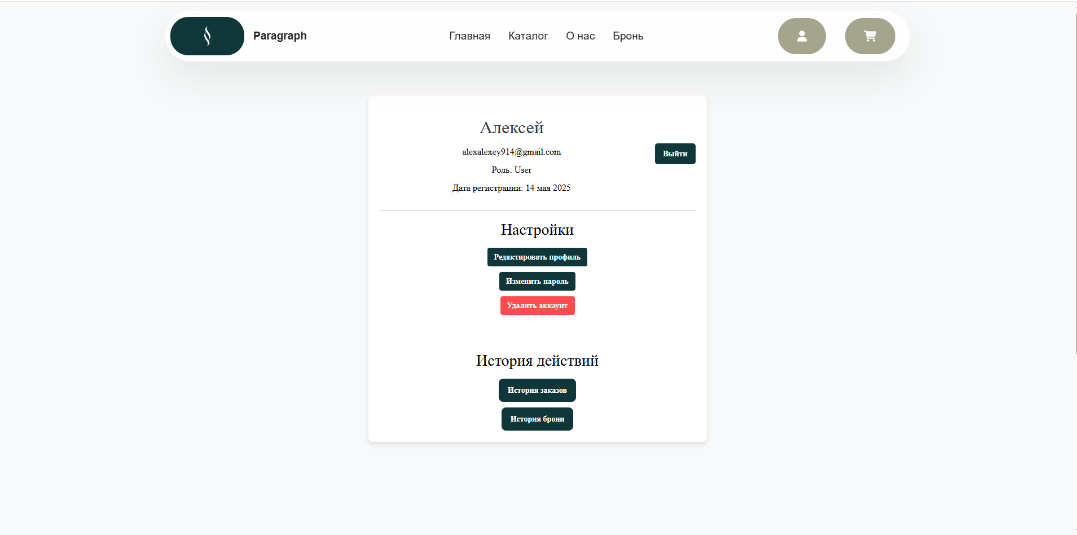
1. Макет страницы «Оформления заказа»



1. Макет формы «Авторизации»



1. Макет формы «Регистрации»



1. Макет страницы «Профиль»

Приложение Г  
Код модулей веб – приложения

AppDbContext.cs

using Microsoft.EntityFrameworkCore;

using CafeApi.Models;

*namespace* CafeApi.Data

{

*public* *class* AppDbContext : DbContext

    {

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

*public* DbSet<Product> Products { *get*; *set*; }

*public* DbSet<User> Users { *get*; *set*; }

*public* DbSet<CartItem> CartItems { *get*; *set*; }

*public* DbSet<Booking> Bookings { *get*; *set*; }

*protected* *override* void OnModelCreating(ModelBuilder modelBuilder)

        {

            modelBuilder.Entity<Order>()

                .HasOne(o => o.User)

                .WithMany()

                .HasForeignKey(o => o.UserId);

        }

*public* DbSet<Order> Orders { *get*; *set*; }

    }

}

BookingsController.cs

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using System.Linq;

using CafeApi.Data;

using CafeApi.Models;

using CafeApi.Dtos;

*namespace* CafeApi.Controllers

{

    [ApiController]

    [Route("api/bookings")]

*public* *class* BookingsController : ControllerBase

    {

*private* *readonly* AppDbContext \_context;

*public* BookingsController(AppDbContext context)

        {

            \_context = context;

        }

*// POST api/bookings/book*

        [HttpPost("book")]

*public* *async* Task<IActionResult> BookTable([FromBody] BookingRequest model)

        {

            if (!ModelState.IsValid)

                return BadRequest(ModelState);

*var* user = await \_context.Users.FindAsync(model.UserId);

            if (user == null)

                return NotFound("Пользователь не найден");

*var* bookingStart = model.BookingDateTime;

*var* bookingEnd = bookingStart.AddHours(1); *// предполагаемая длительность бронирования*

*var* bookingExists = await \_context.Bookings.AnyAsync(b =>

                b.TableId == model.TableId &&

                b.BookingDateTime < bookingEnd &&

                b.BookingDateTime.AddHours(1) > bookingStart

            );

            if (bookingExists)

                return Conflict("Столик уже забронирован на это время");

*var* booking = new Booking

            {

                UserId = model.UserId,

                TableId = model.TableId,

                Capacity = model.Capacity,

                BookingDateTime = bookingStart,

                Username = user.Name,

                UserEmail = user.Email *// берем email из базы для надежности*

            };

            \_context.Bookings.Add(booking);

            await \_context.SaveChangesAsync();

            return Ok(new { message = "Бронирование успешно создано", bookingId = booking.Id });

        }

*// GET api/bookings*

*// Вывести все бронирования (если нужно)*

        [HttpGet]

*public* *async* Task<IActionResult> GetAllBookings()

        {

*var* bookings = await \_context.Bookings.ToListAsync();

            return Ok(bookings);

        }

        [HttpGet("user")]

*public* *async* Task<IActionResult> GetUserBookings([FromQuery] int userId)

        {

            if (userId <= 0)

                return BadRequest("UserId не передан или некорректен");

*var* bookings = await \_context.Bookings

                .Where(b => b.UserId == userId)

                .ToListAsync();

            return Ok(bookings);

        }

        [HttpDelete]

*public* *async* Task<IActionResult> DeleteAllBookings()

        {

*var* bookings = await \_context.Bookings.ToListAsync();

            \_context.Bookings.RemoveRange(bookings);

            await \_context.SaveChangesAsync();

            return NoContent(); *// или Ok()*

        }

    }

}

CartController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Threading.Tasks;

using CafeApi.Data;

using CafeApi.Models;

using CafeApi.Dtos;

[ApiController]

[Route("api/[controller]")]

*public* *class* CartController : ControllerBase

{

*private* *readonly* AppDbContext \_context;

*public* CartController(AppDbContext context)

    {

        \_context = context;

    }

*public* *class* AddToCartRequest

    {

*public* int UserId { *get*; *set*; }

*public* int ProductId { *get*; *set*; }

*public* int Quantity { *get*; *set*; }

    }

    [HttpPost]

*public* *async* Task<IActionResult> AddToCart([FromBody] AddToCartRequest request)

    {

        if (request.Quantity < 1)

            return BadRequest("Количество должно быть не меньше 1.");

*var* cartItem = await \_context.CartItems

            .FirstOrDefaultAsync(ci => ci.UserId == request.UserId && ci.ProductId == request.ProductId);

        if (cartItem == null)

        {

            cartItem = new CartItem

            {

                UserId = request.UserId,

                ProductId = request.ProductId,

                Quantity = request.Quantity

            };

            \_context.CartItems.Add(cartItem);

        }

        else

        {

            cartItem.Quantity += request.Quantity;

            \_context.CartItems.Update(cartItem);

        }

        await \_context.SaveChangesAsync();

        return Ok(cartItem);

    }

    [HttpDelete("clear")]

*public* *async* Task<IActionResult> ClearCart([FromQuery] int userId)

    {

*var* cartItems = await \_context.CartItems

            .Where(ci => ci.UserId == userId)

            .ToListAsync();

        if (cartItems.Count == 0)

            return NotFound("Корзина пользователя уже пуста.");

        \_context.CartItems.RemoveRange(cartItems);

        await \_context.SaveChangesAsync();

        return NoContent();

    }

    [HttpGet]

*public* *async* Task<IActionResult> GetCartItems([FromQuery] int userId)

    {

*var* cartItems = await \_context.CartItems

            .Include(ci => ci.Product)

            .Where(ci => ci.UserId == userId)

            .ToListAsync();

*var* result = cartItems.Select(ci => new

        {

            id = ci.Id,

            productId = ci.ProductId,

            name = ci.Product?.Name ?? string.Empty,

            weight = ci.Product?.Weight ?? 0,

            price = ci.Product?.Price ?? 0m,

            img = ci.Product?.ImageUrl ?? string.Empty,

            quantity = ci.Quantity,

            category = ci.Product?.Category ?? string.Empty,

        });

        return Ok(result);

    }

    [HttpPut("{id}")]

*public* *async* Task<IActionResult> UpdateQuantity(int id, [FromBody] CartItemUpdateDto updateDto)

    {

*var* cartItem = await \_context.CartItems.FindAsync(id);

        if (cartItem == null)

            return NotFound();

        cartItem.Quantity = updateDto.Quantity;

        await \_context.SaveChangesAsync();

        return NoContent();

    }

    [HttpDelete("{id}")]

*public* *async* Task<IActionResult> DeleteCartItem(int id)

    {

*var* item = await \_context.CartItems.FindAsync(id);

        if (item == null)

            return NotFound();

        \_context.CartItems.Remove(item);

        await \_context.SaveChangesAsync();

        return NoContent();

    }

}

OrdersController.cs

using Microsoft.AspNetCore.Mvc;

using CafeApi.Data;

using CafeApi.Models;

using CafeApi.Dtos;

using Microsoft.EntityFrameworkCore;

using System.Net;

using System.Net.Mail;

*namespace* CafeApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

*public* *class* OrdersController : ControllerBase

    {

*private* *readonly* AppDbContext \_context;

*public* OrdersController(AppDbContext context)

        {

            \_context = context;

        }

        [HttpPost]

*public* *async* Task<IActionResult> CreateOrder([FromBody] OrderDto dto)

        {

            if (dto.PaymentMethod == "Онлайн")

            {

                if (string.IsNullOrWhiteSpace(dto.CardNumber) ||

                    string.IsNullOrWhiteSpace(dto.CardExpiry) ||

                    string.IsNullOrWhiteSpace(dto.CardCvv))

                {

                    return BadRequest("Для онлайн-оплаты необходимо указать данные карты.");

                }

            }

*var* order = new Order

            {

                UserId = dto.UserId,

                PickupAddress = dto.PickupAddress,

                PaymentMethod = dto.PaymentMethod,

                OrderDescription = dto.OrderDescription,

                TotalPrice = dto.TotalPrice,

                OrderDate = DateTime.UtcNow,

                Status = dto.Status

            };

            \_context.Orders.Add(order);

            await \_context.SaveChangesAsync();

*// Отправка чека на почту*

*var* user = await \_context.Users.FindAsync(dto.UserId);

            if (user != null && !string.IsNullOrEmpty(user.Email))

            {

*var* subject = "Ваш заказ в Кофейне";

*var* body = $@"

Здравствуйте, {user.Name}!

Ваш заказ успешно оформлен:

📦 Состав заказа:

{dto.OrderDescription}

📍 Адрес самовывоза:

{dto.PickupAddress}

💰 Общая сумма: {dto.TotalPrice} BYN

Спасибо за заказ!";

                try

                {

*var* smtpClient = new SmtpClient("smtp.gmail.com")

                    {

                        Port = 587,

                        Credentials = new NetworkCredential("alexalexey914@gmail.com", "bxwa cgei ymiu zvyu"),

                        EnableSsl = true,

                    };

*var* mail = new MailMessage("alexalexey914@gmail.com", user.Email, subject, body);

                    await smtpClient.SendMailAsync(mail);

                }

                catch (Exception ex)

                {

                    Console.WriteLine("Ошибка отправки email: " + ex.Message);

                }

            }

            return Ok(order);

        }

        [HttpGet("{userId}")]

*public* *async* Task<IActionResult> GetOrdersForUser(int userId)

        {

*var* orders = await \_context.Orders

                .Where(o => o.UserId == userId)

                .OrderByDescending(o => o.OrderDate)

                .ToListAsync();

            if (orders == null || !orders.Any())

                return NotFound("Заказы не найдены.");

            return Ok(orders);

        }

        [HttpGet]

*public* *async* Task<IActionResult> GetAllOrders()

        {

*var* ordersFromDb = await \_context.Orders

                .Include(o => o.User)

                .OrderByDescending(o => o.OrderDate)

                .ToListAsync();

*var* orders = ordersFromDb.Select(o => new OrderWithUserDto

            {

                Id = o.Id,

                OrderDate = o.OrderDate,

                PickupAddress = o.PickupAddress,

                PaymentMethod = o.PaymentMethod,

                OrderDescription = o.OrderDescription,

                TotalPrice = o.TotalPrice,

                Status = o.Status,

                UserName = o.User?.Name ?? string.Empty,

                UserEmail = o.User?.Email ?? string.Empty

            }).ToList();

            return Ok(orders);

        }

        [HttpDelete]

*public* *async* Task<IActionResult> DeleteAllOrders()

        {

*var* orders = await \_context.Orders.ToListAsync();

            if (orders.Count == 0)

                return NoContent();

            \_context.Orders.RemoveRange(orders);

            await \_context.SaveChangesAsync();

            return NoContent();

        }

        [HttpPut("{id}/status")]

*public* *async* Task<IActionResult> UpdateOrderStatus(int id, [FromBody] OrderStatusUpdateDto dto)

        {

*var* order = await \_context.Orders.FindAsync(id);

            if (order == null) return NotFound();

            order.Status = dto.Status;

            await \_context.SaveChangesAsync();

            return NoContent();

        }

*public* *class* OrderStatusUpdateDto

        {

*public* string? Status { *get*; *set*; }

        }

    }

}

ProductsController.cs

using Microsoft.AspNetCore.Mvc;

using CafeApi.Data;

using CafeApi.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Http;

using System.IO;

using System.Threading.Tasks;

using System.Collections.Generic;

using System.Globalization;

*namespace* CafeApi.Controllers

{

    [Route("api/products")]

    [ApiController]

*public* *class* ProductsController : ControllerBase

    {

*private* *readonly* AppDbContext \_context;

*public* ProductsController(AppDbContext context)

        {

            \_context = context;

        }

        [HttpGet]

*public* *async* Task<ActionResult<IEnumerable<Product>>> GetProducts()

        {

            return await \_context.Products.ToListAsync();

        }

        [HttpPost]

*public* *async* Task<ActionResult<Product>> AddProduct([FromForm] Product product, IFormFile file)

        {

            if (product.Weight <= 0)

            {

                return BadRequest("Вес товара должен быть положительным.");

            }

            if (file != null && file.Length > 0)

            {

*var* fileName = Path.GetFileName(file.FileName);

*var* filePath = Path.Combine("wwwroot", "images", fileName);

                using (*var* stream = new FileStream(filePath, FileMode.Create))

                {

                    await file.CopyToAsync(stream);

                }

                product.ImageUrl = $"/images/{fileName}";

            }

            product.CreatedAt = DateTime.UtcNow;

            \_context.Products.Add(product);

            await \_context.SaveChangesAsync();

            return CreatedAtAction(nameof(GetProducts), new { id = product.Id }, product);

        }

        [HttpPut("toggleHidden/{id}")]

*public* *async* Task<IActionResult> ToggleHidden(int id)

        {

*var* product = await \_context.Products.FindAsync(id);

            if (product == null)

            {

                return NotFound();

            }

            product.IsHidden = !product.IsHidden;

            await \_context.SaveChangesAsync();

            return Ok(product);

        }

        [HttpPut("{id}")]

*public* *async* Task<IActionResult> UpdateProduct(int id, [FromForm] Product product, IFormFile? file)

        {

            Console.WriteLine($"Получен запрос на обновление товара с ID: {id}");

            if (id != product.Id)

            {

                return BadRequest("ID товара не соответствует");

            }

            Console.WriteLine($"Обновляемый товар: {product.Name}");

            Console.WriteLine($"Цена товара: {product.Price}");

            Console.WriteLine($"Вес товара: {product.Weight}");

            Console.WriteLine($"Категория товара: {product.Category}");

            Console.WriteLine($"Описание товара: {product.Description}");

            Console.WriteLine($"Изображение: {product.ImageUrl ?? "Нет изображения"}");

            if (product.Weight <= 0)

            {

                Console.WriteLine("Ошибка: Вес товара должен быть положительным.");

                return BadRequest("Вес товара должен быть положительным.");

            }

            if (product.Price <= 0)

            {

                Console.WriteLine("Ошибка: Цена товара должна быть положительной.");

                return BadRequest("Цена товара должна быть положительной.");

            }

            if (file != null && file.Length > 0)

            {

                Console.WriteLine($"Загружаем файл: {file.FileName}");

*var* fileName = Path.GetFileName(file.FileName);

*var* filePath = Path.Combine("wwwroot", "images", fileName);

                using (*var* stream = new FileStream(filePath, FileMode.Create))

                {

                    await file.CopyToAsync(stream);

                }

                product.ImageUrl = $"/images/{fileName}";

            }

            else

            {

                Console.WriteLine("Изображение не было загружено.");

            }

            \_context.Entry(product).State = EntityState.Modified;

            try

            {

                await \_context.SaveChangesAsync();

                Console.WriteLine("Данные успешно обновлены в базе данных.");

            }

            catch (DbUpdateConcurrencyException)

            {

                if (!ProductExists(id))

                {

                    Console.WriteLine("Товар не найден в базе данных.");

                    return NotFound();

                }

                else

                {

                    throw;

                }

            }

            return NoContent();

        }

        [HttpDelete("{id}")]

*public* *async* Task<IActionResult> DeleteProduct(int id)

        {

*var* product = await \_context.Products.FindAsync(id);

            if (product == null)

            {

                return NotFound();

            }

            \_context.Products.Remove(product);

            await \_context.SaveChangesAsync();

            return NoContent();

        }

*private* bool ProductExists(int id)

        {

            return \_context.Products.Any(e => e.Id == id);

        }

    }

}

UploadController.cs

using Microsoft.AspNetCore.Mvc;

using System.IO;

using System.Threading.Tasks;

*namespace* CafeApi.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

*public* *class* UploadController : ControllerBase

    {

*private* *readonly* string \_uploadFolder;

*public* UploadController()

        {

            \_uploadFolder = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images");

            Directory.CreateDirectory(\_uploadFolder);

        }

        [HttpPost("upload")]

*public* *async* Task<IActionResult> Upload([FromForm] IFormFile file)

        {

            if (file == null || file.Length == 0)

            {

                return BadRequest("No file uploaded.");

            }

*var* fileName = Path.GetFileName(file.FileName);

*var* filePath = Path.Combine(\_uploadFolder, fileName);

            using (*var* stream = new FileStream(filePath, FileMode.Create))

            {

                await file.CopyToAsync(stream);

            }

            return Ok(new { filePath = $"images/{fileName}" });

        }

    }

}

UsersController.cs

using Microsoft.AspNetCore.Mvc;

using CafeApi.Data;

using CafeApi.Models;

using CafeApi.Dtos;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Identity;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

*namespace* CafeApi.Controllers

{

    [Route("api/users")]

    [ApiController]

*public* *class* UsersController : ControllerBase

    {

*private* *readonly* AppDbContext \_context;

*private* *readonly* IPasswordHasher<User> \_passwordHasher;

*public* UsersController(AppDbContext context, IPasswordHasher<User> passwordHasher)

        {

            \_context = context;

            \_passwordHasher = passwordHasher;

        }

        [HttpPost("register")]

*public* *async* Task<ActionResult<User>> Register([FromBody] UserRegistrationDto registration)

        {

            if (registration == null)

                return BadRequest("Данные для регистрации не были переданы.");

            if (string.IsNullOrWhiteSpace(registration.Name) || string.IsNullOrWhiteSpace(registration.Email) || string.IsNullOrWhiteSpace(registration.Password))

                return BadRequest("Все поля обязательны для заполнения.");

            if (await \_context.Users.AnyAsync(u => u.Email == registration.Email))

                return BadRequest("Пользователь с таким email уже существует.");

*var* emailRegex = new Regex(@"^[^@\s]+@[^@\s]+\.[^@\s]+$");

            if (!emailRegex.IsMatch(registration.Email))

                return BadRequest("Некорректный email.");

            if (registration.Password.Length < 8)

                return BadRequest("Пароль должен содержать минимум 8 символов.");

*var* user = new User(registration.Name, registration.Email, "");

            user.PasswordHash = \_passwordHasher.HashPassword(user, registration.Password);

            \_context.Users.Add(user);

            await \_context.SaveChangesAsync();

            return CreatedAtAction(nameof(Register), new { id = user.Id }, user);

        }

        [HttpPost("login")]

*public* *async* Task<ActionResult<User>> Login([FromBody] UserLoginDto login)

        {

*var* user = await \_context.Users.FirstOrDefaultAsync(u => u.Email == login.Email);

            if (user == null)

                return Unauthorized("Неверный email или пароль.");

*var* result = \_passwordHasher.VerifyHashedPassword(user, user.PasswordHash, login.Password);

            if (result == PasswordVerificationResult.Failed)

                return Unauthorized("Неверный email или пароль.");

            return Ok(user);

        }

        [HttpPut("update")]

*public* *async* Task<IActionResult> UpdateUser([FromBody] UpdateUserDto dto)

        {

*var* user = await \_context.Users.FindAsync(dto.Id);

            if (user == null)

                return NotFound("Пользователь не найден");

            user.Name = dto.Name;

            user.Email = dto.Email;

            await \_context.SaveChangesAsync();

            return Ok();

        }

        [HttpPut("change-password")]

*public* *async* Task<IActionResult> ChangePassword([FromBody] ChangePasswordDto dto)

        {

            if (dto == null)

                return BadRequest("Данные не были переданы");

*var* user = await \_context.Users.FindAsync(dto.UserId);

            if (user == null)

                return NotFound("Пользователь не найден");

            user.PasswordHash = \_passwordHasher.HashPassword(user, dto.NewPassword);

            await \_context.SaveChangesAsync();

            return Ok("Пароль успешно обновлён");

        }

        [HttpDelete("{id}")]

*public* *async* Task<IActionResult> DeleteUser(int id)

        {

*var* user = await \_context.Users.FindAsync(id);

            if (user == null)

                return NotFound();

            \_context.Users.Remove(user);

            await \_context.SaveChangesAsync();

            return Ok();

        }

        [HttpGet("{id}")]

*public* *async* Task<ActionResult<User>> GetUser(int id)

        {

*var* user = await \_context.Users.FindAsync(id);

            if (user == null)

                return NotFound();

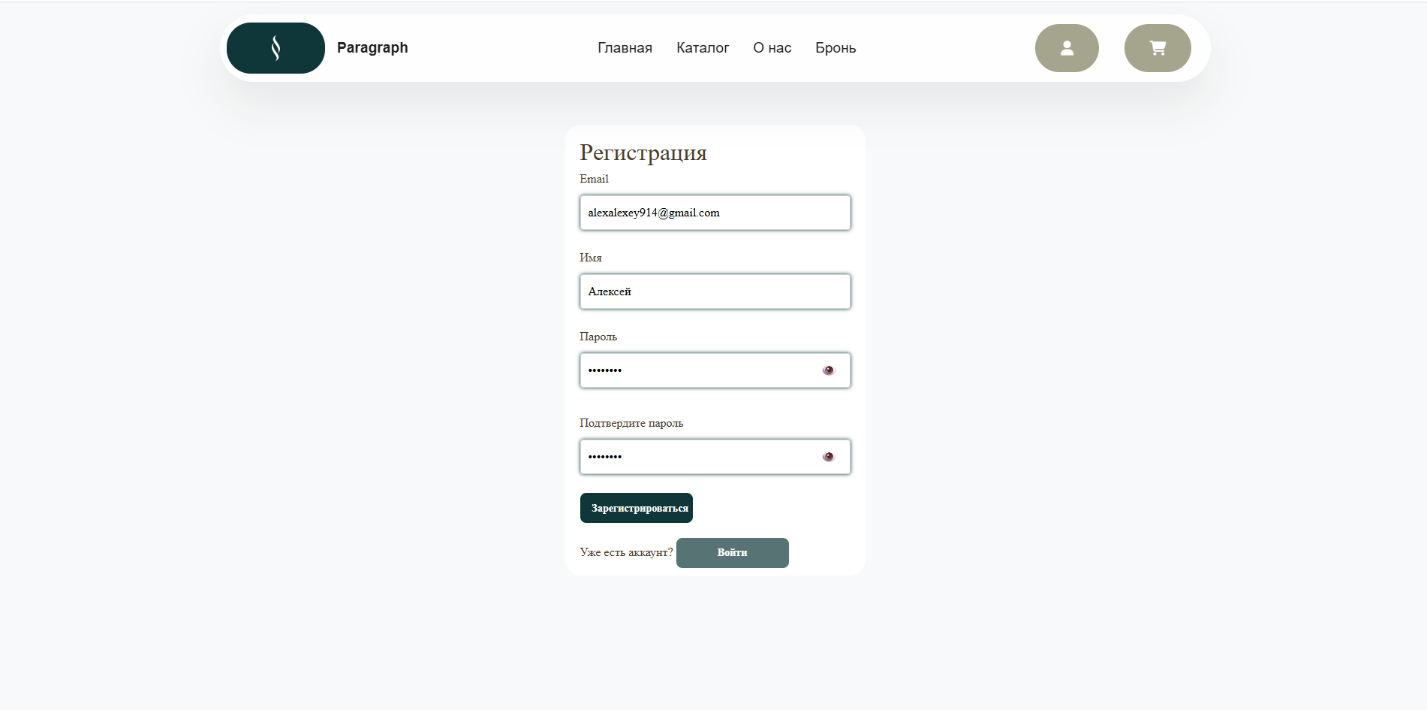
            return Ok(user);

        }

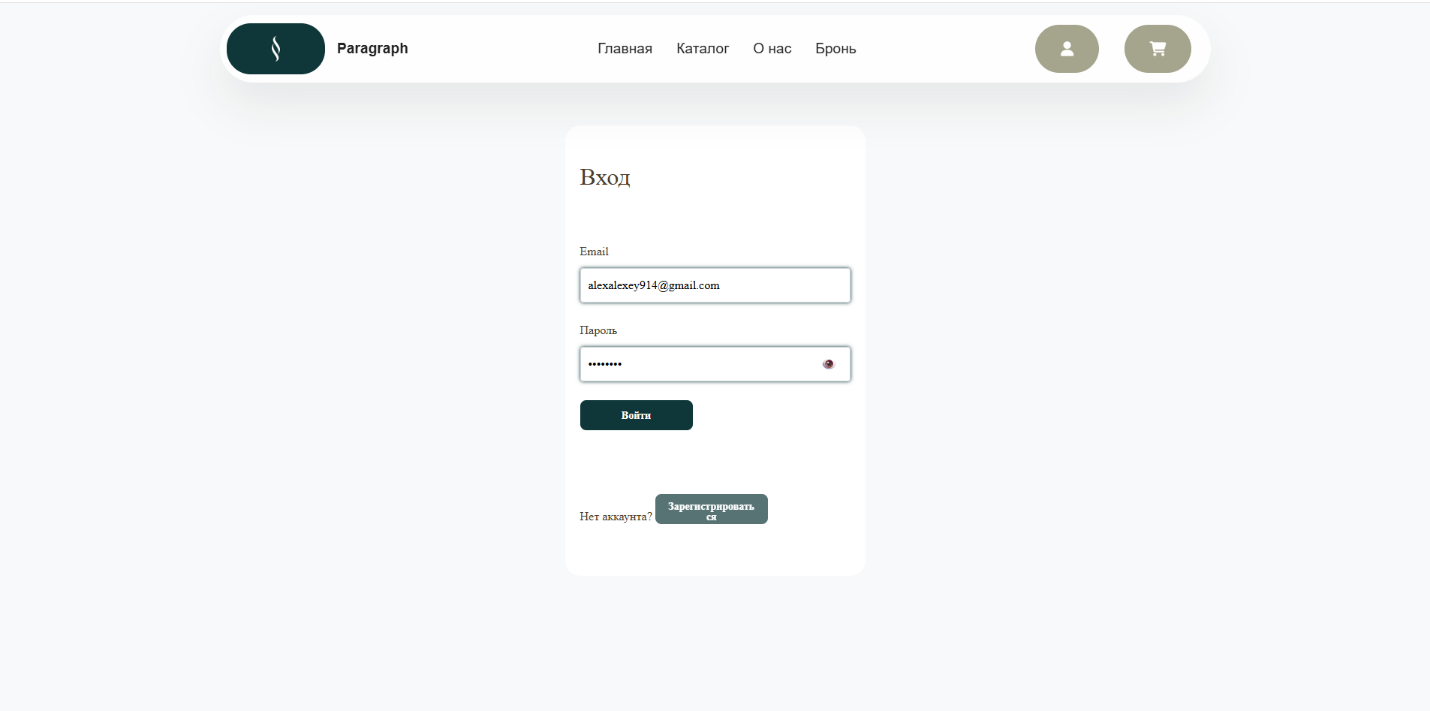
    }

}

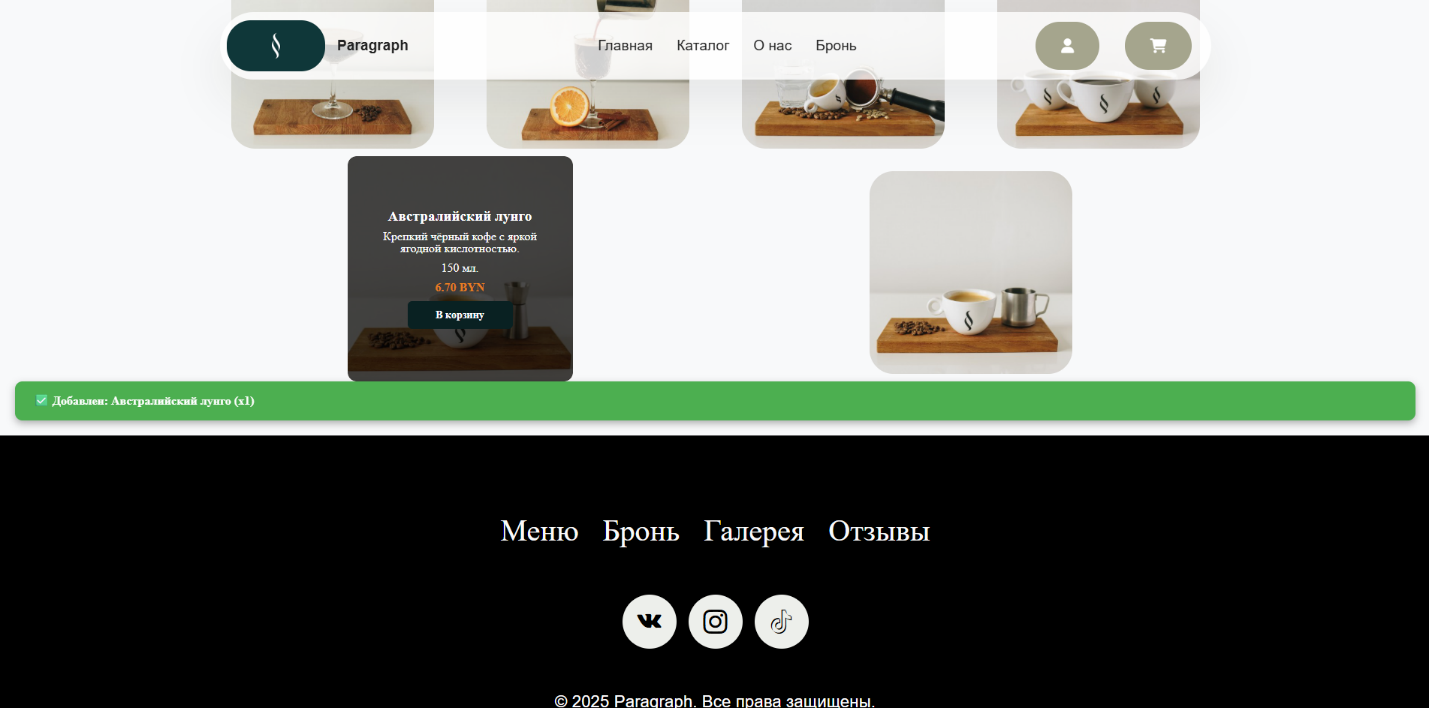
Приложение Д  
Результаты работы веб – приложения



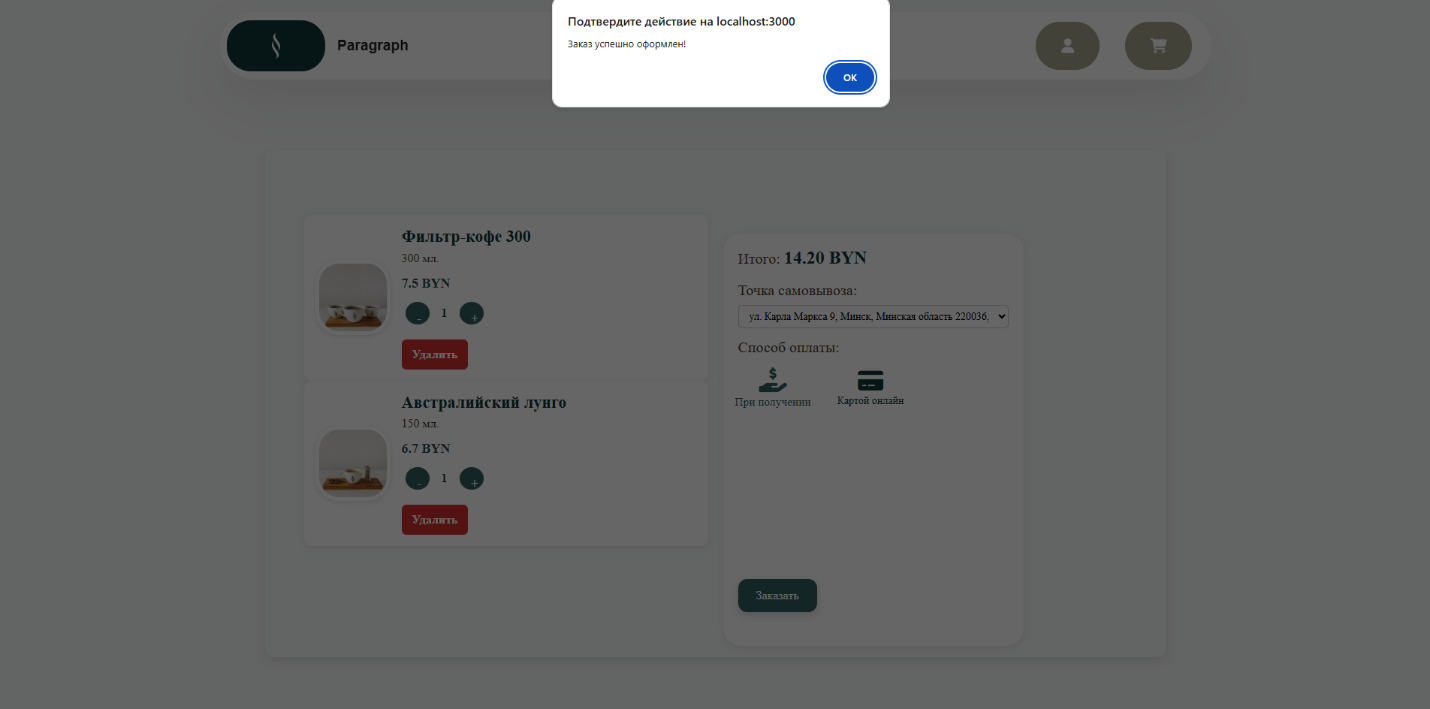
1. ‒ Результат тестирование «Регистрация»



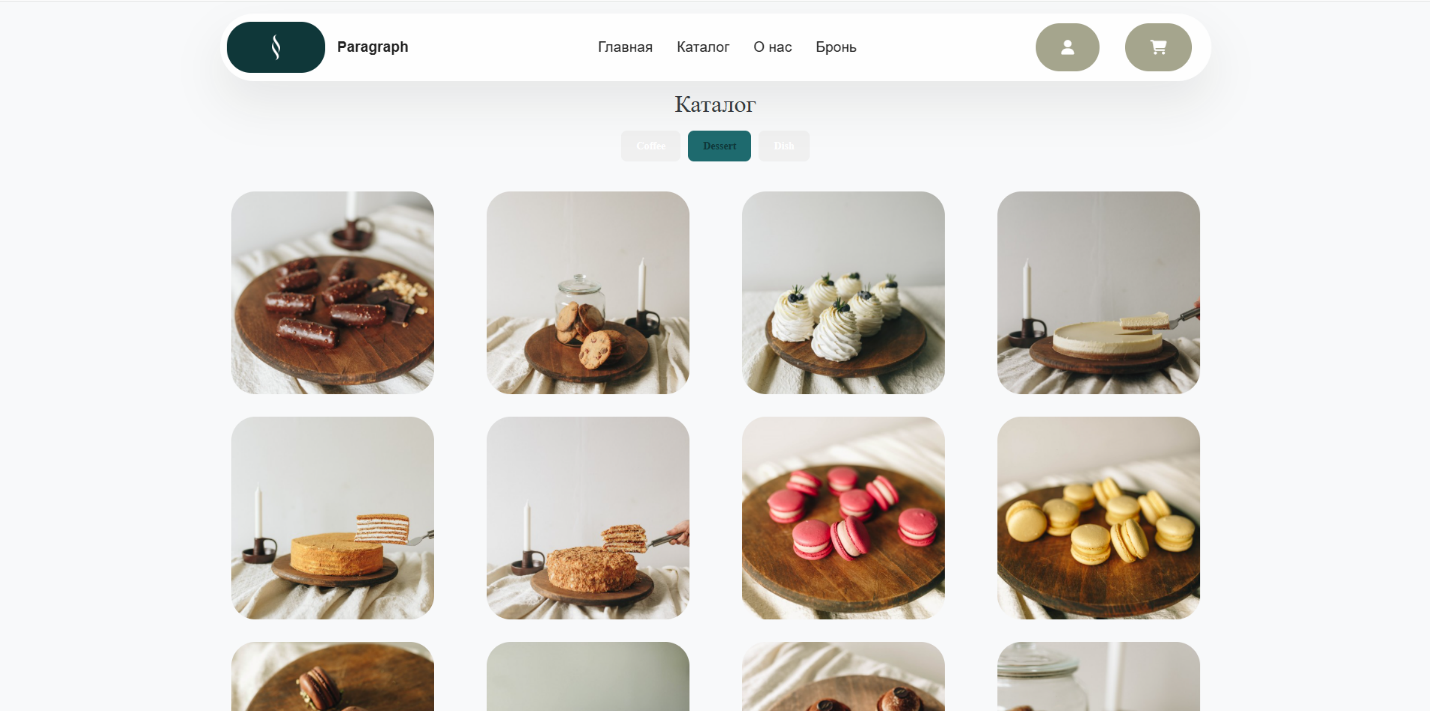
1. ‒ Результат тестирование «Авторизация»



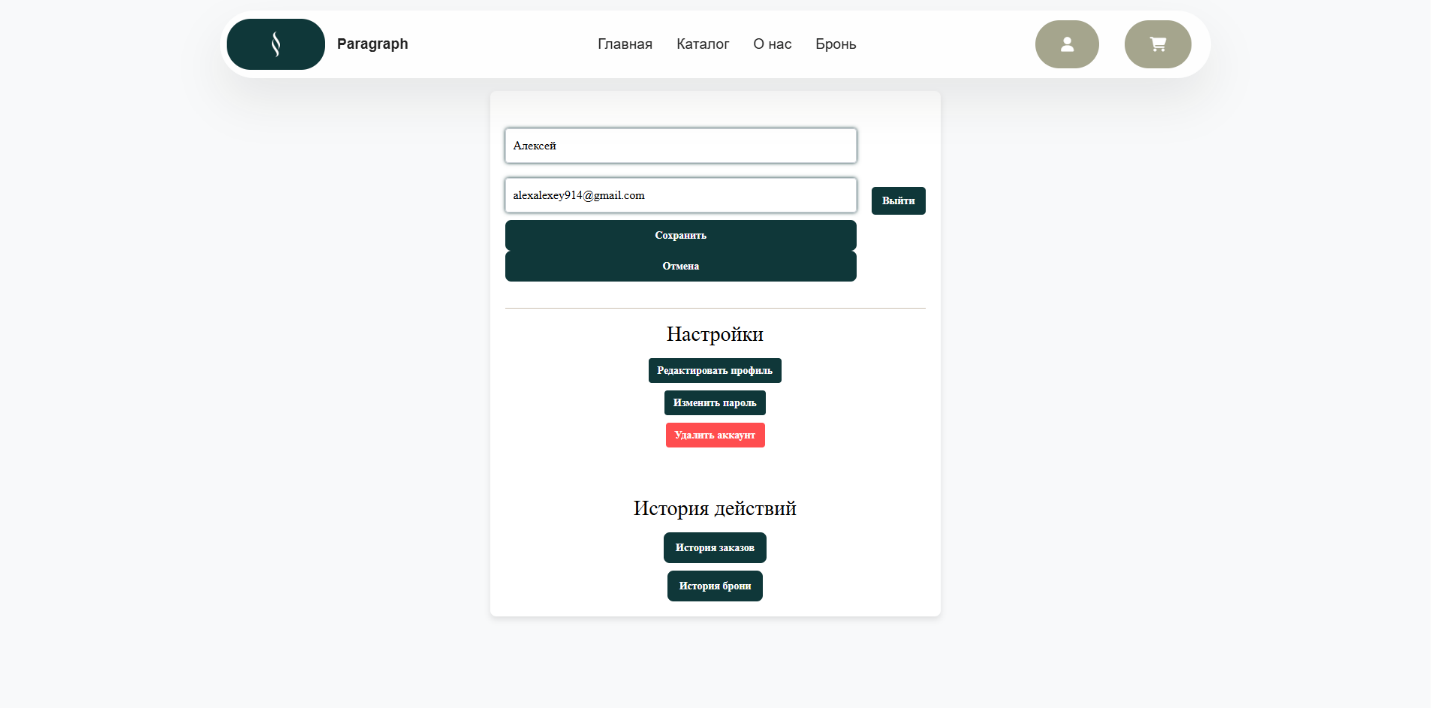
1. ‒ Результат тестирование «Добавление товара в корзину»



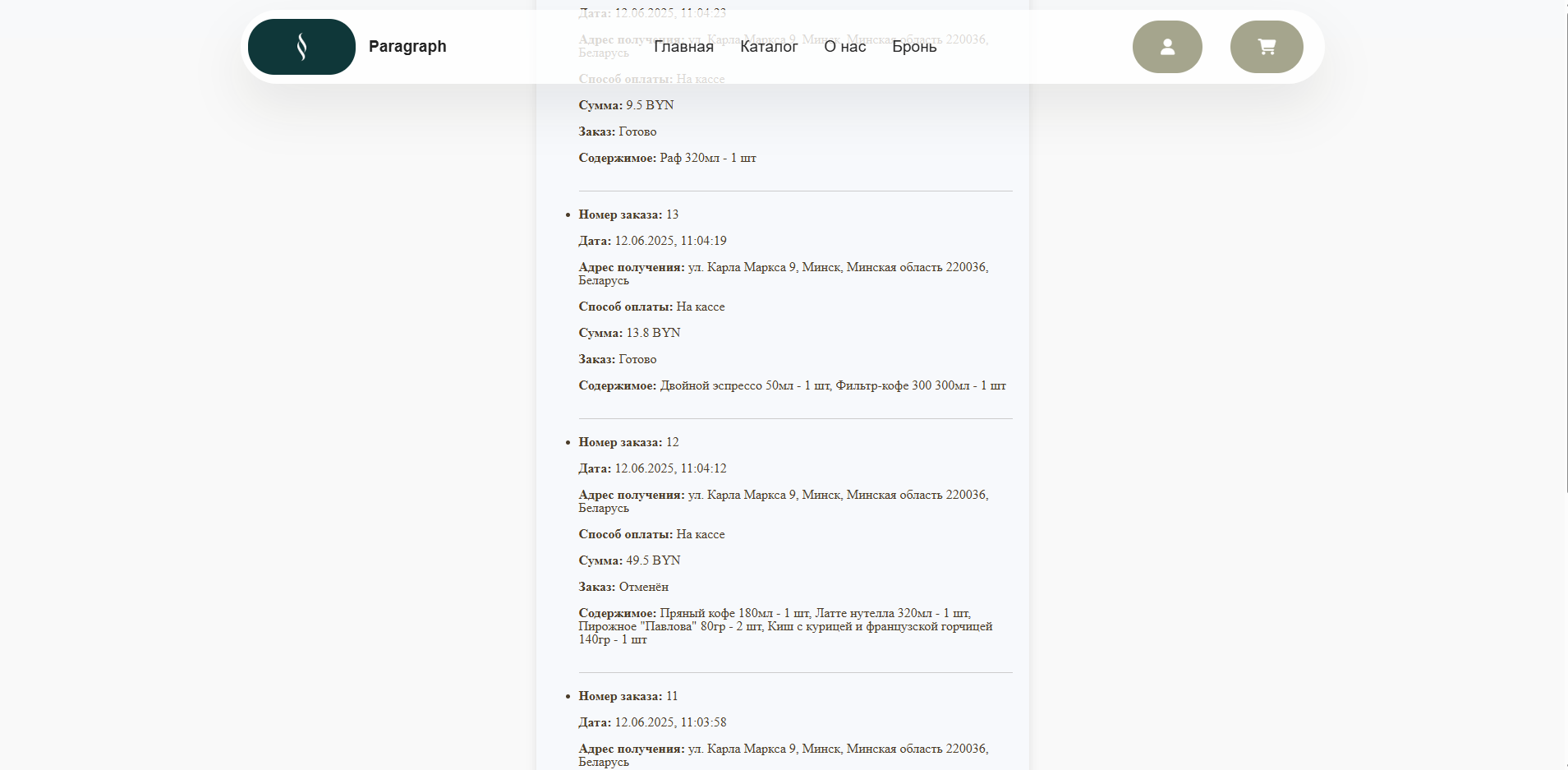
1. ‒ Результат тестирование «Оформление заказа»



1. ‒ Результат тестирование «Фильтрация по категории»



1. ‒ Результат тестирование «Редактирование профиля»



1. ‒ Результат тестирование «Просмотр заказов»

КБП

У

Т. Контр.

Разраб. Разраб.

Латушко А.А.

Провер. Провер.

Кривошеина А.В.

Реценз.

Утверд.

Н. Контр.

Диаграмма прецендентов

№ докум.

Подпись

Дата

Изм.

Лист

Масса

Лит.

Масштаб

Лист 1

Листов 3

Инв.№подл.

Подп. и дата

Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т.318013.401 ГЧ Т.ХХХХХХ.401 ГЧ

КП Т.318013.401 ГЧ



*Разработка веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph»*

*Разработка веб-приложения по автоматизации работы сети магазинов техники «COOLZY»*

КП Т.318013.401 ГЧ

КБП

У

Т. Контр.

Разраб. Разраб.

Латушко А.А.

Провер. Провер.

Кривошеина А.В.

Реценз.

Утверд.

Н. Контр.

№ докум.

Подпись

Дата

Изм.

Лист

Масса

Лит.

Масштаб

Лист 2

Листов 3

Инв.№подл.

Подп. и дата

Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т.318013.401 ГЧ Т.ХХХХХХ.401 ГЧ

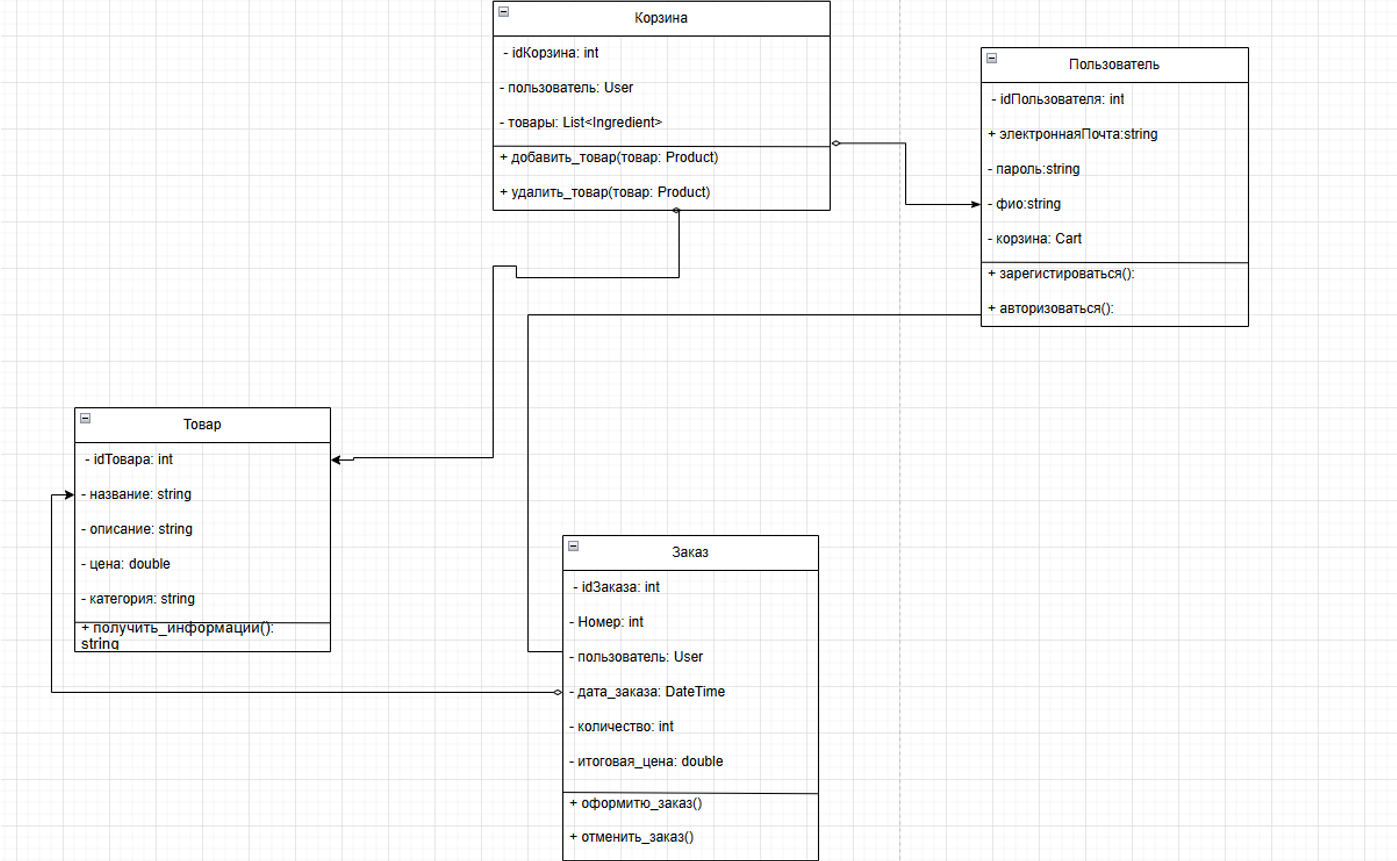


Диаграмма классов

*Разработка веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph»*

*Разработка веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph»*

Диаграмма классов

КП Т.318013.401 ГЧ

КБП

У

Т. Контр.

Разраб. Разраб.

Латушко А.А.

Провер. Провер.

Кривошеина А.В.

Реценз.

Утверд.

Н. Контр.

№ докум.

Подпись

Дата

Изм.

Лист

Масса

Лит.

Масштаб

Лист 3

Листов 3

Инв.№подл.

Подп. и дата

Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т.318013.401 ГЧ Т.ХХХХХХ.401 ГЧ

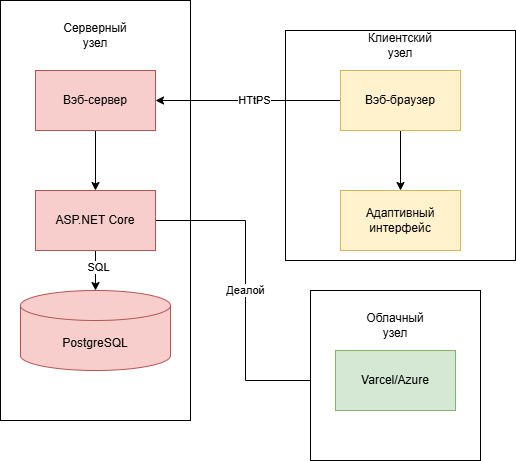


Диаграмма развертывания

*Разработка веб-приложения для автоматизации формирования заказа и оплаты услуг кофейни «Paragraph»*

**Удостоверяющий лист**

электронного документа – курсовой проект

Тема КПРазработка веб-приложения по автоматизации формирования заказа и оплата услуг кофейни «Paragraph»

Обозначение КП Т.318013.401

Разработчик Латушко А.А. Руководитель Кривошеина А.В.

(Ф.И.О.) (Ф.И.О.)

Подписи лиц, ответственных за разработку электронного документа:

|  |  |  |
| --- | --- | --- |
| Состав электронного документа | Разработчик | Руководитель |
| Пояснительная записка (на бумажном носителе формата А4), ПЗ\_Латушко\_Т–318.docх |  |  |
| ГЧ, Диаграммы.docx |  |  |
| Папка с проектом «Сafe» |  |  |
| Файл базы данных «Caffe» |  |  |
| Тип носителя: USB – флешка |  |  |

**Этикетка**

для курсового проекта

**Курсовой проект**

Тема «Разработка веб-приложения по автоматизации формирования заказа и оплата услуг кофейни «Paragraph»

КП Т.318013.401

Разработан:

Разработчик: Латушко А.А.

Руководитель: Кривошеина А.В.

Технические средства: Персональный компьютер с процессором Intel Core i5 или выше, не менее 8 ГБ оперативной памяти, жёсткий диск объёмом от 100 ГБ, монитор с разрешением не ниже 1920×1080, клавиатура, мышь.

Программные средства: Операционная система Windows 10/11, среда разработки Microsoft Visual Studio 2022, платформа .NET 6/7 с использованием ASP.NET Core MVC, система управления базами данных PostgreSQL, браузер Google Chrome, фреймворк React для клиентской части

Состав документа:

Пояснительная записка – ПЗ\_Латушко\_Т-318.docх

Графическая часть –Диаграммы.docx

Папка с проектом – Cafe

Файл базы данных – Caffe