

МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ
Государственное бюджетное профессиональное образовательное учреждение
Московской области «Ногинский колледж»

Курсовой проект

по МДК.09.01 Проектирование и разработка веб-приложений
ПМ.09 Проектирование, разработка и оптимизация веб-приложений

Тема:

ИСПОЛЬЗОВАНИЕ ФРЕЙМВОРКА FLASK В РАЗРАБОТКЕ ВЕБ-
ПРИЛОЖЕНИЯ «СИСТЕМА ДОСТАВКИ ЕДЫ»

Разработчик:
студент группы ЗИСПР
Юманов Е.В.

(подпись)

Оценка защиты курсового проекта

«_____»

Дата защиты

«_____» _____ 2025г.

Руководитель проекта:
преподаватель
Степанов С.О.

(подпись)

Ногинск, 2025 г

СОДЕРЖАНИЕ КУРСОВОГО ПРОЕКТА

ВВЕДЕНИЕ	3
1. Описание предметной области	5
1.1 Краткая характеристика веб-приложения	5
1.1.2 Краткая характеристика “Система доставки еды”	7
1.2 Описание программных инструментов и средств разработки	9
1.2.1 Язык программирования «PYTHON»	9
1.2.2 Фреймворк «FLASK»	10
1.2.3 Серверная часть	11
1.2.4 Среда разработки	12
2. Практическая часть	14
2.1 Проектирование веб-приложения	14
2.2 Подготовка к работе	15
2.3 Разработка веб-приложения	17
2.3.1 Серверная часть	18
2.3.2 Пользовательский интерфейс	24
2.4 Размещение веб-приложения на хостинге	30
Заключение	33
Список использованной литературы	34

ВВЕДЕНИЕ

Задача: Главная задача в курсовой работе – изучение фреймворка Flask и создание на его основе веб-приложения, соответствующего требованиям технического задания. Flask представляет собой легковесный и гибкий инструмент для разработки серверной части веб-приложений с использованием языка программирования Python.

Актуальность: Веб-приложения находят широкое применение в самых разных областях, включая электронную коммерцию, социальные сети, образовательные платформы и системы управления контентом. В данном проекте рассматривается разработка приложения для управления задачами, которое будет полезно как для индивидуальных пользователей, так и для небольших команд.

На сегодняшний день рынок доставки еды представлен множеством платформ, предлагающих разнообразные блюда и кулинарные предложения. С ростом популярности онлайн-заказов еды конкуренция среди сервисов доставки становится все более острой, что побуждает их постоянно совершенствоваться и использовать новые стратегии для привлечения клиентов.

Сервисы доставки еды предлагают широкий ассортимент блюд: от фаст-фуда и пиццы до блюд высокой кухни и вегетарианских опций. Многие из них также предоставляют дополнительные услуги, такие как собственные бренды, программы лояльности и скидок, что помогает удерживать клиентов и стимулировать повторные заказы.

Одним из значимых трендов в сфере доставки еды является быстрое развитие мобильных приложений и мобильной торговли. Пользователи предпочитают заказывать еду с мобильных устройств благодаря удобству и доступности, поэтому многие сервисы активно развивают мобильные версии своих сайтов и создают собственные приложения.

Система доставки еды также сталкивается с проблемами в области защиты данных и конфиденциальности пользователей. В связи с ростом

онлайн-мошенничества и утечек данных возрастает ответственность сервисов за защиту личной информации клиентов и обеспечение безопасных платежей.

В общем, рынок доставки еды демонстрирует стабильный рост и разнообразие, предлагая клиентам широкий выбор блюд и кулинарных услуг, удобные способы оплаты и доставки, а также возможность пользоваться скидками и акциями. Для успешной работы в этой сфере важно следить за актуальными трендами, улучшать сервис и адаптироваться к изменяющимся потребностям клиентов.

Цели:

- Изучить фреймворк Flask и его возможности для создания веб-приложений.
- Освоить дополнительные библиотеки Python, необходимые для реализации требуемой функциональности.
- Разработать веб-приложение, используя Flask для серверной части и SASS для клиентской части.
- Защитить курсовую работу, продемонстрировав понимание процесса создания, настройки и развёртывания приложения.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Краткая характеристика веб-приложения

В эпоху цифровой трансформации веб-приложение является важным элементом современной информационной инфраструктуры. В широком смысле, веб-приложение – это программное обеспечение, доступ к которому осуществляется посредством сети Интернет с использованием веб-браузера. Это отличает его от традиционных десктопных приложений, требующих установки на локальный компьютер пользователя; веб-приложения работают на удаленном сервере, а браузер выступает лишь как интерфейс для взаимодействия с пользователем.

Основой для работы веб-приложений служит клиент-серверная архитектура. Клиент (веб-браузер) отправляет запросы на сервер. Этот сервер обрабатывает их и возвращает результат в виде HTML-страницы, данных в формате JSON или XML, или других типов контента. Этот контент отображается в браузере, предоставляя пользователю возможность взаимодействовать с приложением.

Веб-приложения обладают рядом характеристик, определяющих их значимость и широкое распространение. Веб-приложение характеризуется доступностью: оно доступно с любого устройства, подключенного к сети Интернет, независимо от операционной системы или аппаратной платформы. Это обеспечивает широкую аудиторию пользователей и удобство использования.

Кроме того, для веб-приложений характерно централизованное управление: обновление и обслуживание веб-приложений осуществляется на сервере, что упрощает администрирование и обеспечивает единообразие функциональности для всех пользователей. Также важно отметить масштабируемость: веб-приложения могут быть легко масштабированы для обработки возрастающего числа пользователей и объемов данных. Это достигается за счет использования современных технологий, таких как

облачные вычисления и распределенные системы. Нельзя забывать и о безопасности: разработчики веб-приложений уделяют большое внимание вопросам безопасности, реализуя механизмы аутентификации, авторизации, защиты от уязвимостей и шифрования данных. Наконец, интерактивность также важна: веб-приложения предоставляют пользователям возможность активно взаимодействовать с данными и функциональностью, обеспечивая динамический и персонализированный пользовательский опыт.

Разработка веб-приложений включает в себя использование различных технологий и подходов. На стороне клиента (front-end) применяются языки HTML, CSS и JavaScript, а также фреймворки, такие как React, Angular и Vue.js. На стороне сервера (back-end) используются языки программирования, такие как Python, Java, PHP, Node.js, и фреймворки, такие как Flask, Django, Spring, Express.js. Для хранения и управления данными применяются системы управления базами данных (СУБД), такие как MySQL, PostgreSQL, MongoDB и другие.

В современном мире веб-приложения используются в самых разных областях: от электронной коммерции и социальных сетей до банковского дела и здравоохранения. Они стали неотъемлемой частью нашей жизни, предоставляя нам удобный и эффективный способ доступа к информации и услугам.

1.1.2 Краткая характеристика “Система доставки еды”

Система доставки еды представляет собой комплексное программное решение, предназначенное для автоматизации и оптимизации процессов, связанных с заказом, приготовлением и доставкой еды от поставщика (ресторана, кафе, службы доставки) к потребителю. Эти системы сочетают в себе функциональность веб-приложения для взаимодействия с пользователями и внутренних инструментов для управления операциями.

Ключевая цель системы доставки еды – обеспечить удобный и эффективный способ для клиентов заказывать еду онлайн, а также предоставление поставщикам инструментов для эффективного управления заказами, курьерами и другими аспектами бизнеса.

Основные функциональные компоненты системы доставки еды включают пользовательский интерфейс (клиентское приложение). Этот компонент позволяет пользователям просматривать меню, выбирать блюда, добавлять их в корзину, указывать адрес доставки, выбирать способ оплаты и оформлять заказ. Важными аспектами являются удобство навигации, понятное представление информации о блюдах (состав, калорийность, фотографии) и возможность персонализации заказа.

Важна также панель управления рестораном (или поставщиком): этот компонент предоставляет поставщикам инструменты для управления меню (добавление, редактирование, удаление блюд), ценами, акциями, графиком работы, а также для отслеживания заказов, управления курьерами и получения отчетов о продажах. Кроме того, существует система управления курьерами: этот компонент позволяет отслеживать местоположение курьеров, назначать им заказы, оптимизировать маршруты доставки и контролировать время доставки.

Также важна система оплаты: интеграция с платежными шлюзами обеспечивает возможность приема онлайн-платежей различными способами (банковские карты, электронные кошельки и т.д.). Существует и система уведомлений: автоматические уведомления информируют клиентов о статусе заказа (принят, готовится, в пути, доставлен), а также оповещают поставщиков

о новых заказах и других важных событиях. Наконец, нельзя забывать о системе аналитики и отчетности: этот компонент предоставляет данные о продажах, популярных блюдах, эффективности работы курьеров и другие показатели, необходимые для принятия управленческих решений.

Современные системы доставки еды часто включают в себя дополнительные функции. Среди них интеграция с картографическими сервисами, которая нужна для точного определения местоположения клиента и построения оптимальных маршрутов доставки. Также часто используется система лояльности, предназначенная для привлечения и удержания клиентов с помощью программ лояльности, скидок и бонусов. Распространена поддержка различных языков и валют, что необходимо для работы на международном рынке. И, наконец, часто используется чат-бот, предназначенный для автоматической обработки типовых запросов клиентов.

Разработка системы доставки еды требует учета множества факторов, включая потребности целевой аудитории, особенности бизнес-процессов поставщика и требования безопасности. Успешная система доставки еды способна значительно повысить эффективность бизнеса, улучшить качество обслуживания клиентов и увеличить прибыль.

1.2 ОПИСАНИЕ ПРОГРАММНЫХ ИНСТРУМЕНТОВ И СРЕДСТВ РАЗРАБОТКИ

1.2.1 Язык программирования «PYTHON»

Данный язык программирования, разработанный Гвидо ван Россумом и впервые представленный в 1991 году, представляет собой высокоуровневую, интерпретируемую и объектно-ориентированную среду, предназначенную для решения широкого спектра задач. Философия его дизайна делает акцент на удобочитаемости кода, а синтаксические конструкции позволяют программистам выражать сложные концепции в относительно небольшом объеме кода, что особенно заметно при сравнении с такими языками, как C++ или Java.

Ключевые характеристики данной среды программирования включают простоту и выразительность синтаксиса, что значительно облегчает его освоение и понимание. Это достигается за счет использования отступов для структурирования кода, что отличает его от языков, использующих фигурные скобки или ключевые слова.

Важным аспектом является также динамическая типизация, когда тип переменной определяется непосредственно во время выполнения программы, что упрощает разработку, но требует более тщательного тестирования. Автоматическое управление памятью, реализуемое через сборщик мусора, освобождает разработчика от ручного управления ресурсами, снижая вероятность ошибок. Обширная стандартная библиотека предоставляет множество готовых модулей для решения разнообразных задач, включая обработку текстовой информации, взаимодействие с файловой системой и сетевое программирование.

Поддержка различных парадигм, таких как объектно-ориентированное, процедурное и функциональное программирование, позволяет адаптировать подход к особенностям конкретной задачи. Наконец, нельзя не отметить кроссплатформенность, обеспечивающую возможность работы на различных операционных системах, включая Windows, macOS и Linux.

Сфера применения этого языка программирования весьма широка и включает веб-разработку (благодаря таким фреймворкам, как Django и Flask), научные вычисления и анализ данных (с использованием библиотек NumPy, SciPy, Pandas и Matplotlib), машинное обучение и искусственный интеллект (где применяются библиотеки TensorFlow, Keras и PyTorch), автоматизацию задач и написание скриптов для управления системами, а также разработку игр (с использованием, например, библиотеки Pygame).

1.2.2 Фреймворк «FLASK»

Фреймворк представляет собой комплексный набор инструментов, библиотек и установленных соглашений, предназначенных для упрощения и ускорения разработки программного обеспечения. Он предлагает готовую структуру и переиспользуемые компоненты, позволяя разработчикам сосредоточиться на логике приложения, а не на рутинных задачах.

Данный фреймворк, разработанный для веб-разработки на базе языка Python, классифицируется как микрофреймворк. Это обусловлено тем, что он предоставляет только основные инструменты и функциональность, необходимые для создания веб-приложений, не навязывая при этом определенные библиотеки или инструменты, что позволяет разработчикам выбирать оптимальные решения для конкретного проекта.

Ключевые особенности рассматриваемого фреймворка включают легкость и гибкость, что облегчает его освоение и использование. Он не диктует строгие правила, предоставляя разработчикам свободу принятия решений. Кроме того, он характеризуется расширяемостью, благодаря множеству сторонних расширений, добавляющих функциональность, такую как взаимодействие с базами данных или аутентификация пользователей. Интегрированный шаблонизатор Jinja2 упрощает создание динамических веб-страниц, а наличие встроенного веб-сервера облегчает разработку и тестирование. Поддержка WSGI (Web Server Gateway Interface) обеспечивает совместимость с различными веб-серверами.

Преимуществами использования данного фреймворка являются простота

и скорость разработки, обеспечиваемые его гибкостью. Он предоставляет разработчикам полный контроль над проектом, позволяя выбирать подходящие инструменты и библиотеки. Активное сообщество пользователей обеспечивает поддержку и доступ к различным ресурсам. Он особенно хорошо подходит для небольших и средних проектов, требующих гибкости и индивидуального подхода.

Рассматриваемый фреймворк может применяться для создания различных типов веб-приложений, включая веб-сайты с динамическим контентом, веб-API, предоставляющие данные и функциональность другим приложениям, микросервисы, представляющие собой небольшие независимые приложения, и прототипы, позволяющие быстро проверить идеи и концепции.

1.2.3 Серверная часть

Для успешной реализации проекта с использованием Flask требуется надлежащая настройка серверной части. Это включает в себя установку и конфигурацию необходимых программных компонентов, обеспечивающих функционирование веб-приложения.

Ключевыми элементами являются необходимость установки интерпретатора языка программирования, на котором базируется фреймворк. Рекомендуется применение актуальной стабильной версии, относящейся к третьему поколению.

Также необходим менеджер пакетов, предназначенный для управления библиотеками и фреймворками, такими как используемый в проекте. Как правило, он поставляется в комплекте с интерпретатором. Рекомендуется создание изолированной среды для каждого проекта, что позволяет избежать конфликтов зависимостей между различными проектами. Для этой цели можно использовать стандартный модуль `venv`. Непосредственно сам фреймворк также необходим, он устанавливается в созданной изолированной среде с помощью менеджера пакетов.

Для развертывания готового приложения необходимо наличие веб-

сервера. В качестве вариантов можно рассматривать WSGI-сервер, отличающийся простотой настройки и хорошей производительностью. Альтернативой является другой WSGI-сервер, предоставляющий расширенные возможности конфигурации. Также возможно использование популярных веб-серверов в качестве проксирующих для вышеупомянутых WSGI-серверов, что повышает безопасность и производительность.

При необходимости хранения данных следует установить и настроить соответствующую систему управления базами данных. Поддерживаются различные варианты, от легковесной встраиваемой системы, подходящей для небольших проектов, до реляционных СУБД, предназначенных для средних и крупных проектов. Кроме того, возможно использование NoSQL-решений, обеспечивающих гибкость в структуре данных.

Процесс настройки включает установку интерпретатора и менеджера пакетов, создание и активацию изолированной среды, установку фреймворка и других необходимых библиотек, настройку веб-сервера, а также конфигурацию базы данных (при необходимости).

1.2.4 Среда разработки

Среда разработки представляет собой программный комплекс, предоставляющий разработчикам интегрированный набор инструментов для написания, отладки и тестирования программного кода. Как правило, IDE включает в себя редактор кода с подсветкой синтаксиса, средства отладки, систему контроля версий и другие полезные функции.

Мы будем использовать Py Charm, IDE разработанная компанией JetBrains, является популярным решением для разработки на языке Python. Она предоставляет широкий спектр инструментов и функций, упрощающих и ускоряющих процесс создания приложений.

Основные возможности данной среды включают интеллектуальный редактор кода, обеспечивающий подсветку синтаксиса, автодополнение и мгновенную проверку ошибок. Встроенный отладчик позволяет осуществлять пошаговое выполнение кода, устанавливать точки останова и просматривать

значения переменных. Интеграция с системами контроля версий, такими как Git, Mercurial и Subversion, упрощает управление проектом. Поддержка различных фреймворков, включая Django, Flask и Pyramid, расширяет возможности разработки.

Интегрированные инструменты для тестирования позволяют запускать тесты и анализировать результаты. Функции рефакторинга кода обеспечивают возможность изменения структуры кода без нарушения его функциональности. Кроме того, предусмотрена возможность подключения к различным базам данных и работа с данными. Поддержка плагинов позволяет расширять функциональность среды разработки.

Преимуществами использования данного решения являются повышение производительности за счет упрощения и ускорения процесса разработки, улучшение качества кода благодаря инструментам проверки, рефакторинга и тестирования, удобный и интуитивно понятный интерфейс, а также наличие активного сообщества пользователей, обеспечивающего поддержку и доступ к ресурсам.

Процесс установки и настройки включает загрузку дистрибутива с официального сайта, установку в соответствии с инструкциями, запуск приложения, настройку параметров (тема оформления, шрифты и т.д.), выбор интерпретатора (автоматическое обнаружение или ручное указание пути) и создание или открытие проекта.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Проектирование веб-приложения

Процесс проектирования веб-приложения начинается с тщательного анализа потребностей пользователей и понимания специфики сферы, для которой разрабатывается продукт. В рамках создания веб-приложения **Kitchen Cloud** — ресторана высокого класса, предоставляющего услуги по онлайн-заказу и доставке еды, особое внимание уделялось удобству использования, безопасности и надёжности работы системы. Каждое решение при проектировании было обусловлено стремлением удовлетворить все запросы целевой аудитории.

Важной особенностью веб-приложений доставки еды является то, что они должны обеспечивать максимально простой и быстрый доступ к меню, а также удобный механизм оформления заказов. Для этого был выбран стек технологий на основе языка Python и фреймворка Flask. Такой выбор позволяет быстро разрабатывать и развивать функционал, адаптируя проект под меняющиеся требования пользователей и рынка. Flask отличается простотой в освоении, гибкостью и расширяемостью, а также поддержкой множества библиотек, что существенно ускоряет процесс разработки. В сочетании с системой шаблонов Jinja2, он позволяет создавать динамические страницы, где контент подстраивается под конкретные запросы пользователя.

Одной из ключевых задач на этапе проектирования является проработка структуры базы данных. В приложении реализована база данных SQLite, которая благодаря своей лёгкости и простоте обслуживания отлично подходит для небольших и средних проектов. В ней предусмотрены таблицы для хранения информации о пользователях, заказах, позициях в заказе и блюдах. Это позволяет гибко управлять данными и быстро получать необходимые сведения при выполнении запросов.

Важным аспектом при проектировании является обеспечение безопасности пользователей. Приложение предусматривает аутентификацию и

разграничение прав доступа, чтобы курьеры и администраторы могли получать только те функции, которые им необходимы. Это снижает риски несанкционированного доступа к информации и гарантирует корректную работу всех бизнес-процессов. С помощью Flask-Login реализована авторизация пользователей, а для защиты персональных данных в формах используются механизмы проверки и валидации.

Визуальная составляющая приложения также не осталась без внимания. Интерфейс выполнен с применением современных принципов UX/UI-дизайна, обеспечивая пользователю лёгкость навигации и интуитивно понятный доступ ко всем функциям. Для лучшего взаимодействия с пользователями приложение адаптировано под различные устройства, включая мобильные. Это особенно актуально для сферы доставки еды, так как всё больше заказов оформляется именно со смартфонов.

Кроме того, при проектировании учитывается возможность масштабирования приложения и последующего развития функционала. Flask позволяет легко подключать дополнительные модули, такие как электронная почта или интеграция с другими системами. Это гарантирует, что приложение сможет расти вместе с потребностями бизнеса, сохраняя при этом стабильность и производительность.

Таким образом, проектирование веб-приложения Kitchen Cloud сочетает в себе внимательное изучение требований конечного пользователя, выбор оптимального стека технологий, обеспечение безопасности и надёжности, а также адаптивный и современный дизайн. Всё это создаёт фундамент, на котором строится удобный и функциональный сервис, готовый к работе в условиях реальной эксплуатации.

2.2 Подготовка к работе

Перед началом непосредственной разработки веб-приложения Kitchen Cloud была проведена комплексная подготовка рабочего окружения, которая позволила обеспечить корректное и стабильное функционирование всех

элементов проекта. Этот этап включал установку всех необходимых инструментов и библиотек, а также настройку системы управления версиями и подготовку к взаимодействию с базой данных.

Для реализации веб-приложения используется язык программирования Python, который предоставляет широкие возможности для серверной логики и взаимодействия с базой данных.

Установка всех необходимых зависимостей начинается с установки самого Python, с учетом моих потребностей и выбранных библиотек я сделал выбор в сторону версии - 3.10.11. Для обеспечения виртуальной изоляции проекта и недопущения конфликтов версий пакетов используется модуль `venv`, позволяющий создавать виртуальные окружения. Это обеспечило стабильность разработки и возможность гибкой работы с зависимостями.

После установки Python, для установки самого Flask следует зайти в командную строку (cmd) и прописать команду `python3 -m pip install flask` (либо же `pip3 install flask`). Дополнительные модули – `flask-login` (`python3 -m pip install flask-login`), `smtplib`, `sqlite3`, `string`, `random`, `os` (идут в комплекте при установке Python)

Для создания виртуального окружения в командной строке Windows используется следующая команда: `python -m venv venv`, А активация виртуального окружения производится с помощью команды: `venv\Scripts\activate`.

После активации виртуального окружения происходит установка всех зависимостей, которые были перечислены в файле `requirements.txt`. Этот файл содержал список всех необходимых библиотек и их версий, что позволило легко воспроизвести окружение проекта на любой машине. Установка зависимостей осуществляется командой: `pip install -r requirements.txt`.

Основные зависимости, используемые в проекте, включают в следующие пакеты:

- Flask — фреймворк для создания веб-приложения.
- Flask-Login — для реализации аутентификации пользователей.

- Flask-WTF — для работы с веб-формами.
- Flask-SQLAlchemy — для взаимодействия с базой данных.
- SQLite — легковесная СУБД для хранения данных приложения.
- Jinja2 — для шаблонизации HTML-страниц.
- И другие необходимые библиотеки, перечисленные в requirements.txt.

Помимо этого, важной составляющей подготовки является создание файла конфигурации `config.py`, который содержит все основные настройки проекта, такие как путь к базе данных, секретный ключ и список возможных статусов заказов.

Настройка базы данных будет производиться при помощи SQLite. Для первоначальной инициализации структуры базы данных был написан отдельный скрипт `init_db.py`, который запускался следующей командой: `python init_db.py`.

Это обеспечивает создание всех необходимых таблиц, таких как `users`, `products`, `orders` и `order_items`, а также возможность дальнейшей их модификации при необходимости.

Подготовка к работе завершается проверкой корректности установки всех зависимостей и запуском сервера разработки для тестирования основных функций приложения: `python app.py`.

Результатом всех подготовительных действий стало полностью готовое к разработке окружение, которое позволяло сосредоточиться на реализации всех необходимых функций приложения. Этот этап обеспечил надежную основу для последующих шагов, таких как разработка интерфейсов, подключение базы данных и реализация логики приложения.

2.3 Разработка веб-приложения

Создание веб-приложения Kitchen Cloud представляет собой поэтапный процесс, в ходе которого разрабатываются и внедряются все основные компоненты: от базы данных и серверной логики до элементов

пользовательского интерфейса. Разработка ведется с опорой на современные подходы, которые обеспечивают надежность, безопасность и удобство в использовании конечного продукта.

Важной особенностью приложения является его модульная архитектура. Серверная часть построена с использованием микрофреймворка Flask. Этот фреймворк позволяет организовать гибкую структуру проекта, разделив его на отдельные модули: маршруты для работы с пользователями, заказами, панелью администратора и панелью курьера.

2.3.1 Серверная часть

Фундаментом серверной логики приложения выступал файл `app.py`, который запускал сервер Flask и обеспечивал инициализацию всех необходимых маршрутов и конфигураций. Он подключал такие модули, как `models.py` (отвечающий за взаимодействие с базой данных и хранение моделей), а также различные модули маршрутов (Для начала работы нужно создать файлы, в котором будет писаться весь код. Основной файл запуска сервера называется `app.py`, в нем можно писать код единым целым, но я сделал иначе. Я разделил код на разные файлы. К примеру за страницу с админ-панелью отвечает файл `admin_routes.py`, а за страницу с авторизацией отвечает `auth_routes.py`).

Работа с базой данных реализована через SQLite. Для взаимодействия с ней использовалась библиотека `sqlite3`, а для удобства запросов применяется ORM-библиотека `Flask-SQLAlchemy`, которая обеспечивает более высокоуровневый подход к работе с данными. Таблицы базы данных включают пользователей, заказы, позиции в заказе и продукты.

```
def init_db():
    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            is_admin TEXT NOT NULL,
            is_courier TEXT NOT NULL)''')

        cursor.execute('''CREATE TABLE IF NOT EXISTS orders (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            courier_id INTEGER,
            status TEXT NOT NULL,
            created_at TEXT DEFAULT (datetime('now'))''')

        cursor.execute('''CREATE TABLE IF NOT EXISTS products (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            price TEXT NOT NULL,
            image TEXT,
            category TEXT NOT NULL)''')

        cursor.execute('''CREATE TABLE IF NOT EXISTS order_items (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            order_id TEXT NOT NULL,
            name TEXT,
            price TEXT,
            quantity INTEGER DEFAULT 1)''')
```

Рис. 1. Создание таблиц бд.

Реализация авторизации и аутентификации пользователей производится с использованием расширения Flask-Login. Оно позволяет пользователям входить в систему, защищая маршруты, требующие авторизации, и ограничивая доступ к функциям, доступным только администраторам или курьерам. Для начала мы находим поля username и password в HTML. Затем генерируем хешированный пароль для обеспечения безопасности, чтобы в случае утечки данных злоумышленники не могли использовать их. После этого обращаемся к базе данных и заполняем введенные данные. Происходит автоматическая авторизация в аккаунт пользователя. Если имя пользователя уже занято, сайт выдаст ошибку о том, что такое имя пользователя уже существует.

```

@auth_routes.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.find_by_username(username)
        if user and check_password_hash(user.password, password):
            login_user(user)
            return redirect(url_for('main_routes.index'))
        else:
            flash('Неверное имя пользователя или пароль.', 'error')
            return render_template('login.html')

@auth_routes.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')
        try:
            with sqlite3.connect(Config.DATABASE) as conn:
                cursor = conn.cursor()
                cursor.execute('INSERT INTO users (username, password, is_admin, is_courier) VALUES (?, ?, ?, ?)',
                               (username, hashed_password, "no", "no"))
                conn.commit()
                cursor.execute('SELECT id FROM users WHERE username = ?', (username,))
                user_id = cursor.fetchone()[0]
                new_user = User(id=user_id, username=username, password=hashed_password, is_admin=False, is_courier=False)
                login_user(new_user)
                return redirect(url_for('main_routes.index'))
        except sqlite3.IntegrityError:
            flash('Пользователь с таким именем уже существует.', 'error')
            return redirect(url_for('auth_routes.register'))
    return render_template('register.html')

@auth_routes.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('main_routes.index'))

```

Рис. 2. Страница авторизации и регистрации.

Файл `main_routes.py` отвечает за отображение главной страницы с меню, обработку заказов, работу с корзиной пользователя и просмотр истории заказов. Для каждого действия были созданы отдельные маршруты. Так, например, оформление заказа происходило в маршруте `/checkout`, который принимает данные из формы, проверяет их корректность, сохраняет заказ в базу данных и очищает корзину пользователя после успешной покупки.

```

@main_routes.route('/checkout', methods=['POST'])
@login_required
def checkout():
    cart_items = session.get('cart', [])
    email = request.form['email']
    phone = request.form['phone']
    address = request.form['address']
    delivery = request.form['delivery_option']
    order_number = generate_order_number()
    status = 'Курьер забрал заказ и направляется к вам'

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('INSERT INTO orders (user_id, status) VALUES (?, ?)',
                       (current_user.id, status))
        order_id = cursor.lastrowid

        for item in cart_items:
            cursor.execute('INSERT INTO order_items (order_id, name, price, quantity) VALUES (?, ?, ?, ?)',
                           (order_id, item['name'], item['price'], item['quantity']))

        conn.commit()

    session.pop('cart', None)
    return redirect(url_for('main_routes.view_order', order_id=order_id))

```

Рис. 3. Создание заказа.

Также удобно реализована функция добавления в корзину. При добавлении корзину товар он появляется в ней, но при повторном добавлении того же товара в корзину, он не добавляется еще одним отдельным товаром, а просто увеличивает значение поля quantity из запроса. Корзина работает в сессии, что удобно ведь для хранения корзины не нужна отдельная база данных.

```
@main_routes.route('/cart', methods=['GET', 'POST'])
@login_required
def cart():
    if request.method == 'POST':
        product_name = request.form.get('product_name')
        product_price = float(request.form.get('product_price'))
        product_type = request.form.get('product_type')

        if 'cart' not in session:
            session['cart'] = []

        cart = session['cart']

        found = False
        for item in cart:
            if item['name'] == product_name:
                item['quantity'] += 1
                found = True
                break

        if not found:
            cart.append({
                'name': product_name,
                'price': product_price,
                'product_type': product_type,
                'quantity': 1
            })

        session['cart'] = cart
        session.modified = True
        return redirect(url_for('main_routes.index'))

    cart_items = session.get('cart', [])
    total = calculate_total(cart_items)
    return render_template('cart.html', cart_items=cart_items, total=total)
```

Рис. 4. Корзина.

Особое внимание при разработке уделяется обработке ошибок. Все критические операции, такие как взаимодействие с базой данных, оборачиваются в конструкции try-except, что позволит избежать сбоев в работе приложения и предоставляет пользователю понятные уведомления об ошибках.

Курьерская панель веб-приложения Kitchen Cloud представляет собой специализированный раздел, предназначенный для управления статусами заказов. Она разработана с учетом прав доступа пользователей и позволяет курьерам и администраторам просматривать текущие заказы и вносить изменения в их состояние.

Функция courier_panel, реализованная в модуле маршрутов courier_routes.py, отвечает за обработку двух режимов работы: отображение

всех заказов и обновление их статуса. При получении POST-запроса об изменении статуса заказов она считывает переданные данные, а затем обновляет соответствующую запись в базе данных. Для предотвращения несанкционированного доступа используется специальный декоратор `courier_required`, который проверяет роль пользователя. Такой подход гарантирует, что только уполномоченные пользователи смогут управлять статусами заказов, обеспечивая безопасность и целостность данных.

Результатом работы панели является вывод таблицы заказов, включающей уникальный идентификатор и текущий статус каждого заказа. Курьерская панель служит важным инструментом взаимодействия между администрацией ресторана и пользователями, повышая прозрачность и эффективность доставки.

```
def courier_required(f):
    @login_required
    def decorated_function(*args, **kwargs):
        if not (current_user.is_courier == "yes" or current_user.is_admin == "yes"):
            flash('У вас нет прав для просмотра данной страницы.', 'error')
            return redirect(url_for('auth_routes.login'))
        return f(*args, **kwargs)
    decorated_function.__name__ = f.__name__
    return decorated_function

@courier_routes.route('/courier', methods=['GET', 'POST'])
@courier_required
def courier_panel():
    if request.method == 'POST':
        order_id = request.form['order_id']
        new_status = request.form['status']
        with sqlite3.connect(Config.DATABASE) as conn:
            cursor = conn.cursor()
            cursor.execute('UPDATE orders SET status = ? WHERE id = ?', (new_status, order_id))
            conn.commit()
            flash('Статус заказа успешно обновлен.', 'success')
            return redirect(url_for('courier_routes.courier_panel'))

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('SELECT id, status FROM orders')
        orders = cursor.fetchall()

    return render_template('couriers.html', current_user=current_user, orders=orders)
```

Рис. 5. Панель курьеров.

Панель администратора приложения Kitchen Cloud разработана для централизованного управления данными и пользователями. В основе панели лежит модуль `admin_routes.py`, который предоставляет функционал для управления правами пользователей, созданием и удалением продуктов, а также для добавления учетных записей курьеров.

Основная функция `admin_panel` отображает панель управления с возможностью быстрого доступа ко всем административным функциям. Изменение статусов пользователей осуществляется через маршрут `change_admin_status`, который позволяет назначать или снимать права администратора у конкретных пользователей.

Функция `create_product` предоставляет гибкие возможности для администрирования товарного каталога. Она обрабатывает файлы изображений, проверяя их расширения и сохраняет загруженные изображения в директорию проекта `/uploads`. Добавление новых позиций в меню и их удаление осуществляется посредством соответствующих SQL-запросов. При этом предусмотрена валидация данных, которая предотвращает ошибки при работе с базой данных.

```
@admin_routes.route('/create_product', methods=['POST'])
@admin_required
def create_product():
    name = request.form['product_name']
    price = request.form['product_price']
    category = request.form['category']
    action = request.form['action4']
    file = request.files.get('product_image')
    filename = None

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath = os.path.join(current_app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        if action == 'create':
            cursor.execute('INSERT INTO products (name, price, category, image) VALUES (?, ?, ?, ?)', (name, price, category, filename))
            conn.commit()
            flash(f"Товар успешно добавлен.", 'success')
        else:
            cursor.execute('DELETE FROM products WHERE name = ? AND price = ? AND category = ?', (name, price, category))
            conn.commit()
            if cursor.rowcount == 0:
                flash(f"Данный товар не найден.", 'error')
            else:
                flash(f"Товар успешно удален", 'success')

    return redirect(url_for('admin_routes.admin_panel'))
```

Рис. 6. Функция добавления и удаления товара.

Кроме того, реализован маршрут `change_courier_status`, обеспечивающий возможность администратору создать учетные записи курьеров с зашифрованными паролями, что позволяет легко добавлять новых курьеров при такой потребности. Для генерации идентификатора заказа используется простая реализация инкремента.

```

@admin_routes.route('/change_courier_status', methods=['POST'])
@admin_required
def change_courier_status():
    courier_username = request.form['courier_username']
    courier_password = request.form['courier_password']
    hashed_password = generate_password_hash(courier_password, method='pbkdf2:sha256')

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('INSERT INTO users (username, password, is_admin, is_courier) VALUES (?, ?, ?, ?)', (courier_username, hashed_password, "no", "yes"))
        conn.commit()
        flash('Аккаунт для курьера успешно создан.', 'success')

    return redirect(url_for('admin_routes.admin_panel'))

def generate_order_number(length=6):
    characters = string.ascii_uppercase + string.digits
    return ''.join(random.choice(characters) for _ in range(length))

```

Рис. 7. Функция добавления нового курьера.

Все операции в административной панели сопровождаются всплывающими уведомлениями, которые информируют администратора об успешности или неудаче выполняемой операции. Такой подход способствует удобству использования панели и минимизации ошибок при работе с данными, повышая прозрачность и увеличивая четкость действий администратора.

Таким образом, курьерская и административная панели являются важными составными частями веб-приложения Kitchen Cloud. Они реализуют основные бизнес-процессы ресторана — управление заказами, контроль пользователей и продуктов — и обеспечивают эффективное взаимодействие между всеми участниками системы.

2.3.2 Пользовательский интерфейс

Фронтенд веб-приложения разрабатывался с использованием HTML, CSS и JavaScript. Шаблонизатор Jinja2, встроенный во Flask, обеспечивал генерацию динамического HTML с учетом данных, полученных с сервера. Это позволяло создавать страницы с обновляемыми данными, такими как список блюд, корзина пользователя и история заказов. Встречает нас при переходе на сайт главная страница.

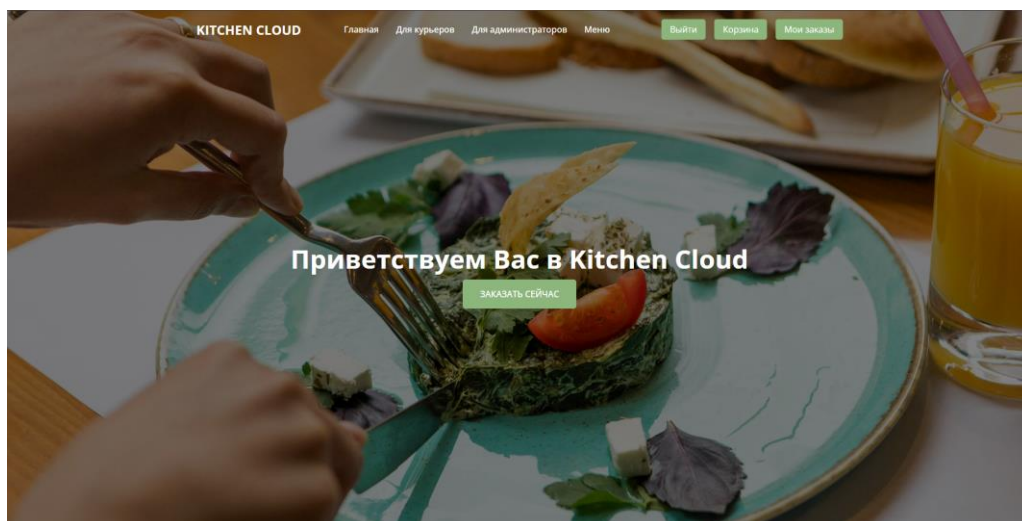


Рис. 8. Главная страница.

Встречает нас простой navbar с ссылками на главную страницу, меню и отдельную панель для курьеров. Кнопки «Для курьеров» и «Для администраторов» появляются только для пользователей которые имеют статусы `is_admin` & `is_courier`.

```
<ul class="nav-links">
  <li><a href="#home">Главная</a></li>
  {% if current_user.is_authenticated and (current_user.is_courier == 'yes' or current_user.is_admin == 'yes') %}
  <li><a href="{{ url_for('courier_routes.courier_panel') }}">Для курьеров</a></li>
  {% endif %}
  {% if current_user.is_authenticated and (current_user.is_admin == 'yes') %}
  <li><a href="{{ url_for('admin_routes.admin_panel') }}">Для администраторов</a></li>
  {% endif %}
  <li><a href="#menu">Меню</a></li>
</ul>
```

Рис. 9. Функция показа скрытых кнопок.

В правой верхней части экрана представлены кнопки 3 кнопки: Кнопка входа/выхода из аккаунта, она меняется в зависимости от того вошел ли в аккаунт человек или нет. Корзина с добавленными в нее позициями и кнопка показывающая все заказы для конкретного пользователя, она появляется только у людей которые вошли в аккаунт.

```
<div class="auth-cart">
  {% if current_user.is_authenticated %}
  <a href="{{ url_for('auth_routes.logout') }}" class="auth-btn">Выйти</a>
  {% else %}
  <a href="{{ url_for('auth_routes.login') }}" class="auth-btn">Войти</a>
  {% endif %}
  <a id="cart-link" href="{{ url_for('main_routes.cart') }}" class="cart-btn">Корзина</a>
  {% if current_user.is_authenticated %}
  <a href="{{ url_for('main_routes.my_orders') }}" class="cart-btn">Мои заказы</a>
  {% endif %}
</div>
```

Рис. 10. Код 3 главных кнопок.

Для визуального оформления используются собственные CSS-стили, которые придают страницам единый стиль, выдержанный в современном минималистичном дизайне. С помощью JavaScript реализовываются интерактивные элементы — например, фильтрация меню по категориям блюд.

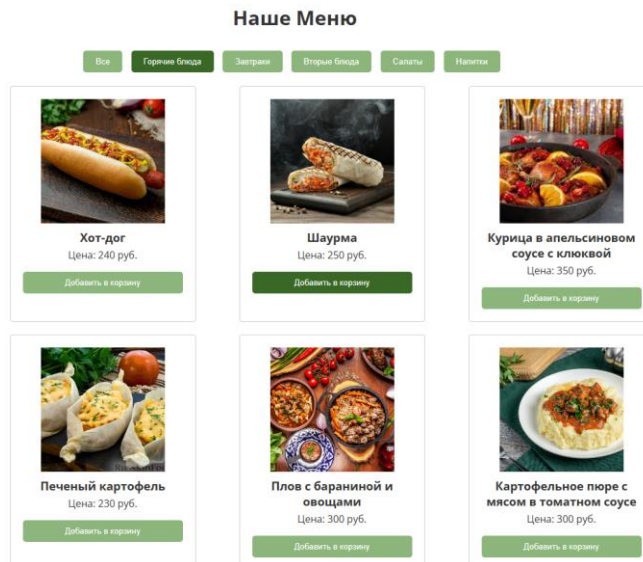


Рис. 11. Фильтрация на сайте

```
function filterProducts(category) {  
  const productList = document.getElementById('product-list');  
  const products = productList.querySelectorAll('.product');  
  
  products.forEach(product => {  
    if (category === 'all' || product.dataset.category === category) {  
      product.style.display = 'block';  
    } else {  
      product.style.display = 'none';  
    }  
  });  
  
  const categoryButtons = document.querySelectorAll('.category-buttons button');  
  categoryButtons.forEach(btn => {  
    if (btn.textContent === category || (category === 'all' && btn.textContent === 'Все')) {  
      btn.classList.add('active-category');  
    } else {  
      btn.classList.remove('active-category');  
    }  
  });  
}
```

Рис. 12. Функция фильтрации по категориям.

Важным элементом интерфейса стала панель курьера. Курьерская панель позволяет просматривать все заказы пользователей и изменять их статусы, обеспечивая прозрачность процесса доставки. Для администраторов была разработана собственная панель управления, позволяющая контролировать пользователей и управлять контентом сайта.

После всех блюд из меню располагается блок «О нас» и блок footer. Блок «О нас» содержит описание ресторана, а в блоке footer содержится ссылки на полезные страницы, социальные сети и навигации по сайту.



Рис. 13. Блок «О нас» и блок footer.

Перейдем к более детальному рассмотрению страниц панели курьера|админа, панели входа, корзины и моих заказов.

Сначала нам нужно авторизоваться на сайте в этом нам поможет кнопка в правом верхнем углу header'а войти. Так как у нас еще нет аккаунта регистрируемся. После регистрации нас автоматически перекидывает на главную страницу с уже войденным аккаунтом.

The image shows a screenshot of a web form for authorization. At the top, there is a link "На главную" and a heading "Авторизация". Below the heading, there are two input fields: "Имя пользователя" and "Пароль". Below the "Пароль" field, there is a green button labeled "Войти". At the bottom of the form, there is a link "Ещё не с нами? Регистрация".

Рис. 14. Авторизация

После авторизации нам доступны все базовые функции для обычных пользователей. Сделаем заказ. Добавив нужные товары в корзину перейдем в нее. В корзине отображаются выбранные позиции. Ниже располагаются поля для ввода номера телефона почты и способа получения. При выборе способа получения «Доставка» у нас появляется еще одно поле для указания адреса доставки.

На главную
Корзина

Список товаров:

Хот-дог	240.0 руб.	Количество: 6
Оливье	200.0 руб.	Количество: 9
Безалкогольные коктейли	250.0 руб.	Количество: 6

Номер телефона:
+79175153710

Почта:
wddasfdffds@yandex.ru

Способ получения:
Доставка

Адрес: Комсомольская 78 кв 6г

Оформить заказ

Итого: 4740.0 руб.

Очистить корзину

Рис. 15. Корзина

После оформления заказа нас перекидывает на страницу просмотра содержимого заказа, с номером и его статусом подтверждающим то, что он был принят в доставку.

Статус заказа

Номер заказа: 15
Статус заказа: Курьер забрал заказ и направляется к вам

Вы заказали:

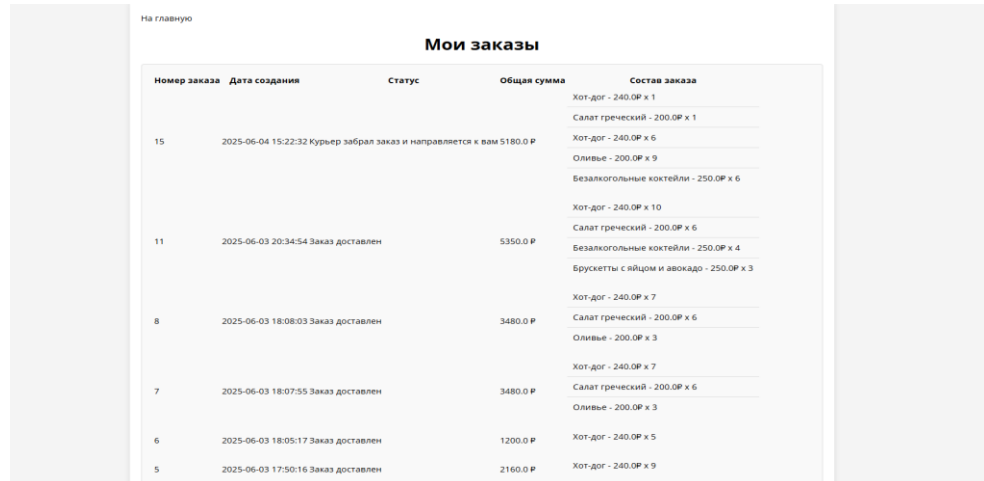
Хот-дог Цена за единицу: 240.0 руб. Количество: 1 Сумма позиции: 240.0 руб.
Салат греческий Цена за единицу: 200.0 руб. Количество: 1 Сумма позиции: 200.0 руб.
Хот-дог Цена за единицу: 240.0 руб. Количество: 6 Сумма позиции: 1440.0 руб.
Оливье Цена за единицу: 200.0 руб. Количество: 9 Сумма позиции: 1800.0 руб.
Безалкогольные коктейли Цена за единицу: 250.0 руб. Количество: 6 Сумма позиции: 1500.0 руб.

Общая сумма: 5180.0 руб.

На главную

Рис. 16. Подтверждение заказа.

Текущий и прошлые заказы можно посмотреть на странице «Мои заказы» вернувшись на главную и нажав на кнопку «Мои заказы». На странице мы видим все заказы с их статусами, временем, id, суммой и всеми позициями входящими в него.



Номер заказа	Дата создания	Статус	Общая сумма	Состав заказа
15	2025-06-04 19:22:32	Курьер забрал заказ и направляется к вам	5180.0 Р	Хот-дог - 240.0Р x 1 Салат греческий - 200.0Р x 1 Хот-дог - 240.0Р x 6 Оливье - 200.0Р x 9 Безалкогольные коктейли - 250.0Р x 6
11	2025-06-03 20:34:54	Заказ доставлен	5350.0 Р	Хот-дог - 240.0Р x 10 Салат греческий - 200.0Р x 6 Безалкогольные коктейли - 250.0Р x 4 Брускетты с яйцом и авокадо - 250.0Р x 3
8	2025-06-03 18:08:03	Заказ доставлен	3480.0 Р	Хот-дог - 240.0Р x 7 Салат греческий - 200.0Р x 6 Оливье - 200.0Р x 3
7	2025-06-03 18:07:55	Заказ доставлен	3480.0 Р	Хот-дог - 240.0Р x 7 Салат греческий - 200.0Р x 6 Оливье - 200.0Р x 3
6	2025-06-03 18:05:17	Заказ доставлен	1200.0 Р	Хот-дог - 240.0Р x 5
5	2025-06-03 17:50:16	Заказ доставлен	2160.0 Р	Хот-дог - 240.0Р x 9

Рис. 17. «Мои заказы».

На этом базовые функции заканчиваются, перейдем к курьерским и администраторским функциям. Для этого войдем как администратор (Для входа на сайт в качестве администратора я использую логин и пароль 123). По центру header'а у нас появилось 2 скрытых кнопки.



Рис. 18. Скрытые кнопки.

Нажмем на кнопку «Для администраторов». Нас перекидывает в панель администратора, на странице располагаются 2 блока: 1 предназначен для добавления нового блюда в меню, второй для добавления нового курьера.

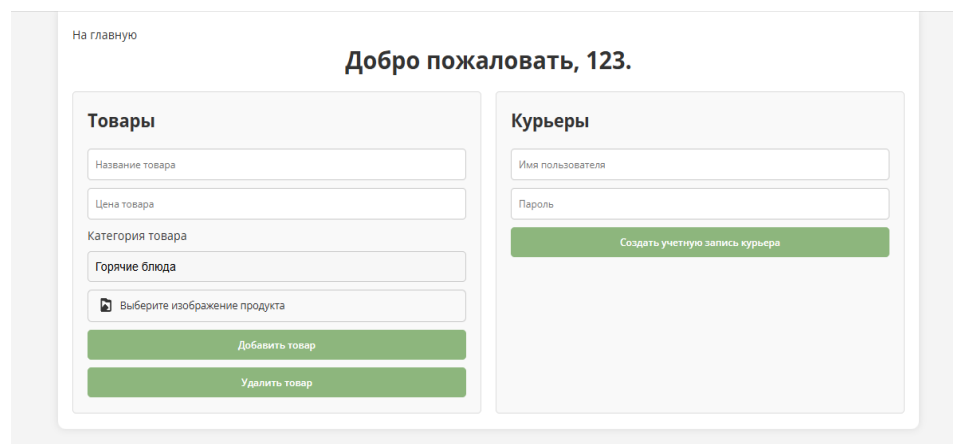


Рис. 19. Панель администратора.

Этот интерфейс позволяет администратору легко управлять всеми аспектами работы системы: создавать и удалять товары, а также управлять курьерами.

Нажмем на кнопку «На главную» в левом верхнем углу экрана и вернемся назад, чтобы перейти на страницу «Для курьеров». На странице мы видим номера заказов, их статусы и поле для изменения этих статусов. Изменим 1 заказ на доставлено.

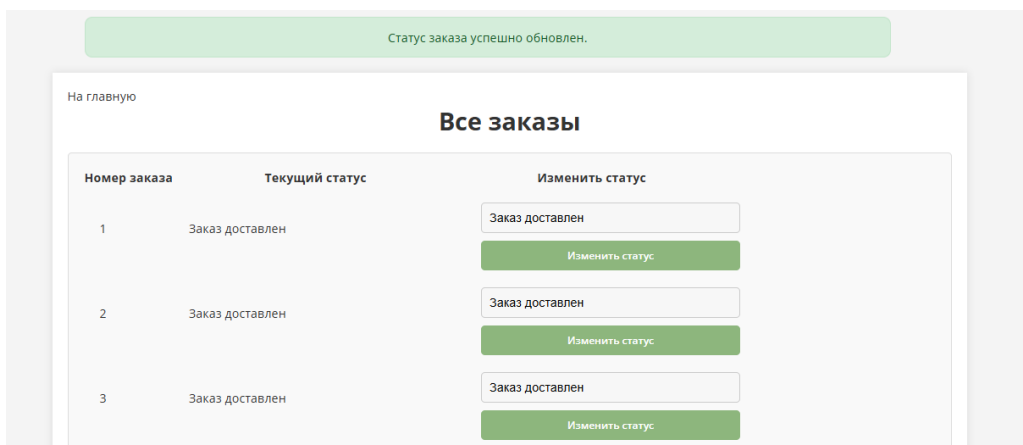


Рис. 20. Курьерская панель.

2.4 Размещение веб-приложения на хостинге

В качестве хостинга для проекта «Kitchen Cloud» был выбран облачный сервис Render, который предоставляет удобную платформу для размещения веб-приложений с поддержкой Python, включая популярный микрофреймворк Flask. Render обеспечивает простой и интуитивно понятный интерфейс для развертывания приложений, поддержку непрерывной интеграции с системами контроля версий, а также масштабирование ресурсов при необходимости.

Размещение веб-приложения на Render осуществляется в несколько этапов:

Первоначально необходимо подготовить проект для размещения на хостинге. Для этого весь проект, включая файлы `app.py`, каталоги `static/` и `templates/`, а также файл зависимостей `requirements.txt`, загружается в репозиторий на GitHub. Использование GitHub обеспечивает удобное хранение кода и его версию.

Для начала работы необходимо зарегистрироваться на платформе Render, используя учетную запись GitHub или электронную почту. После регистрации пользователь попадает в панель управления Render, где можно создать новый сервис для развертывания веб-приложения.

В панели управления Render выбирается опция New Web Service. Сервис запрашивает подключение к GitHub, что позволяет Render автоматически подтягивать обновления кода при каждом коммите в репозитории.

При создании веб-приложения Render запрашивает указание следующих параметров:

Название проекта – указывается произвольное имя для веб-приложения, например, «kitchen-cloud».

Build Command – команда для установки всех зависимостей. В данном случае: `pip install -r requirements.txt`.

Start Command – команда для запуска приложения. Flask-приложения обычно запускаются с помощью Gunicorn: `gunicorn app:app`.

Здесь `app` — имя файла (без `.py`), а второй `app` — переменная, в которой создается экземпляр Flask (`app = Flask(__name__)`).

Render также автоматически определяет среду выполнения Python и устанавливает её, обычно достаточно указать версию Python в файле `runtime.txt` или в настройках (например, 3.10).

После завершения настройки Render самостоятельно выполнит установку всех зависимостей, указанных в `requirements.txt`, и запустит приложение. В случае успешного деплоя в панели управления появится сообщение об успешной активации сервиса, а также будет предоставлен URL-адрес, по которому доступно веб-приложение.

На последнем этапе проверяется корректная работа веб-приложения. Открывая указанный Render URL, можно убедиться в том, что проект корректно отображает страницы и обрабатывает все запросы. При необходимости в Render можно подключить собственный домен для более профессионального внешнего вида сайта.

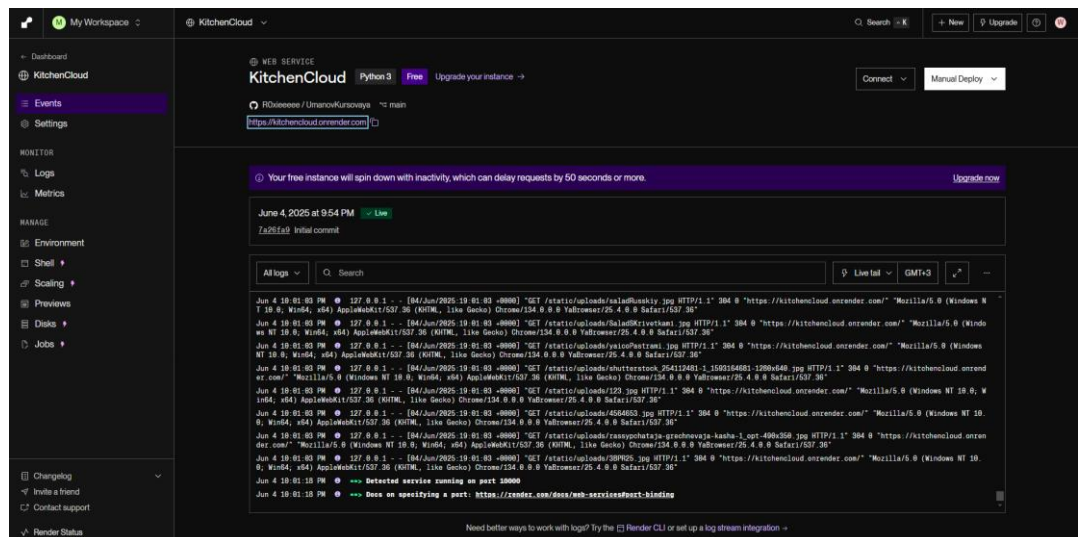


Рис. 21. Работа сайта на Render.

Готово. Наш сайт крутится на хостинге, а заняло это от силы 5 минут, не нужно делать ничего кроме того как создать репозиторий и войти в аккаунт на Render.

После завершения всех этапов развертывания выполняется тестирование работы приложения в боевых условиях. Следует проверить корректность всех основных функций: отображение страниц, авторизация пользователей, оформление заказов и другие операции. Это позволяет убедиться, что все элементы системы работают стабильно и корректно взаимодействуют друг с другом.

Таким образом, размещение веб-приложения *Kitchen Cloud* на хостинге Render позволяет сделать систему доступной для пользователей в режиме 24/7, обеспечивая бесперебойное предоставление услуг ресторана и оптимальную производительность. Данный этап завершает цикл разработки, открывая возможности для дальнейшего масштабирования проекта и внедрения новых функций.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была успешно реализована полноценная веб-система для онлайн-заказа блюд под названием Kitchen Cloud. Проект выполнен с использованием фреймворка Flask, что позволило гибко реализовать функционал и обеспечить масштабируемость. В процессе разработки были проработаны все ключевые компоненты: проектирование структуры веб-приложения, подготовка рабочего окружения, создание баз данных, реализация серверной и клиентской частей, а также настройка интерфейсов для различных ролей пользователей, включая администратора и курьера.

Разработка велась с учетом современных стандартов безопасности и удобства пользователей, что отражается в логичной архитектуре, качественном визуальном оформлении и надежной защите пользовательских данных.

Особое внимание уделялось корректному взаимодействию всех компонентов системы: от ввода данных пользователем до их обработки на сервере и отображения в веб-интерфейсе. Это позволило достичь высокой степени готовности и протестировать работоспособность приложения в условиях боевого хостинга.

Результатом работы стала стабильная веб-система, готовая к развертыванию на реальном хостинге и доступу для конечных пользователей. Проект *Kitchen Cloud* демонстрирует высокий уровень подготовки в области разработки веб-приложений, а также понимание ключевых принципов проектирования современных информационных систем.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Кряжева Е.В., Васина Т.А. Общие подходы к проектированию ВЕБ-приложений // Заметки ученого. – 2021. – № 9-2. – С. 32-36.
2. Стасышин В. Проектирование информационных систем и баз данных. – Электронный ресурс. – Litres, 2022. – URL: <https://www.litres.ru/> (дата обращения: 04.06.2025).
3. Lutz M. Learning Python / М. Lutz. – 5-е изд. – Sebastopol: O'Reilly Media, 2013.
4. Python. Официальная документация [Электронный ресурс] // Python.org. – URL: <https://docs.python.org/3/> (дата обращения: 04.06.2025).
5. Flask. Официальная документация [Электронный ресурс] // Flask.palletsprojects.com. – URL: <https://flask.palletsprojects.com/> (дата обращения: 04.06.2025).
6. Grinberg M. Flask Web Development: Developing Web Applications with Python / М. Grinberg. – 2-е изд. – Sebastopol: O'Reilly Media, 2018.
7. PyCharm. Официальная документация [Электронный ресурс] // JetBrains. – URL: <https://www.jetbrains.com/help/pycharm/> (дата обращения: 04.06.2025).
8. Grinberg M. The Flask Mega-Tutorial [Электронный ресурс] // Blog Miguel Grinberg. – URL: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world> (дата обращения: 04.06.2025).
9. Matthes E. Python Crash Course / E. Matthes. – Электронный ресурс. – URL: <https://nostarch.com/pythoncrashcourse2e> (дата обращения: 04.06.2025).

```
def init_db():
    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            is_admin TEXT NOT NULL,
            is_courier TEXT NOT NULL)''')

        cursor.execute('''CREATE TABLE IF NOT EXISTS orders (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            courier_id INTEGER,
            status TEXT NOT NULL,
            created_at TEXT DEFAULT (datetime('now'))''')

        cursor.execute('''CREATE TABLE IF NOT EXISTS products (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            price TEXT NOT NULL,
            image TEXT,
            category TEXT NOT NULL)''')

        cursor.execute('''CREATE TABLE IF NOT EXISTS order_items (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            order_id TEXT NOT NULL,
            name TEXT,
            price TEXT,
            quantity INTEGER DEFAULT 1)''')
```

Рис. 1. Создание таблиц бд.

```
@auth_routes.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.find_by_username(username)
        if user and check_password_hash(user.password, password):
            login_user(user)
            return redirect(url_for('main_routes.index'))
        else:
            flash('Неверное имя пользователя или пароль.', 'error')
            return render_template('login.html')

@auth_routes.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')
        try:
            with sqlite3.connect(Config.DATABASE) as conn:
                cursor = conn.cursor()
                cursor.execute('''INSERT INTO users (username, password, is_admin, is_courier) VALUES (?, ?, ?, ?)''',
                                (username, hashed_password, "no", "no"))
                conn.commit()
                cursor.execute('SELECT id FROM users WHERE username = ?', (username,))
                user_id = cursor.fetchone()[0]
                new_user = User(id=user_id, username=username, password=hashed_password, is_admin=False, is_courier=False)
                login_user(new_user)
                return redirect(url_for('main_routes.index'))
        except sqlite3.IntegrityError:
            flash('Пользователь с таким именем уже существует.', 'error')
            return redirect(url_for('auth_routes.register'))
    return render_template('register.html')

@auth_routes.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('main_routes.index'))
```

Рис. 2. Страница авторизации и регистрации.

```

@main_routes.route('/checkout', methods=['POST'])
@login_required
def checkout():
    cart_items = session.get('cart', [])
    email = request.form['email']
    phone = request.form['phone']
    address = request.form['address']
    delivery = request.form['delivery_option']
    order_number = generate_order_number()
    status = 'Курьер забрал заказ и направляется к вам'

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('INSERT INTO orders (user_id, status) VALUES (?, ?)',
                        (current_user.id, status))
        order_id = cursor.lastrowid

        for item in cart_items:
            cursor.execute('INSERT INTO order_items (order_id, name, price, quantity) VALUES (?, ?, ?, ?)',
                            (order_id, item['name'], item['price'], item['quantity']))

        conn.commit()

    session.pop('cart', None)
    return redirect(url_for('main_routes.view_order', order_id=order_id))

```

Рис. 3. Создание заказа.

```

@main_routes.route('/cart', methods=['GET', 'POST'])
@login_required
def cart():
    if request.method == 'POST':
        product_name = request.form.get('product_name')
        product_price = float(request.form.get('product_price'))
        product_type = request.form.get('product_type')

        if 'cart' not in session:
            session['cart'] = []

        cart = session['cart']

        found = False
        for item in cart:
            if item['name'] == product_name:
                item['quantity'] += 1
                found = True
                break

        if not found:
            cart.append({
                'name': product_name,
                'price': product_price,
                'product_type': product_type,
                'quantity': 1
            })

        session['cart'] = cart
        session.modified = True
        return redirect(url_for('main_routes.index'))

    cart_items = session.get('cart', [])
    total = calculate_total(cart_items)
    return render_template('cart.html', cart_items=cart_items, total=total)

```

Рис. 4. Корзина.

```

def courier_required(f):
    @login_required
    def decorated_function(*args, **kwargs):
        if not (current_user.is_courier == "yes" or current_user.is_admin == "yes"):
            flash('У вас нет прав для просмотра данной страницы.', 'error')
            return redirect(url_for('auth_routes.login'))
        return f(*args, **kwargs)
    decorated_function.__name__ = f.__name__
    return decorated_function

@courier_routes.route('/courier', methods=['GET', 'POST'])
@courier_required
def courier_panel():
    if request.method == 'POST':
        order_id = request.form['order_id']
        new_status = request.form['status']
        with sqlite3.connect(Config.DATABASE) as conn:
            cursor = conn.cursor()
            cursor.execute('UPDATE orders SET status = ? WHERE id = ?', (new_status, order_id))
            conn.commit()
            flash('Статус заказа успешно обновлен.', 'success')
            return redirect(url_for('courier_routes.courier_panel'))

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('SELECT id, status FROM orders')
        orders = cursor.fetchall()

    return render_template('couriers.html', current_user=current_user, orders=orders)

```

Рис. 5. Панель курьеров.

```

@admin_routes.route('/create_product', methods=['POST'])
@admin_required
def create_product():
    name = request.form['product_name']
    price = request.form['product_price']
    category = request.form['category']
    action = request.form['action4']
    file = request.files.get('product_image')
    filename = None

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath = os.path.join(current_app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        if action == 'create':
            cursor.execute('INSERT INTO products (name, price, category, image) VALUES (?, ?, ?, ?)', (name, price, category, filename))
            conn.commit()
            flash(f"Товар успешно добавлен.", 'success')
        else:
            cursor.execute('DELETE FROM products WHERE name = ? AND price = ? AND category = ?', (name, price, category))
            conn.commit()
            if cursor.rowcount == 0:
                flash(f"Данный товар не найден.", 'error')
            else:
                flash(f"Товар успешно удален", 'success')

    return redirect(url_for('admin_routes.admin_panel'))

```

Рис. 6. Функция добавления и удаления товара.

```

@admin_routes.route('/change_courier_status', methods=['POST'])
@admin_required
def change_courier_status():
    courier_username = request.form['courier_username']
    courier_password = request.form['courier_password']
    hashed_password = generate_password_hash(courier_password, method='pbkdf2:sha256')

    with sqlite3.connect(Config.DATABASE) as conn:
        cursor = conn.cursor()
        cursor.execute('INSERT INTO users (username, password, is_admin, is_courier) VALUES (?, ?, ?, ?)', (courier_username, hashed_password, "no", "yes"))
        conn.commit()
        flash('Аккаунт для курьера успешно создан.', 'success')

    return redirect(url_for('admin_routes.admin_panel'))

def generate_order_number(length=6):
    characters = string.ascii_uppercase + string.digits
    return ''.join(random.choice(characters) for _ in range(length))

```

Рис. 7. Функция добавления нового курьера.

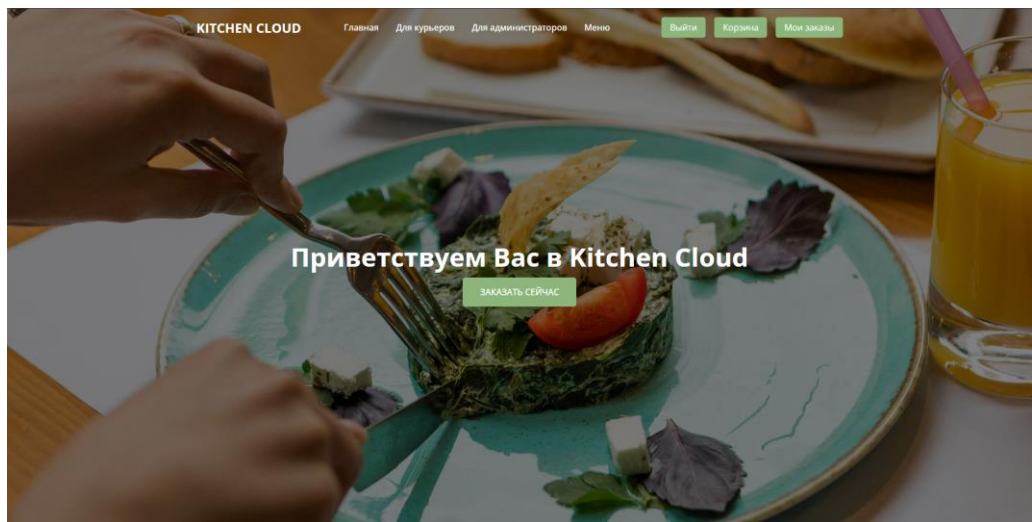


Рис. 8. Главная страница.

```

<ul class="nav-links">
  <li><a href="#home">Главная</a></li>
  {% if current_user.is_authenticated and (current_user.is_courier == 'yes' or current_user.is_admin == 'yes') %}
  <li><a href="{{ url_for('courier_routes.courier_panel') }}">Для курьеров</a></li>
  {% endif %}
  {% if current_user.is_authenticated and (current_user.is_admin == 'yes') %}
  <li><a href="{{ url_for('admin_routes.admin_panel') }}">Для администраторов</a></li>
  {% endif %}
  <li><a href="#menu">Меню</a></li>
</ul>

```

Рис. 9. Функция показа скрытых кнопок.

```

<div class="auth-cart">
  {% if current_user.is_authenticated %}
    <a href="{{ url_for('auth_routes.logout') }}" class="auth-btn">Выйти</a>
  {% else %}
    <a href="{{ url_for('auth_routes.login') }}" class="auth-btn">Войти</a>
  {% endif %}
  <a id="cart-link" href="{{ url_for('main_routes.cart') }}" class="cart-btn">Корзина</a>
  {% if current_user.is_authenticated %}
    <a href="{{ url_for('main_routes.my_orders') }}" class="cart-btn">Мои заказы</a>
  {% endif %}
</div>

```

Рис. 10. Код 3 главных кнопок.

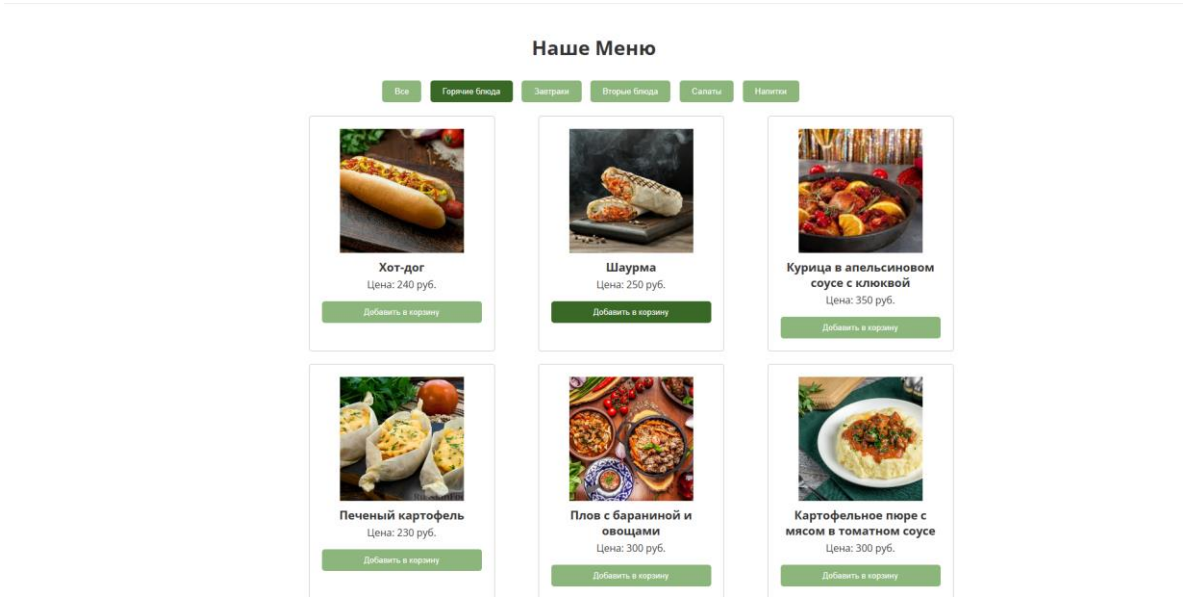


Рис. 11. Фильтрация на сайте

```
function filterProducts(category) {
  const productList = document.getElementById('product-list');
  const products = productList.querySelectorAll('.product');

  products.forEach(product => {
    if (category === 'all' || product.dataset.category === category) {
      product.style.display = 'block';
    } else {
      product.style.display = 'none';
    }
  });

  const categoryButtons = document.querySelectorAll('.category-buttons button');
  categoryButtons.forEach(btn => {
    if (btn.textContent === category || (category === 'all' && btn.textContent === 'Все')) {
      btn.classList.add('active-category');
    } else {
      btn.classList.remove('active-category');
    }
  });
}
```

Рис. 12. Функция фильтрации по категориям.

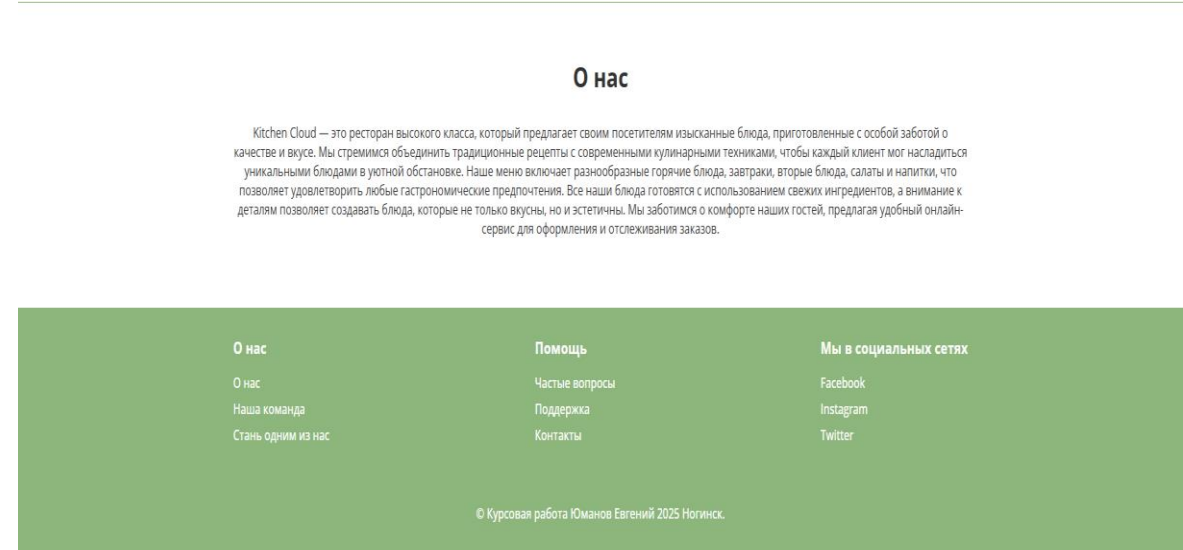


Рис. 13. Блок «О нас» и блок footer.

[На главную](#)

Авторизация

Имя пользователя

Пароль

[Войти](#)

Ещё не с нами? [Регистрация](#)

Рис. 14. Авторизация

[На главную](#)

Корзина

Список товаров:

Хот-дог	240.0 руб.	Количество: <input type="text" value="6"/>
Оливье	200.0 руб.	Количество: <input type="text" value="9"/>
Безалкогольные коктейли	250.0 руб.	Количество: <input type="text" value="6"/>

Номер телефона:

Почта:

Способ получения:

Адрес:

[Оформить заказ](#)

[Очистить корзину](#)

Итого: 4740.0 руб.

Рис. 15. Корзина

Статус заказа

Номер заказа: 15
Статус заказа: Курьер забрал заказ и направляется к вам
Вы заказали:

Хот-дог

Цена за единицу: 240.0 руб.
Количество: 1
Сумма позиции: 240.0 руб.

Салат греческий

Цена за единицу: 200.0 руб.
Количество: 1
Сумма позиции: 200.0 руб.

Хот-дог

Цена за единицу: 240.0 руб.
Количество: 6
Сумма позиции: 1440.0 руб.

Оливье

Цена за единицу: 200.0 руб.
Количество: 9
Сумма позиции: 1800.0 руб.

Безалкогольные коктейли

Цена за единицу: 250.0 руб.
Количество: 6
Сумма позиции: 1500.0 руб.

Общая сумма: 5180.0 руб.

На главную

Рис. 16. Подтверждение заказа.

На главную

Мои заказы

Номер заказа	Дата создания	Статус	Общая сумма	Состав заказа
15	2025-06-04 15:22:32	Курьер забрал заказ и направляется к вам	5180.0 Р	Хот-дог - 240.0Р x 1 Салат греческий - 200.0Р x 1 Хот-дог - 240.0Р x 6 Оливье - 200.0Р x 9 Безалкогольные коктейли - 250.0Р x 6
11	2025-06-03 20:34:54	Заказ доставлен	5350.0 Р	Хот-дог - 240.0Р x 10 Салат греческий - 200.0Р x 6 Безалкогольные коктейли - 250.0Р x 4 Брускетты с яйцом и авокадо - 250.0Р x 3
8	2025-06-03 18:08:03	Заказ доставлен	3480.0 Р	Хот-дог - 240.0Р x 7 Салат греческий - 200.0Р x 6 Оливье - 200.0Р x 3
7	2025-06-03 18:07:55	Заказ доставлен	3480.0 Р	Хот-дог - 240.0Р x 7 Салат греческий - 200.0Р x 6 Оливье - 200.0Р x 3
6	2025-06-03 18:05:17	Заказ доставлен	1200.0 Р	Хот-дог - 240.0Р x 5
5	2025-06-03 17:50:16	Заказ доставлен	2160.0 Р	Хот-дог - 240.0Р x 9

Рис. 17. «Мои заказы»



Рис. 18. Скрытые кнопки.

На главную

Добро пожаловать, 123.

Товары

Название товара

Цена товара

Категория товара

Горячие блюда

Выберите изображение продукта

Добавить товар

Удалить товар

Курьеры

Имя пользователя

Пароль

Создать учетную запись курьера

Рис. 19. Панель администратора.

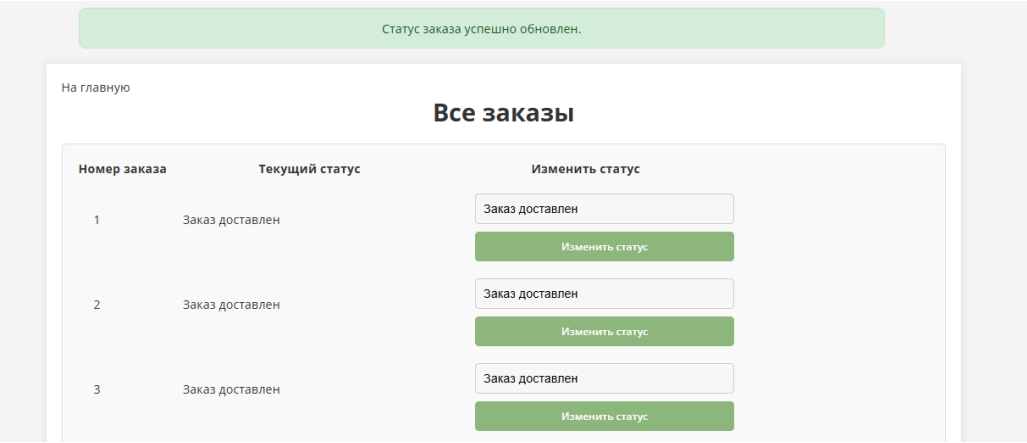


Рис. 20. Курьерская панель.

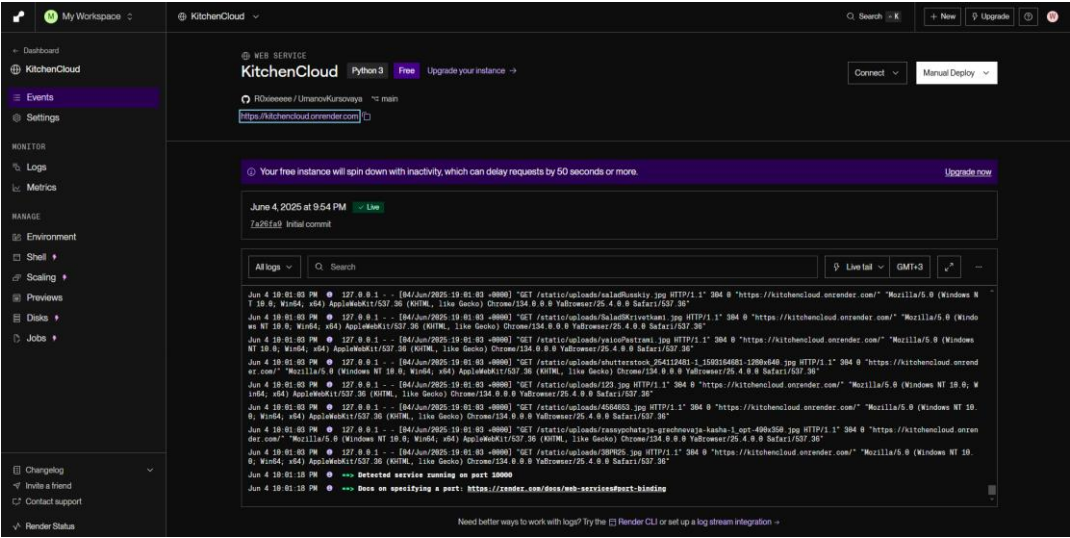


Рис. 21. Работа сайта на Render.