Geekbrains

**Разработка веб-приложения на Java для осуществления дистанционных заказов в организациях общественного питания**(на примере пиццы)

Программа: Разработчик – Программист

Специализация: Разработчик – Веб-разработка на Java

ФИО: Ахмеджанов Надирджан Уткурджанович

Москва

2025

Оглавление

[ВВЕДЕНИЕ 3](#_Toc192518325)

[ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc192518326)

[**1.1 Обзор существующих решений** 8](#_Toc192518327)

[**1.2 Требования к системе** 9](#_Toc192518328)

[**1.3 Выводы по анализу предметной области** 11](#_Toc192518329)

[Глава 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ 12](#_Toc192518330)

[**2.1 Архитектура приложения** 12](#_Toc192518331)

[**2.2 Проектирование базы данных** 14](#_Toc192518332)

[**2.3 Бизнес логика и последовательность в программе** 17](#_Toc192518333)

[**2.4. Системные настройки приложения (application.properties)** 19](#_Toc192518334)

[**2.5 Выводы по проектированию** 21](#_Toc192518335)

[ГЛАВА 3. РАЗРАБОТКА ПРОГРАММЫ 23](#_Toc192518336)

[**3.1 Разработка пакетов приложения** 23](#_Toc192518337)

[**3.2 Тестирование системы** 29](#_Toc192518338)

[**3.3 Юнит-тесты: Описание покрытых компонентов** 29](#_Toc192518339)

[**3.4 Тестирование Контроллеров** 31](#_Toc192518340)

[**3.5 Тестирование Сервисов** 35](#_Toc192518341)

[**3.6 Тестирование репозиториев** 37](#_Toc192518342)

[ЗАКЛЮЧЕНИЕ 42](#_Toc192518343)

[Список использованной литературы 46](#_Toc192518344)

## ВВЕДЕНИЕ

Развитие цифровых технологий и рост интернет-коммерции открыли новые возможности для упрощения повседневных процессов, включая   
покупку продуктов и услуг.

Одним из таких примеров является заказ   
еды через интернет, который становится все более популярным среди потребителей.

Особенно востребованы онлайн-сервисы для доставки пиццы, так как этот продукт является любимым у большинства людей, и его заказ через интернет позволяет сэкономить время и усилия.

При этом конкуренция на рынке доставки пиццы растет, и для того, чтобы привлечь и удержать клиентов, необходимо предложить не только вкусную продукцию, но и качественный, удобный и быстрый сервис.

В связи с этим возникает необходимость разработки веб-приложения, которое бы сочетало в себе простоту интерфейса, высокую производительность и удобство для пользователей.

Так как целевым рынком для внедрения данного продукта является Республика Узбекистан, рассмотрим так же имеющиеся предпосылки для развития данного продукта.

Согласно отчету KPMG, рынок электронной коммерции в Узбекистане достиг 311 млн долл. США, что свидетельствует о почти пятикратном росте   
за последние пять лет.

При этом ожидается, что к концу 2027 года объем розничного рынка Узбекистана достигнет 19,6 млрд долларов США, при этом доля электронной коммерции в общем объеме розничной торговли составит   
от 9% до 11%, что соответствует среднегодовому темпу роста (CAGR) в период с 2022 по 2027 годы от 41,4% до 47,4%.

Всё это объясняется уровнем интернет проникновения, который   
по состоянию на 31 декабря 2023 года составил 77%, что обеспечивает широкую аудиторию для онлайн-торговли.

Кроме того, основой для дальнейшего развития электронной коммерции в Республике Узбекистан является ежегодный рост реальных доходов населения на 10-12%, что в свою очередь подталкивает людей к более удобным и дорогим сервисам, такими как и разрабатываемый нами продукт по дистанционному оформлению доставки еды.

Процесс заказа пиццы через интернет должен быть максимально удобным, быстрым и интуитивно понятным. Современные пользователи ожидают от веб-приложений минимальных усилий при оформлении заказа,   
а также интеграции с платёжными системами для быстрой и безопасной оплаты.

Кроме того, важным аспектом является возможность персонализации заказа, например, добавление различных ингредиентов или изменение размера пиццы, а также возможность выбора дополнительных товаров, таких как напитки или десерты.

В текущей ситуации на рынке онлайн-заказа пиццы существует множество сервисов, однако не все из них предлагают удобный и понятный интерфейс, а также гибкие возможности настройки заказа, что затрудняет процесс использования для конечного потребителя.

Разработка предлагаемого веб-приложения для заказа пиццы на Java имеет целью создание простого и удобного интерфейса, который позволит пользователю без лишних шагов выбрать и заказать пиццу.

Важным элементом является интеграция с внешними системами доставки, что позволит отслеживать статус заказа в реальном времени.

Ключевыми аспектами при разработке такого приложения станут   
высокая производительность, безопасность, поддержка   
многопользовательского доступа и возможность масштабирования решения для различных бизнесов.

**Актуальность исследования** объясняется тем, что с каждым годом увеличивается количество пользователей, предпочитающих заказывать еду через интернет, включая пиццу, а также растет число компаний, предоставляющих такие услуги.

**Цель исследования** — разработка веб-приложения для заказа пиццы, которое будет легко масштабируемым и обеспечит высокий уровень   
удобства для пользователей. Особое внимание уделяется   
безопасности данных пользователей и простоте интеграции с платёжными системами.

Для достижения поставленной цели решаются следующие **задачи:**

1. Проведение анализа существующих веб-приложений для заказа пиццы, выявление их недостатков и слабых мест.
2. Определение требований к функциональности приложения на основе анализа потребностей целевой аудитории.
3. Проектирование архитектуры приложения с использованием фреймворков Java, таких как Spring Boot и Hibernate.
4. Реализация функций выбора пиццы, оставление отзывов, общения с оператором.

**Объект исследования** — процесс создания веб-приложений для заказа пиццы, включающий проектирование, разработку и тестирование программного обеспечения.

**Предмет исследования** — программное обеспечение для онлайн-заказа пиццы, включая систему управления заказами.

**Методы исследования** включают:

* Анализ существующих решений на рынке онлайн-заказа пиццы, выявление их проблем и недостатков. Причём изучение рынков будет осуществляется как на примере Российской Федерации, так и Республики Узбекистан.
* Применение методов объектно-ориентированного проектирования для создания структуры приложения.
* Использование Java-технологий, таких как Spring Boot и Hibernate, для разработки серверной части приложения.
* Осуществление тестирования разработанного кода, как юнит тестов, так и интеграционных тестов.

Хотя в название в виде примера указывается онлайн заказ именно Пиццы, но на практике этот программный продукт можно будет применить к любому блюду либо товару (цветы, книги и т.д.).

Практическая значимость разработки заключается в создании веб-приложения, которое может быть использовано как для личного пользования, так и для коммерческого использования в малых и средних компаниях, предоставляющих услуги доставки пиццы.

Приложение будет легко масштабируемым, что позволит адаптировать его под различные запросы и масштабы бизнеса.

Реализованное решение будет доступно для использования в любых организациях, желающих улучшить процесс заказа пиццы и повысить удовлетворённость своих клиентов.

## ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

### **1.1 Обзор существующих решений**

На сегодняшний день существует множество веб-приложений для онлайн-заказов, включая системы для заказов в ресторанах и организациях общественного питания. Вместе с тем, рассмотрим общие для узбекского и российского рынка:

1. **Uber Eats** – популярная платформа для заказа еды с доставкой, которая связывает пользователей с ресторанами. Приложение предоставляет удобный интерфейс для выбора блюд, оформления заказа и отслеживания доставки.

Преимущества:

* + Удобный и интуитивно понятный интерфейс.
  + Широкий выбор ресторанов и блюд.
  + Поддержка нескольких способов оплаты.

Недостатки:

* + Высокие комиссии для ресторанов.
  + Ограниченная возможность персонализации заказа для клиентов.
  + Платформа зависит от качества и скорости доставки партнеров.

1. **Яндекс.Еда** – российский сервис, предоставляющий пользователям возможность заказывать еду из ресторанов с доставкой. Он также включает опцию для заказов на вынос.

Преимущества:

* + Широкая сеть ресторанов по всей стране.
  + Простой процесс оформления заказа.
  + Встроенная система лояльности и скидок.

Недостатки:

* + Проблемы с доставкой в регионах.
  + Ограниченные возможности для настройки пользовательского интерфейса для ресторанов.
  + Платформа не всегда предоставляет точную информацию о времени доставки.

Кроме того, в Узбекистане в сторону «суперапов» двигаются несколько финтех решений, таких как **Клик, Пайнет и Узум**, в мобильных приложениях которых существует возможность дистанционного заказа еды и напитков   
из объектов общественного питания.

Анализ существующих решений позволяет выделить несколько основных факторов: удобство интерфейса, возможности персонализации заказов, а также проблемы с доставкой. Для успешной разработки веб-приложения для дистанционных заказов в общественном питании важно учитывать эти аспекты и предоставить пользователям удобный, надежный и персонализированный сервис.

### **1.2 Требования к системе**

Разработка системы для дистанционных заказов в организациях общественного питания требует внимательного подхода к функциональным   
и нефункциональным требованиям.

Функциональные требования:

1. Регистрация и авторизация пользователей:
   * Пользователи должны иметь возможность зарегистрироваться и войти в систему, используя учетные записи (например, через номер телефона).
2. Просмотр меню и выбор блюд:
   * Система должна предоставлять пользователям возможность просматривать меню с возможностью фильтрации по категориям и сортировки по популярности или цене.
3. Оформление заказа:
   * После выбора блюд пользователь должен иметь возможность оформить заказ.
   * В дальнейшем возможна поддержка опций, таких как выбор времени доставки и добавление комментариев к заказу.
4. Общение с оператором:
   * Система должна поддерживать возможность общения в онлайн режиме для облегчения заказа клиенту путём своевременного предоставления ответов на возникающие вопросы.
5. Оставление отзывов:
   * Пользователи должны иметь возможность оставить отзывы о продукте.
6. Административный интерфейс:
   * Администраторы должны иметь возможность управлять меню, ценами и отслеживать статусы заказов.

Нефункциональные требования:

1. Производительность:
   * Система должна обеспечивать быстрый отклик при оформлении заказов и обновлении статуса.
   * Время отклика сервера не должно превышать 3 секунд.
2. Безопасность:
   * Все персональные данные пользователей должны быть защищены с использованием современных методов шифрования.
   * Необходимо использовать HTTPS для передачи данных.
3. Масштабируемость:
   * Приложение должно быть масштабируемым, с возможностью увеличения нагрузки при росте числа пользователей.

### **1.3 Выводы по анализу предметной области**

Изучение существующих решений показало, что для создания успешной системы дистанционных заказов в общественном питании важно учитывать удобство осуществления заказа, возможность общения с оператором в процессе осуществления заказа, а также доступ к потребительскому опыту предыдущих пользователей (фидбек).

Основными требованиями к системе являются:

* Функциональные: регистрация пользователей, просмотр меню, оформление заказа, общение с оператором, прочтение и оставление отзывов о продукте и сервисе;
* Нефункциональные: высокая производительность, безопасность данных и масштабируемость системы.

## Глава 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

### **2.1 Архитектура приложения**

Архитектура разрабатываемого веб-приложения для осуществления дистанционных заказов в организациях общественного питания основана   
на модели **MVC** (Model-View-Controller). Данная модель на сегодняшний день является самым широко используемым методом архитектуры кода.

**Модель (Model)**

Данная часть кода отвечает за управление данными, выполнение бизнес-логики и взаимодействие с базой данных.

Исходя из поставленных задач, в нашей программе существуют следующие сущности:

* **Product (Пица)** — описывает пицы, доступные для заказа в меню, включая название, описание, цену, доступность, фото.
* **Order (Заказ)** — описывает заказ пользователя в организации общественного питания, включая информацию о выбранных блюдах, количестве, цене.
* **Message (Сообщение)** — включает информацию о сообщениях между клиентом и оператором в процессе заказа пицы.
* **History (История)** — история заказов клиентов, осуществленных через сервис. Необходима для маркетинговых акций, упрощения заказов, подсказок и т.д.
* **Feedback (отзыв)** — отзыв клиентов, оставленных по результатам опыта использования сервиса и заказа пицы.

Для взаимодействия с базой данных используется **Spring Data JPA**, который минимизирует необходимость написания SQL-запросов вручную. Репозитории, основанные на Spring Data JPA, обеспечивают удобную работу с данными, включая операции CRUD (создание, чтение, обновление и удаление).

**Контроллеры (Controller)**

Контроллеры отвечают за обработку HTTP-запросов, взаимодействие с пользователями и передачу данных в представление (например, в формате JSON).

Так, программой предусмотрены следующие контроллеры:

* **ProductController** — эндпоинты для осуществления заказов, просмотра заказов и т.д.
* **MessagesController** — предоставляет эндпоинты для общения между оператором и клиентом.
* **AdminController** — позволяет управлять доступными блюдами: добавление новых блюд, обновление информации, удаление блюд из меню и получение списка блюд.
* **FeedbackController** — эндпоинты для работы с отзывами (оставление, чтение отзывов)

**Технологии, используемые в приложении**

1. **Spring Boot**

Spring Boot является основным фреймворком для разработки серверной части приложения. Он предоставляет удобные средства для настройки и развертывания, а также интеграцию с различными библиотеками, что ускоряет процесс разработки.

1. **Lombok**  
   **Lombok** используется для автоматической генерации часто повторяющихся методов, таких как геттеры, сеттеры, конструкторы, что упрощает код и делает его более читабельным.
2. **Spring Security**

Для обеспечения безопасности приложения используется **Spring Security**. Он отвечает за аутентификацию пользователей и управление их правами доступа, что важно для ограничения доступа к определённым функциям системы в зависимости от ролей пользователя (например, клиент, администратор, сотрудник).

**Обработка ошибок**

Для обработки ошибок и исключений в системе существует отдельный пакет “Exception”, что позволяет централизованно управлять ошибками и возвращать понятные сообщения для пользователей и разработчиков.

### **2.2 Проектирование базы данных**

Для разработки системы дистанционных заказов в организациях общественного питания была спроектирована база данных, которая должна поддерживать ключевые сущности приложения Product (Пицца), Order (Заказ), Message (Сообщение), History (История), Feedback (отзыв).

**Описание сущностей**

1. **Product (Пица)**:
   * Хранит информацию о конкретных блюдах в меню.
   * Основные атрибуты:
     + id: уникальный идентификатор блюда.
     + name: название блюда (например, “Пицца Маргарита").
     + price: цена пицы.
     + description: описание пицы (состав ингредиентов).
     + category: категория пицы (мясные, сырные, комбо и т.д.).
     + quantityAvailable – количество пицы, доступной для продажи;
     + img – изображение пицы (фото реального продукта).
2. **Order (Заказ)**:
   * Хранит данные о заказах.
   * Основные атрибуты:
     + orderId: уникальный идентификатор заказа.
     + userNumber: идентификатор пользователя (клиента), сделавшего заказ (предполагается номер телефона).
     + order\_date: дата и время создания заказа.
     + productId: идентификатор пиццы.
     + orderQuantity: количество заказанного товара.
3. **Message (Сообщение):** 
   * Описывает диалог между клиентом и оператором.
   * Основные атрибуты:
     + id - уникальный идентификатор переписки.
     + title: заголовок сообщения.
     + user\_text: nекст сообщения от пользователя.
     + operatorEmail: Email оператора (может быть использован также ник).
     + operatorText: текст ответа от администратора.
     + closed: Флаг, указывающий на закрытие сообщения.
4. **History (История):** 
   * Описывает историю заказа пользователя, имеющего место быть в прошлом.
   * Основные атрибуты:
     + userNumber: идентификатор пользователя (клиента), сделавшего заказ (предполагается номер телефона).
     + order\_date: дата и время создания заказа.
     + name: название блюда (например, “Пицца Маргарита").
     + orderQuantity: количество заказанного товара.
     + description: описание пиццы (состав ингредиентов).
     + img – изображение пиццы (фото реального продукта).
5. **Feedback (Отзывы/комментарии):** 
   * Описывает историю заказа пользователя, имеющего место быть в прошлом.
   * Основные атрибуты:
     + id: Уникальный идентификатор фидбека.
     + userNumber: Номер телефона пользователя, оставившего отзыв.
     + date: дата создания отзыва.
     + rating: Рейтинг, присвоенный продукту в отзыве.
     + productId: Идентификатор продукта, к которому относится отзыв.
     + feedbackDescription: Текст отзыва

### **2.3 Бизнес логика и последовательность в программе**

Бизнес логика данной программы состоит из следующих возможностей:

* 1. Клиент дистанционно выбирает из каталога блюд (в нашем случае пиццы) понравившиеся ему пиццу и указывает также количество заказываемого товара.
  2. Если количество товара и наличие пиццы соответствует заказу, то принимается заказ. По номеру телефона запрашиваются остальные детали заказа – способ оплаты, место и время доставки, другие предпочтения.
  3. Клиент в момент заказа может начать общаться с оператором по вопросам, его интересующим, таким как способы оплаты, среднее время доставки, качество используемых ингредиентов и т.д.
  4. После получения доставки, клиент также может оставить комментарии с выражением благодарности, либо предложения по усовершенствованию тех иных аспектов. Возможно оставить также негативный отклик о качестве товара либо других услугах.

Диаграмма последовательности показывает, как различные компоненты системы взаимодействуют друг с другом при выполнении операций. В нашем случае это будет взаимодействие между клиентом (пользователем), контроллером, сервисом и репозиторием, а также с базой данных.

Рассмотрим основной сценарий: **Создание заказа**

1. **Клиент** (пользователь) отправляет **POST-запрос** на "/secure/order", содержащим данные нового заказа.
2. **Контроллер** OrderController принимает запрос и вызывает метод createOrder у сервиса **OrderService**
   * Этот метод создает новый объект заказа и передает его в сервис для дальнейшей обработки.
3. В **OrderService**:
   * Происходит создание объекта заказа на основе данных, полученных из запроса.
   * Для каждой позиции в заказе вызываются репозитории (OrderRepository), чтобы получить информацию о каждой пице по АйДи.
4. Далее, OrderService передает объект заказа в **OrderRepository** для сохранения в базе данных.
5. Репозиторий (OrderRepository) сохраняет заказ в базе данных и возвращает сохраненный экземпляр заказа.
6. OrderService возвращает сохраненный заказ обратно в контроллер.
7. Контроллер преобразует объект и отправляет его клиенту.

Аналогичная схема работает и в случае исользования GET- запросов, когда необходимо проверить, заказан ли продукт пользователем и для получения текущего количества заказов пользователя.

**Основные преимущества подхода:**

1. **Модульность**: Разделение слоев приложения (контроллер, сервис, репозиторий) позволяет поддерживать и тестировать систему с минимальными усилиями. Каждый слой отвечает за свою часть работы, что упрощает добавление новых функций.
2. **Масштабируемость**: Система спроектирована так, что новые сущности и функциональные возможности можно добавлять без значительных изменений в уже существующий код. Например, для добавления новых типов блюд или способов оплаты достаточно добавить новые сущности и расширить сервисы.
3. **Четкость взаимодействия**: Диаграмма последовательности ясно отображает, как различные компоненты приложения взаимодействуют, что важно для понимания архитектуры системы и упрощает ее дальнейшее развитие.

### **2.4. Системные настройки приложения (application.properties)**

Файл **application.properties** служит основным инструментом конфигурации приложения в Spring Boot. Он позволяет централизованно управлять настройками, делая приложение гибким, легко настраиваемым   
и удобным в поддержке.

Для проекта "Система дистанционных заказов в организациях общественного питания" файл application.properties выполняет следующие функции:

**1. Подключение к базе данных MySQL** (возможно и H2)

* spring.datasource.url: Это URL подключения к базе данных MySQL. Он указывает, что приложение будет подключаться к базе данных shop\_db, которая находится на сервере localhost. Далее указаны параметры для:
  + useSSL=false: Отключение использования SSL-соединения, что может быть полезно в случае разработки на локальном компьютере.
  + useUnicode=yes: Указывает, что приложение будет поддерживать Unicode (важно для работы с текстами на разных языках).
  + characterEncoding=UTF-8: Устанавливает кодировку UTF-8 для передачи данных.
  + allowPublicKeyRetrieval=true: Разрешает получение открытого ключа для аутентификации, что важно для новых версий MySQL, где это требуется для безопасного подключения.
  + serverTimezone=UTC: Устанавливает часовой пояс для подключения, в данном случае используется UTC.
* spring.datasource.driver-class-name: Указывает на класс драйвера для MySQL (com.mysql.cj.jdbc.Driver), который будет использоваться для подключения к базе данных.

spring.datasource.username=root

spring.datasource.password=12345678

* spring.datasource.username: Имя пользователя для подключения к базе данных (в данном случае это root).
* spring.datasource.password: Пароль для указанного пользователя   
  (в данном случае это 12345678).
* spring.jpa.properties.hibernate.dialect: Это настройка для Hibernate, которая указывает, какой диалект SQL использовать для работы с базой данных MySQL. В данном случае используется диалект для MySQL 8.

**2**. Настройки REST API

spring.data.rest.base-path=/api

* spring.data.rest.base-path: Указывает базовый путь для REST API, который будет обслуживать приложение. В данном случае все REST-эндпоинты будут начинаться с /api, например, /api/users, /api/products и так далее.

3. Настройки для Okta OAuth2

* okta.oauth2.client-id: Это уникальный идентификатор клиента, который предоставляется Okta. Он используется для аутентификации приложения в системе Okta при использовании OAuth2. В данном случае это ID вашего клиента Okta.
* okta.oauth2.issuer: Это URL, указывающий на OAuth2 провайдера Okta, который будет использоваться для аутентификации.

### **2.5 Выводы по проектированию**

Процесс проектирования системы дистанционных заказов   
в организациях общественного питания включает несколько ключевых аспектов, которые направлены на создание гибкой и надежной системы:

1. **Архитектура приложения:**
   * Применение архитектуры типа MVC (Model-View-Controller) обеспечило разделение бизнес-логики, представления и доступа к данным. Это улучшило масштабируемость и упростило поддержку системы.
   * Выбранные технологии, такие как Spring Boot, Lombok и Swagger, предоставляют мощные инструменты для ускоренной разработки, улучшения тестирования и удобства взаимодействия с API.
2. **Проектирование базы данных:**
   * Структура базы данных полностью покрывает требования системы, обеспечивая целостность данных и поддержку различных функциональных возможностей.
   * Реляционная модель базы данных с внешними ключами и индексами позволяет эффективно работать с большими объемами данных, улучшая производительность и упрощая анализ.
3. **Конфигурация приложения:**
   * Использование файла application.properties позволило централизованно настроить параметры приложения, такие как подключение   
     к базе данных и интеграция с внешними сервисами.
4. **Гибкость и масштабируемость:**
   * Проект был спроектирован с учетом возможного роста функционала и увеличения числа пользователей, что позволяет легко масштабировать систему.
   * Модульная структура и продуманный дизайн облегчают добавление новых функций, таких как интеграция с дополнительными системами оплаты или расширение функционала заказа.

Таким образом, проектирование системы управления заказами   
в организациях общественного питания основывается на лучших практиках разработки и проектирования, обеспечивая гибкость, надежность и высокое качество системы.

## ГЛАВА 3. РАЗРАБОТКА ПРОГРАММЫ

### **3.1 Разработка пакетов приложения**

Разработка backend-части нашего веб-приложения для дистанционных заказов в организациях общественного питания основана на архитектурном подходе REST.

Этот подход позволяет обеспечить простоту, гибкость   
и масштабируемость взаимодействия между клиентами и сервером, что особенно важно для приложений, которые обслуживают большое количество пользователей и обеспечивают высокую доступность сервисов.

Основной задачей backend-части является предоставление RESTful API для выполнения CRUD-операций (создание, чтение, обновление   
и удаление) над сущностями **Order** (заказ), **Product** (каталог пиц), **Message** (сообщения) и **Feedback** (отзывы).

**Основные компоненты реализации REST API:**

1. **Контроллеры:** Контроллеры обрабатывают HTTP-запросы, направляют их к соответствующим сервисам и возвращают клиентам нужные данные. Каждый контроллер реализует спецификацию API.
   * **OrderController** управляет операциями с заказами, включая:
     + Создание нового заказа, поступившего от клиента.
     + Получение списка всех текущих заказов клиента.
     + Получение заказа по ID для его подробного просмотра.
   * **AdminController** отвечает за операции с меню, в частности:
     + Добавление новых продуктов, в нашем случае Пиццы в каталог Пицц.
     + Удаление из каталога продукции Пиццы;
     + Увеличение количества доступных пицц для продажи на складе обслуживающих точек;
     + Уменьшение количества доступных пицц для продажи на складе обслуживающих точек.
   * **MessageController** управляет обменом сообщений между клиентом  
     и оператором предприятия в процессе заказа Пиццы, в частности:
     + Отправка сообщения пользователем в процессе заказа Пиццы или просмотра каталога продукции;
     + Ответ оператора на сообщение клиента по сути задаваемого вопроса.
   * **FeedbackController** управляет доступом к комментариям, отзывам, обмену мнениями и т.д., в частности:
     + Размещение сообщения/отзыва/отклика пользователем о продукции либо сервисе;
     + Получение всех оставленных отзывов и откликов пользователя по его номеру телефона

Все контроллеры снабжены аннотациями OpenAPI для автоматического создания спецификаций Swagger. Это упрощает процесс документирования API, позволяет сторонним разработчикам быстро ознакомиться с доступными методами и их параметрами, а также использовать сгенерированную документацию для тестирования API.

1. **Сервисный слой:** Сервисный слой инкапсулирует бизнес-логику приложения и взаимодействует с репозиториями. Каждый сервис предоставляет методы для выполнения операций с сущностями приложения. Этот слой позволяет легко модифицировать логику без необходимости изменения контроллеров или репозиториев.
   * **OrderService** управляет созданием и управлением заказами. В сервисе реализована логика, которая также уменьшает количество товаров на складе исходя из количества заказанного товара. Кроме того, разработаны методы по определению, есть ли заказ у пользователя и список текущих заказов пользователя (клиента) черех данную программу.
   * **AdminService** (или ProductService) обрабатывает запросы, связанные с позициями меню. В сервисе реализована логика для работы с меню (каталогом блюд), а также методы для регулировки количества доступнқх для заказа товаров на складе. При этом при удалении Продукта из базы данных, также удаляются все отзывы о нём и текущие и прошлые заказы.
   * **MessageService** выполняет операции c сообщениями, которые отправляются между клиентом и оператором в процессе заказа или ознакомления с меню.
   * **FeedbackService** выполняет операции c добавлениями отзывов   
     и откликов, а также проверяет имеется ли отзыв конкретного клиента   
     о конкретном товаре.
2. **Репозитории:** для работы с базой данных используются интерфейсы JpaRepository, которые предоставляют готовую реализацию для базовых операций (например, сохранение, удаление, обновление). Помимо этого, добавлены пользовательские запросы с использованием аннотаций @Query для выполнения более сложных операций. Репозитории позволяют эффективно работать с большими объемами данных, что критически важно для работы системы, обслуживающей большое количество заказов.

В нашей программе существуют следующие репозитории:

**OrderRepository** – интерфейс для работы с базой данных заказов, осуществленных пользователем;

**ProductRepository** - интерфейс для работы с базой данных каталога блюд (видов Пиццы);

**MessageRepository** - интерфейс для работы с базой данных сообщений, направляемых между оператором и клиентом;

**FeedbackRepository -** интерфейс для работы с базой данных отзывов, откликов и других проявлений субъективного отношения к той или иной продукции;

**HistoryRepository** - интерфейс для работы с базой данных заказов, осуществлённых конкретным пользователем.

**Обработка исключений:** В системе реализован механизм обособленной обработки каждого вида исключений. Кодом перехватываются и обрабатываются различные исключения, такие как:

* + Ошибки при поиске несуществующих записей (например, попытка получить заказ или позицию меню, которой нет в базе данных).
  + Ошибки, связанные с некорректными данными (например, если клиент пытается разместить заказ без указания адреса доставки или если выбраны недоступные блюда).

Обработка ошибок позволяет централизованно управлять ситуациями, которые могут возникать в процессе работы с API, и возвращать клиентам понятные и информативные сообщения.

Так в нашем коде используются следующие виды исключения:

- **FeedbackAlreadyExistsException** — это исключение, которое выбрасывается, когда пользователь пытается оставить отзыв о продукте, о котором он уже оставлял отзыв. Это может означать, что система предотвращает дублирование отзывов, обеспечивая, чтобы каждое мнение было уникальным. Такой механизм помогает поддерживать чистоту данных и предотвращает спам или повторные отзывы на одну и ту же тему.

- **ForbiddenException** — это исключение возникает, когда пользователь пытается выполнить операцию или доступиться к ресурсу, для которого у него нет соответствующих прав.

Например, если обычный пользователь пытается получить доступ к администраторской панели или изменить настройки, которые доступны только администраторам, система выбросит это исключение, сигнализируя о том, что операция запрещена для текущего пользователя.

- **InsufficientStockException** — это исключение выбрасывается в случае, если клиент пытается заказать товар в количестве, которое превышает доступный остаток на складе. Система проверяет количество товара перед оформлением заказа, и если на складе нет достаточного количества товара, чтобы выполнить запрос, выбрасывается это исключение, информируя пользователя о нехватке товара.

- **MessageNotFoundException** — это исключение появляется, когда система не может найти сообщение, которое пользователь пытается получить или отправить. Это может случиться, если ссылка на сообщение была повреждена, сообщение было удалено или никогда не существовало.

Это исключение помогает информировать систему о невозможности найти конкретное сообщение и предотвращать дальнейшие ошибки.

- **ProductAlreadyOrderedException** — это исключение выбрасывается, когда пользователь пытается заказать тот же товар, который уже был заказан ранее. Например, если в корзине уже есть определенный товар, и клиент пытается добавить его еще раз, система предотвращает это с помощью данного исключения, что помогает избежать дублирования и лишних записей в заказах.

- **ProductNotFoundException** — это исключение возникает, когда система не может найти продукт, который был запрашиваемым или искомым пользователем. Это может происходить, например, когда товар был удален из базы данных, или был неверно указан идентификатор продукта. Это исключение гарантирует, что пользователь получит корректную информацию, если запрашиваемый товар не существует в системе

**Преимущества реализованного API:**

1. **Стандартизация:** Все методы API реализованы согласно REST-принципам, что обеспечивает предсказуемое и удобное взаимодействие с системой. Это облегчает интеграцию с внешними системами, такими как CRM, платежные шлюзы или приложения для доставки.
2. **Документированность:** Дальнейшая интеграция с Swagger UI позволяет не только автоматизировать процесс создания документации, но и предоставляет пользователям и разработчикам доступ к функционалу API. Это значительно ускоряет разработку и тестирование, а также улучшает взаимодействие между командами.
3. **Расширяемость:** Архитектура API легко расширяется, позволяя добавлять новые функции, такие как обработка различных типов оплаты (картами, через электронные кошельки и другие методы) или поддержка языков и валют.

Таким образом, backend-часть системы была спроектирована и реализована с упором на масштабируемость, удобство использования и эффективную работу с клиентами. Все это позволяет обеспечить стабильное и быстрое выполнение дистанционных заказов, что критически важно для организаций общественного питания, работающих в условиях онлайн-сервисов.

### **3.2 Тестирование системы**

Тестирование является неотъемлемой частью разработки, обеспечивающей стабильность, надежность и соответствие системы требованиям. В процессе тестирования системы были разработаны юнит- и интеграционные тесты, охватывающие ключевые компоненты, что позволяет убедиться в правильности работы всех частей приложения. Данная технология уже на этапе разработки помогла исправить ошибки, которые были выявлены по результатам тестирования.

### **3.3 Юнит-тесты: Описание покрытых компонентов**

Юнит-тесты являются неотъемлемой частью процесса разработки и нацелены на проверку корректности работы отдельных компонентов системы, таких как контроллеры, сервисы и репозитории.

Основная цель юнит-тестирования заключается в том, чтобы убедиться, что каждая часть системы функционирует как ожидается в изоляции, без зависимости от внешних компонентов.

Это позволяет выявить ошибки на ранних стадиях разработки и минимизировать их влияние на систему в целом. В рамках юнит-тестирования используется принцип тестирования отдельных единиц кода, что обеспечивает точность и локализацию ошибок.

Для тестирования контроллеров, таких как AdminController и OrderController, применялись моки с использованием библиотеки **Mockito**. Контроллеры тестируются с помощью аннотации @WebMvcTest, которая позволяет изолировать контроллеры от других компонентов системы, таких как сервисы и репозитории.

В тестах проверяется корректность HTTP-запросов (GET, POST, PUT, DELETE), а также взаимодействие контроллеров с сервисами через mock-объекты. Мокируются такие зависимости, как сервисы, например, AdminService, для имитации поведения этих компонентов без необходимости выполнения реальных операций с базой данных или сторонними сервисами.

Для проверки прав доступа, таких как в методах для добавления или удаления продукта, используется мокация JWT токенов с помощью библиотеки **Spring Security Test**. Также в тестах учитываются различные сценарии с разными кодами состояния HTTP (например, 200 OK, 403 Forbidden, 500 Internal Server Error).

Для сервисов, таких как AdminService, использовалась мокация репозиториев и внешних зависимостей. При тестировании сервисов акцент делается на проверку бизнес-логики, например, на добавление, обновление или удаление продуктов (пицц), а также на проверку обработки исключений. В этом случае использовалась аннотация @Service для создания экземпляра сервиса   
и мокация зависимостей через **Mockito**.

Это позволяет протестировать поведение методов в условиях различных исключений, таких как отсутствие продукта в базе данных, что симулируется   
с помощью метода Optional.empty() для возвращаемого значения репозитория.

При этом важно протестировать не только успешные сценарии, но и негативные — например, когда происходит попытка уменьшить количество продукта до отрицательного значения.

Репозитории, такие как ProductRepository, тестировались   
с использованием аннотации @DataJpaTest, которая запускает тесты   
с встраиваемой базой данных H2, предоставляя изолированную среду для работы с JPA.

В этом случае проверяется корректность работы запросов, таких как сохранение, обновление и удаление сущностей, а также сложные запросы   
с фильтрацией по нескольким полям.

Для тестирования репозиториев использовались аннотации **@Query**   
для создания настраиваемых запросов в JPQL (например, для поиска продуктов по частям названия или категории). Тесты проверяли корректность работы метода findByNameContaining, который осуществляет поиск продуктов по части названия, и метода findProductByProductIds, который выполняет запрос   
с использованием списка идентификаторов продуктов.

В тестах также проверялась корректность работы транзакций, обеспечиваемых Spring, и изоляция данных между различными тестами, что позволяло исключить влияние одного теста на другие.

Таким образом, юнит-тесты охватывают не только проверку корректности бизнес-логики, но и изоляцию компонентов с помощью mock-объектов, мокацию зависимостей, а также проверки работы с базой данных. Все эти методы в совокупности помогают удостовериться в том, что каждый компонент системы работает должным образом, что в свою очередь способствует повышению надежности и стабильности всей системы.

### **3.4 Тестирование Контроллеров**

Тесты были разработаны для основных классов бизнес логики рассматриваемой программы, а именно для Продукта и Заказа, и всеми последовательностями с ними связанными.

Так для соответствующих контроллеров были разработаны тесты, проверяющие их основные методы. В тестах использовались mock-объекты для изоляции от других компонентов системы и проверки работы контроллеров   
в независимости от реальных зависимостей.

**AdminControllerTest**:

Юнит-тесты для контроллера **AdminController** управляют операциями администрирования для веб-приложения по заказу пиццы. Тесты используют **MockMvc** для имитации HTTP-запросов и проверяют корректность работы методов контроллера. Основное внимание уделено тестированию функций добавления, удаления и изменения количества продуктов, а также обработке доступа с использованием JWT-токенов. Для этого используются моки, чтобы изолировать сервисные компоненты от других частей системы. Тесты проверяют как успешные, так и неудачные сценарии выполнения операций. Так в частности используются следующие тесты:

**- Тестирование успешного добавления продукта (testPostProduct\_Success)**

В этом тесте проверяется, что при корректном JWT-токене (передаваемом в заголовке запроса) метод postProduct контроллера вызывает соответствующий сервисный метод, и ответ имеет статус 200 OK. Тест выполняет запрос   
на добавление нового продукта с данными, представленными в JSON-формате, и ожидает успешное выполнение операции.

**- Тестирование попытки добавления продукта с некорректным токеном (testPostProduct\_Forbidden)**

Этот тест проверяет, как система реагирует на попытку добавления продукта с некорректным токеном (например, токеном пользователя без прав администратора). В данном случае метод контроллера должен вернуть ответ с кодом 403 (Forbidden), и метод сервиса не должен быть вызван.

**- Тестирование успешного удаления продукта (testDeleteProduct\_Success)**

Здесь тестируется удаление продукта через API. При корректном токене метод deleteProduct контроллера вызывает соответствующий сервисный   
метод для удаления продукта. После выполнения запроса ожидается статус   
200 OK.

**- Тестирование попытки удаления продукта с некорректным токеном (testDeleteProduct\_Forbidden)**

В этом тесте проверяется, что при использовании некорректного токена (не администратора) запрос на удаление продукта будет отклонен с кодом 403. Метод сервиса не должен быть вызван.

**- Тестирование успешного увеличения количества продукта (testIncProductCount\_Success)**

Этот тест проверяет увеличение количества продукта. При корректном токене метод incProductCount контроллера вызывает сервис для увеличения количества, и операция должна завершиться с успешным статусом 200 OK.

**- Тестирование попытки увеличения количества продукта   
с некорректным токеном (testIncProductCount\_Forbidden)**

В данном тесте проверяется то, что если передан некорректный токен, метод не будет вызван, а система вернет статус 403 (Forbidden), отклоняя запрос.

**- Тестирование успешного уменьшения количества продукта (testDecProductCount\_Success)**

Этот тест проверяет успешное уменьшение количества продукта. При корректном токене ожидается, что метод decProductCount контроллера вызовет сервис для уменьшения доступного количества и вернет статус 200 OK.

**- Тестирование попытки уменьшения количества продукта   
с некорректным токеном (testDecProductCount\_Forbidden)**

Тест проверяет, что запрос на уменьшение количества продукта   
с некорректным токеном будет отклонен с кодом 403, а метод сервиса не будет вызван.

**OrderControllerTest**

Тест фокусируется на проверке метода, который возвращает количество текущих заказов для пользователя, извлекая его идентификатор из JWT токена.

Таки образом, тест проверяет, что метод currentOrderCount контроллера правильно взаимодействует с сервисом OrderService для получения текущего количества заказов пользователя. Тест также проверяет, что токен авторизации корректно обрабатывается и передается в метод сервиса. В этих целях используется следующая последовательность действий:

**- Мокаем статический метод JWTParser.extractNumber**: для того чтобы получить номер пользователя из токена, используется статический метод JWTParser.extractNumber. С помощью MockedStatic мокаем этот метод, чтобы возвращать предсказуемое значение для теста. В данном случае токен "Bearer header.payload.signature" будет извлекать номер пользователя "123".

**- Мокаем поведение OrderService**: создаем моки для OrderService, чтобы метод currentOrderCount возвращал нужное количество заказов —   
в данном случае 5.

Это нужно для того, чтобы изолировать тест от реальной логики сервиса и проверить, правильно ли контроллер обрабатывает возвращаемое значение.

**Выполнение HTTP-запроса через MockMvc**: Используя **MockMvc**, выполняем GET-запрос к конечной точке /api/products/secure/currentorder/count, передавая в заголовке запроса токен авторизации.

В данном случае ожидается, что сервис вернет количество заказов пользователя, и проверяем, что статус ответа будет 200 OK, а содержимое ответа будет равно 5 (количеству заказов).

**Проверка вызова сервиса**: Используя метод verify из Mockito, проверяем, что метод currentOrderCount сервиса был вызван ровно один раз с нужным идентификатором пользователя.

### **3.5 Тестирование Сервисов**

Тестирование сервисного слоя сосредоточено на проверке бизнес-логики. Тесты помогают убедиться, что сервисы выполняют необходимые операции корректно, включая обработку исключений и правильное взаимодействие   
с репозиториями.

**AdminService**:

Здесь осуществляются юнит-тесты для методов, которые отвечают   
за управление продуктами в системе. Так тестируются различные операции, такие как добавление, удаление и изменение количества продуктов.

Тестирование выполняется с помощью библиотеки **JUnit 5** и **Mockito** для создания моков зависимостей, а также проверки корректности взаимодействий   
с репозиториями.

**Тестирование метода postProduct (добавление нового продукта):**В тесте создается новый продукт с помощью AppendProductRequest,   
и вызывается метод postProduct сервиса. После этого проверяется, что метод save репозитория был вызван один раз.

**Тестирование метода deleteProduct (удаление продукта)   
с отсутствием продукта:** Мокируется ситуация, когда продукт не найден в базе данных (метод findById возвращает Optional.empty()).

Ожидается, что будет выброшено исключение с сообщением "Товар отсутствует".

**Тестирование метода deleteProduct (удаление продукта) с успешным удалением:** мокируется ситуация, когда продукт найден в базе данных. Затем выполняется удаление, и проверяется, что были вызваны методы delete для продукта, а также методы удаления заказов и отзывов, связанные с этим продуктом.

**Тестирование метода incProductCount (увеличение количества продукта) с отсутствием продукта:** мокируется ситуация, когда продукт   
не найден в базе данных, и проверяется, что при вызове метода увеличения количества будет выброшено исключение с сообщением "Товар отсутствует".

**Тестирование метода incProductCount (увеличение количества продукта) с успешным увеличением:** Мокируется ситуация, когда продукт найден в базе данных, и выполняется увеличение количества. Ожидается,   
что количество продукта увеличится на 1, и метод save будет вызван для сохранения обновленного продукта.

**Тестирование метода decProductCount (уменьшение количества продукта) с отсутствием продукта:** Мокируется ситуация, когда продукт не найден в базе данных, и проверяется, что при вызове метода уменьшения количества будет выброшено исключение с сообщением "Товар отсутствует".

**Тестирование метода decProductCount (уменьшение количества продукта) с продуктом, у которого количество равно 0:** Мокируется ситуация, когда количество продукта равно 0. Проверяется, что метод уменьшения количества не изменяет количество, и метод save не вызывается.

**Тестирование метода decProductCount (уменьшение количества продукта) с успешным уменьшением:** Мокируется ситуация, когда продукт найден в базе, и выполняется уменьшение количества.

Проверяется, что количество продукта уменьшилось на 1, и метод save был вызван для сохранения обновленного продукта.

### **3.6 Тестирование репозиториев**

Тесты выполняются с использованием библиотеки **JUnit 5** и Spring Boot, обеспечивая интеграционное тестирование, в котором данные сохраняются   
и извлекаются из реальной базы данных через репозитории. Основные задачи тестирования — проверка методов репозитория, которые взаимодействуют   
с сущностями заказов и продуктов.

**OrderRepositoryTest**:

**- Тестирование метода findOrderByUserNumberAndProductId:** проверить, что заказ можно найти по номеру телефона пользователя и ID продукта. В тесте:

* + Создается продукт, сохраняется в базе данных.
  + Создается заказ с номером пользователя и ID продукта.
  + Проверяется, что заказ можно найти с помощью метода findOrderByUserNumberAndProductId, и что данные заказа корректны.

**Проверки**:

* Наличие найденного заказа.
* Сравнение номера пользователя и ID продукта с ожидаемыми значениями.

**-  Тестирование метода findOrdersByUserNumber:** Проверить, что все заказы для указанного пользователя можно найти.

В тесте:

* + Создаются два заказа с одинаковым номером телефона, но с разным количеством.
  + Метод findOrdersByUserNumber вернет список всех заказов для этого пользователя.

**Проверки**:

* Размер списка заказов для пользователя должен быть равен 2   
  (так как было добавлено два заказа).

**3. Тестирование метода deleteAllByProductId:** проверить, что заказы можно удалить по ID продукта.

* В тесте:
  + Создаются два заказа с разными номерами пользователей, но с одним и тем же продуктом.
  + Метод deleteAllByProductId вызывается для удаления всех заказов, содержащих этот продукт.
  + Проверяется, что заказы для данного продукта были удалены.

**Проверки**:

* Перед удалением проверяется, что заказ с указанным номером пользователя существует.
* После удаления проверяется, что заказ для этого продукта был удален.

**ProductRepositoryTest**:

Репозиторий ProductRepository обеспечивает работу с сущностью **Product** в базе данных.

В тестах проверяется правильность выполнения методов репозитория, таких как поиск продуктов по частичному совпадению   
в названии и категории, а также поиск продуктов по списку их идентификаторов.

**Тестирование метода findByNameContaining:** Проверить поиск продуктов по части названия с пагинацией.

* В тесте:
  + Создается несколько продуктов с названиями, содержащими слово "Pizza".
  + Используется метод findByNameContaining, который ищет продукты, содержащие "Pizza" в названии, с пагинацией (первые 2 продукта).
  + Проверяется, что на странице содержится 2 продукта, и что хотя бы один из них содержит "Pizza" в названии.

**Проверки**:

* Размер страницы равен 2.
* На странице присутствуют продукты, содержащие слово "Pizza".

**Тестирование метода findByNameContainingNoResults:** проверить, что метод findByNameContaining возвращает пустой результат, если нет продуктов, соответствующих запросу.

* В тесте:
  + Используется метод findByNameContaining для поиска продуктов   
    с названием "NonExistent", которого нет в базе.
  + Проверяется, что результат пустой (не найдено продуктов).

**Проверки**:

* Количество найденных продуктов на странице равно 0.

**Тестирование метода findByCategoryContaining:** проверить поиск продуктов по категории с пагинацией.

* В тесте:
  + Создаются продукты с категорией "Italian" и "Vegetarian".
  + Используется метод findByCategoryContaining, который ищет продукты, содержащие "Italian" в категории.
  + Проверяется, что на странице содержатся два продукта с категорией "Italian" (с пагинацией на 2 продукта).

**Проверки**:

* Размер страницы равен 2.
* На странице присутствуют продукты с категорией "Italian".

**Тестирование метода findByCategoryContainingNoResults:** проверить, что метод findByCategoryContaining возвращает пустой результат, если нет продуктов, соответствующих запросу.

* В тесте:
  + Используется метод findByCategoryContaining для поиска продуктов   
    с категорией "NonExistent", которой нет в базе.
  + Проверяется, что результат пустой (не найдено продуктов).

**Проверки**:

* Количество найденных продуктов на странице равно 0.

**Тестирование метода findProductByProductIds:** проверить поиск продуктов по списку их идентификаторов.

* В тесте:
  + Создается список с двумя идентификаторами продуктов.
  + Используется метод findProductByProductIds, чтобы найти продукты с этими идентификаторами.
  + Проверяется, что найдено два продукта, соответствующие указанным идентификаторам.

**Проверки**:

* Размер списка найденных продуктов равен 2.
* Продукты с указанными идентификаторами присутствуют   
  в результатах.

**Тестирование метода findProductByProductIdsNoResults:** проверить, что метод findProductByProductIds возвращает пустой результат, если продукты с указанными идентификаторами не существуют.

* В тесте:
  + Используется метод findProductByProductIds для поиска продуктов   
    с идентификаторами, которых нет в базе.
  + Проверяется, что результат пустой (не найдено продуктов).

**Проверки**:

* Список найденных продуктов пустой.

**Основные элементами реализации в файлах Репозиториев являются:**

1. **Spring Boot Test**:
   * Аннотация @DataJpaTest инициирует тестирование слоев данных   
     в Spring.

Она настроена на использование базы данных в памяти (например, H2), чтобы тесты могли выполняться быстро и не зависели от реальной базы данных.

* + Аннотация @ExtendWith(SpringExtension.class) используется для поддержки Spring TestContext Framework в JUnit 5.

1. **Создание и сохранение данных**:
   * В каждом тесте создаются объекты продуктов, которые сохраняются в базе данных перед выполнением тестов.
   * Репозитории используются для сохранения и извлечения данных.
2. **Пагинация**:
   * В тестах используется пагинация для ограничения количества возвращаемых продуктов на одной странице (PageRequest.of(0, 2) — первая страница, 2 продукта на странице).
3. **JUnit Assertions**:
   * assertEquals — проверяет, что два значения равны.
   * assertTrue — проверяет, что условие истинно.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была проведена комплексная работа по проектированию, разработке и внедрению веб-приложения, предназначенного для дистанционных заказов в организациях общественного питания.

Результаты проделанной работы демонстрируют, что все поставленные цели достигнуты, а созданная система соответствует заявленным требованиям и предоставляет широкие возможности для дальнейшего развития.

Основной целью разработки было создание системы, которая   
бы позволяла пользователям удобно и эффективно делать дистанционные заказы в сфере общественного питания. Эта система должна была автоматизировать процессы заказа, минимизировать ошибки при оформлении, а также повысить общую эффективность обслуживания клиентов.

В процессе реализации были решены задачи разработки архитектуры приложения, создания и тестирования его компонентов, а также интеграции всех элементов в единую рабочую систему.

В процессе разработки функционала был реализован ряд важных операций, таких как импорт и экспорт данных, поддержка REST API и работа с базой данных.

Это делает систему удобной для интеграции с другими приложениями и внешними сервисами, а также позволяет адаптировать её под специфические требования пользователей. Встроенные функции для отслеживания заказов и анализа данных предоставляют дополнительную ценность для конечных пользователей и позволяют повысить качество обслуживания.

Одним из ключевых факторов стабильности и надежности системы является внедрение юнит- и интеграционных тестов, что позволило обеспечить качество программного обеспечения и снизить риски возникновения ошибок.

Использование современных инструментов тестирования, таких как Mockito и JUnit 5, значительно улучшило процесс поиска и устранения багов, повысив производительность команды разработки и уменьшив время   
на проверку функциональности.

Архитектура приложения была спроектирована с учетом принципов модульности и разделения ответственности. Это позволило создать код, который легко поддерживать и модифицировать.

Например, слой контроллеров отвечает за обработку запросов   
от пользователей, слой сервисов реализует бизнес-логику, а репозитории занимаются взаимодействием с базой данных. Такой подход упрощает работу   
с кодом и ускоряет внедрение изменений.

Результаты работы показывают, что разработанная система решает задачи, связанные с автоматизацией процесса заказа в сфере общественного питания. Она минимизирует влияние человеческого фактора, повышает точность обработки данных и делает процесс обслуживания клиентов более удобным и быстрым. Эти преимущества делают систему полезным инструментом для ресторанов, кафе и других организаций общественного питания.

Несмотря на достигнутые результаты, существуют направления   
**для дальнейшего улучшения системы**, как с точки зрения разработки, так и с точки зрения функций системы.

Данная работа была проделана в рамках сдачи курсовой работы. Вместе с тем на практике данный код можно дополнить множеством дополнительных функций, таких как:

- **определение места доставки** по предоставленной локации либо предоставленных клиентом;

- **выбор способов оплаты** за заказ, в том числе наличностью либо посредством банковских карт;

- **отслеживание доставки** до места назначения заказа, указанного клиентом во время заказа;

- смс подтверждение номера телефона в целях создания уникальных клиентов для дальнейшей персональной работы с ними и т.д.

Другим важным направлением является внедрение аналитических инструментов, таких как генерация отчетов по заказам и анализ данных о предпочтениях пользователей.

Это позволит организациям общественного питания лучше понимать потребности клиентов и оптимизировать меню и услуги. Возможности для визуализации данных и прогнозирования спроса также будут полезны для менеджеров и управляющих.

Оптимизация производительности системы — это еще одно направление для дальнейшей работы. В случае роста числа пользователей или увеличения объема заказов можно внедрить механизмы кэширования, которые ускорят обработку данных.

Переход на облачные решения или использование распределенных баз данных поможет обеспечить систему для работы с большими объемами данных, сохраняя при этом высокую скорость работы.

Важным аспектом в долгосрочной перспективе является обеспечение безопасности данных. Необходимо разработать систему аутентификации   
и авторизации, а также меры защиты информации от несанкционированного доступа, особенно в условиях работы с конфиденциальными данными клиентов и заказов.

Использование новых технологий также открывает возможности для улучшения системы. Например, внедрение элементов машинного обучения может помочь в анализе предпочтений пользователей и прогнозировании спроса на те или иные блюда. Такие функции будут полезны для крупных ресторанных сетей, где необходима обработка больших объемов данных.

Кроме того, улучшение пользовательского интерфейса, добавление дополнительных настроек для кастомизации и возможность работы через мобильные устройства сделают систему ещё более удобной для конечных пользователей.

Это расширит её функциональность и обеспечит доступность для более широкой аудитории.

Таким образом, проект представляет собой надежную и гибкую основу для дальнейшего развития.

Применение выбранных подходов к разработке доказало свою эффективность, и созданное веб-приложение способно удовлетворить потребности организаций общественного питания.

Реализация предложенных улучшений и расширений значительно повысит функциональность системы и обеспечит её востребованность на рынке в будущем.

## Список использованной литературы

1. **Joshua Bloch.** Effective Java. Addison-Wesley, 3rd Edition, 2018. Книга, описывающая лучшие практики программирования на Java, включая работу   
   с объектами, потоками и коллекциями, которые были полезны при разработке веб-приложения для дистанционных заказов.
2. **Craig Walls.** Spring in Action. Manning Publications, 6th Edition, 2022. Руководство по Spring Framework, включая Spring Boot, которое использовалось для создания backend-части приложения.
3. **Christian Bauer, Gavin King, и Gary Gregory.** Java Persistence with Hibernate. Manning Publications, 2nd Edition, 2015. Основной источник знаний   
   о работе с Hibernate и JPA, использованный для реализации работы с базой данных в проекте.
4. **Baeldung.** Spring Tutorials. Серия статей по Spring Framework и Spring Boot, доступных на: <https://www.baeldung.com/>. Ресурс, который предоставил полезные материалы для понимания принципов разработки на Spring.
5. **Oracle.** Java SE Documentation. Официальная документация Java SE, доступная на: <https://docs.oracle.com/en/java/javase/>. Основной источник информации о языке программирования Java, использованной для разработки приложения.
6. **H2 Database Engine.** Official Documentation. Документация H2 базы данных, доступная на: <http://h2database.com/html/main.html>. Информация   
   о настройке и использовании базы данных H2, которая применялась для хранения данных о заказах.
7. **JUnit 5 Team.** JUnit 5 User Guide. Руководство по тестированию   
   с использованием JUnit 5, доступное на: <https://junit.org/junit5/docs/current/user-guide/>. Документация, используемая для реализации юнит-тестирования   
   в процессе разработки веб-приложения.
8. **Mockito Team.** Mockito Documentation. Руководство по библиотеке Mockito, доступное на: <https://site.mockito.org/>. Ресурс, использованный для создания и выполнения юнит-тестов с использованием mock-объектов.
9. **Martin Fowler.** Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2nd Edition, 2018. Книга о рефакторинге и улучшении существующего кода, которая оказала влияние на архитектурное решение   
   в проекте.
10. **Spring Boot Reference Documentation.** Полное руководство по работе с Spring Boot, доступное на: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>. Документация, использованная при разработке backend-части веб-приложения с использованием Spring Boot.
11. **Bruce Eckel.** Thinking in Java. Prentice Hall, 4th Edition, 2006. Классическое руководство по языку Java, которое позволило глубже понять принципы работы с объектами и многими аспектами Java.
12. **JetBrains.** IntelliJ IDEA User Guide. Руководство пользователя для интегрированной среды разработки IntelliJ IDEA, доступное на: <https://www.jetbrains.com/idea/documentation/>. Руководство по использованию IntelliJ IDEA для разработки веб-приложения.
13. **Fowler, Beck, Brant, et al.** Patterns of Enterprise Application Architecture. Addison-Wesley, 1st Edition, 2002. Книга, посвященная архитектурным шаблонам для разработки корпоративных приложений, которая послужила источником для разработки архитектуры веб-приложения.
14. **Docker Documentation.** Документация по настройке и использованию Docker: <https://docs.docker.com/>. Документация, используемая для контейнеризации приложения и обеспечения его развёртывания.
15. **Postman API Platform Documentation.** Руководство по тестированию API с использованием Postman: <https://www.postman.com/api-documentation/>. Ресурс для тестирования API, что оказалось полезным при проверке функциональности и корректности работы системы заказов.
16. **RESTful Web Services Documentation.** Стандартные практики   
    и принципы построения RESTful API: <https://restfulapi.net/>. Описание принципов создания RESTful сервисов, которые использовались для разработки интерфейса взаимодействия с веб-приложением.
17. **Spring Security Reference Documentation.** Официальная документация Spring Security для интеграции с Spring Boot: <https://docs.spring.io/spring-security/reference/>. Руководство по использованию Spring Security для обеспечения безопасности веб-приложения.
18. **Eric Evans.** Domain-Driven Design: Tackling Complexity in the Heart   
    of Software. Addison-Wesley, 1st Edition, 2003. Книга о концепциях проектирования для сложных систем, которая использовалась при разработке архитектуры приложения для дистанционного заказа.
19. **Vlad Mihalcea.** High-Performance Java Persistence. Amazon Digital Services LLC, 1st Edition, 2016. Руководство по оптимизации производительности JPA и Hibernate, используемое для повышения эффективности работы с базой данных.
20. **Linus Torvalds.** Git Documentation. Официальное руководство   
    по использованию Git: <https://git-scm.com/doc>. Ресурс для работы с системой контроля версий Git, использовавшийся в процессе разработки проекта.
21. **PostgreSQL Documentation.** Официальная документация по работе   
    с PostgreSQL, доступная на: <https://www.postgresql.org/docs/>. Ресурс, использованный для настройки и работы с базой данных PostgreSQL, которая может быть использована в будущем для расширения системы.
22. **JSON.org.** Основная информация о формате JSON, доступная   
    на: <https://www.json.org/>. Документация по формату JSON, использованная   
    для обмена данными между клиентом и сервером.
23. **Mark Lutz.** Programming Python. O’Reilly Media, 4th Edition, 2010. Книга, применявшаяся для анализа взаимодействия с внешними системами   
    и для дополнительных возможностей в интеграции с Python.
24. **OpenAPI Initiative.** OpenAPI Specification. Документация для проектирования API: <https://swagger.io/specification/>. Описание стандарта OpenAPI для проектирования API, используемого при создании RESTful API для веб-приложения.
25. **Apache Maven Project.** Maven: The Complete Reference. Документация Maven, доступная на официальном сайте: <https://maven.apache.org/>. Ресурс по настройке и использованию Maven для управления зависимостями и сборкой проекта.
26. **Martin Kleppmann.** Designing Data-Intensive Applications. O’Reilly Media, 1st Edition, 2017. Книга о проектировании масштабируемых и надежных систем, использовавшаяся для выбора подходящих технологий обработки данных в проекте.
27. **GitHub Actions Documentation.** Официальное руководство по настройке и использованию CI/CD пайплайнов: <https://docs.github.com/en/actions>. Документация по настройке автоматических сборок и тестирования с помощью GitHub Actions.