Geekbrains

**Разработка веб приложения «Система управления заказами еды» на JavaSpring.**

Студент:

Разработка веб приложений на Java

Ковалев Е.М.

Курск

2025

**Дипломный проект**

**«Разработка веб приложения «Система управления заказами еды» на JavaSpring»**

**Содержание**

|  |  |
| --- | --- |
| Введение……………………………………………………………………. | 3 |
| Глава 1. Предпроектная стадия…………………………………………… | 5 |
| 1.1. Порядок проведения работ…………………………………………… | 5 |
| 1.2. Определение требований к приложению…………………………… | 6 |
| 1.3. Выбор стека технологий……………………………………………… | 8 |
| Глава 2. Проектная стадия………………………………………………… | 14 |
| 2.1. Проектирование общей архитектуры приложения………………… | 14 |
| 2.2. Проектирование web-интерфейса приложения…………………….. | 56 |
| 2.3. Проектирование базы данных……………………………………….. | 63 |
| 2.4. Безопасность приложения……………………………………………. | 64 |
| Заключение………………………………………………………………… | 70 |
| Список использованной литературы…………………………………….. | 72 |

**Введение**

В связи с ростом популярности онлайн-заказов и стремлением пользователей быстро и удобно управлять своими покупками, мы разработали универсальное веб-приложение для оформления заказов, которое может быть адаптировано под любой вид товаров. В качестве примера реализации выбрана система управления заказами еды, демонстрирующая гибкость и масштабируемость платформы.

Система управления заказами — это инструмент, который упрощает процесс выбора товаров, оформления заказов и отслеживания их статуса. Пользователи могут просматривать каталог, добавлять товары в корзину и следить за этапами выполнения заказа. Приложение может поддерживать различные категории товаров (например, еду, электронику, одежду), но для демонстрации функционала в этом проекте акцент сделан на работу с меню онлайн-ресторанов.

Приложение помогает клиентам экономить время, получать актуальную информацию о доступных товарах и статусах заказов, а администраторам — эффективно управлять каталогом, анализировать спрос и оптимизировать процессы.

 Основная функциональность приложения включает:

**Универсальный каталог товаров**: Поддержка различных типов товаров (еда, товары, услуги) с возможностью кастомизации атрибутов (например, размер одежды, технические характеристики). В примере реализовано меню с блюдами (пицца, роллы), где для каждого типа заданы уникальные параметры (диаметр пиццы, количество кусочков роллов).

**Корзина и оформление заказов**: Добавление товаров в корзину, выбор статуса заказа (**NEW, IN\_PROGRESS, COMPLETED** ).

**Админ-панель:** Управление каталогом (добавление/удаление товаров), Мониторинг заказов и редактирование статусов.

**Безопасность**: Шифрование паролей (BCrypt), ролевая модель доступа (CLIENT/STAFF/ADMIN).

**Универсальность платформы обеспечивается:**

1. **Модульной архитектурой**:
   * Использование абстрактных классов и наследования (например, MenuItem для разных типов товаров).
   * Поддержка кастомных атрибутов через поля в БД (например, diameter для пиццы, size для одежды).
2. **Гибкими настройками** :
   * Возможность добавлять новые категории товаров без изменения ядра приложения.

Целью данного проекта было создание универсальной платформы, которая может быть адаптирована под любые типы товаров, сохраняя при этом простоту и интуитивность интерфейса. Пример с заказом еды демонстрирует, как система справляется с задачами в условиях высокой вариативности атрибутов товаров и требований пользователей.

**Глава 1. Пред проектная стадия**

**1.1. Порядок проведения работ**

В ходе проектирования и создания проекта были проведены следующие работы:

1. **Планирование проекта**:
   * Определение возможностей приложения, а также нефункциональных требований (производительность, безопасность и др.).
   * Определение архитектурных особенностей, выбор структуры приложения и технологий для реализаци.
2. **Настройка окружения разработки**:

* Установка инструментов :
  + Java Development Kit (JDK) для разработки.
  + Apache Maven для управления зависимостями и сборки.
  + Spring Boot как основа для создания приложения.

1. **Создание проекта**:

* Инициализация через Spring Initializr :
  + Генерация базового проекта с настройками по умолчанию.
* Подключение зависимостей :
  + Spring Web (веб-функционал).
  + Spring Data JPA (работа с базой данных).
  + Lombok (упрощение кода).
  + Spring Security (безопасность).
  + Thymeleaf (шаблонизация интерфейса).

1. **Разработка компоненктов:**

* Модель данных: Определение сущностей (пользователи, записи, категории).
* Репозитории: Интерфейсы для взаимодействия с БД.
* Сервисы: Бизнес-логика приложения.
* Контроллеры: Обработка HTTP-запросов.

1. **Разработка пользовательского интерфейса**:

* HTML-шаблоны :
  + Использование Thymeleaf для динамического отображения данных.
  + Интеграция Bootstrap для стилизации.

1. **Отладка и оптимизация**:

* Исправление ошибок : Тестирование и улучшение кода.
* Повышение эффективности :
  + Оптимизация запросов к БД.
  + Настройка кэширования и ресурсов.

1. **Документирование**:
   * Создание документации по проекту, включая описание архитектуры, инструкции по установке и запуску, описание API (если применимо) и другие справочные материалы.

**1.2. Определение требований к приложению**

Для разработки приложения «Система управления заказами еды» были определены следующие требования:

1. Функциональные требования:
   * Пользователи должны иметь возможность создавать и редактировать и заказы через корзину.
   * Выбор блюд из категоризированного меню (пицца, роллы), где каждое блюдо имеет уникальные атрибуты (диаметр, количество кусочков, цена).
   * Справочник меню с описанием блюд, ценами и характеристиками (хранится в Н2).
   * Отслеживание заказов:
   * Визуализация статуса заказа в реальном времени (NEW, IN\_PROGRESS, COMPLETED).
   * Аналитика для администраторов:
   * Имеется возможность редактировать меню в заведениях, отслеживать количество и состав заказов по выбранным заведениям.
   * У служащих в заведении должна иметься возможность просматривать весь список заказов по заведениям, подключенным к системе.
2. Нефункциональные требования:   
   • Удобство использования: Интерфейс должен быть интуитивно понятен для всех категорий пользователей (клиенты, сотрудники, администраторы), обеспечивать простоту навигации, оформления заказов и управления данными.  
   • Безопасность: Аутентификация пользователей и передача данных должны быть защищены с использованием шифрования (BCrypt для паролей, HTTPS для коммуникации). Ролевая модель (ADMIN, STAFF, USER) должна строго ограничивать доступ к функциям.  
   • Производительность: Приложение должно обеспечивать мгновенный доступ к меню, корзине и статусам заказов, а также обрабатывать запросы в реальном времени без задержек, даже при высокой нагрузке.  
   • Масштабируемость: Архитектура должна позволять расширять функционал (например, добавление новых категорий блюд или новых видов товаров) и увеличивать количество пользователей без снижения скорости работы.
3. Требования к интерфейсу пользователя:
   * Интуитивный дизайн для просмотра меню, добавления блюд в корзину, оформления заказов и отслеживания их статусов (NEW, IN\_PROGRESS, COMPLETED).
   * Удобные инструменты для управления меню (добавление/удаление пицц и роллов), продуктами и заказами.
   * Возможность фильтрации и поиска заказов по статусам, дате или пользователю.
4. Требования к безопасности:
   * Разделение прав доступа на основе ролей: ADMIN (полный доступ), STAFF (управление заказами и меню для работников ресторана), USER (клиентские функции).
   * Хранение паролей в зашифрованном виде (BCrypt), защита сессий от перехвата.
5. Требованияк производительности:
   * Быстрый доступ к данным и отзывчивый интерфейс приложения.
6. Требования к масштабируемости**:**

• Возможность расширения функционала:

* + Добавление новых категорий блюд (например, напитки, десерты).
  + Интеграция платежных систем и уведомлений.

**1.3. Выбор стека технологий**

**Выбор языка программирования**: В качестве основного языка для серверной части приложения был выбран Java. Этот язык обладает рядом преимуществ, включая кроссплатформенность, высокую производительность и богатую экосистему библиотек. Java идеально подходит для создания масштабируемых решений благодаря поддержке многопоточности, механизмам сборки мусора и строгой типизации. Его широкое применение в корпоративной среде и надежность делают его оптимальным выбором для долгосрочных проектов, требующих стабильности и безопасности.

* **Фреймворк для веб-разработки**: Для ускорения разработки использован Spring Boot — современный фреймворк, который минимизирует ручную настройку за счет автоматической конфигурации компонентов. Модуль **Spring MVC** обеспечивает структурированное создание веб-интерфейсов, разделяя логику обработки запросов и представления данных. **Spring Security** интегрирован для реализации комплексной системы безопасности: аутентификации через формы и OAuth2, авторизации на основе ролей (ADMIN, USER), защиты от CSRF-атак и SQL-инъекций. Эти инструменты позволяют быстро развернуть безопасный бэкенд с минимальными усилиями.
* **База данных** В качестве СУБД выбрана **H2** — легковесная встраиваемая реляционная база данных с открытым исходным кодом, поддерживающая ACID-транзакции и SQL-стандарты. Её ключевые особенности, такие как высокая скорость работы в in-memory режиме, компактность и простота настройки, делают её идеальной для тестирования и разработки. H2 позволяет хранить структурированные данные (например, информацию о пользователях, заказах и продуктах) и поддерживает механизмы резервного копирования, что обеспечивает базовую защиту данных.
* **Хранение данных о чувствах**: Для записи информации о существующих блюдах, меню конкретных заведений, а также информации о заказах пользователей используется отдельная схема в Н2. Данные организованы в нормализованные таблицы с индексами для оптимизации запросов. Для работы с базой применен **Hibernate** — ORM-инструмент, который автоматизирует маппинг Java-объектов на SQL-сущности, что упрощает CRUD-операции и снижает риск ошибок при ручном написании запросов.
* **Клиентская часть:** Интерфейс приложения построен на HTML с использованием семантической разметки для улучшения доступности. Для ускорения разработки и обеспечения адаптивности задействован фреймворк Bootstrap, предоставляющий готовые компоненты (кнопки, формы, сетки) и утилиты для создания отзывчивых макетов. Его интеграция позволяет единообразно отображать контент на мобильных устройствах и десктопах, а также сокращает время на стилизацию.
* **Шаблонизатор**: Генерация динамических HTML-страниц реализована через **Thymeleaf** — современный шаблонизатор, тесно интегрированный с Spring. Он поддерживает естественные шаблоны (Natural Templates), которые можно просматривать в браузере без запуска сервера. Thymeleaf упрощает передачу данных из контроллеров в представления, поддерживает фрагменты для повторного использования кода и обеспечивает защиту от XSS-уязвимостей через экранирование переменных.
* **Механизмы безопасности:** Spring Security настроен для многоуровневой защиты. Аутентификация выполняется через форму входа с валидацией учетных данных в базе данных. Пароли хранятся в хешированном виде с использованием алгоритма BCrypt. Авторизация реализована на основе ролей: например, доступ к административным панелям ограничен для пользователей с ролью ADMIN. Дополнительно настроены HTTPS-шифрование, ограничение попыток входа и сессионные таймауты.
* **Среда разработки**: Основная среда — **IntelliJ IDEA**, которая предоставляет продвинутые функции: умное автодополнение кода, статический анализ, интеграцию с Maven/Gradle и Docker. Для отладки используется встроенный дебаггер с точками останова и мониторингом переменных. Дополнительные плагины (Lombok, Spring Assistant) ускоряют работу с аннотациями и конфигурацией Spring-компонентов.

Архитектура приложения, разработанного на основе Java Spring, базируется на популярном паттерне проектирования **MVC (Model-View-Controller)**. Этот подход позволяет четко разделить приложение на три ключевых компонента: модель (Model), представление (View) и контроллер (Controller). Такое разделение обеспечивает структурированность кода, упрощает его поддержку и позволяет масштабировать приложение по мере необходимости. В нашем приложении каждый из этих компонентов выполняет свою уникальную роль, а также добавлены дополнительные слои, такие как репозитории, сервисы и сервер, которые в совокупности формируют полноценную архитектуру. Рассмотрим каждый из них подробнее:

**Модель (Model)**:

Модель представляет собой основу для работы с данными в приложении. Она включает в себя классы, которые описывают сущности, используемые в системе. Например, в нашем приложении это могут быть такие сущности, как "Пользователь" (User) и "Продукт" (Product). Каждая сущность моделируется в виде Java-класса с полями, соответствующими атрибутам объекта, и методами для работы с этими данными. Модель также может включать связи между сущностями, такие как "один ко многим" или "многие ко многим", что позволяет отражать сложные взаимосвязи в данных. Модель не зависит от представления и контроллера, что делает её универсальной и легко переиспользуемой.

**Представление (View)**:

Представление отвечает за отображение данных пользователю в удобной и понятной форме. В веб-приложениях на Java Spring представление чаще всего реализуется с использованием шаблонизаторов, таких как **Thymeleaf** или **JSP**. Эти инструменты позволяют динамически вставлять данные, полученные от контроллера, в HTML-шаблоны. Например, список блюд в заказе может быть отображен в виде таблицы, где каждая строка соответствует одной позиции заказа. Шаблонизаторы также поддерживают условные конструкции, циклы и фрагменты, что делает их мощным инструментом для создания сложных интерфейсов. Представление не содержит бизнес-логики, а лишь отображает данные, что обеспечивает разделение ответственности.

**Контроллер (Controller)**:

Контроллер играет роль посредника между пользователем, моделью и представлением. Он обрабатывает запросы, поступающие от клиента (например, через браузер), и определяет, какие действия необходимо выполнить. В Java Spring контроллеры реализуются в виде классов, аннотированных **@Controller** или **@RestController**. Каждый метод контроллера связан с определенным HTTP-запросом (GET, POST, PUT, DELETE) и вызывает соответствующие сервисы для выполнения бизнес-логики. Например, при получении запроса на добавление нового заказа контроллер может вызвать метод сервиса, который сохранит данные в базе. После выполнения операции контроллер возвращает результат, который может быть представлением (HTML-страницей) или JSON-ответом в случае REST API.

**Репозиторий (Repository)**:

Репозиторий отвечает за взаимодействие с базой данных. В Java Spring репозитории обычно реализуются через интерфейсы, расширяющие **JpaRepository**. Эти интерфейсы предоставляют готовые методы для выполнения стандартных операций с данными, таких как поиск, сохранение, обновление и удаление. Например, репозиторий для сущности "Заказ" может содержать методы для поиска определенных заказов по статусу его готовности или получения списка всех заказов. Репозитории используются сервисами для доступа к данным, что позволяет абстрагироваться от деталей работы с базой данных и сосредоточиться на бизнес-логике.

**Сервис (Service)**:

Сервисный слой содержит основную бизнес-логику приложения. Сервисы координируют взаимодействие между различными компонентами, такими как контроллеры и репозитории. Например, сервис для работы с блюдами в меню ресторанов может включать методы для добавления нового блюда, редактирования информации о существующих блюдах или расчет стоимости заказа. Сервисы также могут выполнять сложные операции, такие как валидация данных, обработка исключений и вызов внешних API. Использование сервисного слоя позволяет разделить логику приложения на отдельные модули, что упрощает тестирование и поддержку кода.

**Сервер (Server)**:

Сервер представляет собой среду, в которой работает приложение. В Java Spring приложение обычно запускается на встроенном сервере, таком как **Tomcat** или **Jetty**. Эти серверы обрабатывают HTTP-запросы, управляют жизненным циклом приложения и обеспечивают его взаимодействие с внешним миром. Встроенный сервер позволяет быстро развернуть приложение без необходимости настройки внешнего веб-сервера. Кроме того, Spring Boot автоматически конфигурирует сервер, что значительно упрощает процесс разработки и развертывания.

Каждый из перечисленных компонентов играет важную роль в архитектуре приложения на Java Spring. Модель обеспечивает структуру данных, представление отвечает за их отображение, контроллер обрабатывает запросы, репозитории взаимодействуют с базой данных, сервисы реализуют бизнес-логику, а сервер обеспечивает выполнение приложения. Такое разделение на слои делает приложение модульным, гибким и легко масштабируемым. Кроме того, оно позволяет команде разработчиков работать над разными частями приложения независимо, что повышает эффективность и снижает вероятность ошибок. В совокупности эти компоненты формируют надежную и производительную архитектуру, которая соответствует современным стандартам разработки веб-приложений.

|  |
| --- |
| **Глава 2. Проектная стадия** |
| **2.1. Проектирование общей архитектуры приложения** |

В рамках разработки системы управления заказами еды ключевым элементом является организация данных о блюдах, доступных к заказу. Для эффективного управления пиццами, роллами и другими продуктами была создана иерархия классов на основе **Product**, которая позволяет унифицировать обработку продуктов, сохраняя их специфичные характеристики.

Класс **Product** является абстрактным и служит основой для всех типов продуктов в системе. Он определяет общие атрибуты, такие как название, описание, цена и статус продукта. Также в классе реализована логика для управления статусом продукта, который может быть одним из следующих: NOT\_STARTED, IN\_PROGRESS, READY. Эти статусы используются для отслеживания состояния продукта в процессе приготовления.

**Основные атрибуты класса Product:**

* **id** — уникальный идентификатор продукта.
* **name** — название продукта.
* **description** — описание, включающее ключевые особенности блюда
* **price** — стоимость продукта, которая используется для расчета общей суммы заказа.
* **productStatus** — статус продукта, который может быть одним из значений перечисления ProductStatus.
* **menuId** — идентификатор меню, к которому относится продукт.

Класс Product использует стратегию наследования SINGLE\_TABLE, что означает, что все наследники этого класса будут храниться в одной таблице базы данных с использованием дискриминатора для различения типов продуктов. Это позволяет эффективно работать с разнородными данными, сохраняя целостность структуры.

**Статус готовности:**

Продукты могут находиться в одном из трех статусов, определяемых перечислением **ProductStatus**:

* **NOT\_STARTED** — приготовление не начато.
* **IN\_PROGRESS** — продукт готовится.
* **READY** — продукт готов к выдаче.

Этот статус может автоматически обновляется в процессе выполнения заказа и влияет на общий статус заказа.

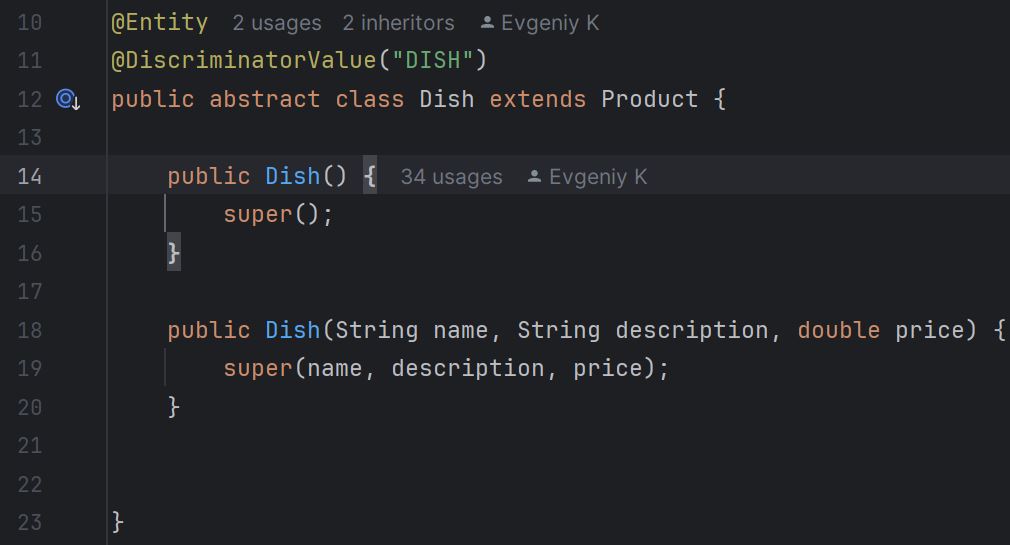


**Наследники класса Product**

Класс Product имеет несколько наследников, которые представляют собой конкретные типы продуктов, такие как пицца и роллы. Каждый из этих классов добавляет свои уникальные атрибуты, которые характерны для конкретного типа продукта.

**Класс Dish**

Класс Dish является абстрактным классом, который наследует от Product. Он представляет собой базовый класс для всех блюд, которые могут быть добавлены в меню. Этот класс добавляет общие атрибуты и методы для всех блюд, таких как пицца и роллы. В дальнейшем этот класс может содержать специфические поля, относящиеся только к пищевым продуктам, например поля содержащее информацию о калорийности, БЖУ, сроки годности и условия хранения.

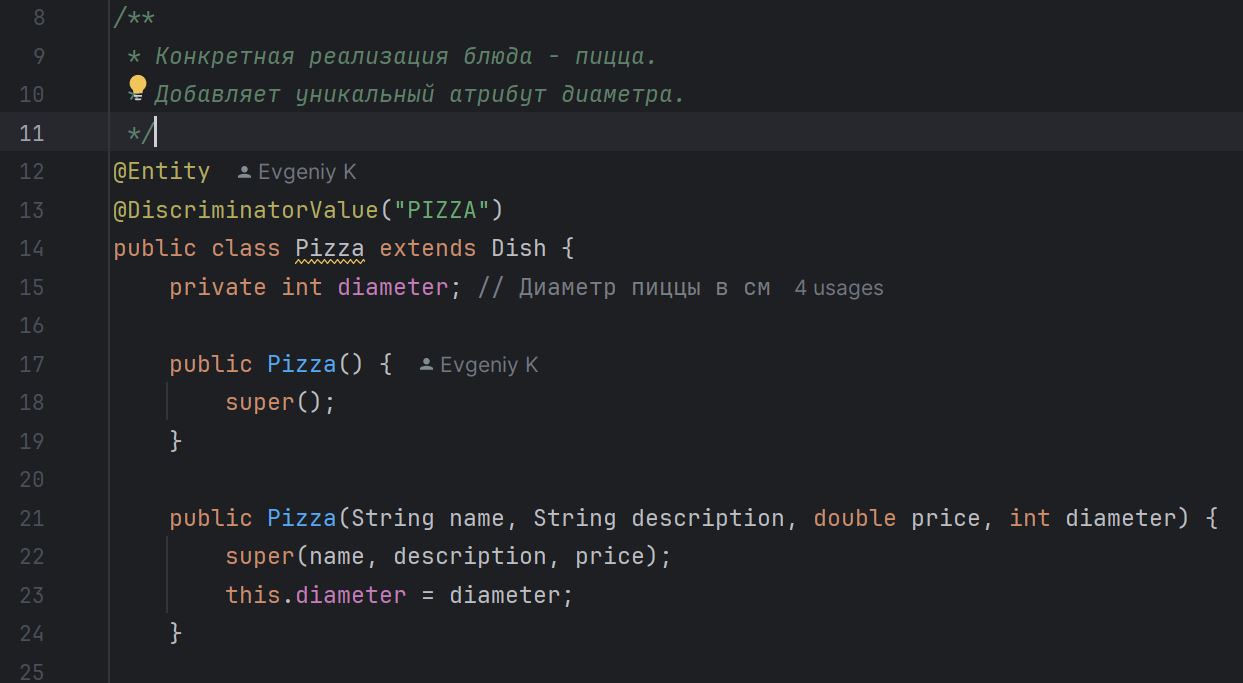


**Класс Pizza**

Класс Pizza наследует от Dish и представляет собой конкретный тип продукта — пиццу. В дополнение к общим атрибутам, унаследованным от Product, класс Pizza добавляет атрибут diameter, который определяет диаметр пиццы.

**Основные атрибуты класса Pizza:**

* **diameter** — диаметр пиццы, указывается в сантиметрах и может в дальнейшем использоваться для уточнения стоимости и времени приготовления.

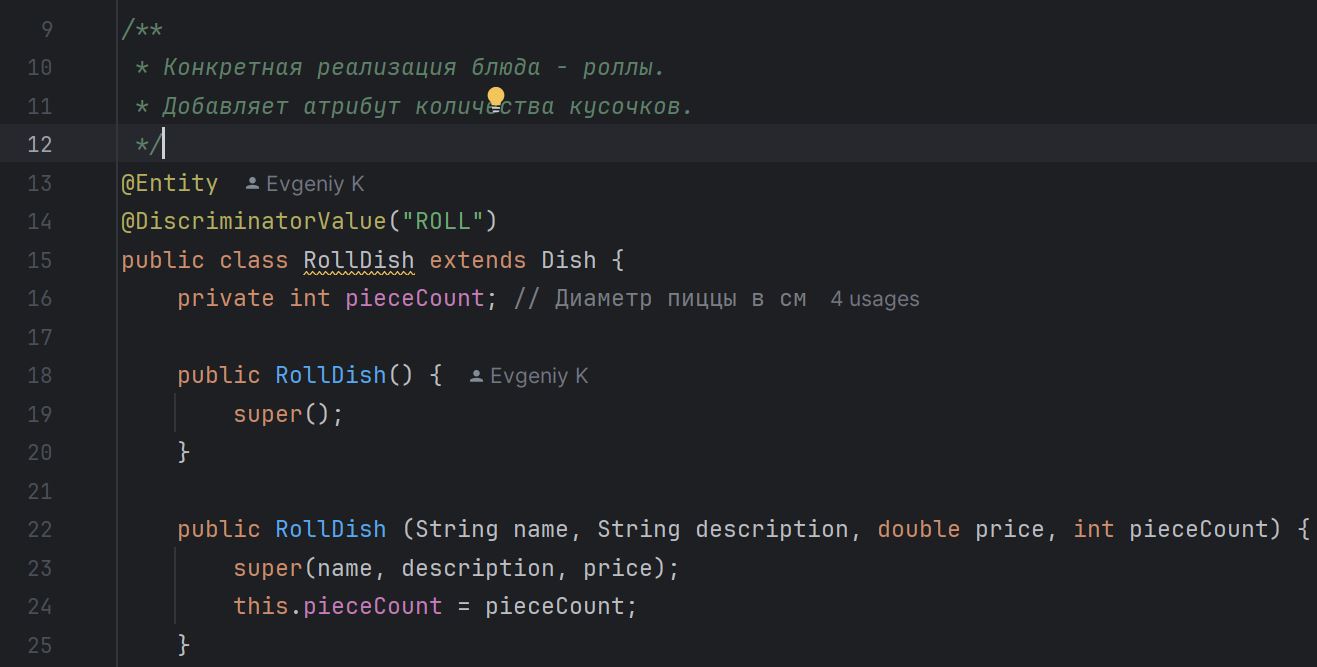


**Класс RollDish**

Класс RollDish также наследует от Dish и представляет собой другой тип продукта — роллы. В дополнение к общим атрибутам, унаследованным от Product, класс RollDish добавляет атрибут pieceCount, который определяет количество кусочков в ролле.

**Основные атрибуты класса RollDish:**

* **pieceCount** — количество кусочков в ролле, и также может в дальнейшем использоваться для уточнения стоимости и времени приготовления.



**Использование классов в системе**

Классы Product, Pizza и RollDish используются в системе для управления продуктами в меню ресторана. Они позволяют создавать, обновлять и удалять продукты, а также отслеживать их статус в процессе приготовления.

Например, при создании нового продукта, такого как пицца, система создает объект класса Pizza, устанавливает его атрибуты (название, описание, цена, диаметр) и сохраняет его в базе данных. Затем, в процессе приготовления, статус продукта может быть изменен с NOT\_STARTED на IN\_PROGRESS, а затем на READY.

Класс Product и его наследники играют ключевую роль в системе управления продуктами в меню ресторана. Они обеспечивают гибкость и расширяемость системы, позволяя добавлять новые типы продуктов с минимальными изменениями в коде. Благодаря использованию наследования и полиморфизма, система может легко адаптироваться к новым требованиям и типам продуктов, что делает ее удобной и эффективной для использования в реальных условиях.

Таким образом, класс Product и его наследники являются важными компонентами системы, которые обеспечивают управление продуктами и их статусами, что позволяет эффективно управлять процессом приготовления блюд в ресторане.

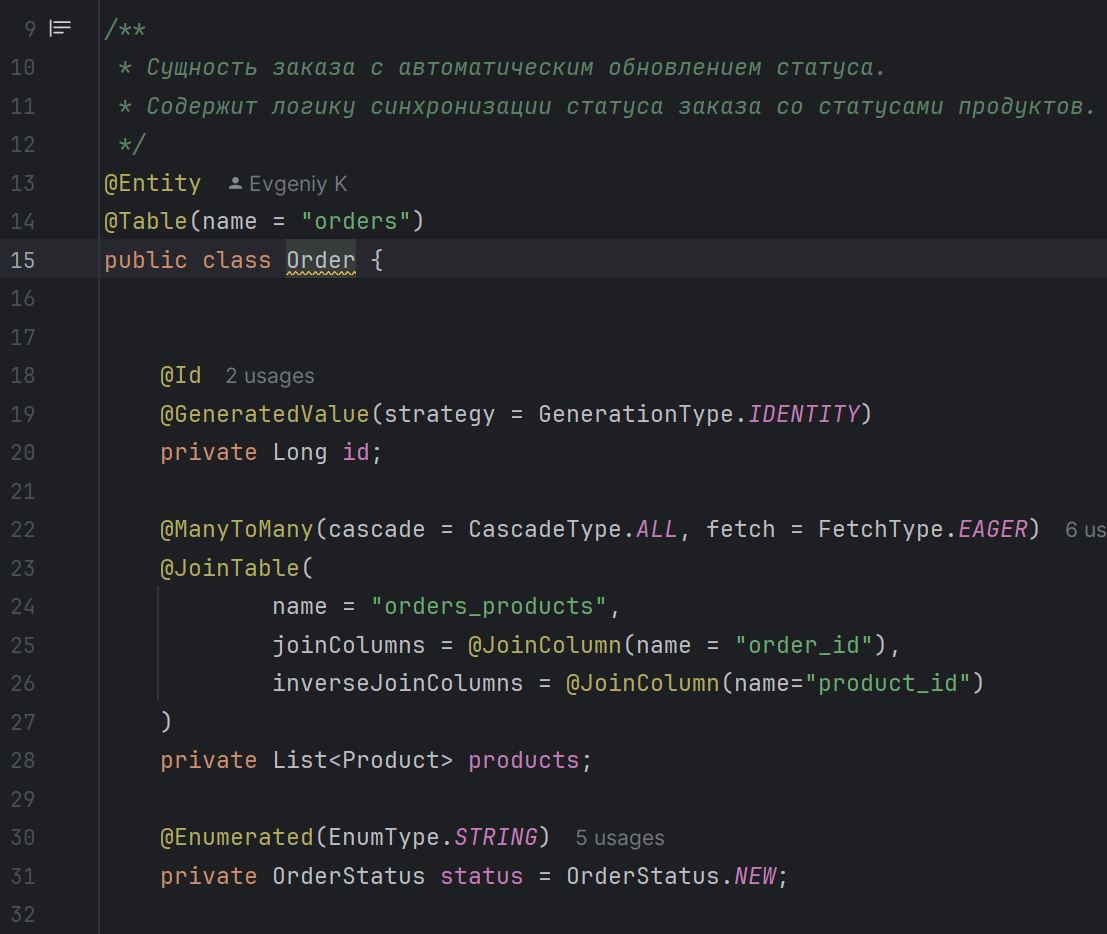
**Класс Order (Заказ)**

Класс Order представляет собой сущность, которая описывает заказ в ресторане. Он содержит информацию о продуктах, которые были заказаны, статусе заказа и других атрибутах, связанных с процессом выполнения заказа. Этот класс является центральным элементом системы, так как он связывает продукты из меню с конкретными заказами клиентов.

**Основные атрибуты класса Order:**

* **id** — уникальный идентификатор заказа. Генерируется автоматически при создании нового заказа.
* **products** — список продуктов, которые были заказаны. Это может быть пицца, роллы или другие блюда. Связь между заказом и продуктами реализована через отношение ManyToMany, что позволяет одному заказу содержать несколько продуктов.
* **status** — статус заказа, который может принимать одно из значений перечисления OrderStatus: NEW (новый), IN\_PROGRESS (в процессе), COMPLETED (завершен). Статус заказа автоматически обновляется в зависимости от статусов продуктов, входящих в заказ.

Когда клиент делает заказ, система создает новый объект класса Order, добавляет в него выбранные продукты и устанавливает начальный статус NEW. По мере приготовления продуктов их статусы обновляются, что автоматически влияет на статус заказа. Когда все продукты готовы, заказ переходит в статус COMPLETED.

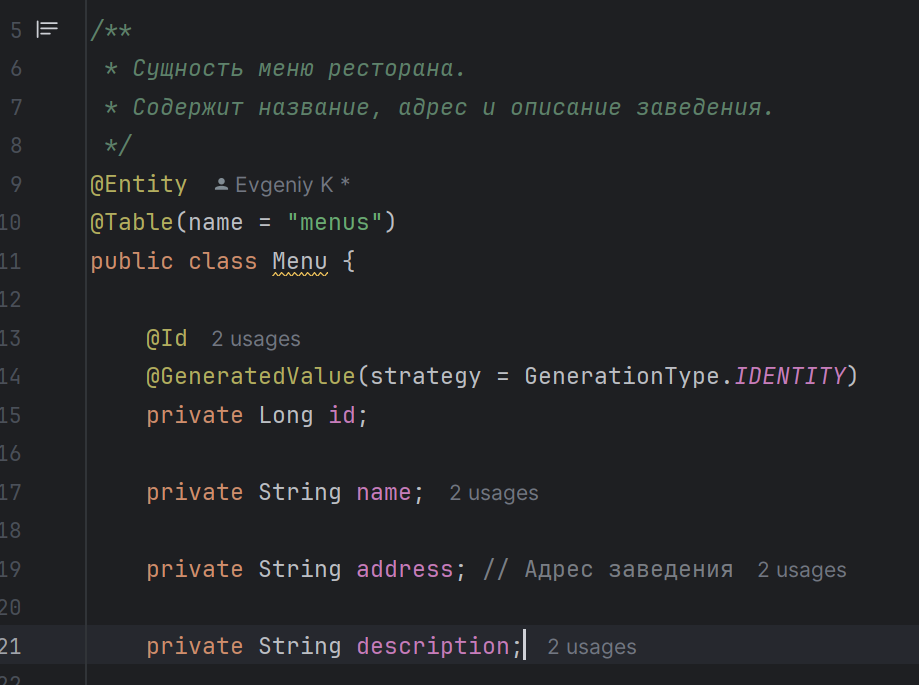


**Класс Menu (Меню)**

Класс Menu представляет собой сущность, которая описывает меню ресторана. Он содержит информацию о доступных блюдах, которые могут быть заказаны клиентами. Меню может включать в себя различные категории блюд, такие как пицца, роллы и другие.

**Основные атрибуты класса Menu:**

1. **id** — уникальный идентификатор меню. Генерируется автоматически при создании нового меню.
2. **name** — название меню. Например, "Основное меню" или "Сезонное меню".
3. **address** — адрес ресторана, к которому относится меню. Это может быть полезно, если у ресторана несколько филиалов.
4. **description** — описание меню, которое может включать информацию о кухне, особенностях блюд и т.д.



**Взаимодействие с другими классами:**

* Класс Menu связан с классом MenuItem через отношение OneToMany. Это означает, что одно меню может содержать множество элементов меню (блюд).
* Класс MenuItem является абстрактным и представляет собой базовый класс который описывает элемент меню, таких как пицца (PizzaMenuItem) и роллы (RollMenuItem).

**Пример использования:**

Когда ресторан создает новое меню, система создает объект класса Menu, устанавливает его атрибуты (название, описание, адрес) и добавляет в него элементы меню. Каждый элемент меню может быть добавлен в заказ клиента.

Классы Order и Menu являются ключевыми элементами системы управления заказами и меню ресторана. Они обеспечивают гибкость и удобство в управлении заказами, статусами и доступными блюдами. Благодаря их взаимодействию с сервисами и другими классами, система позволяет эффективно управлять процессами заказа и приготовления блюд, что делает ее удобной для использования как клиентами, так и персоналом ресторана.

Таким образом, классы Order и Menu играют важную роль в обеспечении функциональности системы, позволяя ресторану эффективно управлять своими заказами и меню.

**Класс MenuItem (Элемент меню)**

Класс MenuItem представляет собой абстрактный базовый класс, который описывает элемент меню. Он содержит общие атрибуты и методы, которые характерны для всех блюд, доступных в меню ресторана. Этот класс используется для хранения информации о блюдах, таких как название, описание, цена и связь с конкретным меню.

**Основные атрибуты класса MenuItem:**

* **id** — уникальный идентификатор элемента меню. Генерируется автоматически при создании нового элемента.
* **name** — название блюда. Например, "Пицца Маргарита" или "Ролл Филадельфия".
* **description** — описание блюда, которое может включать информацию о составе, ингредиентах и т.д.
* **price** — цена блюда.
* **menu** — связь с объектом класса Menu. Каждый элемент меню принадлежит определенному меню, что позволяет организовать структуру меню ресторана.



**Функциональные отличия между классами MenuItem и Product**

Классы MenuItem и Product имеют схожие атрибуты, такие как название, описание и цена, но они выполняют разные роли в системе и имеют функциональные отличия.

**1. Назначение классов:**

* **MenuItem** — представляет элемент меню, который может быть заказан клиентом. Этот класс используется для хранения информации о блюдах, доступных в меню ресторана.
* **Product** — представляет продукт, который может быть частью заказа. Этот класс используется для управления продуктами в процессе заказа и приготовления.

**2. Связь с меню:**

* **MenuItem** — имеет явную связь с классом Menu через атрибут menu. Каждый элемент меню принадлежит определенному меню.
* **Product** — не имеет прямой связи с меню. Вместо этого он имеет атрибут menuId, который указывает на меню, к которому относится продукт.

**3. Статус продукта:**

* **MenuItem** — не имеет атрибута статуса, так как он представляет собой элемент меню, а не конкретный заказанный продукт.
* **Product** — имеет атрибут productStatus, который используется для отслеживания статуса продукта в процессе приготовления (например, NOT\_STARTED, IN\_PROGRESS, READY).

**4. Использование в заказе:**

* **MenuItem** — используется для отображения блюд в меню и выбора их клиентом. Когда клиент выбирает блюдо из меню, создается объект класса Product, который добавляется в заказ.
* **Product** — используется для управления продуктами в заказе. Каждый продукт в заказе имеет свой статус, который влияет на общий статус заказа.

**5. Наследование:**

* **MenuItem** — является абстрактным классом, который имеет наследников, таких как PizzaMenuItem и RollMenuItem. Эти наследники добавляют специфические атрибуты для конкретных типов блюд.
* **Product** — также является абстрактным классом, но его наследники (например, Pizza и RollDish) используются для представления продуктов в заказе, а не в меню.

**Пример использования классов MenuItem и Product**

* **Создание меню**:
  + Администратор создает новое меню и добавляет в него элементы, такие как пицца (PizzaMenuItem) и роллы (RollMenuItem).
* **Оформление заказа**:
  + Клиент выбирает блюда из меню, и система создает объекты класса Product на основе выбранных элементов меню. Эти продукты добавляются в заказ.
* **Приготовление блюд**:
  + Повар начинает готовить блюда, и статусы продуктов (Product) обновляются. Это влияет на статус заказа.

6. **Заключение:**

Класс MenuItem и его наследники (PizzaMenuItem, RollMenuItem) играют важную роль в системе управления меню ресторана. Они позволяют хранить информацию о блюдах, доступных в меню, и обеспечивают гибкость при добавлении новых типов блюд. В то же время класс Product используется для управления продуктами в заказе и отслеживания их статусов в процессе приготовления.

Функциональные отличия между классами MenuItem и Product заключаются в их назначении: MenuItem используется для представления блюд в меню, а Product — для управления продуктами в заказе. Эти классы работают вместе, обеспечивая эффективное управление заказами и меню в ресторане.

**Наследники класса MenuItem**

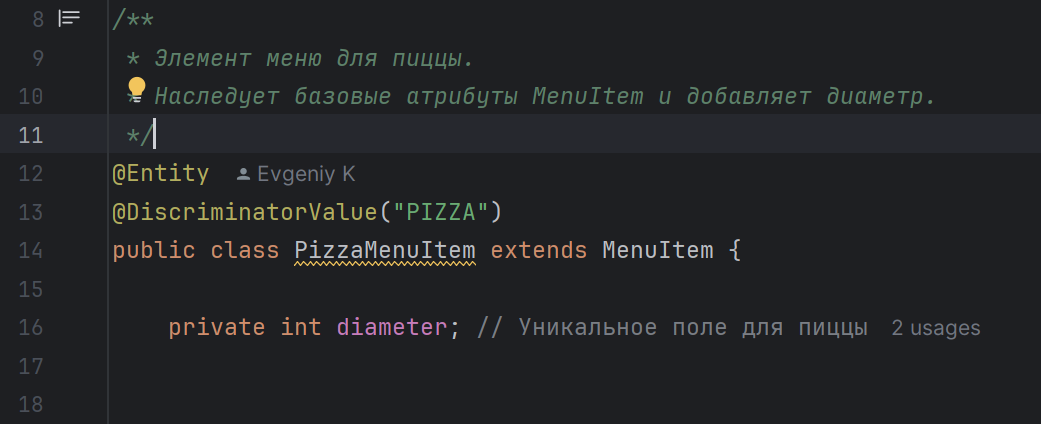
Класс MenuItem имеет несколько наследников, которые представляют собой конкретные типы блюд. Каждый из этих классов добавляет свои уникальные атрибуты, которые характерны для конкретного типа блюда.

**Класс PizzaMenuItem (Элемент меню: Пицца)**

Класс PizzaMenuItem наследует от MenuItem и представляет собой элемент меню, который описывает пиццу. В дополнение к общим атрибутам, унаследованным от MenuItem, класс PizzaMenuItem добавляет атрибут diameter, который определяет диаметр пиццы.

**Основные атрибуты класса PizzaMenuItem:**

* **diameter** — диаметр пиццы. Это уникальный атрибут, который отличает пиццу от других блюд.

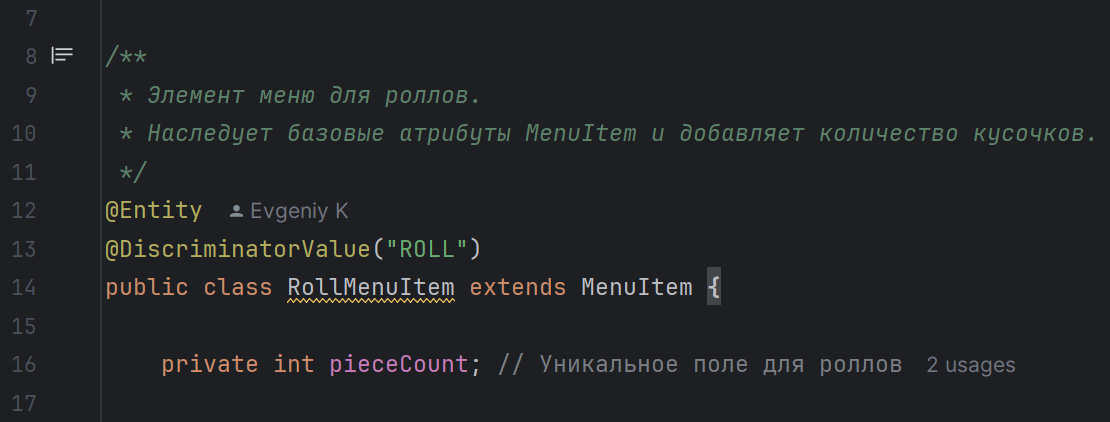


**Класс RollMenuItem (Элемент меню: Ролл)**

Класс RollMenuItem также наследует от MenuItem и представляет собой элемент меню, который описывает роллы. В дополнение к общим атрибутам, унаследованным от MenuItem, класс RollMenuItem добавляет атрибут pieceCount, который определяет количество кусочков в ролле.

**Основные атрибуты класса RollMenuItem:**

* **pieceCount** — количество кусочков в ролле. Это уникальный атрибут, который отличает роллы от других блюд.



**Класс MainController**

Класс MainController является центральным компонентом системы управления меню и заказами ресторана. Он отвечает за обработку HTTP-запросов, взаимодействие с пользовательским интерфейсом и координацию работы сервисов, связанных с меню, заказами и продуктами. В этой части мы рассмотрим структуру MainController, его ключевые методы, а также взаимодействие с сервисами MenuService, OrderService и ProductService. Основное внимание уделяется функциональности контроллера и его роли в системе.

MainController — это класс, аннотированный как @Controller в Spring Framework, что указывает на его роль в обработке веб-запросов и управлении представлениями (HTML-страницами). Он использует три основных сервиса:

1. **MenuService** — управление меню и элементами меню.
2. **OrderService** — обработка заказов и их статусов.
3. **ProductService** — работа с продуктами (в текущей реализации используется косвенно).

Контроллер предназначен для:

* Отображения главной страницы с доступными меню.
* Показа конкретного меню с его блюдами.
* Отображения списка заказов.
* Управления сессией пользователя (например, сохранение выбранного меню).

**Ключевые методы MainController:**

**1. Метод home() — Главная страница**

* **URL**: / (корневой путь).
* **Функциональность**:
  + Получает список всех меню через MenuService.getAllMenus().
  + Передает данные в модель для отображения на странице index.html.
  + Отображает приветственное сообщение для пользователя.
* **Роль сервисов**:
  + MenuService предоставляет данные о меню, что позволяет пользователю выбирать из доступных вариантов.

**2. Метод menu() — Страница меню**

* **URL**: /menu.
* **Функциональность**:
  + Принимает параметр mid (ID меню) или использует сохраненный в сессии currentMenuId.
  + Получает элементы выбранного меню через MenuService.getMenuItems(menuId).
  + Управляет корзиной покупок, хранящейся в сессии пользователя.
  + Рассчитывает общую стоимость товаров в корзине.
  + Передает данные в модель для отображения на странице menu.html.
* **Роль сервисов**:
  + MenuService обеспечивает загрузку элементов меню.
  + OrderService косвенно участвует через корзину, которая может быть преобразована в заказ.

**3. Метод viewOrders() — Просмотр заказов**

* **URL**: /view.
* **Функциональность**:
  + Получает список всех заказов с итоговой суммой через OrderService.getAllOrdersWithTotals().
  + Отображает страницу orders.html с деталями заказов.
* **Роль сервисов**:
  + OrderService предоставляет данные о заказах, включая их статусы и суммы.

**Примеры сценариев работы**

**Сценарий 1: Просмотр меню**

1. Пользователь заходит на главную страницу (/).
2. MainController запрашивает у MenuService список всех меню.
3. Пользователь выбирает меню, нажимая на ссылку с параметром mid.
4. Контроллер сохраняет mid в сессии и загружает элементы меню через MenuService.getMenuItems().
5. На странице menu.html отображаются блюда выбранного меню и текущая корзина.

**Сценарий 2: Оформление заказа**

1. Пользователь добавляет блюда в корзину, которая хранится в сессии.
2. При переходе к оформлению заказа корзина конвертируется в продукты через convertMenuItemsToProducts.
3. OrderService создает новый заказ и обновляет его статусы по мере приготовления блюд.

**Сценарий 3: Просмотр заказов**

1. Администратор переходит на страницу /view.
2. MainController запрашивает у OrderService список всех заказов с суммами.
3. На странице orders.html отображаются детали заказов, включая названия ресторанов и адреса.

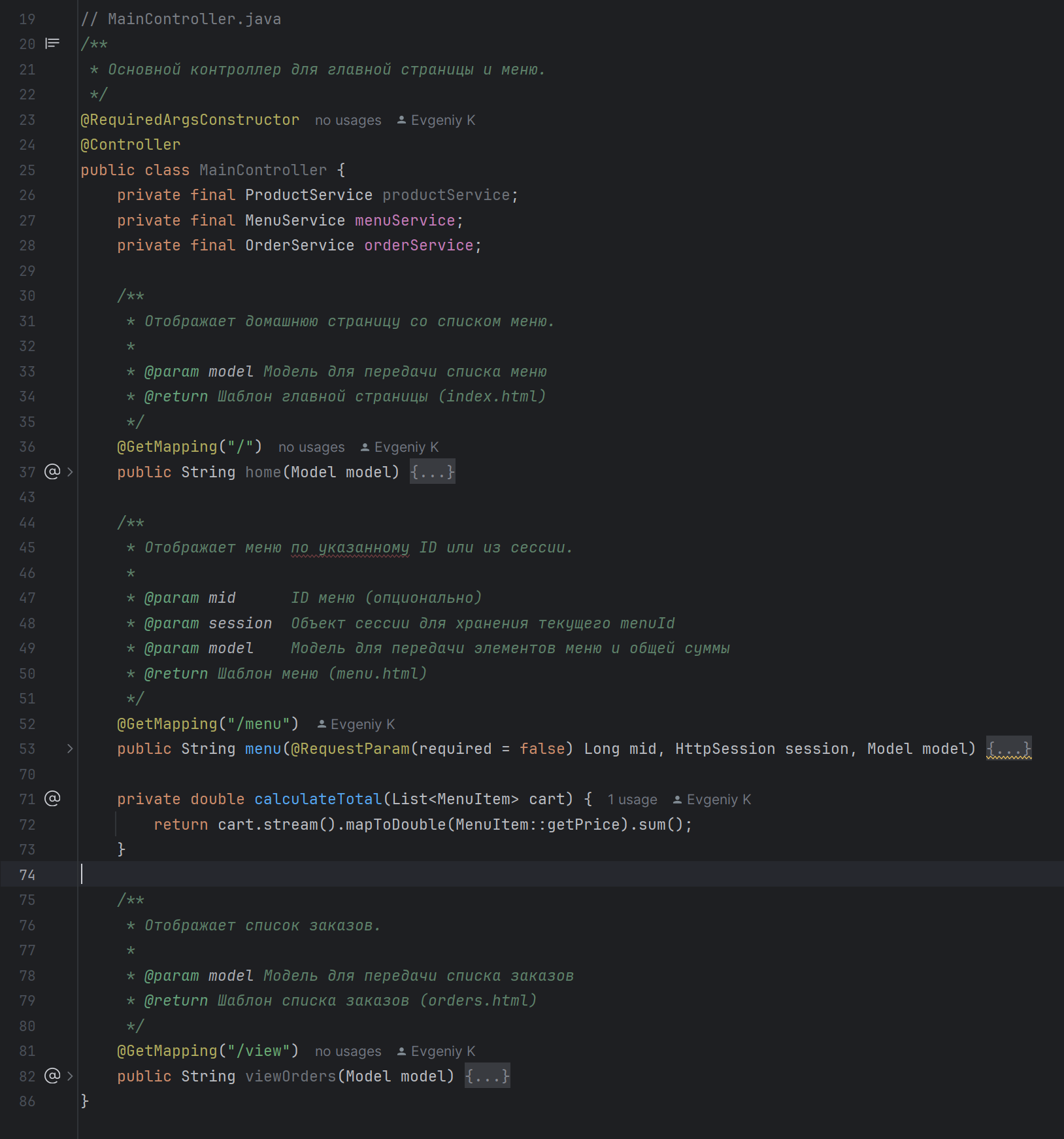
**Функциональные особенности**

1. **Управление сессией**:
   * Контроллер сохраняет currentMenuId в сессии, что позволяет пользователю возвращаться к выбранному меню без повторного выбора.
   * Корзина покупок хранится в сессии, обеспечивая непрерывность взаимодействия.
2. **Разделение ответственности**:
   * MainController фокусируется на обработке запросов и передаче данных в представления.
   * Бизнес-логика (работа с меню, заказами, продуктами) делегирована сервисам.
3. **Гибкость отображения**:

Использование Thymeleaf (предположительно) позволяет динамически формировать HTML-страницы на основе данных из модели.

**Взаимодействие с сервисами**

Сервисы MenuService, OrderService и ProductService разделяют ответственность за управление данными, что делает систему модульной и легко расширяемой. MainController объединяет их функциональность, предоставляя пользователю единый интерфейс для работы с меню и заказами. Это позволяет ресторану эффективно управлять своими процессами, от формирования меню до отслеживания выполнения заказов.



**Сервис OrderService**

Сервис OrderService отвечает за управление заказами в системе. Он предоставляет методы для создания, обновления, удаления и получения информации о заказах. Этот сервис играет важную роль в процессе оформления заказов и отслеживания их статусов.

**Основные функции OrderService:**

1. **Создание заказа**:
   * Метод createOrder(Order order) создает новый заказ и сохраняет его в базе данных.
   * Заказ может содержать несколько продуктов, которые добавляются через метод addProductFromMenu.
2. **Получение заказа**:

Метод getOrderById(Long id) возвращает заказ по его идентификатору. Если заказ не найден, выбрасывается исключение.

1. **Получение всех заказов**:

Метод getAllOrders() возвращает список всех заказов, что полезно для администратора ресторана.

1. **Удаление заказа**:

Метод deleteOrder(Long id) удаляет заказ по его идентификатору. Если заказ не найден, выбрасывается исключение.

1. **Обновление заказа**:

Метод updateOrder(Order order) обновляет информацию о заказе, например, его статус или список продуктов.

1. **Добавление продукта из меню в заказ**:

Метод addProductFromMenu(Long orderId, Long menuItemId) добавляет продукт из меню в заказ. Продукт создается на основе элемента меню и добавляется в заказ.

1. **Расчет общей суммы заказа**:

Метод calculateOrderTotal(Long orderId) рассчитывает общую сумму заказа на основе цен продуктов, входящих в заказ.

1. **Преобразование заказа в DTO**:

Метод orderToOrderResponse(Order orderById, double total) преобразует заказ в объект OrderResponse, который содержит информацию о заказе, его статусе, продуктах и общей сумме.

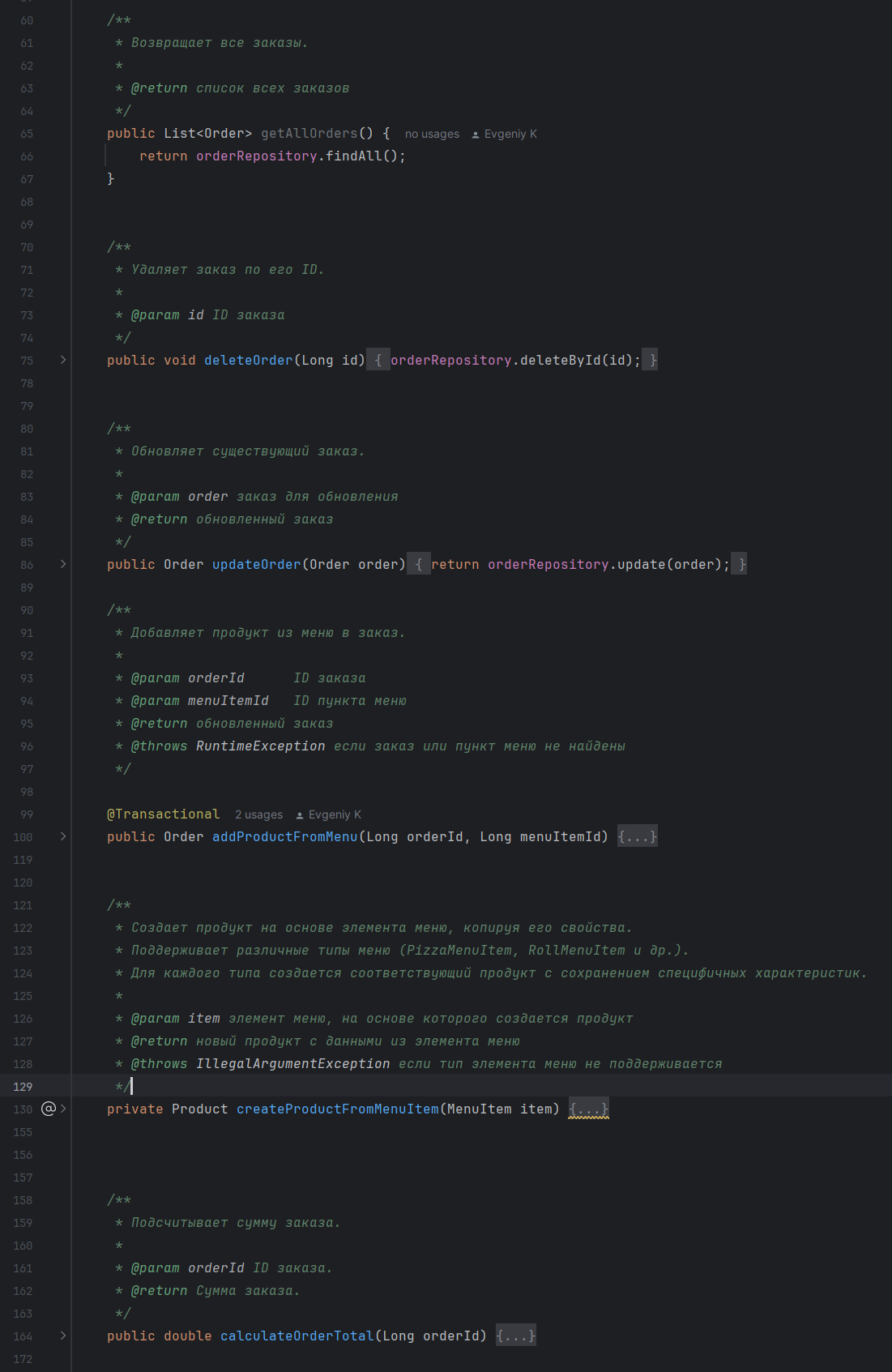
**Взаимодействие с другими компонентами:**

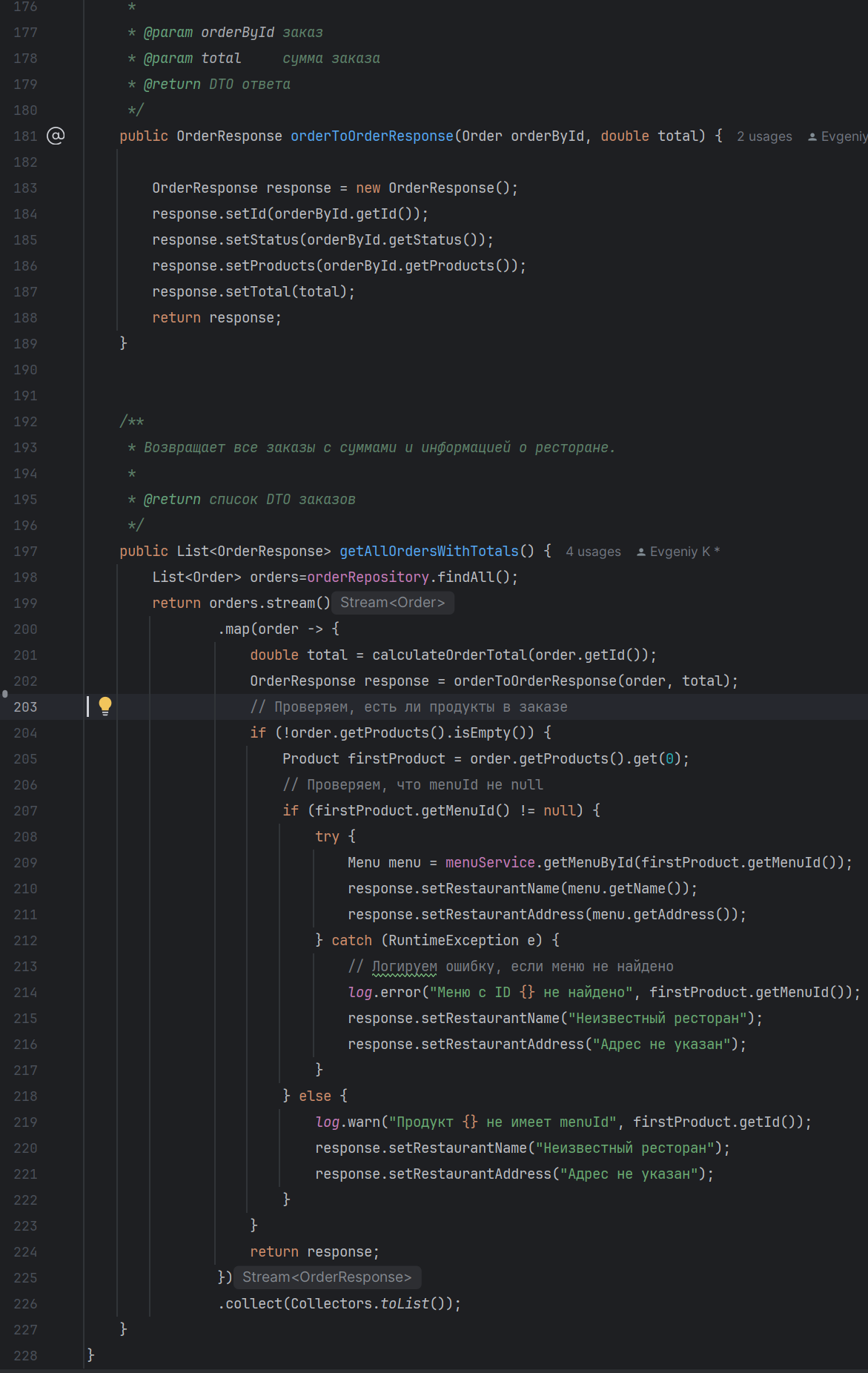
* **MenuService**: Используется для получения информации о продуктах, которые добавляются в заказ.
* **ProductService**: Используется для создания продуктов на основе элементов меню и управления их статусами.

**Пример использования:**

Когда клиент оформляет заказ, система создает новый заказ через OrderService.createOrder(). Затем продукты из меню добавляются в заказ через addProductFromMenu. По мере приготовления блюд их статусы обновляются, что автоматически влияет на статус заказа.







**Сервис ProductService**

Сервис ProductService отвечает за управление продуктами в системе. Продукты представляют собой блюда, которые могут быть добавлены в заказ. Этот сервис предоставляет методы для создания, обновления, удаления и получения информации о продуктах.

**Основные функции ProductService:**

1. **Создание продукта**:

Метод createProduct(Product product) создает новый продукт и сохраняет его в базе данных. Если продукт с таким ID уже существует, выбрасывается исключение.

1. **Получение продукта**:

Метод getProductById(Long id) возвращает продукт по его идентификатору. Если продукт не найден, выбрасывается исключение.

1. **Получение всех продуктов**:

Метод getAllProducts() возвращает список всех продуктов.

1. **Удаление продукта**:

Метод deleteProduct(Long id) удаляет продукт по его идентификатору. Если продукт не найден, выбрасывается исключение.

1. **Обновление продукта**:

Метод updateProduct(Product product) обновляет информацию о продукте, например, его название, описание или цену.

1. **Проверка существования продукта**:

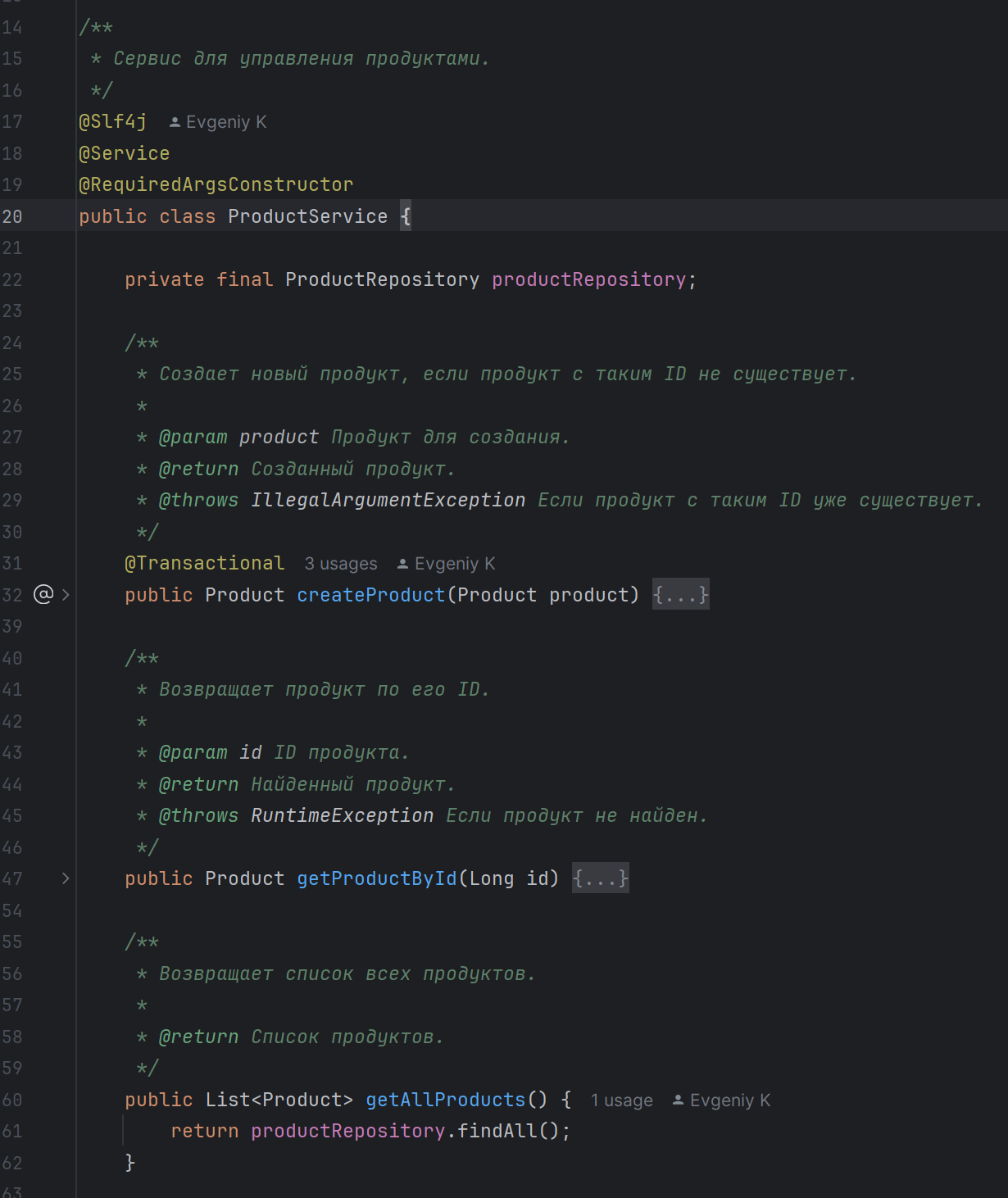
Метод existsById(Long id) проверяет, существует ли продукт с указанным идентификатором.

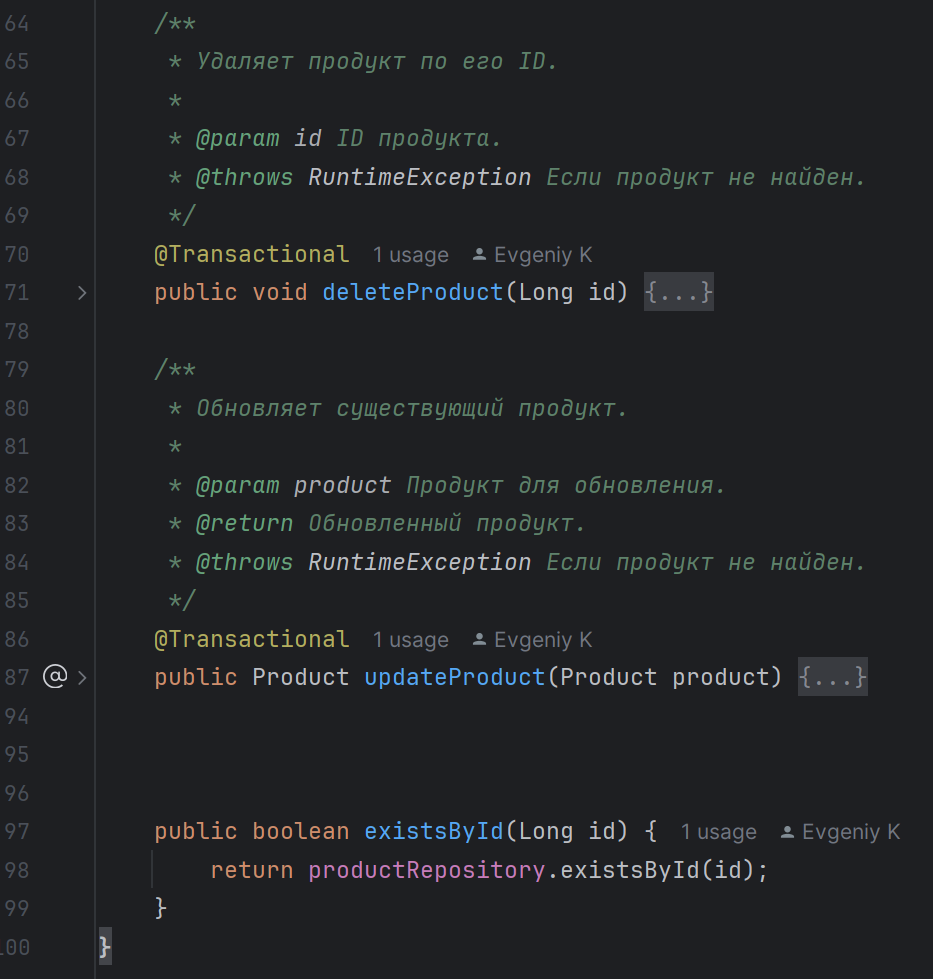
**Взаимодействие с другими компонентами:**

* **OrderService**: Используется для добавления продуктов в заказы и управления их статусами.
* **MenuService**: Используется для создания продуктов на основе элементов меню.

**Пример использования:**

Когда клиент выбирает блюдо из меню, система создает продукт через ProductService.createProduct() и добавляет его в заказ. По мере приготовления блюда его статус обновляется через updateProduct.





**Сервис MenuService**

Сервис MenuService отвечает за управление меню и элементами меню в системе. Он предоставляет методы для создания, обновления, удаления и получения информации о меню и его элементах.

**Основные функции MenuService:**

1. **Создание меню**:

Метод createMenu(Menu menu) создает новое меню и сохраняет его в базе данных.

1. **Добавление элемента в меню**:

Метод addItemToMenu(Long menuId, MenuItem item) добавляет элемент меню (например, пиццу или ролл) в указанное меню.

1. **Получение элементов меню**:

Метод getMenuItems(Long menuId) возвращает список элементов меню для указанного меню.

1. **Получение всех меню**:

Метод getAllMenus() возвращает список всех меню.

1. **Получение меню по ID**:

Метод getMenuById(Long menuId) возвращает меню по его идентификатору. Если меню не найдено, выбрасывается исключение.

1. **Получение элемента меню по ID**:

Метод getMenuItemById(Long menuItemId) возвращает элемент меню по его идентификатору.

**Взаимодействие с другими компонентами:**

* **OrderService**: Используется для добавления элементов меню в заказы.
* **ProductService**: Используется для создания продуктов на основе элементов меню.

**Пример использования:**

Когда администратор создает новое меню, он использует MenuService.createMenu(). Затем он добавляет блюда в меню через addItemToMenu. Когда клиент выбирает блюдо из меню, система создает продукт на основе элемента меню и добавляет его в заказ



**Взаимодействие сервисов**

Сервисы OrderService, ProductService и MenuService тесно взаимодействуют друг с другом для обеспечения работы системы. Рассмотрим пример взаимодействия:

1. **Создание заказа**:
   * Клиент выбирает блюда из меню.
   * MenuService предоставляет информацию о выбранных блюдах.
   * ProductService создает продукты на основе элементов меню.
   * OrderService создает новый заказ и добавляет в него продукты.
2. **Обновление статуса заказа**:
   * Повар начинает готовить блюда, и их статусы обновляются через ProductService.
   * OrderService автоматически обновляет статус заказа на основе статусов продуктов.
3. **Управление меню**:
   * Администратор добавляет новые блюда в меню через MenuService.
   * Эти блюда становятся доступными для заказа через OrderService.

В заключении осервисах OrderService, ProductService и MenuService нужно сказать, что ониявляются ключевыми компонентами системы управления заказами и меню ресторана. Они обеспечивают:

* Управление заказами и их статусами (OrderService).
* Управление продуктами и их статусами (ProductService).
* Управление меню и его элементами (MenuService).

Эти сервисы работают вместе, обеспечивая гибкость и удобство в управлении заказами, продуктами и меню. Благодаря их взаимодействию система позволяет эффективно управлять процессами заказа и приготовления блюд, что делает ее удобной для использования как клиентами, так и персоналом ресторана.

**Взаимодействие MainController c CartController**

CartController и MainController тесно связаны, так как они управляют разными аспектами процесса заказа. Рассмотрим их взаимодействие:

**1. Переход между меню и корзиной:**

* Пользователь просматривает меню через MainController.menu().
* При добавлении товара в корзину через CartController.addToCart() пользователь перенаправляется обратно на страницу меню.
* Это обеспечивает плавный переход между просмотром меню и управлением корзиной.

**2. Общая сессия:**

Оба контроллера используют сессию для хранения данных:

* + MainController сохраняет currentMenuId (ID текущего меню).
  + CartController сохраняет корзину покупок.

Это позволяет пользователю легко переключаться между меню и корзиной без потери данных.

**3. Оформление заказа:**

* После добавления товаров в корзину через CartController пользователь может оформить заказ.
* CartController взаимодействует с OrderService для создания заказа, а MainController может отображать список заказов через метод viewOrders().

**Контроллер CartController**

Контроллер CartController играет важную роль в системе управления заказами и корзиной покупок ресторана. Он отвечает за обработку действий, связанных с добавлением и удалением товаров в корзину, а также за оформление заказов. Взаимодействие CartController с MainController обеспечивает плавный переход между просмотром меню и управлением корзиной, что делает процесс заказа удобным для пользователя. В этой части мы рассмотрим функциональность CartController и его ключевые методы.

CartController — это класс, аннотированный как @Controller в Spring Framework. Он отвечает за управление корзиной покупок, которая хранится в сессии пользователя. Корзина представляет собой список элементов меню (MenuItem), которые пользователь выбрал для заказа. Контроллер взаимодействует с сервисами OrderService и MenuService для выполнения операций, связанных с корзиной и заказами.

**Основные функции CartController:**

1. Добавление товаров в корзину.
2. Удаление товаров из корзины.
3. Оформление заказа.
4. Отображение содержимого корзины.

**Ключевые методы CartController:**

**1. Метод viewCart() — Просмотр корзины**

* URL: /cart.
* Функциональность:
  + Получает текущую корзину из сессии пользователя.
  + Рассчитывает общую стоимость товаров в корзине.
  + Передает данные в модель для отображения на странице cart.html.

**2. Метод addToCart() — Добавление товара в корзину**

* URL: /cart/add.
* Функциональность:
  + Принимает menuItemId (ID элемента меню) и добавляет соответствующий товар в корзину.
  + Корзина хранится в сессии пользователя.
  + После добавления товара пользователь перенаправляется на страницу меню.
* Роль сервисов:
  + MenuService используется для получения информации о товаре по его ID.

**3. Метод removeFromCart() — Удаление товара из корзины**

* URL: /cart/remove.
* Функциональность:
  + Принимает menuItemId и удаляет соответствующий товар из корзины.
  + Корзина обновляется в сессии пользователя.
  + После удаления товара пользователь перенаправляется на страницу меню.
* Роль сервисов:
  + Сервисы не используются напрямую, так как корзина управляется через сессию.

**4. Метод checkout() — Оформление заказа**

* URL: /cart/checkout.
* Функциональность:
  + Преобразует товары в корзине в продукты (Product) через метод convertMenuItemsToProducts.
  + Создает новый заказ через OrderService.createOrder().
  + Очищает корзину после успешного оформления заказа.
  + Перенаправляет пользователя на страницу меню с сообщением об успешном оформлении заказа.
* Роль сервисов:
  + OrderService используется для создания заказа.
  + MenuService косвенно участвует через преобразование элементов меню в продукты.

**5. Метод viewAllOrders() — Просмотр всех заказов**

* URL: /cart/all.
* Функциональность:
  + Получает список всех заказов с итоговой суммой через OrderService.getAllOrdersWithTotals().
  + Отображает страницу orders.html с деталями заказов.
* Роль сервисов:
  + OrderService предоставляет данные о заказах.

**Пример сценария работы**

**Сценарий: Оформление заказа**

1. Пользователь просматривает меню через MainController.menu().
2. Пользователь добавляет товары в корзину через CartController.addToCart().
3. Корзина сохраняется в сессии, и пользователь продолжает выбирать товары.
4. Пользователь переходит к оформлению заказа через CartController.checkout().
5. Товары из корзины преобразуются в продукты, и создается новый заказ через OrderService.
6. Пользователь перенаправляется на страницу меню с сообщением об успешном оформлении заказа.

**Функциональные особенности**

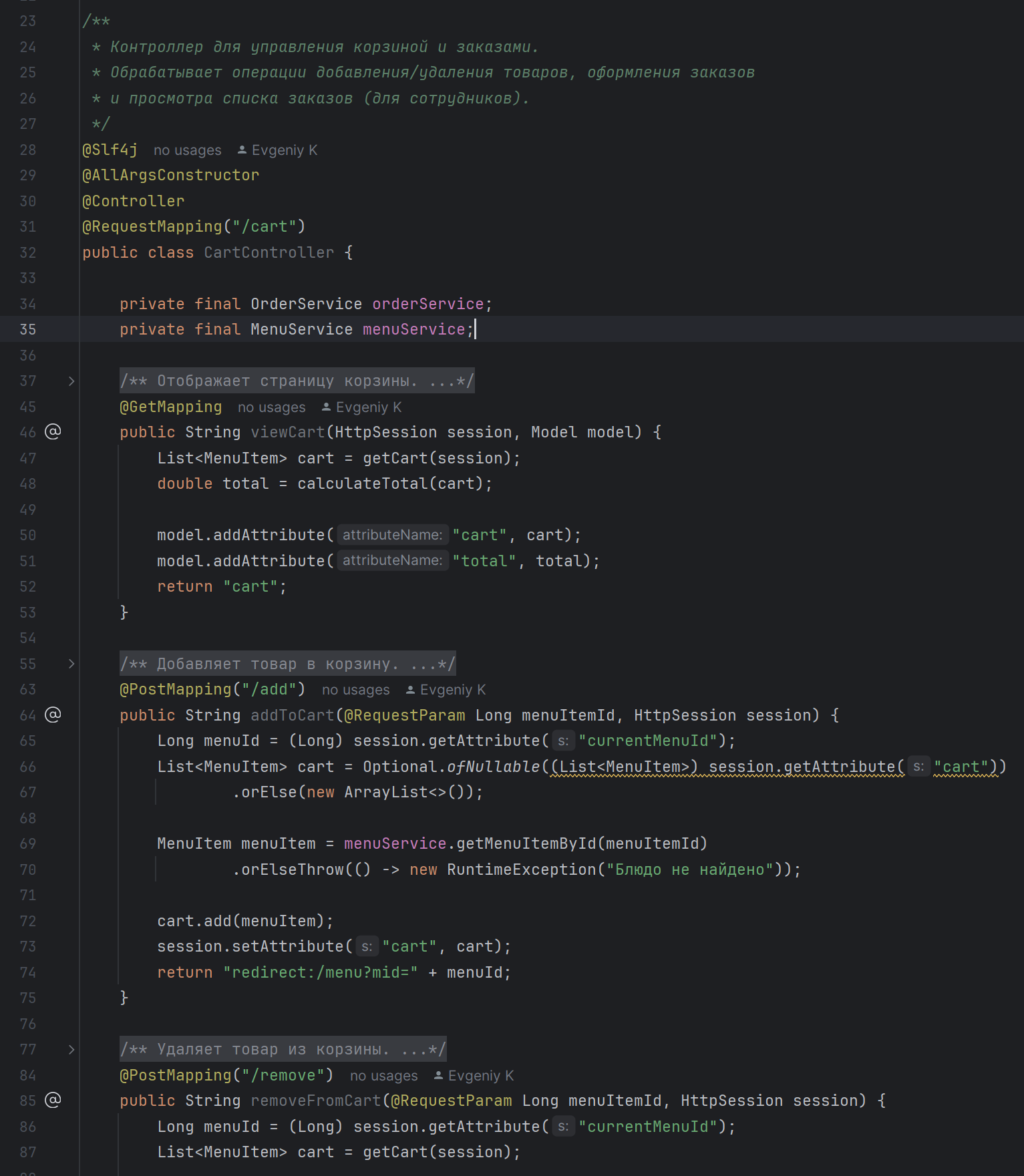
**1. Управление сессией**:

Корзина и текущее меню хранятся в сессии, что обеспечивает непрерывность взаимодействия.Это позволяет пользователю добавлять товары в корзину и возвращаться к меню без потери данных.

**2**. **Разделение ответственности**:

MainController отвечает за отображение меню и заказов. CartController управляет корзиной и процессом оформления заказа. Такое разделение делает код более модульным и поддерживаемым.

3. **Гибкость отображения**:

Использование Thymeleaf (предположительно) позволяет динамически формировать HTML-страницы на основе данных из модели.





**Контроллер ProductController**

ProductController — это REST-контроллер, который управляет продуктами, которые представляют собой экземпляры блюд, добавленные в заказы. Он обеспечивает создание, обновление и удаление продуктов, а также их связь с меню.

**Основные функции:**

* Создание, обновление и удаление продуктов (пицц, роллов).
* Преобразование данных из DTO в сущности для сохранения в БД.
* Интеграция с административной панелью для управления меню.

**Ключевые методы:**

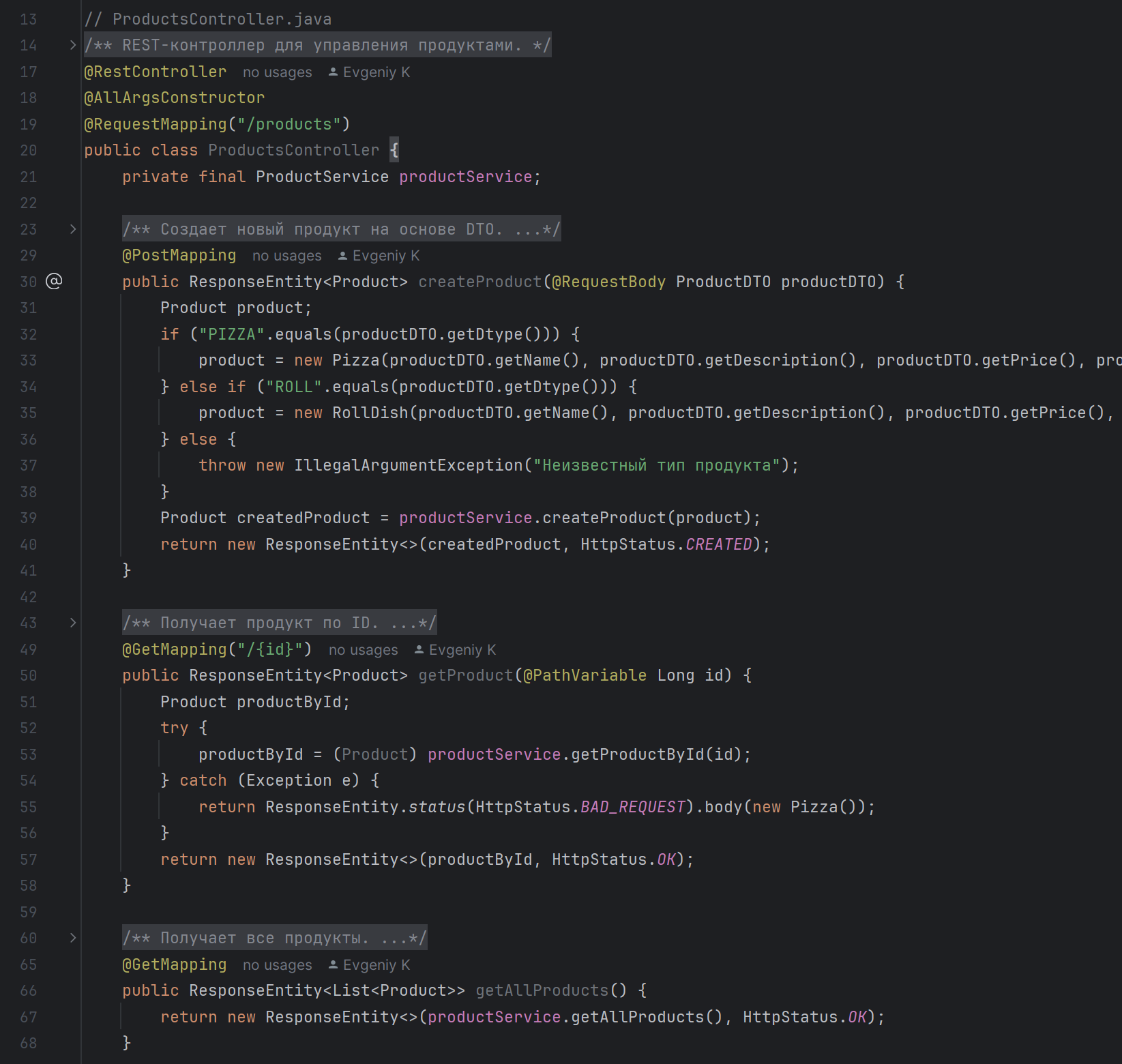
1. **Метод createProduct()** 
   * **URL:** /products (POST).
   * **Функциональность:**   
     Принимает ProductDTO, проверяет тип продукта (PIZZA/ROLL), конвертирует его в сущность (Pizza/RollDish) и сохраняет в БД.
   * **Пример сценария:** Администратор добавляет новую пиццу «Маргарита» с диаметром 30 см через форму администрирования. Данные передаются в ProductDTO, сохраняются в БД.
2. **Метод getAllProducts()** 
   * **URL:** /products (GET).
   * **Функциональность:**   
     Возвращает список всех продуктов для административной панели.
   * **Пример сценария:** Администратор может просматривать все блюда находящиеся на данный момент в заказах.

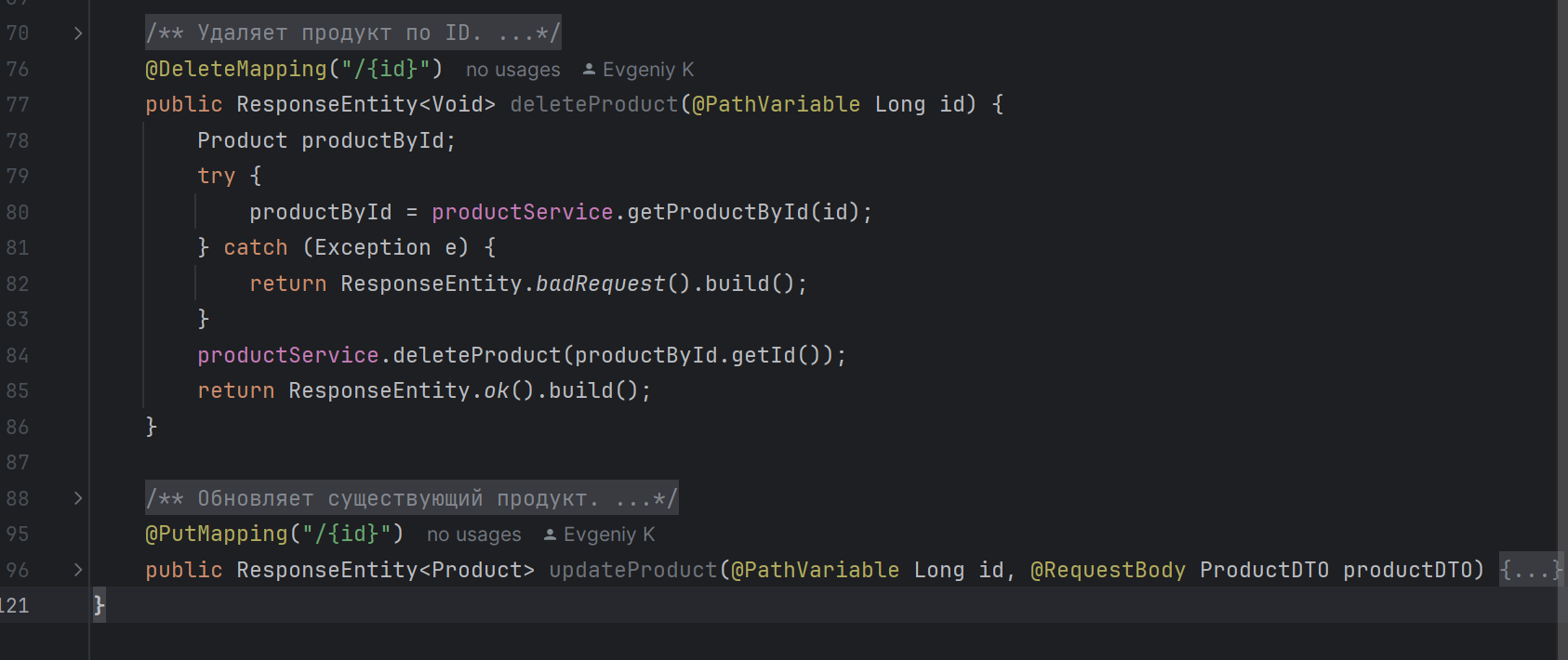
**Функциональные особенности:**

* Использует **DTO** для безопасной передачи данных между клиентом и сервером.
* Работает с сервисом ProductService, который управляет CRUD-операциями.
* Доступ к методам разрешен только для роли **ADMIN** (настраивается в SecurityConfig).

**Пример сценария:**

Когда клиент добавляет пиццу из меню в корзину, ProductController создаёт объект Pizza на основе PizzaMenuItem и сохраняет его через ProductService. При оформлении заказа этот продукт связывается с заказом через OrderService.





**Контроллер OrderController**

OrderController — это REST-контроллер, который отвечает за управление заказами: их создание, отслеживание статусов и взаимодействие с продуктами. Этот контроллер обеспечивает связь между клиентскими запросами и бизнес-логикой обработки заказов.

**Основные функции:**

* Создание и обновление заказов.
* Синхронизация статусов заказов с продуктами.
* Интеграция с корзиной и меню.

**Ключевые методы:**

1. **Метод createOrder()** 
   * **URL:** /orders (POST).
   * **Функциональность:**   
     Создает заказ на основе данных из корзины, конвертируя пункты меню (MenuItem) в продукты (Product).
2. **Метод updateOrderStatus()** 
   * **URL:** /orders/{id}/status (PUT).
   * **Функциональность:**   
     Обновляет статус заказа (например, с **IN\_PROGRESS** на **COMPLETED** ).
   * **Пример сценария:**   
     Сотрудник кухни завершает приготовление заказа, что автоматически переводит его статус в **COMPLETED** .
3. **Метод getAllOrders() — Получение всех заказов**

* **URL**: /orders.
* **Функциональность**:
  + Возвращает список всех заказов с расчетом их общей стоимости.

1. **Метод getOrder() — Получение заказа по ID**
   * **URL: /orders/{id}.**
   * **Функциональность:**

Получает заказ по его ID.Возвращает заказ с расчетом общей стоимости в формате JSON.

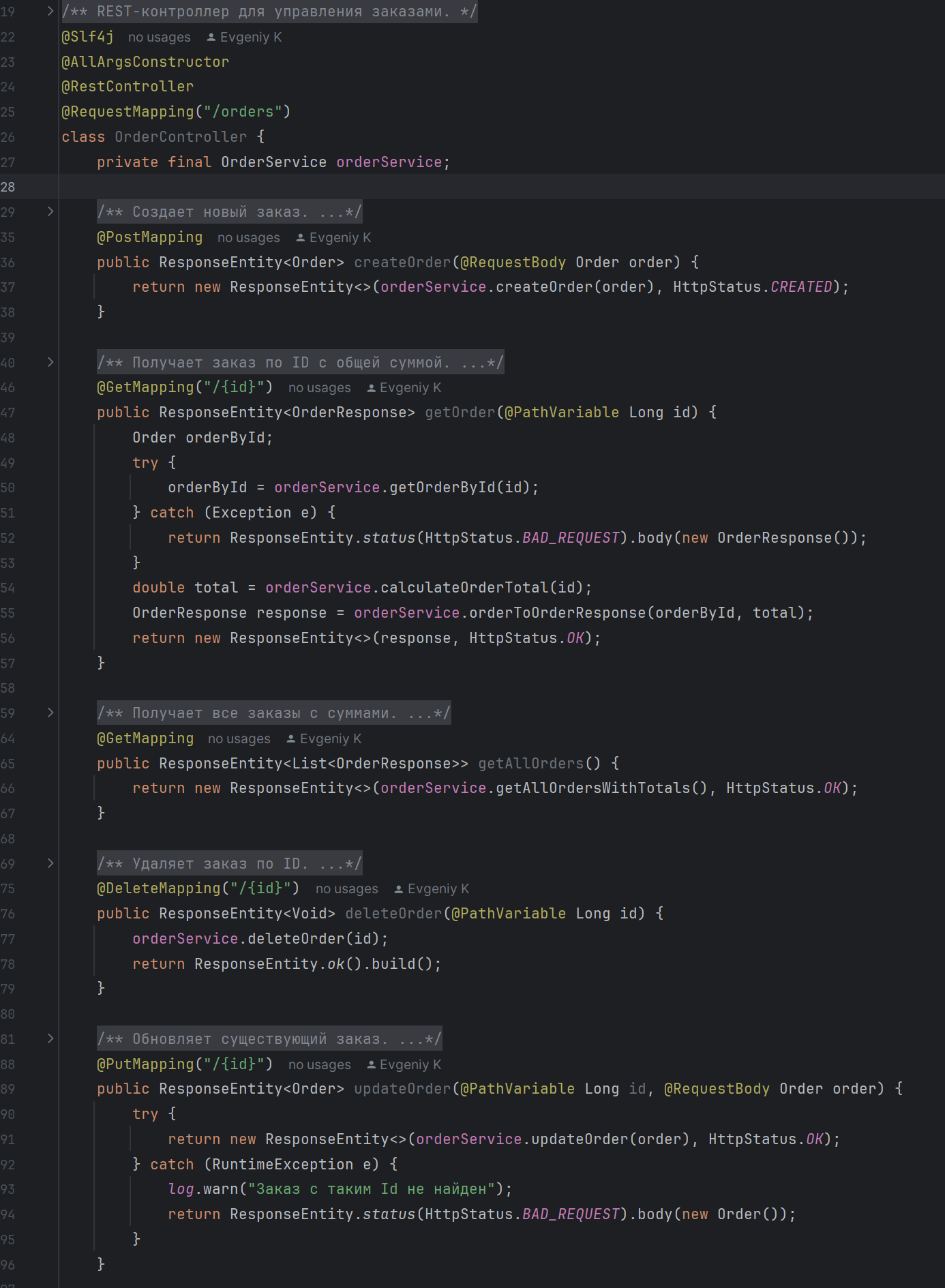
1. **Метод addProductFromMenu() — Добавление продукта из меню в заказ**

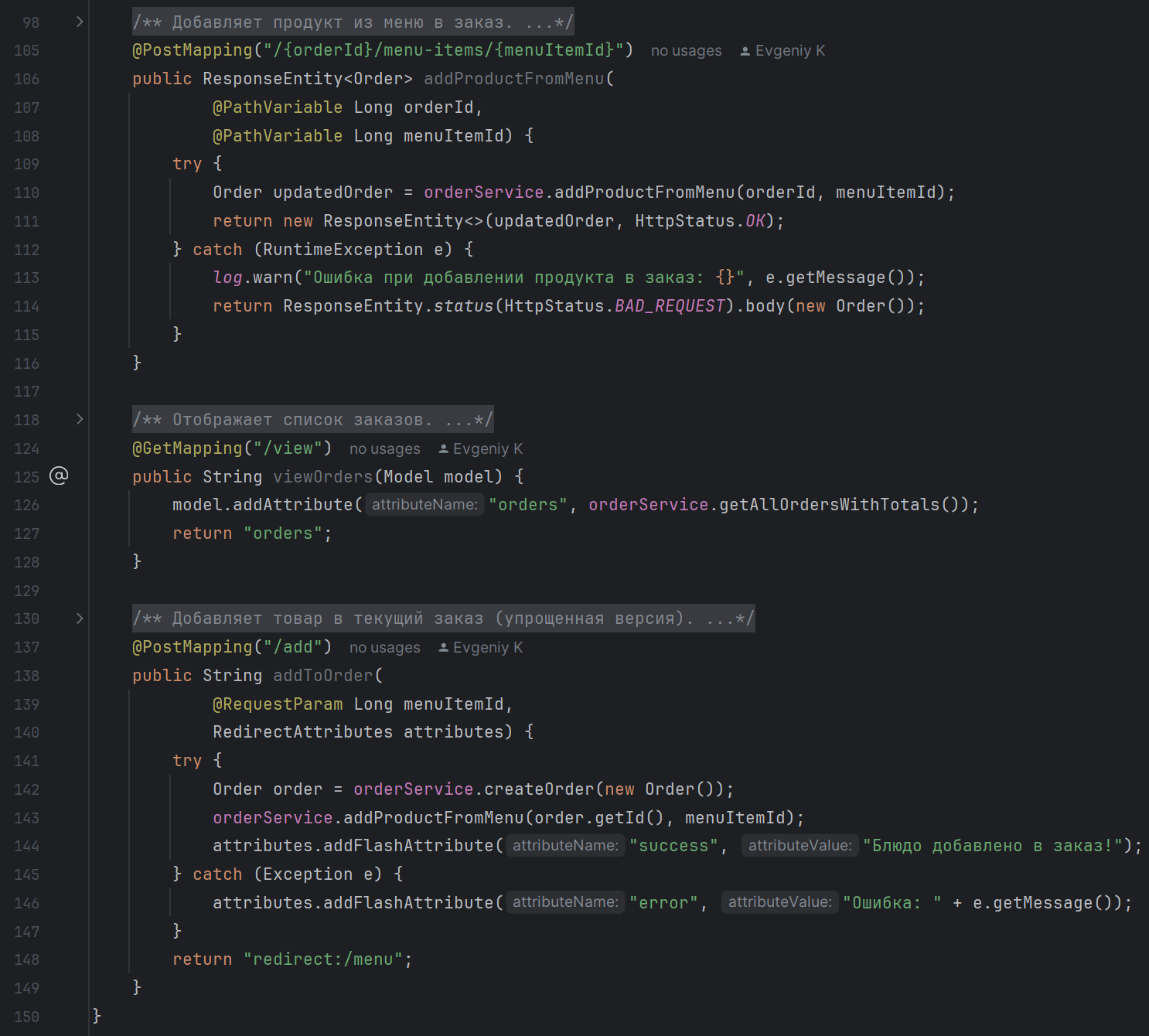
* **URL**: /orders/{orderId}/menu-items/{menuItemId}.
* **Функциональность**:

Добавляет продукт из меню в заказ.Возвращает обновленный заказ в формате JSON.

**Функциональные особенности:**

* Автоматически обновляет статус заказа при изменении статусов продуктов.
* Использует OrderService для логики заказов и ProductService для конвертации данных.
* Доступ к методам разрешен для ролей **ADMIN** и **STAFF**.





**Контроллер MenuController**

MenuController — это REST-контроллер, предназначенный для управления меню ресторана. Он обеспечивает создание, модификацию и получение информации о меню и его элементах. Этот контроллер служит основным интерфейсом для администраторов, позволяя им настраивать ассортимент блюд, доступных для заказа.

**Основные функции:**

* Создание меню.
* Добавление элементов в меню.
* Получение информации: возвращает список всех меню или конкретного меню по ID. Предоставляет элементы выбранного меню для отображения клиентам.

**Ключевые методы MenuController:**

**Метод createMenu() — Создание меню**

* **URL**: /menus.
* **Функциональность**:
  + Создает новое меню на основе переданных данных.
  + Возвращает созданное меню в формате JSON.

**Метод addPizzaToMenu() — Добавление пиццы в меню**

* **URL**: /menus/{menuId}/pizzas.
* **Функциональность**:
  + Добавляет пиццу в указанное меню.
  + Возвращает добавленную пиццу в формате JSON.

**Метод addRollsToMenu() — Добавление роллов в меню**

* **URL**: /menus/{menuId}/rolls.
* **Функциональность**:
  + Добавляет роллы в указанное меню.
  + Возвращает добавленные роллы в формате JSON.

**Метод getMenuItems() — Получение элементов меню**

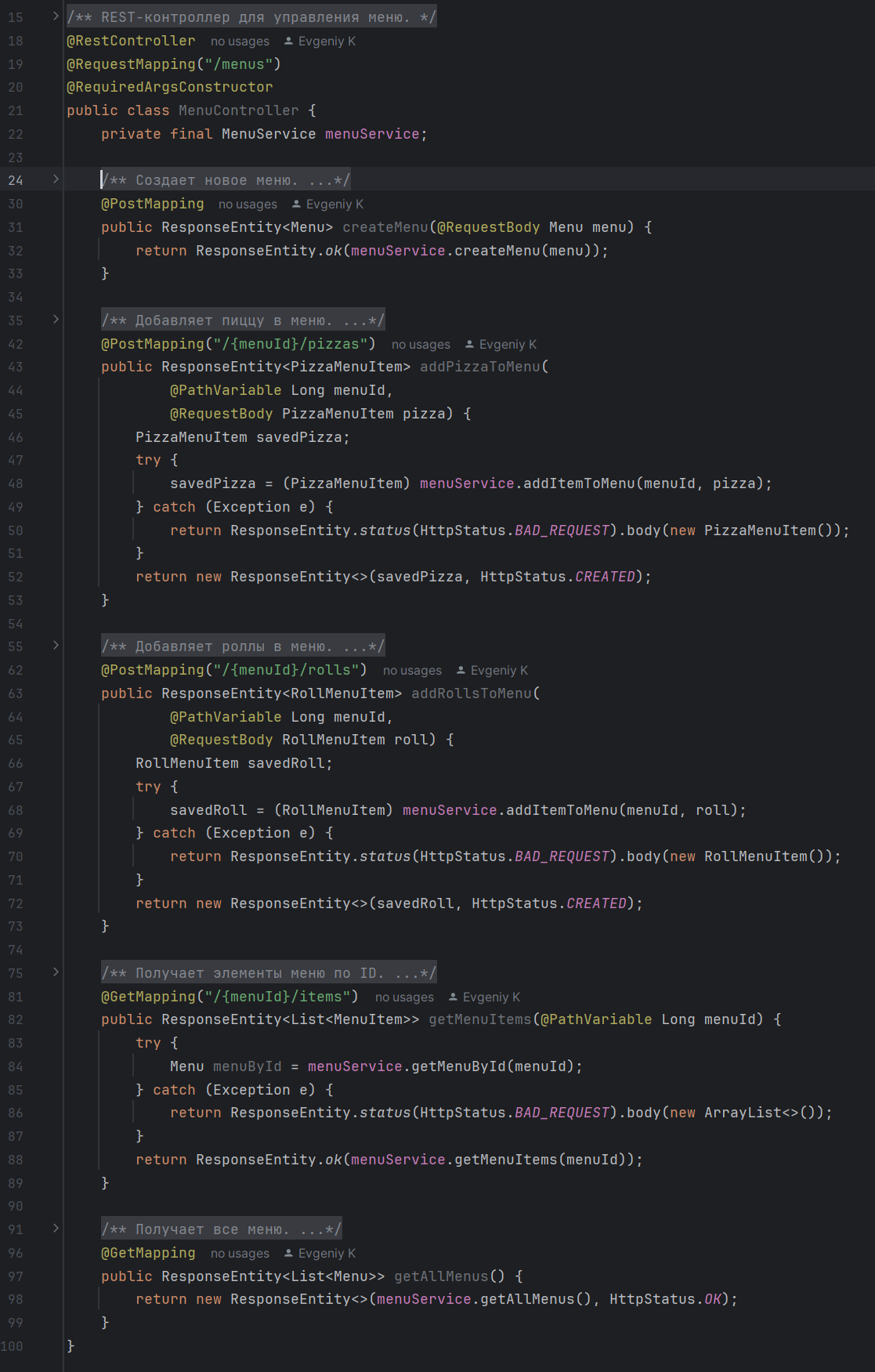
* **URL**: /menus/{menuId}/items.
* **Функциональность**:
  + Возвращает список всех элементов меню (пицц и роллов) для указанного меню.

**Метод getAllMenus() — Получение всех меню**

* **URL**: /menus.
* **Функциональность**:
  + Возвращает список всех меню, доступных в системе.

**Функциональные особенности:**

* Работает с MenuService для загрузки и сохранения данных.



**2.2. Проектирование WEB – интерфейса приложения**

Веб-интерфейс приложения решено было построить на основе **Spring Boot** с использованием **Thymeleaf** для рендеринга HTML-шаблонов. Основные компоненты:

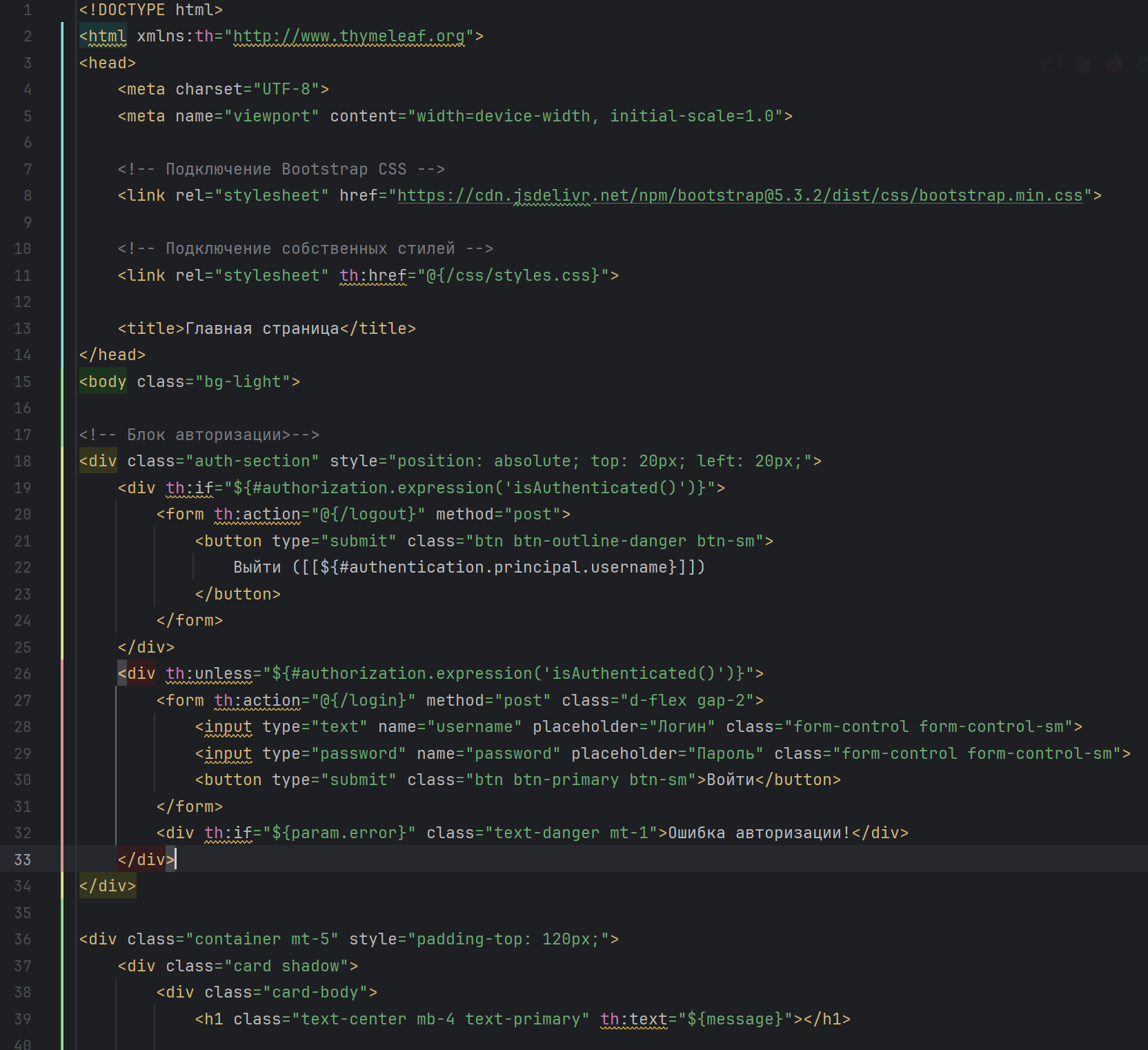
* **Контроллеры**: MainController, CartController, LoginController — обрабатывают запросы, взаимодействуют с сервисами и передают данные в представления.
* **Сервисы**: MenuService, OrderService, ProductService — реализуют бизнес-логику работы с меню, заказами и продуктами.
* **Сущности**: Menu, MenuItem, Order, Product и их наследники — отражают структуру данных в БД.
* **Шаблоны**: index.html, menu.html, cart-content.html, orders.html — обеспечивают визуализацию данных.

**Основные функции веб-интерфейса**

**1. Главная страница**

* **Маршрут**: / (обрабатывается MainController.home()).
* **Функционал**:
  + Отображает список доступных меню ресторанов.
  + Использует Bootstrap для адаптивного дизайна.
  + Интегрирует блок аутентификации: вход/выход пользователя через Spring Security.
* **Технологии**:
  + Thymeleaf для динамической подгрузки данных (например, списка меню через ${menus}).
  + Сессии для сохранения выбранного menuId.





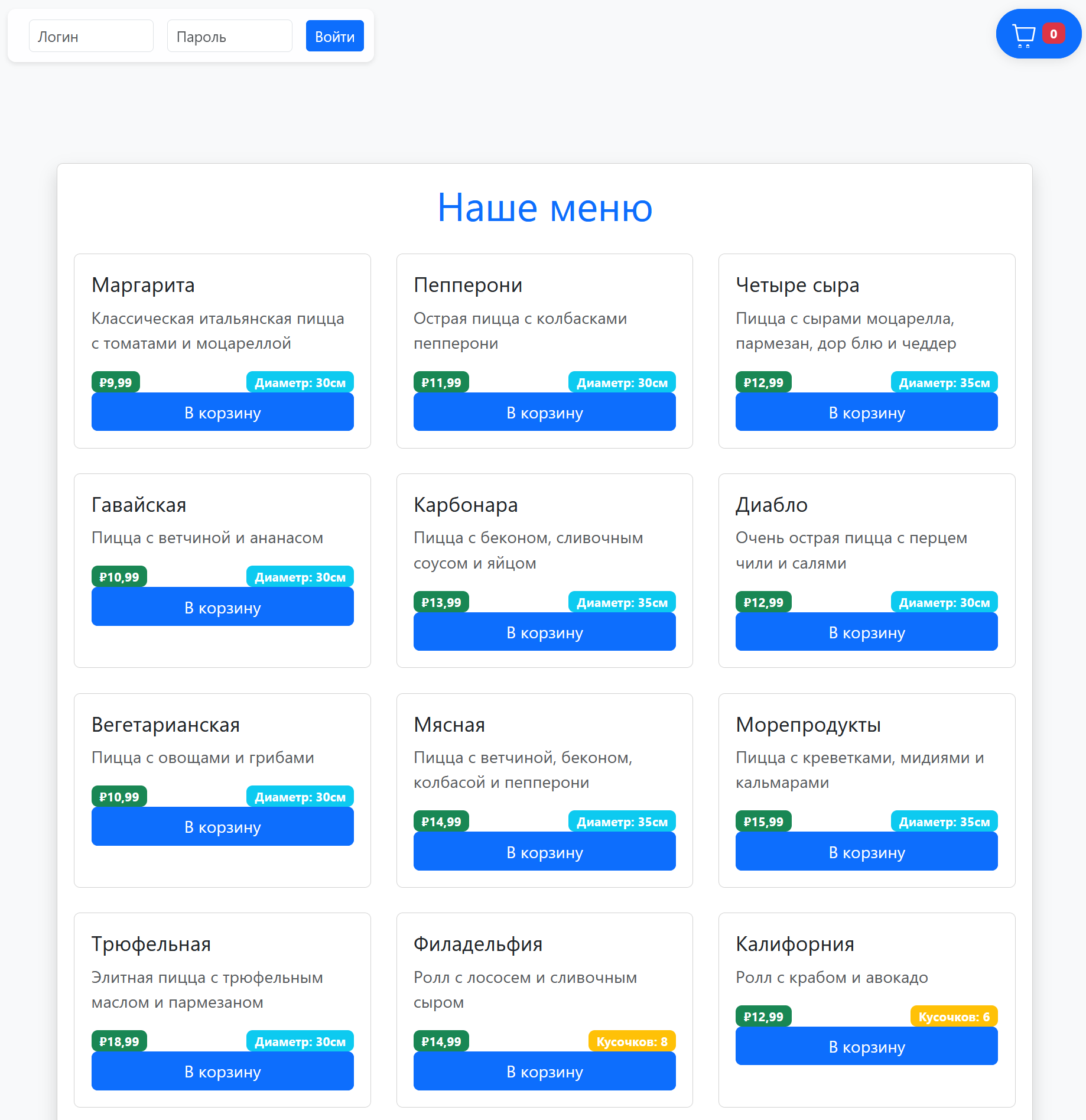


После выбора ресторана открывается страница с меню данного ресторана.

**2. Страница меню**

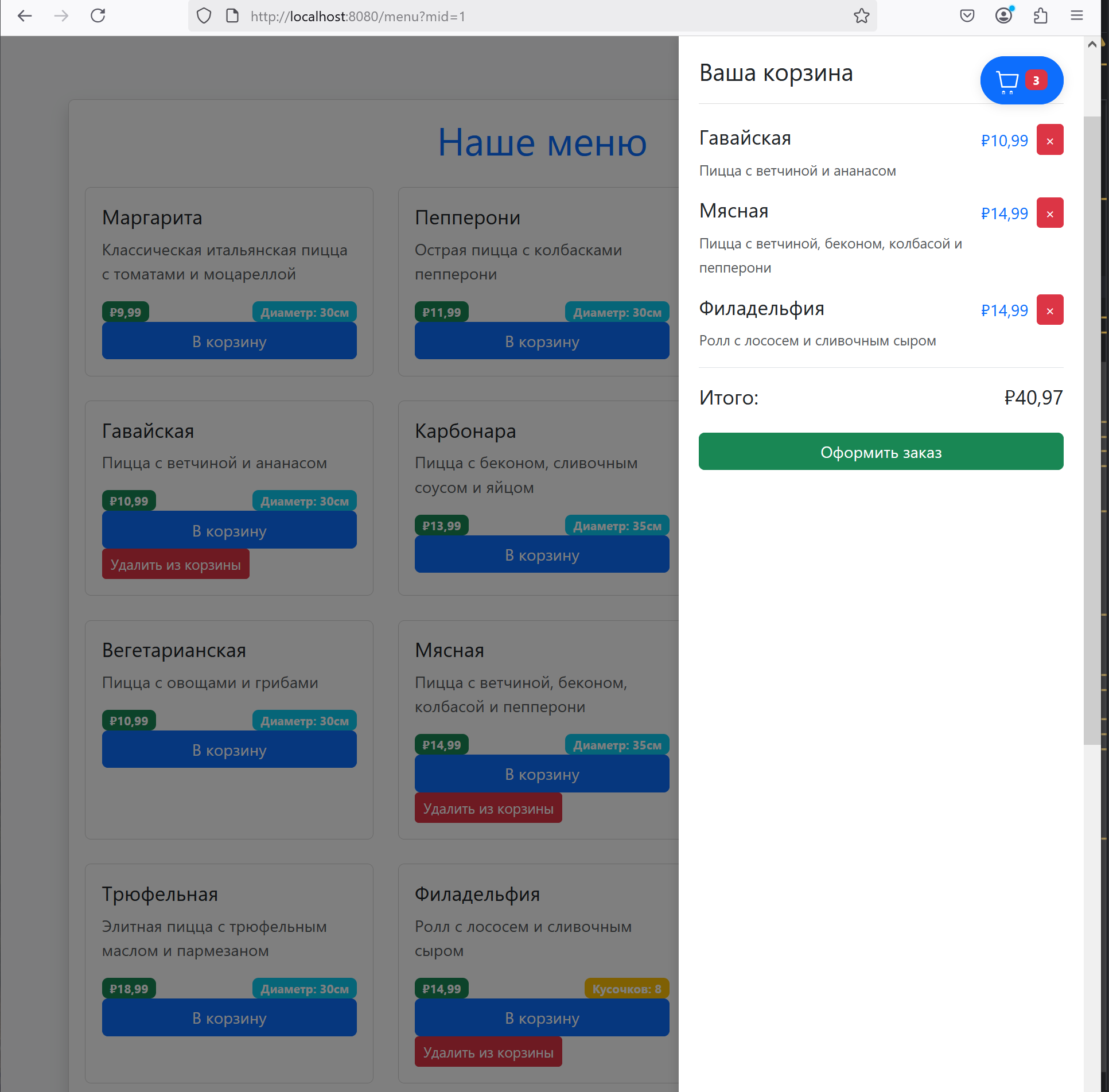
* **Маршрут**: /menu (обрабатывается MainController.menu()).
* **Функционал**:
  + Отображает пункты меню выбранного ресторана.
  + Позволяет добавлять/удалять товары в корзину.
  + Отображает иконку корзины с количеством выбранных товаров.
  + Динамически показывает характеристики товаров (диаметр пиццы, количество кусочков роллов).
* **Особенности**:
  + Использование наследования в сущностях (PizzaMenuItem, RollMenuItem) для поддержки разных типов блюд.
  + Проверка наличия товара в корзине через th:if="${#lists.contains(session.cart.![id], menuItem.id)}".

Выбранные пункты в меню сразу же попадают в корзину товаров реализованную при помощи скрытой боковой панели, которая открывается при нажатии на иконку корзины.



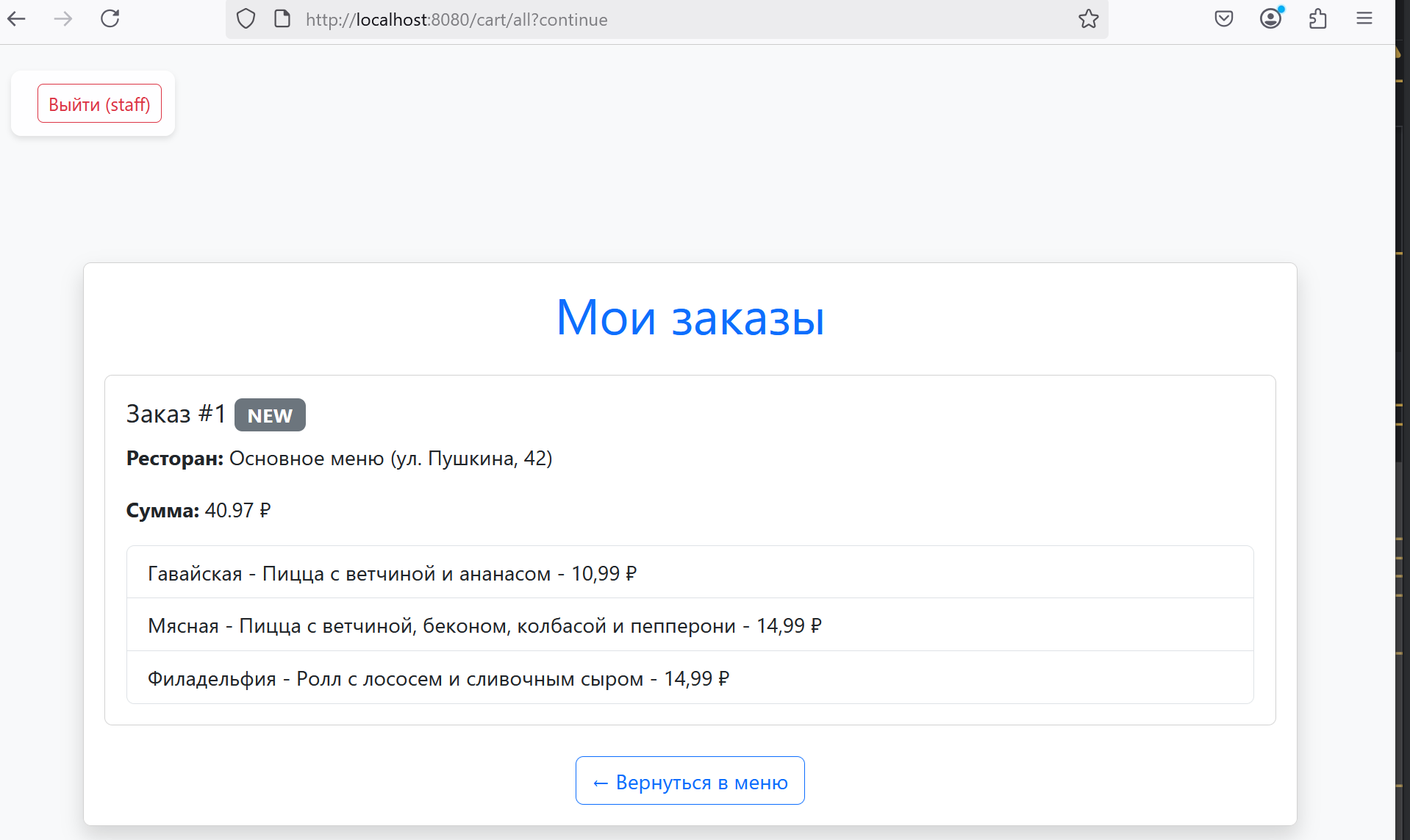
**3. Корзина**

* **Маршруты**: /cart, /cart/add, /cart/remove, /cart/checkout (обрабатываются CartController).
* **Функционал**:
  + Хранение корзины в сессии (List<MenuItem>).
  + Расчет общей суммы заказа (calculateTotal()).
  + Оформление заказа с конвертацией MenuItem в Product (например, PizzaMenuItem → Pizza).
* **Особенности**:
  + Боковая панель корзины с анимацией (реализована через JavaScript).
  + Обработка ошибок: пустая корзина, отсутствие меню у товара.
  + Использование @Transactional для сохранения заказа и связанных продуктов.



**4. Просмотр заказов**

* **Маршрут**: /cart/all (доступен для ролей STAFF и ADMIN).
* **Функционал**:
  + Отображает список всех заказов с деталями: статус, сумма, ресторан, состав.
  + Преобразует заказы в DTO (OrderResponse) для передачи в шаблон.
* **Особенности**:
  + Автоматическое определение ресторана на основе menuId первого товара.
  + Логирование ошибок при отсутствии меню.



В верхнем левом углу интерфейса расположена форма авторизации. После успешного входа в систему данный блок трансформируется: вместо формы отображается кнопка «Выйти» с указанием роли пользователя в скобках (например, «Выйти (staff)»).

**Заключение**

Веб-интерфейс приложения обеспечивает полный цикл работы с заказами: от выбора меню до оформления и просмотра истории заказов. Использование Spring Boot и Thymeleaf позволило создать структурированное и масштабируемое решение. Ключевые особенности — поддержка разных типов продуктов, интеграция с Spring Security и динамическое обновление корзины.

**2.3. Проектирование базы данных**

Проект реализует реляционную базу данных с использованием **JPA (Hibernate)** для объектно-реляционного маппинга (ORM). Для простоты разработки приложения было принято решение использовать базу данных H2. Основные сущности и их связи:

 **Menu (Меню)**

* Поля: id, name, address, description.
* Связь: OneToMany с MenuItem (пункты меню).

 **MenuItem (Пункт меню)**

* Абстрактный класс с наследованием через стратегию SINGLE\_TABLE (дискриминатор dtype).
* Наследники: PizzaMenuItem, RollMenuItem.
* Поля: id, name, description, price, menu\_id (связь с Menu).

 **Product (Продукт)**

* Абстрактный класс с наследованием через SINGLE\_TABLE.
* Наследники: Pizza, RollDish.
* Поля: id, name, description, price, status, menu\_id.
* Связь: ManyToMany с Order через таблицу orders\_products.

 **Order (Заказ)**

* Поля: id, status (NEW, IN\_PROGRESS, COMPLETED).
* Связь: ManyToMany с Product.
* Содержит список продуктов и обновляет статус автоматически.

|  |
| --- |
| **2.4. Безопасность приложения** |

Система безопасности приложения построена на базе Spring Security, интегрированной в Spring Boot-приложение. Основные компоненты включают:

* Аутентификацию пользователей через форму входа.
* Авторизацию на основе ролей (CLIENT, STAFF, ADMIN).
* Шифрование паролей.
* Управление сессиями и доступом к ресурсам.

**Аутентификация**  
Аутентификация реализована через кастомный сервис CustomUserDetailsService, который загружает данные пользователя из базы данных. Ключевые особенности:

* Роли пользователей: Пользователи (сущность User) связаны с ролями (сущность Role) через отношение ManyToMany. Роли определяют уровень доступа (например, ADMIN имеет полные права, STAFF — доступ к заказам, USER — базовый функционал).
* Шифрование паролей: Используется BCryptPasswordEncoder для хеширования паролей перед сохранением в БД. Это исключает хранение паролей в открытом виде.
* Форма входа: Настроена собственная страница входа (/login). После успешной аутентификации пользователь перенаправляется на запрашиваемую ранее страницу.

**Авторизация**  
Доступ к функционалу ограничен на уровне URL и методов:

* **Публичные ресурсы**: CSS, JS, страницы входа, меню и корзины доступны всем.
* **Ролевая модель**:
  + STAFF и ADMIN имеют доступ к просмотру всех заказов (/cart/all).
  + ADMIN обладает исключительными правами на управление продуктами, меню и системными настройками.
* **Аннотации безопасности**: В контроллерах (например, CartController) используется @PreAuthorize, чтобы ограничить доступ к методам по ролям.

**Защита от угроз**

* **CSRF-защита**: Включена по умолчанию в Spring Security. Все формы используют токены для предотвращения подделки запросов.
* **Управление сессиями**: Корзина пользователя хранится в HttpSession, что обеспечивает изоляцию данных между пользователями. Сессии уничтожаются при выходе (/logout).
* **Обработка ошибок**: При отказе в доступе пользователь перенаправляется на страницу входа с сообщением об ошибке (/login?error=access\_denied).

**Безопасность данных**

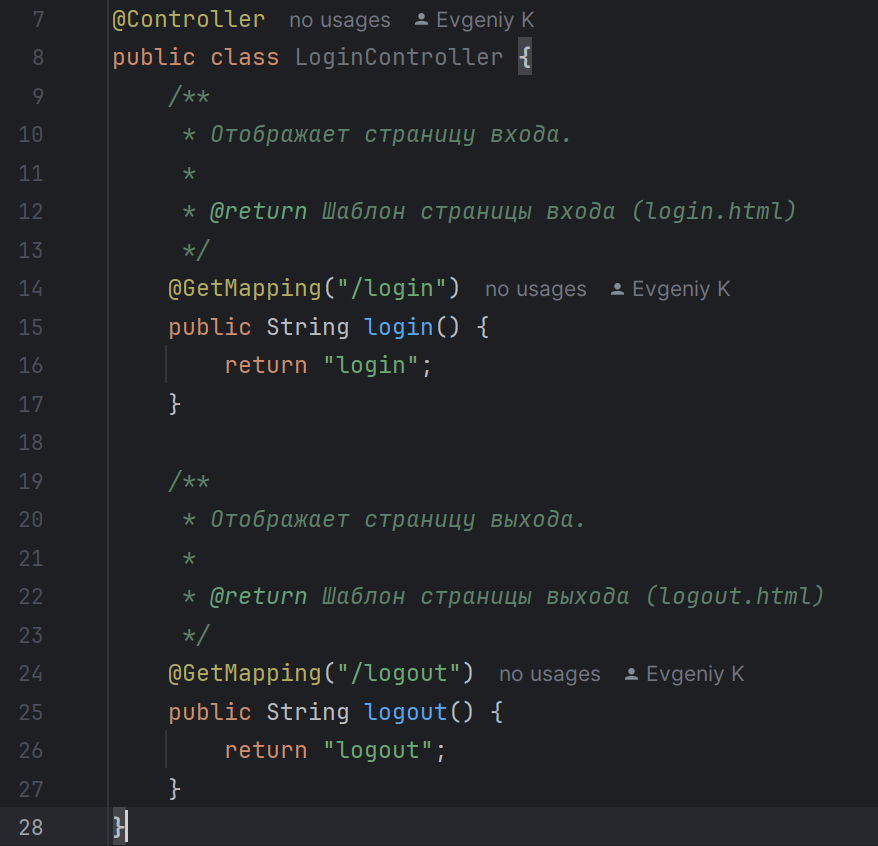
* **HTTPS**: Рекомендуется настроить приложение для работы по HTTPS в production-среде, чтобы шифровать трафик между клиентом и сервером.
* **Валидация входных данных**: В контроллерах (например, OrderService) проверяются корректность ID меню и заказов, что предотвращает SQL-инъекции и обработку несуществующих сущностей.

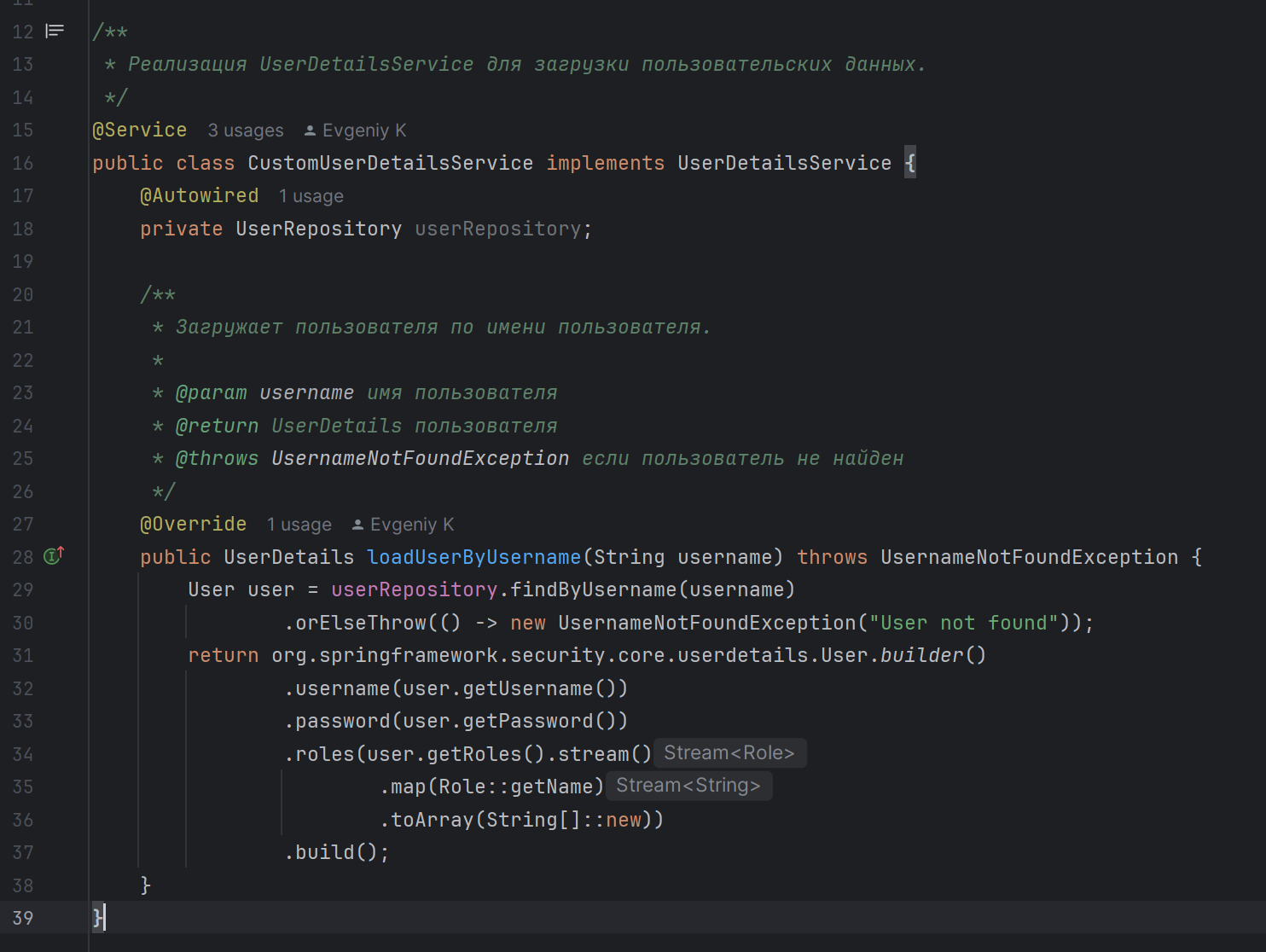
**Управление сессиями**

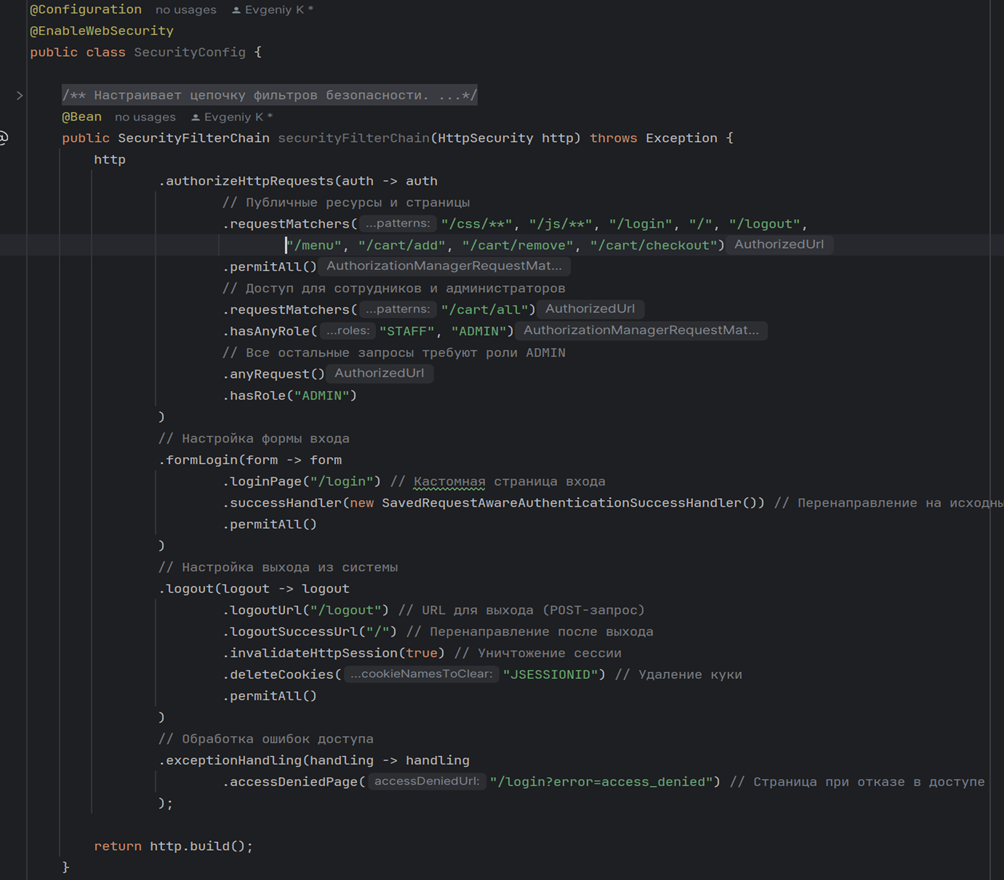
* **Таймаут сессии:** Не настроен явно, что может быть улучшено для автоматического завершения неактивных сессий.
* **Куки:** После выхода удаляются куки JSESSIONID, что снижает риск перехвата сессии.

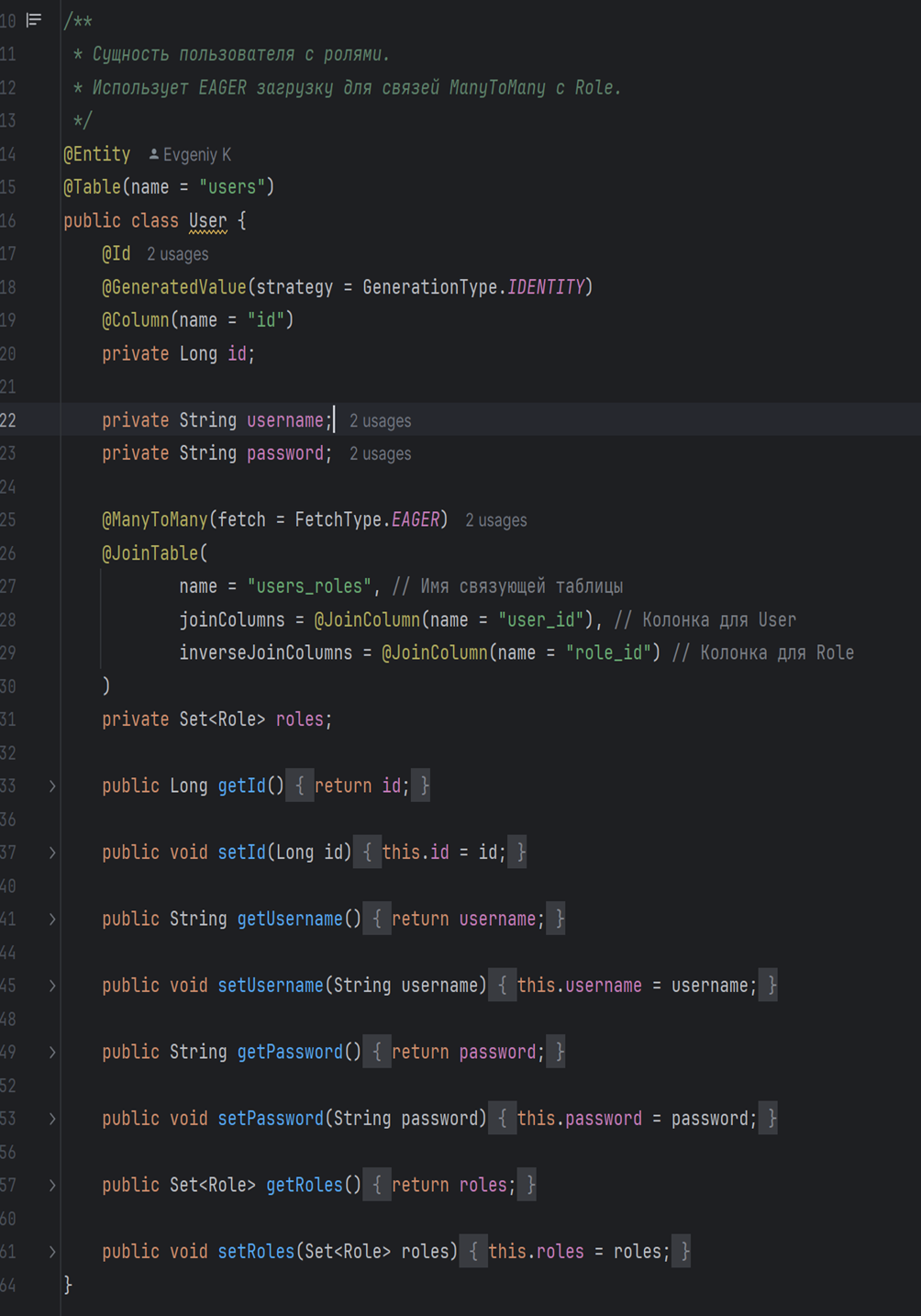
**Мониторинг и аудит**

* **Логирование**: В сервисах (например, OrderService) используются логи для отслеживания ошибок (например, попыток доступа к несуществующим заказам).











**Заключение**

В рамках данного дипломного проекта было разработано веб-приложение под названием «Система управления заказами еды», основная цель которого заключается в создании комплексной производственной системы, охватывающей широкий спектр функций. Это включает в себя предоставление пользователям возможности выбора товаров, оформления заказов, а также хранение и мониторинг статусов заказов. Проект был реализован с использованием современных технологий веб-разработки, таких как Java Spring Framework для бэкенда, Thymeleaf для генерации HTML-шаблонов, Bootstrap для создания адаптивного и эстетичного интерфейса, а также H2 и Hibernate в качестве инструментов для работы с базами данных.

В процессе разработки были успешно выполнены следующие ключевые задачи:

* Создание удобного и интуитивно понятного пользовательского интерфейса, который обеспечивает простоту взаимодействия с приложением для всех категорий пользователей.
* Реализация базового функционала, включающего добавление, редактирование и удаление заказов блюд, а также управление меню ресторанов.
* Обеспечение высокого уровня безопасности данных пользователей благодаря внедрению механизмов аутентификации и авторизации с использованием Spring Security.
* Применение принципов SOLID и популярных паттернов проектирования, что способствовало повышению гибкости, поддерживаемости и расширяемости кодовой базы.

В перспективе приложение может быть усовершенствовано за счет добавления новых функций, таких как:

* Анализ и визуализация статистики заказов, что позволит ресторанам лучше понимать предпочтения клиентов и оптимизировать свою работу.
* Внедрение бонусной системы, которая будет начислять баллы клиентам за совершение заказов, что повысит их лояльность.
* Разработка мобильной версии приложения, которая предоставит пользователям возможность оформлять заказы из любого места, что значительно повысит удобство использования.

Таким образом, дипломный проект «Система управления заказами еды» не только продемонстрировал навыки разработки веб-приложений с использованием современных технологий, но и подчеркнул важность создания безопасных, удобных и функциональных решений, ориентированных на потребности конечных пользователей.

**Список литературы**

1. **Spring 5 для проффесионалов,** Кларенс Хо, Крис Шефер, Юлиана Козмина, Роб Харроп (5-е издание, 2019 год)
2. Учебный материал Гикбрейнс.
3. Статьи и видео-материалы из сети интернет.