Geekbrains

**Разработка Интернет-магазина**

**на базе микросервисной архитектуры**

Разработчик Java

Гулькина Е.М.

г. Санкт-Петербург

2024

**Содержание**

1. Введение

1.1. Описание предметной области проекта (интернет-магазин).

1.2. Цели и задачи проекта.

1.3. Обзор используемых технологий и инструментов.

2. Анализ требований

2.1. Функциональные требования. Описание требований к административной

панели, личному кабинету пользователя, корзине, странице товара и оплате.

2.2. Нефункциональные требования к интернет-магазину.

2.3. Архитектурные документы

3. Архитектура проекта

3.1. Обзор архитектуры приложения.

3.2. Описание слоев приложения (представление, бизнес-логика, доступ к данным).

3.3. Использование Spring Boot для управления микросервисами и конфигурацией.

3.4. Use-Case диаграмма

4. Проектирование базы данных

4.1. Модель данных интернет-магазина.

4.2. ER-диаграмма базы данных.

4.3. Использование JPA/Hibernate для доступа к данным.

5. Реализация

5.1. Дизайн сайта

5.1.1. Общий дизайн

5.1.2. Навигационная панель

5.1.3. Каталог

5.1.4. Страница товара

6.1.6. Корзина

6.1.7. Страница авторизации

6.1.8. Личный кабинет пользователя

6.1.9. Личный кабинет администратора: товары

6.2. Реализация нефункционального блока

6.2.1. Конфигурационный класс Config:

6.2.2. Конфигурация безопасности Spring Security

6.2.3. Настройки приложений.

6.3. Разработка каталога и страницы товара.

6.4. Реализация функционала корзины

6.5. Авторизация пользователя

6.6. Разработка личного кабинета администратора.

7. Мониторинг приложения с помощью Actuator, Micrometer, Prometheus, Grafana

8. Заключение

8.1. Общий обзор выполненной работы.

8.2. Полученные результаты и достижения.

8.3. Выводы и дальнейшие перспективы развития проекта.

9. Приложения

9.1. Код приложения «Интернет-магазин».

9.2. Дополнительные материалы.

**1. Введение**

**1.1. Описание предметной области проекта (интернет-магазин с каталогом)**

На данный момент, интернет-магазины играю ключевую роль в развитии коммерции. С растущей популярностью интернета и технологического прогресса, купить и продать онлайн гораздо проще, чем сходить пешком в обычный магазин. В современном мире онлайн магазин перестал быть чем-то необычным, наоборот, он стал неотъемлемой частью любого бизнеса. Раскрутить новый магазин на много легче в виртуальном мире, который дает понятные и рабочие маркетинговые рычаги, а также более полную и точную статистику. Интернет-магазин — самый быстрый способ осуществления покупки, а соответственно решения проблем.

**1.2. Цели и задачи проекта**

Целью настоящего проекта является создание полнофункционального интернет-

магазина, способного удовлетворить потребности как покупателей, так и продавцов. Для

достижения этой цели поставлены следующие задачи:

1. Разработка стартовой страницы с каталогом и функцией фильтрации
2. Создание микросервиса управления учетными записями пользователей
3. Реализация личного кабинета пользователя с возможностью просмотра и
4. управления заказами, а также настройками аккаунта.
5. Разработка микросервиса административной панели для управления товарами и заказами
6. Обеспечение безопасности данных пользователей и администраторов приложения.
7. Интеграция микросервиса учета оплаты
8. Разработка механизмов мониторинга и администрирования для обеспечения
   1. стабильной работы приложения.

**1.3. Обзор используемых технологий и инструментов**

Для реализации поставленных целей и задач будут использоваться современные

технологии и инструменты разработки программного обеспечения.

**Spring Framework**: платформа Spring Framework предоставляет комплексную модель программирования и настройки для современных корпоративных приложений на базе Java на любой платформе развертывания. Ключевым элементом Spring является инфраструктурная поддержка на уровне приложений: Spring фокусируется на "настройке" корпоративных приложений, чтобы команды могли сосредоточиться на бизнес-логике на уровне приложений без ненужной привязки к конкретным средам развертывания.

**Spring Boot**: по причине громоздкой конфигурации зависимостей настройкаSpring Framework для корпоративных приложений превратилась в весьма утомительное и подверженное ошибкам занятие. Чтобы ускорить процесс управления зависимостями, Spring Boot неявно упаковывает необходимые сторонние зависимости для каждого типа приложения на основе Spring и предоставляет их разработчику посредством так называемых starter-пакетов (spring-boot-starter-web, spring-boot-starter-data-jpa и т.д.)

**Spring Data JPA и JpaRepository**: Spring Data JPA обеспечивает поддержку репозитория для Java Persistence API (JPA). Это упрощает разработку приложений, которым требуется доступ к источникам данных JPA. JpaRepository представляет собой интерфейс, который предоставляет методы CRUD (Create, Read, Update, Delete) для работы с сущностями базы данных. Querydsl основанная на идее сгенерированных классов метаданных, но реализованная с помощью гибкого и читаемого API.

**Spring Security**: это Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для промышленных приложений, созданных с помощью Spring Framework.

**Postgres Database**: представляет собой легковесную встроенную базу данных, которая

идеально подходит для разработки и тестирования приложений. Она обеспечивает

высокую производительность и поддерживает стандарт SQL, что делает ее удобной

для использования в различных проектах.

**Thymeleaf**: является шаблонным движком, который используется для создания

динамических веб-страниц с помощью HTML и CSS. Он обладает мощными

функциями, такими как вставка переменных, циклы и условные операторы, что

делает его идеальным выбором для разработки пользовательского интерфейса в

веб-приложениях.

**Lombok**: это библиотека для сокращения кода в классах и расширения функциональности языка Java. Она подключается к среде разработки (IDE) или инструменту сборки приложений Maven, Gradle в качестве плагина. Принцип работы библиотеки основан на использовании аннотаций, которые: многократно сокращают шаблонный код; экономят время разработчиков; улучшают читаемость кода.

Использование Lombok позволяет автоматически преобразовать объёмный Java-код в оптимизированную и лаконичную структуру. Также плагин избавляет разработчика от ручного набора однотипных фрагментов кода и предотвращает появление ошибок.

**Actuator:** очень важно следить за его работоспособностью. Особенно это актуально для критически важных приложений, таких как банковские системы, в которых отказ приложений напрямую влияет на бизнес. До Spring Actuator, нужно было писать код для проверки работоспособности приложения, но Spring Actuator предоставляет готовые точки (endpoint) для мониторинга приложения. Некоторые из них: /health для проверки состояния здоровья приложения, /metrics для получения метрик, /info для информации о

приложении, а также другие.

**Grafana:** универсальная обертка для работы с аналитическими данными, которые хранятся в разных источниках. Она сама ничего не хранит и не собирает, а является лишь универсальным клиентом для систем хранения метрик. Например, с помощью нее можно ходить за цифрами как в традиционную базу PostgreSQL, так и в специализированные аналитические системы типа Prometheus или Influx. Графану можно подключать к любому хранилищу статистических данных. Разные отделы компании могут использовать разные СУБД и системы сбора статистики. Так вот, Grafana умеет работать с любой популярной системой хранения данных. Конечно, делает она это не сама — первоначальную настройку и подключение к СУБД выполняют администраторы. Но на этом их работа заканчивается — дальше аналитики могут самостоятельно строить свои запросы.

**Micrometer и Prometheus**: Micrometer является библиотекой метрик для Java

приложений, которая предоставляет удобные средства для сбора и отображения

различных метрик приложения, таких как использование памяти, количество

запросов и т.д. Prometheus, в свою очередь, является системой мониторинга и

алертинга, которая используется для сбора метрик и анализа производительности

приложения.

**Spring Cloud Eureka**: Этот проект обеспечивает интеграцию Netflix OSS с приложениями Spring Boot посредством автоматической настройки и привязки к среде Spring и другим особенностям модели программирования Spring. С помощью нескольких простых пояснений вы сможете быстро включить и настроить общие шаблоны в своем приложении и создавать большие распределенные системы с использованием проверенных в боях компонентов Netflix. Предоставляемые шаблоны включают в себя обнаружение служб (Eureka).

**Spring Cloud Gateway**: представляет собой микросервисный шлюз API, который

используется для маршрутизации и фильтрации запросов в распределенных

системах. Он позволяет создавать гибкие и масштабируемые API, обеспечивая

централизованное управление маршрутами, обработкой запросов и защитой от

атак.

**2. Анализ требований**

**2.1. Функциональные требования**

**Каталог товаров:**

Требуется реализовать максимально оптимизированный с точки зрения запросов и времени отклика каталог. Пользователь должен иметь возможность отфильтровать товар, просмотреть его наличие на складе и возможность резервирования. Должна быть доступна галерея с изображениями.

**Корзина:**

Корзина представляет собой виртуальную покупательскую тележку, в которую

пользователи могут добавлять выбранные товары для последующего оформления заказа.

Пользователь может добавить товар в корзину, удалить товар из корзины, просмотреть его на индивидуальной странице продукта. В корзине должен быть учет всех ранее приобретённых товаров. Любые действия с корзиной возможны только после прохождения регистрации/авторизации.

**Личный кабинет пользователя:**

Личный кабинет пользователя предоставляет возможность пользователям

управлять своими персональными данными, просматривать историю заказов.

Проход в личный кабинет возможен только после регистрации/авторизации. На аккаунте пользователя хранится email, логин, имя, фамилия, дата рождения и аватар, роль. На данный момент, роли не используются, но они заложен под программу лояльности

**Административная панель:**

Административная панель предназначена для управления аспектами работы

интернет-магазина со стороны администраторов. Основные функциональные требования

включают в себя управление каталогом товаров. Администратор должен иметь возможность добавить товар, удалить, отредактировать.

**Платежи:**

Учет платежей должен быть доступен для пользователей, у которых есть привязанные карты. Клиент может посмотреть номер привязанной карты, сумму списания и баланс.

**2.2. Нефункциональные требования**

**Безопасность:**

Безопасность данных — это защитные меры, применяемые для защиты данных от несанкционированного доступа и сохранения конфиденциальности, целостности и доступности данных. Лучшие практики по обеспечению безопасности данных включают такие методы защиты данных, как шифрование данных, управление ключами, редактирование данных, разделение данных на подмножества и маскирование данных, а также контроль доступа привилегированных пользователей, аудит и мониторинг.

**Производительность:**

Производительность интернет-магазина имеет прямое влияние на

пользовательский опыт и удовлетворенность клиентов. Для обеспечения быстрой загрузки

страниц и обработки запросов необходимо оптимизировать работу приложения. Это

включает в себя оптимизацию кода, использование кэширования данных, а также

масштабируемость системы для обработки большого количества запросов при увеличении

нагрузки.

**Надежность:**

Это один из ключевых факторов, влияющих на лояльность покупателей. Магазин должен не только устойчиво работать даже в условия ограниченного интернета, но и хранить все учетные данные пользователей. На данный момент, законодательством РФ предусмотрен штраф в 400 000 за каждого человека, чьи личные данные были раскрыты.

**2.3. Архитектурные документы**

**Диаграмма компонентов:**

Это структурная диаграмма языка унифицированного моделирования, она описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами. Диаграмма компонентов предоставляет общую картину архитектуры системы, помогает разработчикам и архитекторам лучше понять ее структуру и взаимосвязи, а также является полезным инструментом для коммуникации и документирования архитектурных решений. Диаграмма компонентов разрабатывается для следующих целей:

1. визуализация общей структуры исходного кода программной системы;
2. спецификация исполнимого варианта программной системы;
3. обеспечение многократного использования отдельных фрагментов программного кода;
4. представление концептуальной и физической схем баз данных

**Диаграмма развертывания:**

Диаграмма развертывания (deployment diagram) предназначена для представления

общей конфигурации или топологии распределенной программной системы и содержит

изображение размещения различных артефактов по отдельным узлам системы.

Диаграмма развертывания предназначена для визуализации элементов и

компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом

представляются только компоненты-экземпляры программы, являющиеся исполнимыми

файлами или динамическими библиотеками. Те компоненты, которые не используются на

этапе исполнения, на диаграмме развертывания не показываются.

**Диаграмма классов:**

Диаграмма классов предназначена для представления внутренней структуры программы в виде классов и связей между ними. Все сущности реального мира, с которыми собирается работать программист, должны быть представлены объектами классов в программе. При этом у каждого класса должно быть только одно назначение и уникально осмысленное имя, которое будет связано с этой целью.

**Диаграмма последовательности:**

Диаграмма последовательности показывает последовательность взаимодействия

компонентов системы в рамках определенных сценариев использования. В контексте

интернет-магазина на диаграмме могут быть представлены процессы оформления заказа,

оплаты и обработки запросов. Эта диаграмма помогает понять взаимодействие между

компонентами системы и порядок выполнения операций.

**Диаграмма состояний:**

это тип диаграммы, используемый в UML для описания поведения систем, который основан на концепции диаграмм состояний Дэвида Харела. Диаграммы состояний отображают разрешенные состояния и переходы, а также события, которые влияют на эти переходы. Она помогает визуализировать весь жизненный цикл объектов и, таким образом, помогает лучше понять системы, основанные на состоянии.

**Диаграмма данных:**

Один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML. Итак DFD – это нотация, предназначенная для моделирования информационный систем с точки зрения хранения, обработки и передачи данных, то есть используется разработчиками ИС для разработчиков ИС.

**3. Архитектура проекта**

**3.1. Обзор архитектуры приложения**

Архитектура интернет-магазина представляет собой совокупность структурных

элементов и принципов организации, которые обеспечивают его функционирование и

развитие. Она определяет основные компоненты системы, их взаимосвязи, а также

принципы построения и взаимодействия между ними.

**Компоненты архитектуры:**

**Клиентская часть (Frontend):** это область разработки ПО, которая фокусируется на создании пользовательского интерфейса и взаимодействии с пользователем. В основе фронтенда лежат языки HTML, CSS и JavaScript, которые применяются для создания веб-страниц, их оформления и добавления интерактивности.

**Серверная часть (Backend):** эта область отвечает за создание логики приложения, хранение данных, безопасность, скорость отклика и общую надежность перед всеми типами атак.

**База данных (Database):** это хранилище всех данных, самая ценная и важная часть любой системы. Если база данных настроена неверно, особенно ее индексация, то приложение будет работать медленно и ненадежно. На данный момент, клиентская база данный считается самой важной и ценной частью любой системы.

**Административная панель:** Предоставляет администраторам возможность

управлять содержимым магазина, таким как добавление, редактирование и

удаление товаров, просмотр списка заказов.

**Принципы организации:**

**Модульность:**

Модульные монолиты состоят из более мелких независимых модулей, каждый из которых отвечает за определенную функциональность или бизнес-область. Эти модули организованы вокруг четких границ и обязанностей, способствуя разделению задач и ремонтопригодности.

**Тесная интеграция:**

Несмотря на модульную структуру, все модули тесно интегрированы в рамках единой кодовой базы и среды выполнения. Это означает, что нет необходимости в отдельных развертываниях или механизмах связи между модулями.

**Общая кодовая база и данные:**

Все модули используют одну и ту же кодовую базу, библиотеки и хранилище данных. Это упрощает процессы разработки и развертывания по сравнению с распределенными системами, сохраняя при этом некоторые преимущества модульности, такие как организация кода и разделение задач.

**Масштабируемость и ремонтопригодность:**

Модульные монолиты обеспечивают преимущества масштабируемости и ремонтопригодности по сравнению с традиционными монолитными архитектурами. Разбивая приложение на модули, разработчики могут более эффективно управлять сложностью и независимо масштабировать отдельные компоненты.

**Гибкость:**

Несмотря на единую кодовую базу, модульные монолиты обеспечивают гибкость с точки зрения разработки и развертывания. Разработчики могут легко добавлять, удалять или модифицировать модули для адаптации к меняющимся требованиям, не оказывая влияния на всю систему.

**Простота развертывания:**

Развертывание модульного монолита проще по сравнению с распределенными системами, поскольку существует только один артефакт развертывания. Это снижает сложность развертывания и операций, упрощая управление жизненным циклом приложения.

**Высокая когезия, низкое сцепление:**

Модули должны демонстрировать высокую когезию, что означает, что каждый модуль должен иметь четкое, целенаправленное назначение и отвечать за определенный аспект приложения. В то же время модули должны иметь низкое сцепление, что означает, что они должны быть слабо соединены и не должны сильно зависеть друг от друга.

**Четкие границы:**

Определите четкие границы между модулями, чтобы минимизировать зависимости и облегчить независимую разработку и тестирование. Избегайте смешивания модулей или введения жесткой связи между ними, поскольку это может привести к усложнению процесса внесения изменений.

**Принцип единой ответственности (SRP):**

Следуйте принципу единой ответственности, который гласит, что у каждого модуля должна быть только одна причина для изменения. Это помогает поддерживать ясность и простоту внутри модулей и гарантирует, что изменения в одном модуле не повлияют на несвязанные части системы.

**3.2. Описание слоев приложения (представление, бизнес-логика, доступ к данным)**

Архитектура приложения обычно разделяется на три основных слоя: представления,

бизнес-логику и доступ к данным.

**Слой представления (View Layer):**

Thymeleaf — это механизм шаблонов Java для обработки и создания HTML, XML, JavaScript, CSS и текста. В этом руководстве мы обсудим, как использовать Thymeleaf с Spring , а также некоторые основные варианты использования на уровне представления приложения Spring MVC. Библиотека чрезвычайно расширяема, а ее естественная возможность создания шаблонов гарантирует, что мы можем создавать прототипы шаблонов без серверной части. Это делает разработку очень быстрой по сравнению с другими популярными механизмами шаблонов, такими как JSP.

**Естественный синтаксис:** Thymeleaf использует синтаксис, который очень похож

на обычный HTML, что делает его легким для изучения и использования даже для

новичков.

**Полная поддержка стандартов HTML и CSS:** Thymeleaf поддерживает все

основные элементы HTML и CSS, что обеспечивает возможность создания

красивых и функциональных пользовательских интерфейсов.

**Интеграция с Spring Framework:** Thymeleaf интегрируется непосредственно с

Spring Framework, что упрощает передачу данных между контроллерами и

представлениями, а также обеспечивает удобное управление данными.

**Мощные функции и возможности:** Thymeleaf предоставляет множество функций

и возможностей для работы с данными, таких как циклы, условные операторы,

форматирование и международизация, что делает его идеальным инструментом для

создания динамических и интерактивных пользовательских интерфейсов.

**Поддержка множества форматов:** Thymeleaf поддерживает различные форматы

файлов, такие как HTML, XML, XHTML и другие, что делает его универсальным

инструментом для разработки веб-приложений.

**Слой бизнес-логики (Business Logic Layer):**

Этот слой играет ключевую роль в

архитектуре приложения, так как содержит основную логику, определяющую его

поведение и функциональность. В нем выполняются все операции, связанные с

обработкой запросов пользователя, расчетами цен, проверкой прав доступа и

другими задачами, необходимыми для корректной работы приложения.

В интернет-магазине используются различные модули и библиотеки для реализации

слоя бизнес-логики:

**Spring Security:** Этот модуль обеспечивает механизм аутентификации и

авторизации пользователей, а также защиту приложения от различных видов атак.

Он интегрируется с другими компонентами Spring, что делает его удобным и

мощным инструментом для обеспечения безопасности приложения.

**Lombok:**

Это библиотека, которая автоматически генерирует код для

управления геттерами, сеттерами, конструкторами и другими методами в Java-

классах. Она делает многословный код java читаем. Без нее можно устать крутить колесико мышки, что бы дойти ну нужной логики. Кроме того, аннотации вверху сразу делают понятным происходящее в классе.

**Spring Boot Starter Validation:**

Это стартер, предоставляющий возможности валидации данных, которые приходят в сервис @Validated используются для запуска процесса валидации при обработке запроса в Spring. @Validated — это специфическая для Spring аннотация, предоставляемая самим Spring. @Validated используется для проверки параметров метода на Spring-бине. Зачастую она используется, когда метод компонента Spring имеет параметры, которые должны проходить валидацию. @Validated поддерживает группировку ограничений. Это полезно, когда для одного и того же объекта при разных обстоятельствах требуются разные группы проверок.

**3.2.3. Слой доступа к данным (Data Access Layer):** в программном обеспечении — это слой компьютерной программы, который предоставляет упрощённый доступ к данным, хранимым в постоянном хранилище какого-либо типа, например, в реляционной базе данных. DAL может возвращать ссылку на объект с его атрибутами вместо строк полей из таблицы базы данных. Это позволяет создавать клиентские (или пользовательские) модули с более высоким уровнем абстракции. Spring Data JPA и JpaRepository предоставляют удобный и мощный способ работы с

базой данных в приложении:

**Spring Data JPA:** Spring Data JPA, входящий в более крупное семейство Spring Data, упрощает внедрение репозиториев на основе JPA (Java Persistence API). Это упрощает создание приложений на базе Spring, использующих технологии доступа к данным.

Реализация уровня доступа к данным для приложения может быть довольно громоздкой. Для выполнения простейших запросов приходится писать слишком много шаблонного кода. Добавьте такие вещи, как разбивка на страницы, аудит и другие часто необходимые опции, и вы в конечном итоге потеряетесь.

Spring Data JPA стремится значительно улучшить реализацию уровней доступа к данным, сократив трудозатраты до уровня, который действительно необходим.

**JpaRepository:** JpaRepository - это интерфейс, предоставляемый Spring Data JPA,

который содержит множество методов для работы с базой данных. Он

предоставляет готовую реализацию основных операций CRUD (Create, Read,

Update, Delete) и позволяет разработчикам быстро и удобно выполнять запросы к

базе данных без необходимости писать SQL-запросы вручную.

Использование Spring Data JPA и JpaRepository значительно упрощает доступ к

данным в приложении, делая процесс работы с базой данных более эффективным и

удобным. Разработчики могут быстро создавать репозитории для различных сущностей и

выполнять с ними операции CRUD, не тратя время на написание и отладку сложных SQL-

запросов. Благодаря этому приложение становится более гибким и легко

поддерживаемым, что существенно улучшает его качество и производительность.

**3.3. Использование Spring Boot для управления микросервисами и конфигурацией.**

В проекте используется комбинация монолитного подхода для основного

функционала и микросервисной архитектуры для оплаты с помощью Bank-сервиса.

Разделение на монолит и микросервисы позволяет достичь оптимального сочетания

простоты разработки и гибкости в обновлении и масштабировании.

**Микросервисы и их связь:** Для обеспечения оплаты заказов мы вынесли

функциональность в отдельный микросервис Bank. Взаимодействие между монолитным

приложением и микросервисом осуществляется с помощью следующих компонентов:

**Spring Cloud Gateway:** Для маршрутизации запросов между микросервисами и

основным приложением используется Spring Cloud Gateway. Это обеспечивает

единый точку входа в систему и позволяет легко добавлять, удалять или изменять

маршруты без изменения самого приложения.

**Spring Cloud Eureka:** Для обнаружения и регистрации микросервисов в сети

используется Spring Cloud Eureka. Это облегчает поиск и взаимодействие между

различными компонентами системы и обеспечивает их высокую доступность и

отказоустойчивость.

**Взаимодействие и мониторинг:** Для взаимодействия между микросервисами и

мониторинга их работы используются следующие инструменты:

**Micrometer и Prometheus:** Для сбора и анализа метрик производительности и

состояния микросервисов используются Micrometer и Prometheus. Они позволяют

мониторить работу приложения и выявлять проблемы производительности для их

дальнейшего устранения.

**3.4. Использование Spring Boot для управления микросервисами и**

**конфигурацией**

Spring Boot представляет собой набор инструментов и фреймворк для быстрой

разработки и развертывания микросервисных приложений. Он предоставляет удобные

средства для создания, настройки и управления микросервисами, а также автоматизирует

многие рутинные задачи, связанные с конфигурацией и развертыванием приложения.

**Основные возможности:**

 **Автоматическая конфигурация:** позволяет автоматически настраивать многие

компоненты приложения на основе классов и аннотаций, что значительно

упрощает процесс разработки и уменьшает объем написанного кода.

 **Встроенные серверы приложений:** поставляется с встроенными серверами

приложений, такими как Tomcat или Jetty, что позволяет запускать приложение без

необходимости настройки и установки дополнительных серверов.

 **Управление зависимостями:** автоматически управляет зависимостями и

библиотеками, необходимыми для работы приложения, что упрощает управление

зависимостями и обеспечивает их совместимость.

 **Мониторинг и управление приложением:** предоставляет инструменты для

мониторинга и управления работой приложения, такие как Actuator, который

позволяет получать информацию о состоянии приложения, метрики

производительности и другую полезную информацию.

Использование Spring Boot позволяет значительно упростить процесс разработки,

развертывания и управления микросервисными приложениями, что делает его одним из

наиболее популярных инструментов для создания современных веб-приложений.

**3.5. Use-Case диаграмма**

Use-Case диаграмма представляет собой инструмент для описания

функциональных требований к системе на основе взаимодействия ее актеров с системой.

В контексте интернет-магазина она позволяет идентифицировать основные действия

пользователей и их роли в системе.

17

**Основные актеры:**

Покупатель - пользователь, который просматривает каталог товаров, добавляет их в

корзину и совершает покупки.

Администратор - пользователь, который управляет каталогом товаров, управляет

заказами и контролирует работу интернет-магазина.

**Сценарии использования:**

 Просмотр каталога товаров:

 Покупатель просматривает доступные товары в каталоге.

 Покупатель может использовать фильтры для уточнения результатов поиска.

 Добавление товара в корзину:

 Покупатель выбирает товар и добавляет его в корзину.

 Покупатель может указать количество товара для покупки.

 Оформление заказа:

 Покупатель переходит к оформлению заказа после добавления товаров в корзину.

 Покупатель указывает данные для доставки и выбирает метод оплаты.

 Управление каталогом (для администратора):

 Администратор добавляет новые товары в каталог.

 Администратор изменяет информацию о товарах (цена, описание, изображение).

Примеры сценариев использования:

 *Пользователь просматривает каталог товаров и добавляет в корзину несколько*

*товаров.*

 *Пользователь оформляет заказ с доставкой на дом и оплачивает его онлайн.*

 *Администратор добавляет новый товар в каталог и устанавливает его цену.*

***Диаграммы Use-Case:***

*1. Приложение № 1 Use-Case диаграмма пользователя*

*2. Приложение № 2 Use-Case диаграмма администратора*

*18*

***4. Проектирование базы данных***

***4.1. Модель данных интернет-магазина:***

*Модель данных интернет-магазина представляет собой структуру, описывающую*

*основные сущности и их взаимосвязи в системе. В данном случае используется набор*

*классов сущностей, определенных в пакете* ***domen****, которые отражают ключевые аспекты*

*работы интернет-магазина.*

***Car (Автомобиль):*** *Этот класс представляет сущность автомобиля, доступного для*

*продажи в магазине. Есть связь с заказами, в которых этот автомобиль участвует. Он*

*содержит следующие поля:*

* id: уникальный идентификатор автомобиля,*

* maker: марка автомобиля,*

* model: модель автомобиля,*

* year: год выпуска,*

* typeCar: тип кузова (COUPE, HATCHBACK, SEDAN и т.д.),*

* gearbox: тип коробки передач (MANUAL, AUTOMATIC, ROBOT и т.д.),*

* engine: объем двигателя,*

* fuel: тип топлива (GASOLINE, DIESEL, ELECTRIC и т.д.),*

* colour: цвет автомобиля (BLACK, WHITE, RED и т.д.),*

* img: ссылка на изображение автомобиля,*

* price: цена автомобиля,*

* volume: количество единиц товара в наличии,*

* purchased: флаг, указывающий на то, был ли автомобиль когда-либо приобретен,*

* orders: связь с заказами, в которых участвует данный автомобиль.*

***Cart (Корзина****): Этот класс представляет собой корзину покупок, в которой*

*хранятся выбранные пользователем автомобили. Он содержит следующие поля:*

* carList: список выбранных автомобилей,*

* coupon: примененный купон скидки.*

***Lot (Лот):*** *Класс, представляющий отдельный лот автомобилей в корзине. Он*

*содержит следующие поля:*

* id: уникальный идентификатор лота,*

* lot: количество автомобилей в лоте.*

*19*

***Coupon (Купон):*** *Этот класс представляет собой купон, который может быть*

*использован для получения скидки при оформлении заказа. Есть связь с заказами, в*

*которых этот купон был использован. Он содержит следующие поля:*

* id: уникальный идентификатор купона,*

* name: наименование купона,*

* discount: размер скидки,*

* valid: флаг, указывающий на действительность купона,*

* creatAt: дата создания купона,*

* orders: связь с заказами, в которых использован данный купон.*

***Orders (Заказы): Класс, представляющий заказы, сделанные пользователями в***

***магазине. Также есть связь с купонами и автомобилями через таблицу OrdersCar.***

***Он содержит следующие поля:***

*** id: уникальный идентификатор заказа,***

*** person: пользователь, сделавший заказ,***

*** coupon: примененный купон скидки,***

*** creatAt: дата создания заказа,***

*** status: статус заказа (CREATED, COMPLETED, CANCELLED, PAID),***

*** cars: список автомобилей в заказе.***

***OrdersCar (Связь Заказ-Автомобиль): Это связующий класс @ManyToMany***

***между заказами и автомобилями. Он содержит следующие поля:***

*** orders: заказ, к которому относится автомобиль,***

*** car: автомобиль, включенный в заказ,***

*** lot: количество автомобилей данной модели в заказе.***

***Person (Пользователь): Этот класс представляет сущность пользователя магазина.***

***Есть связь с заказами сделанными этим пользователем. Он содержит следующие поля:***

*** id: уникальный идентификатор пользователя,***

*** name: имя пользователя,***

*** age: возраст пользователя,***

*** card: номер карты пользователя,***

*** phone: номер телефона пользователя,***

*** email: электронная почта пользователя,***

*** role: роль пользователя,***

***20***

*** password: пароль пользователя,***

*** orders: список заказов, сделанных пользователем.***

***Seller (Продавец): Класс, представляющий сущность продавца магазина. Он***

***содержит следующие поля:***

*** id: уникальный идентификатор продавца,***

*** company: название компании продавца,***

*** card: номер карты продавца,***

*** phone: номер телефона продавца,***

*** email: электронная почта продавца,***

*** role: роль продавца.***

***ContactEmail (Контактная информация по электронной почте): Класс,***

***представляющий контактную информацию, отправленную пользователем магазина. Он***

***содержит следующие поля:***

*** fName: имя отправителя,***

*** lName: фамилия отправителя,***

*** emailGuest: электронная почта отправителя,***

*** message: сообщение отправителя.***

***Эти классы представляют основные сущности и связи между ними в базе данных***

***интернет-магазина. Каждый из них играет важную роль в функционировании системы и***

***обеспечивает необходимую функциональность для работы магазина.***

***4.2. ER-диаграмма базы данных.***

***Сущности и их связи на основе аннотаций JPA:***

***Сущность Car:***

***@Entity***

***@Table(name = "Car")***

***public class Car implements Cloneable {***

*** @Entity: Обозначает, что класс Car является сущностью базы данных.***

*** @Table(name = "Car"): Указывает на таблицу базы данных, в которой будут***

***храниться данные о машинах.***

*** @Id, @GeneratedValue: Определяют первичный ключ id и автоматическую***

***генерацию его значений.***

***21***

***@ManyToMany***

***@JoinTable(name = "orders\_car",***

***joinColumns = @JoinColumn(name = "car\_id"),***

***inverseJoinColumns = @JoinColumn(name = "orders\_id"))***

***public List<Orders> orders = new ArrayList<>();***

*** @ManyToMany: Определяет связь "многие ко многим" с сущностью Orders, что***

***означает, что одна машина может быть в нескольких заказах, а один заказ может***

***содержать несколько машин.***

*** @JoinTable: Используется для определения промежуточной таблицы для связи***

***многие ко многим между таблицами Car и Orders.***

***Сущность Coupon:***

***@Entity***

***@Table(name = "Coupon")***

***public class Coupon {***

*** @Entity: Обозначает, что класс Coupon является сущностью базы данных.***

*** @Table(name = "Coupon"): Указывает на таблицу базы данных, в которой будут***

***храниться данные о купонах.***

*** @Id, @GeneratedValue: Определяют первичный ключ id и автоматическую***

***генерацию его значений.***

***@OneToMany(mappedBy = "coupon")***

***private List<Orders> orders;***

*** @OneToMany: Указывает на связь "один ко многим" с сущностью Orders,***

***обозначая, что один купон может быть использован в нескольких заказах.***

***Сущность Orders:***

***@Entity***

***@Table(name = "Orders")***

***public class Orders {***

*** @Entity: Обозначает, что класс Orders является сущностью базы данных.***

*** @Table(name = "Orders"): Указывает на таблицу базы данных, в которой будут***

***храниться данные о заказах.***

*** @Id, @GeneratedValue: Определяют первичный ключ id и автоматическую***

***генерацию его значений.***

***@ManyToOne***

***@JoinColumn(name = "person\_id", referencedColumnName = "id")***

***private Person person;***

*** @ManyToOne: Указывает на связь "многие к одному" с сущностью Person,***

***обозначая, что множество заказов может принадлежать одному пользователю.***

***22***

***@ManyToMany***

***@JoinTable(name = "orders\_car",***

***joinColumns = @JoinColumn(name = "orders\_id"),***

***inverseJoinColumns = @JoinColumn(name = "car\_id"))***

***public List<Car> cars = new ArrayList<>();***

*** @ManyToMany: Определяет связь "многие ко многим" с сущностью Car, что***

***означает, что один заказ может содержать несколько машин, а одна машина может***

***быть в нескольких заказах.***

*** @JoinTable: Используется для определения промежуточной таблицы для связи***

***многие ко многим между таблицами Orders и Car.***

***Для дополнения ER-диаграммы базы данных мы должны учитывать все сущности и***

***связи между ними, включая OrdersCar и IdOrdersCar.***

***OrdersCar:***

***@Entity***

***@Table(name = "Orders\_Car")***

***@IdClass(IdOrdersCar.class)***

***public class OrdersCar {***

***@Id***

***@ManyToOne***

***private Orders orders;***

***@Id***

***@ManyToOne***

***private Car car;***

*** Сущность OrdersCar представляет собой таблицу, которая содержит информацию о***

***заказанных автомобилях и количестве единиц каждого автомобиля в заказе. Эта***

***сущность создается для учета связи между заказами и автомобилями.***

***IdOrdersCar:***

***public class IdOrdersCar implements Serializable {***

***private int orders;***

***private int car;***

***}***

*** Сущность IdOrdersCar представляет собой составной ключ, используемый для***

***идентификации записей в таблице OrdersCar. Она состоит из двух частей:***

***идентификатор заказа (orders) и идентификатор автомобиля (car).***

***23***

***Сущность Person:***

***@Table(name = "person")***

***public class Person {***

***@Id***

***@GeneratedValue(strategy = GenerationType.IDENTITY)***

***@Column(name = "id")***

***private int id;***

*** @Entity: Обозначает, что класс Person является сущностью базы данных.***

*** @Table(name = "person"): Указывает на таблицу базы данных, в которой будут***

***храниться данные о пользователях.***

*** @Id, @GeneratedValue: Определяют первичный ключ id и автоматическую***

***генерацию его значений.***

*** @OneToMany: Указывает на связь "один ко многим" с сущностью Orders,***

***обозначая, что один пользователь может иметь множество заказов.***

***@OneToMany(mappedBy = "person")***

***private List<Orders> orders;***

***Сущность Seller:***

***@Entity***

***@Table(name = "seller")***

***public class Seller {***

***@Id***

***@GeneratedValue(strategy = GenerationType.IDENTITY)***

***@Column(name = "id")***

***private int id;***

*** @Entity: Обозначает, что класс Seller является сущностью базы данных.***

*** @Table(name = "seller"): Указывает на таблицу базы данных, в которой будут***

***храниться данные о продавцах.***

*** @Id, @GeneratedValue: Определяют первичный ключ id и автоматическую***

***генерацию его значений.***

***Эти сущности и их связи формируют основу базы данных интернет-магазина,***

***позволяя хранить информацию о машинах, купонах, заказах, пользователях и продавцах, а***

***также устанавливать связи между ними для обеспечения работы приложения.***

***В Приложении № 3 Таблица ERD.***

***24***

***4.3. Использование JPA/Hibernate для доступа к данным.***

***JPA (Java Persistence API) и Hibernate предоставляют ряд встроенных методов для***

***работы с базой данных. Давайте рассмотрим основные из них:***

*** save(entity): сохраняет сущность в базе данных. Если сущность уже существует в***

***базе данных, то она обновляется, в противном случае создается новая запись.***

*** findById(id): ищет сущность в базе данных по ее идентификатору.***

*** findAll(): извлекает все сущности из базы данных.***

*** delete(entity): удаляет сущность из базы данных.***

*** deleteById(id): удаляет сущность из базы данных по ее идентификатору.***

*** count(): возвращает количество сущностей в базе данных.***

*** existsById(id): проверяет, существует ли сущность с указанным идентификатором в***

***базе данных.***

***Рассмотрим, какие дополнительные методы добавлены в интерфейсы репозиториев:***

***CarRepo:***

***@Repository***

***public interface CarRepo extends JpaRepository<Car, Integer> {***

***@Query("SELECT car FROM Car car WHERE car.volume > 0")***

***List<Car> findTop3(Pageable pageable);***

***List<Car> findByVolumeGreaterThan(int volume);***

***@Query("SELECT car FROM Car car WHERE car.volume > 0")***

***Page<Car> findVolumeNotNull(Pageable pageable);}***

*** findTop3(Pageable pageable): возвращает три первых автомобиля с наличием на***

***складе, используя параметры страницы.***

*** findByVolumeGreaterThan(int volume): находит автомобили с объемом больше***

***указанного.***

*** findVolumeNotNull(Pageable pageable): находит автомобили с ненулевым объемом,***

***используя параметры страницы.***

***CouponRepo:***

***@Repository***

***public interface CouponRepo extends JpaRepository<Coupon, Integer> {***

***Optional<Coupon> findFirstByNameAndValid(String name, boolean valid);}***

*** findFirstByNameAndValid(String name, boolean valid): находит первый купон с***

***указанным именем и статусом действительности.***

***25***

***OrdersCarRepo:***

***@Repository***

***public interface OrdersCarRepo extends JpaRepository<OrdersCar, IdOrdersCar>***

***{***

***List<OrdersCar> findOrdersCarByOrdersId(int id);***

***Optional<OrdersCar> findOrdersCarByOrdersAndCar(Orders orders, Car car);***

***Optional<OrdersCar> findOrdersCarByOrdersIdAndCarId(int idOrder, int***

***idCar);}***

*** findOrdersCarByOrdersId(int id): находит записи заказанных автомобилей по***

***идентификатору заказа.***

*** findOrdersCarByOrdersAndCar(Orders orders, Car car): находит запись заказанного***

***автомобиля по заказу и автомобилю.***

*** findOrdersCarByOrdersIdAndCarId(int idOrder, int idCar): находит запись***

***заказанного автомобиля по идентификаторам заказа и автомобиля.***

***OrdersRepo:***

***@Repository***

***public interface OrdersRepo extends JpaRepository<Orders, Integer> {***

***Optional<Orders> findById(int id);}***

*** findById(int id): находит заказ по идентификатору.***

***PersonRepo:***

***@Repository***

***public interface PersonRepo extends JpaRepository<Person, Integer> {***

***Optional<Person> findByName(String name);***

***Optional<Person> findById(int id);***

***Optional<Person> findByEmail(String email);***

***Optional<Person> findByCard(Long card);***

***Optional<Person> findByPhone(Long phone);}***

*** findByName(String name): находит пользователя по имени.***

*** findByEmail(String email): находит пользователя по адресу электронной почты.***

*** findByCard(Long card): находит пользователя по номеру карты.***

*** findByPhone(Long phone): находит пользователя по номеру телефона.***

***SellerRepo:***

***@Repository***

***public interface SellerRepo extends JpaRepository<Seller, Integer> {}***

*** в данном интерфейсе используются стандартные методы JpaRepository, такие как***

***save, findById, findAll и т. д.***

***26***

***5. IML диаграмма классов***

***IML (Class Diagram) диаграмма классов представляет собой визуальное отображение***

***структуры классов в проекте. Она включает в себя сущности, их атрибуты, методы и***

***взаимосвязи между ними. IML диаграмма классов используется для наглядного***

***представления объектной модели приложения и позволяет легче понять его архитектуру.***

***В данном контексте, IML диаграмма классов будет представлена в рамках проектирования***

***базы данных интернет-магазина и включит классы сущностей, их атрибуты и связи, а***

***также используемые аннотации JPA для доступа к данным.***

***5.1. Монолитного приложения Shop***

***Приложение № 4 IML диаграмма классов монолитного приложения Shop***

***5.2. Микросервиса Bank***

***Приложение № 5 IML диаграмма классов микросервиса Bank***

***27***

***6. Реализация***

***6.1. Дизайн сайта***

***6.1.1. Общий дизайн***

***При проектировании навигационной панели и дизайна сайта использовалась***

***технология Thymeleaf для создания динамических таблиц и страниц.***

***Функциональность:***

*** Пользователь может навигироваться по сайту с помощью навигационной панели,***

***переходя между различными разделами.***

*** Магазин предоставляет информацию о продукции с изображениями, названиями и***

***ценами, а также позволяет пользователям исследовать товары и переходить к их***

***страницам.***

*** Секция помощи содержит информацию о том, как сайт может помочь***

***пользователю, и рассказывает о предоставляемых услугах.***

***28***

***6.1.2. Навигационная панель***

***Каждый контроллер отвечает за свою навигационную кнопку в меню навигации.***

***AuthController: отвечает за страницы входа в личный кабинет и регистрации.***

***ShopController: управляет каталогом с товарами и позволяет перелистывать страницы***

***каталога.***

***CartController: отвечает за управление корзиной покупок.***

***StartController: обрабатывает другие запросы, не отнесенные к вышеуказанным***

***контроллерам.***

***Код панели навигации:***

***<!-- Start Header/Navigation -->***

***<nav class="custom-navbar navbar navbar navbar-expand-md navbar-dark bg-dark"***

***arial-label="Furni navigation bar">***

***<div class="container">***

***<a class="navbar-brand" th:href="@{/}">Fastest<span>.</span></a>***

***<button class="navbar-toggler" type="button" data-bstoggle="***

***collapse" data-bs-target="#navbarsFurni"***

***aria-controls="navbarsFurni" aria-expanded="false" arialabel="***

***Toggle navigation">***

***<span class="navbar-toggler-icon"></span>***

***</button>***

***<div class="collapse navbar-collapse" id="navbarsFurni">***

***<ul class="custom-navbar-nav navbar-nav ms-auto mb-2 mb-md-0">***

***<li class="nav-item active">***

***<a class="nav-link" th:href="@{/}">Home</a>***

***</li>***

***<li><a class="nav-link" th:href="@{/shop}">Shop</a></li>***

***<li><a class="nav-link"***

***th:href="@{/services}">Services</a></li>***

***<li><a class="nav-link" th:href="@{/about}">About us</a></li>***

***<li><a class="nav-link" th:href="@{/contact}">Contact***

***us</a></li>***

***</ul>***

***<ul class="custom-navbar-cta navbar-nav mb-2 mb-md-0 ms-5">***

***<li><a class="nav-link" th:href="@{/auth/login}"><img***

***th:src="@{/images/user.svg}"></a></li>***

***<li><a class="nav-link" th:href="@{/cart}"><img***

***th:src="@{/images/cart.svg}">***

***<span class="badge bg-dark text-white ms-1 rounded-pill"***

***th:text="${cart.itemsCart()}">0</span>***

***</a></li>***

***</ul>***

***</div>***

***</div>***

***</nav>***

***<!-- End Header/Navigation -->***

***29***

***6.1.3. Каталог***

***Секция товаров:***

*** Отображает список товаров, представленных в виде карточек.***

*** Для каждого товара выводится изображение, название и цена.***

*** Используется цикл th:each для перебора списка товаров ${cars} и создания***

***карточек для каждого товара.***

*** Каждая карточка товара является ссылкой, которая перенаправляет пользователя на***

***страницу с подробной информацией о товаре.***

***Функциональность:***

*** Пользователь может просматривать список товаров и переходить на страницы с***

***подробной информацией о товарах.***

*** Пагинация \_\_\_\_\_\_\_обеспечивает удобное перемещение по списку товаров, особенно когда***

***список товаров велик и требует разделения на страницы для улучшения***

***производительности и удобства использования.***

***30***

***6.1.4. Страница товара***

***Функциональность:***

***Отображение информации о товаре:***

*** В правой части секции выводится информация о товаре, такая как марка и модель,***

***цена, а также характеристики.***

*** Над названием товара выводится SKU в формате "SKU: ID-XXXX", где XXXX -***

***идентификатор товара.***

***Добавление товара в корзину:***

*** Ниже информации о товаре находится форма для добавления товара в корзину.***

*** При отправке формы с помощью кнопки "Add to cart", товар добавляется в корзину.***

*** Используется метод POST и указывается адрес, куда будет отправлена информация***

***о добавлении товара.***

*** В случае ошибки при добавлении товара, например, если его количество***

***недоступно, выводится сообщение об ошибке в красном цвете.***

***Сообщение о наличии товара:***

*** Под формой добавления в корзину выводится сообщение о наличии товара.***

***31***

*** Если товар есть в наличии, сообщение выводится в зеленом цвете и указывается***

***количество доступных единиц товара.***

*** Если товар под заказ, сообщение выводится в оранжевом цвете и указывается***

***информация о том, что товар доступен только под заказ и потребуется время на***

***доставку.***

***6.1.5. Страница «Контакт с нами»***

***Форма обратной связи:***

*** First name: Поле для ввода имени пользователя.***

*** Last name: Поле для ввода фамилии пользователя.***

*** Your email address: Поле для ввода электронной почты пользователя.***

*** Message: Поле для ввода текста сообщения.***

*** Send Message: Кнопка для отправки сообщения.***

***Функциональность:***

*** Пользователь может ввести свои данные и отправить сообщение через форму***

***обратной связи.***

*** При некорректном заполнении полей выводятся сообщения об ошибках.***

*** Все введенные данные обрабатываются и отправляются на сервер для***

***последующей обработки.***

***32***

***6.1.6. Корзина***

***Функциональность:***

*** Пользователь может просматривать товары, добавленные в корзину.***

*** Есть возможность изменять количество товаров в корзине.***

*** Пользователь может удалять товары из корзины.***

*** Есть возможность применения купона для получения скидки на заказ.***

*** Пользователь может просматривать итоговую сумму заказа.***

*** После подтверждения заказа пользователь перенаправляется на страницу***

***оформления заказа.***

***33***

***6.1.7. Оформление заказа***

***Функциональность:***

*** В левой части раздела выводится информация о заказе, такая как выбранные***

***товары и их стоимость, итоговая сумма заказа.***

*** Используется таблица для отображения товаров и их стоимости, а также различных***

***сумм (подытог, примененный купон, итоговая сумма).***

*** Для оплаты наличными курьеру предоставляется информация и инструкции в***

***раскрывающемся блоке.***

***Оплата онлайн через MirPay:***

*** Используется форма для ввода данных покупателя, а также скрытые поля для***

***передачи данных о продавце, сумме заказа и прочее.***

*** После заполнения формы и нажатия кнопки "Оформить и оплатить заказ" данные***

***отправляются на сервер для обработки.***

*** Для отображения ошибок и информации о заказе используются условные***

***выражения на Thymeleaf.***

***34***

***6.1.8. Заказ оформлен***

***Функциональность:***

***На странице отображается иконка корзины с галочкой и заголовок "Thank you!".***

***Отображение информации о заказе:***

*** Используются условные выражения Thymeleaf для вывода информации о статусе***

***заказа в зависимости от его состояния (создан, оплачен и т.д.).***

*** Под информацией о заказе размещена кнопка "Back to shop" для возврата на***

***страницу магазина.***

***35***

***6.1.9. Страница авторизации***

***Функциональность:***

*** Регистрация нового пользователя:***

***Форма входа:***

*** Ниже навигационной панели расположена форма входа в систему.***

*** Форма состоит из двух полей: "Введите имя пользователя" и "Введите пароль".***

*** В форме присутствует скрытое поле с CSRF-токеном, предназначенным для***

***защиты от атак типа Cross-Site Request Forgery (CSRF).***

***Обработка ошибок:***

*** Если при входе произошла ошибка (например, неправильное имя пользователя или***

***пароль), сообщение об ошибке выводится под формой входа.***

*** При успешной аутентификации пользователь будет перенаправлен на страницу,***

***указанную в конфигурации приложения.***

***36***

***6.1.10. Личный кабинет пользователя***

***Функциональность:***

***86 о "Alter": При нажатии на эту кнопку пользователь перенаправляется на страницу***

***изменения данных аккаунта***

*** "Logout": При нажатии на эту кнопку пользователь выходит из системы.***

*** В этой секции выводится информация о пользователе: имя, телефон, электронная***

***почта и пароль.***

***Заказы пользователя:***

*** Для каждого заказа выводится информация о товарах, их ценах и статусе заказа.***

*** Если у заказа есть применённый купон, выводится его название.***

*** Для каждого товара в заказе отображается его наименование, цена за единицу,***

***количество и общая сумма.***

*** Если заказ имеет статус "Отменён", выводится соответствующее сообщение о его***

***отмене и кнопка для возможности отмены заказа.***

***37***

***6.1.11. Личный кабинет администратора: товары***

***Функциональность:***

***Таблица с машинами:***

*** Если машины присутствуют, создается таблица, в которой каждая строка***

***представляет одну машину.***

*** Каждая строка таблицы содержит информацию о машине: идентификатор (Id),***

***название (Info car), а также опции для редактирования и удаления машины.***

*** Для каждой машины создается кнопка "Редактировать", которая позволяет***

***изменить информацию о машине.***

*** Если машина не была приобретена (purchased == false), создается кнопка***

***"Удалить", позволяющая удалить машину из списка.***

***Пагинация:***

*** Под таблицей находится блок с пагинацией, который позволяет пользователю***

***перемещаться между страницами машин.***

*** Пагинация включает ссылки на предыдущие и следующие страницы, а также на***

***ближайшие 10 страниц в обе стороны от текущей.***

***38***

***6.1.12. Личный кабинет администратора: пользователи***

***Функциональность:***

***Таблица с людьми:***

*** Если люди присутствуют, создается таблица, в которой каждая строка***

***представляет одного человека.***

*** Каждая строка таблицы содержит информацию о человеке: идентификатор (Id) и***

***его имя (Info person).***

*** Если роль человека не администратор, создается кнопка "Редактировать", которая***

***позволяет изменить информацию о человеке.***

*** Если у человека нет заказов и он не администратор, позволяющая удалить человека***

***из списка.***

***Пагинация:***

*** Под таблицей находится блок с пагинацией, который позволяет пользователю***

***перемещаться между страницами машин.***

*** Пагинация включает ссылки на предыдущие и следующие страницы, а также на***

***ближайшие 10 страниц в обе стороны от текущей.***

***39***

***6.2. Реализация нефункционального блока***

***Нефункциональный блок проекта включает в себя ряд компонентов и настроек,***

***которые обеспечивают общую работоспособность и надежность приложения, но не прямо***

***связаны с его функциональностью. В этом блоке определяются основные параметры, а***

***также настраивается инфраструктура для поддержки функциональных модулей***

***приложения.***

***6.2.1. Конфигурационный класс Config***

***RestTemplate: представляет собой инструмент для взаимодействия с внешними***

***HTTP-ресурсами. Этот бин используется для выполнения HTTP-запросов к удаленным***

***серверам в рамках приложения.***

***@Bean***

***public RestTemplate template(){ return new RestTemplate();};***

***HttpHeaders: представляет HTTP-заголовки, которые могут быть отправлены***

***вместе с HTTP-запросами. Этот бин используется для создания и настройки заголовков,***

***отправляемых с HTTP-запросами и ответами.***

***@Bean***

***public HttpHeaders headers(){ return new HttpHeaders();}***

***ModelMapper: предоставляет удобный способ выполнения преобразования***

***объектов между различными типами данных в Java. Этот бин используется для***

***преобразования объектов между слоями приложения или для преобразования DTO (Data***

***Transfer Object) в сущности и наоборот.***

***@Bean***

***public ModelMapper modelMapper() { return new ModelMapper();}***

***SessionEventService: представляет сервис, отвечающий за обработку событий***

***сеансов пользователей в приложении. Этот бин может быть использован для регистрации***

***событий входа в систему, выхода из системы и других событий, связанных с***

***пользовательскими сеансами.***

***@Bean***

***public SessionEventService sessionEventService() {***

***return new SessionEventService();}***

***40***

***6.2.2. Конфигурация безопасности Spring Security***

***SecurityConfig: отвечает за настройку безопасности веб-приложения с помощью***

***Spring Security. В нем определяются правила аутентификации и авторизации***

***пользователей, а также настраиваются фильтры безопасности.***

***Аннотации класса:***

***@Configuration***

***@EnableWebSecurity***

***@RequiredArgsConstructor***

***public class SecurityConfig {***

***@Configuration: Обозначает, что данный класс является конфигурационным и***

***содержит определения бинов.***

***@EnableWebSecurity: Включает поддержку безопасности веб-приложения.***

***@RequiredArgsConstructor: Аннотация из библиотеки Lombok, которая***

***автоматически генерирует конструктор с аргументами для всех final полей.***

***Методы класса:***

***@Bean***

***public AuthenticationManager authManager(HttpSecurity http) throws Exception***

***{ AuthenticationManagerBuilder authenticationManagerBuilder =***

***http.getSharedObject(AuthenticationManagerBuilder.class);***

***authenticationManagerBuilder.authenticationProvider***

***(daoAuthenticationProvider());***

***return authenticationManagerBuilder.build(); }***

***authManager(HttpSecurity http): Создает и настраивает AuthenticationManager,***

***который управляет аутентификацией пользователей. Этот метод использует***

***AuthenticationManagerBuilder для настройки AuthenticationProvider, который***

***предоставляет информацию о пользователях для аутентификации.***

***@Bean***

***public DaoAuthenticationProvider daoAuthenticationProvider() {***

***DaoAuthenticationProvider daoAuthenticationProvider = new***

***DaoAuthenticationProvider();***

***daoAuthenticationProvider.setUserDetailsService(detailsService);***

***daoAuthenticationProvider.setPasswordEncoder(getPasswordEncoder());***

***return daoAuthenticationProvider;}***

***daoAuthenticationProvider(): Создает и настраивает DaoAuthenticationProvider,***

***который используется для аутентификации пользователей. В качестве источника данных о***

***пользователях используется PersonDetailsService, а пароли пользователей шифруются с***

***помощью getPasswordEncoder().***

***@Bean***

***public PasswordEncoder getPasswordEncoder() {***

***return new BCryptPasswordEncoder();}***

***41***

***getPasswordEncoder(): Возвращает объект PasswordEncoder для шифрования***

***паролей. В данном случае используется BCryptPasswordEncoder.***

***@Bean***

***public SecurityFilterChain securityFilterChain(HttpSecurity http) throws***

***Exception {***

***http***

***.csrf(Customizer.withDefaults())***

***.authorizeHttpRequests(auth -> auth***

***.requestMatchers("/item/\*\*",***

***"/person/\*\*").hasAnyRole("ADMIN")***

***.requestMatchers("/checkout",***

***"/thankyou").hasAnyRole("USER", "ADMIN")***

***.anyRequest().permitAll())***

***.formLogin(login -> login***

***.loginPage("/auth/login")***

***.permitAll()***

***.loginProcessingUrl("/process\_login")***

***.defaultSuccessUrl("/auth/account", false)***

***.failureUrl("/auth/login?error"))***

***.exceptionHandling(except -> except***

***.authenticationEntryPoint((request, response,***

***authException) -> {***

***response.sendRedirect("http://localhost:8080/auth/login");***

***}))***

***.logout(logout -> logout***

***.logoutUrl("/logout")***

***.logoutSuccessUrl("/auth/login"))***

***.sessionManagement((session) -> session***

***.invalidSessionUrl("/")***

***.sessionCreationPolicy(SessionCreationPolicy.ALWAYS)***

***.maximumSessions(1)***

***.maxSessionsPreventsLogin(true)***

***);***

***return http.build();***

***}***

***securityFilterChain(HttpSecurity http): Настраивает фильтр безопасности, который***

***определяет права доступа к различным URL-адресам приложения. Данный метод***

***определяет правила авторизации для конкретных URL, настраивает форму входа,***

***обрабатывает исключения аутентификации, настраивает выход из системы и управляет***

***сессиями пользователей.***

***Этот класс играет ключевую роль в обеспечении безопасности веб-приложения и***

***контролирует аутентификацию и авторизацию пользователей, а также управляет доступом***

***к различным ресурсам.***

***42***

***6.2.3. Настройки Приложений***

***Настройки приложения включают общие настройки, которые задаются с***

***использованием Spring Cloud Config Server, а также частные настройки, определяемые в***

***файле application.yaml.***

***Общие и частные настройки (Spring Cloud Config Server):***

***Spring Cloud Config Server позволяет централизованно управлять конфигурациями***

***приложения. Файлы на GitHub:***

***application-client.propertie: содержат общие настройки, которые могут включать***

***параметры базы данных, URL внешних сервисов и другие глобальные параметры.***

***server.port=0***

***eureka.client.service-url.defaultZone=http://localhost:8761/eureka***

***bank.properties: настройки для микросервиса оплаты.***

***spring.application.name=bank***

***spring.h2.console.enabled=true***

***spring.h2.console.path=/h2***

***spring.datasource.url=jdbc:h2:mem:born\_shop***

***spring.datasource.driverClassName=org.h2.Driver***

***spring.datasource.username=sa***

***spring.datasource.password=***

***spring.jpa.show-sql=true***

***spring.jpa.hibernate.ddl-auto=update***

***spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect***

***eserver.properties: настройки для микросервиса управления событиями.***

***server.port=8761***

***spring.application.name=eserver***

***eureka.client.register-with-eureka=false***

***eureka.client.fetch-registry=false***

***eureka.client.service-url.defaultZone=http://localhost:8761/eureka***

***gateway.properties: настройки для шлюза приложения.***

***server.port=8765***

***spring.application.name=gateway***

***spring.cloud.gateway.discovery.locator.enabled=true***

***spring.cloud.gateway.discovery.locator.lower-case-service-id=true***

***eureka.client.service-url.defaultZone=http://localhost:8761/eureka***

***43***

***shop.properties: настройки для основного приложения интернет-магазина.***

***server.port=8080***

***spring.application.name=shop***

***eureka.client.service-url.defaultZone=http://localhost:8761/eureka***

***spring.h2.console.enabled=true***

***spring.h2.console.path=/h2***

***spring.datasource.url=jdbc:h2:mem:born51***

***spring.datasource.driverClassName=org.h2.Driver***

***spring.datasource.username=sa***

***spring.datasource.password=***

***spring.jpa.show-sql=true***

***spring.jpa.hibernate.ddl-auto=update***

***spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect***

***spring.mvc.hiddenmethod.filter.enabled=true***

***Эти настройки загружаются приложением через Spring Cloud Config Server.***

***Частные настройки приложений (application.yaml):***

***Файл application.yaml содержит частные настройки, которые специфичны для***

***каждого экземпляра приложения или его окружения. Это могут быть параметры,***

***зависящие от конкретной среды развертывания (например, адреса и порты для***

***взаимодействия с другими сервисами), а также специфичные для приложения настройки***

***(например, логгирование, уровни доступа и т. д.).***

***application.yaml для основного приложения shop***

***spring.application.name=shop***

***spring.config.import=configserver:http://localhost:8888***

***spring.mail.host=smtp.gmail.com***

***spring.mail.port=587***

***spring.mail.username=born17121982@gmail.com***

***spring.mail.password=ltpuiznhkjmssety***

***spring.mail.properties.mail.smtp.auth=true***

***spring.mail.properties.mail.smtp.starttls.enable=true***

***management.endpoints.web.exposure.include=metrics,prometheus***

***management.endpoint.health.show-details=always***

***Наименование приложения:***

*** spring.application.name=shop: Задает имя приложения "shop".***

***Импорт конфигурации:***

*** spring.config.import=configserver:http://localhost:8888: Импортирует конфигурацию из***

***удаленного сервера конфигурации, расположенного по адресу http://localhost:8888.***

***44***

***Настройки почтового сервера:***

*** spring.mail.host=smtp.gmail.com: Указывает хост SMTP-сервера для отправки***

***электронной почты.***

*** spring.mail.port=587: Указывает порт SMTP-сервера.***

*** spring.mail.username=born17121982@gmail.com: Указывает имя пользователя для***

***аутентификации на SMTP-сервере.***

*** spring.mail.password=ltpuiznhkjmssety: Указывает пароль для аутентификации на***

***SMTP-сервере.***

*** spring.mail.properties.mail.smtp.auth=true: Указывает, что требуется аутентификация***

***при отправке электронной почты.***

*** spring.mail.properties.mail.smtp.starttls.enable=true: Указывает, что необходимо***

***включить STARTTLS для защиты соединения.***

***Управление конечными точками:***

*** management.endpoints.web.exposure.include=metrics,prometheus: Включает экспозицию***

***конечных точек metrics и prometheus для мониторинга приложения через веб.***

***Настройки конечной точки проверки состояния:***

*** management.endpoint.health.show-details=always: Указывает, что детальная информация***

***о состоянии должна всегда отображаться при обращении к конечной точке проверки***

***состояния (например, /actuator/health).***

***application.yaml для приложения bank***

***spring.application.name=bank***

***spring.profiles.active=client***

***spring.config.import=configserver:http://localhost:8888***

***Наименование приложения:***

*** spring.application.name=shop: Задает имя приложения "shop".***

***Активные профили:***

*** spring.profiles.active=client: Устанавливает активный профиль приложения как "client".***

***Это означает, что будут загружены конфигурационные файлы, соответствующие***

***профилю "client".***

***Импорт конфигурации:***

*** spring.config.import=configserver:http://localhost:8888: Импортирует конфигурацию из***

***удаленного сервера конфигурации, расположенного по адресу http://localhost:8888.***

***45***

***application.yaml для приложения config-server***

***server.port=8888***

***spring.application.name=config-server***

***spring.cloud.config.server.git.uri=https://github.com/Boraldan/configspringcloud.***

***git***

***spring.cloud.config.server.git.default-label=main***

***Порт сервера:***

*** server.port=8888: Задает порт, на котором будет запущен сервер, в данном случае -***

***порт 8888.***

***Наименование приложения:***

*** spring.application.name=config-server: Устанавливает имя приложения "config-server".***

***Это имя будет использоваться для идентификации приложения в среде выполнения.***

***Настройки сервера конфигурации в облаке:***

*** spring.cloud.config.server.git.uri=https://github.com/Boraldan/config-springcloud.git:***

***Указывает URI репозитория Git, откуда конфигурационные файлы будут загружены. В***

***данном случае, конфигурационные файлы будут загружены из репозитория по***

***указанному адресу.***

*** spring.cloud.config.server.git.default-label=main: Устанавливает метку (ветку) Git по***

***умолчанию для загрузки конфигурационных файлов. В данном случае, используется***

***метка "main".***

***application.yaml для приложения eserver***

***spring.application.name=eserver***

***spring.config.import=configserver:http://localhost:8888***

***Наименование приложения:***

*** spring.application.name=eserver: Задает имя приложения "eserver".***

***Импорт конфигурации:***

*** spring.config.import=configserver:http://localhost:8888: Импортирует конфигурацию из***

***удаленного сервера конфигурации, расположенного по адресу http://localhost:8888.***

***application.yaml для приложения gateway***

***spring.application.name=gateway***

***spring.config.import=configserver:http://localhost:8888***

***## Actuator***

***management.endpoint.gateway.enabled=true***

***management.endpoints.web.exposure.include=health***

***46***

***Наименование приложения:***

*** spring.application.name=gateway: Задает имя приложения "gateway".***

***Импорт конфигурации:***

*** spring.config.import=configserver:http://localhost:8888: Импортирует конфигурацию из***

***удаленного сервера конфигурации, расположенного по адресу http://localhost:8888.***

***Приложение будет получать свои настройки из этого внешнего источника***

***конфигурации.***

***Настройки управления (Actuator):***

*** management.endpoint.gateway.enabled=true: Включает конечную точку gateway для***

***управления. Это позволяет использовать управление для определенных аспектов***

***шлюза.***

*** management.endpoints.web.exposure.include=health: Включает экспозицию конечной***

***точки /actuator/health для мониторинга состояния приложения через веб.***

***6.2.4. Spring Cloud Компоненты:***

*** Spring Cloud Eureka: Служба регистрации и обнаружения микросервисов. Она***

***позволяет микросервисам находить друг друга и общаться между собой.***

***Добавление зависимости: В приложение Eureka Server:***

***<dependency>***

***<groupId>org.springframework.cloud</groupId>***

***<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>***

***</dependency>***

***@EnableEurekaServer: Приложение, помеченное этой аннотацией, будет работать как***

***реестр, который регистрирует микросервисы и позволяет им общаться друг с другом.***

***@SpringBootApplication***

***@EnableEurekaServer***

***public class EurekaServerApplication {***

***public static void main(String[] args) {***

***SpringApplication.run(EurekaServerApplication.class, args);***

***}***

***}***

***47***

*** Spring Cloud Config Server: Сервер конфигурации, который обеспечивает***

***централизованное управление конфигурациями для микросервисов.***

***Добавление зависимости: В приложение Config Server***

***<dependency>***

***<groupId>org.springframework.cloud</groupId>***

***<artifactId>spring-cloud-config-server</artifactId>***

***</dependency>***

***@EnableConfigServer: Эта аннотация включает сервер конфигураций Spring Cloud Config***

***Server. Приложение, помеченное этой аннотацией, будет обрабатывать запросы на***

***получение конфигураций из централизованного репозитория.***

*** Spring Cloud Gateway: Шлюз, который обеспечивает маршрутизацию запросов и***

***управление трафиком в микросервисной архитектуре.***

***Добавление зависимости: В приложение Gatewey***

***<dependency>***

***<groupId>org.springframework.cloud</groupId>***

***<artifactId>spring-cloud-starter-config</artifactId>***

***</dependency>***

*** Spring Cloud OpenFeign: Инструмент для создания клиентов REST API на основе***

***интерфейсов, что упрощает взаимодействие между микросервисами.***

***Используем Spring Cloud OpenFeign в приложении интернет-магазина (shop):***

***Добавление зависимости: В приложение shop добавлена зависимость на Spring Cloud***

***OpenFeign для обеспечения возможности создания клиентов REST API на основе***

***интерфейсов.***

***<dependency>***

***<groupId>org.springframework.cloud</groupId>***

***<artifactId>spring-cloud-starter-openfeign</artifactId>***

***</dependency>***

***Создание интерфейса: Для взаимодействия с микросервисом оплаты bank***

***создается интерфейс BankFeign. Аннотация @FeignClient указывает на имя микросервиса,***

***с которым будет установлено соединение. В этом интерфейсе определены методы для***

***вызова удаленных API, которые будут использоваться для выполнения операций оплаты.***

***48***

***@FeignClient(name = "bank")***

***public interface BankFeign {***

***@GetMapping***

***String getMany();***

***@PostMapping("/transfershop")***

***ResponseEntity<?> transferShop(@RequestBody PayDTO payDTO);***

***}***

***Использование в классах приложения: Для взаимодействия с микросервисом bank в***

***нужные классы shop внедряется поле типа BankFeign, которое будет использоваться для***

***вызова методов, определенных в интерфейсе.***

***@Controller***

***@RequiredArgsConstructor***

***public class CartController {***

***private final BankFeign bankFeign;***

***Например, при выполнении операций оплаты в приложении магазина можно***

***использовать метод transferShop, передавая соответствующие параметры.***

***@PostMapping("/thankyou")***

***public String creatAndPayOrder(Model model, @ModelAttribute("payDTO") PayDTO***

***payDTO) {***

***Cart cart = (Cart) session.getAttribute("cart");***

***if (cart.getCarList().isEmpty()) {***

***return "redirect:/cart";***

***}***

***Orders orders = orderServiceImpl.saveOrdersCar(cart);***

***try {***

***HttpStatusCode httpStatus =***

***bankFeign.transferShop(payDTO).getStatusCode();***

***if (httpStatus.is2xxSuccessful()) {***

***orders.setStatus(Status.PAID);***

***}***

***} catch (FeignException e) {***

***System.out.println(e.getMessage());***

***}***

***emailService.sendMessageThread(orders);***

***model.addAttribute("Order", orders);***

***cart.getCarList().clear();***

***cart.setCoupon(null);***

***return "/thankyou";***

***}***

***Эти компоненты обеспечивают основу для построения масштабируемых, гибких и***

***надежных микросервисных приложений, позволяя эффективно управлять конфигурацией,***

***безопасностью и коммуникацией между компонентами системы.***

***49***

***6.3. Разработка каталога и страницы товара.***

***Класс ShopController представляет собой контроллер веб-приложения, отвечающий за***

***взаимодействие с магазином автомобилей.***

***@Controller***

***@RequiredArgsConstructor***

***@RequestMapping("/shop")***

***public class ShopController {***

***private final CarService carService;***

***private final CartService cartService;***

***private final LotValidator lotValidator;***

***@RequestMapping("/shop"): Указывает базовый URL-адрес для всех методов***

***контроллера.***

***Поля класса:***

***private final CarService carService: Сервис, предоставляющий методы для работы с***

***автомобилями.***

***private final CartService cartService: Сервис, управляющий корзиной покупок.***

***private final LotValidator lotValidator: Валидатор, проверяющий корректность лота***

***(количества товаров).***

***Методы:***

***@GetMapping***

***public String shop(Model model,***

***@RequestParam(value = "page", required = false) Integer page,***

***@RequestParam(value = "byYear", required = false) boolean byYear)***

*** отображает список автомобилей с пагинацией на странице магазина. Обрабатывает***

***GET-запросы на страницу магазина. Принимает модель (Model), номер страницы***

***(page) и флаг сортировки по году (byYear). Возвращает имя представления "shop".***

***@GetMapping("/car/{id}")***

***public String getCar(Model model, @PathVariable("id") int id)***

*** отображает информацию об одном конкретном автомобиле. Обрабатывает GET-***

***запросы на страницу с информацией об одном автомобиле. Принимает модель (Model)***

***и идентификатор автомобиля (id). Возвращает имя представления "car".***

***@PostMapping("/car/{id}")***

***public String addToCart(@ModelAttribute("lot") Lot lot, @PathVariable("id")***

***int id, Model model, BindingResult bindingResult)***

*** добавляет выбранный автомобиль в корзину покупок, проверяя корректность***

***выбранного количества товаров с помощью LotValidator.***

***50***

*** метод принимает объект Lot, идентификатор автомобиля (id), модель (Model) и***

***результаты привязки (BindingResult). Возвращает имя представления "car" или***

***перенаправляет на страницу с информацией об автомобиле.***

***public Car validateLot(Object target, Errors errors) {***

***Lot lot = (Lot) target;***

***Car validCar = carService.findById(lot.getId()).get();***

***if (validCar.getVolume() < lot.getLot() || lot.getLot() < 1) {***

***errors.rejectValue("lot", "", "Нет такого количества");***

***return validCar;***

***}***

***return validCar;***

***}***

***Представления магазина и страницы с информацией об автомобиле создаются***

***динамически с использованием Thymeleaf, что позволяет отображать актуальную***

***информацию в зависимости от контекста запроса.***

***6.4. Реализация функционала Корзины и оформления заказов.***

***Корзина в интернет-магазине - это виртуальный контейнер, в который пользователь***

***добавляет выбранные товары перед их покупкой. Она обеспечивает удобство покупателя,***

***позволяя ему просматривать и редактировать выбранные товары, отслеживать стоимость***

***покупок и сохранять информацию о покупках. Для магазина корзина представляет***

***ценность в анализе покупательского поведения и оптимизации ассортимента и***

***маркетинговых стратегий.***

***Последовательность реализации:***

***Переопределяем, путём наследовании, работу класса HttpSessionEventPublisher,***

***предоставляемый Spring, который отвечает за публикацию событий сеанса HTTP.***

***51***

***@RequiredArgsConstructor***

***public class SessionEventService extends HttpSessionEventPublisher {***

***private final CarService carService;***

***@Override***

***public void sessionCreated(HttpSessionEvent event) {***

***event.getSession().setAttribute("cart", new Cart());***

***event.getSession().setMaxInactiveInterval(500);***

***super.sessionCreated(event); }***

***Функциональность данного класса заключается в том, чтобы обеспечить создание и***

***инициализацию корзины покупок Cart() для каждого нового сеанса пользователя. Также***

***он устанавливает максимальное время неактивности сеанса, после которого сеанс будет***

***автоматически закрыт. Это помогает эффективно управлять ресурсами сервера и***

***безопасностью приложения.***

***При обращение к интренет-магазину все запросы ловит класс GlobalController,***

***который используется для глобальной обработки запросов в приложении.***

***@ControllerAdvice***

***@RequiredArgsConstructor***

***public class GlobalController {***

***private final HttpSession session;***

***@ModelAttribute("cart")***

***public Cart addCart() {***

***return (Cart) session.getAttribute("cart");***

***}}***

*** addCart() возвращает объект корзины покупок, полученный из сессии пользователя***

***по ключу "cart". Таким образом, корзина покупок становится доступной для***

***использования в представлениях при обработке запросов в контроллерах.***

***Класс GlobalController обеспечивает удобный способ сделать общие атрибуты или***

***объекты доступными для всех контроллеров в приложении. В данном случае, он***

***обеспечивает доступ к объекту корзины покупок из сессии, что позволяет использовать***

***эту информацию в любом контроллере без необходимости повторного извлечения из***

***сессии.***

***Класс CartController является одной из основных частей веб-приложения и отвечает***

***за управление корзиной покупок. В данном классе реализованы методы для добавления,***

***удаления товаров, проверки купонов, оформления заказа и его оплаты.***

***52***

***@Controller***

***@RequiredArgsConstructor***

***public class CartController {***

***private final HttpSession session;***

***private final CartService cartService;***

***private final OrderServiceImpl orderServiceImpl;***

***private final EmailService emailService;***

***private final CouponValid couponValid;***

***private final SellerService sellerService;***

***private final BankFeign bankFeign;***

*** HttpSession session: Поле для доступа к сеансу HTTP, используемому для хранения***

***информации о сеансе пользователя, включая корзину покупок.***

*** CartService cartService: Сервис, который предоставляет методы для управления***

***содержимым корзины покупок.***

*** OrderServiceImpl orderServiceImpl: Реализация сервиса заказов, используемая для***

***создания и сохранения заказов на основе содержимого корзины.***

*** EmailService emailService: Сервис отправки электронных писем, используемый для***

***отправки уведомлений о заказах.***

*** CouponValid couponValid: Компонент для проверки корректности купонов,***

***применяемых к покупкам.***

*** SellerService sellerService: Сервис для работы с информацией о продавцах, например,***

***для получения данных о продавце при оформлении заказа.***

*** BankFeign bankFeign: Клиент Feign для взаимодействия с внешним банковским***

***сервисом, например, для обработки платежей.***

***Методы***

***@GetMapping("/cart")***

***public String shop(@RequestParam(value = "car", required = false) Integer***

***idCar, Model model)***

*** при передаче идентификатора товара (idCar) выполняется удаление этого товара из***

***корзины путем вызова метода cartService.dellItem(idCar). Иначе, создается атрибут***

***корзины и купона, и возвращается страница корзины покупок.***

***53***

***@PostMapping("/cart")***

***public String checkLot(@ModelAttribute("cart") Cart cart)***

*** проверяет правильность количества товаров в корзине покупок. Если данные***

***некорректны, после выполнения cartService.checkFalseLot(cart, oldCarList)***

***пользователь перенаправляется на страницу корзины с сообщением об ошибке. В***

***противном случае происходит обновление корзины и перенаправление на страницу***

***корзины покупок.***

***@PostMapping("/cart/coupon")***

***public String checkCoupon(@ModelAttribute("coupon") @Valid Coupon coupon,***

***BindingResult bindingResult)***

*** проверяет введенный купон на корректность. Если купон неверный, после***

***выполнения couponValid.validateCoupon(coupon, bindingResult), пользователь***

***возвращается на страницу корзины с сообщением об ошибке. В случае успешной***

***валидации, купон применяется к корзине и происходит перенаправление на страницу***

***корзины покупок.***

***@GetMapping("/checkout")***

***public String checkout(Model model, @AuthenticationPrincipal PersonDetails***

***personDetails)***

*** отображает страницу оформления заказа с данными о продавце, пользователе и***

***объекте для оплаты. Если корзина пуста, пользователь перенаправляется на***

***страницу корзины.***

***@GetMapping("/thankyou")***

***public String creatOrder(Model model)***

*** создает и сохраняет заказ на основе содержимого корзины. После этого отправляется***

***уведомление о заказе по электронной почте, и происходит очистка корзины и купона.***

***@PostMapping("/thankyou")***

***public String creatAndPayOrder(Model model, @ModelAttribute("payDTO") PayDTO***

***payDTO)***

*** создает и сохраняет заказ, а затем пытается выполнить оплату через сервис банка***

***используя bankFeign.transferShop(payDTO). В случае успешной оплаты, статус заказа***

***54***

***изменяется на "оплачен". После этого отправляется уведомление о заказе по***

***электронной почте методом emailService.sendMessageThread(orders), который***

***реализуется в отдельном потоке. Далее происходит очистка корзины и купона.***

***@GetMapping("/orders/cancel")***

***public String canselOrders(@RequestParam(value = "id", required = false)***

***Integer id)***

*** отменяет заказ по его идентификатору и перенаправляет на страницу аккаунта***

***пользователя методом orderServiceImpl.cancelOrders(id), если идентификатор заказа***

***предоставлен.***

***Класс CartController в связке с вспомогательными классами, обеспечивает полный***

***цикл управления корзиной покупок, начиная с добавления товаров, проверки купонов и***

***заканчивая оформлением и оплатой заказа.***

***6.5. Авторизация пользователя***

***Класс контроллер AuthController отвечает за обработку запросов, связанных с***

***аутентификацией и авторизацией пользователей.***

***@Controller***

***@RequestMapping("/auth")***

***@RequiredArgsConstructor***

***public class AuthController {***

***private final RegistrationService registrationService;***

***private final PersonValidator personValidator;***

***private final PersonDTOValid personDTOValid;***

***private final PersonService personService;***

***private final ModelMapper modelMapper;***

***55***

***Поля класса:***

*** registrationService: Сервис, отвечающий за регистрацию пользователей.***

*** personValidator: Валидатор объектов типа Person.***

*** personDTOValid: Валидатор DTO (Data Transfer Object) объектов типа PersonDTO.***

*** personService: Сервис, предоставляющий функциональность для работы с объектами***

***типа Person.***

*** modelMapper: Инструмент для маппинга объектов из одного типа в другой.***

***Методы:***

***@GetMapping("/login")***

***public String loginPage() {***

***if (!getRole().equals("ROLE\_ANONYMOUS")) {***

***return "redirect:/auth/account"; }***

***return "auth/login"; }***

*** Если пользователь аутентифицирован (его роль не "ROLE\_ANONYMOUS"), метод***

***перенаправляет на страницу аккаунта (/auth/account), иначе отображает страницу***

***входа (auth/login) для аутентификации или регистрации.***

***@GetMapping("/registration")***

***public String registrationPage(@ModelAttribute("person") Person person) {***

***if (!getRole().equals("ROLE\_ANONYMOUS")) {***

***return "redirect:/auth/login"; }***

***return "auth/registration"; }***

*** Если пользователь аутентифицирован (его роль не "ROLE\_ANONYMOUS"), метод***

***перенаправляет на страницу входа (/auth/login), иначе отображает страницу***

***регистрации (auth/registration).***

***56***

***@PostMapping("/registration")***

***public String performRegistration(@ModelAttribute("person") @Valid Person***

***person, BindingResult bindingResult)***

*** метод обрабатывает POST-запросы для выполнения процесса регистрации***

***пользователя. Производит валидацию введенных данных с помощью personValidator,***

***регистрирует пользователя через registrationService при отсутствии ошибок при вводе***

***всех необходимых полей и перенаправляет на страницу входа (/auth/login).***

***@GetMapping("/account")***

***public String accountPage(Model model) {***

*** метод обрабатывает GET-запросы для отображения страницы аккаунта пользователя.***

***Получает данные пользователя через personService, проверяет его роль, и, если***

***пользователь аутентифицирован и имеет роль "ROLE\_USER" или "ROLE\_ADMIN",***

***отображает страницу аккаунта (auth/account). В случае роли "ROLE\_ADMIN",***

***перенаправляет на страницу администратора (/admin). Если пользователь не***

***аутентифицирован или у него нет правильной роли, перенаправляет на страницу***

***входа (auth/login).***

***@GetMapping("/account/alter")***

***public String accountAlter(Model model)***

*** метод обрабатывает GET-запросы для отображения страницы редактирования данных***

***аккаунта пользователя. Получает данные аутентифицированного пользователя через***

***SecurityContextHolder, проверяет его роль, и, если пользователь не анонимен,***

***отображает страницу редактирования данных аккаунта (auth/alter). В противном***

***случае, перенаправляет на страницу аккаунта (auth/account).***

***57***

***@PostMapping("/account/alter")***

***public String postAccountAlter(@ModelAttribute("personDTO") @Valid PersonDTO***

***personDTO, BindingResult bindingResult)***

*** метод обрабатывает POST-запросы для применения изменений в данных аккаунта***

***пользователя. Производит валидацию введенных данных с помощью personDTOValid,***

***применяет изменения через personService при отсутствии ошибок и перенаправляет на***

***страницу аккаунта (/auth/account).***

***Эти методы предоставляют основной функционал для обработки запросов, связанных***

***с аутентификацией, регистрацией и управлением аккаунтом пользователей в веб-***

***приложении.***

***6.6. Разработка личного кабинета администратора.***

***Кабинет администратора в интернет-магазине предназначен для управления***

***различными аспектами функционирования магазина. Основные цели и функциональность***

***кабинета администратора:***

*** Управление товарами и каталогом: Администратор может добавлять, редактировать и***

***удалять товары, управлять их категориями, характеристиками, изображениями и***

***описаниями. Он может управлять наличием товаров, устанавливать цены и скидки.***

*** Управление заказами: Администратор имеет доступ к списку заказов, может***

***просматривать информацию о заказах, подтверждать или отменять заказы, изменять***

***их статусы, управлять доставкой и оплатой.***

*** Управление пользователями: Администратор может управлять списком пользователей***

***магазина, просматривать информацию о них, редактировать профили пользователей,***

***блокировать или удалять учетные записи, назначать права доступа.***

*** Управление акциями и скидками: Администратор может создавать и редактировать***

***акции, скидки и специальные предложения для привлечения покупателей, управлять***

***условиями и сроками действия акций.***

*** Управление контентом и информацией: Администратор может управлять***

***содержимым сайта, например, добавлять новости, статьи, блоги, обновлять***

***информацию о доставке, контактную информацию и другие разделы сайта.***

*** Отчетность и аналитика: Администратор имеет доступ к отчетам и статистике о***

***продажах, посещаемости сайта, поведении покупателей и другим аналитическим***

***данным, что позволяет принимать обоснованные решения для развития бизнеса.***

***58***

***В проекте реализованы функции:***

*** управление товарами***

*** управление пользователями***

***Два контроллера PersonController и ItemController предоставляют***

***функциональность управления данными в личном кабинете администратора, включая***

***операции CRUD (Create, Read, Update, Delete).***

***@Controller***

***@RequiredArgsConstructor***

***@RequestMapping("/item")***

***public class ItemController {***

***private final ModelMapper modelMapper;***

***private final CarService carService;***

*** Отвечает за управление информацией о предметах (в данном случае, автомобилях).***

*** Позволяет просматривать, создавать, редактировать и удалять записи о предметах.***

*** Предоставляет возможность администратору просматривать список предметов***

***(автомобилей), добавлять новые предметы, редактировать их данные и удалять***

***предметы из системы.***

***59***

***@Controller***

***@RequestMapping("/person")***

***@RequiredArgsConstructor***

***public class PersonController {***

***private final PersonService personService;***

***private final RegistrationService registrationService;***

***private final PersonValidator personValidator;***

***private final PersonDTOValid personDTOValid;***

***private final ModelMapper modelMapper;***

*** Отвечает за управление информацией о пользователях.***

*** Позволяет просматривать, создавать, редактировать и удалять записи о пользователях.***

*** Предоставляет возможность администратору просматривать список пользователей,***

***добавлять новых пользователей, редактировать их данные и удалять пользователей из***

***системы.***

***60***

***Оба контроллера используют аутентификацию для проверки роли пользователя: если***

***пользователь не является администратором, ему будет отказано в доступе к операциям***

***создания и редактирования. Кроме того, контроллеры обрабатывают валидацию данных***

***перед их сохранением и предоставляют соответствующие представления для ввода***

***данных и отображения результатов операций.***

***6.7. Тестирование приложения оплаты Bank***

***В данном модуле для тестирования используются следующие библиотеки:***

***Mockito: библиотека для создания макетов (mock objects) в юнит-тестировании Java-***

***приложений. Она позволяет создавать фиктивные объекты, которые имитируют***

***поведение реальных объектов. Это особенно полезно для изоляции тестируемого кода от***

***его зависимостей, таких как внешние сервисы или репозитории базы данных.***

*** Простота использования: предоставляет простой и понятный API для создания***

***макетов и настройки их поведения.***

*** Гибкость: можно эффективно тестировать различные сценарии и случаи***

***использования, изменяя поведение макетов в зависимости от тестовых условий.***

*** Читабельность: Тесты легко читаются и понимаются, что упрощает поддержку кода и***

***обнаружение ошибок.***

*** Интеграция с JUnit: Mockito хорошо интегрируется с JUnit, что облегчает написание и***

***запуск тестов.***

***MockitoExtension: расширение для интеграции Mockito с JUnit. Оно предоставляет***

***дополнительные возможности для автоматической инициализации макетов и их***

***внедрения в тестируемые объекты.***

***61***

*** Автоматическая инициализация макетов: автоматически инициализирует объекты,***

***помеченные аннотацией @Mock, и внедряет их в объекты, помеченные аннотацией***

***@InjectMocks, что уменьшает повторяемый код в тестах.***

*** Поддержка новых функций JUnit: позволяет использовать новые функции JUnit 5***

***вместе с Mockito, такие как параметризованные тесты, повторяемые тесты и***

***динамическое группирование тестов.***

***В итоге, использование библиотек Mockito и MockitoExtension в данном проекте***

***упрощает написание тестового кода, улучшает его читаемость и обеспечивает надежное***

***тестирование функциональности приложения.***

***Пример кода***

***@ExtendWith(MockitoExtension.class)***

***public class BankPayTest {***

***@Mock***

***private AccountRepository accountRepository;***

***@InjectMocks***

***private TransferService transferService;***

***@Test***

***public void transactionExpectCorrect(){***

***Long source = 11111L;***

***Account sourceAccount = new Account();***

***sourceAccount.setCard(source);***

***BigDecimal sourceBalance = new BigDecimal(1000);***

***sourceAccount.setAmount(sourceBalance);***

***Long destination = 55555L;***

***Account destinationAccount = new Account();***

***destinationAccount.setCard(destination);***

***BigDecimal destinationBalance = new BigDecimal(0);***

***destinationAccount.setAmount(destinationBalance);***

***BigDecimal sum = new BigDecimal(10);***

***given(accountRepository.findByCard(source))***

***.willReturn(sourceAccount);***

***given(accountRepository.findByCard(destination))***

***.willReturn(destinationAccount);***

***transferService.transferShop(sourceAccount.getCard(),***

***destinationAccount.getCard(), sum);***

***verify(accountRepository).findByCard(source);***

***verify(accountRepository).findByCard(destination);***

***verify(accountRepository).save(sourceAccount);***

***verify(accountRepository).save(destinationAccount);***

***Assertions.assertEquals(sourceBalance.subtract(sum),***

***sourceAccount.getAmount());***

***Assertions.assertEquals(destinationBalance.add(sum),***

***destinationAccount.getAmount());***

***}}***

***Приложение № 6 Итоги тестирования модуля оплаты сервиса Bank***

***62***

***7. Мониторинг приложения с помощью Actuator, Micrometer, Prometheus, Grafana***

***Для обеспечения эффективного мониторинга приложений на основе Spring Boot***

***чрезвычайно важно использовать соответствующие инструменты. В рамках данного***

***раздела будет рассмотрен процесс настройки и использования Actuator, Micrometer,***

***Prometheus и Grafana для сбора, анализа и визуализации метрик приложений.***

***Настройка Actuator:***

***Actuator представляет собой набор инструментов Spring Boot, обеспечивающих***

***доступ к различным конечным точкам для мониторинга и управления приложением. Для***

***интеграции Actuator необходимо добавить соответствующую зависимость в файл pom.xml***

***проекта:***

***<dependency>***

***<groupId>org.springframework.boot</groupId>***

***<artifactId>spring-boot-starter-actuator</artifactId>***

***</dependency>***

***Применяем настройки к Actuator:***

***management:***

***endpoints:***

***web:***

***exposure:***

***include: metrics,prometheus***

***endpoint:***

***health:***

***show-details: always***

***management.endpoints.web.exposure.include: metrics, prometheus:***

*** Эта конфигурация определяет, какие конечные точки (endpoints) Actuator будут***

***доступны через HTTP-запросы.***

*** В данном случае, конечные точки metrics и prometheus будут доступны из внешних***

***источников, таких как браузер или другие приложения через HTTP-запросы.***

*** Конечная точка metrics обычно предоставляет информацию о метриках приложения, а***

***prometheus обычно используется для сбора метрик в формате, совместимом с***

***Prometheus.***

***management.endpoint.health.show-details: always:***

*** Эта конфигурация определяет, какие детали о состоянии здоровья (health) приложения***

***будут показываться при запросе состояния здоровья.***

*** Значение always означает, что детальная информация о состоянии здоровья всегда***

***будет включена в ответ при запросе состояния здоровья.***

***63***

*** Это позволяет получить дополнительную информацию о состоянии здоровья***

***приложения, помимо базового статуса, такого как "UP" или "DOWN".***

***После этого Actuator будет автоматически включен в приложение, предоставляя***

***доступ к различным конечным точкам мониторинга.***

***Интеграция с Micrometer:***

***Micrometer используется для сбора метрик приложения. Для интеграции Micrometer***

***с приложением необходимо добавить соответствующую зависимость в файл pom.xml:***

***<dependency>***

***<groupId>io.micrometer</groupId>***

***<artifactId>micrometer-core</artifactId>***

***<version>1.12.3</version>***

***</dependency>***

***Это позволит приложению экспортировать метрики в формате, совместимом с***

***Prometheus.***

***Настройка Prometheus:***

***Prometheus представляет собой систему мониторинга и оповещения. Для***

***интеграции приложением необходимо добавить соответствующую зависимость в файл***

***pom.xml:***

***<dependency>***

***<groupId>io.micrometer</groupId>***

***<artifactId>micrometer-registry-prometheus</artifactId>***

***<version>1.12.3</version>***

***</dependency>***

***Для настройки Prometheus необходимо скачать и настроить его для конкретной***

***операционной системы. Далее следует добавить конфигурацию для сбора метрик от***

***приложения в файл конфигурации Prometheus (prometheus.yml):***

***64***

***Затем запустите Prometheus, указав путь к файлу конфигурации.***

***65***

***Настройка Grafana:***

***Grafana - платформа для визуализации данных из различных источников. Для***

***настройки Grafana необходимо скачать и настроить его для конкретной операционной***

***системы. Затем добавьте источник данных Prometheus в Grafana, указав URL вашего***

***сервера Prometheus. Создайте дашборд в Grafana и добавьте необходимые графики и***

***панели для визуализации метрик вашего приложения.***

***Использование Actuator, Micrometer, Prometheus и Grafana позволяет создать***

***мощную систему мониторинга и управления для приложений на основе Spring Boot.***

***Actuator предоставляет удобные конечные точки для мониторинга и управления***

***приложением, Micrometer обеспечивает сбор и экспорт разнообразных метрик, Prometheus***

***эффективно сохраняет и анализирует эти метрики, а Grafana предоставляет гибкие***

***средства визуализации данных. Вместе эти инструменты обеспечивают обширный набор***

***функций для отслеживания производительности, выявления проблем и принятия***

***информированных решений для оптимизации работы приложения.***

***66***

***8. Заключение***

***8.1. Общий обзор выполненной работы.***

***В ходе данного проекта был разработан интернет-магазин, охватывающий***

***различные аспекты функционирования такого приложения, включая административную***

***панель, личный кабинет пользователя, корзину, оформление заказа, а также процессы***

***авторизации и аутентификации. Проект был организован с использованием современных***

***технологий и инструментов разработки программного обеспечения, обеспечивающих его***

***эффективность и масштабируемость.***

***В разработке интернет-магазина использовались следующие технологии и***

***инструменты:***

*** Spring Framework: фреймворк для разработки приложений на языке Java.***

*** Spring Boot: инструмент для упрощения создания приложений на основе Spring***

***Framework.***

*** Spring MVC: модуль для разработки веб-приложений с использованием шаблона***

***проектирования MVC.***

*** Spring Data JPA и JpaRepository: модули для упрощения доступа к данным и работы с***

***базами данных с использованием Java Persistence API.***

*** Spring Security: модуль для обеспечения безопасности приложения, включая***

***аутентификацию и авторизацию пользователей.***

*** Spring Integration: интеграционный фреймворк для реализации различных паттернов***

***взаимодействия между компонентами приложения.***

*** Spring Boot Starter Mail: стартер для работы с электронной почтой в приложениях на***

***Spring Boot.***

*** ModelMapper: библиотека для преобразования объектов одного типа в объекты***

***другого типа.***

*** H2 Database: встроенная база данных для разработки и тестирования приложений.***

*** Thymeleaf: шаблонизатор для создания веб-страниц с использованием HTML и Java.***

*** Lombok: библиотека для автоматической генерации кода в Java.***

***67***

***8.2. Полученные результаты и достижения.***

***Благодаря проделанной работе был создан полноценный интернет-магазин с***

***удобным пользовательским интерфейсом, функциональным административным разделом***

***и надежной системой обработки заказов. Реализация нефункциональных требований,***

***таких как масштабируемость, безопасность и мониторинг, позволила создать надежное и***

***производительное приложение.***

***8.3. Выводы и дальнейшие перспективы развития проекта.***

***Планируется дальнейшее развитие проекта в направлении создания полноценного***

***микросервисного приложения. Переход к микросервисной архитектуре позволит еще***

***больше улучшить масштабируемость и гибкость системы. Предполагается разделение***

***функционала на отдельные сервисы, такие как магазин (Shop), банк (Bank), хранилище***

***(Storage), заказы (Order), учетные записи (Account), корзина (Cart), веб-клиент (Web-client)***

***и сервер авторизации на базе KeyCloak.***

***Модули Spring Cloud, такие как Eureka, Config Server и Gateway, будут***

***использоваться для упрощения интеграции и управления микросервисами. Это позволит***

***создать более гибкую и расширяемую систему, способную эффективно решать задачи***

***электронной коммерции. Кроме того, планируется внедрение инструментов мониторинга,***

***таких как Actuator, Micrometer, Prometheus и Grafana, для непрерывного контроля и***

***анализа производительности приложения.***

***68***

***9. Приложения***

***9.1. Код приложения «Интернет-магазин»***

***Код приложения прикладывается к архиву.***

***9.2. Дополнительные материалы.***

***69***

***Приложение № 1 Use-Case диаграмма пользователя***

***70***

***Приложение № 2 Use-Case диаграмма администратора***

***71***

***Приложениe № 3 Таблица ERD.***

***72***

***Приложение № 4 IML диаграмма классов монолитного приложения Shop***

***73***

***Приложение № 5 IML диаграмма классов микросервиса Bank***

***74***

***Приложение № 6 Итоги тестирования модуля оплаты сервиса Bank\_\_***