Geekbrains

**Разработка интернет-магазина с использованием Spring**

Программа: Разработчик

Специализация: Веб-разработка на Java

ФИО: Дёмочкин Алексей Владимирович

Москва

2024

## Содержание

[**Содержание 2**](#_1fob9te)

[**Введение 3**](#_2et92p0)

Исследование рынка интернет-магазинов 4

Определение требований к будущему сервису 5

[**Основы разработки приложений с помощью Spring 6**](#_tyjcwt)

[Что такое язык программирование, фреймворк 6](#_3dy6vkm)

[Что такое База Данных 7](#_1t3h5sf)

[Сборщик проектов Apache Maven 8](#_4d34og8)

[Библиотеки которые будут использоваться 9](#_2s8eyo1)

[Контейнеризация с помощью Docker 11](#_17dp8vu)

[Работа с контролем версий с помощью Git 12](#_3rdcrjn)

[**Разработка приложения 13**](#_26in1rg)

[Spring initializr 13](#_lnxbz9)

[Pom.xml 14](#_35nkun2)

[Cоздание Git репозитория 15](#_1ksv4uv)

[Развертывание контейнера MySQL в Docker. 16](#_44sinio)

[Entities 17](#_2jxsxqh)

[Репозитории 18](#_z337ya)

[Сервисы 19](#_3j2qqm3)

[Контроллеры 20](#_1y810tw)

[ExceptionHandler 21](#_4i7ojhp)

[Spring Security Configuration 22](#_2xcytpi)

[Custom UserDetailsService 23](#_1ci93xb)

[Password Encoder 23](#_3whwml4)

[Заключение 24](#_2bn6wsx)

[**Используемые технологии 24**](#_qsh70q)

[Состав команды 25](#_3as4poj)

**Список используемой литературы 26**

[**Приложения 27**](#_49x2ik5)

# Введение

В современном мире интернет-торговля становится все более популярной, и многие компании стремятся создать свой собственный интернет-магазин. Статистика продаж в онлайне растет с каждым годом, особенно среди молодежи и людей среднего возраста. Основными причинами этого являются удобство, доступность и широкий выбор товаров.

Java является хорошим языком для разработки интернет-магазина из-за его надежности, производительности и широкого сообщества разработчиков. Он также предлагает множество библиотек и фреймворков, которые упрощают разработку веб-приложений. Кроме того, Java имеет высокую степень абстракции, что позволяет разработчикам создавать более эффективные и модульные программы.

Spring Boot является одним из популярных фреймворков Java для создания веб-приложений, и в данной дипломной работе мы рассмотрим, как использовать Spring Boot для разработки собственного интернет-магазина.

Целью данной работы является разработка интернет-магазина на основе Spring Boot с использованием современных технологий и подходов. Мы рассмотрим процесс проектирования архитектуры, разработки базы данных, создания пользовательского интерфейса и реализуем авторизацию пользователей с помощью Spring Security. Также мы проведем анализ существующих решений на рынке и оценим экономическую эффективность разработанного интернет-магазина.

# 

## Исследование рынка интернет-магазинов

Если несколько лет назад можно было говорить о том, что рынок интернет-торговли быстроразвивающаяся сфера, которая может составить конкуренцию офлайн-магазинам, то сейчас эти две отрасли ведения бизнеса практически равнозначны по уровню объема продаж.

Рынок электронной коммерции глобально поделен на два вида: маркетплейсы и интернет-магазины. Первые имеют большие обороты и их популярность за последние годы стремительно выросла, в то же время практически все крупные компании имеют свой интернет-магазин, который также пользуется спросом.

Среди наиболее популярных интернет-магазинов в России можно назвать: mvideo.ru, lamoda.ru, samokat.ru, vkusvill.ru, dns-shop.ru, apteka.ru, goldapple.ru, detmir.ru.

Лидирующие строчки в статистике по объему продаж занимают сразу три популярнейших маркетплейса: wildberries.ru, ozon.ru и market.yandex.ru.

Если рассматривать мировую статистику продаж в электронной коммерции, то первые два места в топе занимают такие интернет-магазины, как Amazon и eBay. Следует отметить, что именно эти сайты написаны на языке Java. Это как ничто другое подчеркивает его актуальность и преимущество, несмотря на то, что языку уже более 25 лет.

## Определение требований к будущему сервису

Нередко предприниматели, которые задаются целью создать интернет-магазин для своего бизнеса обращаются к IT-специалистам для решения нестандартных задач. Чаще это происходит, при продаже нетипичного для интернет-магазинов товара, например весового товара или отрезного, такого как пряжа или ткани. В таком случае не всегда подойдет готовый шаблон для создания интернет-магазина и рациональней будет написание сайта с нуля, с возможностью добавления к нему всевозможных функций. Для того, чтобы процесс наслаивания возможностей и расширения функционала был наиболее гладким необходима четкая и уверенная основа. - поискать теорию про разработку интернет-магазинов, что самое важно ?

В связи с этим, основным требованием к будущему сервису является то, что мы должны создать устойчивый, функционирующий сервис который включает в себя самые необходимые функции, такие как:

1. Авторизация
2. База данных для хранения информации о товарах, покупателях и заказах.
3. Пользовательский интерфейс для просмотра товаров, оформления заказов.
4. Функционал для обработки заказов, включая управление товарами и обработку платежей.
5. Интеграция разработанного интернет-магазина с платежными системами и службами доставки.

# Основы разработки приложений с помощью Spring

### Что такое язык программирование, фреймворк

Язык программирования — формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под её управлением.

Со времени создания первых программируемых машин человечество придумало более восьми тысяч языков программирования. Каждый год их число увеличивается. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Например о таких языках как Java или Python в современном мире мало кто не слышал, а вот о Delphi и VBA знают далеко не многие.

В данной дипломной работе я буду использовать язык программирования Java и фреймворк Spring. Java является одним из наиболее популярных и широко используемых языков программирования в мире. Он обладает рядом преимуществ. Он надежен и имеет строгий контроль типов, это обеспечивает стабильность работы вашего приложения. Java легко масштабируется благодаря поддержке многопоточности и многопроцессорности. Имеет огромное число сторонних библиотек и инструментов, которые можно использовать для разработки приложений. Несмотря на свою долгую историю, Java продолжает развиваться и оставаться актуальным инструментом для разработчиков.

Фреймворк Spring – это прикладной фреймворк, который является частью экосистемы Java. Прикладной фреймворк – это пакет типичных функций программного обеспечения, образующих базовую структуру для разработки приложения, позволяя тратить меньше времени на написание приложения, так как не приходится создавать весь код программы с нуля. Spring является целой экосистемой фреймворков. В разработке моего интернет-магазина я сделаю упор на Spring MVC (model-view-controller, «модель-представление-контроллер». Это часть фреймворка позволяет создавать веб-приложения, обрабатывающие HTTP-запросы.

HTTP (Hypertext Transfer Protocol) - это протокол, который используется для передачи данных между веб-серверами и веб-браузерами. HTTP запросы - это запросы, которые отправляются от браузера на сервер, чтобы получить информацию или выполнить действие на веб-сайте. Например, когда вы вводите адрес сайта в адресной строке браузера, ваш компьютер отправляет HTTP запрос на сервер, чтобы получить HTML-страницу этого сайта.

### Что такое База Данных

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные в наиболее распространенных типах современных баз данных обычно хранятся в виде строк и столбцов формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL). Рассмотри реляционные базы данных и нереляционные.

Реляционные базы данных (РБД) - это один из основных типов баз данных, который использует табличное представление данных. В РБД данные хранятся в таблицах, которые состоят из строк (записей) и столбцов (атрибутов). Каждая строка таблицы представляет отдельную запись, а каждый столбец содержит значения одного типа. РБД имеют ряд преимуществ перед другими типами баз данных, такими как простота структуры данных, эффективность поиска и обработки данных, а также возможность использования стандартного языка запросов SQL для работы с данными. Однако, РБД могут быть менее эффективными при работе с большими объемами данных и сложными структурами данных.

SQL (Structured Query Language) - это язык программирования для хранения и обработки информации в реляционной базе данных. Реляционная база данных хранит информацию в табличной форме со строками и столбцами, представляющими различные атрибуты данных и различные связи между значениями данных. Инструкции SQL можно использовать для хранения, обновления, удаления, поиска и извлечения информации из базы данных. Можно также использовать SQL для поддержания и оптимизации производительности базы данных.

Нереляционные базы данных - это тип баз данных, которые не используют строгую табличную структуру данных, как это делают реляционные базы данных. Вместо этого, данные в этих базах могут храниться в различных форматах, таких как документы, графы, тексты и т.д.. Эти базы данных могут быть более эффективными для работы с большими объемами неструктурированных или полуструктурированных данных, но они могут быть менее удобными для работы со строгими табличными данными, которые характерны для реляционных баз данных.

NoSQL базы данных - это класс нереляционных систем управления базами данных, которые предлагают гибкость и масштабируемость для работы с большими объемами данных. NoSQL базы данных обычно не используют структурированный язык запросов SQL и вместо этого предлагают различные модели данных, такие как документные, столбцовые, графовые и другие. Примеры NoSQL баз данных включают Apache Cassandra, MongoDB, Redis и Couchbase.

В своей работе я буду использовать реляционную базу данных MySQL. Я выбрал эту базу данных потому, что MySQL имеет простой и понятный интерфейс, который облегчает работу с ним, обладает высокой производительностью и масштабируемостью, а так же имеет большое сообщество пользователей.

### Сборщик проектов Apache Maven

Apache Maven - это инструмент управления проектами для Java-разработки. Он помогает в управлении зависимостями, компиляции, тестировании и сборке проектов. Maven позволяет создавать проекты с использованием стандартных форматов и шаблонов, а также предоставляет возможность настройки проекта под конкретные нужды.

Основная цель Maven - позволить разработчику в кратчайшие сроки получить полное представление о ходе разработки. Для достижения этой цели Maven решает несколько задач:

* Обеспечение единой системы сборки
* Предоставление качественной информации о проекте
* Поощрение лучших практик разработки
* Упрощение процесса сборки

Хотя использование Maven не избавляет от необходимости разбираться в базовых механизмах, Maven избавляет разработчиков от многих деталей.

### Библиотеки которые будут использоваться

Первый и пожалуй один из самых важных модулей которые я добавил в свой проект Spring-Boot Web. Spring Boot Web - это модуль Spring Boot, который предоставляет инструменты для создания веб-приложений. Он включает в себя поддержку Spring MVC, Spring Security и других технологий, необходимых для создания современных веб-приложений. Spring Boot Web позволяет быстро начать разработку веб-приложения, так как предоставляет готовые конфигурации и автоматическую настройку зависимостей.

Для работы с базой данных в Spring-Boot есть модуль Spring-Boot DATA JPA (Java Persistence API). Это модуль, который упрощает работу с реляционными базами данных. н позволяет автоматически генерировать и обновлять сущности Java на основе структуры базы данных, а также обеспечивает автоматическое управление транзакциями и кэширование данных.

Hibernate — библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (ORM), самая популярная реализация спецификации JPA. Распространяется свободно на условиях GNU Lesser General Public License. Позволяет сократить объёмы низкоуровневого программирования при работе с реляционными базами данных; может использоваться как в процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой. Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода. Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос (портирование) приложения на любые базы данных SQL. Hibernate обеспечивает прозрачную поддержку сохранности данных (persistence) для «POJO» (то есть для стандартных Java-объектов); единственное строгое требование для сохраняемого класса — наличие конструктора по умолчанию (без параметров). Для корректного поведения в некоторых приложениях требуется также уделить внимание методам equals() и hashCode().

Spring Boot Security - это модуль Spring Security, который предназначен для обеспечения безопасности веб-приложений, созданных с использованием Spring Boot. Он предоставляет механизмы для защиты веб-приложений от различных видов атак, таких как XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) и SQL-инъекции. Spring Boot Security также поддерживает различные способы аутентификации, такие как базовая аутентификация, аутентификация с использованием токенов и аутентификация с применением LDAP или Active Directory.

Spring Boot Validation - это механизм валидации данных, который позволяет проверять правильность ввода пользователя. Он поддерживает различные типы валидации, такие как проверка длины строки, проверка на соответствие регулярному выражению и проверка на уникальность. Spring Boot Validation также позволяет создавать свои собственные правила валидации и обрабатывать ошибки валидации.

Для удобства написания кода я буду использовать библиотеку Lombok. Lombok - это библиотека, которая позволяет сократить объем кода, необходимого для создания объектов и обработки исключений в Java. Она предоставляет удобные аннотации, которые позволяют автоматически генерировать методы getters и setters, конструкторы, equals() и hashCode() и т.д. Использование Lombok может значительно упростить разработку и уменьшить количество ошибок.

Для отображения пользовательского интерфейса мной выбрана библиотека Thymeleaf, т.к он имеет простой и понятный синтаксис, широкий выбор технологий (JavaScript, CSS, XML и другие), так же он отлично интегрируется с Spring-Boot. Thymleaf - это шаблонизатор, который позволяет создавать HTML-страницы на основе данных, полученных из Java-объектов. Он позволяет использовать различные теги и атрибуты для оформления HTML-кода, а также поддерживает обработку выражений и условных конструкций. Thymeleaf используется для создания динамических HTML-страниц.

HTML (HyperText Markup Language) - это стандартный язык разметки документов для Всемирной паутины. Он используется для создания структуры веб-страницы, определения содержания и оформления текста, ссылок, изображений и других элементов. HTML является основой для всех веб-страниц и необходим для отображения веб-контента в браузере.

### Контейнеризация с помощью Docker

Контейнеризация - это процесс упаковки приложения и его зависимостей в контейнер, такой как Docker, для легкого развертывания и масштабирования в облаке или на локальной машине. Контейнеры позволяют изолировать приложение от операционной системы и других приложений, обеспечивая тем самым гибкость и надежность в разработке и развертывании приложений.

Docker - это платформа контейнеризации, которая позволяет создавать, развертывать и управлять контейнерами в облаке или на локальном компьютере. Docker предоставляет инструменты для сборки образов контейнеров, управления ими и доставки их в облачные сервисы, такие как Amazon EC2, Google Compute Engine и Microsoft Azure. С помощью Docker можно быстро создавать, масштабировать и обновлять приложения без необходимости изменять операционную систему или другие приложения.

### Работа с контролем версий с помощью Git

Git — это распределённая система управления версиями, которая позволяет работать совместно над проектом нескольким пользователям. С её помощью можно отслеживать изменения в проектах, а также быстро возвращаться к предыдущим версиям проекта.

Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой. Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и, наряду с SSH, является наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

В своей работе я буду использовать [GitHub](https://github.com/). GitHub это хостинг git репозиториев. С помощью которого пользователи могут работать над одним проектом, видеть изменения проекта его ветки и оставлять комментарии.

# Разработка приложения

### Spring initializr

Для создания проекта у Spring есть несколько возможностей. Можно посетить сайт [https://start.spring.io](https://start.spring.io/) (Рисунок 1) и там собрать свой проект, либо создать Maven проект в IDE и там добавить нужные зависимости в pom.xml файл. Я воспользуюсь первым способом.

Переходим на сайт [Spring initializr](https://start.spring.io/). Для начала нужно выбрать сборщик проектов, я выбираю Maven (Рисунок 2).

Далее нужно выбрать язык программирования.

Версию Spring-Boot, я выбрал версию 3.2.4 т.к она является самой новой стабильной версией (Рисунки 3 и 4).

В следующей форме (Рисунок 5) нужно заполнить meta-данные о проекте:

* Group – идентификатор группы, команды которая занимается разработкой проекта
* Artifact — это идентификатор артефакта. Артефактами Maven называет приложения, пакеты и файлы. Как и идентификатор группы, он нужен, чтобы не запутаться в проекте. По умолчанию идентификатор артефакта соответствует имени проекта, которое мы указали при его создании
* Name – имя проекта
* Description – описание проекта
* Package name – название папки в которой будет собран проект

Далее выбираем в каком формате архивировать проект .jar или .war и версию Java (Рисунок 6).

Основное отличие .jar и .war файлов заключается в их предназначении: .jar файлы используются для распространения Java приложений или библиотек, в то время как .war файлы используются для распространения веб-приложений на платформе Java EE.

Добавляю нужные зависимости (Рисунок 7)

### Pom.xml

POM (Project Object Model) - это фундамент работы Maven. Pom.xml - файл, содержащий информацию о проекте и сведения о конфигурации, используемые Maven для сборки проекта. Он содержит значения по умолчанию для большинства проектов. Например, каталог сборки - target, каталог исходников - src/main/java, каталог тестовых исходников - src/test/java и так далее. При выполнении поставленной задачи или цели Maven ищет POM в текущем каталоге. Он считывает POM, получает необходимую конфигурационную информацию, а затем выполняет задачу.

Некоторые из конфигураций, которые могут быть указаны в POM, - это зависимости проекта и плагины которые могут быть выполнены, профили сборки и так далее. Также можно указать и другую информацию, такую как версия проекта, описание, разработчиков, списки рассылки и т. д.

Конфигурации pom.xml (Рисунок 8)

Заивсимости добавленные в проект (Рисунок 9 и 10)

### Cоздание Git репозитория

Создавать Git репозиторий я буду с помощь коммандной строки термина прямо из IDE.

Команда для создания Git репозитория:

git init

После внесения изменений в проект в котором инициализирован гит, он помечает эти файлы как изенённые (modified), для того что бы гит проиндексировал эти файлы нужно ввести в терминал команду:

git add .

точка обозначает, что проиндексируется вся папка (из которой в терминале запущена команда). После того как файлы проиндексированы они попадут в следующий коммит и статус файлов поменяется на зафиксирован (committed). Команда ввода коммита:

git commit -m ‘Сообщение’

Флаг -m обозначает что коммиту будет присвоено сообщение указанное после влага в ковычках.

Далее того что, бы проект был доступен другим пользователям я воспользуюсь сервисом GitHub. Его нужно туда загрузить для этого существует команда

git push

Но пока что нам нечего туда передать в качестве аргумента пути, который скажет гиту куда загружать проект. Что бы получить путь нужно зайти на GitHub, создать там репозиторий и бырать формат пути. На выбор есть для формата HTTPS и SSH. Я использую HTTPS подключение. Копирую путь репозитория(рисунок GitHub и подкставляю его в качесте аргумента.

git push origin main https://github.com/Framuga93/e-commerce.git

### Развертывание контейнера MySQL в Docker.

Сначала устанавливаем Docker Desktop с официального [сайта](https://www.docker.com/products/docker-desktop/). После установки приложение доступно из командной строки. В командной строке нужно набрать команду:

docker run --name ecommerce -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=root -d mysql/mysql-server:8.0

Docker создаст контейнер с развернутой внутри базой данных MySQL

* --name – имя контейнера
* -p – порт на котором будет запущено приложение (база данных)
* -e используется для передачи переменных окружения
* -d – запуск приложения в фоновом режиме

Для того что бы создать базу данных есть много способов. Можно воспользоваться одним из редакторов баз данных таких как DBeaver, MySQL Workbrench и многих других, можно использовать командную строку терминала, но в моем проекте я отдам создание базы данных и таблиц Hebirnate.

### Entities

Entities в Spring Boot нужны для работы с данными. Они позволяют определять структуру и свойства данных, которые будут храниться в базе данных. С помощью entities можно создавать объекты, которые представляют собой сущности в базе данных, такие как пользователи, продукты и заказы. Entities используются в сочетании с репозиториями для выполнения операций CRUD (создание, чтение, обновление, удаление) над данными.

Мною было создано 4 сущности:

* User - содержит в себе информацию о пользователях
  + Логин
  + Пароль
  + Email
  + И роли авторизации
* Role – содержит в себе имя ролей авторизации
* Product – содержит информацию о продукте
  + Название
  + Цену
  + Количество
* Order – содержит в себе информацию о заказе
  + Дату создания
  + Пользователя
  + Продукт

Зависимость сущностей множно увидеть на ER-диаграмме (Таблица 1)

### Репозитории

Репозитории в Spring Boot используются для работы с данными, хранящимися в базе данных. Они предоставляют интерфейс для выполнения различных операций с данными, таких как создание, чтение, обновление и удаление. Репозитории также используются для управления жизненным циклом объектов, например, для автоматического сохранения изменений в базе данных после выполнения некоторых операций.

Для создания репозиториев использую библеотеку Spring Data JPA. Для этого нужно унаследовать класс интерфейс репозитория от JpaRepository. JpaRepository это специфичное для JPA расширение класса Repository. И в аргументах указать объект с которым будут проходить операции репозитория и тип его поля Id:

public interface OrderRepository extends JpaRepository<Order,Long> {}

После этого объекатам класса этого репозитория доступны CRUD операции прямо «из коробки». Такие как

* Метод S save(S entity) сохраняет данные в базу:
* Метод Optional<T> findById(ID id) извлекает сущность по id:
* Метод List<T> findAll() возвращает список всех сущностей, что полезно для справочников и других небольших таблиц:
* Метод long count() возвращает количество сущностей, то есть записей в таблице:
* Метод void deleteById(ID id) удаляет сущность (запись в базе) по id
* Метод void delete(T entity) удаляет переданную сущность из базы данных:

### Сервисы

Сервисы в Spring Boot приложении используются для разделения логики приложения на более мелкие и управляемые компоненты. Они предоставляют уровень абстракции между контроллерами и репозиториями, облегчая тем самым тестирование и изменение кода. Сервисы также могут содержать бизнес-логику, которая не связана непосредственно с получением или сохранением данных, например, проверку входных данных, управление состоянием системы или генерацию отчетов.

В приложение сервисы буду реализованы с помощью наследования от интерфейсов, это нужно для возможности замены реализации не затрагивая основной код. При использовании интерфейсов основной код ничего не знает об деталях реализации (слабая связанность). Соответственно, реализацию можно вынести в отдельный модуль (изолировать сложность). Если основной код ссылается непосредственно на класс, содержащий реализацию (сильная связанность), то сложность программы возрастает и сопровождение программы усложняется. Поэтому интерфейсы особенно нужны на границах подсистем.

Для сущности User интерфейс будет иметь функции:

* SaveUser – сохранение User в базу данных
* FindUserByEmail – поиск User по полю Email
* FindAllUser – для поиска всех User

Для сущностей Product и Order сервис будет состоять из функций CRUD:

* Get – возвращает из базы объект с таким же ID
* Create – создает новый объект в базе данных
* Update – обновляет данные об объекте
* Delete – удаляет объект
* List- возвращает список всех объектов из базы данных

Реализация сервисов представлена на Рисунках 14-16.

### Контроллеры

Контроллеры в Spring Boot приложении нужны для обработки HTTP-запросов и маршрутизации их к соответствующим методам-обработчикам. Контроллеры определяют, как приложение будет обрабатывать входящие запросы, какие методы будут вызываться при получении определенных запросов и какие ответы будут возвращаться клиентам. Контроллеры также могут использовать различные механизмы обработки ошибок и обеспечивать дополнительную логику для улучшения пользовательского опыта.

В проекте использованы две разные аннотации для контроллера:

@Controller - используется для определения классов в качестве контроллеров в Spring MVC. Это помогает Spring определить компоненты, которые будут обрабатывать входящие запросы. Их я использовал для отображения пользовательского интерфейса и взаимодействия с ними с помощью Thymleaf. Отображение интерфейса будет реализовано с помощью шаблонов HTML страниц. Для этого методы контроллера должны возвращать тип данных строку, а значением будет HTML страница. Чтобы передать данные из приложения в шаблон нужно использовать Thymleaf:

* В метод контроллера нужно передать экземпляр класса Model
* Воспользоваться функцией класса Model addAttribute и передать в качестве аргументов строку с названием и сам объект, который нужно передать в шаблон.
* В шаблоне использовать специальный синтаксис Thymleaf для обращения к атрибутам этой модели

Для того, чтобы получить данные из шаблона:

* В метод контроллера нужно передать экземпляр класса получаемого объекта с аннотацией @ModelAttribute

@RestController - объединяет в себе аннотации @Controller и @ResponseBody. Это означает, что она не только помечает класс как Spring MVC Controller, но и автоматически преобразует возвращаемые контроллером данные в формат JSON или XML. Служат для управления данными с помощью REST запросов

Шаблоны HTML страниц в Приложение “Шаблоны HTML”, Реализация на Рисунках 17 и 18

### ExceptionHandler

Модуль Spring framework MVC имеет отличные возможности для обработки ошибок. И разработчику остается использовать эти возможности для обработки исключений и возвращения содержательных ответов клиенту API.

ExceptionHandler используется для обработки исключений, которые могут возникнуть во время выполнения приложения. Он позволяет обрабатывать различные типы исключений и возвращать пользователю соответствующие сообщения об ошибках. Кроме того, ExceptionHandler может быть использован для логирования ошибок и уведомления разработчиков о проблемах в работе приложения.

Реализация Рисунок 19

### Spring Security Configuration

Конфигурация Spring Security настраивается через SecurityFilterChain. SecurityFilterChain используется FilterChainProxy для определения того, какие экземпляры Spring Security Filter должны быть вызваны для текущего запроса. Как это показано в Таблице 2.

Фильтры безопасности в SecurityFilterChain обычно представляют собой регистрируемые в FilterChainProxy, а не в DelegatingFilterProxy. FilterChainProxy предоставляет ряд преимуществ по сравнению с регистрацией непосредственно в контейнере сервлетов или DelegatingFilterProxy. Во-первых, он является отправной точкой для всей поддержки сервлетов в Spring Security. Во-вторых, поскольку FilterChainProxy занимает центральное место в использовании Spring Security, он может выполнять дополнительные задачи. Например, он очищает SecurityContext, чтобы избежать утечек памяти. Он также применяет HttpFirewall от Spring Security для защиты приложений от определенных типов атак.

Spring Security позволяет настраивать авторизацию на уровне запроса. Например, с помощью Spring Security вы можете настроить, что все страницы под /admin требуют определенных прав, а все остальные страницы просто требуют аутентификации. По умолчанию Spring Security требует, чтобы каждый запрос был аутентифицирован. Тем не менее, каждый раз, когда вы используете экземпляр HttpSecurity, необходимо объявлять свои настройки авторизации.

Сначала фильтр авторизации создает поставщика, который извлекает аутентификацию из SecurityContextHolder. Во-вторых, он передает Supplier<Authentication> и HttpServletRequest в AuthorizationManager. Менеджер авторизации сопоставляет запрос с шаблонами в authorizeHttpRequests и запускает соответствующее правило. Если в авторизации отказано, публикуется событие AuthorizationDeniedEvent и выбрасывается исключение AccessDeniedException. В этом случае фильтр ExceptionTranslationFilter обрабатывает исключение AccessDeniedException. Если доступ разрешен, публикуется событие AuthorizationGrantedEvent, и AuthorizationFilter продолжает работу с цепочкой фильтров, что позволяет приложению нормально работать.

Реализация Рисунок 20

### Custom UserDetailsService

Для того что бы хранить пользователей в базе данных, нужно создать класс CustomUserDetailService который реализует UserDetailsService с единственным методом loadUserByUsername этот метод принимает строку. По этой строке метод должен найти в базе данных пользователя и сформировать его в UserDetail. Я передаю в качестве строки Email пользователя.

UserDetail - реализация не используются непосредственно для обеспечения безопасности. Она хранит информацию о пользователе, которая позже инкапсулируется в объекты аутентификации. Это позволяет хранить в удобном месте информацию о пользователе, не связанную с безопасностью

Реализация Рисунок 21

### Password Encoder

Интерфейс PasswordEncoder в Spring Security используется для одностороннего преобразования пароля с целью его безопасного хранения. Поскольку PasswordEncoder - это одностороннее преобразование, его не стоит использовать, когда преобразование пароля должно быть двусторонним (например, при хранении учетных данных, используемых для аутентификации в базе данных). Обычно PasswordEncoder используется для хранения пароля, который необходимо сравнить с паролем, предоставленным пользователем при аутентификации.

Реализация BCryptPasswordEncoder использует широко поддерживаемый алгоритм bcrypt для хэширования паролей.

### Заключение

Разработка интернет-магазина благоприятно скажется на работе практически любой компании. Частично или полностью заменив офлайн-продажи сайтом, предприниматель имеет экономически обоснованные преимущества. Например, такие, как снижение расходов на аренду помещений и наемных работников, возможность получать статистические данные (пол, возраст покупателей, пиковые часы и дни продаж, наиболее популярные товары), а также возможность большего географического охвата потенциальных покупателей.

На основании проделанной работы, можно сделать вывод, что преимущества языка Java делают его отличным выбором для разработки интернет-магазинов, обеспечивая надежность, масштабируемость, безопасность и гибкость в разработке приложений для электронной торговли.

Также следует заметить, что при выборе технологий и инструментов для разработки интернет-магазина следует опираться на актуальность и поддерживаемость. А также большим преимуществом той или иной технологии является ее широкая применяемость и популярность в IT-сообществе, так как это позволит решить множество задач быстрее и эффективнее, опираясь на существующий опыт.

В ходе практической части работы, я столкнулся с определенными сложностями при использовании шаблонизатора Thymleaf. Он менее гибкий в функционале и не имеет большого комьюнити. Такими требованиями обладает, например, Java Script, что несомненно является преимуществом.

Следует добавить, что немаловажным является использование новейших технологий для разработки высокоэффективного интернет-магазина.

В ходе выполнения данной цели стало возможным решить следующие задачи:

1. Проведен анализ рынка интернет-магазинов и определены требований к будущему сервису.
2. Описаны выбранные технологий для разработки интернет-магазина.
3. Спроектирована архитектура сервиса и базы данных.
4. Создан пользовательский интерфейс с помощью Thymleaf.
5. Реализован функционала для обработки заказов, включая управление товарами и обеспечение безопасности данных пользователей.

Таким образом задачи решены в полном объеме, цель достигнута - разработан интернет-магазин на основе Spring Boot с использованием современных технологий и подходов. Сервис работает и имеет перспективы для развития и совершенствования.

### Используемые технологии

* Spring Boot
* Spring Data JPA
* Spring Security
* Hibernate
* Maven
* Lombok
* Thymleaf
* HTML

### Состав команды

Дёмочкин Алексей Владимирович (Java-разработчик)

# Список используемой литературы

* *Книга: Лауренциу Спилкэ. "Spring быстро". Город: Питер, 2023.*
* *Статья: Автор. "Заголовок статьи." Название журнала Том, номер (Год): страницы.*
* *Статистические данные по интернет-магазинам// Электронная версия на сайте* [*https://top100.datainsight.ru*](https://top100.datainsight.ru)
* *Общие сведения о языке программирования Java // Электронная версия на сайте https://ru.wikipedia.org/wiki/Java*
* *Робин Никсон // создаём динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5, 2016 г. - 768с.*
* [*https://habr.com/ru/articles/*](https://habr.com/ru/articles/)
* *Официальная документация* [*https://docs.spring.io/spring-framework/reference/index.html*](https://docs.spring.io/spring-framework/reference/index.html)
* *Официальная документация* [*https://dev.mysql.com/doc/*](https://dev.mysql.com/doc/)
* *Официальная документация* [*https://docs.oracle.com/en/java/*](https://docs.oracle.com/en/java/)
* *Официальная документация https://docs.docker.com*

Приложения

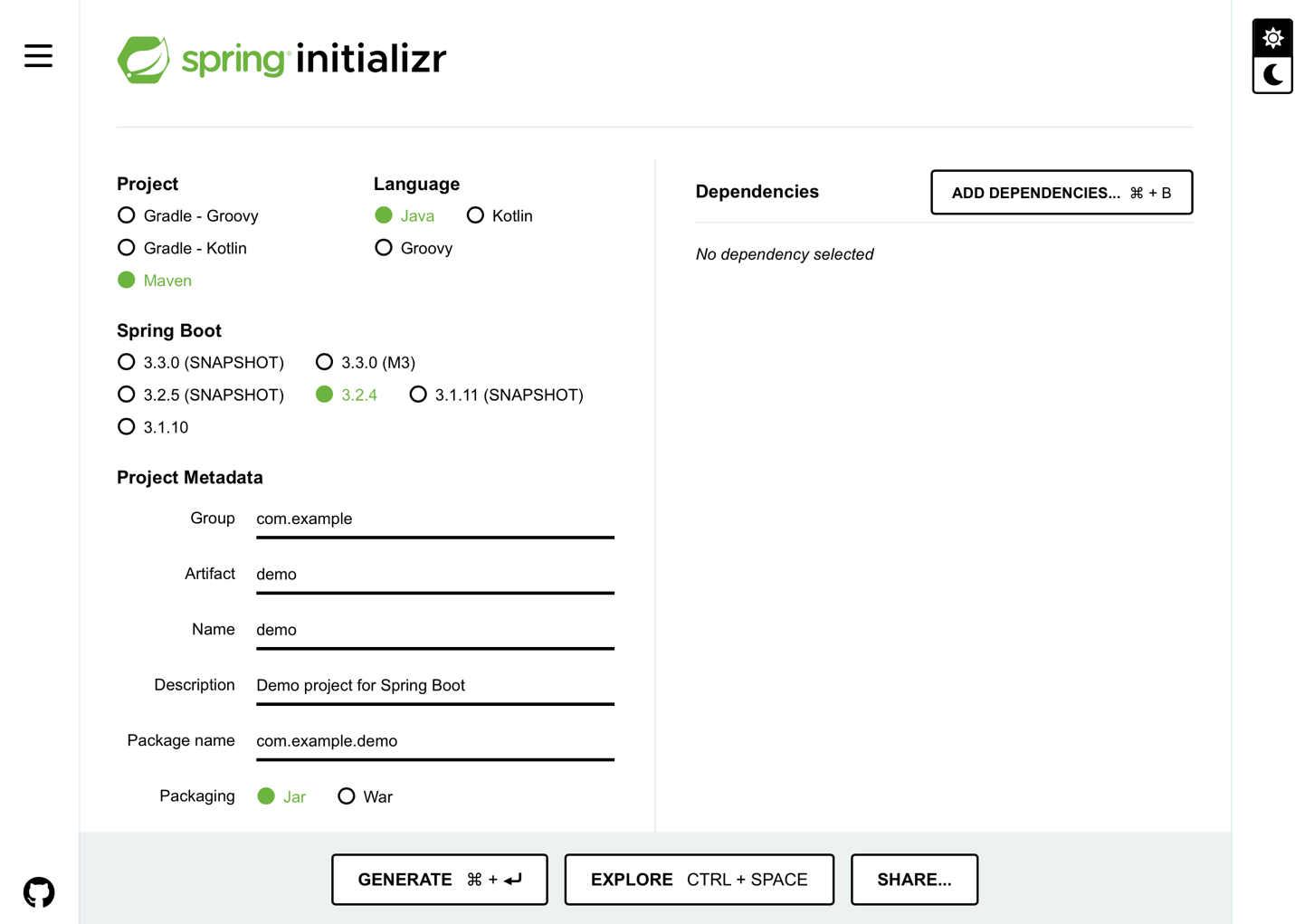


Рисунок 1 Cайт Spring initializr

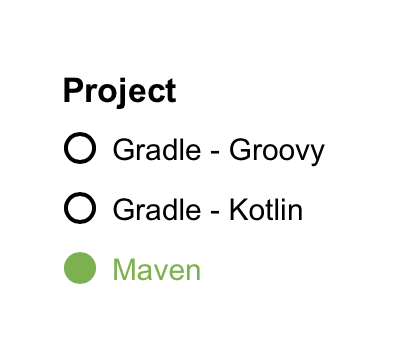


Рисунок 2 Выбор сборщика проекта

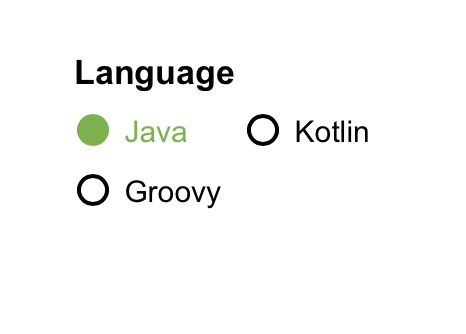


Рисунок 3 Выбор языка программирования

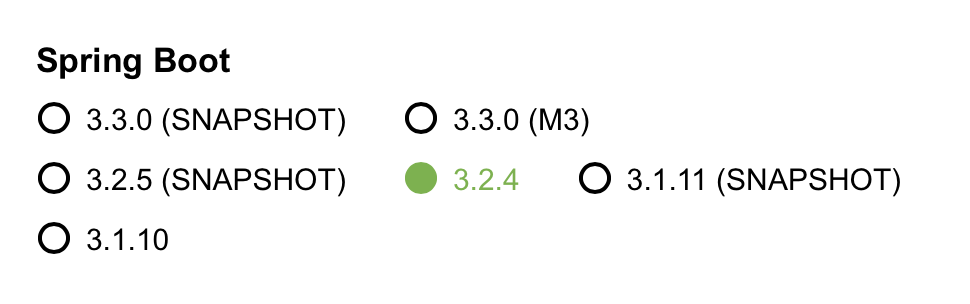


Рисунок 4 Выбор версии Spring-Boot

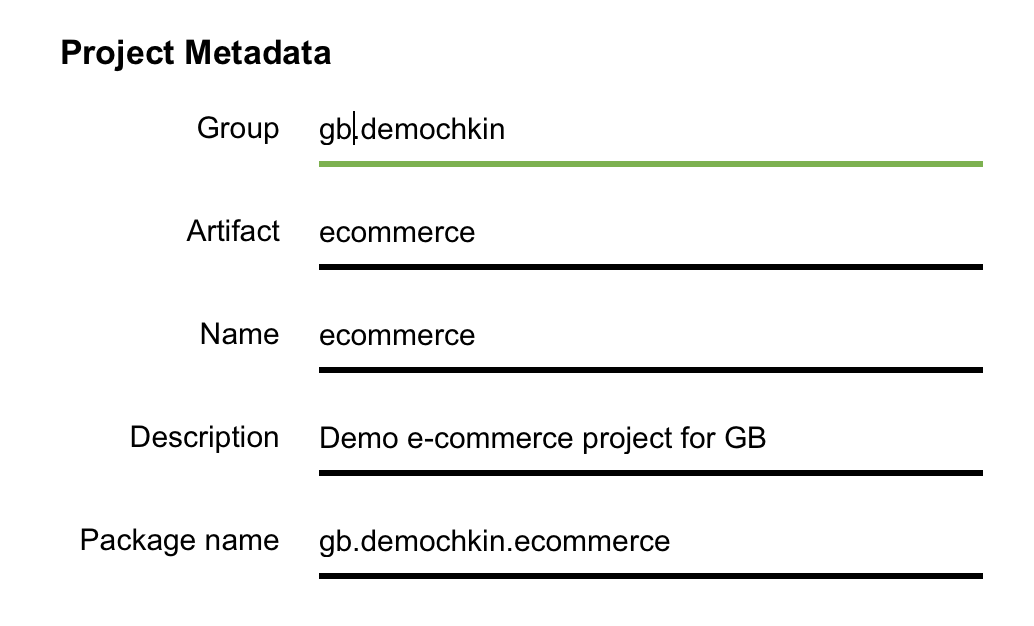


Рисунок 5 Форма meta данных проекта



Рисунок 6 Выбор формата архива и версию Java

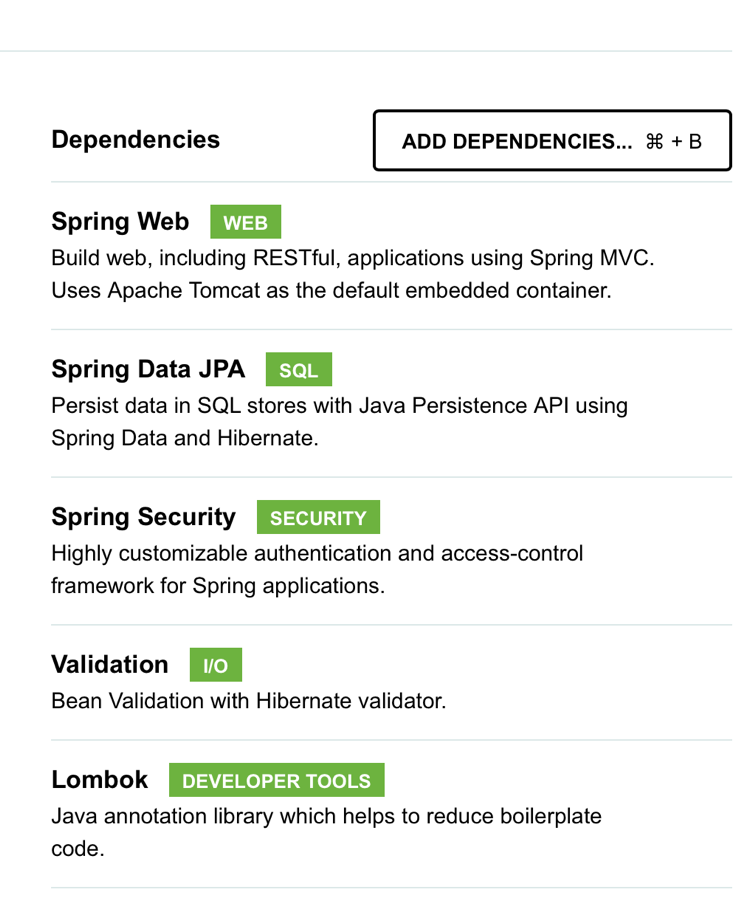


Рисунок 7 Выбор зависимостей

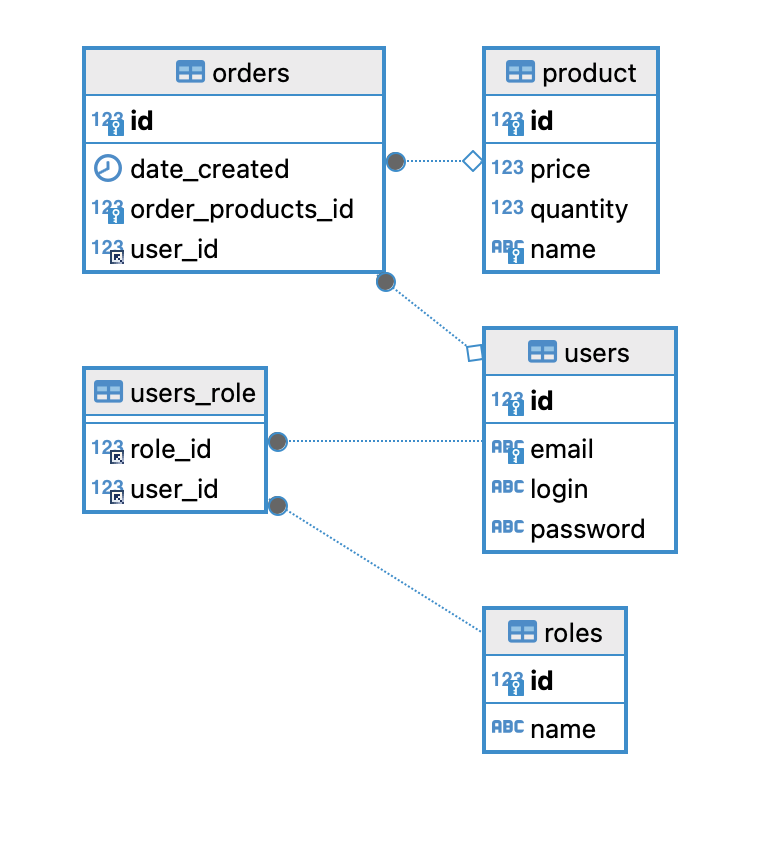


Таблица 1

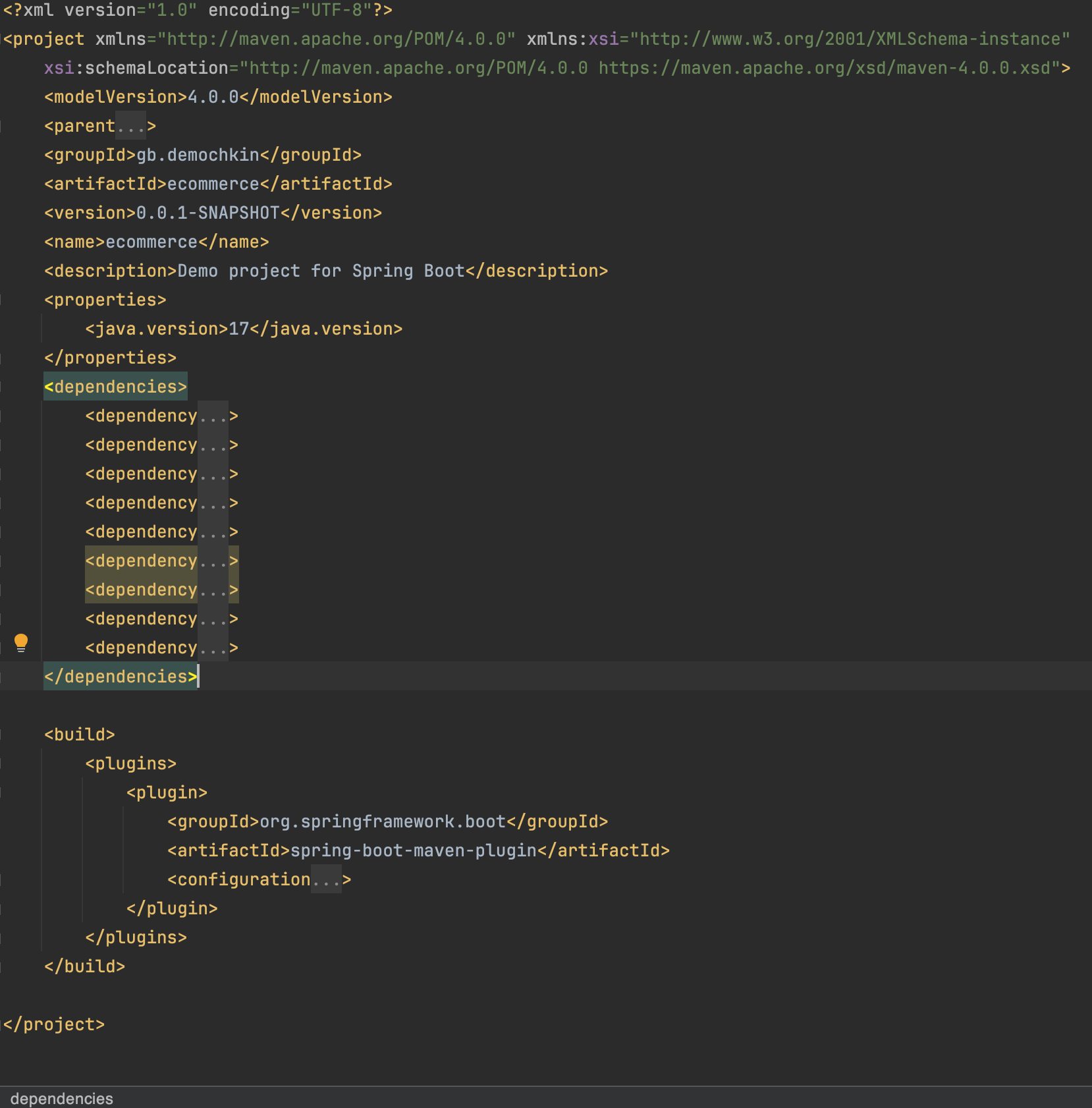


Рисунок 8 Конфигурация pom.xml

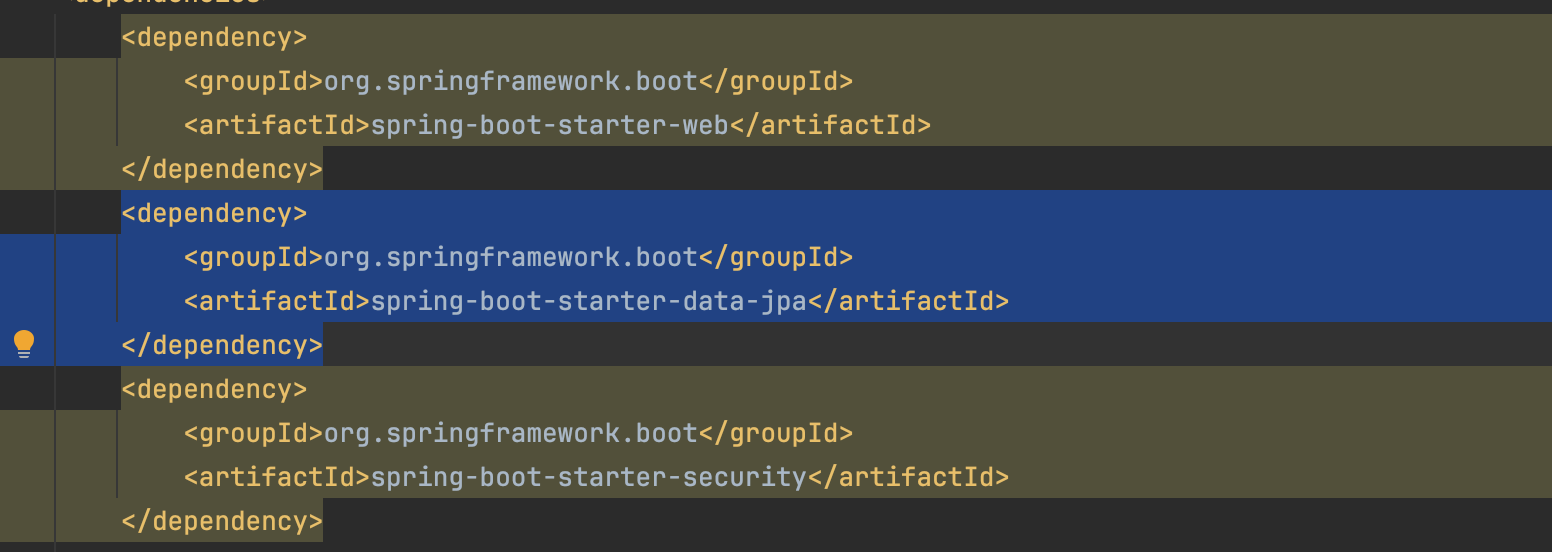


Рисунок 9 Зависимости



Рисунок 10 Зависимости

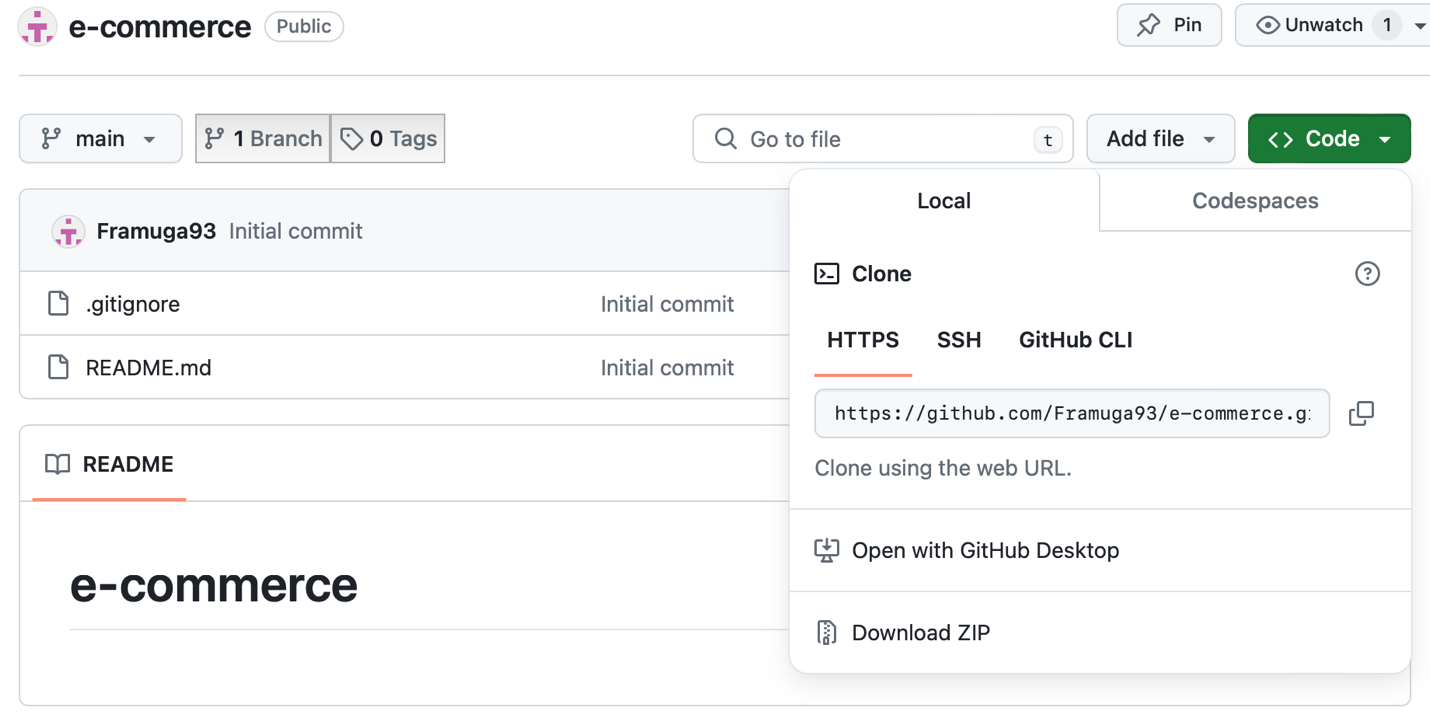


Рисунок 11 Копирование пути GitHub

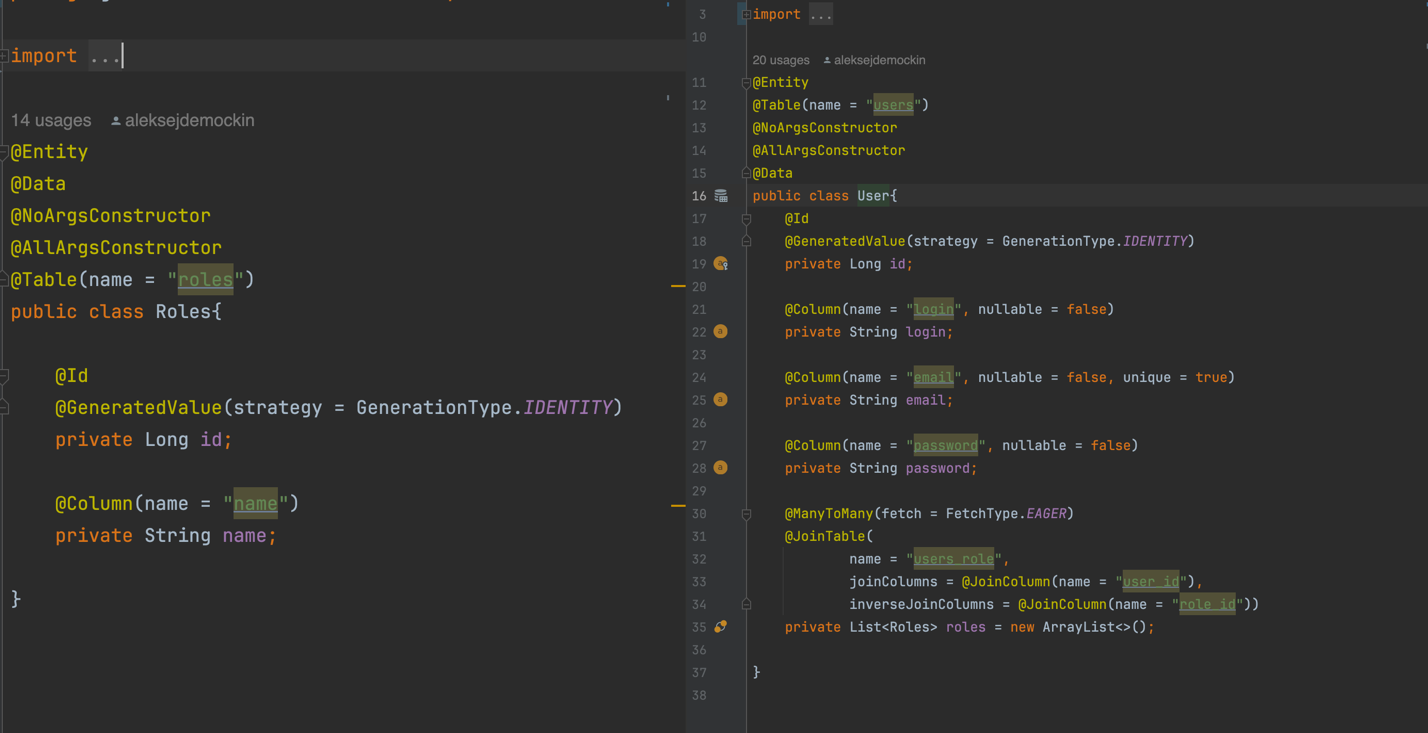


Рисунок 12 Roles и Users

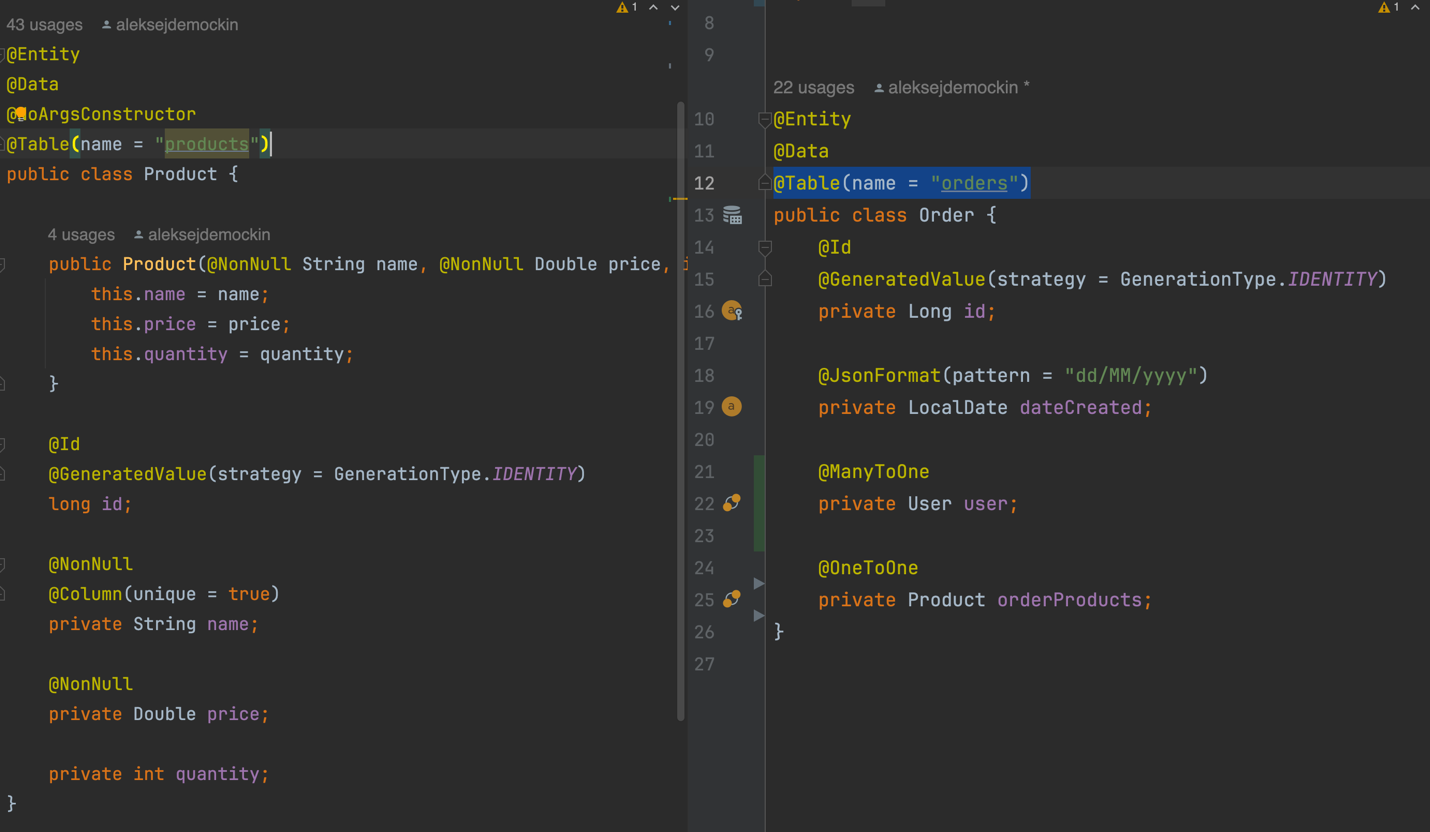


Рисунок 13 Products и Orders

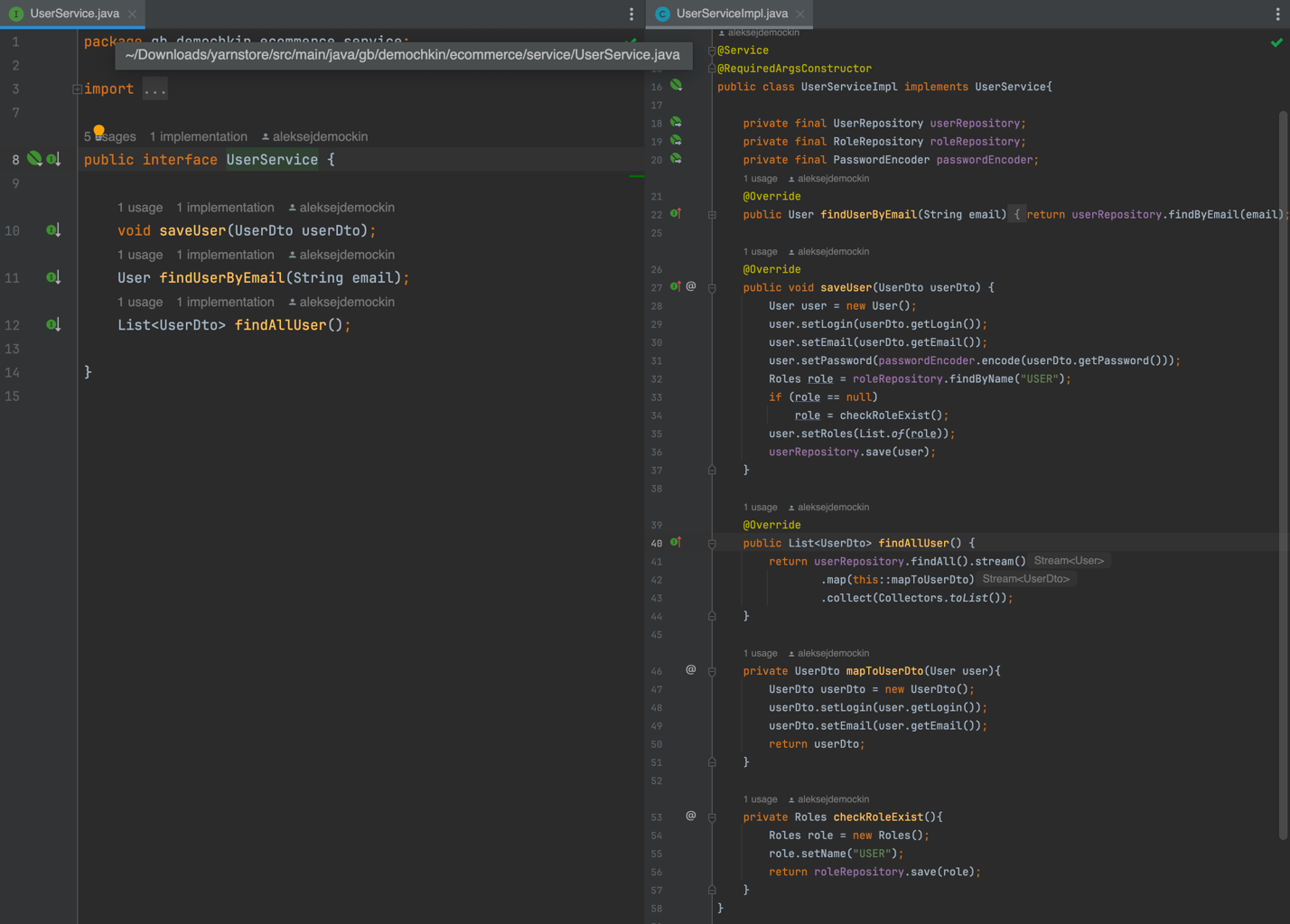


Рисунок 14 User Service

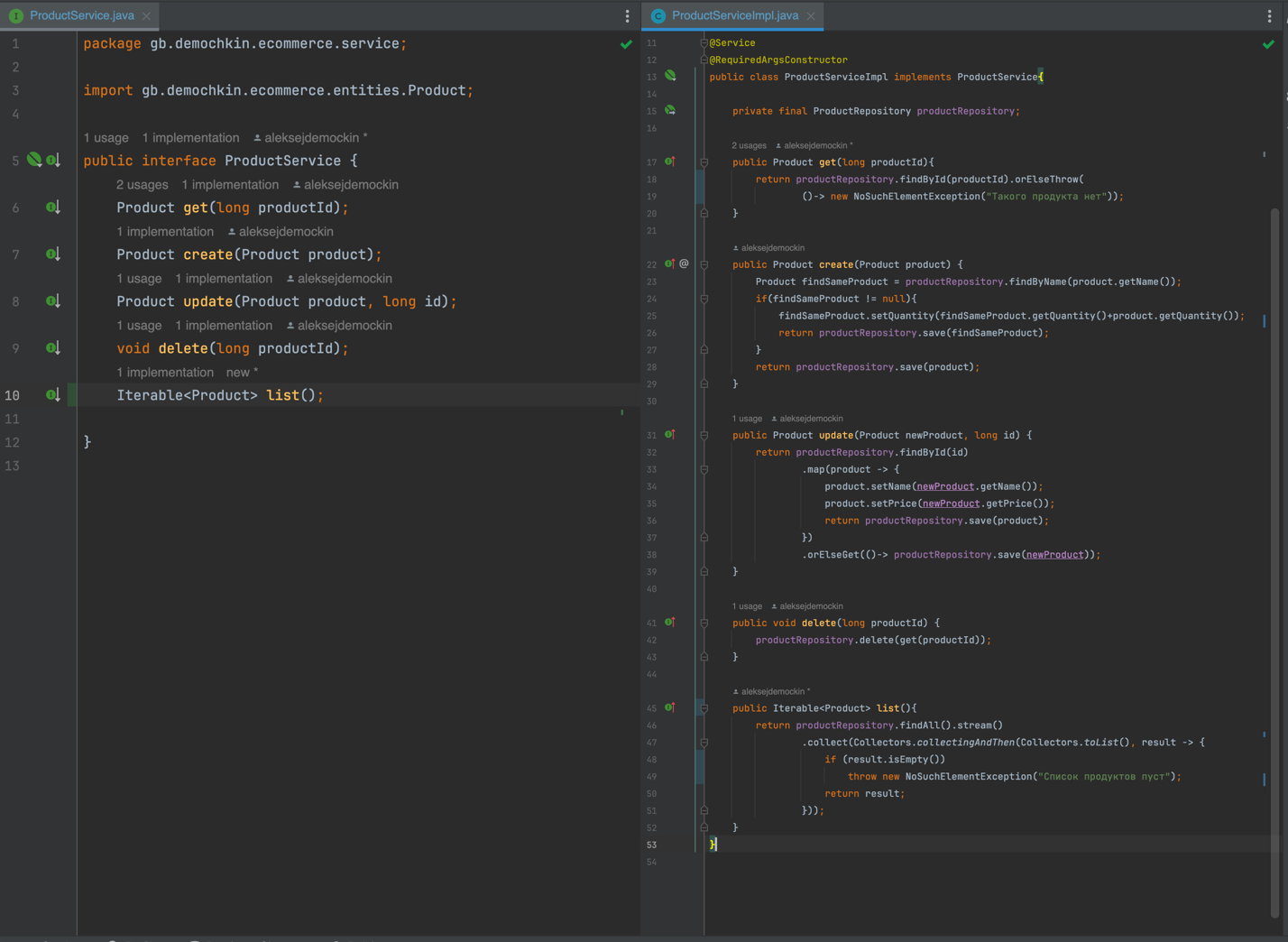


Рисунок 15 Product Service



Рисунок 16 Order Service

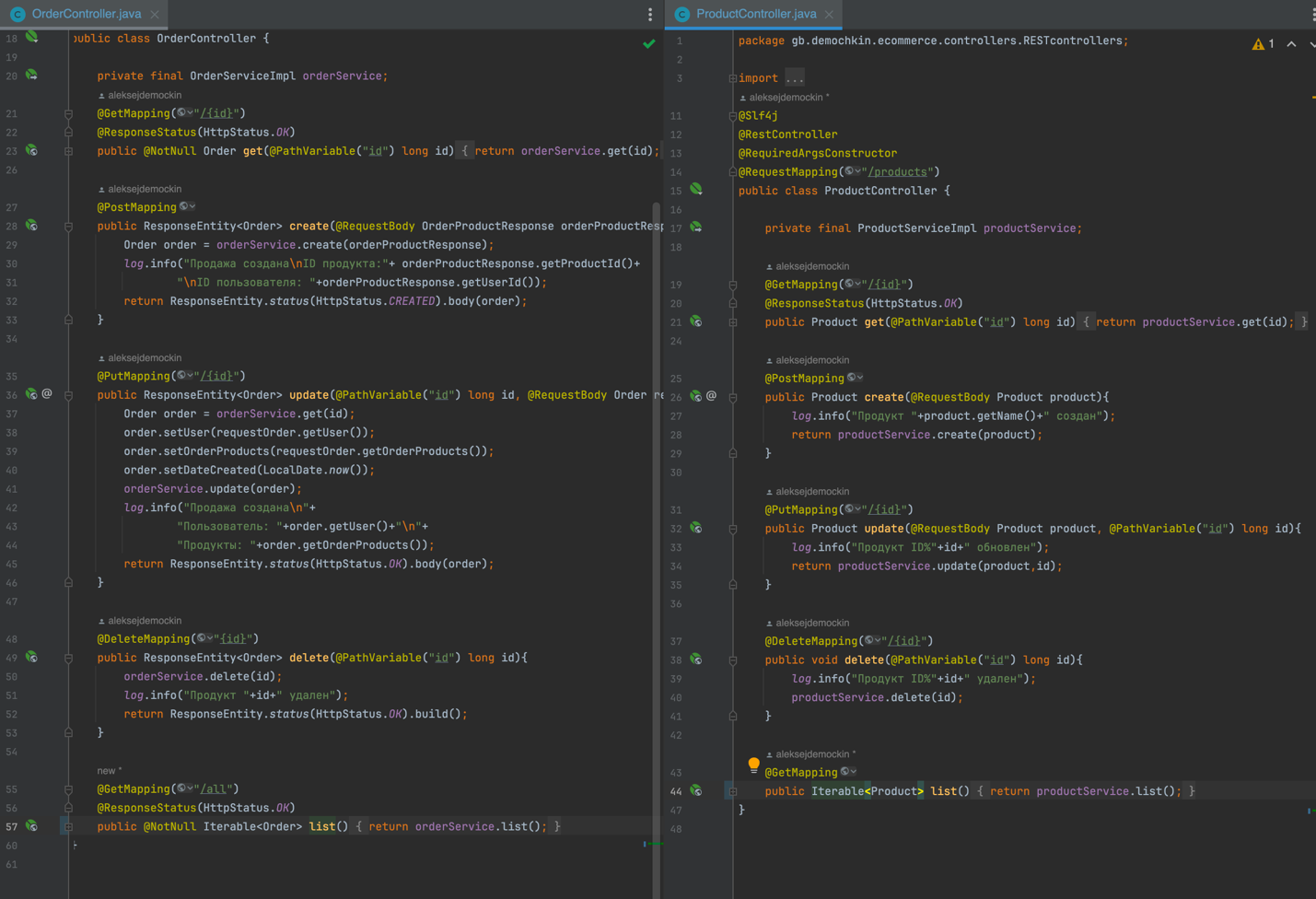


Рисунок 17 Rest Controllers

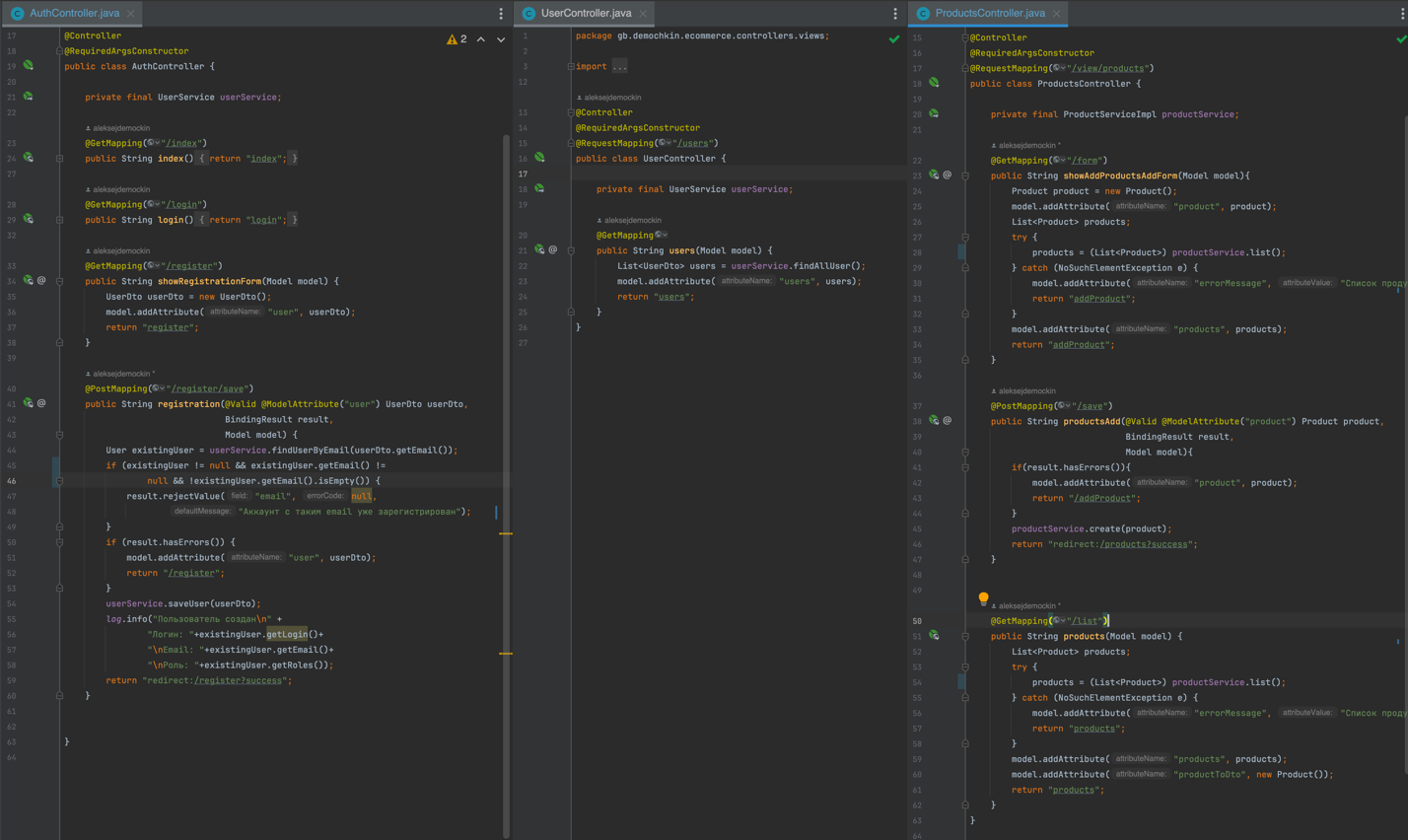


Рисунок 18 Controllers

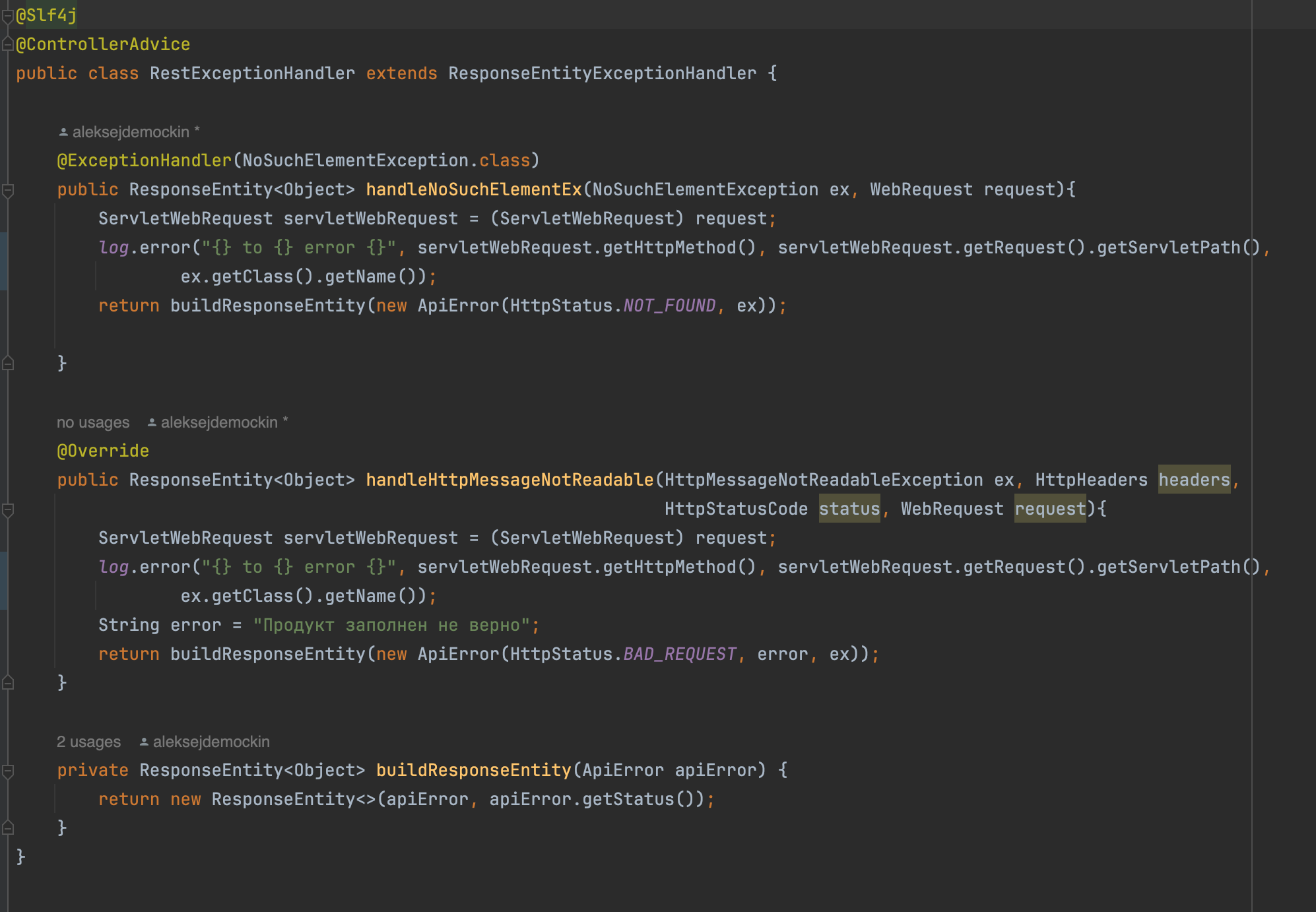


Рисунок 19 Exception Handler

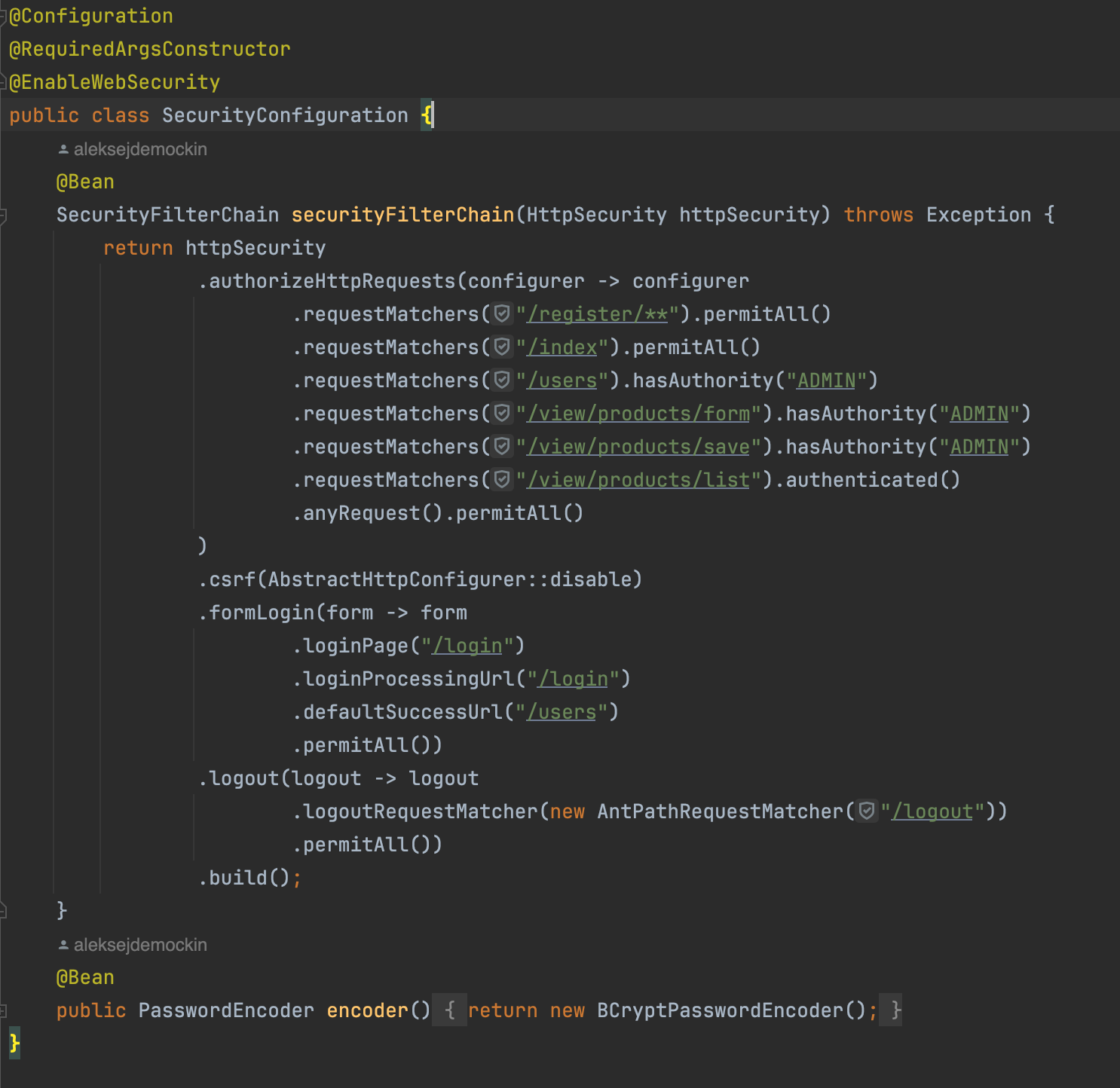


Рисунок 20 Security Configuration

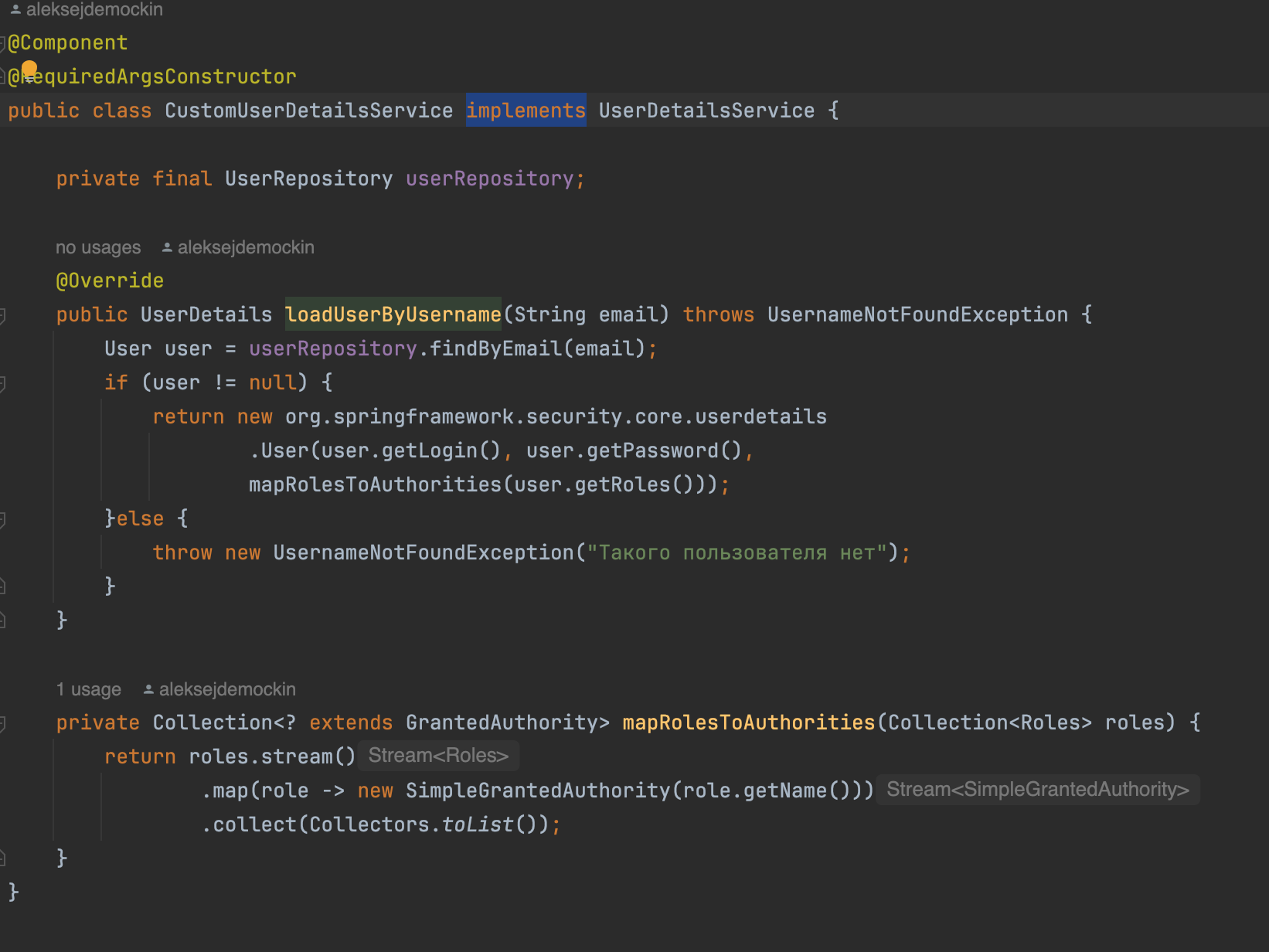


Рисунок 21 CustomUserDetailServiceщ>ко

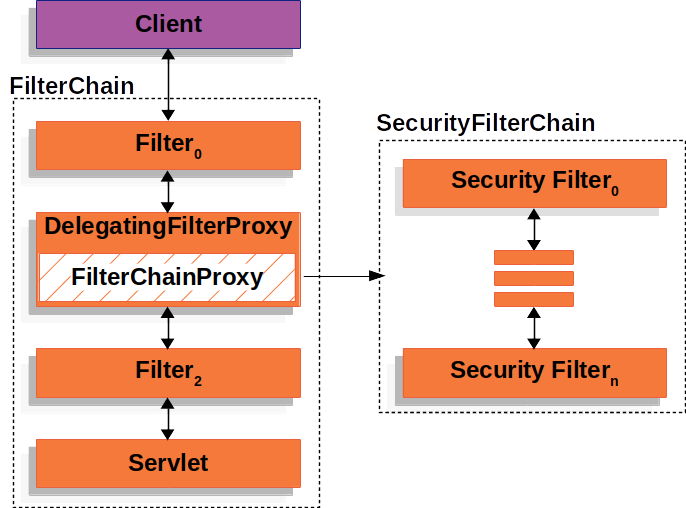


Таблица 2

**Шаблоны HTML страниц:**

Addproduct.html

<!DOCTYPE html>  
<html lang="en"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns="http://www.w3.org/1999/html">  
<head>  
 <meta charset="UTF-8">  
 <title>Registered users</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"  
 rel="stylesheet"  
 integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"  
 crossorigin="anonymous">  
</head>  
<body>  
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">  
 <div class="container-fluid">  
 <a class="navbar-brand" th:href="@{/index}">E-commerce shop demo</a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/logout}">Logout</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</nav>  
<div class="container">  
 <div class="row col-md-10">  
 <h2>Добавление товара</h2>  
 </div>  
 <div class="card-body">  
 <form  
 method="post"  
 role="form"  
 th:action="@{/products/save}"  
 th:object="${product}"  
 >  
 <div class="form-group mb-3">  
 <label class="form-label">Name</label>  
 <input  
 class="form-control"  
 id="name"  
 name="name"  
 placeholder="Введите название товара"  
 th:field="\*{name}"  
 type="text"  
 />  
 <p th:errors = "\*{name}" class="text-danger"  
 th:if="${#fields.hasErrors('Name')}"></p>  
 </div>  
  
 <div class="form-group mb-3">  
 <label class="form-label">Price</label>  
 <input  
 class="form-control"  
 id="price"  
 name="price"  
 placeholder="Введите цену товара"  
 th:field="\*{price}"  
 type="number"  
 />  
 <p th:errors = "\*{price}" class="text-danger"  
 th:if="${#fields.hasErrors('Price')}"></p>  
 </div>  
  
 <div class="form-group mb-3">  
 <label class="form-label">Quantity</label>  
 <input  
 class="form-control"  
 id="quantity"  
 name="quantity"  
 placeholder="Введите количество товара"  
 th:field="\*{quantity}"  
 type="number"  
 />  
 <p th:errors = "\*{quantity}" class="text-danger"  
 th:if="${#fields.hasErrors('Quantity')}"></p>  
 </div>  
 <br/>  
 <div class="form-group">  
 <button class="btn btn-primary" type="submit">Добавить</button>  
 </div>  
 <br/>  
 </form>  
 </div>  
 <table class="table table-bordered table-hover">  
 <thead class="table-dark">  
 <tr>  
 <th>Товар</th>  
 <th>Цена</th>  
 <th>Количество</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each = "pr : ${products}">  
 <td th:text = "${pr.name}"></td>  
 <td th:text = "${pr.price}"></td>  
 <td th:text = "${pr.quantity}"></td>  
 </tr>  
 </tbody>  
 </table>  
</div>  
</body>  
</html>

Index.html

<!DOCTYPE html>  
<html lang="en"  
 xmlns:th="http://www.thymeleaf.org"  
>  
<head>  
 <meta charset="UTF-8">  
 <title>E-commerce store</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"  
 rel="stylesheet"  
 integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"  
 crossorigin="anonymous">  
</head>  
<body>  
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">  
 <div class="container-fluid">  
 <a class="navbar-brand" th:href="@{/index}">E-commerce shop demo</a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/register}">Регистрация</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</nav>  
<br /><br />  
<div class="container">  
 <div class="row">  
 <h1 class="text-center"> </h1>  
 </div>  
</div>  
</body>  
</html>

Products.html

<!DOCTYPE html>  
<html lang="en"  
 xmlns:th="http://www.thymeleaf.org"  
>  
<head>  
 <meta charset="UTF-8">  
 <title>Registered users</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"  
 rel="stylesheet"  
 integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"  
 crossorigin="anonymous">  
</head>  
<body>  
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">  
 <div class="container-fluid">  
 <a class="navbar-brand" th:href="@{/index}">E-commerce shop demo</a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"  
 aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/logout}">Logout</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</nav>  
<div class="container">  
 <div class="row col-md-10">  
 <h2>Список продуктов</h2>  
 </div>  
 <table class="table table-bordered table-hover">  
 <thead class="table-dark">  
 <tr>  
 <th>Продукт</th>  
 <th>Цена</th>  
 <th>Доступно на складе</th>  
 <th>Количество</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="product : ${products}">  
 <td th:text="${product.name}"></td>  
 <td th:text="${product.price}"></td>  
 <td th:text="${product.quantity}"></td>  
 </tr>  
 </tbody>  
 </table>  
 <div th:if="${errorMessage}">  
 <div th:text="${errorMessage}"></div>  
 </div>  
</div>  
</body>  
</html>

Login.html

<!DOCTYPE html>  
<html lang="en"  
 xmlns:th="http://www.thymeleaf.org"  
>  
<head>  
 <meta charset="UTF-8">  
 <title>Login</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"  
 rel="stylesheet"  
 integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"  
 crossorigin="anonymous">  
</head>  
<body>  
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">  
 <div class="container-fluid">  
 <a class="navbar-brand" th:href="@{/index}">E-commerce shop demo</a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/register}">Регистрация</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</nav>  
<br /><br />  
<div class="container">  
 <div class="row">  
 <div class="col-md-6 offset-md-3">  
  
 <div th:if="${param.error}">  
 <div class="alert alert-danger">Неверный Email или Password</div>  
 </div>  
 <div th:if="${param.logout}">  
 <div class="alert alert-success"> Вы вышли из аккаунта.</div>  
 </div>  
  
 <div class="card">  
 <div class="card-header">  
 <h2 class="text-center">Авторизация</h2>  
 </div>  
 <div class="card-body">  
 <form  
 method="post"  
 role="form"  
 th:action="@{/login}"  
 class="form-horizontal"  
 >  
 <div class="form-group mb-3">  
 <label class="control-label"> Email</label>  
 <input  
 type="text"  
 id="username"  
 name="username"  
 class="form-control"  
 placeholder="Введите email"  
 />  
 </div>  
  
 <div class="form-group mb-3">  
 <label class="control-label"> Пароль</label>  
 <input  
 type="password"  
 id="password"  
 name="password"  
 class="form-control"  
 placeholder="Введите пароль"  
 />  
 </div>  
 <div class="form-group mb-3">  
 <button type="submit" class="btn btn-primary" >Submit</button>  
 <br/>  
 <span> Еще не зарегистрированы?  
 <br/>  
 <a th:href="@{/register}">Регистрация</a>  
 </span>  
 </div>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
</body>  
</html>

Register.html

<!DOCTYPE html>  
<html lang="en"  
 xmlns:th="http://www.thymeleaf.org"  
>  
<head>  
 <meta charset="UTF-8">  
 <title>Registration form</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"  
 rel="stylesheet"  
 integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"  
 crossorigin="anonymous">  
</head>  
<body>  
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">  
 <div class="container-fluid">  
 <a class="navbar-brand" th:href="@{/index}">E-commerce shop demo</a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/login}">Login</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</nav>  
<br /><br /><br />  
<div class="container">  
 <div class="row col-md-8 offset-md-2">  
 <div class="card">  
 <div class="card-header">  
 <h2 class="text-center">Регистрация</h2>  
 </div>  
 <div th:if="${param.success}">  
 <div class="alert alert-info">  
 Вы успешно зарегистрированы!  
 </div>  
 </div>  
 <div class="card-body">  
 <form  
 method="post"  
 role="form"  
 th:action="@{/register/save}"  
 th:object="${user}"  
 >  
 <div class="form-group mb-3">  
 <label class="form-label">Login</label>  
 <input  
 class="form-control"  
 id="login"  
 name="login"  
 placeholder="Введите логин"  
 th:field="\*{login}"  
 type="text"  
 />  
 <p th:errors = "\*{login}" class="text-danger"  
 th:if="${#fields.hasErrors('Login')}"></p>  
 </div>  
  
 <div class="form-group mb-3">  
 <label class="form-label">Email</label>  
 <input  
 class="form-control"  
 id="email"  
 name="email"  
 placeholder="Введите email"  
 th:field="\*{email}"  
 type="email"  
 />  
 <p th:errors = "\*{email}" class="text-danger"  
 th:if="${#fields.hasErrors('email')}"></p>  
 </div>  
  
 <div class="form-group mb-3">  
 <label class="form-label">Password</label>  
 <input  
 class="form-control"  
 id="password"  
 name="password"  
 placeholder="Введите пароль"  
 th:field="\*{password}"  
 type="password"  
 />  
 <p th:errors = "\*{password}" class="text-danger"  
 th:if="${#fields.hasErrors('Password')}"></p>  
 </div>  
 <div class="form-group">  
 <button class="btn btn-primary" type="submit">Зарегистрироваться</button>  
 <br/>  
 <br/>  
 <span>Уже зарегистрированы? <a th:href="@{/login}">Авторизуйтесь здесь</a></span>  
 </div>  
 </form>  
 </div>  
 </div>  
 </div>  
</div>  
</body>  
</html>

Users.html

<!DOCTYPE html>  
<html lang="en"  
 xmlns:th="http://www.thymeleaf.org"  
>  
<head>  
 <meta charset="UTF-8">  
 <title>Registered users</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"  
 rel="stylesheet"  
 integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"  
 crossorigin="anonymous">  
</head>  
<body>  
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">  
 <div class="container-fluid">  
 <a class="navbar-brand" th:href="@{/index}">E-commerce shop demo</a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/logout}">Logout</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</nav>  
<div class="container">  
 <div class="row col-md-10">  
 <h2>Список зарегистрированных пользователей</h2>  
 </div>  
 <table class="table table-bordered table-hover">  
 <thead class="table-dark">  
 <tr>  
 <th>Логин</th>  
 <th>Email</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each = "user : ${users}">  
 <td th:text = "${user.login}"></td>  
 <td th:text = "${user.email}"></td>  
 </tr>  
 </tbody>  
 </table>  
</div>  
</body>  
</html>