Geekbrains

**Разработка интернет-магазина с использованием Java Spring, Docker**

Программа: Разработчик

Специализация: Веб-разработка на Java

ФИО: Истомин Сергей Андреевич

Город Новосибирск

2024 Год

Оглавление

[Введение 3](#_Toc171856866)

[Теоретическая и практическая главы. 6](#_Toc171856867)

[1. Получение технического задания от заказчика. 6](#_Toc171856868)

[2. Выбор технологий. 7](#_Toc171856869)

[3. Создание проекта. 7](#_Toc171856870)

[3.1 Системы сборки Maven и Gradle для разработки Java приложений. 8](#_Toc171856871)

[3.2 Сравнение Maven и Gradle, что будет использовано в проекте. 10](#_Toc171856872)

[3.3 Краткая теория Maven. 11](#_Toc171856873)

[3.4 Установка Maven. 17](#_Toc171856874)

[3.5 Генерация проекта. 18](#_Toc171856875)

[4. Spring Boot. 19](#_Toc171856876)

[4.1 Основы Spring. 20](#_Toc171856877)

[4.2 Разница между Spring и Spring Boot. 22](#_Toc171856878)

[4.3 Типы бинов. 23](#_Toc171856879)

[4.4 Spring Framework две основные реализации контейнера. 24](#_Toc171856880)

[4.5 Конфигурация Spring Boot приложения. 25](#_Toc171856881)

[4.6 Клиент-сервер. 26](#_Toc171856882)

[4.7 Виды HTTP-запросов и коды ответов. 27](#_Toc171856883)

[4.8 Контроллеры в Spring. 28](#_Toc171856884)

[5. Spring MVC. 30](#_Toc171856885)

[6. Подключения базы данных в проекте. 31](#_Toc171856886)

[7. API. 35](#_Toc171856887)

[8. Spring Security. 37](#_Toc171856888)

[9. Spring AOP. 40](#_Toc171856889)

[10. Реализация проекта. 51](#_Toc171856890)

[Заключение 59](#_Toc171856891)

[Список используемой литературы 61](#_Toc171856892)

[Приложения 62](#_Toc171856893)

# Введение

**Что из себя представляет проект.**

Проект на разработку web-сайта интернет-магазина (ЗОО магазин). Для продвижения своего бизнеса, привлечения внимания и увеличения лояльности потенциальных и действующих клиентов, создание позитивного образа малого предприятия, увеличения количества продаж с использованием web-сайта.

**Обоснование темы проекта.**

В наше время всеобщей компьютеризации тяжело представить компанию, способную выжить и выдержать конкуренцию без использования современных технологий.

Отказ от нововведений и применения современных технологий ведет к проигрышу конкурентной борьбы. А предприятия, использующие эти технологии, смогут повысить производительность, поток клиентов и соответственно денежную прибыль.

Бурный рост всемирной сети заставляет привлекать к себе внимание, т. к. из глобальной сети можно привлечь большое количество потенциальных клиентов. Для успешного конкурирования в сети Интернет, предприятию необходимо место, где оно могло бы представить информацию о себе — сайт или группа в социальной сети. Но все же компании предпочтительнее иметь собственный web-сайт, т. к. он является более «представительным».

Вне зависимости от размера предприятия, наличие сайта — это плюс для имиджа любой компании, для узнаваемости и для привлечения клиентов. Web-сайт при хорошо выстроенном бизнесе значительно увеличивает прибыль любой компании — напрямую или опосредованно. Сайт нужен не только различным предприятиям, но и государственным структурам, и известным людям. Можно сказать, что сайт необходим абсолютно всем, кто хочет «заявить о себе».

Отказ от продвижения своего бизнеса в сети Интернет, означает потерю клиентов в реальном мире. Web-сайт компании является важнейшим источником информации для потенциальных клиентов.

**Цель проекта.**

Разработать интернет-магазин по продаже товаров.

**План работы.**

1. Получение технического задания от заказчика;
2. Выбор технологий;
3. Создание проекта;

4. Реализация проекта согласно техническому заданию;

1. Подключение к базе данных;
2. Тестирование приложения (написание и запуск тестов);
3. Запуск приложения и поверка приложения;
4. Выводы и заключения о проделанной работе.

**Какую проблему решает.**

Создание web-сайта для предпринимателя позволит увеличить поток клиентов, увеличить количество продаж, привлечь к себе внимание со стороны потенциальных покупателей, позволит оставаться актуальным на рынке.

**Специализация дипломного проекта:**

Backend разработка web-сайта.

**Задачи:**

Работа заключается в изучении современных технологий и средств разработки различных сайтов (визитки, корпоративные, интернет-магазины и т. д.), в выборе оптимального средства реализации проекта, самой реализации и документировании проделанной работы в ПЗ. У выбранного предпринимателя отсутствует личный web-сайт, поэтому для поддержания его конкурентоспособности на рынке необходимо было разработать и внедрить сайт в деятельность предприятия. Глобальная задача заключалась в необходимости опробовать знания, полученные за все время обучения, подтвердить их актуальность и при необходимости расширить.

**Инструменты:** Java, Java Spring, Git, Docker, ProgreSQL

**Состав команды**: Истомин Сергей Андреевич (Разработчик)

# Теоретическая и практическая главы.

В данном разделе будут описаны технологии и инструменты применяемые в проекте.

## 1. Получение технического задания от заказчика.

Заказчиком предоставлено следующее техническое задание:

Разработать минимальную модель web-сайта интернет-магазина с возможностью дальнейшего развития функциональности сайта для ИП Грига Алена Геннадиевна, с целью анализа увеличения клиентского трафика и создания позитивного образа магазина.

В данный момент сайт нужен с минимальным набором функций.

Страницы и функционал доступный для посетителей сайта:

Страницы: домашняя страница, с товарами, авторизации, желаемых товаров, список заказов.

Регистрация для посетителей сайта.

Просмотр и покупка товаров.

Возможность добавления товаров в список желаний.

Переход к покупке из списка желаний.

Страницы и функционал доступный для администратора сайта:

Страницы: панель администратора, панель добавления продукта.

Возможность выполнения операций над товарами.

Хочется понять, насколько данный инструмент позволит выйти на новый уровень в работе бизнеса.

## 2. Выбор технологий.

Для реализации проекта по предоставленному техническому заданию были выбраны следующие технологии:

- Spring Boot;

- Hibernate;

- Thymeleaf;

- Maven;

- PostgreSQL;

- Docker.

## 3. Создание проекта.

Термины:

**Maven** – инструмент для автоматизации сборки проектов, основанный на модели проекта Project Object Model (POM), который управляет компиляцией, тестированием, пакетированием и развертыванием приложений.

**Gradle** – современная система сборки, которая объединяет лучшие черты Maven и Ant, позволяя автоматизировать процесс сборки проектов с помощью гибкого и мощного языка Groovy.

**Зависимости** – внешние библиотеки и модули, которые необходимы для работы проекта и должны быть загружены и подключены в процессе сборки.

**Репозиторий** – хранилище, содержащее артефакты (библиотеки, плагины) и метаданные, используемые системами сборки для управления зависимостями проекта.

**Артефакт** – результат сборки проекта (например, JAR-, WAR- или EAR-файлы), который может быть опубликован в репозитории и использован другими проектами как зависимость.

**Плагин** – дополнительный компонент, расширяющий возможности системы сборки и позволяющий автоматизировать различные задачи, такие как компиляция, тестирование, пакетирование и деплой приложений.

### 3.1 Системы сборки Maven и Gradle для разработки Java приложений.

Системы сборки играют ключевую роль в разработке программного обеспечения.

Системы сборки облегчают разработку, поддержку и развертывание программного обеспечения. Они автоматизируют процессы компиляции, тестирования, пакетирования и дистрибуции приложений, а также обеспечивают управление зависимостями между различными модулями и библиотеками.

Выбор будем делать среди двух популярных инструментов для Java-проектов: Maven и Gradle.

Maven - это мощный инструмент, который использует структуру проектов на основе XML. Он предоставляет широкий набор плагинов и позволяет разработчикам создавать сложные проекты с множеством зависимостей и настроек. Один из его основных принципов - “соглашение против конфигурации”, что означает, что Maven предполагает наличие определенной структуры и настроек в вашем проекте, что упрощает работу с ним.

Gradle - это относительно новый и гибкий инструмент сборки, который использует язык Groovy или Kotlin DSL для определения сценариев сборки. Gradle позволяет создавать мощные и сложные сценарии сборки, но в то же время его легко настроить под нужды разработчика. Это делает его особенно популярным среди разработчиков Android-приложений.

Помимо компиляции исходного кода, системы сборки выполняют множество дополнительных функций, таких как:

1. Управление зависимостями: Одна из самых важных функций систем сборки - это управление зависимостями между библиотеками и модулями вашего проекта. Maven и Gradle позволяют автоматически загружать и интегрировать библиотеки из удаленных репозиториев, таких как Maven Central или JCenter. Например, если одному из модулей вашего проекта потребуется библиотека Jackson, системы сборки смогут ее добавить.

2. Тестирование: Системы сборки интегрируются с различными фреймворками тестирования, такими как JUnit или TestNG, и автоматически запускают тесты во время сборки проекта. Это обеспечивает постоянное контролирование качества вашего кода.

3. Генерация документации: Maven и Gradle могут автоматически генерировать документацию для вашего кода, используя такие инструменты, как Javadoc. Это значительно облегчает поддержку и развитие проекта на долгосрочной основе.

4. Пакетирование и дистрибуция: Системы сборки создают исполняемые файлы и архивы (например, JAR, WAR или EAR) для развертывания и распространения вашего приложения. Это делает процесс дистрибуции быстрым и непринужденным.

5. Поддержка плагинов: Maven, и Gradle имеют мощную систему плагинов, которая позволяет расширять функциональность системы сборки путем добавления дополнительных задач и интеграции с другими инструментами, такими как системы контроля версий или инструменты для непрерывной интеграции.

### 3.2 Сравнение Maven и Gradle, что будет использовано в проекте.

Maven и Gradle - две мощные системы сборки, каждая со своими особенностями и преимуществами.

|  |  |  |
| --- | --- | --- |
| **Критерий** | **Maven** | **Gradle** |
| *Структура проекта* | XML-структура (pom.xml) | Groovy или Kotlin DSL (build.gradle или build.gradle.kts) |
| *Читабельность* | Менее читабельный из-за XML | Более читабельный благодаря DSL |
| *Скорость* | Стабильный, но медленнее | Быстрый и производительный |
| *Гибкость* | Меньше гибкости, ограничения | Больше гибкости и возможностей |

Теперь, когда обозначены основные различия между Maven и Gradle, необходимо сделать выбор между этими инструментами.

Критерии выбора:

Простота и упорядоченность, работа над проектом, где можно использовать стандартные соглашения, Maven является отличным выбором;

Если проект требует большей гибкости, или необходимо использовать мощные сценарии сборки и быстро разрабатывать, то лучше использовать Gradle.

Важно помнить, что выбор между Maven и Gradle не является однозначным и зависит от предпочтений, опыта и особенностей проекта.

Для данного проекта мною был выбран Maven.

### 3.3 Краткая теория Maven.

Основные понятия Maven:

1. POM (Project Object Model): POM - это XML-файл, который описывает проект и его настройки. Он является сердцем проекта Maven и включает информацию о зависимостях, плагинах, версиях и других параметрах.

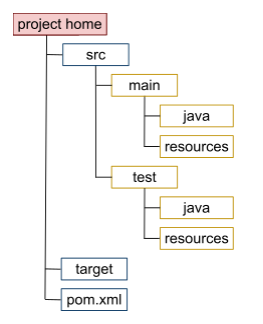
2. Зависимости - это внешние библиотеки или модули, необходимые для корректной работы вашего приложения. Maven автоматически управляет зависимостями, загружая их из удаленных репозиториев и интегрируя в ваш проект.

3. Плагины - это компоненты, которые расширяют функциональность Maven, добавляя дополнительные задачи и интеграцию с другими инструментами. Примерами плагинов являются maven-compiler-plugin (для компиляции кода) и maven-surefire-plugin (для тестирования).

4. Жизненный цикл - это последовательность фаз, определяющих процесс сборки проекта. Основные циклы жизни в Maven включают “clean” (очистка сборки), “default” (основная сборка) и “site” (генерация сайта проекта).

Структура проекта Maven:

Стандартная структура проекта Maven состоит из следующих каталогов и файлов:



src/main/java: Здесь хранится исходный код приложения.

src/main/resources: Здесь находятся ресурсы приложения, такие как файлы конфигурации, изображения и т. д.

src/test/java: Здесь размещается исходный код тестов приложения.

src/test/resources: Здесь хранятся ресурсы, связанные с тестированием.

pom.xml: Это основной файл конфигурации Maven, содержащий информацию о проекте и его настройках.

Жизненные циклы Maven:

Maven определяет три стандартных жизненных цикла:

1. clean: Этот жизненный цикл отвечает за удаление всех файлов, созданных в результате предыдущей сборки. Он состоит из трех фаз: pre-clean, clean и post-clean.

2. default: Этот жизненный цикл является основным и отвечает за сборку, тестирование, пакетирование и развертывание вашего проекта. Он состоит из большого количества фаз, включая compile, test, package, install и deploy. Рассмотрим основные фазы этого жизненного цикла ниже.

3. site: Этот жизненный цикл отвечает за создание документации и сайта вашего проекта. Он включает фазы pre-site, site и post-site.

Основные фазы жизненного цикла default:

1. validate: в этой фазе проверяется корректность настроек проекта и отсутствие проблем с конфигурацией.

2. compile: в этой фазе исходный код проекта компилируется в байт-код Java.

3. test: в этой фазе выполняются тесты приложения. Обратим внимание, что этот этап не вызывает остановку сборки в случае неудачных тестов.

4. package: в этой фазе создается артефакт проекта (например, JAR-файл).

5. verify: в этой фазе выполняются проверки, чтобы убедиться, что пакет соответствует качеству и критериям проекта.

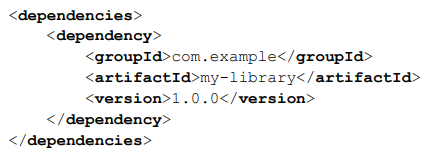
6. install: в этой фазе артефакт проекта устанавливается в локальный репозиторий Maven, чтобы быть доступным для других проектов на компьютере.

7. deploy: в этой фазе артефакт проекта развертывается в удаленном репозитории, чтобы быть доступным для других разработчиков и команд.

**Зависимости.**

Зависимости — это внешние библиотеки или модули, которые требуются для корректной работы приложения. В Maven зависимости объявляются в POM-файле, и система автоматически управляет ими, загружая необходимые артефакты из репозиториев и интегрируя их в проект.

Пример объявления зависимости в POM-файле:



В примере указана зависимость от библиотеки my-library версии 1.0.0 от com.example. Maven автоматически загрузит эту библиотеку и сделает ее доступной для проекта.

**Репозитории.**

Репозитории — это централизованные хранилища артефактов, таких как библиотеки и плагины. В Maven существует три типа репозиториев:

1. Локальный репозиторий — это каталог на сервере, где Maven хранит загруженные артефакты. По умолчанию локальный репозиторий находится в каталоге ~/.m2/repository. Загруженные артефакты доступны для всех проектов на компьютере.

2. Центральный репозиторий — это публичное хранилище артефактов, доступное всем разработчикам. Maven Central Repository (https://repo.maven.apache.org/maven2/) — это наиболее известный и широко используемый центральный репозиторий. Большинство популярных библиотек и плагинов доступны в Maven Central.

3. Удаленный репозиторий — это хранилище артефактов, находящееся на удаленном сервере. Удаленные репозитории могут быть публичными или приватными и предназначены для совместной работы разработчиков внутри команды или организации. Один из примеров публичного удаленного репозитория — JCenter (https://jcenter.bintray.com/). Для приватных репозиториев обычно используются такие решения, как Nexus Repository Manager или JFrog Artifactory. Maven автоматически ищет зависимости в локальном репозитории, затем в центральном репозитории. Если артефакт не найден, можно настроить POM-файл, чтобы добавить ссылку на удаленный репозиторий. Вот пример добавления удаленного репозитория в POM-файл:

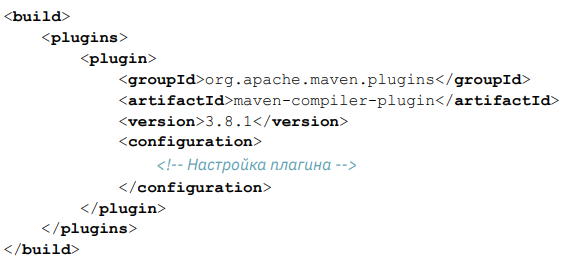


После добавления удаленного репозитория, Maven будет искать артефакты в этом хранилище в случае, если они не найдены в локальном и центральном репозиториях.

**Плагины.**

Плагины — это расширения Maven, предоставляющие дополнительные функции и возможности для управления процессом сборки проекта. Плагины состоят из одной или нескольких задач (goals), которые могут быть вызваны в различных фазах жизненного цикла сборки.

Maven предоставляет множество стандартных плагинов, таких как компиляция исходного кода, тестирование, пакетирование и развертывание. В проекте можно использовать сторонние плагины или создать собственные, чтобы расширить функциональность Maven. Чтобы добавить плагин в проект, нужно объявить его в POM-файле внутри элемента (ниже приведен пример):



В этом примере добавлен плагин maven-compiler-plugin, который отвечает за компиляцию исходного кода проекта. Элемент позволяет настроить плагин в соответствии с требованиями проекта.

Настройка проекта:

POM-файл является основой для настройки проекта в Maven. В POM-файле необходимо указать информацию о проекте, такую как группа, идентификатор артефакта, версия, а также настроить зависимости, плагины, репозитории и другие элементы.

Основные элименты в POM-файле:

1. Основная информация: включает элементы groupId, artifactId, version, packaging. Они определяют уникальный идентификатор проекта, версию и тип пакета (например, JAR или WAR).

2. Зависимости: элемент используется для указания зависимостей проекта.

3. Репозитории: элемент позволяет добавить удаленные репозитории для поиска артефактов, которые недоступны в центральном репозитории.

4. Плагины: внутри элемента находится элемент, который содержит объявления плагинов, используемых в проекте.

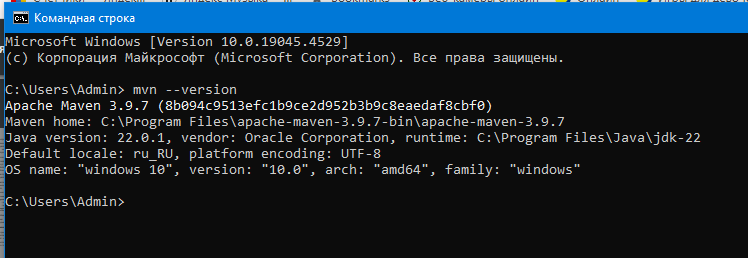
5. Свойства: элемент позволяет задавать переменные, которые могут быть использованы для настройки плагинов и других элементов POM-файла.

6. Профили: элемент позволяет определить различные конфигурации для разных сред разработки или сценариев сборки.

Настройка POM-файла является важным аспектом разработки с использованием Maven, так как это позволяет адаптировать процесс сборки к требованиям проекта и сделать его максимально эффективным.

### 3.4 Установка Maven.

Установка Maven выполнена согласно инструкциям на официальном сайте: https://maven.apache.org/install.html. Убедиться, что Maven работает, можно выполнив команду mvn --version в командной строке. Видим, что все работает.



### 3.5 Генерация проекта.

Для генерации проекта был использован такой web инструмент как spring initializr, представленный в сети по адресу <https://start.spring.io/>

Поскольку для проекта был выбран Maven, в генирации проекта были использованы следующие параметры:

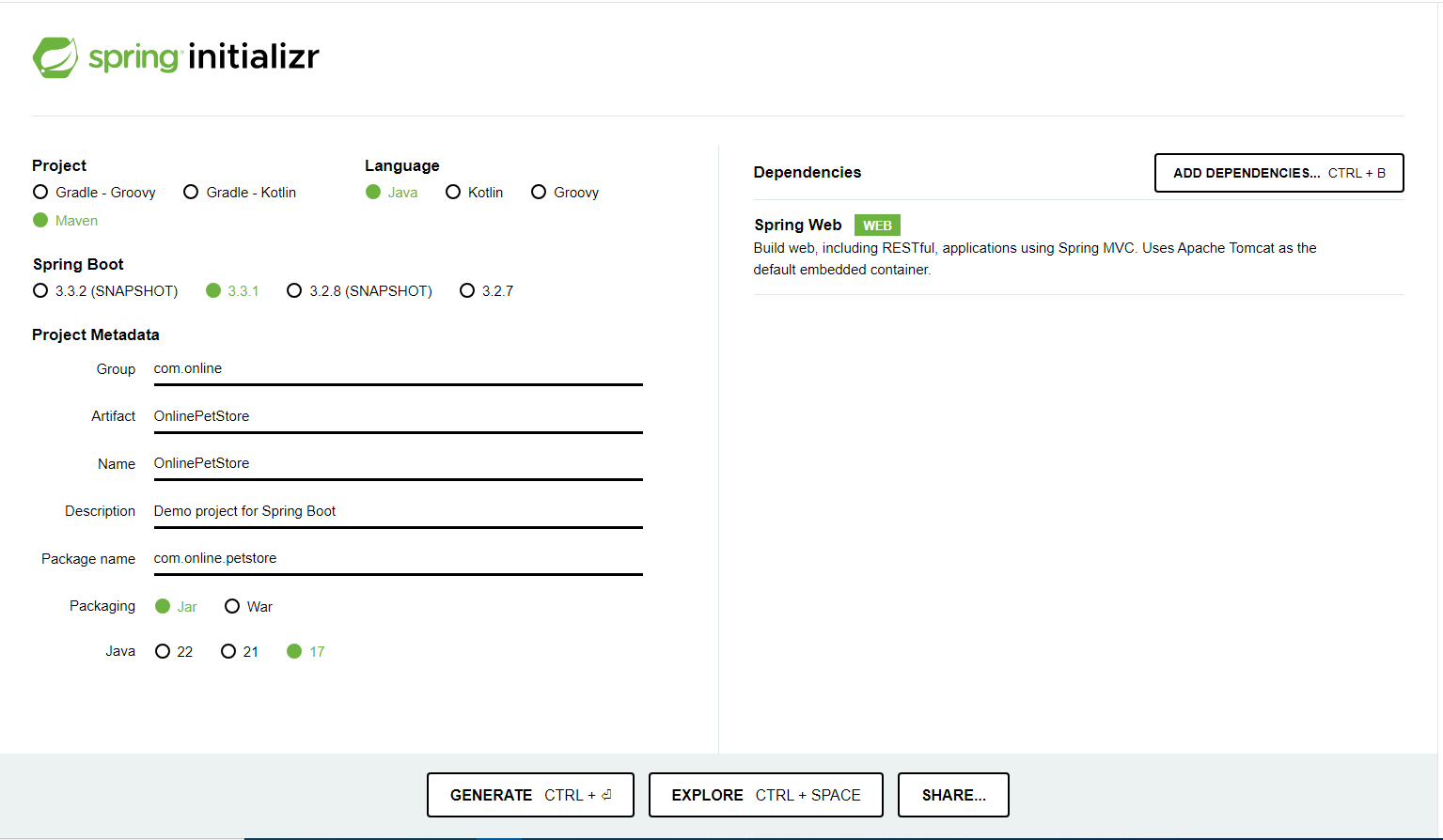
- Maven;

- Java (так как приложение будет написано на Java);

- Spring Boot последней доступной версии;

- версия java 18;

- из зависимостей на данном этапе достаточно будет достаточно Spring Web остальное будет добавляться по мере необходимости.



Генерируем проект и открываем его в среде разработки. В проекте используется intellij idea.

## 4. Spring Boot.

Термины:

**Spring** –фреймворк, созданный для упрощения разработки Java-приложений.

**Java-приложения** – программы, написанные на языке программирования Java.

**Бины (Beans)** – объекты, управляемые Spring и добавляемые в контейнер IoC.

**@Component** – общая аннотация Spring, которую можно использовать для определения любого бина.

**@Service** – специализированная версия аннотации @Component, предназначенная для классов, которые представляют бизнес-логику приложения @Repository – специализированная версия аннотации.

**@Component**, предназначенная для классов, которые взаимодействуют с системой хранения данных.

**@Controller** – специализированная версия аннотации @Component, предназначенная для классов, которые обрабатывают HTTP-запросы в веб-приложениях Spring MVC или Spring WebFlux.

**@Configuration** – аннотация, используемая для классов, которые определяют бины с помощью методов @Bean

**@Bean** – аннотация, используемая вместе с @Configuration для определения бинов.

**DI (Dependency Injection)** – концепция, позволяющая делать приложения более гибкими и легкими для тестирования путем внедрения зависимостей извне, а не создания их внутри класса.

**IoC (Inversion of Control**) – концепция, в соответствии с которой не приложение.

**Tomcat** – открытый веб-сервер и сервлет-контейнер, реализующий спецификации Java Servlet и JavaServer Pages (JSP).

### 4.1 Основы Spring.

Что такое Spring и какие преимущества он имеет.

**Spring** — это мощный фреймворк, созданный для упрощения разработки Java-приложений. Он базируется на нескольких ключевых принципах, вот два из них: DI (Dependency Injection) и IoC (Inversion of Control).

**DI**, или внедрение зависимостей, - это концепция, которая позволяет нам делать приложения более гибкими и легкими для тестирования.

**IoC**, или инверсия управления, - это еще одна важная концепция в Spring. Она означает, что не приложение контролирует жизненный цикл его компонентов (как это обычно происходит), а наоборот - фреймворк контролирует приложение.

Когда запускается приложение на Spring, первое, что происходит, - это запуск встроенного сервера, такого как Tomcat. Этот сервер слушает входящие HTTP-запросы и передает их приложению. Все начинается с файла конфигурации Spring, который описывает, какие компоненты (или “бины”, как их называют в Spring) должны быть созданы и как они связаны друг с другом. Когда сервер запускается, Spring читает этот файл и создает все необходимые бины. Затем он “внедряет” эти бины в нужные места, используя DI. После этого приложение готово к работе. Spring также предлагает множество других возможностей. Он поддерживает различные способы конфигурации, включая аннотации и Java-конфигурацию. Он предлагает мощную поддержку тестирования, что позволяет вам легко тестировать отдельные компоненты вашего приложения. Он также включает в себя множество других модулей для различных задач, таких как работа с базами данных, обработка веб-запросов и многое другое. В Spring Framework существует множество аннотаций, которые используются для определения и конфигурации бинов. Вот несколько основных из них:

**1.** **@Component**: это общая аннотация, которую можно использовать для определения любого бина. Классы, аннотированные как @Component, автоматически сканируются Spring и регистрируются в контейнере IoC.

**2. @Service**: это специализированная версия @Component, предназначенная для классов, которые представляют бизнес-логику приложения. Она не добавляет дополнительной функциональности по сравнению с @Component, но помогает лучше структурировать код.

**3. @Repository**: это еще одна специализированная версия @Component, предназначенная для классов, которые взаимодействуют с системой хранения данных. Она может интегрироваться с механизмом перехвата исключений Spring Data Access, который автоматически преобразует исключения хранилища данных в исключения Spring DataAccessException.

**4. @Controller**: это специализированная версия @Component, предназначенная для классов, которые обрабатывают HTTP-запросы в веб-приложениях Spring MVC или Spring WebFlux.

**5. @Configuration**: эта аннотация используется для классов, которые определяют бины с помощью методов @Bean. Эти классы играют роль источников определения бинов для контейнера IoC.

**6. @Bean**: эта аннотация используется вместе с @Configuration для определения бинов. Методы, аннотированные как @Bean, создают объекты, которые управляются Spring и добавляются в контейнер IoC.

### 4.2 Разница между Spring и Spring Boot.

В чем разница между Spring и Spring Boot? В основе Spring Boot лежит Spring, но он предлагает множество дополнительных возможностей. В частности, Spring Boot предоставляет “оптимальные настройки по умолчанию”, что позволяет быстро начать разработку без необходимости настраивать все с нуля. Он также включает в себя множество “стартеров”, которые автоматически включают в приложение нужные зависимости. Использование Spring предоставляет множество возможностей и позволяет сосредоточиться на самой разработке, а не на решении технических проблем. Упрощает разработку, делает код более чистым и понятным, а также помогает быстро и эффективно создавать высококачественные приложения.

Spring Boot, по сути, это Spring, но с дополнительными возможностями, предоставляемыми “из коробки”. Он автоматизирует конфигурацию, упрощает управление зависимостями и предлагает множество других удобных функций.

### 4.3 Типы бинов.

В Spring Framework существует несколько типов бинов, основанных на их области видимости. Область видимости бина определяет, когда и как создается новый экземпляр бина. В Spring определены следующие области видимости:

**1. Singleton**: это область видимости по умолчанию. Когда бин определен как Singleton, Spring IoC контейнер создает единственный экземпляр бина, и все запросы на получение этого бина возвращают один и тот же объект. Это подходит для бинов, которые не содержат состояния, таких как сервисы или DAO.

**2. Prototype**: когда бин определен как Prototype, Spring IoC контейнер создает новый экземпляр бина каждый раз, когда он запрашивается. Это подходит для бинов, которые содержат состояния и не могут быть использованы одновременно в разных компонентах или потоках.

**3. Request, Session, и Application**: эти области видимости применяются только в веб-приложениях. Бин области видимости Request создается для каждого HTTP-запроса. Бин области видимости Session создается для каждого HTTP-сеанса. Бин области видимости Application связан с жизненным циклом ServletContext и обычно используется для хранения данных на уровне приложения.

**4. WebSocket**: эта область видимости доступна для бинов, которые должны быть связаны с жизненным циклом WebSocket. Для определения области видимости бина в Spring, вы можете использовать аннотацию @Scope.

### 4.4 Spring Framework две основные реализации контейнера.

Spring Framework имеет две основные реализации контейнера:

**1. BeanFactory**: это базовый контейнер, который предоставляет функциональность IoC. Определяет основной контракт, который должны выполнять все контейнеры. BeanFactory использует конфигурацию (XML или Java-based) для создания и управления объектами и их зависимостями. Однако не предоставляет дополнительных возможностей, таких как интеграция с другими компонентами Spring, поддержка аннотаций и т. д.

**2. ApplicationContext**: это расширение BeanFactory, которое предоставляет дополнительные возможности, такие как интеграция с другими компонентами Spring (например, AOP, Web), поддержка аннотаций и др. ApplicationContext является предпочтительным контейнером для большинства приложений, поскольку предоставляет более широкий набор функций.

Контейнер работает следующим образом:

1. Чтение конфигурации: контейнер считывает конфигурацию приложения, которая может быть определена с помощью XML, аннотаций или Java-based конфигурации.

2. Создание бинов: на основе предоставленной конфигурации контейнер создает и инициализирует объекты (бины) и их зависимости.

3. Управление жизненным циклом бинов: контейнер управляет жизненным циклом бинов, вызывая их методы инициализации и разрушения в соответствии с конфигурацией.

4. Внедрение зависимостей: контейнер автоматически внедряет зависимости между бинами, используя конфигурацию и аннотации.

5. Предоставление бинов: когда приложение запрашивает бин из контейнера, он предоставляет соответствующий экземпляр бина, учитывая его область видимости (singleton, prototype и т. д.).

### 4.5 Конфигурация Spring Boot приложения.

В Spring Boot есть два основных способа конфигурации: через файлы application.properties или application.yml и через Java-код. В проекте будет использован файл конфигурации .properties

.properties - это старый добрый формат файлов конфигурации Java. Каждый параметр задается в виде пары “ключ-значение”, разделенных знаком равенства.

При запуске приложения Spring Boot автоматически загружает файлы конфигурации. Внутренне Spring использует библиотеку spring-boot-autoconfigure, которая предоставляет функциональность автоматической настройки. Spring Boot автоматически сопоставляет значения из файлов конфигурации с полями в классах, аннотированных как @ConfigurationProperties. Эти классы обычно используются для группировки связанных свойств конфигурации.

### 4.6 Клиент-сервер.

Клиентом в разрабатываемом приложении будет выступать браузер, который отправляет HTTP-запросы на сервер. Сервер – это само приложение, которое слушает эти запросы, обрабатывает их и отправляет обратно HTTP-ответы. HTTP-запросы и ответы являются фундаментальными блоками веб-взаимодействия. Запрос может включать в себя метод (GET, POST, PUT, DELETE), URL (адрес, куда отправляем запрос), заголовки (дополнительная информация о запросе) и тело запроса (основная информация, которую отправляем).

Виды протоколов:

HTTP, или гипертекстовый протокол передачи данных, – это основной протокол, используемый в вебе для передачи данных. Представьте это можно как почтовую систему Интернета. Когда отправляется запрос на получение веб-страницы, по сути, отправляется письмо серверу с просьбой отправить веб-страницу.

HTTPS – это просто защищенная версия HTTP. Это все еще та же почтовая система, но теперь все письма запечатаны в безопасные конверты, которые никто не может прочитать, кроме получателя. Иными словами, HTTPS использует шифрование для защиты данных от перехвата.

WebSocket– это другой протокол, который предназначен для обеспечения двусторонней связи между клиентом и сервером. Если HTTP можно представить как почтовую систему, то WebSocket больше похож на телефонный разговор. Вместо того чтобы отправлять и получать письма, можно говорить в обе стороны в реальном времени.

gRPC – это пртокол от Google. Он поддерживает множество языков программирования и позволяет клиентам и серверам обмениваться данными как через единственный запрос/ответ, так и через потоковые вызовы. Если сравневать его с чем-то, то это электронный курьер, который может переносить любой тип пакета очень быстро.

В проекте будет использоваться HTTP.

### 4.7 Виды HTTP-запросов и коды ответов.

**HTTP-методы.**

HTTP определяет ряд методов, которые соответствуют различным операциям, которые можно выполнить с ресурсом:

GET: ииспользуется для запроса данных с сервера. Это как спросить: “Привет, сервер, можно я посмотрю эту веб-страницу?”

POST: ииспользуется для отправки данных на сервер, чтобы создать новый ресурс. Это как сказать: “Привет, сервер, вот новые данные, добавь их, пожалуйста, в свою базу данных.”

PUT: ииспользуется для обновления существующего ресурса на сервере. Это как сказать: “Привет, сервер, вот обновленные данные, пожалуйста, замени ими старые.”

DELETE: ииспользуется для удаления ресурса на сервере. Это как сказать: “Привет, сервер, пожалуйста, удали эти данные.”

**Часто встречающиеся HTTP-коды ответов.**

HTTP-коды ответов – это трехзначные числа, которые сервер возвращает, чтобы указать, что произошло с запросом. Вот несколько наиболее часто встречающихся:

200 OK: запрос был успешно обработан. Это как “Ваша операция прошла успешно!”

201 Created: запрос был успешно обработан, и был создан новый ресурс. Это как “Я создал новые данные, которые вы просили!”

400 Bad Request: запрос неправильно сформирован и сервер не может его обработать. Это как “Я не понимаю, что вы просите.”

404 Not Found: запрошенный ресурс не найден на сервере. Это как “Извините, я не могу найти данные, которые вы просили.”

500 Internal Server Error: на сервере произошла ошибка при обработке запроса. Это как “У меня возникла проблема, и я не смог обработать ваш запрос.”

### 4.8 Контроллеры в Spring.

Связь между Spring и HTTP существует благодаря особой части Spring MVC, известной как контроллер.

Контроллер в Spring — это компонент, который обрабатывает входящие HTTP-запросы. Когда HTTP-запрос приходит к приложению, контроллер указывает, какой метод (или действие) должен быть вызван для обработки этого запроса.

Внедрение зависимостей - это подход, при котором объект не создает или ищет свои зависимости самостоятельно, вместо этого они предоставляются ему извне. Если контроллеру в Spring нужно выполнить какую-то работу (например, обработать HTTP-запрос), все зависимости, которые ему для этого нужны (например, службы для работы с данными), предоставляются ему автоматически благодаря внедрению зависимостей. Это значительно упрощает код и делает его более гибким и тестируемым, поскольку зависимости могут быть легко заменены на фиктивные объекты во время тестирования.

Аннотация @Controller говорит Spring'у, что данный класс является контроллером, и должен быть использован для обработки входящих HTTP-запросов.

Аннотация @RequestMapping используется для указания, какой тип запроса и какой URL должен обрабатывать конкретный метод, может быть использована на уровне класса (для указания общего префикса URL для всех методов контроллера) и на уровне метода (для указания конкретного URL и метода HTTP).

Чтобы принять параметры запроса из URL, строки запроса или тела запроса, мы можем использовать аннотации `@PathVariable`, `@RequestParam` и `@RequestBody` соответственно.

Чтобы вернуть ответ из обработчика, мы просто возвращаем объект, который должен быть включен в тело ответа. Spring автоматически преобразует его в нужный формат (обычно JSON).

## 5. Spring MVC.

Термины:

**Spring MVC** — это фреймворк на базе Spring, который предлагает модель-вид-контроллер (MVC) архитектуру и готовые компоненты, которые могут быть использованы для разработки гибких и свободно масштабируемых веб-приложений.

**Thymeleaf** — это современный серверный Java-шаблонизатор, способный обрабатывать как HTML, так и XML. Он хорошо интегрируется с Spring MVC и обеспечивает полноценную поддержку HTML5.

**Контроллер (Controller)** — в контексте Spring MVC, контроллер - это класс, который обрабатывает веб-запросы от клиента. Контроллеры обычно аннотированы @Controller.

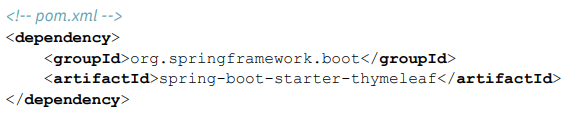
**Модель (Model)** — модель представляет собой данные, которые будут отображаться пользователю. Модель может быть любым Java объектом, который может быть сохранен в базе данных.

**Вид (View)** — в Spring MVC вид представляет собой то, что будет отображаться пользователю. Это может быть JSP-страница, HTML-страница, PDF-документ, Excel-документ и т.д.

**DispatcherServlet** — это сердце Spring MVC, которое обрабатывает входящие запросы и маршрутизирует их к соответствующим контроллерам.

**WebApplicationContext** — специфический для веб-приложений контекст Spring, который предоставляет конфигурацию для приложения в виде bean-компонентов.

Основной целью Thymeleaf является создание элегантного и удобного способа шаблонизации. Чтобы достичь этого, Thymeleaf основывается на концепции Natural Templates, чтобы внедрить свою логику в файлы шаблонов таким образом, чтобы этот шаблон не влиял на отображение прототипа дизайна. Это улучшает коммуникацию в команде и уменьшает разрыв между дизайнерско-программистскими группами.  
  
Thymeleaf также был разработан с самого начала с учетом стандартов Web, особенно **HTML5**, что позволяет вам создавать полностью соответствующие стандарту шаблоны.

Для использования шаблонизатора Thymeleaf необходимо добавить зависимости в POM.xml  


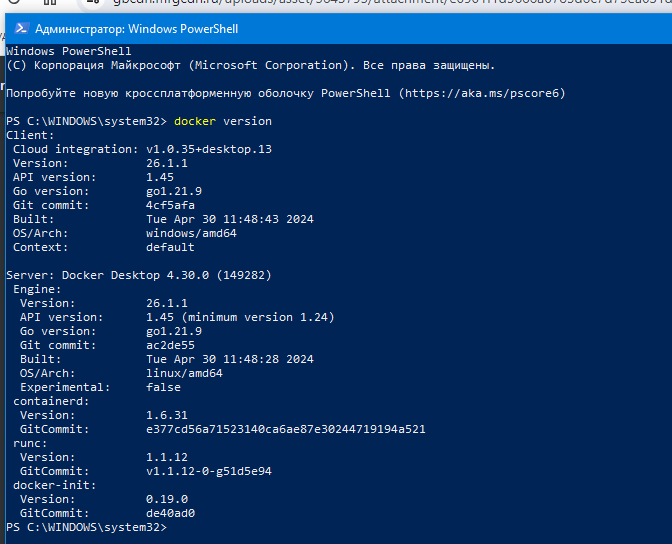
## 6. Подключения базы данных в проекте.

Для реализации обработки и хранения данных проекта используется база данных PostgreSQL в Docker контейнере, запущенном на отдельном сервере в локальной сети.

Использование PostgreSQL в [контейнере](https://www.nic.ru/help/chto-takoe-docker-kontejner_11346.html) предоставляет высокую степень изоляции базы данных (БД). Контейнеры позволяют упаковать PostgreSQL и все его зависимости в одну изолированную среду, что значительно упрощает процесс развертывания и управления базой данных, а также обеспечивает предсказуемость работы приложений, использующих эту БД.

**Docker** — это платформа для разработки, доставки и запуска контейнерных приложений. Docker позволяет создавать контейнеры, автоматизировать их запуск и развертывание, управляет жизненным циклом.   
 **Контейнеры —** это способ стандартизации развертки приложения и отделения его от общей инфраструктуры. Экземпляр приложения запускается в изолированной среде, в проекте создан контейнер с базой данных PostgreSQL.

Для реализации подключения в проекте базы данных, был установлен Docker Desktop с официального сайта: <https://docs.docker.com/desktop/install/windows-install/> (вход осуществлен с помощью учетной записи GitHub).

Проверяем Docker на сервере, в PowerShell вводим команду docker version. Видим что Docker успешно установлен.   
  
 Для создания контейнера с PostgreSQL используем команду docker run, а имеено:

docker run --name dbmy -e POSTGRES\_DB=dbmy -e POSTGRES\_USER=user -e POSTGRES\_PASSWORD=123456 -p 5432:5432 -d postgres  
где:

--name dbmy имя контейнера

-e POSTGRES\_DB=dbmy имя базы данных

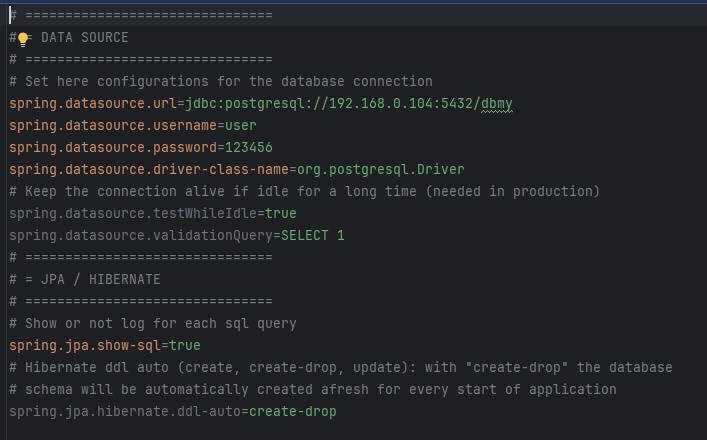
-e POSTGRES\_USER=user имя пользователя

-e POSTGRES\_PASSWORD=123456 пароль

-p 5432:5432 порт для подключения

-d postgres запуск контейнера в фоновом режиме

Затем в проекте создаем файл application.properties, в котором прописываем параметры для подключения к базе данных.



url: jdbc:postgresql://192.168.0.104:5432/dbmy адрес подключения к базе данных (база данных настроена и подключена в локальной сети на другом сервере с ip адресом 192.168.0.104)

username: user имя пользователя

password: 123456 пароль

driverClassName: org.postgresql.Driver указываем драйвер

ddl-auto: update даем разрешение приложению обновлять схему при необходимости

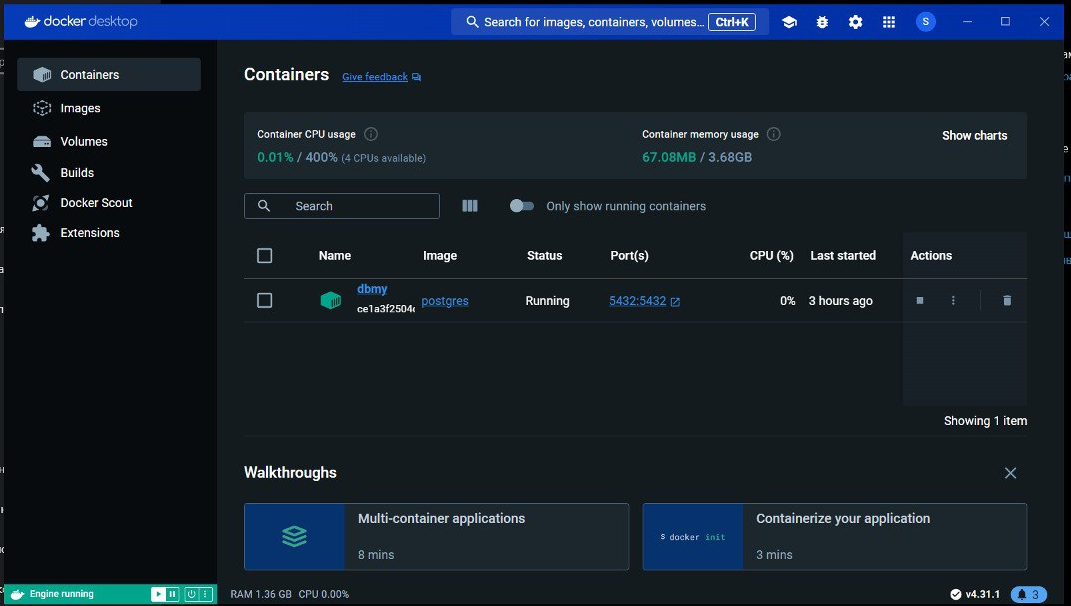
Так же добавляем зависимость для работы с базой данных в файл pom.xml:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>

И драйвер:

<dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <version>42.7.3</version>  
</dependency>

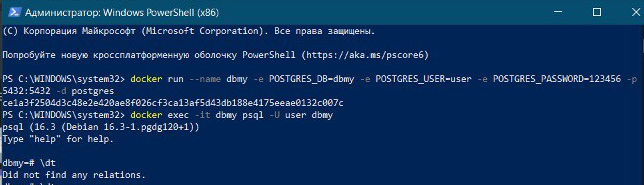
После чего проверяем в Docker Desktop, что контейнер запущен:



В PowerShell заходим в базу данных, используя команду

docker exec -it dbmy psql -U user dbmy

Видим, что пока таблиц нет



## 7. API.

API, или Application Programming Interface, это набор функций и процедур, которые позволяют клиентам взаимодействовать с сервером. Это некий мост между вашими данными и внешним миром.

API, или интерфейс программирования приложений, это своеобразная «договоренность», контракт между клиентом и сервером. Он указывает, какие запросы может делать клиент, и какие ответы он должен ожидать. Этот процесс регулируется API. API могут быть различными: веб-API, операционные системы имеют свои API, библиотеки программирования также используют API. Но в контексте веб-разработки на Spring - это веб-API. Это позволяет клиентам, например, веб-браузерам или мобильным приложениям, общаться с нашим сервером, делать запросы и получать ответы. В общем, API – это ключевой элемент любого веб-приложения.

CRUD – это аббревиатура, которая означает четыре базовые функции, которые большинство веб-сервисов обеспечивают пользователям: Создание (Create), Чтение (Read), Обновление (Update) и Удаление (Delete). Называют их так, потому что каждая буква в слове “CRUD” соответствует начальной букве каждой из этих операций на английском языке.

1. Создание (Create): эта операция обеспечивает возможность создания нового объекта в базе данных.

2. Чтение (Read): операция чтения позволяет пользователям получать данные из базы.

3. Обновление (Update): операция обновления позволяет изменить существующие данные.

4. Удаление (Delete): и, наконец, операция удаления. Она позволяет удалить существующие данные.

**На что необходимо обращать внимание при проектировании API:**

**1. Непредсказуемое поведение:** если API неправильно спроектирован, он может работать не так, как ожидается. Это может привести к ошибкам, багам и проблемам при взаимодействии клиентов с сервером.

**2. Сложности в поддержке:** неправильно спроектированный API может быть сложно поддерживать и обновлять. Это может стать большой проблемой, особенно когда ваше приложение растет и развивается.

**3. Проблемы с безопасностью:** несмотря на то что важность безопасности при проектировании API трудно переоценить, недостаточно хорошо спроектированный API может иметь уязвимости, которые злоумышленники могут использовать для атаки на ваше приложение.

**4. Плохой пользовательский опыт:** если ваш API сложно использовать или понять, разработчики, которые пытаются использовать ваш API, могут столкнуться с проблемами. Это может отпугнуть их от использования вашего API или приложения.

## 8. Spring Security.

**Spring Security** – это мощный и настраиваемый механизм безопасности для приложений Spring, позволяющий интегрировать различные средства аутентификации и авторизации.

**JWT (JSON Web Token)** – это компактное, URL-безопасное средство представления между двумя сторонами. Они обычно используются для передачи данных аутентификации и авторизации. Bearer Token – тип токена авторизации, который предоставляется клиентом при каждом запросе. Это часто JWT.

**Аутентификация** - это процесс подтверждения идентичности пользователя.

**Авторизация** – это процесс, в рамках которого определяется, какие действия разрешены пользователю в системе.

**Принципы безопасности.**

**1. Минимальные привилегии:** принцип минимальных привилегий говорит о том, что каждый пользователь (или процесс) должен иметь минимально необходимые права для выполнения своих функций. Это означает, что пользователи не должны иметь доступа к функциям или данным, которые они не нуждаются для выполнения своих задач. Это прямо связано с авторизацией, о которой мы говорили ранее.

**2. Защита данных:** данные, особенно чувствительные, такие как личная информация пользователей, должны быть защищены во все время. Это может включать в себя шифрование данных, использование безопасных протоколов передачи данных и обеспечение безопасного хранения данных.

**3. Аудит и мониторинг**: всегда важно иметь возможность отслеживать и анализировать действия в вашем приложении. Это может помочь выявить подозрительную активность и быстро реагировать на возможные угрозы.

**4. Обработка ошибок и исключений:** неправильная обработка ошибок может привести к утечке информации, которая может быть использована злоумышленниками. Важно убедиться, что ваше приложение корректно обрабатывает ошибки и исключения, не раскрывая лишней информации.

**5. Обновления и патчи:** уязвимости и ошибки безопасности могут появляться в любой системе. Поэтому важно регулярно обновлять и применять патчи к вашему приложению и его зависимостям. Все эти принципы вместе обеспечивают многоуровневую защиту для вашего приложения и помогают снизить риск нарушения безопасности.

Spring Security – это фреймворк безопасности для Java-приложений. Это как высокотехнологичный охранник, который следит за входами и выходами в ваш замок (ваше приложение) и следит за тем, чтобы все гости (пользователи) вели себя прилично.

Основные компоненты Spring Security - это:

1. Аутентификация: как мы обсуждали ранее, аутентификация - это процесс проверки личности пользователя. Spring Security предоставляет широкий спектр механизмов аутентификации, таких как форма входа, LDAP, OAuth и другие.

2. Авторизация: после того, как пользователь аутентифицирован, Spring Security контролирует доступ к различным частям приложения. Это можно настроить на уровне URL или методов, обеспечивая гибкий и мощный механизм контроля доступа.

3. Защита от атак: Spring Security также обеспечивает защиту от самых распространенных типов атак, таких как CSRF (межсайтовая подделка запроса), XSS (межсайтовый скриптинг) и Session Fixation.

4. Шифрование паролей: Spring Security поддерживает различные механизмы шифрования паролей, что помогает обеспечить безопасное хранение и обработку паролей. Все эти компоненты вместе образуют мощный инструмент для обеспечения безопасности вашего серверного приложения. По сути, Spring Security управляет всей политикой безопасности приложения, обеспечивая защиту от несанкционированного доступа и атак.

**JSON Web Token (JWT).**

**JWT** – это стандарт, который определяет способ безопасной передачи информации между двумя сторонами в виде JSON-объекта. Эта информация может быть подтверждена и доверена, потому что она цифрово подписана. В Spring Security JWT обычно используется вместе с OAuth 2.0 для аутентификации и авторизации.

Схема обычно такова:

1. Пользователь аутентифицируется с помощью своих учетных данных.

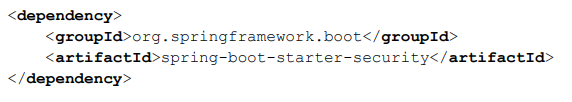
2. После успешной аутентификации сервер создает JWT, который содержит уникальные данные пользователя, и отправляет его обратно пользователю.

3. Пользователь сохраняет этот JWT и отправляет его в заголовке Authorization с каждым последующим запросом.

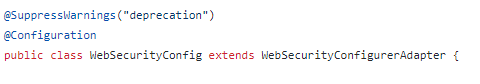
4. Сервер проверяет JWT в каждом запросе, чтобы аутентифицировать пользователя и определить его права доступа. В целом, JWT предоставляет нам способ безопасного обмена информацией между сервером и клиентом, что является важной частью безопасности серверного приложения.

**Настройка Spring Security.**

Шаг 1: Добавление зависимости в POM.xml



Шаг 2: Создание класса конфигурации безопасности. Этот класс должен расширять WebSecurityConfigurerAdapter.



## 9. Spring AOP.

Термины:

**Aspect** – модуль, который определяет Advices, срезы и привязки. Это кросс-функциональные заботы, такие как логирование, аудит, безопасность и т. д.

**Join Point** – точка в программе, такая как выполнение метода, где можно применить Advices (advice). В Spring AOP, join points представляют собой выполнение методов.

**Advice** – действие, предпринимаемое аспектом в определенной точке соединения. Существуют разные типы Advices, такие как before, after, after-returning, after-throwing, и around.

**Pointcut** – выражение, которое выбирает определенные join points. Advices применяются к join points, выбранным через pointcuts.

**Target** – объект, к которому применяется Advices.

**Proxy** – объект, созданный после применения Advicesа к целевому объекту.

**Weaving** – процесс комбинирования аспектов с другим типом приложения для создания прокси-объекта. Это может быть выполнено во время компиляции (CTW), загрузки класса (LTW) или во время выполнения.

**Introduction (Inter-type declaration)** – добавление новых методов или свойств в существующие классы. Транзакция – последовательность действий, которые либо полностью выполняются, либо полностью отменяются.

**ACID** – принципы транзакций — Атомарность, Согласованность, Изоляция и Долговечность (Atomicity, Consistency, Isolation, Durability).

**Propagation** – определяет, как транзакции относятся к друг другу. Например, REQUIRED, REQUIRES\_NEW, SUPPORTS и т. д.

**Isolation** – уровень изоляции транзакции определяет, как данные, доступные одной транзакции, становятся видимыми для других.

**@Transactional** – аннотация Spring для объявления транзакционного метода.

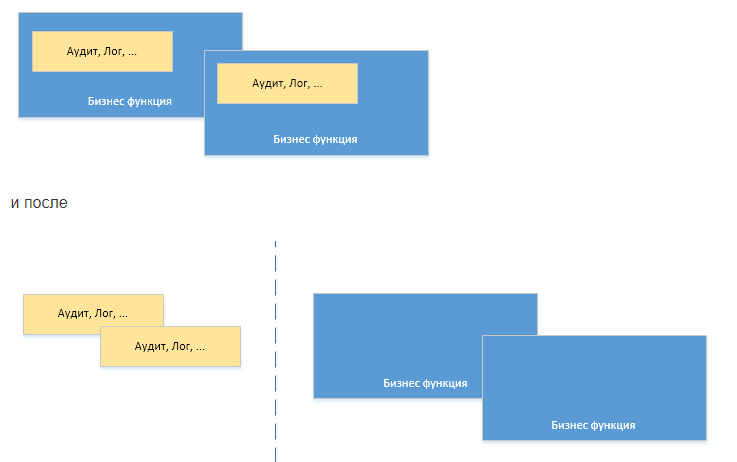
**Transaction Manager** – компонент, который управляет транзакциями. Например, DataSourceTransactionManager для JDBC.

**Rollback** – отмена изменений, выполненных в рамках транзакции.

**AOP** это сокращение от **Aspect-Oriented Programming**. Аспектно-ориентированное программирование (АОП) — это парадигма программирования являющейся дальнейшим развитием процедурного и объектно-ориентированного программирования (ООП). Идея АОП заключается в выделении так называемой сквозной функциональности.

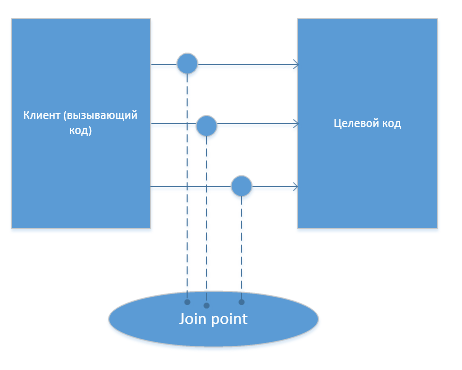
Теперь более подробно:

Выделение сквозной функциональности



Т.е. есть функциональность, которая затрагивает несколько модулей, но она не имеет прямого отношения к бизнес коду, и ее хорошо бы вынести в отдельное место, это и показано на рисунке выше.

**Join point.**

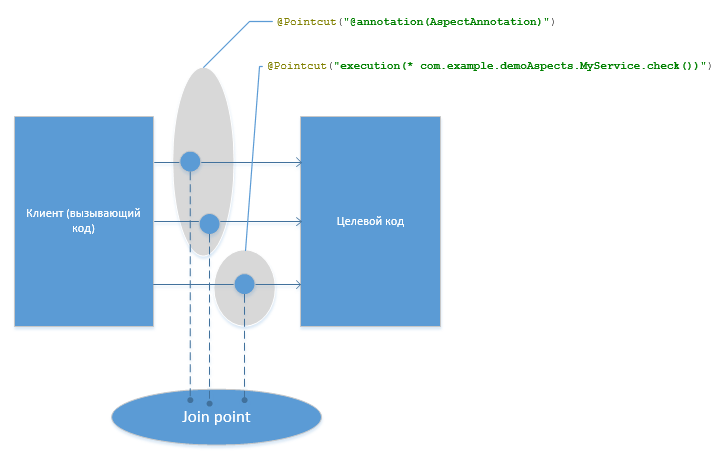


**Join point** — следующее понятие АОП, это точки наблюдения, присоединения к коду, где планируется введение функциональности.

Join Point в Spring AOP — это место в программе, где аспект может быть применен. Это может быть при вызове метода, при обработке исключения, при инициализации объекта и так далее. Join Points обозначают конкретные моменты в выполнении программы, когда можно “вмешаться” с помощью аспектов.

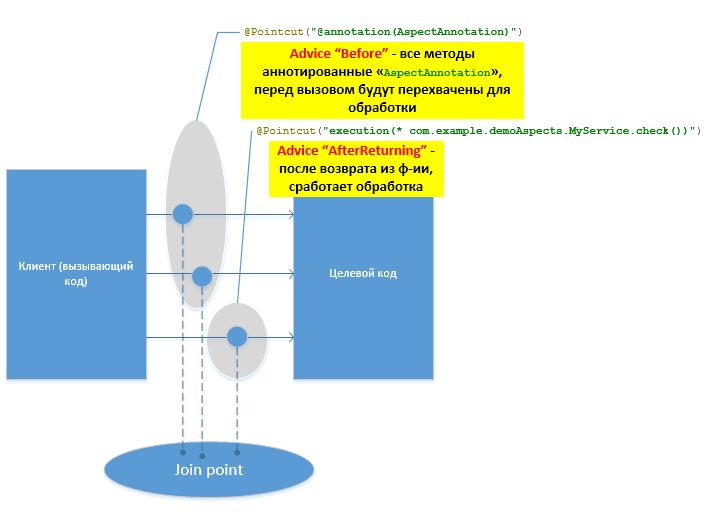
Представьте, что приложение — это книга. Каждая глава книги — это определенный функционал вашего приложения, а каждый абзац — это метод. Join Points в этой аналогии будут ключевыми словами или фразами, которые вы хотите выделить или комментировать. Когда вы найдете такое ключевое слово (Join Point), вы можете добавить примечание или комментарий (Advice) в этом месте. Join Points — это основа AOP. Они указывают, где и когда ваши аспекты будут применяться, давая вам возможность модифицировать или улучшать функциональность вашего приложения.

**Pointcut**



**Pointcut** — это срез, запрос точек присоединения, — это может быть одна и более точек. Правила запросов точек очень разнообразные, на рисунке выше, запрос по аннотации на методе и конкретный метод. Правила можно объединять по &&, ||,!

**Advice**

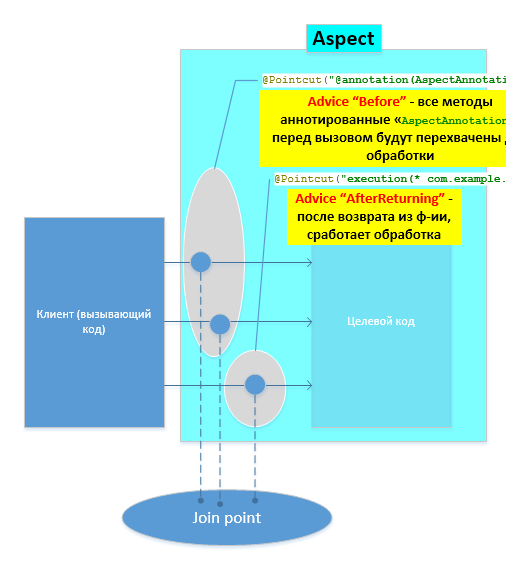
****

**Advice** — набор инструкций выполняемых на точках среза (Pointcut). Инструкции можно выполнять по событию разных типов:

* **Before** — перед вызовом метода;
* **After** — после вызова метода;
* **After returning** — после возврата значения из функции;
* **After throwing** — в случае exception;
* **After finally** — в случае выполнения блока finally;
* **Around** — можно сделать пред., пост., обработку перед вызовом метода, а также вообще обойти вызов метода.

На один Pointcut можно «повесить» несколько Advice разного типа.

**Aspect**



**Aspect** — модуль в котором собраны описания Pointcut и Advice.

**Почему важно использовать АОР?**

В первую очередь, Spring AOP помогает разработчикам писать более чистый и модульный код. Spring AOP позволяет сосредоточиться на основной логике приложения, не беспокоясь о дополнительных функциях, таких как логирование или безопасность. Эти “перекрестные” задачи (cross-cutting concerns) легко управляются с помощью аспектов в AOP.

В крупных компаниях, где десятки, если не сотни разработчиков работают над сложными и масштабными проектами, управление такими “перекрестными” задачами может стать настоящим вызовом. И здесь Spring AOP может стать настоящим спасением.

Как пример, в компании решили изменить формат логирования. Если не используется AOP, то придется искать и менять код логирования в сотнях мест. Но с AOP просто обновляется один аспект, и изменения автоматически применяются во всем проекте.

**Introductions.**

**Introduction (или “Mixin”)** позволяет добавлять новые методы или свойства в существующие бины. Другими словами, с помощью introductions вы можете добавить новую функциональность к уже существующим объектам в вашем приложении без изменения их кода.

**Around-Advices.**

**Around-Advice** — это, пожалуй, самый мощный тип advice. Он объединяет в себе все остальные типы advices, так как позволяет вам вмешиваться в вызов метода до его выполнения, после него и даже изменять возвращаемое **значение или кидать исключение вместо целевого метода.**

**Around-Advice** предоставляет максимальный контроль над тем, как метод будет выполняться, что делает его очень мощным инструментом в арсенале Spring AOP.

**Транзакции и AOP.**

**Транзакции** - это механизмы, которые позволяют группировать несколько операций в одну целостную единицу работы. Если все операции проходят успешно, изменения сохраняются. Если хотя бы одна из операций завершается ошибкой, все изменения отменяются.

Благодаря Spring и AOP этот процесс становится намного проще. Основы управления транзакциями в Spring с использованием AOP Spring предоставляет декларативное управление транзакциями, что означает, что можно управлять ими без написания специфического кода для этого. Вместо этого просто “объявляем” методы, которые должны выполняться в транзакции.

AOP обертывает вызов метода, делая его транзакционным. Если что-то идет не так внутри метода, AOP гарантирует, что транзакция откатывается.

@Transactional аннотация и её настройка Самый простой способ объявить метод транзакционным в Spring - это использовать аннотацию @Transactional. Эта аннотация говорит Spring, что метод должен выполняться в контексте транзакции.

**Перехватчики (Interceptors).**

Под “перехватчиками” в Spring называются механизмы, которые позволяют “перехватывать” входящие и исходящие сообщения или запросы, обычно в контексте веб-приложений. Они часто используются для таких задач, как логирование, безопасность, производительность и другие перекрестные задачи.

Advices, с другой стороны, больше связаны с AOP и обеспечивают перекрестную функциональность на уровне метода. В простых словах: Advices — это действия, которые выполняются до, после или вокруг метода.

Перехватчики — это механизмы, которые “перехватывают” запросы, перед тем как они достигнут их назначения (например, контроллера в веб-приложении).

Таким образом, перехватчики – это мощный инструмент для управления входящими и исходящими запросами в вашем веб-приложении, в то время как Advices AOP позволяют управлять поведением методов в вашем приложении. Оба эти механизма являются чрезвычайно полезными для реализации перекрестной функциональности.

**Плюсы AOP при разработке:**

1. Повторное использование кода и модульность.

При использовании Spring AOP можно выделить общие аспекты и переиспользовать их по всему приложению. Это особенно заметно, когда речь идет о логировании, безопасности, транзакциях и других перекрестных задачах. Вместо того чтобы вставлять один и тот же код в разные части приложения, создается один аспект и Spring понимает, где и когда его применять.

2. Улучшение читаемости и поддержки кода.

Без AOP код может быть перегружен логикой, не относящейся напрямую к основной функциональности метода. С AOP можно вынести эту “побочную” логику в отдельные аспекты, делая основной код чище и проще для понимания.

3. Гибкая настройка.

Потрясающая гибкость в определении, когда и где применять аспекты. Это значит, что можно быстро и легко изменять поведение нашего приложения без необходимости переписывать основной код.

4. Обработка исключений.

С помощью AOP можно легко управлять исключениями на уровне аспектов. Это дает возможность обрабатывать ошибки централизованно, упрощая отладку и предоставляя пользователям более информативные сообщения об ошибках.

5. Безупречные транзакции.

Тема транзакций и AOP показала, как просто управлять транзакциями, определяя их границы с помощью аннотаций. Это делает код проще и устойчивым к ошибкам.

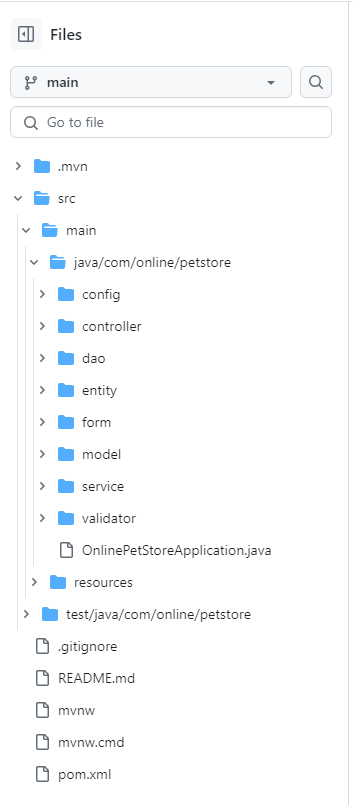
6. Понимание проксирования.

С помощью погружения в прокси и AOP Proxy стало понятно, как Spring создает прокси для реализации аспектов. Это знание крайне полезно при оптимизации и отладке приложений.

В заключение можно сказать: Spring AOP - это мощный инструмент, который помогает управлять повторяющимися и “перекрестными” задачами в вашем приложении. В крупных коммерческих проектах это может существенно ускорить разработку, упростить поддержку кода и улучшить качество продукта.

## 10. Реализация проекта.

Для создания web-приложения была использована следующая структура проекта.



Пакет **config** включающий в себя классы: WebConfiguration, WebSecurityConfig.

Пакет [**controlle**r](https://github.com/Daxrajsinh/Pet-Mart_SPRING-BOOT/tree/main/src/main/java/com/online/petstore/controller) включающий в себя классы: AdminController, MainController.

Пакет [**dao**](https://github.com/Daxrajsinh/Pet-Mart_SPRING-BOOT/tree/main/src/main/java/com/online/petstore/dao) включающий в себя классы: AccountDAO, OrderDAO, ProductDAO.

Пакет **entity** включающий в себя классы: Account, Customer, Order, OrderDetail, Product.

Пакет **form** включающий в себя классы: CustomerForm, ProductForm, UserForm.

Пакет **model** включающий в себя классы: CartInfo, CartLineInfo, CustomerInfo, OrderDetailInfo, OrderInfo, ProductInfo.

Пакет **service** включающий в себя классы: CartService, CustomerService, ListResultService, ProductService, UserDetailsServiceImpl.

Пакет **validator** включающий в себя классы: CustomerFormValidator, ProductFormValidator, UserFormValidator.

Пакет [**templates**](https://github.com/Daxrajsinh/Pet-Mart_SPRING-BOOT/tree/main/src/main/resources/templates) включающий в себя файлы html для отражения страниц сайта в браузере: header.html, index.html, login.html, menu.html, product.html, productList.html, shoppingCart.html, shoppingCartConfirmation.html, shoppingCartCustomer.html, shoppingCartFinalize.html, signup.html.

Файл **application.properties** с конфигурацией подключения к базе данных.

Файл **POM.xml** содержащий в себе все необходимые зависимости и плагины проекта.

А также главный класс всего приложения – **OnlinePetStoreApplication**, содержащий в себе бины: dataSource, sessionFactory, transactionManager для запуска проекта.

**Теперь немного подробнее про реализацию классов.**

**Config.**

В пакете **сonfig** реализованы классы: WebSecurityConfig, WebConfiguration.

В проекте добавлен класс WebSecurityConfig для реализации безопасности в приложении.

Использованные аннотации:

**@Bean** – это аннотация уровня метода и прямой аналог элемента <bean/> на XML. Аннотация поддерживает некоторые атрибуты, предлагаемые <bean/>, такие как: init-method, destroy-method, autowiring, name. Аннотацию **@Bean** можно использовать в классе с аннотацией **@Configuration** или в классе с аннотацией **@Component**. Объявление бина. Чтобы объявить бин, можно аннотировать метод с помощью **@Bean**. Этот метод используется для регистрации определения бина в ApplicationContext того типа, который задан в качестве возвращаемого значения метода. По умолчанию имя бина совпадает с именем метода.

**@SuppressWarnings("deprecation")** - аннотация использованная для отключения предупреждений, связанных с устареванием.

**@Configuration** - аннотация на уровне класса, указывающая на то, что объект является источником определений бина.

**@Autowired** – аннотация используется для автоматического связывания компонентов бина между собой. Она позволяет автоматически настраивать свойства бина и методы, упрощая тем самым процесс инъекции зависимостей.

**Controller.**

В пакете **controller** реализованы классы контроллеры:

В классе **MainController** реализованы различные ручки для возможности работы посетителями сайта в интернет-магазине.

В классе **AdminController** реализованы различные ручки для возможности работы администраторам сайта с интернет-магазином.

Использованные аннотации:

**@Controller** - аннотация используется для определения классов в качестве контроллеров в Spring MVC. Это помогает Spring определить компоненты, которые будут обрабатывать входящие запросы.

**@Transactional** – аннотация Spring для объявления транзакционного метода.

**@Autowired** – аннотация используется для автоматического связывания компонентов бина между собой. Она позволяет автоматически настраивать свойства бина и методы, упрощая тем самым процесс инъекции зависимостей.

**@RequestMapping**  — это многофункциональная аннотация, предоставляющая весьма богатый набор возможностей. **@RequestMapping**, используемая на уровне класса, позволяет работать не только с GET-запросами, но и с другими HTTP-методами, обеспечивая вас гибкостью для будущих изменений.

**@GetMapping** — это специализированная аннотация **@RequestMapping**, предусмотренная конкретно для обработки HTTP GET-запросов, что придаёт коду лаконичность и читаемость.

**@PostMapping** - используется для модификации данных.

**DAO.**

В пакете **dao** реализованы классы: AccountDAO, OrderDAO, ProductDAO. DAO - Поддержка объекта доступа к данным (Data Access Object/DAO) в Spring направлена на упрощение работы с технологиями доступа к данным (такими как JDBC, Hibernate или JPA).

Использованные аннотации:

**@Transactional** – аннотация Spring для объявления транзакционного метода.

**@Repository** – это маркер для любого класса, который исполняет роль или стереотип репозитория (также известного как объект доступа к данным или DAO). Среди способов использования этого маркера - автоматическое преобразование исключений.

**Entity.**

В пакете **entity** собраны сущности используемые в проекте.

**Account** класс для реализации сущности аккаунтов администратора и посетителей сайта. Аналогично в классах **Order, OrderDetail, Product, Customer** использован для реализации сущностей Заказа, Деталей Заказа, Продукта, Покупателя соответственно. Классы включаютв себя: объекты, конструктор, геттеры и сеттеры.

В классах использованы аннотации:

**@Entity** – Аннотация, используемая для обозначения класса в JPA как сущности, что означает, что этот класс может быть отображен на таблицу в базе данных.

**@Table(name = "Accounts")** - указывает, что эта модель считается сущностью, связанной с таблицей Accounts в базе данных.

**@Id** – Аннотация, используемая для обозначения первичного ключа сущности.

**@Column(name = "User\_Name", length = 20, nullable = fals**e) — указывает на имя и свойства колонки.

**@ManyToOne(fetch = FetchType.LAZY)** - аннотация используемая для созданя связи таблиц по стратегии многие к одному.

**@JoinColumn** — применяется когда внешний ключ находится в одной из сущностей. Может применяться с обеих сторон взаимосвязи. Но рекомендуется применять в сущности, которая является владельцем физической информации (**обычно** **сторона @ManyToOne**).

**@Temporal** — применяется к полям или свойствам с типом java.util.Date и java.util.Calendar. Например, если в БД время сохраняется как sql.Date, то чтобы использовать дату из java.util.Date указываем эту аннотацию.

**Form.**

**Классы CustomerForm**, ProductForm, UserForm представленные в пакете **form** содержат в себе поля, конструкторы, геттеры и сеттеры. И служат для описания таких сущностей как Покупатель, Продукт, и Пользователь соответственно.

**Model.**

В пакете **model** представлены классы: CartInfo, CartLineInfo, CustomerInfo, OrderDetailInfo, OrderInfo, ProductInfo. Содержат в себе поля, конструкторы, геттеры и сеттеры. И служат для описания информации о Корзине, Покупателе, Заказе, Продукте.

**Service.**

В пакете **service** представлены классы: CartService, CustomerService, ListResultService, ProductService, UserDetailsServiceImpl. Отвечающие за сервисный слой работы приложения.

В классах использованы аннотации:

**@Autowired** - аннотация в Spring Framework используется для автоматического связывания компонентов бина между собой. Она позволяет автоматически настраивать свойства бина и методы, упрощая тем самым процесс инъекции зависимостей.

**@Service** - аннотация используется для обозначения классов в слое сервисов. Это особый случай **@Component**, который имеет больший семантический вес. Использование **@Service** вместо **@Component** может улучшить читаемость кода, указывая, что класс служит определенной бизнес-логике. Аннотация **@Component** является общей аннотацией, которая указывает, что класс является частью компонентов приложения. Она не предоставляет никакой дополнительной функциональности, и ее можно считать «по умолчанию» для любого Spring-управляемого компонента.

**Validator.**

Немного теории:

Пользователи часто передают в приложение некорректные данные. Такое происходит либо из злого умысла, либо по ошибке. Стоит проверять данные на соответствие бизнес-требованиям.

Эти бизнес-правила влияют на каждый уровень приложения. Веб-интерфейс сообщает пользователю подробные и локализованные сообщения об ошибках. Уровни бизнес-логики и хранения должны проверять приходящие от клиентов значения, перед отправкой в хранилище. База данных SQL делает окончательную проверку, чтобы гарантировать целостность хранимой информации.

Эти задачи поможет решить **Bean Validation**. Он интегрирован со **Spring** и **Spring Boot. Hibernate Validator** считается эталонной реализацией **Bean Validation**.

Идея **Bean Validation** в том, чтобы определять такие правила, как «Это поле не может быть null» или «Это число должно находиться в заданном диапазоне» с помощью аннотаций. Это гораздо проще, чем постоянно писать условные операторы проверок.

**Hibernate Validator** также задаёт правила валидации с помощью аннотаций над полями класса. Этот декларативный подход не загрязняет код. При передаче размеченного таким образом объекта класса в валидатор, происходит проверка на ограничения.

**Spring Validation** — это вторичная инкапсуляция **Hibernate Validation**, используемая для поддержки автоматической валидации параметров **Spring MVC**.

В пакете **validator** представлены классы: CustomerFormValidator, ProductFormValidator, UserFormValidator (валидация покупателя, валидация продукта, валидация пользователя).

# Заключение

В рамках дипломной работы был реализован проект интернет-магазина по продаже домашних животных, создана модель web-сайта, согласно выданного технического задания на разработку. Произведена установка и настройка необходимого для реализации программногоо обеспечения. А именно: среда разработки – Intellij IDEA, контейнеризация - Docker, база данных проектировалась в среде PostgreSQL. В своей работе использовал фреймворк spring boot. Использованы инструменты: Java, Java Spring, Git, Docker, ProgreSQL. Разработана автоматизированная информационная система.

Web-сайт ориентирован на всех пользователей Интернета. С его помощью потенциальные покупатели смогут: зарегистрироваться и пройти авторизацию на сайте, ознакомиться с представленными товарами и приобрести их, сформировать лист желаний и список заказов. Администраторы сайта имеют возможность выполнять операции над товарами. Разработанная автоматизированная система «Интернет магазина» позволяет существенно повысить качество и производительность работы магазина.

Интернет - магазин был развернут, запущен и протестирован локально. Полученные результаты соответствовали ожидаемым. Были сделаны проверки возможных ошибок, недочетов - система функционирует надежно, без сбоев. Разработанный сайт удовлетворяет всем требованиям, поставленным техническом задании.

Практическая значимость для меня как для студента является достаточно весомой, так как была использована возможность на практике применить полученные знания за все время обучения. Для реализации проекта мною были повторены и более глубоко изучены не только темы, пройденные за время обучения, но и дополнительная литература и статьи указанные в разделе «список используемой литературы».

В рамках дипломной работы по техническому заданию был реализован минимальный функционал работы сайта. В дальнейшем интернет магазин можно расширить следующим функционалом:

расширить категории товаров (продавать сопутствующие товары для животных);

реализовать онлайн-чат между клиентом и продавцом;

добавить сервис доставки;

реализовать возможность онлайн оплаты товаров.

# Список используемой литературы

**Книги:**

Изучаем Java . Сьерра Кэти, Бэйтс Берт. Эксмо. 2023 г.;

Чистый код. Мартин Роберт. Питер. 2022 г.;

Java в облаке. Spring Boot, Spring Cloud, Cloud Foundry. Джош Лонг и Кенни Бастани. Питер. 2019 г.;

Spring Boot в действии. Крейг Уоллс. ДМК Пресс. 2022 г.;

Spring Microservices в действии. Джон Карнелл. ДМК Пресс. 2021 г..

**Интернет-сайты:**

**Docker:** статья «Запуск контейнеров» (<https://docs-docker-com.translate.goog/engine/reference/run/?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=sc>);

**Habr.com:** использован для поиска информации и исправления ошибок при написании кода (https://habr.com/ru)

**Javarush:** использован для поиска информации и исправления ошибок при написании кода (<https://javarush.com/forum>);

**MVNrepository:** использован для поиска и добавления плагинов и зависимостей проекта (<https://mvnrepository.com/>);

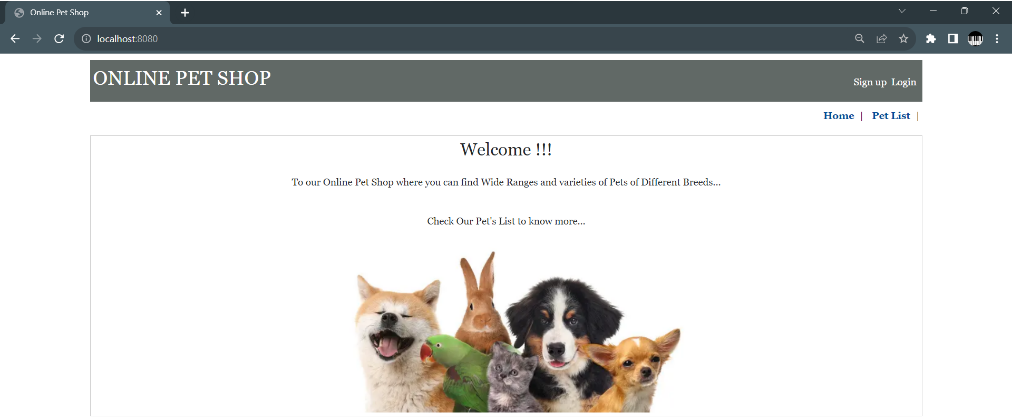
**Stackoverflow:** использован для поиска информации и исправления ошибок при написании кода **(**https://stackoverflow.com/**).**

# Приложения

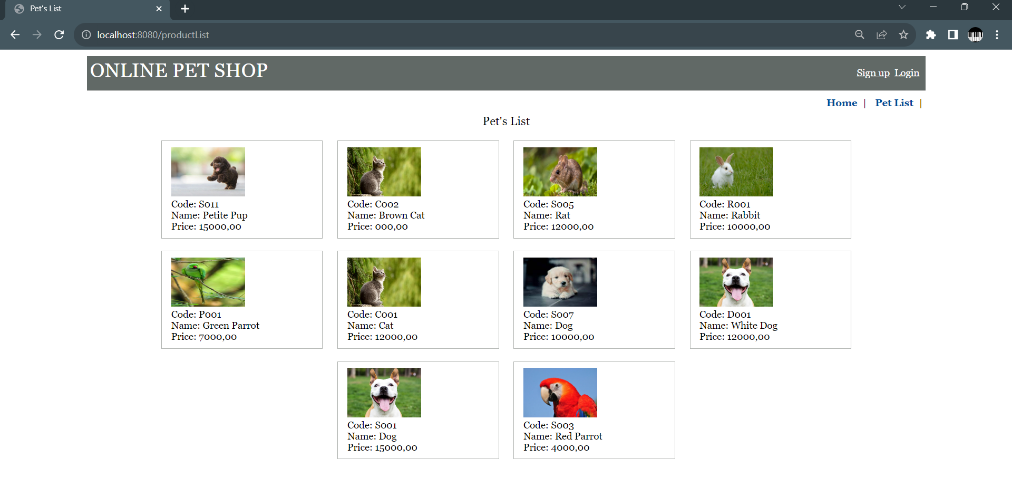
Скрины страниц интернет-магазина.

**Для посетителя сайта:**

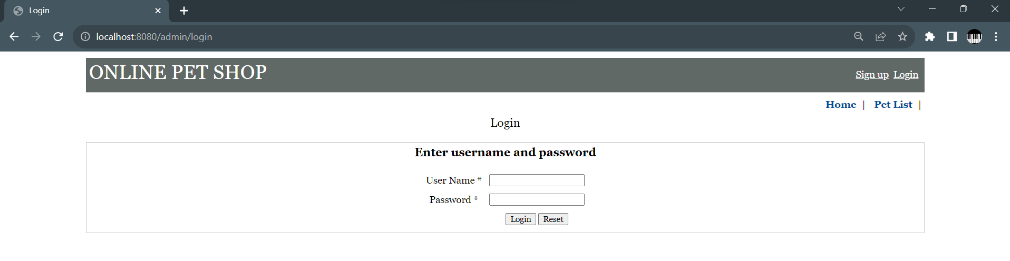
Домашняя страница сайта



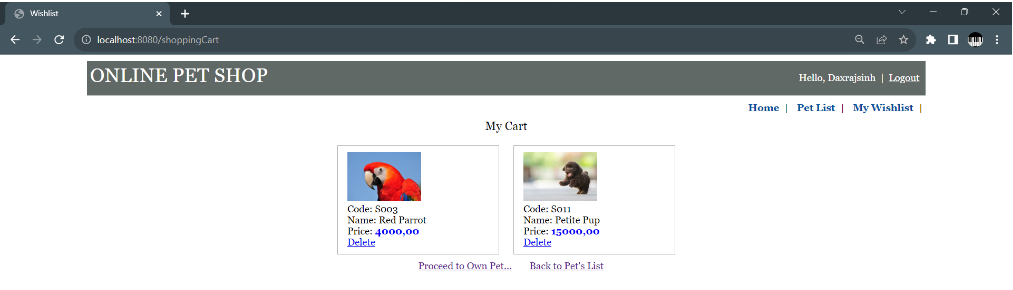
Страница со списком животных представленных на сайте



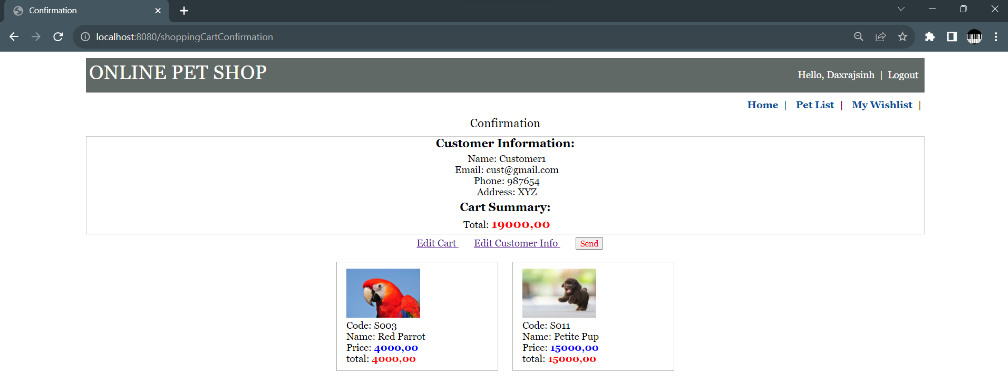
Страница авторизации посетителей сайта

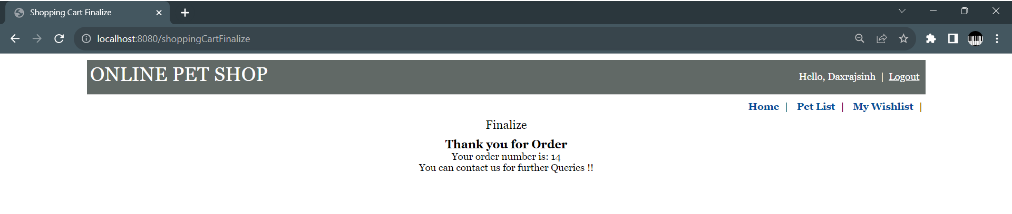


Страница с товарами, добавленными в список желаемых



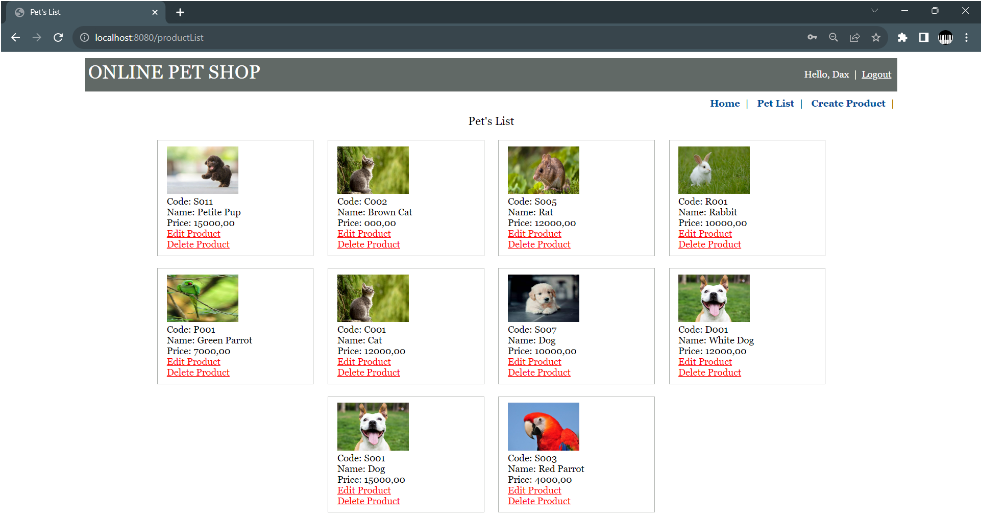
Страницы подтверждения заказа





**Для администратора сайта:**

Страница администратора сайта



Страница редактирования товаров

