Geekbrains

**Разработка веб приложения «EasyMarket» на JavaSpring.**

IT-специалист:

Разработка веб приложений

Грибенюк Д.М.

Москва

2024

**Дипломный проект**

**«Разработка веб приложения «Дневник чувств» на JavaSpring»**

**Содержание**

|  |  |
| --- | --- |
| Введение……………………………………………………………………. | 3 |
| Глава 1. Предпроектная стадия…………………………………………… | 5 |
| 1.1. Порядок проведения работ…………………………………………… | 5 |
| 1.2. Определение требований к приложению…………………………… | 6 |
| 1.3. Выбор стека технологий……………………………………………… | 7 |
| Глава 2. Проектная стадия………………………………………………… | 10 |
| 2.1. Проектирование общей архитектуры приложения………………… | 10 |
| 2.2. Проектирование web-интерфейса приложения…………………….. | 22 |
| 2.3. Проектирование базы данных……………………………………….. | 29 |
| 2.4. Безопасность приложения……………………………………………. | 37 |
| 2.5. Разворачивание Docker контейнера…………………………………. | 51 |
| Заключение………………………………………………………………… | 52 |
| Список использованной литературы…………………………………….. | 53 |

**Введение**

Актуальность темы исследования. С развитием интернет-технологий все больше компаний стремятся расширить свой бизнес за счет онлайн-продаж. В связи с этим разработка веб-приложений становится все более востребованной. В рамках данной работы было решено создать веб-приложение «EasyMarket», предназначенное для удобного и быстрого поиска и покупки товаров онлайн.  
Цель работы. Целью данной дипломной работы является разработка веб-приложения «EasyMarket» на JavaSpring, который позволит пользователям совершать покупки товаров различных категорий, производителей и ценовых категорий.

Задачи исследования. Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить основные принципы разработки веб-приложений на JavaSpring.

2. Разработать структуру и функционал веб-приложения «EasyMarket».

3. Создать пользовательский интерфейс, обеспечивающий удобство пользования приложением.

4. Разработать систему управления товарами, заказами и пользователями.

Для разработки веб-приложения был использован язык программирования Java с фреймворком Spring. Для создания пользовательского интерфейса были применены технологии HTML, CSS и JavaScript. Для управления базой данных была использована технология PostgreSQL.  
Ожидаемые результаты. Ожидается создание полноценного функционального веб-приложения «EasyMarket», который будет способствовать увеличению объема продаж компании, облегчит процесс заказа и доставки товаров для потребителей.

**Глава 1. Предпроектная стадия**

**1.1. Порядок проведения работ**

В ходе создания проекта были проведены следующие виды работ:

1. **Планированиепроекта**:
   * Определение функциональных и нефункциональных требований к приложению.
   * Проектирование архитектуры приложения и выбор используемых технологий.
2. **Настройка окружения разработки**:
   * Установка и настройка JavaDevelopmentKit (JDK), ApacheMaven (для сборки проекта), и SpringBoot (для создания и запуска приложения).
3. **Создание проекта**:
   * Созданновый проект с использованием SpringInitializr.
   * Выбраны необходимые зависимости, такие как SpringWeb, SpringDataJPA, Lombok, SpringSecurity для поддержки функциональности приложения.
4. **Разработка модели данных, репозитория, сервисов, контроллера.**
5. **Разработка пользовательского интерфейса**:
   * Для создание пользовательского интерфейса был использован фреймворк Bootstrap для отображения данных и взаимодействия с пользователем.
6. **Отладка и оптимизация**:
   * Отладка кода для исправления ошибок и улучшения производительности.
   * Оптимизация кода и конфигурации приложения для улучшения скорости работы и эффективности использования ресурсов.
7. **Документирование**:
   * Создание документации по проекту, включая описание архитектуры, инструкции по установке и запуску, описание API (если применимо) и другие справочные материалы.

**1.2. Определение требований к приложению**

Для разработки приложения «EasyMarket» были определены следующие требования:

1. Функциональные требования:
   * Пользователи должны иметь возможность создавать лот для продажи товаров.
   * Пользователи должны иметь возможность просматривать и редактировать свои объявления.
2. Нефункциональные требования:
   * Удобство использования: Интерфейс должен быть интуитивно понятным и привлекательным для пользователя.
   * Безопасность: Данные пользователей должны храниться и передаваться в зашифрованном виде.
   * Производительность: Приложение должно обеспечивать быстрый доступ к данным и отзывчивый интерфейс.
   * Масштабируемость: Возможность расширения функциональности и увеличения числа пользователей без значительного ухудшения производительности.
3. Требования к интерфейсу пользователя:
   * Интуитивно понятный интерфейс, который позволяет пользователям легко создавать и просматривать записи о объявлениях.
4. Требования к безопасности:
   * Аутентификация и авторизация пользователей.
   * Защита данных пользователей от несанкционированного доступа.
5. Требования к производительности:
   * Быстрый доступ к данным и отзывчивый интерфейс приложения.
6. Требования к масштабируемости:
   * Возможность добавления новых функций и поддержка роста числа пользователей.
7. Требования к поддержке и обновлениям:
   * Поддержка приложения и обновления для улучшения функциональности и безопасности.

**1.3. Выбор стека технологий**

1. **Язык программирования**: Java для разработки серверной части приложения так как этот язык хорошо подходит для создания масштабируемых и надежных веб-приложений.
2. **Фреймворк для веб-разработки**: SpringBoot с модулями Spring MVC и SpringSecurity для быстрой и удобной разработки серверной части приложения. SpringSecurity обеспечит безопасность приложения, а Spring MVC позволит создать удобный интерфейс.
3. **База данных**: PostgreSQL для хранения данных пользователей и их эмоций. Обе системы хорошо масштабируются и обеспечивают надежное хранение данных.
4. **Хранение данных о товарах**: для хранения данных о объявлениях и данных пользователей используется **PostgreSQL.**
5. **Фронтенд:** для упрощения разработки интерфейса был использован фреймворк Bootstrap.
6. **Аутентификация и авторизация**: SpringSecurity используется для обеспечения безопасности приложения, включая аутентификацию пользователей и управление доступом к ресурсам.
7. **Среда разработки**: IntelliJ IDEA для разработки серверной части приложения. Этот стек технологий обеспечивает разработку приложения "Дневник Чувств" с учетом требований к функциональности, производительности и безопасности.

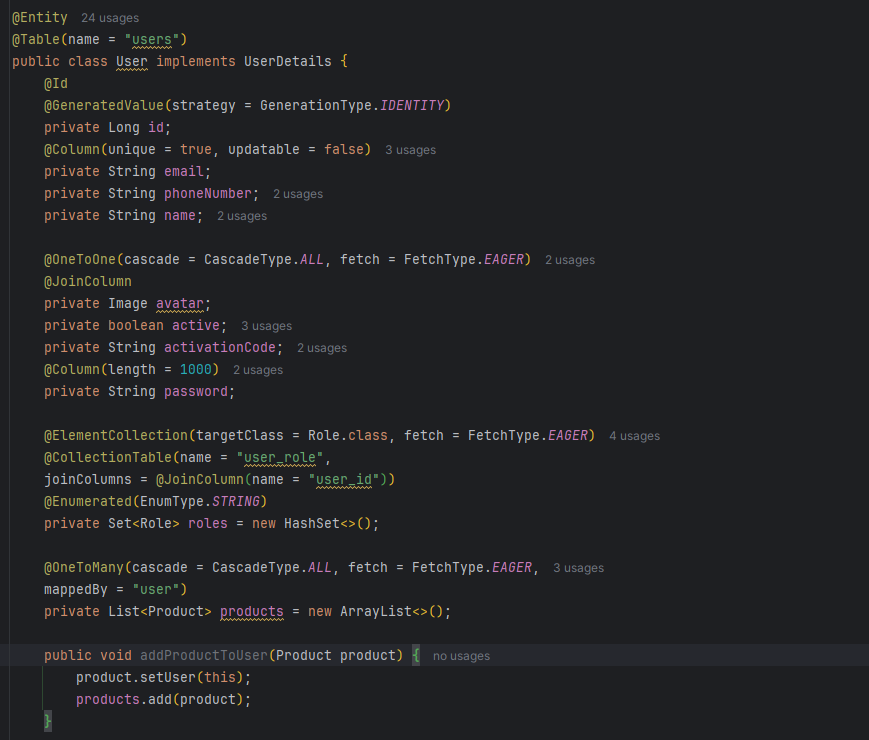
Архитектура приложения в рамках JavaSpring основывается на паттерне проектирования MVC (Model-View-Controller), который позволяет разделить приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Наше приложение состоит из данных составных частей:

1. **Модель (Model)**:
   * Модель представляет собой структуру данных приложения.
   * Включает классы, описывающие сущности приложения
2. **Представление (View)**:
   * Представление отвечает за отображение данных пользователю.
   * В веб-приложениях на JavaSpring представление часто реализуется с использованием шаблонизаторов, таких как Thymeleaf или JSP, которые позволяют вставлять данные из контроллера непосредственно в HTML-шаблоны.
3. **Контроллер (Controller)**:
   * Контроллер обрабатывает запросы от пользователя и взаимодействует с моделью и представлением.
   * В JavaSpring контроллеры представлены классами, аннотированными **@Controller** или **@RestController**.
   * Контроллеры содержат методы для обработки различных типов запросов (GET, POST, PUT, DELETE) и вызывают соответствующие сервисы для выполнения бизнес-логики.
4. **Репозиторий (Repository)**:
   * Репозиторий отвечает за доступ к данным из базы данных.
   * В JavaSpring репозитории обычно реализуются с помощью интерфейсов, расширяющих **JpaRepository**, которые предоставляют базовые методы для работы с данными.
   * Репозитории используются сервисами для получения, сохранения, обновления и удаления данных.
5. **Сервис (Service)**:
   * Сервис содержит бизнес-логику приложения.
   * Он обычно используется для выполнения операций с данными, координирования действий между различными компонентами приложения и обеспечения единого доступа к функциональности приложения.
6. **Сервер (Server)**:
   * Сервер представляет собой окружение, в котором работает приложение.
   * В JavaSpring приложение обычно запускается на встроенном сервере, таком как Tomcat или Jetty, который обрабатывает HTTP-запросы и управляет жизненным циклом приложения.

Каждый из этих компонентов играет важную роль в построении архитектуры приложения на JavaSpring, обеспечивая его модульность, гибкость и масштабируемость. Они позволяют разделить различные аспекты приложения и обеспечить их независимую разработку и тестирование.

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

Первое, с чего начиналась работа - это класс с инициализацией полей класса User. Класс User представляет собой создание новой учетной записи на для аутентификации, создания объявления и добавления контактных данных.



UserController является компонентом веб-приложения, отвечающим за взаимодействие с пользователем и обработку запросов, связанных с управлением аккаунта пользователя.

Контроллер предоставляет различные методы для обработки запросов пользователей и взаимодействия с соответствующими сервисами и использует сервисы UserService и для выполнения операций с данными.

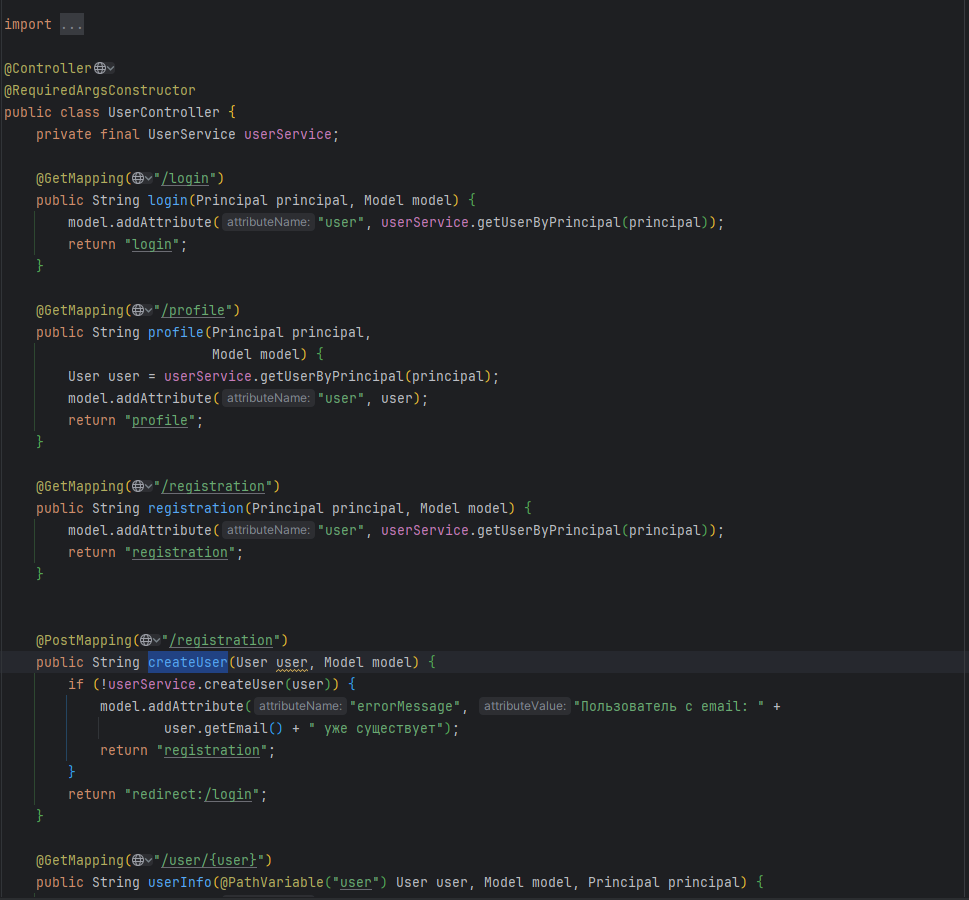
Методы контроллера позволяют пользователю регистрироваться, открывать профиль, изменять профиль. Кроме того, контроллер обеспечивает перенаправление пользователя на соответствующие страницы для отображения результатов операций.

Для обеспечения безопасности приложения контроллер также взаимодействует с модулем безопасности (например, SpringSecurity), обеспечивая аутентификацию и авторизацию пользователей при доступе к различным функциям приложения.

Методы контроллера обрабатывают различные HTTP-запросы, например, GET и POST, для отображения страниц, получения и обновления данных. Например, методы login и registration отображают формы для создания новых пользователей, а метод createUser обрабатывают отправленные данные и добавляют новые записи в базу данных.

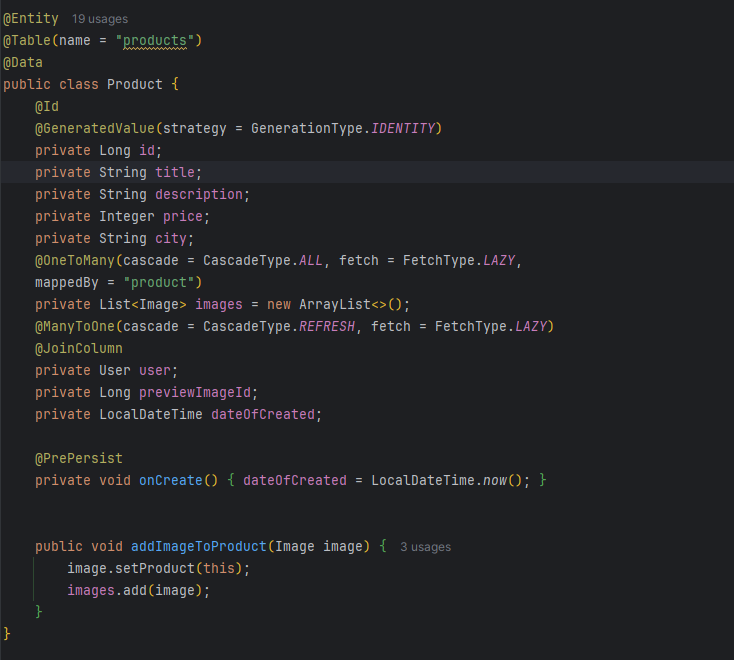
Эти методы позволяют пользователям управлять своими данными и вносить изменения в уже существующие записи.

В целом, UserController является важной частью приложения, обеспечивающей удобный интерфейс для работы с данными.



Класс Product представляет создание объявления, где пользователь может добавить название, описание, цену и город.

Такая модель данных позволяет пользователям систематизировать и анализировать объявления, что может помочь им а также эффективнее обявлением.



Класс UserService в данном контексте является сервисом, предоставляющим функционал для работы с пользователями. Давайте рассмотрим его подробнее:

**@Service**: Аннотация @Service является частью SpringFramework и используется для пометки классов, которые выполняют бизнес-логику, обычно связанную с обработкой данных.

**@AllArgsConstructor**: Это аннотация из библиотеки Lombok, которая генерирует конструктор, принимающий все поля класса в качестве аргументов. Этот конструктор обычно используется для внедрения зависимостей через конструктор.

**@Slf4j:** библиотека для протоколирования, ставящая своей целью предоставить максимально простой, но при этом мощный фасад для различных систем протоколирования на Java. SLF4J предоставляет простой обобщённый интерфейс для систем протоколирования, не зависящий от конкретной реализации.

**Private final UserRepository userRepository**: Поле userRepository является зависимостью класса UserService и предполагается, что Spring внедряет его экземпляр в этот класс через конструктор.

**Private final PasswordEncoder passwordEncoder**: Поле passwordEncoder также является зависимостью класса UserService и используется для кодирования паролей пользователей.

**Public booleanadd** **createUser (User user)**: Метод createUser принимает объект User в качестве аргумента и возвращает булево значение, указывающее на успешность операции добавления пользователя.

**String** **email = user.getUseremail ()**: Получение мейла пользователя из объекта user.

**if (userRepository.findByEmail(email)** **!= null) return false**: Проверка, существует ли уже пользователь с таким мейлом. Если пользователь уже существует, метод возвращает false, что означает, что операция добавления пользователя не выполнена.

**user.setActive(true):** Проверкана блокировку пользователя, возвращает булево значение, указывающее на статус учетной записи пользователя.

**user.setPassword(passwordEncoder.encode(user.getPassword()))**: Кодирование пароля пользователя с использованием passwordEncoder.

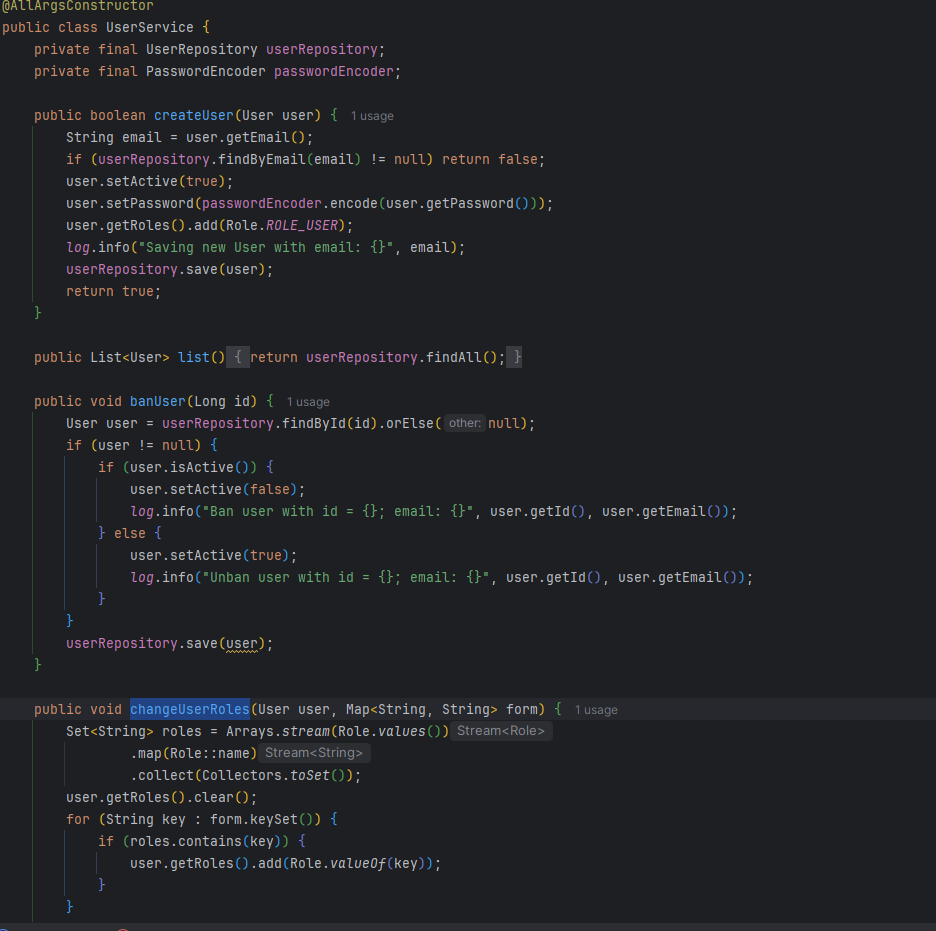
**user.getRoles.add(Role.ROLE\_USER)**:Установка роли пользователя. В данном случае, устанавливается роль "USER".

**log.info("Saving new User with email: {}", email)**: Лог выводимы в консоль о успешном сохранении пользователя

**userRepository.save(user)**: Сохранение пользователя в репозитории.

**returntrue**: Возвращение true, чтобы указать, что операция добавления пользователя выполнена успешно.

Класс UserService предоставляет метод createUser, который добавляет нового пользователя с уникальным именем, кодирует его пароль и устанавливает роль "USER". Если пользователь с таким именем уже существует, операция добавления завершается неудачей. Метод banUser создан для обработки функции блокировки пользователя, а метод changeUserRoles, для присваивания роли нашего пользователя.



Класс ProductService предоставляет функционал по работе товарами. Аннотация @Service говорит о том, что этот класс является сервисным компонентом, который предоставляет бизнес-логику для работы с данными. Аннотация @RequiredArgsConstructor генерирует конструктор с аргументами для всех полей класса, что позволяет Spring автоматически внедрить зависимости.

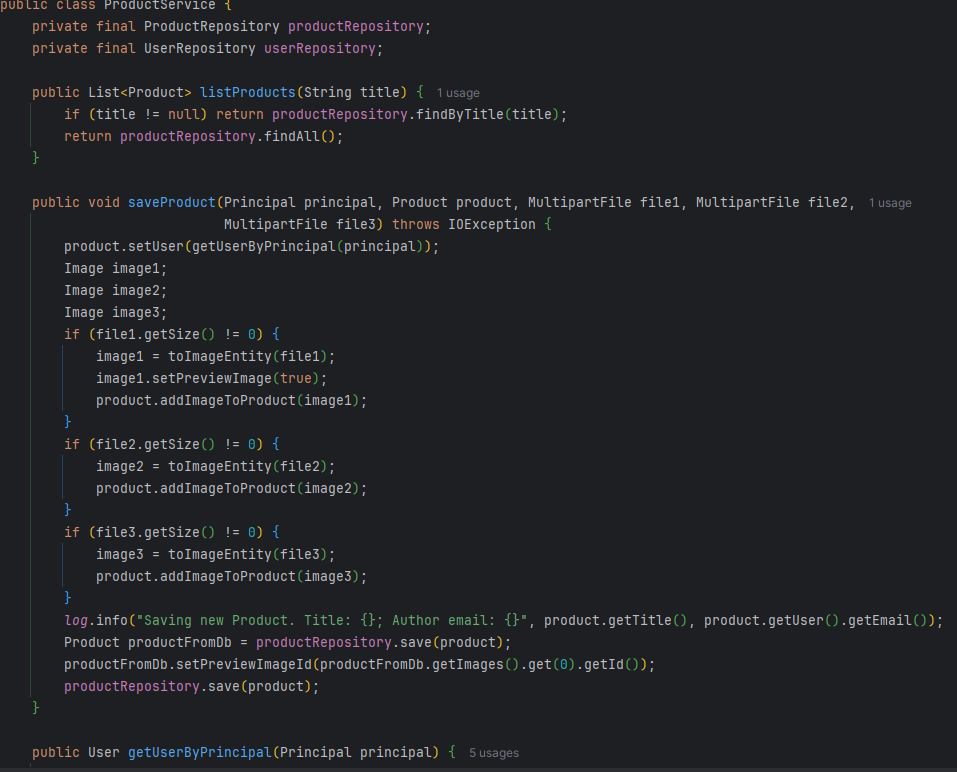
Метод listProducts выводи все созданные товары пользователем.

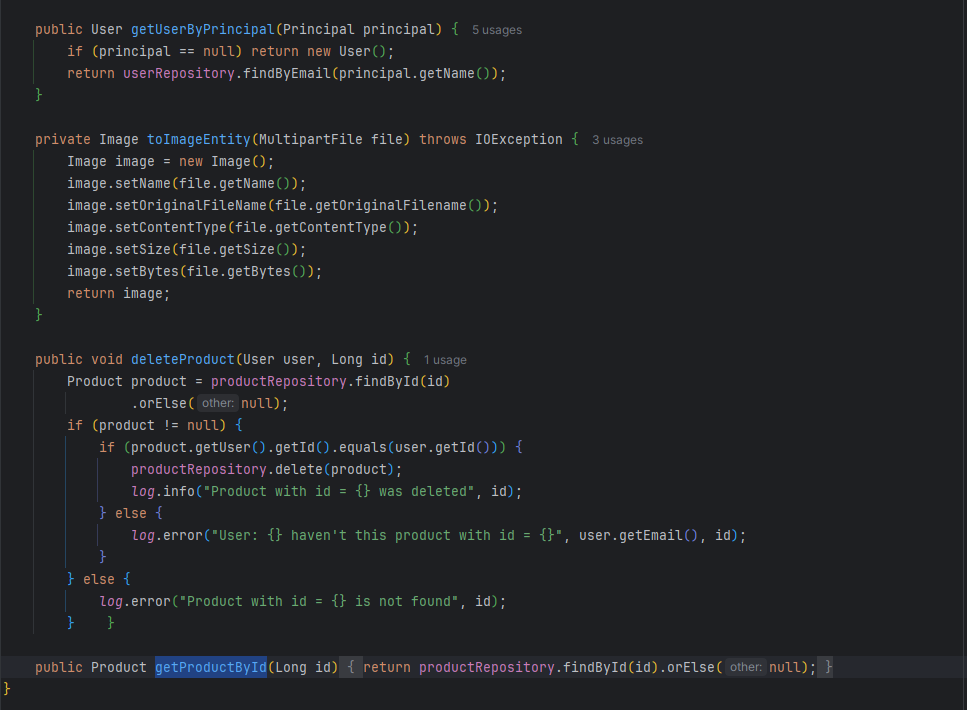
Метод saveProduct создает новое объявление прикрепляет к нему фото, после чего сохраняет заметку в репозитории.

Метод getUserByPrincipal возвращает пользователя по идентификатору. Если заметка не найдена, выбрасывается исключение.

Метод toImageEntity присваивает первое загруженное фото в качестве привью.

Метод deleteProduct удаляет товар по идентификатору.

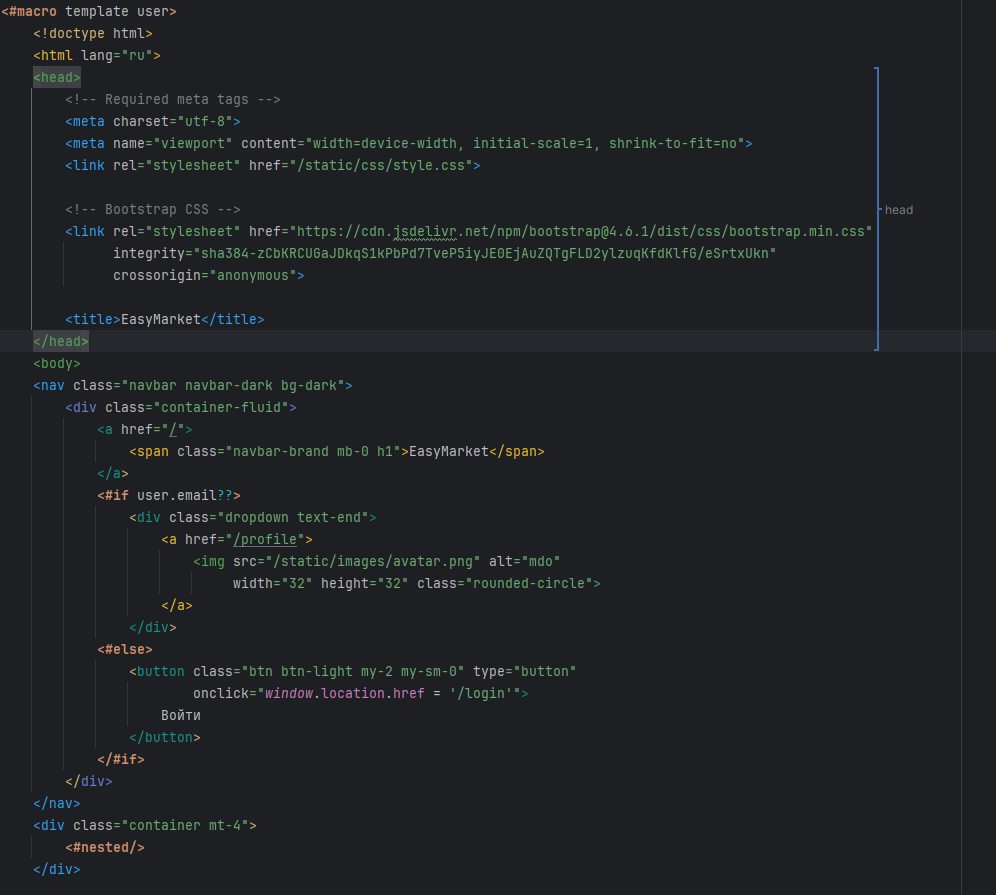
Метод getProductById возвращает товар по идентификатору из репозитория. 



.

**2.2. Проектирование WEB – интерфейса приложения**

Для проектирования WEB – интерфейса приложения был использован Bootstrap. Для подключения нашего Bootstrap мы используем template.ftlh.

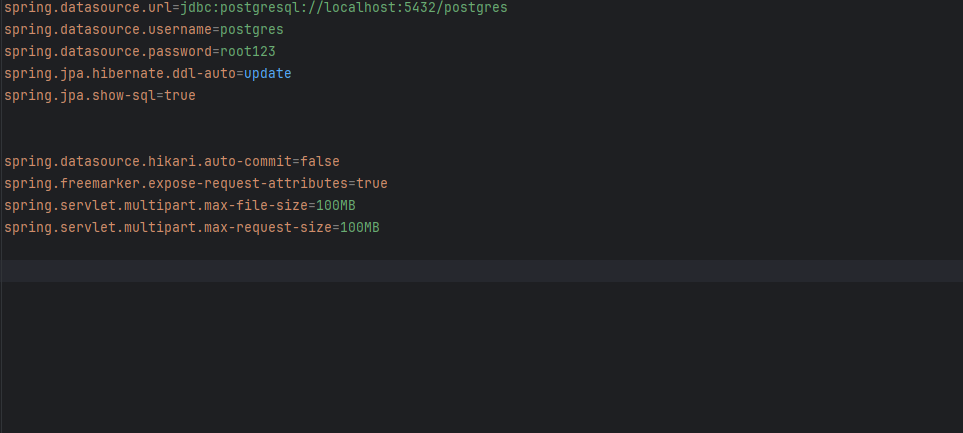


Для управления приложением мы используем такие шаблоны admin, login,

my-products, product-info, products, profile, registration, user-edit, user-info.ftlh.

**2.3. Проектирование базы данных**

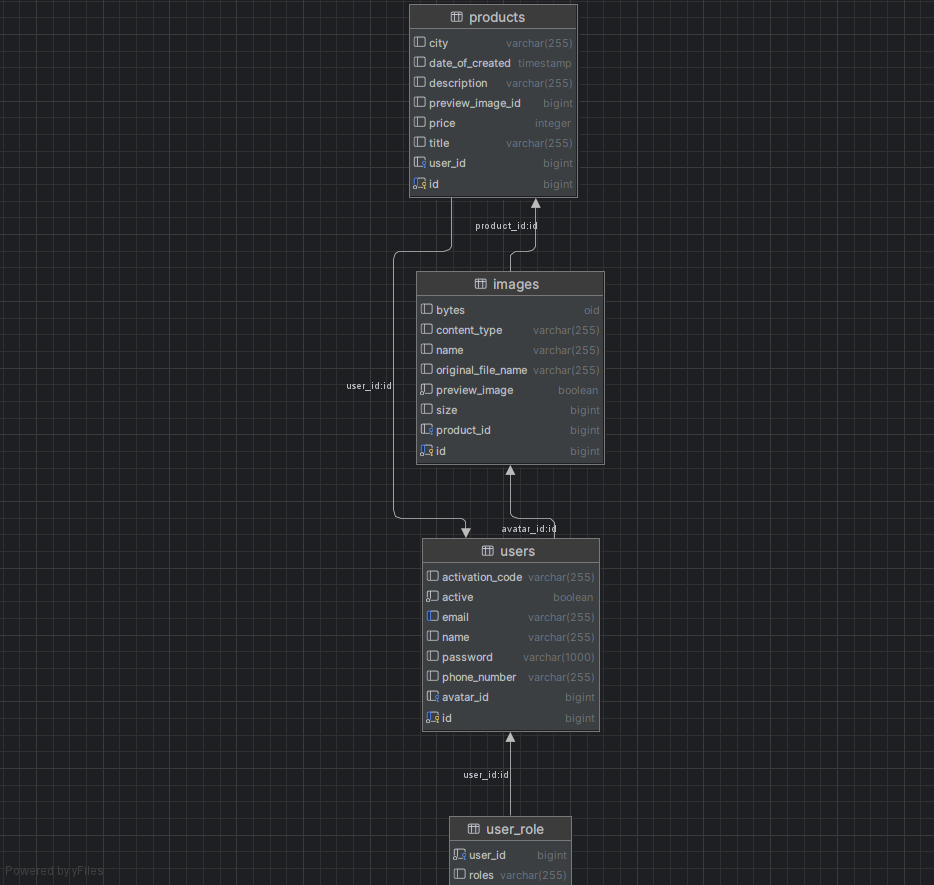
В качестве базы данных для этого проекта был выбран PostgreSQL. В файле application.properties указаны следующие настройки.



В данном примере конфигурации SpringBoot для подключения к базе данных PostgreSQL через JPA и Hibernate используется следующая конфигурация:

1. datasource: Содержит настройки для подключения к базе данных. driver-class-name указывает класс драйвера PostgreSQL. url содержит URL-адрес для подключения к базе данных. username и password содержат учетные данные пользователя базы данных а так же дополнительные настройки для загрузки наших фото.
2. jpa: Содержит настройки JPA и Hibernate. hibernate.ddl-auto определяет стратегию автоматического создания таблиц в базе данных. Значение update указывает Hibernate на автоматическое обновление схемы базы данных при изменении сущностей JPA.

В общем, схема взаимодействия таблиц сделана таким образом:



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

Для обеспечения безопасности приложения был использован Spring Security.

Spring Security - это мощный и гибкий фреймворк аутентификации и авторизации для приложений на платформе Java. Он обеспечивает обширный набор функций для защиты ваших приложений, включая аутентификацию пользователей, управление их правами доступа, защиту от атак, таких как CSRF и сессионные атаки, а также интеграцию с другими инфраструктурными службами, такими как LDAP, OAuth и другими.

Spring Security позволяет легко настраивать правила безопасности с использованием аннотаций или конфигурационных классов Java. Он также интегрируется с другими модулями Spring, такими как Spring MVC, что облегчает создание безопасных и надежных приложений на платформе Java.   
Он предоставляет механизмы аутентификации и авторизации, защиту от атак и другие функции безопасности. Вот основные концепции и функции Spring Security:

1. **Аутентификация**: Это процесс проверки подлинности пользователя. Spring Security поддерживает различные механизмы аутентификации, такие как форма, HTTP Basic, OAuth и другие.
2. **Авторизация**: Это процесс определения разрешений пользователя на доступ к определенным ресурсам или выполнение определенных действий. Spring Security использует роли и разрешения для управления доступом.
3. **Фильтры безопасности**: Spring Security использует цепочку фильтров для обработки запросов и применения правил безопасности. Эти фильтры обеспечивают аутентификацию, авторизацию, защиту от CSRF-атак, управление сессиями и другие функции.
4. **Защита от CSRF-атак**: CSRF (межсайтовая подделка запросов) - это атака, при которой злоумышленник заставляет пользователя совершить нежелательное действие на веб-сайте, на который он авторизован. Spring Security предоставляет защиту от таких атак.
5. **Управление сессиями**: Spring Security предоставляет возможности управления сессиями, включая защиту от смены сессии, контроль одновременных сессий и настройку таймаута сессии.
6. **Шифрование паролей**: Spring Security советует хранить пароли в зашифрованном виде, предоставляя возможность использовать алгоритмы шифрования, такие как BCrypt, SCrypt и PBKDF2.
7. **Интеграция с Spring Framework**: Spring Security интегрируется плавно с Spring Framework, что упрощает обеспечение безопасности в приложениях на основе Spring.
8. **Настройка и расширяемость**: Spring Security предоставляет множество возможностей для настройки и расширения. Вы можете настраивать логику аутентификации и авторизации, создавать собственные фильтры безопасности и интегрироваться с сторонними сервисами аутентификации.

Spring Security обеспечивает полную защиту вашего приложения на Java, делая его популярным выбором для реализации безопасности в проектах, использующих Spring.

**2.5. Разворачивание Docker-контейнера**

Разворачивание Docker-контейнеров представляет собой процесс запуска и управления контейнерами на целевом хосте. Для этого используется Docker-демон, который управляет жизненным циклом контейнеров, такими как создание, запуск, остановка и удаление.

Основные шаги разворачивания Docker-контейнеров:

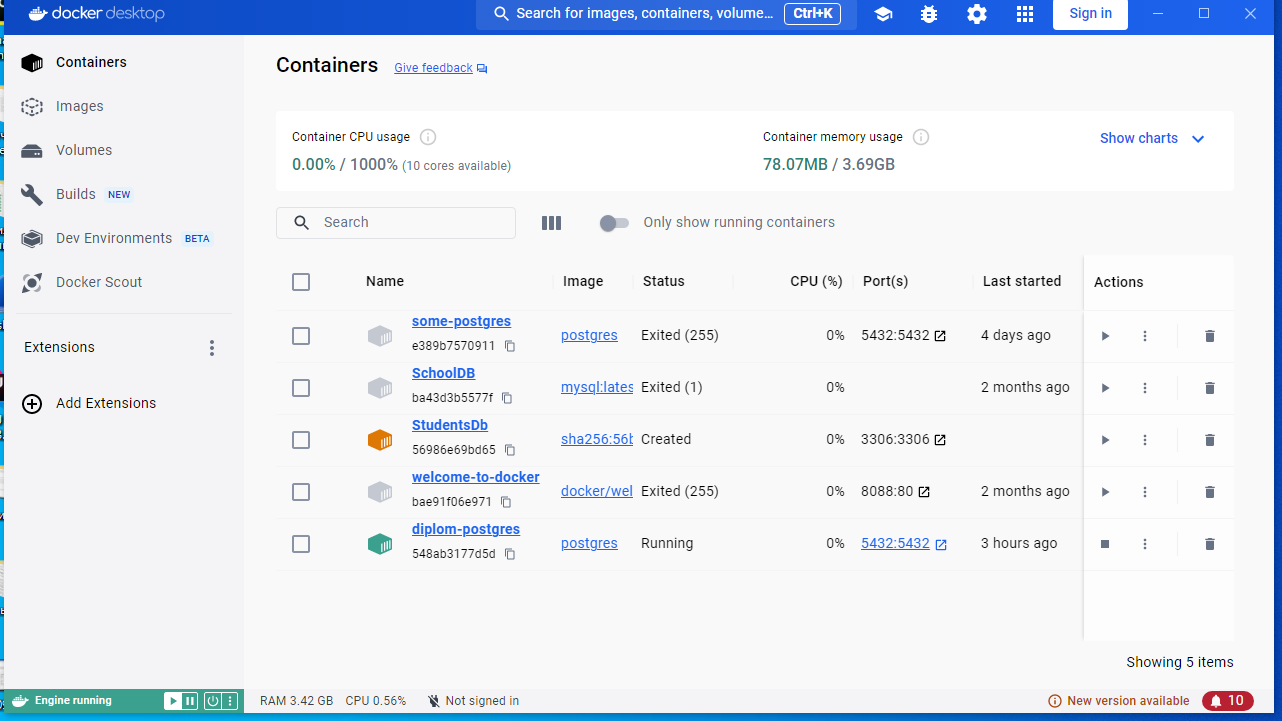
1. **Установка Docker**: Сначала необходимо установить Docker на целевом хосте. Для этого можно воспользоваться официальным сайтом Docker, где доступны инструкции по установке для различных операционных систем.
2. **Создание Docker-образа**: Docker-образ представляет собой шаблон, на основе которого создается контейнер. Образ содержит все необходимые зависимости и конфигурации приложения. Образ можно создать с помощью Dockerfile, который описывает шаги по сборке образа.
3. **Создание контейнера**: После создания образа необходимо создать контейнер на его основе. Это делается с помощью команды docker run. Например, для запуска контейнера на основе созданного образа:

**docker run --name diplom-postgres -e POSTGRES\_PASSWORD=** **root123 -e POSTGRES\_DB=** **MyAppsDb-pg-13.3 -d -p 5432:5432 postgres**

Команда **docker run** используется для создания и запуска контейнера на основе Docker-образа. Давайте разберем каждый параметр данной команды:

* **--name diplom-postgres**: Устанавливает имя контейнера как **diplom-postgres**. Это удобно для идентификации контейнера при его управлении.
* **-e POSTGRES\_PASSWORD=root123**: Устанавливает переменную окружения **POSTGRES\_PASSWORD** со значением **root123** для задания пароля для пользователя **postgres** в базе данных PostgreSQL.
* **-e POSTGRES\_DB=** **MyAppsDb-pg-13.3**: Устанавливает переменную окружения **POSTGRES\_DB** со значением **diplom\_db** для создания базы данных с именем **diplom\_db** в PostgreSQL.
* **-d**: Запускает контейнер в фоновом режиме (detached mode).
* **-p 5432:5432**: Пробрасывает порт 5432 из контейнера на хостовую машину. Первое число (5432) - порт на хостовой машине, второе число (5432) - порт в контейнере, на котором работает PostgreSQL.
* **postgres**: Имя Docker-образа, на основе которого будет создан контейнер. В данном случае, это официальный образ PostgreSQL из Docker Hub.

После выполнения этой команды будет создан и запущен контейнер с именем **diplom-postgres**, в котором будет работать PostgreSQL с базой данных **diplom\_db** и паролем **admin**, пробрасывая порт 5432 для доступа к базе данных извне контейнера.



**Заключение**

В рамках данного дипломного проекта было разработано веб-приложение «EasyMarket», созданное как торговая площадка. Проект был реализован с использованием современных технологий веб-разработки, таких как Java Spring Framework, Thymeleaf, Bootstrap и PostgreSQL.

В процессе разработки были достигнуты следующие цели:

* Создание удобного и интуитивно понятного интерфейса для пользователей приложения.
* Реализация функционала добавления, редактирования и удаления записей о товарах.
* Обеспечение безопасности данных пользователей с использованием механизмов аутентификации и авторизации Spring Security.
* Использование принципов SOLID и паттернов проектирования для обеспечения гибкости и расширяемости кода.

Дальнейшее развитие приложения может включать в себя добавление нового функционала, оплата и доставка товаров в указанные регионы, а также мобильное приложение для удобного доступа к платформе из любой точки.

В целом, дипломный проект " EasyMarket" позволил не только показать навыки разработки веб-приложений с использованием современных технологий, но и продемонстрировать понимание принципов безопасности и удобства использования для конечного пользователя.

**Список литературы**

1. **Spring в действии,** Крейг Уоллс (6-е издание, 2022 год).
2. Учебный материал Гикбрейнс.
3. Статьи и видео-материалы из сети интернет.