**ООО «ГикБреинс»**

**ДИПЛОМНАЯ РАБОТА**

**СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ «БИБЛИОТЕКА»**

**Специальность «Разработчик приложений на языке Java»**

**Автор работы Руткевич В.Ю.**

**Республика Беларусь, г. Гродно**

**2024**

[ВВЕДЕНИЕ 4](#_Toc162274809)

[1. Фреймворки веб-приложений и другие технологии 7](#_Toc162274810)

[1.1 Java 7](#_Toc162274811)

[1.2 Maven 8](#_Toc162274812)

[1.3 Spring Boot 9](#_Toc162274813)

[1.4 Spring Security 10](#_Toc162274814)

[1.5 JDBC 11](#_Toc162274815)

[1.6 Spring mvc 12](#_Toc162274816)

[1.7 Lombok 13](#_Toc162274817)

[1.8 Mockito 14](#_Toc162274818)

[1.9 JUnit 15](#_Toc162274819)

[1.10 Hibernate 16](#_Toc162274820)

[1.11 Liquibase 17](#_Toc162274821)

[1.12 JavaMail 18](#_Toc162274822)

[1.13 MySQL 18](#_Toc162274823)

[1.14 CSS и HTML 19](#_Toc162274824)

[1.15 Velocity 19](#_Toc162274825)

[1.15 Thymeleaf 21](#_Toc162274826)

[1.16 Модульность 22](#_Toc162274827)

[2. Описание разработки веб-приложения 24](#_Toc162274828)

[2.1 Модуль Entity. 24](#_Toc162274829)

[2.1.1 Maven зависимости. 24](#_Toc162274830)

[2.1.2 Класс AEntity. 25](#_Toc162274831)

[2.1.3 Класс Book. 27](#_Toc162274832)

[2.1.4 Класс BookDetails. 29](#_Toc162274833)

[2.1.5 Класс Department. 31](#_Toc162274834)

[2.1.6 Класс Feedback. 33](#_Toc162274835)

[2.1.7 Класс Order. 35](#_Toc162274836)

[2.1.8 Класс Role. 37](#_Toc162274837)

[2.1.9 Класс User. 39](#_Toc162274838)

[2.1.10 Application properties 41](#_Toc162274839)

[2.1.11 Liquibase для управления изменениями базы данных. 42](#_Toc162274840)

[2.2 Модуль API. 45](#_Toc162274841)

[2.2.1 Maven зависимости. 45](#_Toc162274842)

[2.2.2 Иерархия и функциональность модуля API. 46](#_Toc162274843)

[2.3 Модуль Utils. 47](#_Toc162274844)

[2.3.1 Класс MailConfiguration. 48](#_Toc162274845)

[2.3.2 Класс MapperConfiguration. 51](#_Toc162274846)

[2.3.3 Класс EmailSender. 53](#_Toc162274847)

[2.4 Модуль DAO. 56](#_Toc162274848)

[2.5 Модуль WEB. 58](#_Toc162274849)

[2.5.1 Класс WebClientProvider. 59](#_Toc162274850)

[2.5.2 Класс WebScraper . 61](#_Toc162274851)

[2.6 Модуль Controller. 63](#_Toc162274852)

[2.6.1 CustomAccessDeniedHandler. 63](#_Toc162274853)

[2.6.2 Класс MvcConfig. 64](#_Toc162274854)

[2.6.3 Класс SecurityConfiguration. 66](#_Toc162274855)

[2.6.4 AdminController. 68](#_Toc162274856)

[2.6.5 Класс FacebookConnectionSignUp. 73](#_Toc162274857)

[2.6.6 Класс FacebookSignInAdapter. 74](#_Toc162274858)

[2.6.7 Класс ImgFileUploader. 75](#_Toc162274859)

[2.7 Service. 78](#_Toc162274860)

[3. Тестирование приложения в среде разработки и UI-тестирование. 79](#_Toc162274861)

[3.1 Наборы шаблонов и стилей для фронта. 79](#_Toc162274862)

[3.2 JUnit – тестрование. 82](#_Toc162274863)

[3.3 UI – тестирование. 84](#_Toc162274864)

[Заключение 97](#_Toc162274865)

[Список используемой литературы: 99](#_Toc162274866)

ВВЕДЕНИЕ

На сегодняшний день программирование окружает нас во всех сферах нашей жизни. Мы пользуемся различными приложениями в смартфонах, телевизорах, пылесосах и даже автомобилях не задумываясь об этом. Программирование является не только процессом и инструмент по созданию приложений и веб-сайтов, но и средством автоматизации, оптимизации и решения сложных задач. Все это делает его неотъемлемой частью современного мира. Мы взаимодействуем с компьютерами при помощи языков программирования, которых уже существует более 9000, однако на практике используются всего около 20. Программирование включает в себя создание алгоритмов — последовательности шагов, которые решают определенную задачу. Эти алгоритмы затем переводятся в код программы, который компьютер может понять и выполнить. Одним из наиболее востребованных и быстроразвивающихся направлений программирования является разработка веб-приложений.

Веб-приложения играют ключевую роль в современном мире. Они представляют собой программы, которые работают через браузер, не требуя установки на устройство. В отличие от обычных приложений, веб-приложения обладают следующими особенностями:

1. Онлайн-доступ: Пользователи могут взаимодействовать с веб-приложениями через интернет, без необходимости загрузки и установки на свои устройства.

2. Интерактивность: Веб-приложения позволяют пользователям выполнять разнообразные действия, такие как заказ товаров, бронирование билетов, оставление отзывов и редактирование контента.

Примеры: Веб-приложения включают в себя полноценные онлайн-редакторы (например, Google Документы) и системы управления контентом (CMS) (Tilda).

Создание веб-приложения — это прогрессивный путь для развития компании. Оно позволяет:

1. Внедрять системы и программы,которые выполняют рутинные задачи и операции в компании без необходимости вмешательства человека. Это позволяет улучшить эффективность, сократить время выполнения задач и снизить вероятность ошибок. В результате бизнес-процессы становятся более оптимизированными и автоматизированными.

2. Привлекать внимание целевой аудитории, предоставляя пользователям удобный и интересный функционал, а также выделяться на фоне конкурентов предлагая уникальные решения для выполнения запланированных задач.

3. Упрощать, уменьшать риски или предотвращать нежелательные последствияпри выполнении решение задач при помощи следующих методов:

* Сегментация сети.
* Хранение паролей в хешированном виде.
* Управление обновлениями.
* Мониторинг и документирование.

Примером успешного веб-приложения является платформа электронной коммерции Shopify, которая начала свой путь в 2004 году и стала одной из крупнейших онлайн-платформ для развития электронной коммерции.

Веб-приложения работают по клиент-серверному принципу, обеспечивая функциональность как статических, так и динамических страниц. Они используются для различных целей, включая коммуникацию, создание контента, электронную коммерцию и многое другое.

В данном рабочем проекте мною будет реализовано клиент-серверное взаимодействие в веб-приложении «Библиотека». Работа будет реализовывать выполнение следующих задач:

* Проектирование базы данных
* Реализация бизнес-логики
* Разработка бэкенда и фронтенда
* Исправление ошибок
* Оптимизация

Технологии, которые мне будет необходимо применить для реализации поставленных целей:

Java 8, Maven, Spring Boot, Spring Security, JDBC, Spring mvc, Lombok, Mockito, JUnit, Hibernate, Liquibase, JavaMail, MySQL, HtmlUnit, CSS, HTML, Velocity, Thymeleaf.

Более подробно о данных технологиях будет описано в дипломной работе.

1. Фреймворки веб-приложений и другие технологии

1.1 Java

Java — это строго типизированный объектно-ориентированный язык программирования общего назначения, который был разработан компанией Sun Microsystems (впоследствии приобретенной компанией Oracle) [1]. Впервые выпущенный в 1995 году, Java стал мощным инструментом в современном цифровом мире [2].

К ключевым аспектам Java можно отнести следующее:

* Платформа вычислений: Java не только является языком программирования, но и представляет собой платформу для вычислений. Она обеспечивает надежную среду для разработки и запуска различных приложений и сервисов.
* Java Virtual Machine (JVM): JVM — это виртуальная машина, которая выполняет Java-код. Она обеспечивает переносимость приложений, позволяя им работать на разных операционных системах без изменений.
* Java Development Kit (JDK): JDK включает в себя JVM и другие инструменты для разработки, такие как компилятор Java. Разработчики используют JDK для создания Java-приложений.
* Java Runtime Environment (JRE): JRE — это среда выполнения для Java. Она включает в себя JVM и базовые классы платформы Java. JRE необходима для запуска Java-приложений.
* Безопасность: Java обладает мощными механизмами безопасности, такими как контроль доступа и проверка типов. Это делает ее популярной для веб-приложений и других задач, где безопасность играет важную роль.
* Многопоточность: Java поддерживает многопоточность, что позволяет эффективно использовать множество процессорных ядер.
* Большая экосистема: Существует множество библиотек, фреймворков и инструментов, разработанных на Java, что делает ее привлекательной для разработчиков.

Java используется в различных областях, включая веб-разработку, мобильные приложения, большие корпоративные системы и даже встроенные устройства. Она продолжает развиваться, и инновационные продукты будут создаваться на ее основе для будущего [3].

1.2 Maven

Maven — это фреймворк для автоматизации сборки проектов, основанный на описании их структуры в файлах POM (Project Object Model) на языке XML [1]. Основным функционалом Maven является:

* Автоматизация сборки: Maven обеспечивает декларативную сборку проекта, где в файлах описания содержится его спецификация, а не отдельные команды выполнения. Все задачи по обработке файлов выполняются с помощью встроенных и внешних плагинов.
* Структура проекта: Проекты в Maven организуются в определенной структуре, которая упрощает управление зависимостями, ресурсами и тестами.
* Управление зависимостями: Maven автоматически загружает и устанавливает зависимости из центрального репозитория или других источников.
* Жизненный цикл: Maven определяет жизненный цикл проекта, включая фазы, такие как компиляция, тестирование, упаковка и развертывание.
* Плагины: Maven поддерживает множество плагинов, которые расширяют его функциональность. Например, плагины для компиляции, тестирования, деплоя и документации.
* Многопроектные сборки: Maven позволяет управлять несколькими проектами в одной среде, обеспечивая согласованность и повторное использование.

Maven используется для построения и управления проектами на Java, C#, Ruby, Scala и других языках [1]. Он помогает разработчикам эффективно управлять зависимостями, автоматизировать сборку и обеспечивать согласованность в проектах [4].

1.3 Spring Boot

Spring Boot — это фреймворк для разработки приложений на платформе Java. Он предоставляет инструменты и возможности для создания высокопроизводительных и масштабируемых приложений [5]. Вот некоторые ключевые аспекты Spring Boot:

* Упрощение разработки: Spring Boot устраняет необходимость в сложной конфигурации зависимостей, что делает процесс создания приложений более простым и быстрым.
* Автоматическая конфигурация: Spring Boot автоматически настраивает приложение на основе классов и аннотаций, что упрощает жизнь разработчиков.
* Встроенные серверы: Spring Boot поставляется с встроенными серверами (например, Tomcat, Jetty или Undertow), что позволяет запускать приложения без необходимости настройки отдельного сервера.
* Управление зависимостями: Spring Boot автоматически управляет зависимостями, загружая их из центрального репозитория.
* Монолит или микросервисы: Spring Boot подходит как для создания монолитных приложений, так и для разработки микросервисов.
* Активное сообщество: Spring Boot имеет большое и активное сообщество разработчиков, которые создают библиотеки, документацию и обучающие материалы.

Spring Boot позволяет разработчикам сосредоточиться на бизнес-логике приложения, не заботясь о множестве деталей, связанных с настройкой и конфигурацией [6][7].

1.4 Spring Security

Spring Security — это Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework [1][7]. Он играет важную роль в обеспечении безопасности веб-приложений и API.

Рассмотрим некоторые ключевые аспекты Spring Security:

* Аутентификация: Spring Security позволяет проверять подлинность пользователей, например, по паролю, токену или другим методам.
* Авторизация: Фреймворк управляет доступом пользователей к различным ресурсам приложения, определяя, какие действия разрешены.
* Фильтры безопасности: Spring Security использует цепочку фильтров для обработки запросов, включая фильтры аутентификации и авторизации.
* Конфигурация: Разработчики могут настраивать правила безопасности с помощью аннотаций, XML-конфигурации или Java-кода.
* Интеграция с другими технологиями: Spring Security интегрируется с Spring MVC, Spring Boot, OAuth, LDAP и другими инструментами.
* Пользовательские роли и разрешения: Разработчики могут определять пользовательские роли и разрешения для управления доступом.

Spring Security помогает защитить приложения от атак, обеспечивая конфиденциальность данных и контроль над доступом к функциональности [3][4].

1.5 JDBC

JDBC (Java Database Connectivity) — это платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными системами управления базами данных (СУБД) [1]. Впервые включенный в состав JDK 1.1 в 1997 году, JDBC стал частью J2SE и предоставил разработчикам мощный инструмент для работы с данными.

Некоторые ключевые аспекты JDBC:

* Аутентификация и авторизация: JDBC позволяет проверять подлинность пользователей и управлять доступом к базам данных.
* Фильтры безопасности: Фреймворк использует цепочку фильтров для обработки запросов, включая фильтры аутентификации и авторизации.
* Управление зависимостями: JDBC автоматически управляет зависимостями, загружая их из центрального репозитория.
* Интерфейсы: JDBC API содержит интерфейсы для разработчиков приложений и драйверов. Соединение с базой данных описывается классом, реализующим интерфейс «java.sql.Connection».

JDBC помогает разработчикам эффективно управлять зависимостями и обеспечивать безопасность при работе с данными.

1.6 Spring mvc

Spring MVC — это веб-фреймворк Spring, который позволяет создавать веб-сайты и RESTful сервисы, следуя паттерну проектирования Model-View-Controller (MVC) [1][10]. Говоря об основных параметрах Spring MVC выделю следующее:

* Разделение ролей: Spring MVC разделяет каждую роль, где модель, контроллер, представление, командные объекты, резолверы представлений и другие компоненты могут быть реализованы специализированными объектами.
* Легковесность: Он использует легковесный контейнер сервлетов для разработки и развертывания приложений.
* Мощная конфигурация: Предоставляет надежную конфигурацию для фреймворка и классов приложения, включая простое взаимодействие между контроллерами, бизнес-объектами и валидаторами.
* Быстрая разработка: Spring MVC облегчает параллельную разработку.
* Повторное использование бизнес-кода: Позволяет использовать существующие бизнес-объекты.
* Легкость тестирования: В Spring мы создаем классы JavaBeans, которые позволяют внедрять тестовые данные с помощью методов установки.
* Гибкое отображение: Поддерживает различные технологии представления, такие как JSP+JSTL, Apache Velocity, Thymeleaf и FreeMarker.

Spring MVC управляет потоком приложения через DispatcherServlet, который обрабатывает входящие запросы и направляет их на соответствующие контроллеры, модели и представления [7].

1.7 Lombok

Lombok — это библиотека для сокращения кода в классах и расширения функциональности языка Java. Она позволяет уменьшить шаблонный код, который разработчики обычно пишут в классах, таких как геттеры, сеттеры, конструкторы и методы equals, hashCode и toString [7].

Некоторые ключевые возможности Lombok:

1. Аннотации: В Lombok предлагаются различные аннотации, которые автоматически генерируют код. Например:

* @Getter и @Setter: Генерируют геттеры и сеттеры для полей.
* @NoArgsConstructor и @AllArgsConstructor: Создают конструкторы без аргументов и с аргументами соответственно.
* @EqualsAndHashCode: Генерирует методы equals и hashCode.
* @ToString: Создает метод toString.

2. Сокращение кода: Благодаря Lombok, разработчики могут писать более компактный и читаемый код, избегая рутинных операций.

3. Интеграция с IDE и сборщиками: Lombok подключается к среде разработки (IDE) или инструментам сборки приложений (например, Maven или Gradle) в качестве плагина.

Loombok упрощает разработку и улучшает читаемость кода.

1.8 Mockito

Mockito — это фреймворк для тестирования приложений на языке Java, который позволяет легко и быстро создавать фиктивные (мок) объекты. Эти моки заменяют реальные объекты в тестах, что упрощает проверку поведения и взаимодействия между компонентами приложения [1][7].

Некоторые ключевые возможности Mockito:

* Создание моков: Mockito позволяет создавать моки для классов и интерфейсов. Это полезно, когда нужно изолировать тестируемый код от внешних зависимостей.
* Задание поведения моков: Вы можете указать, как моки должны вести себя в разных ситуациях. Например, какой результат вернуть при вызове определенного метода.
* Проверка вызовов методов: Mockito позволяет проверять, были ли вызваны определенные методы у моков и с какими аргументами.
* Интеграция с JUnit: Mockito хорошо интегрируется с фреймворком для юнит-тестирования JUnit.

Mockito помогает создавать надежные и чистые тесты, изолируя код от реальных зависимостей [4].

1.9 JUnit

JUnit — это фреймворк для модульного тестирования программного обеспечения на языке Java. Созданный Кентом Беком и Эриком Гаммой, JUnit принадлежит семье фреймворков xUnit для разных языков программирования, берущей начало в SUnit Кента Бека для Smalltalk [1][7]. Он играет важную роль в обеспечении безопасности веб-приложений и API.

Некоторые ключевые возможности JUnit:

* 1. Аннотации: В JUnit предлагаются различные аннотации, которые автоматически генерируют код. Например:
  + @Test: Обозначает метод как тестовый.
  + @Before и @After: Позволяют выполнять код до и после каждого тестового метода.
  + @BeforeClass и @AfterClass: Запускаются один раз перед началом и после завершения всех тестов.

2. Утверждения (assertions): JUnit предоставляет методы для проверки ожидаемых результатов. Например:

* + assertEquals(expected, actual): Проверяет, что expected равно actual.
  + assertTrue(condition): Проверяет, что condition истинно.

3. Интеграция с средой разработки: JUnit хорошо интегрируется с фреймворками разработки, такими как Eclipse и IntelliJ IDEA.

JUnit помогает разработчикам создавать надежные и автоматизированные тесты, обеспечивая качество кода и быстрое выявление ошибок.

1.10 Hibernate

Hibernate — это фреймворк для работы с базами данных (ORM), который позволяет разработчикам взаимодействовать с реляционными базами данных, используя объектно-ориентированный подход [1][7]. Наиболее важные функции Hibernate:

* Объектно-реляционное отображение (ORM): Hibernate позволяет связать объекты Java с таблицами в базе данных. Он автоматически выполняет преобразование данных между объектами и записями в базе данных.
* Сессии и транзакции: Hibernate управляет сессиями (связь между приложением и базой данных) и обеспечивает транзакционность операций.
* Аннотации и XML-конфигурация: Разработчики могут настраивать Hibernate с помощью аннотаций или XML-файлов.
* Ленивая загрузка: Hibernate поддерживает ленивую загрузку данных, что улучшает производительность приложения.
* Кэширование: Фреймворк предоставляет механизмы кэширования для ускорения доступа к данным.

Hibernate позволяет разработчикам работать с базами данных, не вдаваясь в детали SQL-запросов, и упрощает разработку приложений.

1.11 Liquibase

Liquibase — это система управления миграциями базы данных. Она помогает командам версионировать, отслеживать и развертывать изменения в базе данных. Liquibase позволяет быстрее и безопаснее вносить изменения от разработки до продакшена [11][12].

Некоторые ключевые возможности Liquibase:

* Миграции: Liquibase позволяет описывать изменения в базе данных в виде миграций. Это может быть создание таблиц, добавление столбцов, индексов и другие операции.
* Кросс-платформенность: Liquibase работает с разными СУБД, такими как MySQL, PostgreSQL, Oracle, SQL Server и другие.
* Автоматическое применение изменений: При запуске приложения Liquibase автоматически применяет необходимые миграции к базе данных.
* Контроль версий: Миграции хранятся в контроле версий, что обеспечивает историю изменений.
* Интеграция с средой разработки: Liquibase интегрируется с инструментами разработки, такими как Maven и Gradle.

Liquibase помогает разработчикам управлять изменениями в базе данных, обеспечивая надежность и безопасность.

1.12 JavaMail

JavaMail — это Java API для отправки и получения электронной почты с использованием протоколов SMTP, POP3 и IMAP [1]. Он позволяет разработчикам взаимодействовать с почтовыми серверами, отправлять письма, работать с вложениями и управлять электронной почтой в своих приложениях. JavaMail является частью платформы Java EE и предоставляет надежные инструменты для работы с электронной почтой.

1.13 MySQL

MySQL — это реляционная система управления базами данных (СУБД), которая позволяет хранить, организовывать и извлекать данные. Вот некоторые ключевые аспекты MySQL:

* Бесплатность и открытость: MySQL является бесплатной и открытой системой, доступной для всех.
* Производительность: Она обеспечивает высокую производительность при обработке запросов.
* Многопользовательская поддержка: MySQL позволяет одновременно работать с базой данных нескольким пользователям.
* Поддержка SQL: Она использует язык SQL для создания, изменения и запросов данных.
* Индексы и оптимизация: MySQL поддерживает индексы, что улучшает скорость поиска данных.
* Сообщество и документация: Существует большое сообщество разработчиков и обширная документация.

MySQL широко используется в веб-приложениях, корпоративных системах и других областях.

1.14 CSS и HTML

HTML (Hypertext Markup Language) — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора [1].

CSS — формальный язык декодирования и описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки (чаще всего HTML или XHTML). Также может применяться к любым XML-документам, например, к SVG или XUL [1].

HTML и CSS работают вместе, чтобы создать красивые и структурированные веб-страницы. HTML формирует содержимое, а CSS — его стиль и внешний вид.

1.15 Velocity

Velocity — это процессор шаблонов (обработчик шаблонов), базирующийся на Java. Он обеспечивает простой, но мощный шаблонный язык, который не требует предварительной подготовки модели переменных для шаблона. Вместо этого в шаблон просто передаются ссылки на Java-объекты, а обработчик разбирает указанные методы и, с помощью Java Reflection API, получает их значения.

Цель Velocity заключается в том, чтобы гарантировать чистое разделение между уровнем представления и бизнес-уровнем в веб-приложении (шаблон проектирования Model-View-Controller), но он не ограничен областью веб-приложений.

Пример использования Velocity включает следующие области:

1. Веб-приложения:

* Веб-дизайнеры создают HTML-страницы с указателями мест заполнения, которые будут динамически заменены необходимой информацией.
* Страницы могут обрабатываться с помощью VelocityViewServlet или других фреймворков, поддерживающих Velocity.

2. Генерация исходного кода:

* Velocity может использоваться для генерации кода на Java, SQL, PostScript, JavaScript, Perl и других языках, основываясь на шаблоне.

3. Автоматические электронные письма:

* Множество приложений генерируют автоматические электронные письма для регистрации учетных записей, напоминания о забытых паролях и автоматически сгенерированных отчетов.
* С помощью Velocity шаблон электронного письма может быть сохранен в виде текстового файла, а не встроен непосредственно в Java-код.

4. XML-преобразование:

* Velocity предоставляет задачу Ant под названием Anakia, которая читает XML-файл и делает его доступным для шаблона Velocity.
* Общее приложение может преобразовать документацию, сохраненную в универсальном формате «xdoc», в стилизованный документ HTML [1].

Velocity — это мощный инструмент для работы с шаблонами в Java, который позволяет разделять логику и представление в веб-приложениях и других областях [1].

1.15 Thymeleaf

Thymeleaf — это современный серверный механизм Java-шаблонов, который используется для веб- и автономных сред. Он способен обрабатывать HTML, XML, JavaScript, CSS и даже простой текст [7][13].

Вот краткое описание:

Thymeleaf — библиотека на Java.

Он представляет собой шаблонный движок, который применяет набор преобразований к файлам шаблонов для отображения данных и/или текста, созданного приложениями.

Thymeleaf хорошо подходит для обслуживания XHTML/HTML5 в веб-приложениях, но он также может обрабатывать любой XML-файл как в веб-приложениях, так и в автономных приложениях [13].

Основные области использования Thymeleaf включают веб-приложения, генерацию исходного кода, автоматические электронные письма и преобразование XML [7][13].

1.16 Модульность

Модульность в программировании — это подход, при котором программа разбивается на отдельные, независимые блоки, называемые модулями. Каждый модуль выполняет определенную функцию и взаимодействует с другими модулями через четко определенные интерфейсы. Это позволяет разработчикам работать над разными частями программы одновременно, упрощает тестирование и обнаружение ошибок, а также облегчает обновление и рефакторинг кода.

Мой проект, использующий Maven и модульную структуру, представляет собой отличный пример применения этих принципов. Он включает в себя следующие модули:

* Entity: Основной модуль, отвечающий за сущности и наполнение базы данных с помощью Liquibase.
* API: Модуль, содержащий исключения и интерфейсы ко всем слоям, включая DAO и сервисы.
* Utils: Настройка ModelMapper и почтовой рассылки.
* DAO: Конфигурации DataSource и слой DAO.
* Web: Настройка веб-клиента и сервиса, который обращается к сайту BookFinder, извлекает данные о книгах по ISBN и интегрирует эти данные в само приложение.
* Service: Содержит бизнес-логику приложения.
* Controller: Модуль с настройками веб-разрешений, просмотром безопасности, аутентификацией через Facebook и загрузчиком изображений.

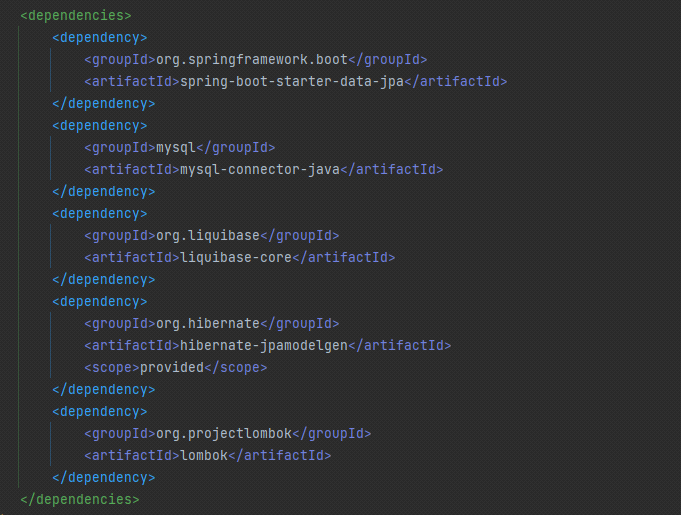
Каждый модуль в моем проекте выполняет свою специфическую роль и зависит от функциональности, предоставляемой предыдущими модулями в иерархии. Это создает четкую структуру, которая облегчает понимание и разработку проекта, а также позволяет легко расширять или модифицировать отдельные части приложения. Использование Maven в данном контексте обеспечивает автоматизацию сборки и управления зависимостями, что является ключевым для поддержания порядка и эффективности в мультимодульном проекте.

2. Описание разработки веб-приложения

2.1 Модуль Entity.

2.1.1 Maven зависимости.

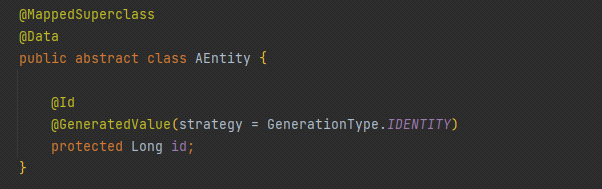
Для разработки модуля Entity мы подключаем следующие зависимости при помощи Maven:



1. spring-boot-starter-data-jpa: Эта зависимость предоставляет инфраструктуру для работы с JPA (Java Persistence API) в Spring Boot приложениях. Она упрощает создание репозиториев, выполнение запросов к базе данных и управление сущностями.
2. mysql-connector-java: Эта зависимость предоставляет драйвер для подключения к базе данных MySQL. Она позволяет приложению взаимодействовать с MySQL сервером.
3. liquibase-core: Эта зависимость используется для управления миграциями базы данных. Liquibase позволяет контролировать изменения в схеме базы данных и автоматически обновлять ее при необходимости.
4. hibernate-jpamodelgen: Эта зависимость используется для генерации метамоделей JPA (классы, представляющие сущности и их атрибуты) во время компиляции. Это помогает избежать ошибок при написании запросов JPA.
5. lombok: Эта зависимость добавляет аннотации, которые автоматически генерируют код для геттеров, сеттеров, конструкторов и других методов. Она упрощает написание кода и уменьшает его объем.

Далее приступим кнаписанию уже самих классов entities.

2.1.2 Класс AEntity.



Этот абстрактный класс AEntity имеет следующие особенности:

1. @MappedSuperclass: Эта аннотация указывает, что AEntity является базовым классом для сущностей (Entity) в JPA (Java Persistence API). Она не создает таблицу в базе данных, но наследуемые от нее классы могут использовать ее поля и аннотации.
2. @Data: Эта аннотация генерирует автоматически код для геттеров, сеттеров, equals, hashCode и других методов на основе полей класса. Она упрощает написание кода и уменьшает объем.

В данном случае, класс AEntity содержит поле id, которое является идентификатором сущности. Он также наследует аннотации и свойства от @MappedSuperclass, что позволяет другим сущностям расширять его функциональность.

2.1.3 Класс Book.



Book — это класс, представляющий сущность (Entity) “книга” в системе. Рассмотрим его основные аспекты:

1. Аннотации:

* @Entity: Эта аннотация указывает, что класс Book является сущностью, которая будет отображаться в базе данных как таблица.
* @Data: Эта аннотация генерирует автоматически код для геттеров, сеттеров, equals, hashCode и других методов на основе полей класса.
* @EqualsAndHashCode(callSuper = true): Генерирует методы equals и hashCode на основе всех полей класса, включая поля, унаследованные от родительского класса.
* @Accessors(fluent = false, chain = true): Эта аннотация позволяет настраивать стиль доступа к полям. Здесь chain = true позволяет использовать цепочку вызовов для методов сеттеров.
* @Table(name = “book”): Указывает имя таблицы в базе данных, которая будет соответствовать сущности Book.

2. Поля:

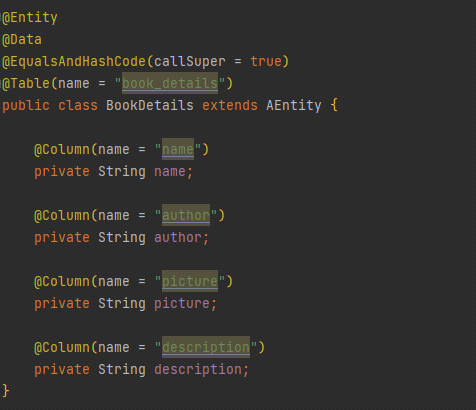
* isbn: Строковое поле, представляющее ISBN книги.
* quantityAvailable: Целочисленное поле, обозначающее количество доступных книг.
* quantityInLibrary: Целочисленное поле, обозначающее количество книг в библиотеке.
* rating: Поле типа Double, представляющее рейтинг книги.
* departments: Список связанных сущностей Department через отношение “многие ко многим”.
* feedbacks: Список связанных сущностей Feedback через отношение “один ко многим”.
* orders: Список связанных сущностей Order через отношение “один ко многим”.
* bookDetails: Связь “один к одному” с сущностью BookDetails, где детали книги хранятся.

3. Примечание:

* При удалении книги также будут удаляться связанные с ней детали (поле bookDetails).

Этот класс Book представляет собой модель для работы с данными о книгах в системе.

2.1.4 Класс BookDetails.



BookDetails — это класс, представляющий сущность (Entity) “детали книги” в системе. Рассмотрим далее его параметры:

1. Аннотации:

* @Entity: Эта аннотация указывает, что класс BookDetails является сущностью, которая будет отображаться в базе данных как таблица.
* @Data: Эта аннотация генерирует автоматически код для геттеров, сеттеров, equals, hashCode и других методов на основе полей класса. Она упрощает написание кода и уменьшает объем.
* @EqualsAndHashCode(callSuper = true): Генерирует методы equals и hashCode на основе всех полей класса, включая поля, унаследованные от родительского класса.
* @Table(name = “book\_details”): Указывает имя таблицы в базе данных, которая будет соответствовать сущности BookDetails.

2. Поля:

* name: Строковое поле, представляющее название книги.
* author: Строковое поле, обозначающее автора книги.
* picture: Строковое поле, содержащее ссылку на изображение обложки книги.
* description: Строковое поле, представляющее описание книги.

Этот класс BookDetails используется для хранения дополнительных данных о книгах, таких как название, автор, обложка и описание.

2.1.5 Класс Department.



Класс Department представляющий сущность (Entity), описывает отдел в контексте системы. Краткое описание его полей и свойств:

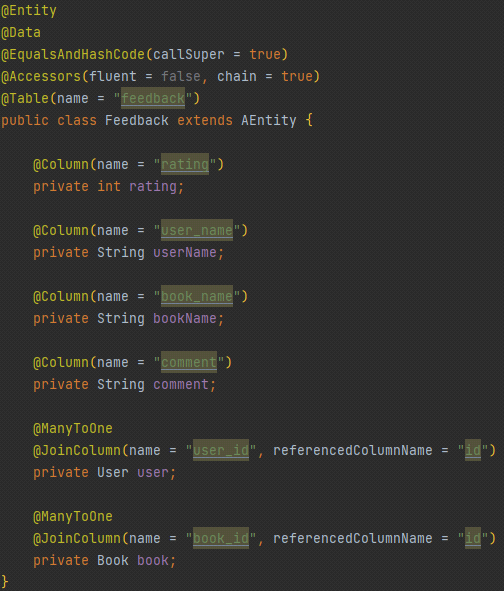
1. Аннотации:

* @Entity: Эта аннотация указывает, что класс Department является сущностью (entity) в базе данных. Такие сущности могут быть сохранены, извлечены и обновлены в базе данных.
* @Data: Эта аннотация генерирует стандартные методы equals(), hashCode(), toString(), getters и setters для всех полей класса.
* @EqualsAndHashCode(callSuper = true): Генерирует методы equals() и hashCode(), включая сравнение суперкласса (в данном случае, сравнивает с AEntity).
* @Accessors(fluent = false, chain = true): Эта аннотация настраивает доступ к полям класса. fluent = false означает, что геттеры и сеттеры не будут иметь префикс “get” и “set”. chain = true позволяет вызывать сеттеры в цепочке (например, department.setName("HR").setBooks(books)).
* @Table(name = "department"): Указывает имя таблицы в базе данных, которая будет соответствовать этому классу.

2. Поля класса:

* name: Строковое поле, представляющее имя отдела.
* books: Множество книг, связанных с этим отделом через отношение “многие ко многим”.
* users: Список пользователей, связанных с этим отделом через отношение “один ко многим”.

2.1.6 Класс Feedback.



Класс Feedback представляющий сущность (Entity), описывает обратную связь или отзыв в системе. Опишем его поля и свойства:

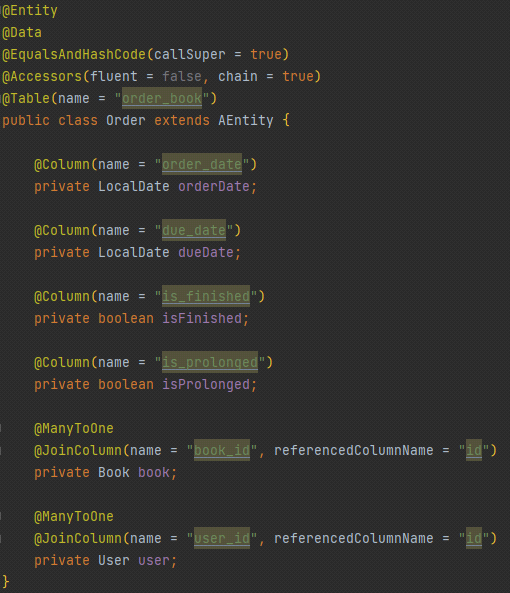
1. Аннотации:

* @Entity: Эта аннотация указывает, что класс Feedback является сущностью (entity) в базе данных. Такие сущности могут быть сохранены, извлечены и обновлены в базе данных.
* @Data: Эта аннотация генерирует стандартные методы equals(), hashCode(), toString(), getters и setters для всех полей класса.
* @EqualsAndHashCode(callSuper = true): Генерирует методы equals() и hashCode(), включая сравнение суперкласса (в данном случае, сравнивает с AEntity).
* @Accessors(fluent = false, chain = true): Эта аннотация настраивает доступ к полям класса. fluent = false означает, что геттеры и сеттеры не будут иметь префикс “get” и “set”. chain = true позволяет вызывать сеттеры в цепочке (например, feedback.setRating(5).setComment("Great book")).
* @Table(name = "feedback"): Указывает имя таблицы в базе данных, которая будет соответствовать этому классу.

2. Поля класса:

* rating: Целочисленное поле, представляющее оценку (рейтинг) отзыва.
* userName: Строковое поле, содержащее имя пользователя, оставившего отзыв.
* bookName: Строковое поле, представляющее название книги, к которой относится отзыв.
* comment: Строковое поле, содержащее текстовый комментарий или обратную связь.
* user: Связь с сущностью User через отношение “многие к одному”. Поле user\_id связывает отзыв с конкретным пользователем.
* book: Связь с сущностью Book через отношение “многие к одному”. Поле book\_id связывает отзыв с конкретной книгой.

2.1.7 Класс Order.



Класс Order представляющий сущность (Entity), описывает заказ книги в системе. Опишем кратко его поля и свойства:

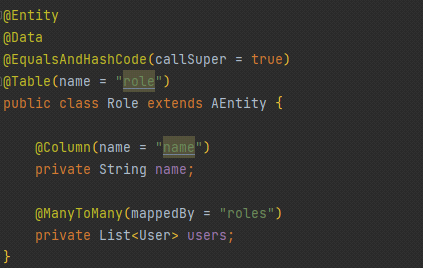
1. Аннотации:

* @Entity: Эта аннотация указывает, что класс Order является сущностью (entity) в базе данных. Такие сущности могут быть сохранены, извлечены и обновлены в базе данных.
* @Data: Эта аннотация генерирует стандартные методы equals(), hashCode(), toString(), getters и setters для всех полей класса.
* @EqualsAndHashCode(callSuper = true): Генерирует методы equals() и hashCode(), включая сравнение суперкласса (в данном случае, сравнивает с AEntity).
* @Accessors(fluent = false, chain = true): Эта аннотация настраивает доступ к полям класса. fluent = false означает, что геттеры и сеттеры не будут иметь префикс “get” и “set”. chain = true позволяет вызывать сеттеры в цепочке (например, order.setOrderDate(LocalDate.now()).setDueDate(LocalDate.now().plusDays(7))).
* @Table(name = "order\_book"): Указывает имя таблицы в базе данных, которая будет соответствовать этому классу.

2. Поля класса:

* orderDate: Поле типа LocalDate, представляющее дату размещения заказа.
* dueDate: Поле типа LocalDate, представляющее дату, когда заказ должен быть выполнен.
* isFinished: Булево поле, указывающее, завершен ли заказ.
* isProlonged: Булево поле, указывающее, был ли заказ продлен.
* book: Связь с сущностью Book через отношение “многие к одному”. Поле book\_id связывает заказ с конкретной книгой.
* user: Связь с сущностью User через отношение “многие к одному”. Поле user\_id связывает заказ с конкретным пользователем.

2.1.8 Класс Role.



Класс Role представляющий сущность (Entity), описывает роль в системе. Вот краткое описание его полей и свойств:

1. Аннотации:

* @Entity: Эта аннотация указывает, что класс Role является сущностью (entity) в базе данных. Такие сущности могут быть сохранены, извлечены и обновлены в базе данных.
* @Data: Эта аннотация генерирует стандартные методы equals(), hashCode(), toString(), getters и setters для всех полей класса.
* @EqualsAndHashCode(callSuper = true): Генерирует методы equals() и hashCode(), включая сравнение суперкласса (в данном случае, сравнивает с AEntity).
* @Table(name = "role"): Указывает имя таблицы в базе данных, которая будет соответствовать этому классу.

2. Поля класса:

* name: Строковое поле, представляющее имя роли.
* users: Список пользователей, связанных с этой ролью через отношение “многие ко многим”.

2.1.9 Класс User.



Класс User представляющий сущность (Entity), описывает пользователя в системе. Рассмотрим кратко его параметры:

1. Аннотации:

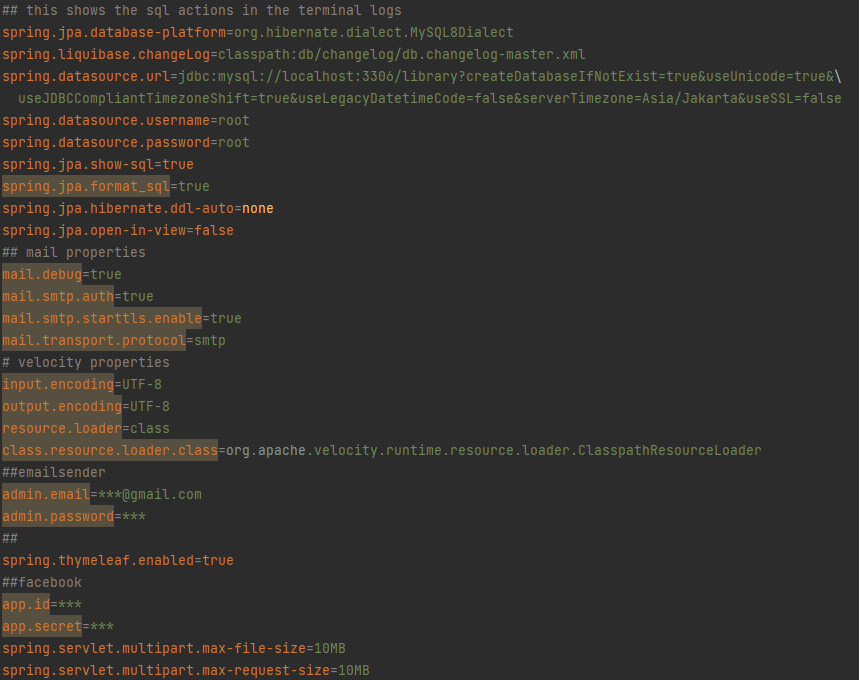
* @Entity: Эта аннотация указывает, что класс User является сущностью (entity) в базе данных. Такие сущности могут быть сохранены, извлечены и обновлены в базе данных.
* @Data: Эта аннотация генерирует стандартные методы equals(), hashCode(), toString(), getters и setters для всех полей класса.
* @EqualsAndHashCode(callSuper = true): Генерирует методы equals() и hashCode(), включая сравнение суперкласса (в данном случае, сравнивает с AEntity).
* @Accessors(fluent = false, chain = true): Эта аннотация настраивает доступ к полям класса. fluent = false означает, что геттеры и сеттеры не будут иметь префикс “get” и “set”. chain = true позволяет вызывать сеттеры в цепочке (например, user.setEmail("example@email.com").setUsername("JohnDoe")).
* @Table(name = "user"): Указывает имя таблицы в базе данных, которая будет соответствовать этому классу.

2. Поля класса:

* email: Строковое поле, представляющее адрес электронной почты пользователя.
* password: Строковое поле, содержащее хэшированный пароль пользователя.
* username: Строковое поле, представляющее имя пользователя.
* img: Строковое поле, содержащее ссылку на изображение профиля пользователя.
* department: Связь с сущностью Department через отношение “многие ко одному”. Поле department\_id связывает пользователя с конкретным отделом.
* roles: Связь с сущностью Role через отношение “многие ко многим”. Поле user\_role связывает пользователя с ролями.
* feedbacks: Список обратной связи (отзывов), связанных с пользователем через отношение “один ко многим”.
* orders: Список заказов, связанных с пользователем через отношение “один ко многим”.

2.1.10 Application properties

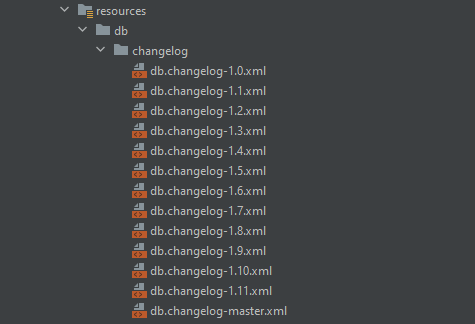
Далее перейдем к файлу конфигурации.

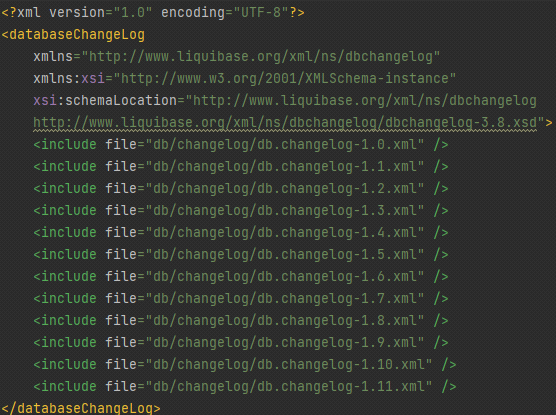


1. spring.jpa.database-platform: Указывает диалект базы данных для Hibernate (в данном случае, MySQL 8).
2. spring.liquibase.changeLog: Путь к файлу с изменениями базы данных, который используется для управления миграциями.
3. spring.datasource.url: URL-адрес базы данных MySQL, включая параметры подключения.
4. spring.datasource.username и spring.datasource.password: Имя пользователя и пароль для подключения к базе данных.
5. spring.jpa.show-sql и spring.jpa.format\_sql: Опции для отображения SQL-запросов и форматирования их вывода.
6. spring.jpa.hibernate.ddl-auto: Управление автоматической генерацией схемы базы данных.
7. spring.jpa.open-in-view: Опция для открытия сессии Hibernate при каждом HTTP-запросе.
8. mail.\*: Свойства для настройки отправки электронной почты.
9. spring.thymeleaf.enabled: Включение или отключение использования Thymeleaf для шаблонов.
10. app.id и app.secret: Идентификатор и секрет приложения Facebook.
11. spring.servlet.multipart.max-file-size и spring.servlet.multipart.max-request-size: Максимальные размеры загружаемых файлов.

2.1.11 Liquibase для управления изменениями базы данных.

В данном дипломном проекте используется Liquibase для управления версиями базы данных, что позволяет отслеживать, версионировать и применять изменения в базе данных в автоматизированном режиме.





Структура папок и файлов в проекте организована следующим образом:

* /resources/: Корневая папка ресурсов, содержащая все необходимые файлы для работы приложения.
* /db/: Папка, содержащая все скрипты и файлы, связанные с базой данных.
* /changelog/: Папка, в которой хранятся файлы изменений (changelogs), управляющие версиями базы данных.
* db.changelog-master.xml: Главный файл changelog, который включает ссылки на все остальные файлы changelog в папке. Этот файл служит точкой входа для Liquibase и определяет порядок применения изменений к базе данных.
* db.changelog-1.0.xml до db.changelog-1.11.xml: Файлы changelog, содержащие конкретные инструкции для изменения структуры базы данных, такие как создание таблиц, добавление столбцов, вставка данных и т.д.

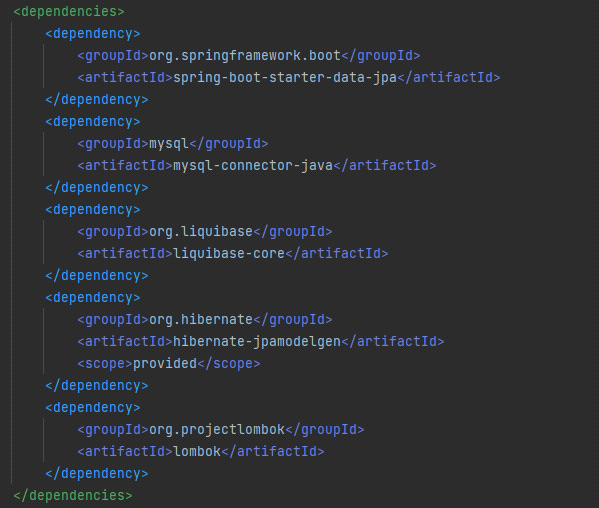
Когда Liquibase запускается, он сначала читает db.changelog-master.xml, который указывает на все остальные файлы changelog. Затем Liquibase последовательно применяет изменения из каждого файла в указанном порядке. Это позволяет разработчикам контролировать и управлять миграциями базы данных в соответствии с изменениями в коде приложения.

Использование такой структуры обеспечивает централизованное управление изменениями базы данных и гарантирует, что все разработчики и среды разработки используют одну и ту же структуру и версии базы данных. Это важно для поддержания целостности и согласованности данных во всем проекте. Скриншот этой структуры наглядно демонстрирует организацию версионирования базы данных в дипломном проекте.

2.2 Модуль API.

2.2.1 Maven зависимости.

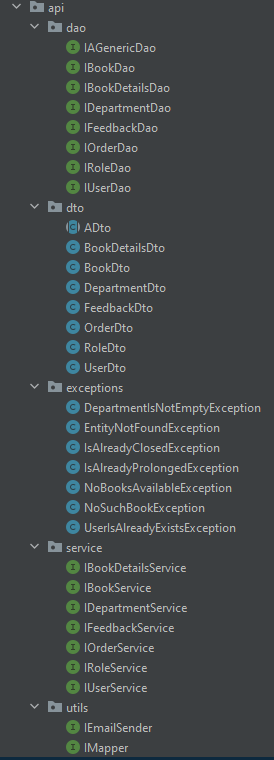
Для разработки модуля API мы подключаем следующие зависимости при помощи Maven:



1. spring-boot-starter-data-jpa: Эта зависимость предоставляет интеграцию Spring Data JPA с Hibernate.
2. mysql-connector-java: Это JDBC-драйвер для подключения к базе данных MySQL.
3. liquibase-core: Liquibase - это инструмент для управления изменениями схемы базы данных.
4. hibernate-jpamodelgen: зависимость используется для генерации метамодели JPA.
5. lombok: автоматически добавляет методы, такие как геттеры, сеттеры, билдеры и логгеры, к классам.

2.2.2 Иерархия и функциональность модуля API.

Модуль API: Центральный элемент архитектуры



Модуль API в моем проекте выполняет критически важную функцию, служа связующим звеном между различными слоями приложения. Он определяет интерфейсы, которые гарантируют строгую абстракцию и контракты между слоями DAO и сервисами. Это обеспечивает следующие преимущества:

* Согласованность: Интерфейсы API обеспечивают единообразие взаимодействия между компонентами, что упрощает интеграцию и тестирование.
* Гибкость: Благодаря четко определенным интерфейсам, можно легко заменять или обновлять отдельные компоненты без влияния на остальную часть системы.
* Масштабируемость: API поддерживает расширение функциональности приложения, позволяя добавлять новые слои или сервис с минимальными изменениями в существующем коде.

Исключения: Определение исключений в модуле API позволяет стандартизировать обработку ошибок по всему приложению. Это улучшает отладку и поддержку, а также предоставляет более чистый и понятный код.

DTO (Data Transfer Objects): DTO используются для передачи данных между слоями и клиентами. Они упрощают передачу данных, минимизируя количество вызовов методов и уменьшая связность.

В целом, модуль API является фундаментальным для обеспечения чистой, модульной и легко поддерживаемой архитектуры приложения. Он способствует созданию гибкой и расширяемой системы, которая может адаптироваться к изменяющимся требованиям и масштабироваться по мере роста проекта.

2.3 Модуль Utils.

2.3.1 Класс MailConfiguration.

В моем дипломном проекте модуль утилит, представленный классом MailConfiguration, играет ключевую роль в настройке и управлении отправкой электронной почты и шаблонизации контента.

Класс MailConfiguration использует аннотацию @Configuration для определения настроек, необходимых для работы с электронной почтой.

С помощью аннотации @Value извлекаются параметры из файла свойств, такие как mail.debug, mail.smtp.auth, mail.smtp.starttls.enable, и другие, которые определяют поведение почтового клиента.

Аннотация @Slf4j предоставляет логгер для класса, позволяя регистрировать информацию о процессе выполнения, что облегчает отладку и мониторинг.

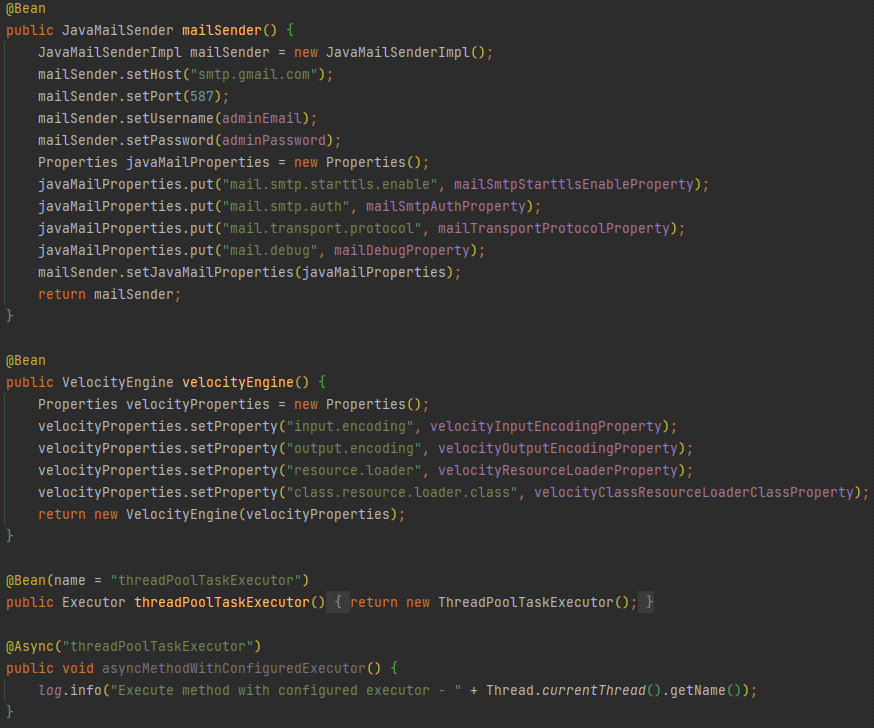
Аннотация @EnableAsync включает асинхронную обработку в приложении, позволяя выполнять методы асинхронно с помощью конфигурируемого исполнителя задач.



Метод mailSender() создает и настраивает экземпляр JavaMailSenderImpl, который отвечает за отправку электронных писем. Этот метод устанавливает хост, порт, имя пользователя и пароль для доступа к почтовому серверу, а также конфигурирует дополнительные свойства, такие как использование TLS и отладочный режим.

Для генерации контента писем используется VelocityEngine, который настраивается через метод velocityEngine(). Этот метод задает параметры кодировки ввода и вывода, а также настройки загрузчика ресурсов, что позволяет использовать шаблоны Velocity для создания динамического контента.

Метод threadPoolTaskExecutor() создает пул потоков, который используется для асинхронного выполнения задач, таких как отправка электронной почты, не блокируя основной поток приложения.

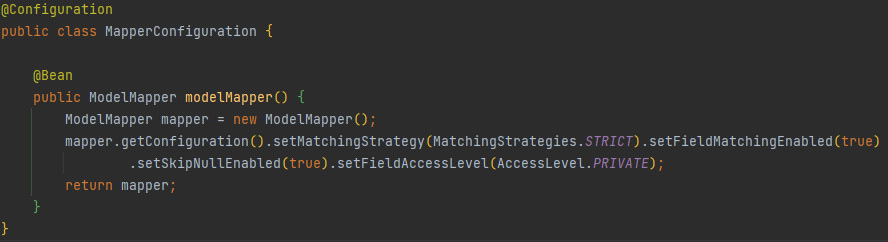


Этот модуль утилит обеспечивает гибкость и масштабируемость функционала почтовых уведомлений и шаблонизации, что является важным аспектом современных приложений.

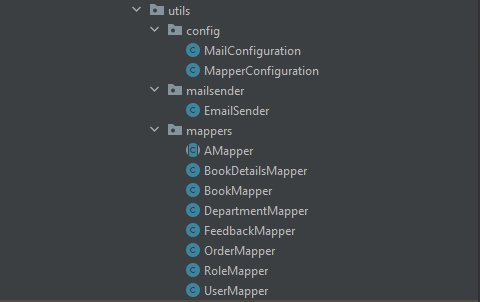
2.3.2 Класс MapperConfiguration.

Класс MapperConfiguration, аннотированный как @Configuration, отвечает за создание и настройку компонента ModelMapper, который является центральным элементом для преобразования и маппинга данных между объектами различных слоев приложения.

* ModelMapper — это мощный инструмент для автоматического маппинга одних объектов в другие, что значительно упрощает процесс трансформации данных в многослойных приложениях. В классе MapperConfiguration метод modelMapper() создает экземпляр ModelMapper и конфигурирует его следующим образом:
* Установка стратегии соответствия MatchingStrategies.STRICT гарантирует, что поля между объектами будут сопоставлены только в случае полного соответствия имен и типов данных, что повышает точность маппинга.
* Включение setFieldMatchingEnabled(true) позволяет ModelMapper сопоставлять поля между объектами, даже если они являются частными (private), что обеспечивает гибкость в работе с различными моделями данных.
* Опция setSkipNullEnabled(true) предотвращает копирование null значений в целевой объект, что помогает избежать нежелательного перезаписывания полей с уже установленными значениями.
* Установка setFieldAccessLevel(AccessLevel.PRIVATE) позволяет ModelMapper работать с приватными полями без необходимости создания геттеров и сеттеров, что упрощает структуру классов.



В модуле также представлены классы, которые используют ModelMapper для стандартного маппинга сущностей и более сложных сценариев с вложенными объектами. Это позволяет разработчикам легко преобразовывать данные между слоями DAO, сервисами и контроллерами, а также обрабатывать сложные структуры данных с множественными уровнями вложенности. Использование ModelMapper значительно упрощает процесс разработки, уменьшая количество шаблонного кода и ускоряя процесс интеграции различных слоев приложения. Это делает систему более гибкой и легко масштабируемой, а также облегчает поддержку и обновление кода.



2.3.3 Класс EmailSender.

В этом проекте я реализовал компонент EmailSender, который предоставляет функционал для асинхронной отправки электронных писем различного содержания пользователям и администраторам. Далее будет описан функционал, который я добавил в приложение.

Компонент EmailSender использует аннотацию @Async для асинхронной отправки электронных писем, что позволяет приложению продолжать обработку других задач, в то время как сообщения отправляются в фоновом режиме. Это улучшает производительность и отзывчивость системы.

Метод sendEmailsFromAdminAboutNewBook отправляет письма всем пользователям определенного отдела с информацией о новых книгах, добавленных в библиотеку. Это способствует повышению осведомленности пользователей о новых ресурсах.

Метод sendEmailsFromAdminAboutDebts уведомляет пользователей о необходимости возврата книг, что помогает поддерживать порядок в учете книг и предотвращает накопление задолженностей.

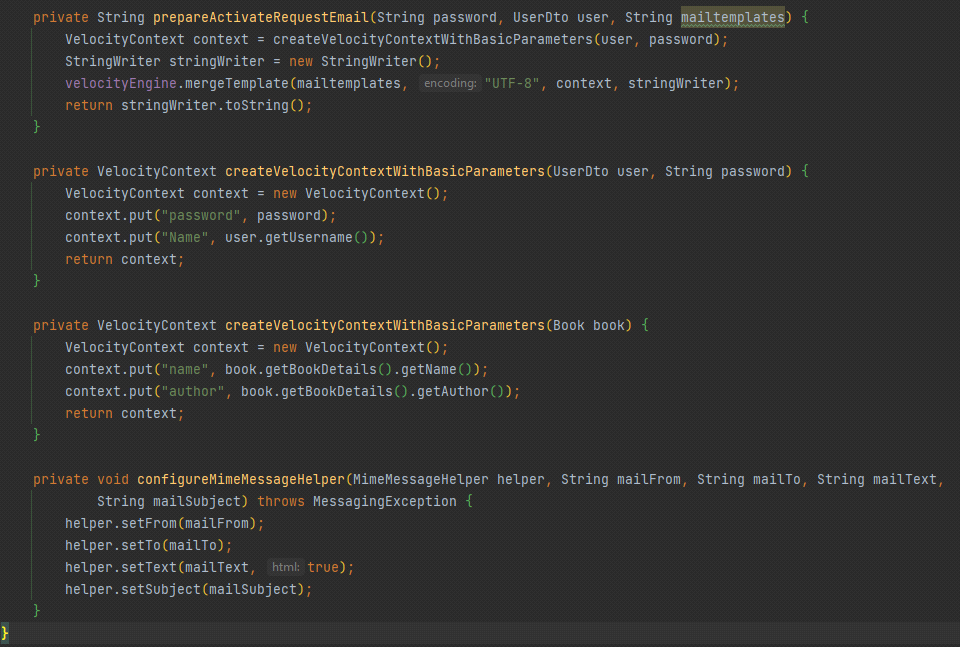
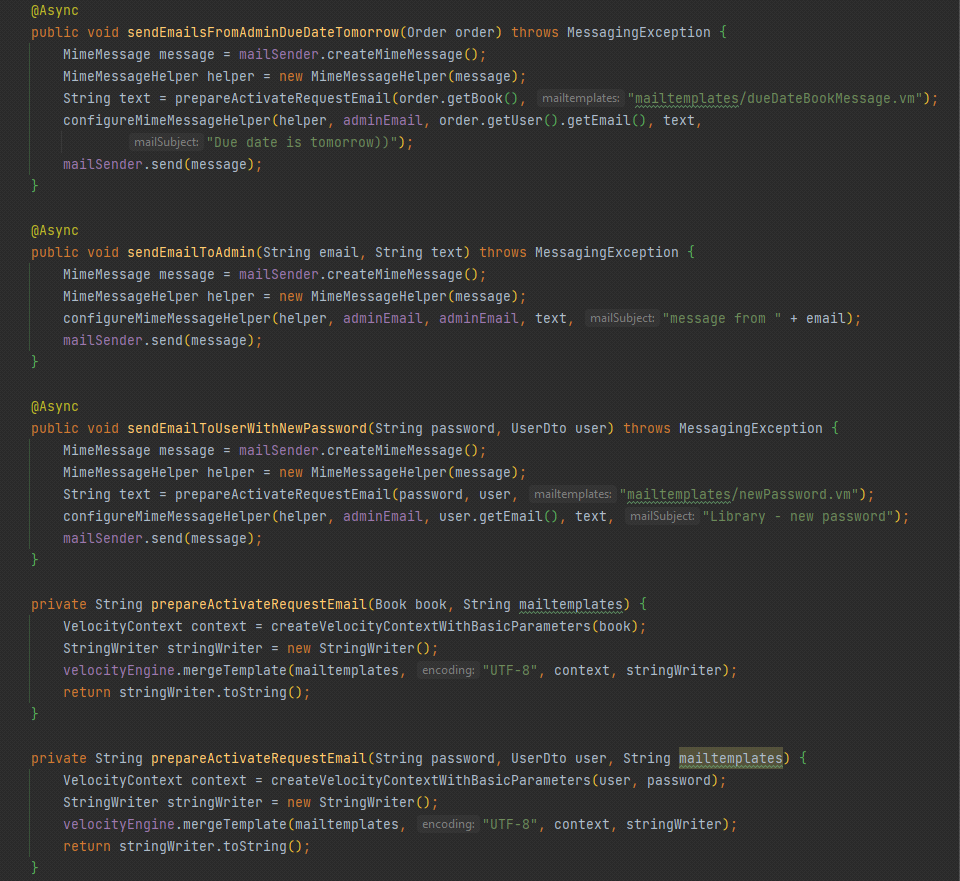
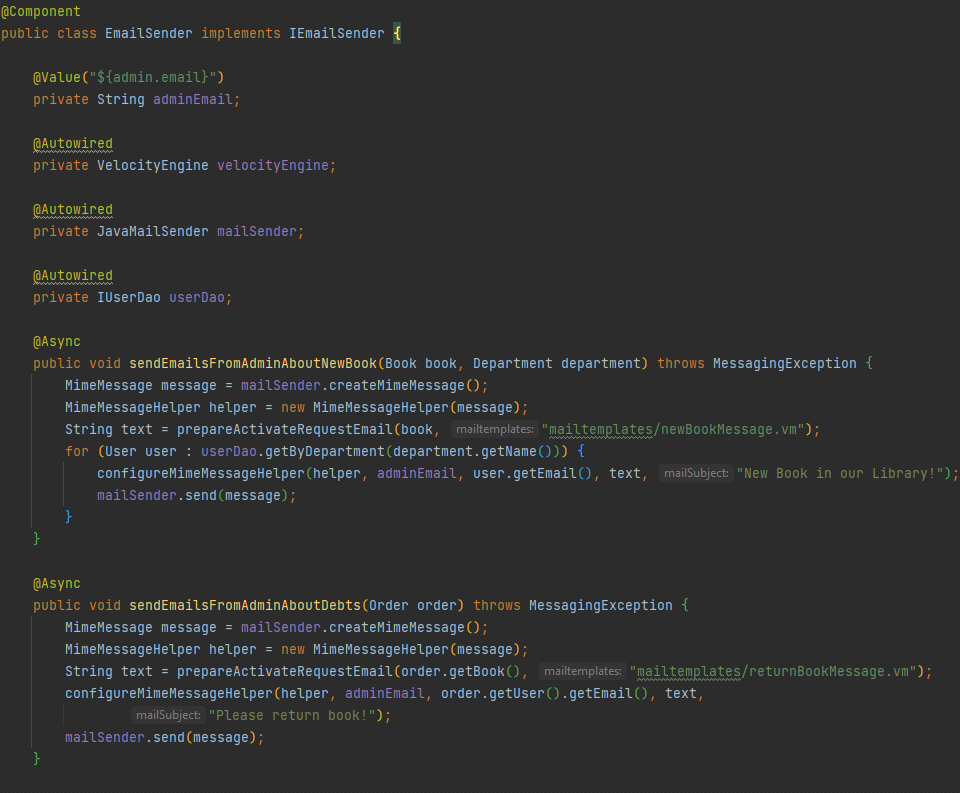
Метод sendEmailsFromAdminDueDateTomorrow предупреждает пользователей о скором истечении срока возврата книг, что является важным для поддержания дисциплины среди читателей.

Метод sendEmailToAdmin позволяет пользователям отправлять сообщения администратору, что упрощает процесс коммуникации и ускоряет решение возникающих вопросов.

Метод sendEmailToUserWithNewPassword используется для отправки нового пароля пользователям, что обеспечивает безопасность доступа к учетным записям и помогает в случаях, когда пользователи забывают свои пароли.

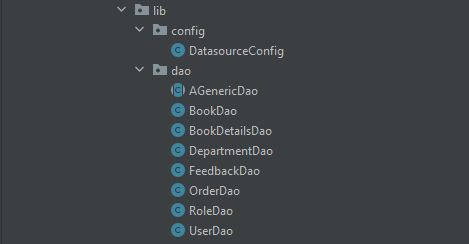
Для создания содержимого писем используется VelocityEngine, который позволяет генерировать персонализированные сообщения на основе шаблонов. Это делает каждое письмо более личным и релевантным для получателя.

Метод configureMimeMessageHelper настраивает параметры сообщения, такие как отправитель, получатель, текст и тема письма, что гарантирует правильную доставку и читаемость сообщений. Этот комплексный функционал по отправке электронных писем значительно улучшает взаимодействие с пользователями и управление библиотечными процессами, делая систему более эффективной и удобной для пользователей.

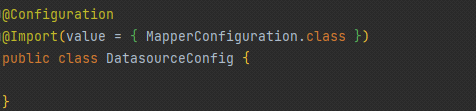


2.4 Модуль DAO.

Далее в моем дипломном проекте реализован модуль DAO (Data Access Object), который представляет собой ключевой компонент архитектуры приложения, отвечающий за доступ к данным в базе данных.



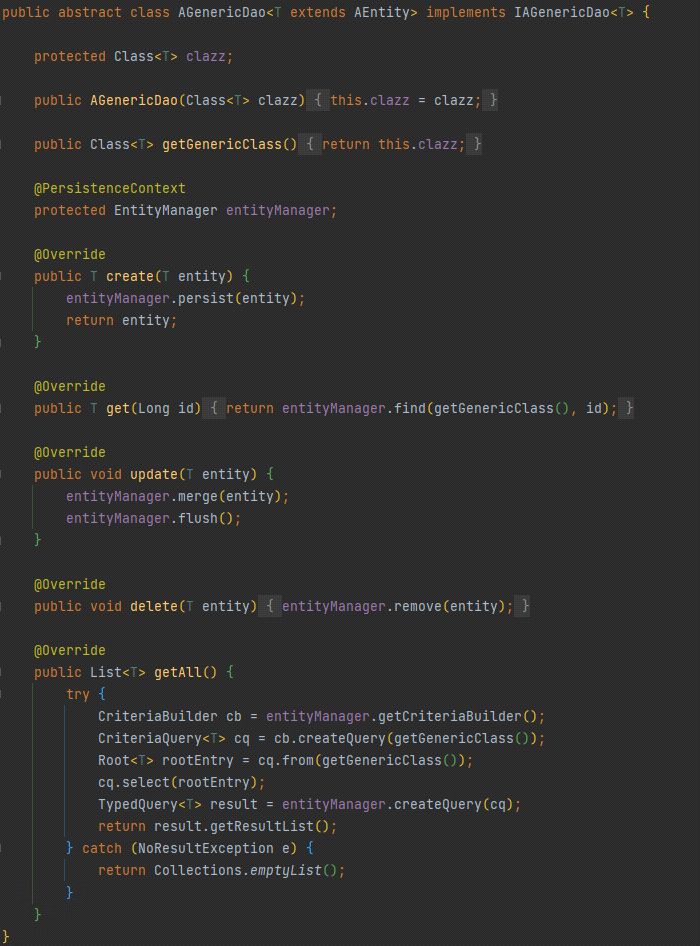
Класс DatasourceConfig в этом модуле является конфигурационным классом.



Назначение: Этот класс предоставляет конфигурацию, связанную с источниками данных.

Фактические детали конфигурации будут определены в импортированном конфигурационном классе (MapperConfiguration).

Класс AGenericDao является абстрактным классом, который определяет общие операции CRUD (создание, чтение, обновление, удаление) для всех сущностей приложения.



Класс AGenericDao использует обобщенный тип T, который расширяет AEntity, позволяя использовать один и тот же класс для работы с различными сущностями. Это обеспечивает универсальность и гибкость в работе с данными разных типов.

Методы create, get, update, и delete предоставляют стандартный набор операций для управления сущностями в базе данных. Они позволяют добавлять новые записи, извлекать существующие, обновлять их и удалять, соответственно.

Метод getAll использует Criteria API для формирования запроса, который извлекает все записи данного типа сущности из базы данных. Это позволяет легко получить полный список объектов для дальнейшей обработки.

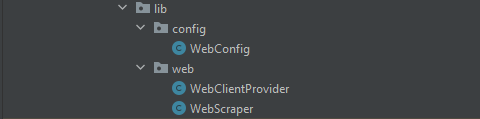
Аннотация @PersistenceContext внедряет EntityManager, который является основным инструментом JPA для взаимодействия с базой данных. Он предоставляет методы для выполнения операций с сущностями и управления транзакциями.

Класс AGenericDao может быть расширен для создания конкретных DAO классов для каждой сущности. Это позволяет добавлять специфические методы доступа к данным, которые необходимы для определенных бизнес-задач.

Для каждого конкретного DAO класса можно реализовать дополнительные методы, использующие CriteriaBuilder для создания сложных запросов с фильтрацией, сортировкой и агрегацией данных, что увеличивает эффективность работы с данными. Этот модуль DAO обеспечивает централизованный и структурированный подход к доступу к данным в приложении, что упрощает разработку, обеспечивает согласованность кода и улучшает поддержку приложения.

2.5 Модуль WEB.

Веб-модуль в приложении отвечает за взаимодействие с пользователем через веб-интерфейс. Он позволяет пользователям взаимодействовать с приложением через браузер, выполнять действия, получать информацию и обновлять данные. В данном приложении веб-модуль имеет следующую иерархию:

****

2.5.1 Класс WebClientProvider.

Класс WebClientProvider представляет собой утилиту для создания и настройки экземпляра WebClient, который используется для взаимодействия с веб-страницами. Функционала этого класса будет представлен далее.



WebClientProvider содержит статический метод getDefaultWebClient, который инициализирует WebClient с настройками для браузера Chrome. Это обеспечивает стандартизированное взаимодействие с веб-страницами, имитируя поведение популярного веб-браузера.

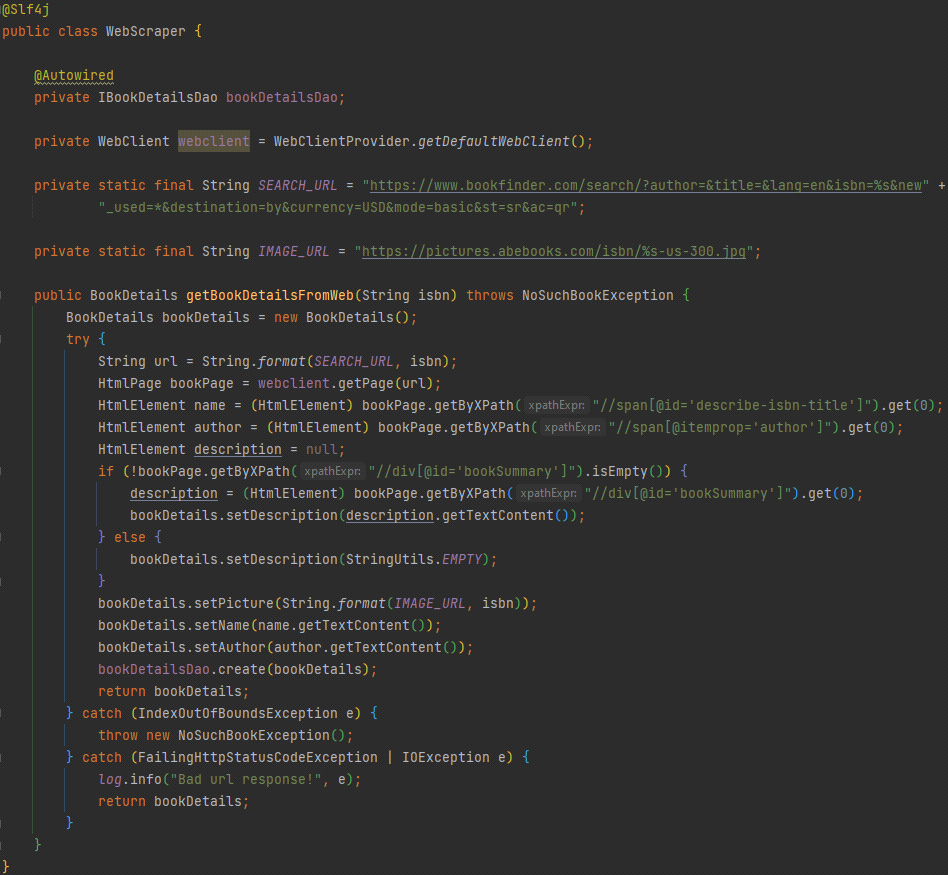
Метод setWebClientDefaultOptions конфигурирует WebClient с определенным набором параметров:

* Установка NicelyResynchronizingAjaxController позволяет WebClient более гладко обрабатывать AJAX-запросы, синхронизируя асинхронные вызовы.
* Отключение исключений для скриптовых ошибок и ошибок статуса HTTP уменьшает вероятность прерывания выполнения из-за несущественных ошибок.
* Включение перенаправлений позволяет WebClient автоматически следовать за HTTP-редиректами.
* Включение поддержки JavaScript позволяет WebClient исполнять скрипты на странице, что необходимо для полноценного взаимодействия с современными веб-сайтами.
* Отключение CSS ускоряет загрузку страниц, так как стили не применяются, что может быть полезно при работе с данными страницы, а не с её визуальным представлением.
* Метод waitForBackgroundJavaScript устанавливает время ожидания выполнения JavaScript, что позволяет дождаться завершения всех скриптов перед взаимодействием со страницей.

Приватный конструктор класса WebClientProvider предотвращает создание экземпляров этого класса, тем самым обеспечивая использование только статических методов. Этот класс является важным инструментом для работы с веб-страницами в данном приложении, позволяя эффективно извлекать и обрабатывать данные, необходимые для выполнения бизнес-логики.

2.5.2 Класс WebScraper .

Класс WebScraper представляет собой инструмент для автоматизированного извлечения данных о книгах с веб-сайтов.



WebScraper использует WebClient для доступа к веб-страницам и извлечения информации о книгах по их ISBN. Это позволяет автоматизировать процесс сбора данных, что значительно ускоряет работу с большим объемом информации.

Класс применяет XPath запросы для поиска и выборки конкретных элементов на странице, таких как название книги, автор и описание. Это обеспечивает точность и гибкость при работе с разметкой страницы.

WebScraper обрабатывает различные исключения, такие как IndexOutOfBoundsException, которое может возникать, если ожидаемые данные отсутствуют на странице, и FailingHttpStatusCodeException или IOException, указывающие на проблемы с доступом к URL.

Полученная информация используется для создания объектов BookDetails, которые затем сохраняются в базе данных с помощью bookDetailsDao. Это позволяет эффективно управлять данными о книгах в приложении.

Класс формирует URL для изображений книг, используя шаблонный литерал, что позволяет легко получить доступ к обложкам книг для дальнейшего использования в приложении.

Использование IBookDetailsDao для сохранения собранных данных обеспечивает интеграцию WebScraper с остальной частью системы, позволяя другим компонентам приложения использовать актуальные данные о книгах. Этот функционал делает WebScraper ценным инструментом для автоматизации процессов сбора и управления данными о книгах в приложении.

2.6 Модуль Controller.

2.6.1 CustomAccessDeniedHandler.

Класс CustomAccessDeniedHandler реализует интерфейс AccessDeniedHandler, предоставляемый Spring Security, для обработки ситуаций, когда пользователю отказано в доступе к определенным ресурсам.



Описание роли и функций CustomAccessDeniedHandler в контексте безопасности приложения:

CustomAccessDeniedHandler переопределяет метод handle, который вызывается, когда пользователь пытается получить доступ к защищенному URL без необходимых прав. Это позволяет приложению корректно реагировать на попытки неавторизованного доступа.

Если аутентификация пользователя была успешной, но у него нет прав доступа к запрашиваемому ресурсу, класс записывает информацию о такой попытке в лог. Это включает имя пользователя и URL, к которому он пытался получить доступ, что помогает в мониторинге безопасности приложения.

После обнаружения отказа в доступе, CustomAccessDeniedHandler перенаправляет пользователя на страницу с кодом ошибки 403, что является стандартной практикой для информирования пользователя о том, что у него нет прав для просмотра данного ресурса.

Использование SecurityContextHolder для получения информации об аутентификации пользователя показывает, как класс интегрируется с остальной частью системы безопасности Spring, обеспечивая единообразную и безопасную обработку доступа. Этот класс является важной частью системы безопасности приложения, обеспечивая надежную обработку случаев, когда пользователям необходимо сообщить о недостаточных правах доступа.

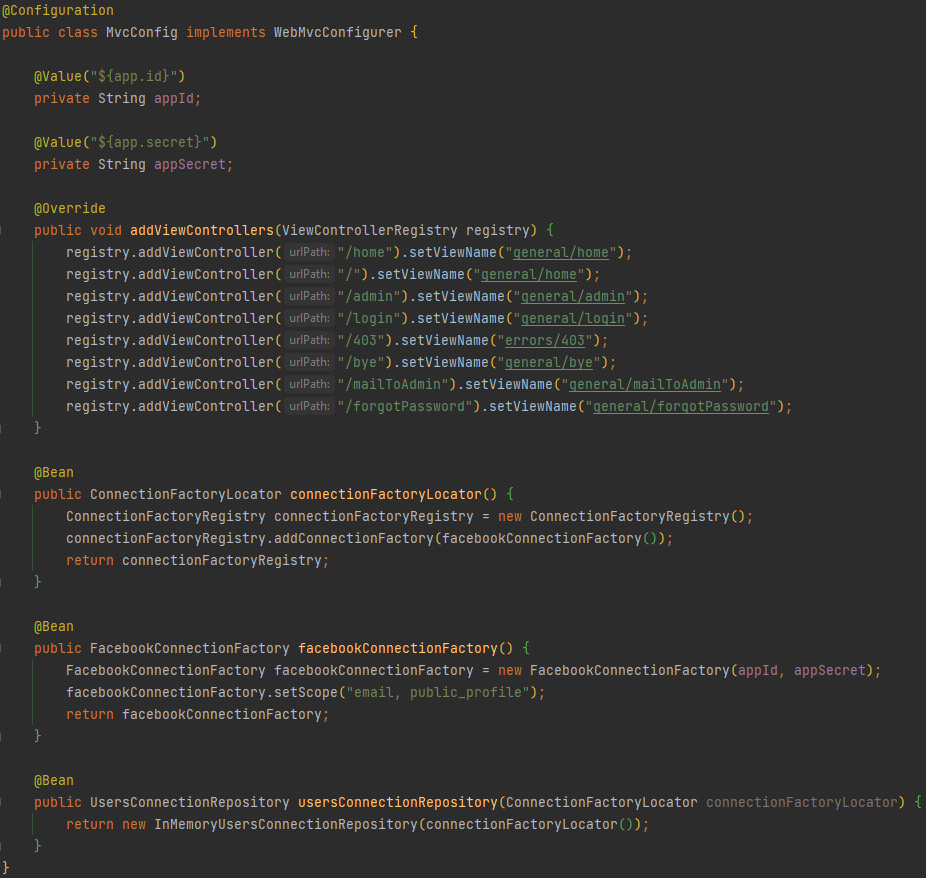
2.6.2 Класс MvcConfig.

Класс MvcConfig, аннотированный как @Configuration, выполняет роль конфигурационного компонента в Spring MVC приложении.

Метод addViewControllers используется для регистрации простых контроллеров, которые связывают URL-адреса с именами представлений без необходимости реализации логики на стороне сервера. Это упрощает навигацию по приложению и улучшает его структуру.

С помощью FacebookConnectionFactory, MvcConfig настраивает подключение к Facebook API, используя идентификатор приложения и секретный ключ. Это позволяет пользователям использовать свои учетные записи Facebook для входа в приложение.

UsersConnectionRepository предоставляет механизм для управления соединениями между учетными записями пользователей в приложении и их учетными записями в социальных сетях, что облегчает процесс аутентификации и авторизации.

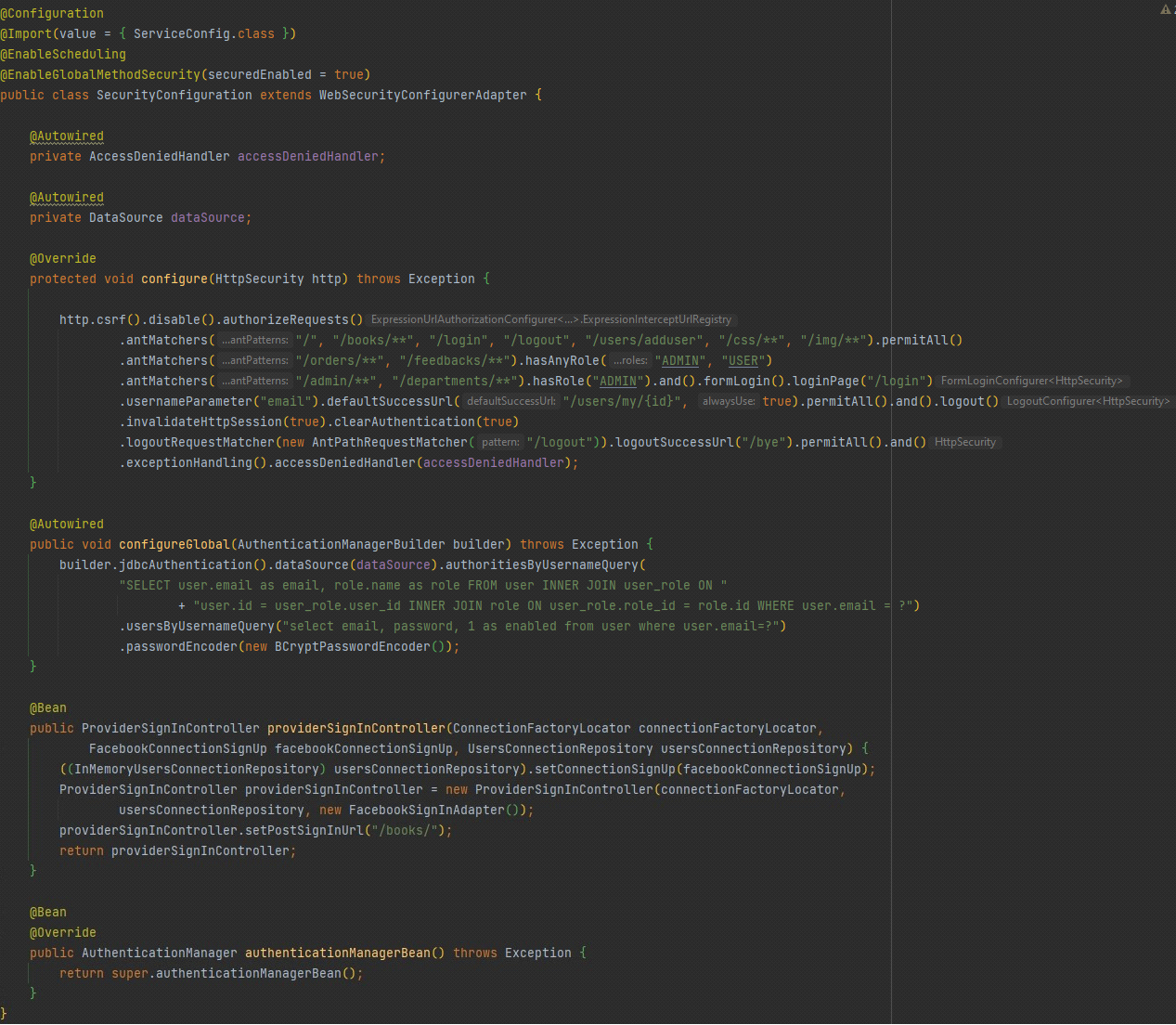


Класс определяет перенаправления для стандартных маршрутов, таких как /login и /403, обеспечивая корректную обработку сценариев входа в систему и отказа в доступе.

Использование аннотации @Value для извлечения параметров конфигурации из внешних источников делает MvcConfig гибким и легко настраиваемым компонентом, что упрощает управление конфигурацией приложения. Этот класс является центральным элементом в конфигурации Spring MVC приложения, обеспечивая необходимую инфраструктуру для управления представлениями, безопасностью и интеграцией с внешними сервисами.

2.6.3 Класс SecurityConfiguration.

Класс SecurityConfiguration, аннотированный как @Configuration, является частью системы безопасности в Spring Framework и расширяет WebSecurityConfigurerAdapter, предоставляя настройки безопасности для веб-приложения.



Метод configure(HttpSecurity http) определяет правила безопасности для различных URL-путей в приложении. Он отключает CSRF защиту и настраивает правила доступа:

* Пути, такие как корневой (/), /books/, /login, /logout, /users/adduser, /css/, /img/ доступны всем пользователям.
* Пути /orders/, /feedbacks/ доступны пользователям с ролями ADMIN или USER.
* Пути /admin/, /departments/ доступны только пользователям с ролью ADMIN.

Форма входа и выхода: Конфигурация включает настройку страницы входа, параметров аутентификации и перенаправления после успешного входа или выхода из системы.

Обработка исключений доступа: В случае отказа в доступе, система использует accessDeniedHandler для перенаправления пользователя на страницу ошибки 403.

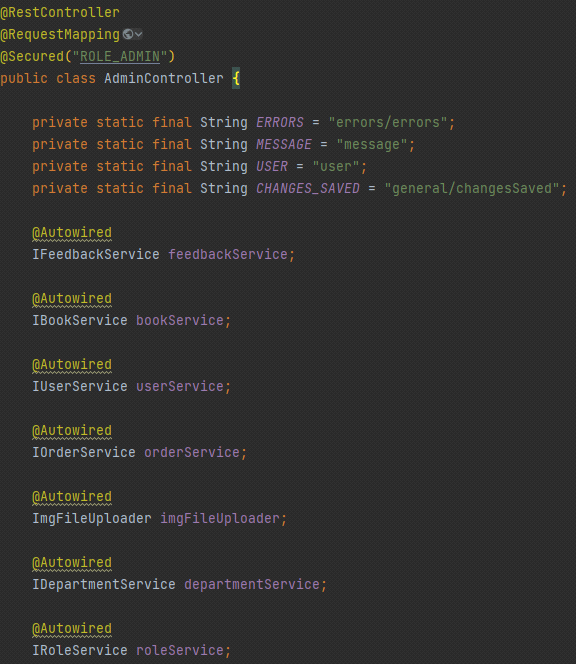
Глобальная конфигурация аутентификации: Метод configureGlobal(AuthenticationManagerBuilder builder) настраивает аутентификацию через JDBC, используя источник данных для получения информации о пользователях и их ролях, а также настраивает BCryptPasswordEncoder для шифрования паролей.

Интеграция с социальными сетями: ProviderSignInController позволяет пользователям регистрироваться и входить в систему через Facebook, используя ConnectionFactoryLocator и UsersConnectionRepository для управления соединениями.

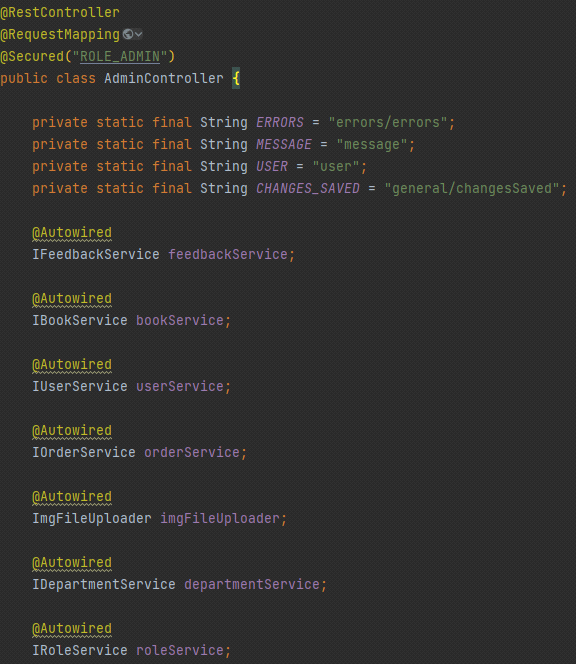
Предоставление AuthenticationManager: Переопределенный метод authenticationManagerBean() предоставляет бин AuthenticationManager, который используется Spring Security для обработки аутентификации. Этот класс обеспечивает комплексную настройку безопасности веб-приложения, включая аутентификацию, авторизацию, обработку исключений и интеграцию с внешними сервисами аутентификации.

2.6.4 AdminController.

Контроллер AdminController обеспечивает управление ключевыми аспектами административной части приложения.



Методы deleteBook и deletebookSubmit позволяют администратору удалять книги из системы, предоставляя форму для подтверждения удаления и обрабатывая сам запрос на удаление.



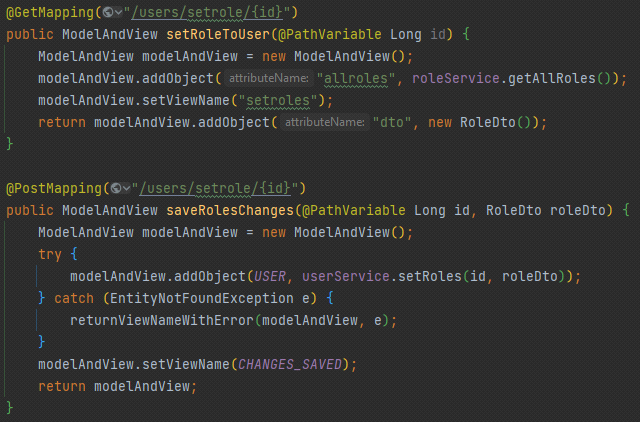
Метод getAllUsers выводит список всех пользователей, а getUserProfile предоставляет детальный профиль отдельного пользователя, включая его заказы.



Методы getUserEditForm и saveUsersChanges позволяют редактировать информацию о пользователях и сохранять изменения, включая загрузку аватаров через ImgFileUploader.



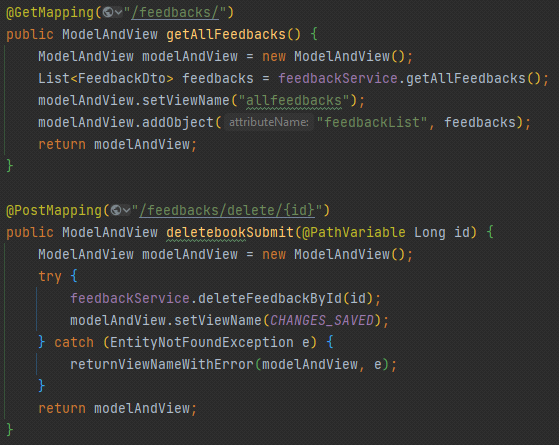
Методы setRoleToUser и saveRolesChanges используются для назначения ролей пользователям, улучшая управление доступом в приложении.



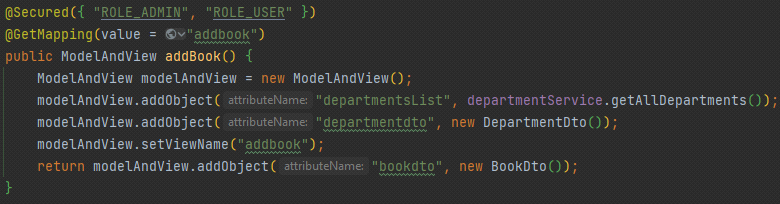
Метод getAllOrders отображает список всех заказов в системе.



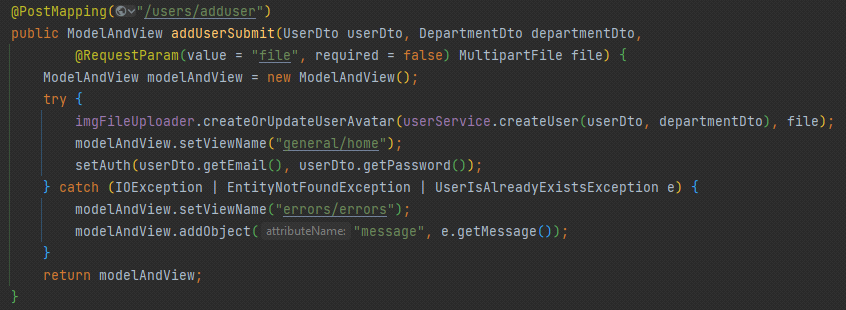
Методы getAllFeedbacks и deletebookSubmit предоставляют возможность просмотра и удаления отзывов пользователей.



Метод addBook предоставляет форму для добавления новых книг в систему.



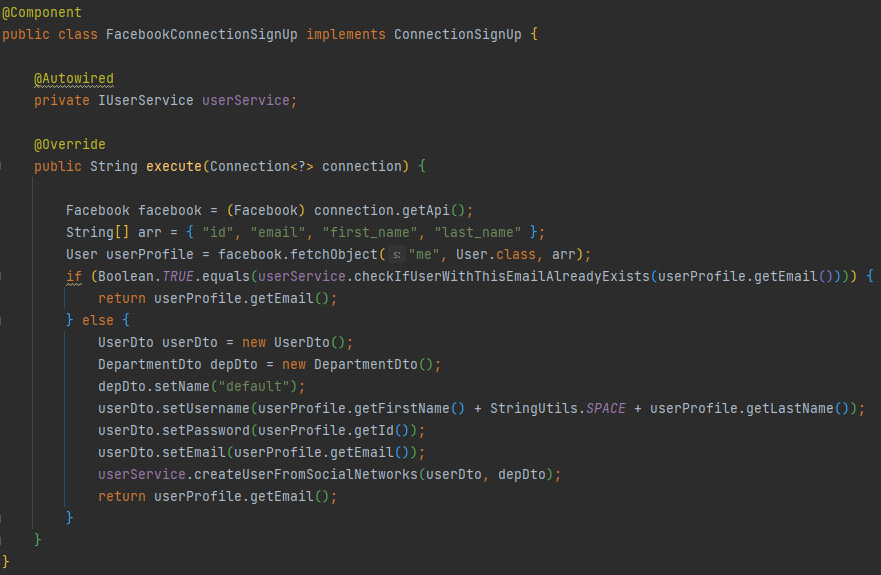
Метод addUserSubmit обрабатывает создание новых пользовательских аккаунтов и загрузку аватаров.



Эти контроллеры играют важную роль в управлении контентом и пользователями, обеспечивая администратору мощные инструменты для поддержания порядка и эффективности работы приложения.

2.6.5 Класс FacebookConnectionSignUp.

Класс FacebookConnectionSignUp реализует интерфейс ConnectionSignUp, который используется для регистрации новых пользователей через социальную сеть Facebook.



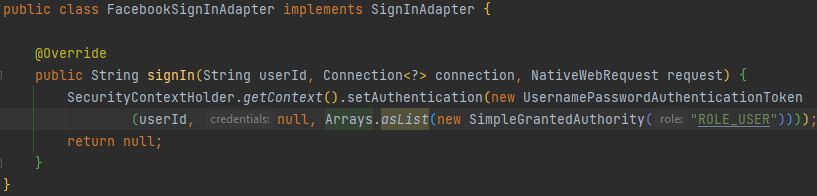
С помощью аннотации @Autowired, класс автоматически внедряет сервис IUserService, который предоставляет методы для работы с пользователями.

Далее в работу вступает переопределенный метод execute. Этот метод вызывается при попытке регистрации пользователя через Facebook. Он получает API соединения с Facebook и извлекает профиль пользователя.Метод проверяет, существует ли уже пользователь с таким же email. Если да, то возвращает email в качестве идентификатора для входа. Если пользователь с таким email не найден, метод создает новый объект UserDto, заполняет его данными из профиля Facebook, включая имя, фамилию и ID в качестве пароля, а также создает объект DepartmentDto с названием "default". Затем он регистрирует нового пользователя с помощью метода createUserFromSocialNetworks.

В конце метод возвращает email пользователя, который может быть использован для входа в систему. Этот класс облегчает интеграцию регистрации и входа через Facebook, автоматизируя процесс создания учетной записи на основе данных профиля пользователя в Facebook.

2.6.6 Класс FacebookSignInAdapter.

Класс FacebookSignInAdapter реализует интерфейс SignInAdapter, который используется в Spring Framework для интеграции входа через социальные сети.



Переопределяем метод signIn. Этот метод вызывается при успешном входе пользователя через социальную сеть. Он принимает идентификатор пользователя (userId), соединение (connection) и веб-запрос (request). Внутри метода signIn, вызывается SecurityContextHolder.getContext().setAuthentication(), чтобы установить аутентификацию пользователя в контексте безопасности. Пользователю назначается роль ROLE\_USER с помощью SimpleGrantedAuthority. Это определяет базовые права доступа пользователя в приложении. Метод возвращает null, что обычно означает, что процесс входа управляется внешним редиректом, а не внутренней логикой.

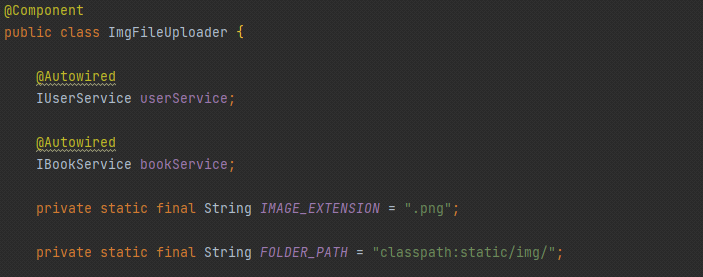
Этот класс обеспечивает безопасный вход пользователя через Facebook, автоматически устанавливая необходимые учетные данные и права доступа.

2.6.7 Класс ImgFileUploader.

Класс ImgFileUploader предназначен для загрузки и обновления изображений в веб-приложении.

С помощью аннотации @Autowired, класс автоматически внедряет сервисы IUserService и IBookService, которые предоставляют методы для работы с пользователями и книгами соответственно.

Класс определяет константы IMAGE\_EXTENSION и FOLDER\_PATH, которые указывают расширение файлов изображений и путь к папке, где они хранятся.



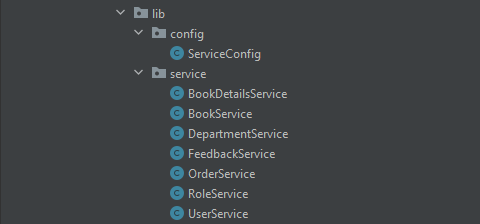
Для выполнения своих функций, ImgFileUploader реализует следующие методы:

* createOrUpdateUserAvatar(UserDto dto, MultipartFile file): Загружает и обновляет аватар пользователя, если файл предоставлен и не пуст.
* createOrUpdateBookCover(BookDto dto, MultipartFile file): Загружает и обновляет обложку книги, используя ISBN книги как имя файла
* setImgInDB(MultipartFile file, UserDto dto): Кодирует изображение в формат Base64 и обновляет аватар пользователя в базе данных.
* createOrUpdate(MultipartFile file, String futureImgName): Сохраняет файл на диск, обрабатывая исключения, связанные с нахождением файла.



Этот класс является ключевым компонентом системы управления изображениями в веб-приложении, позволяя легко загружать и обновлять изображения для пользователей и книг. Он обеспечивает интеграцию с файловой системой и базой данных, что делает его важным элементом для данного веб-приложения. ImgFileUploader демонстрирует реализацию функционала работы с мультимедийными данными в современных веб-приложениях.

2.7 Service.

****

В моем проекте, я создал многоуровневую архитектуру, где каждый объект представлен в соответствующем слое сервиса классом, инкапсулирующим бизнес-логику. Это позволяет четко разграничить ответственность между различными компонентами системы и обеспечивает гибкость и масштабируемость приложения.

3. Тестирование приложения в среде разработки и UI-тестирование.

3.1 Наборы шаблонов и стилей для фронта.

В моем проекте для библиотеки используются HTML-шаблоны для отправки различных уведомлений пользователям. Далее сделаю краткое описание представленных шаблонов:

1. Шаблон напоминания о возврате книги:

Этот шаблон предназначен для отправки уведомления о том, что срок возврата книги истекает. В сообщении указывается название книги и автор, а также просьба не забыть вернуть или продлить срок пользования книгой.

2. Шаблон уведомления о новой книге:

Шаблон информирует пользователя о добавлении новой книги в библиотеку. В нём содержится название и автор новой книги.

3. Шаблон уведомления с новым паролем:

Этот шаблон используется для отправки пользователям нового пароля. В сообщении приветствуют пользователя по имени и предоставляют новый пароль.

4. Шаблон просьбы о возврате книги:

Шаблон содержит просьбу вернуть книгу в библиотеку с указанием названия и автора книги.

Каждый шаблон начинается с приветствия и заканчивается благодарностью. В шаблонах используются переменные ${name}, ${author}, ${Name}, и ${password} для динамической подстановки соответствующих данных. Эти шаблоны хранятся в папке ресурсов модуля контроллеров моего много модульного проекта и предназначены для использования в HTML-фронте.

Также CSS-код представляет собой набор стилей для веб-интерфейса библиотеки. Он включает в себя:

* Основные стили: Определение цвета и стиля заголовков, таблиц и кнопок.
* Эффекты при наведении: Увеличение элементов и добавление теней при наведении курсора.
* Форматирование форм: Центрирование и расположение элементов формы.
* Фиксированные элементы: Закрепление шапки и подвала на странице.
* Анимация кнопок: Изменение цвета и вращение кнопок при наведении.
* Система рейтинга: Стили для интерактивных звезд рейтинга.
* Отзывы: Стили для блоков отзывов, включая медиа-запросы для адаптивности.
* Баннер: Стили для фонового изображения и текста на баннере.

Эти стили предназначены для создания привлекательного и функционального пользовательского интерфейса приложения библиотеки. Код организован и структурирован для обеспечения легкости восприятия и поддержки.

В модуле контроллеров, в папке ресурсов, хранится множество шаблонов Thymeleaf для различных страниц приложения.

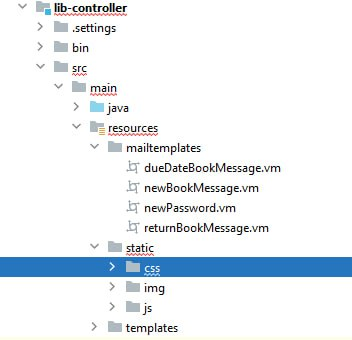
Например, один из шаблонов предназначен для страницы добавления книги в библиотеку. Он содержит форму, где пользователь может ввести ISBN книги и выбрать отдел, к которому она будет принадлежать. Форма отправляется методом POST.

Другой шаблон отображает список пользователей библиотеки с их именами, фотографиями, адресами электронной почты и отделами. Также есть возможность перейти на страницу с более подробной информацией о каждом пользователе.

Третий шаблон используется для страницы заказов, где отображаются детали заказа, такие как дата заказа, срок возврата, название книги, автор, имя пользователя, а также статусы продления и завершения заказа.

Эти шаблоны используют Thymeleaf для динамического отображения данных, полученных из модели Spring MVC. Стилизация осуществляется с помощью CSS, а для компоновки используется Bootstrap. Все шаблоны включают общие элементы, такие как заголовок и подвал, которые вставляются с помощью директив Thymeleaf.

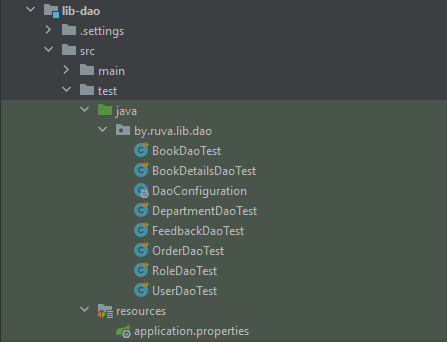
Например, есть шаблон для страницы ошибки, который информирует пользователя о недостаточных правах доступа. Другой шаблон представляет собой административную страницу с ссылками на все книги, пользователей, отделы и заказы. Также имеется шаблон прощания, страница сохранения изменений, страница восстановления пароля и домашняя страница с функциями входа и выхода, а также ссылками на профиль пользователя и обратную связь.

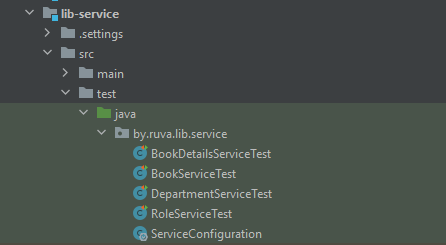


Таким образом, использование всех этих инструментов поможет создать качественное веб-приложение с хорошим дизайном, удобным интерфейсом и динамическим контентом.

3.2 JUnit – тестрование.

JUnit тесты — это фреймворк для модульного тестирования программного обеспечения, который позволяет быстро и просто проверить отдельные единицы кода, такие как методы и классы, на их корректность. Модульное тестирование является важной частью разработки программного обеспечения, поскольку оно помогает обнаружить ошибки на ранних этапах разработки, что экономит время и ресурсы. В моем многомодульном приложении на Java со Spring, я использовал JUnit тесты для обеспечения надежности и стабильности двух ключевых компонентов: модуля DAO и модуля сервисов. Модуль DAO содержит все crud методы для взаимодействия с базой данных, а модуль сервисов определяет всю бизнес-логику приложения. Покрытие этих модулей тестами JUnit позволяет мне быть уверенным в том, что каждый аспект моего приложения работает так, как задумано, и что любые изменения в коде не приведут к непредвиденным проблемам.



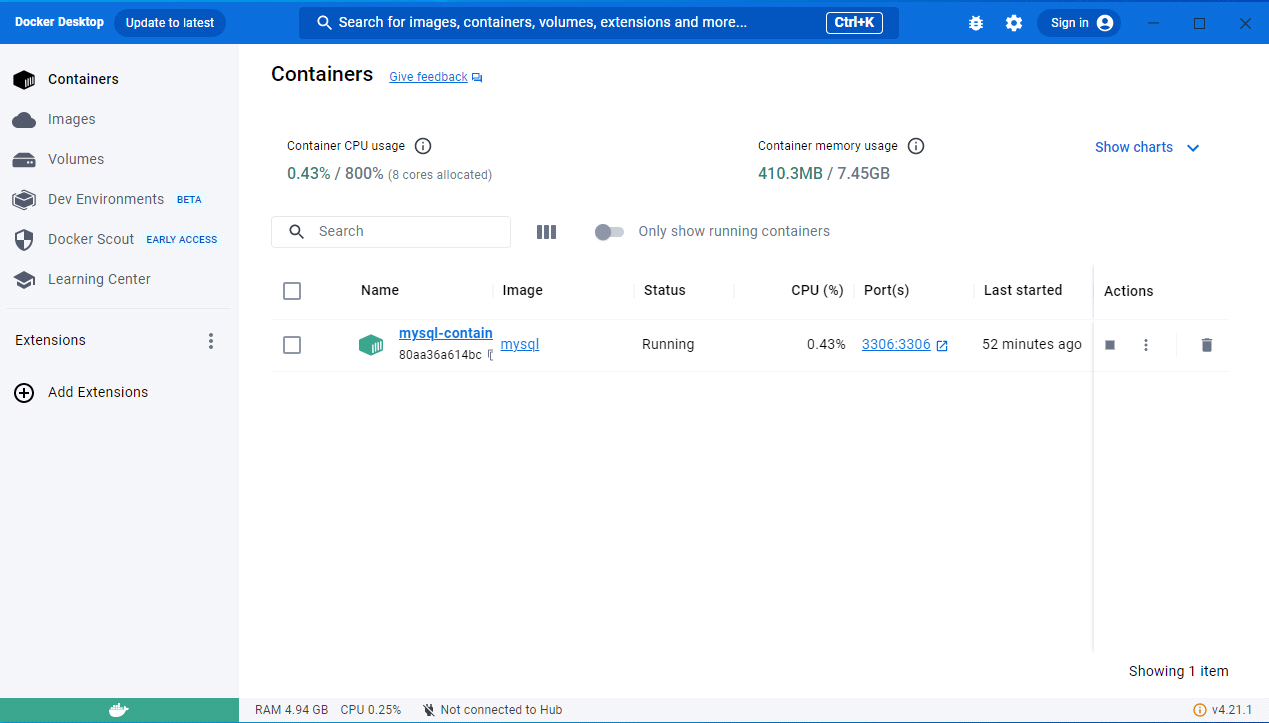


Использование JUnit тестов в моем проекте гарантирует, что бизнес-логика и взаимодействие с базой данных выполняются корректно. Это также облегчает процесс интеграции и непрерывной доставки, так как автоматизированные тесты могут быть запущены в любой момент для проверки текущего состояния кода. Таким образом, JUnit тесты играют ключевую роль в поддержании высокого качества и надежности моего приложения.

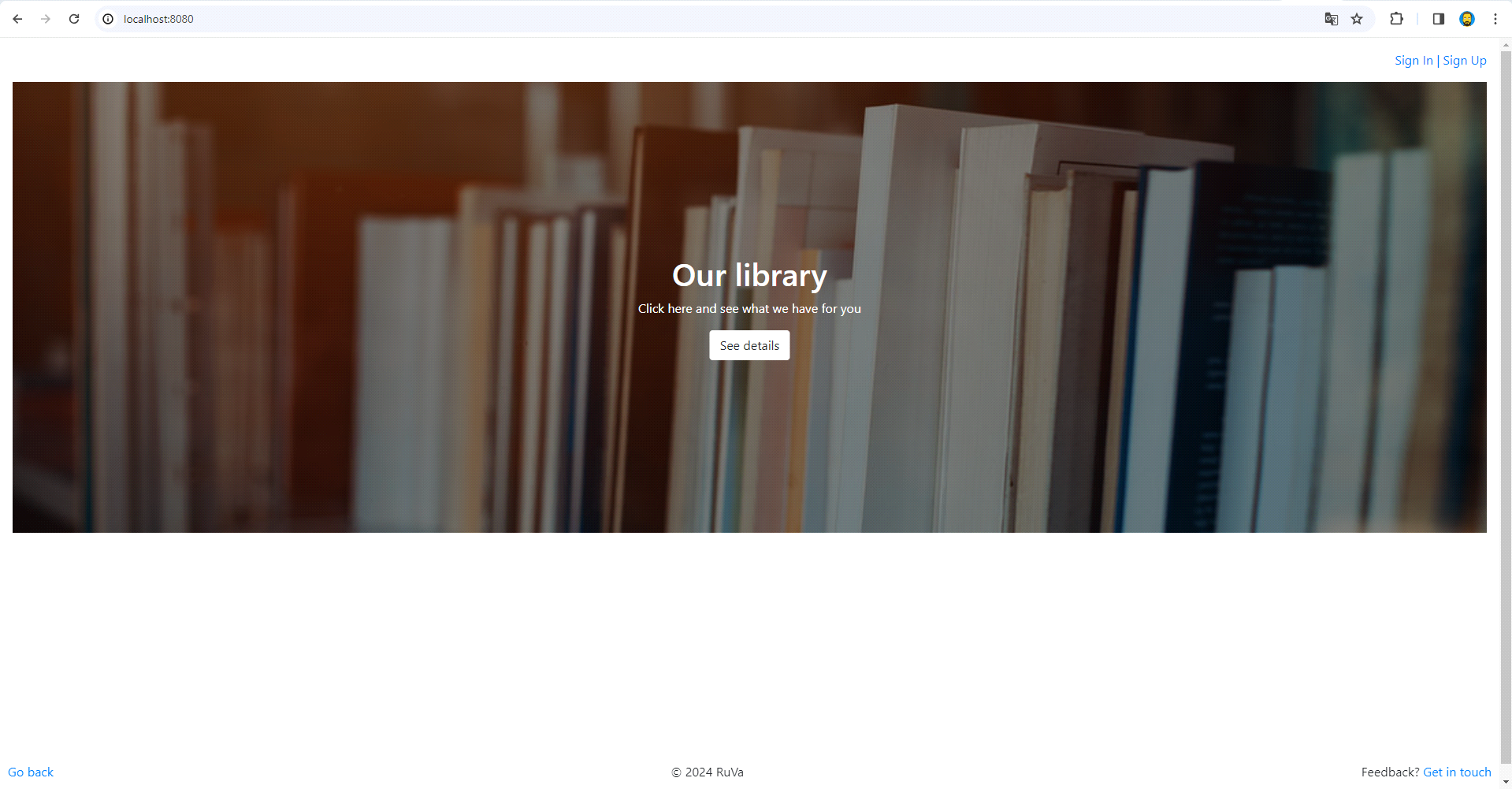
3.3 UI – тестирование.

Тестирование интерфейса — это запуск приложения в браузере и проведение запланированных проверок. В рамках этого тестирования мы удостоверяемся, что приложение взаимодействует с пользователем так, как ожидается.

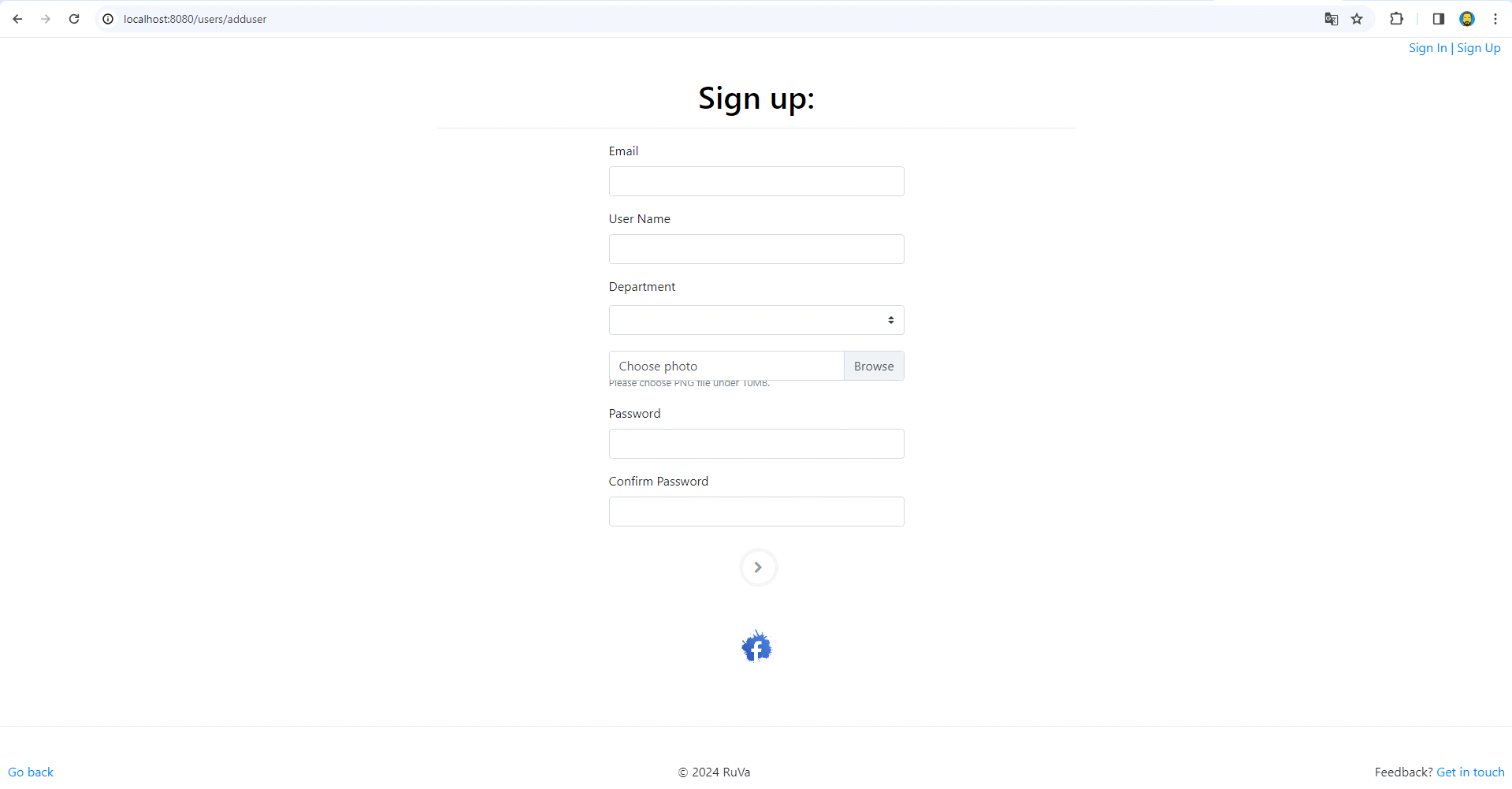
Для выполнения нашей программы необходимо запустить MySql при помощи Docker. Настройки подключения у нас были заданы в application.properties.



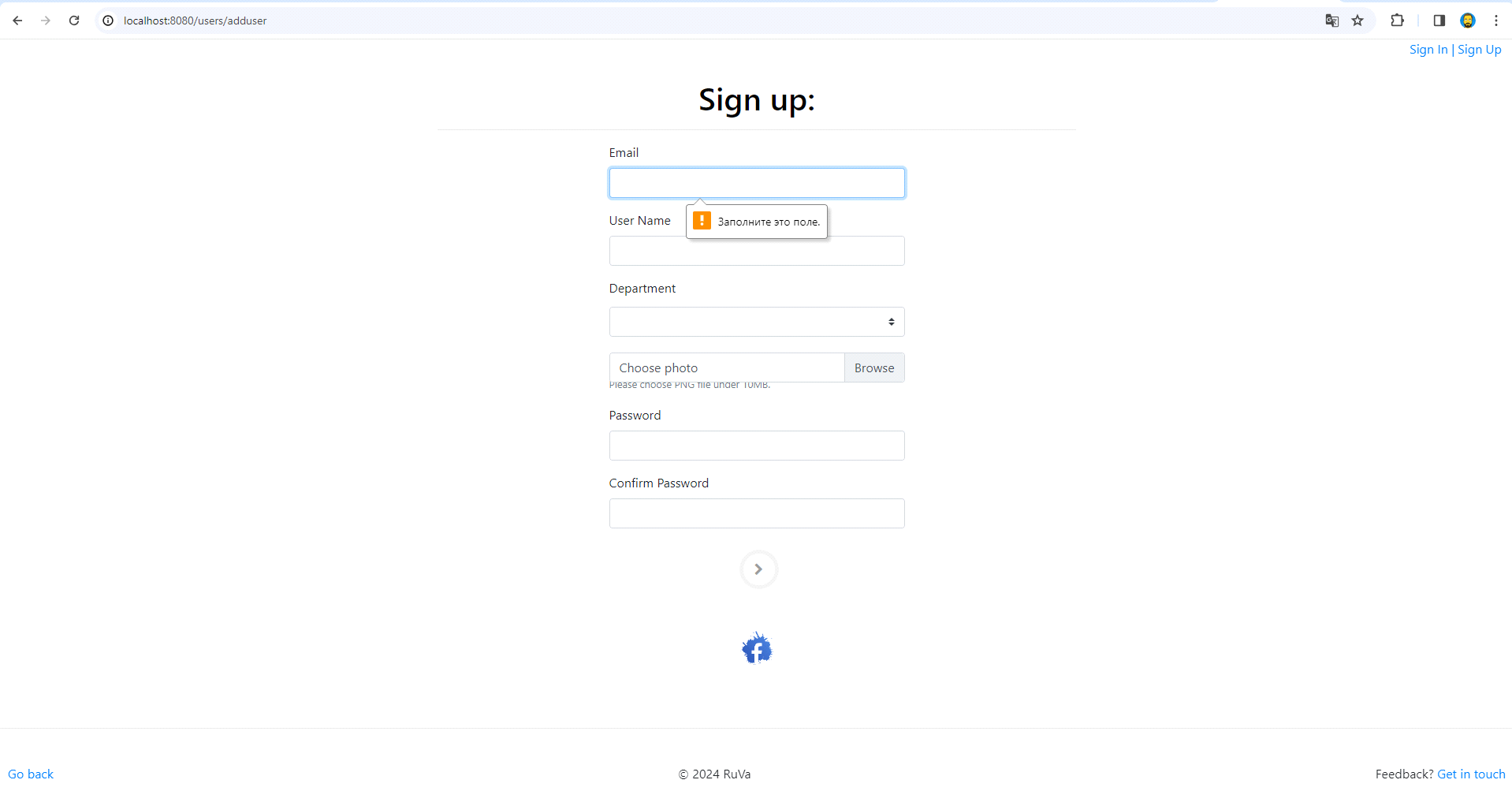
Запустив приложение из пакета lib-main, класса LibApp необходимо перейти по адресу указанному в application: http://localhost:8080/. Мы попадаем на главную страницу нашего сайта.



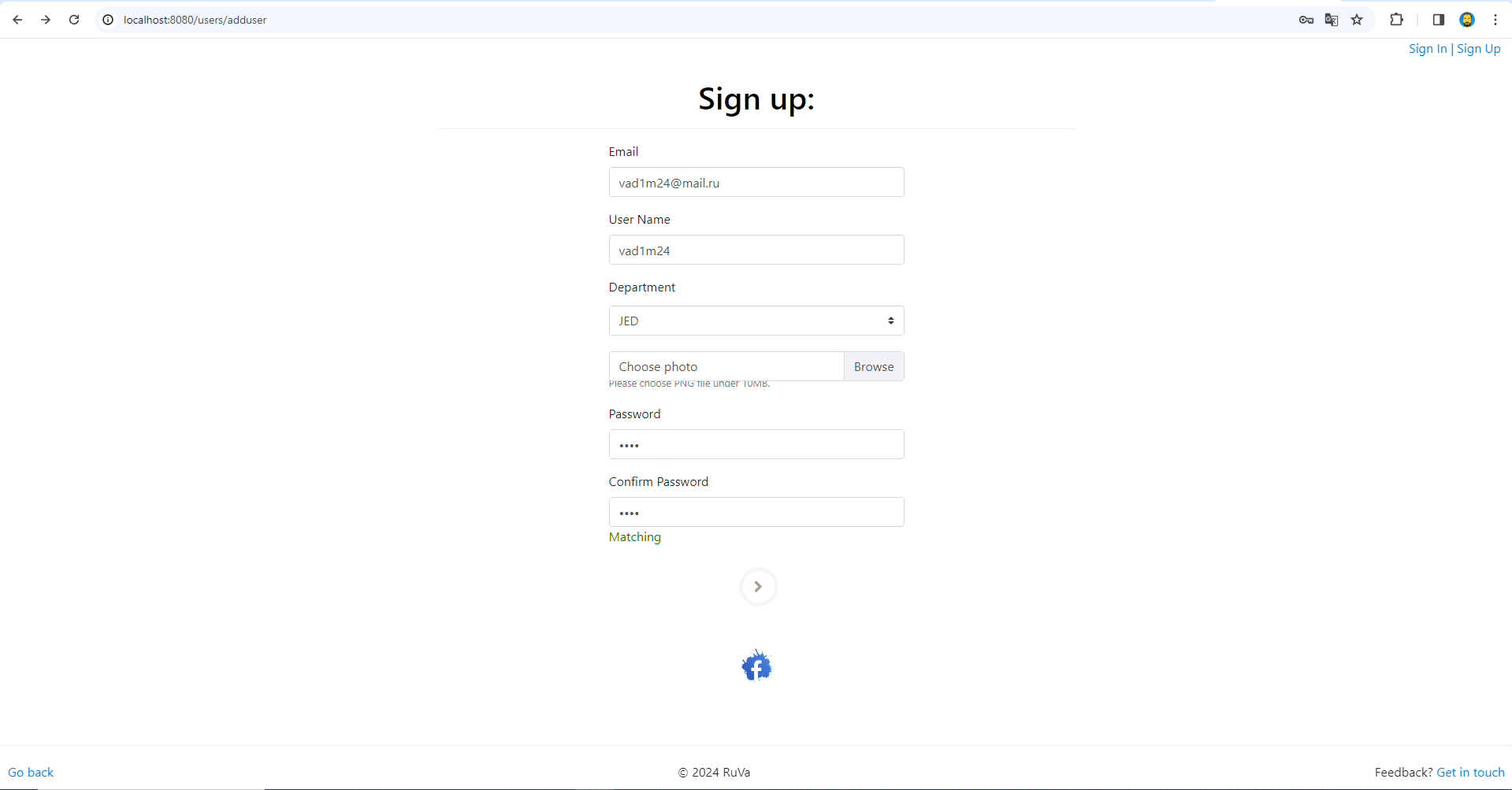
Для регистрации нового пользователя пройдем по ссылке SignUp и попадаем на следующую страницу.



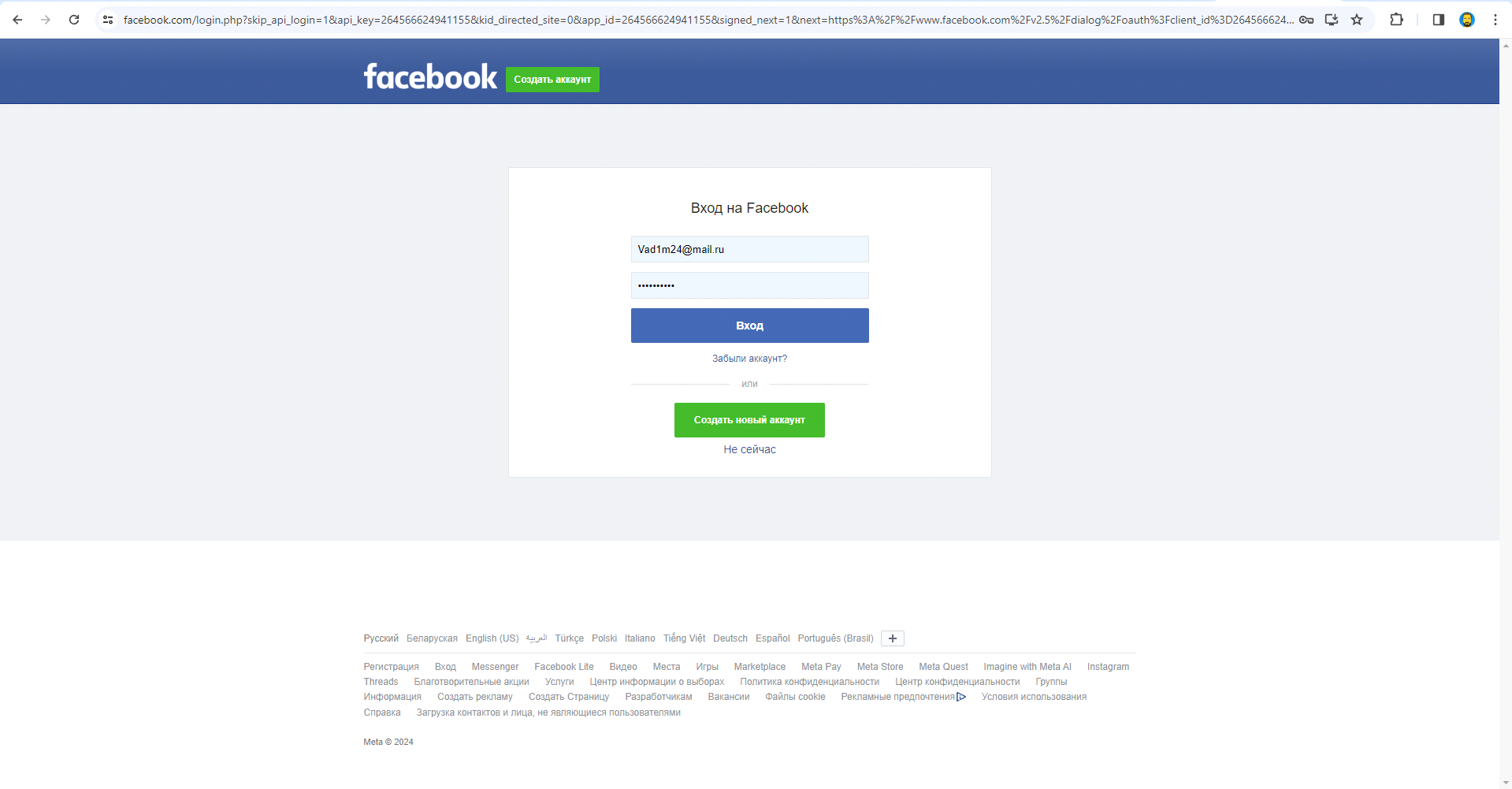
Настройки обязательного ввода информации предотвращают возможность оставления пустых полей.



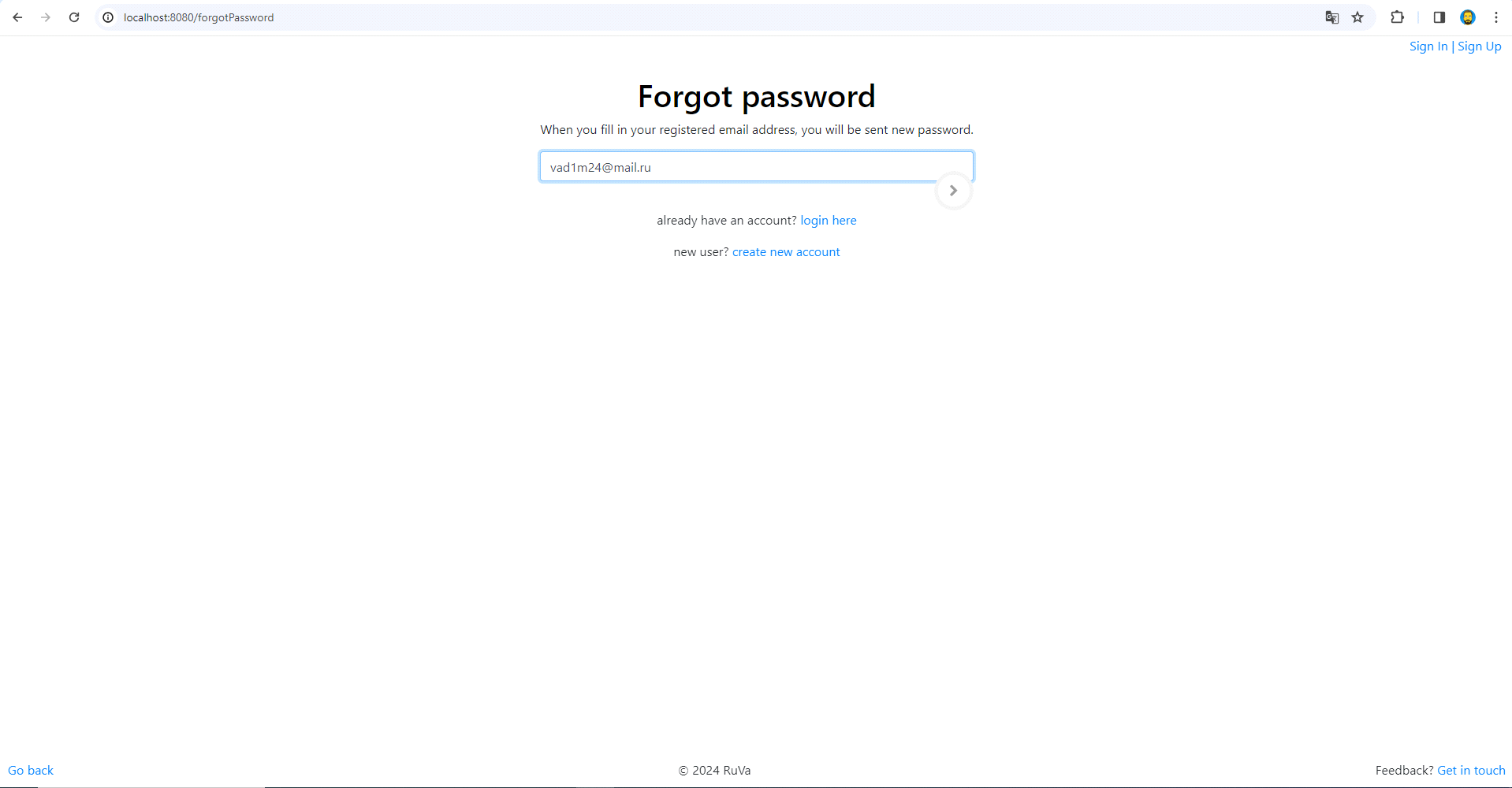
При заполнении полей также идет проверка соответствия ввода пароля и его подтверждение.



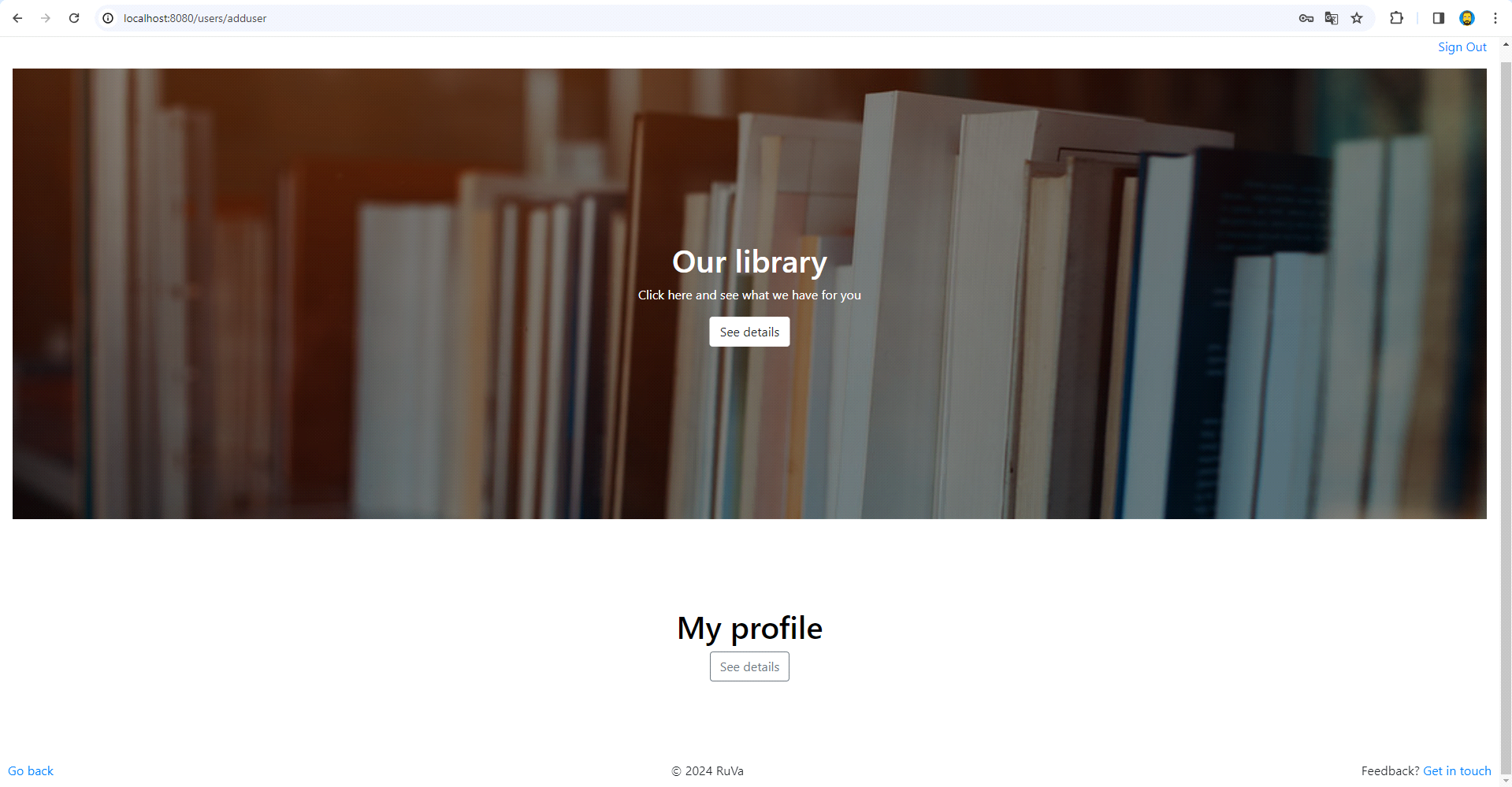
В процессе разработки мною была добавлена авторизация через Facebook, что позволяет пользователям входить на сайт, используя свои учетные записи в данной социальной сети. Основное преимущество этого функционала — это конечно же удобство пользователей для их регистрации. Перейдя по данной ссылке, мы попадаем на страницу авторизации:



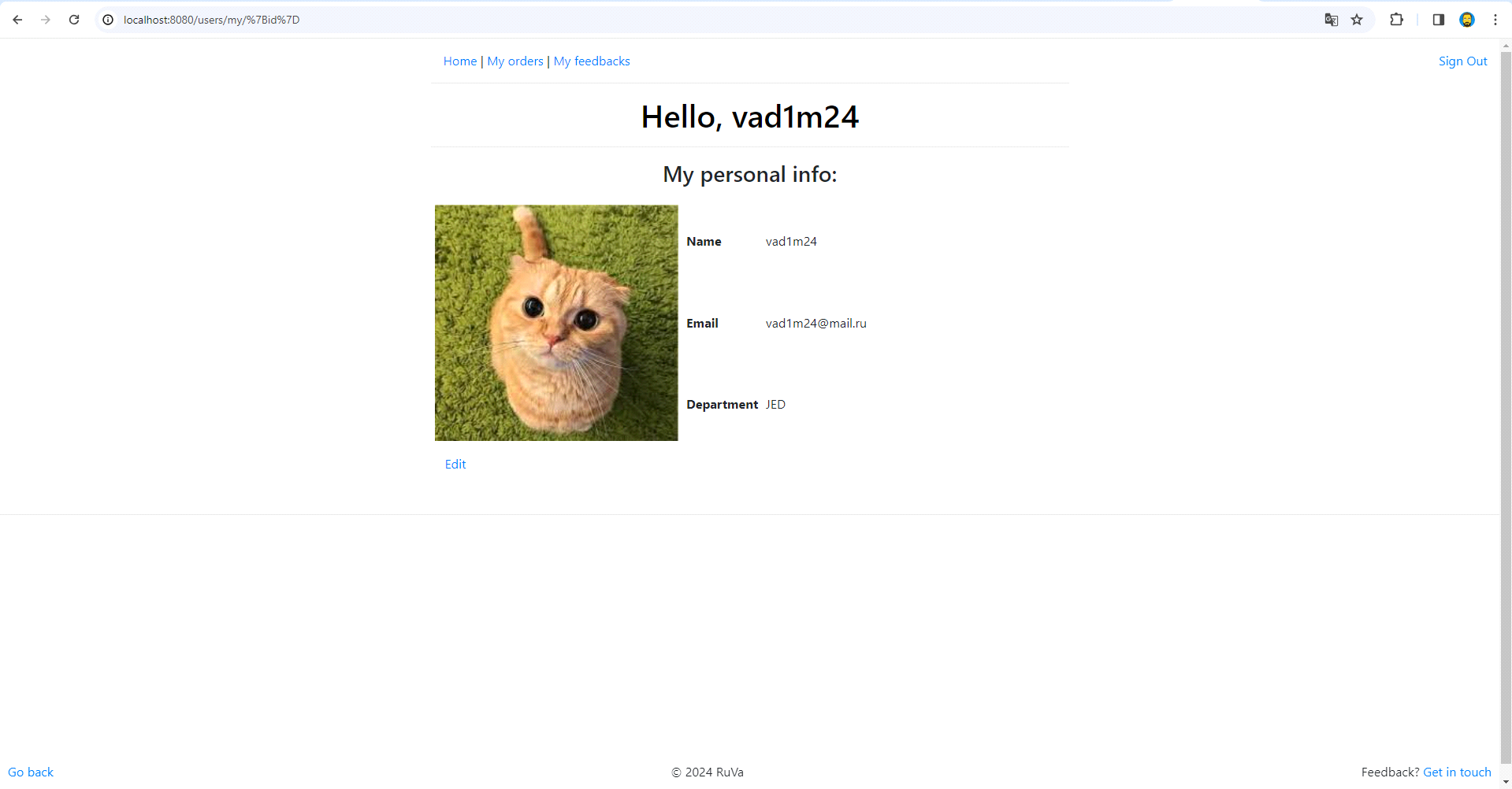
Если же у пользователя уже есть аккаунт, но по нелепой случайности он его забыл, всегда есть возможность проверить информацию и отправить пароль на электронный адрес, указанный при регистрации.



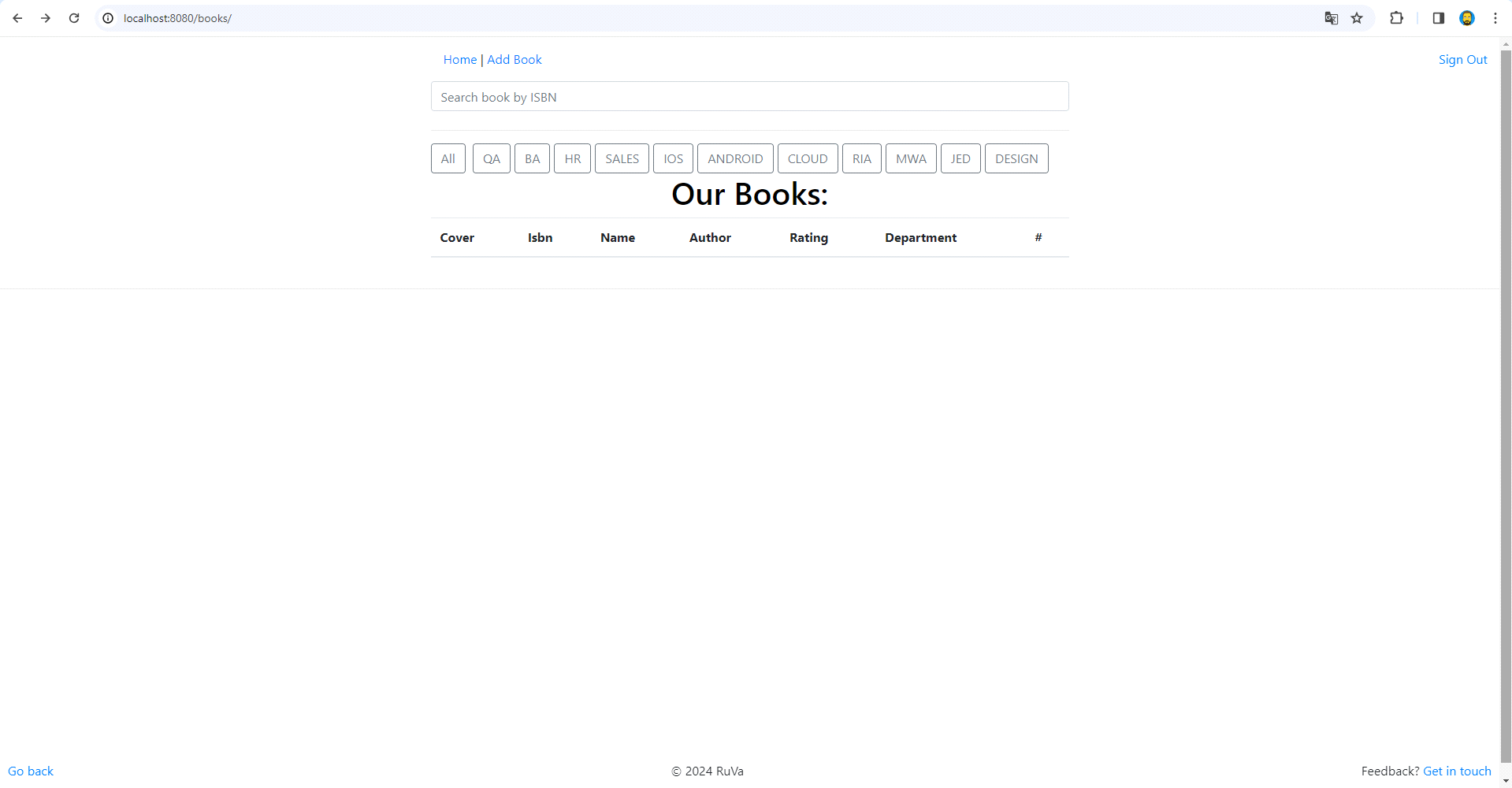
Далее, после входа в профиль, мы можем откорректировать личные данные пользователя или же посмотреть саму библиотеку.



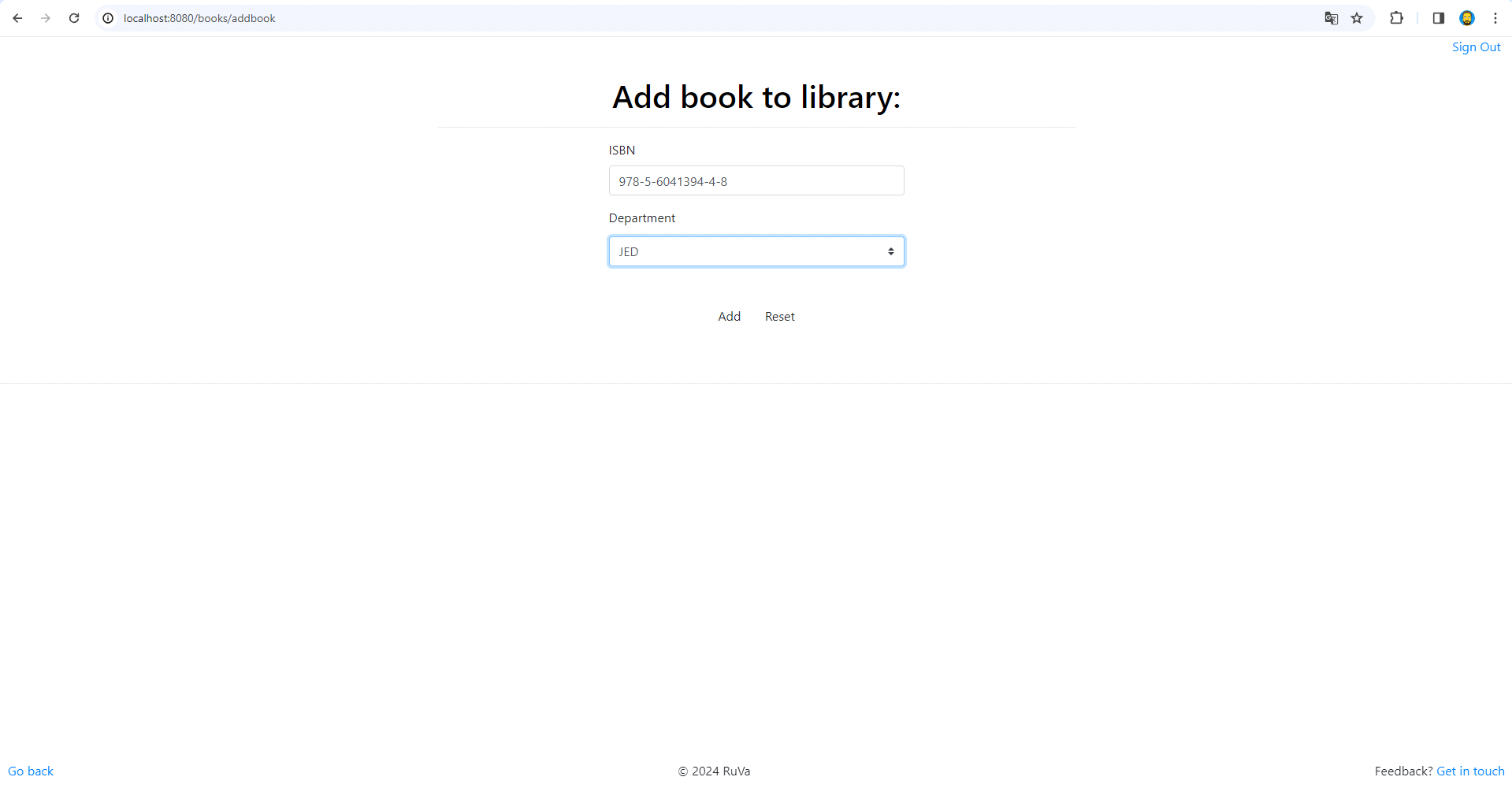
Для начала исследуем наш аккаунт.



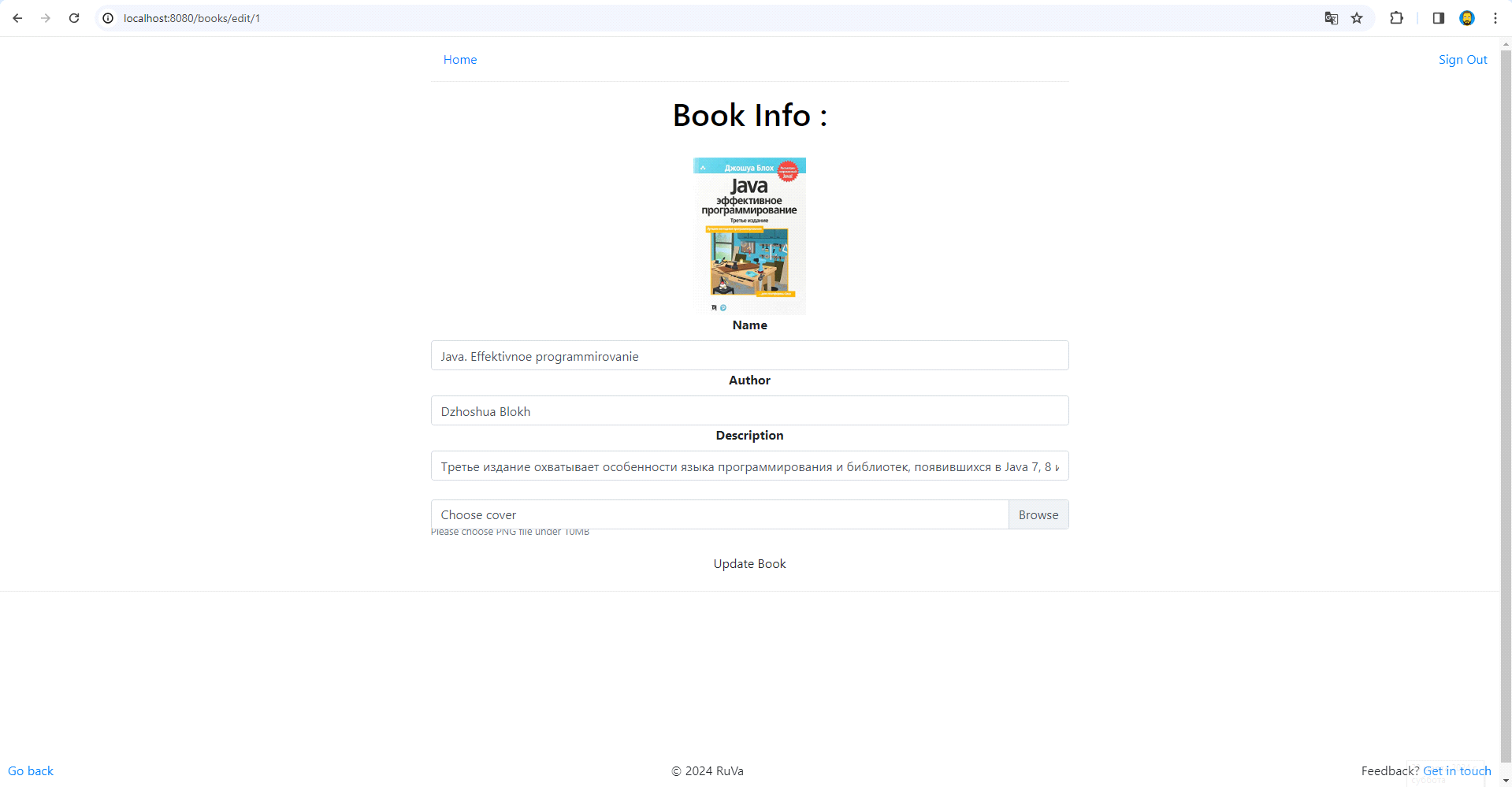
А теперь изучим ассортимент нашей библиотеки.



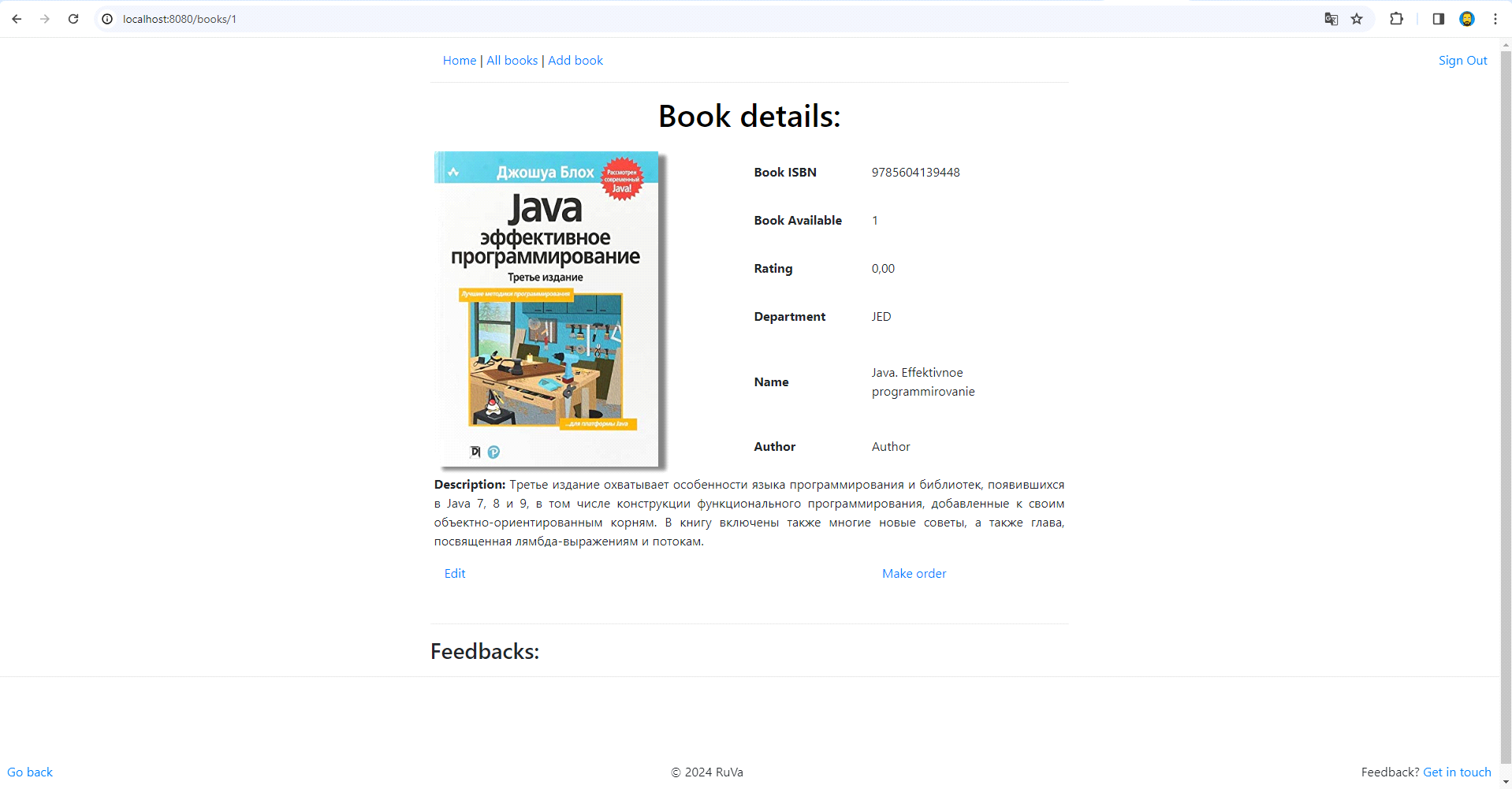
Пока в нашей базе еще нет никаких товаров, но далее мы и разберем как это изменить. В библиотеке, на правах администратора, мы можем добавить книгу, вписав ее ISBN в соответствующее поле, а также определив отдел для кого конкретно данный товар будет необходим. Для этого мы и переходимпо ссылке Add Book.



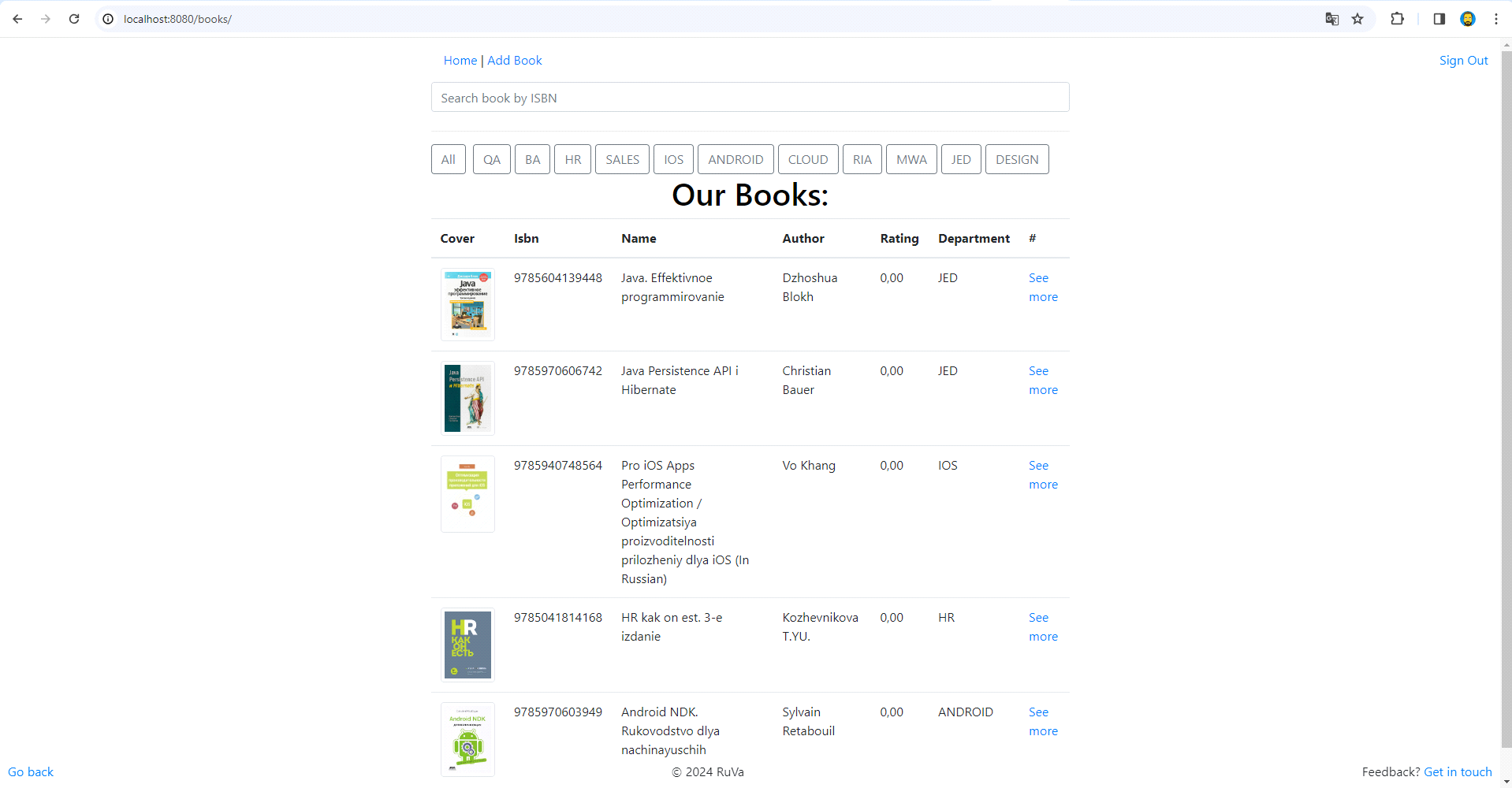
Теперь наша библиотека пополнилась первым экземпляром полезной продукции. Для удобства всех пользователей можно добавить описание книги.



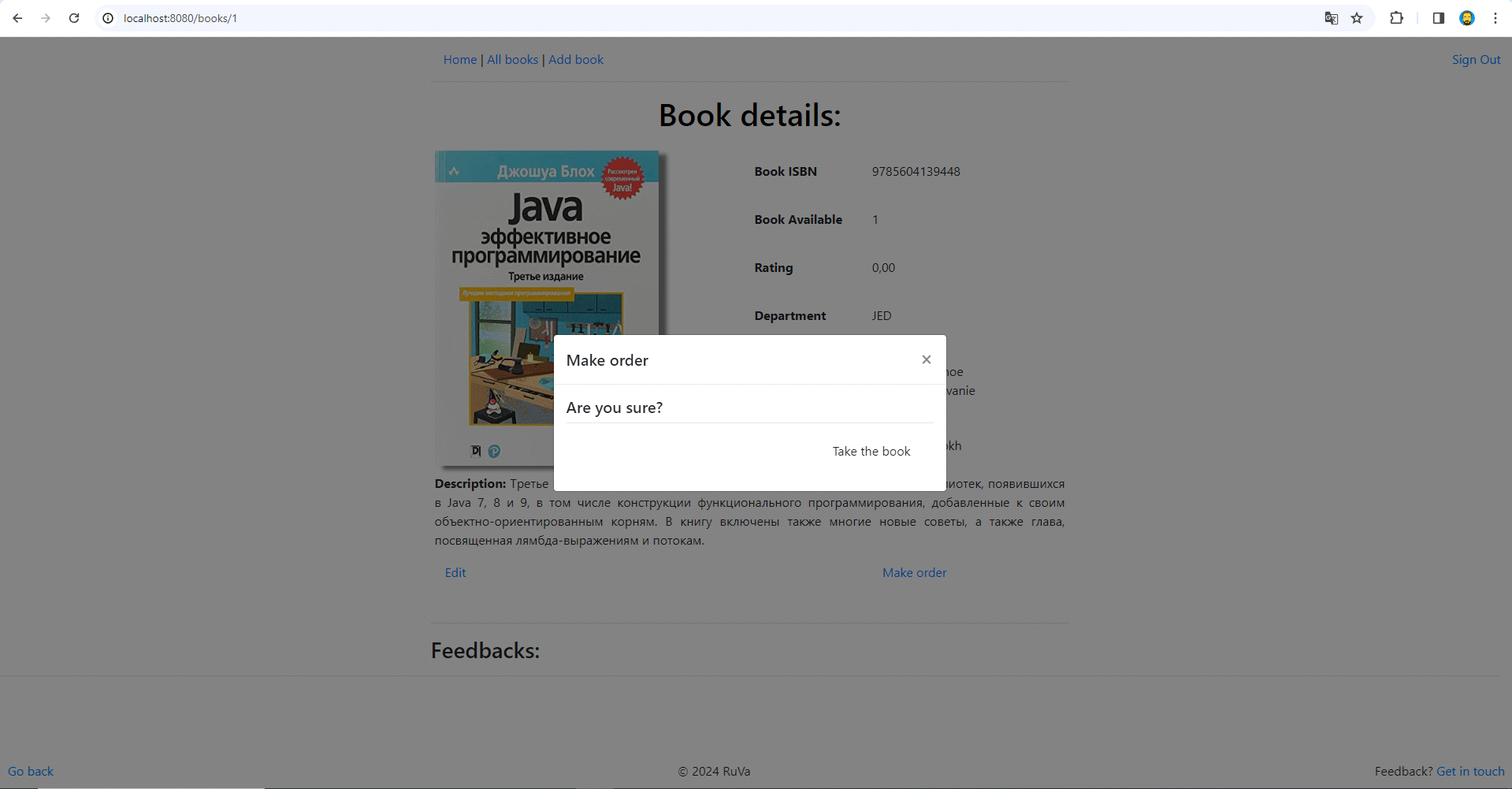
Вот как выглядит полное описание книги при ее просмотре.



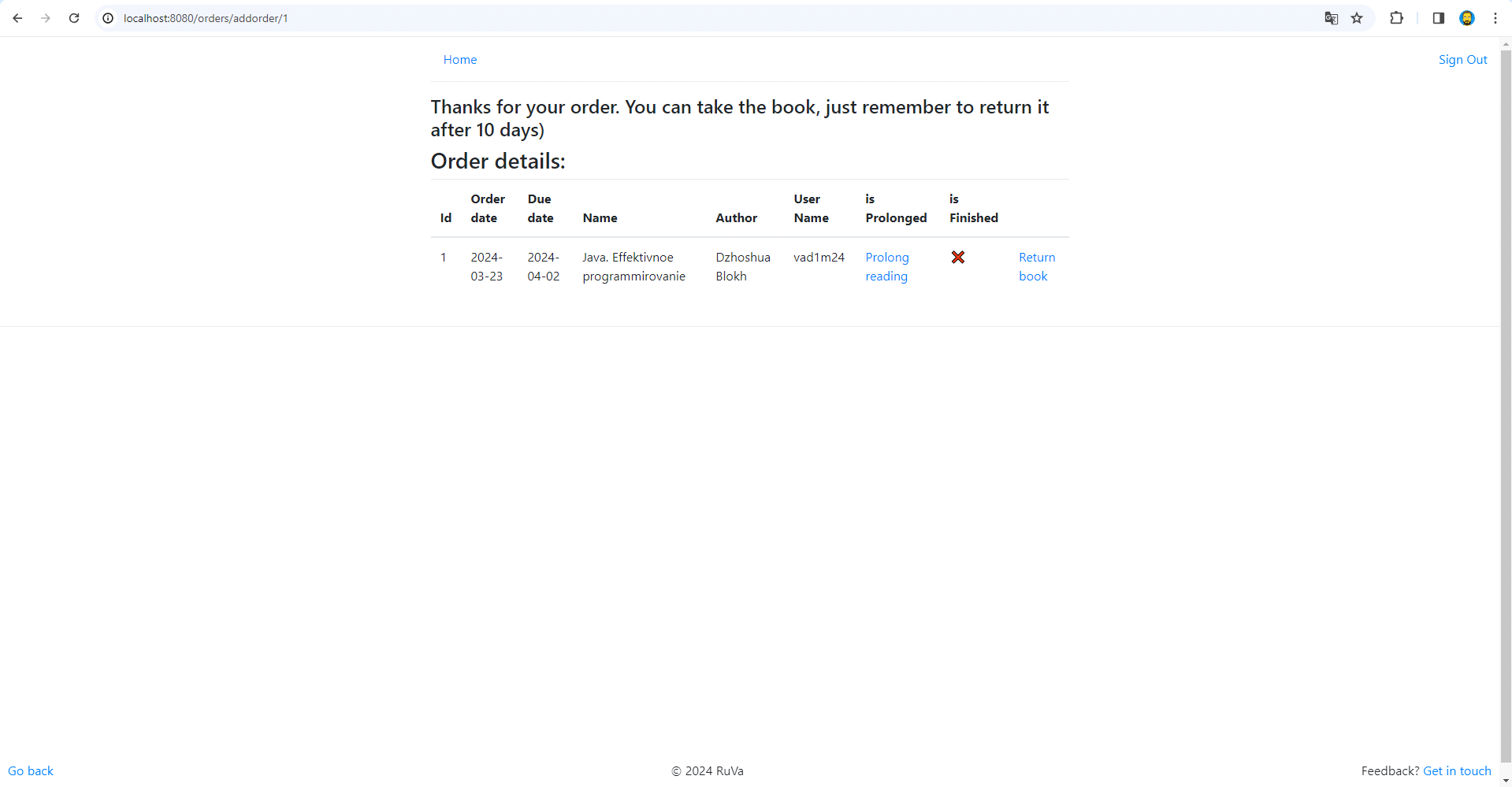
Разнообразим ассортимент, чтобы было из чего выбирать потенциальным пользователям.



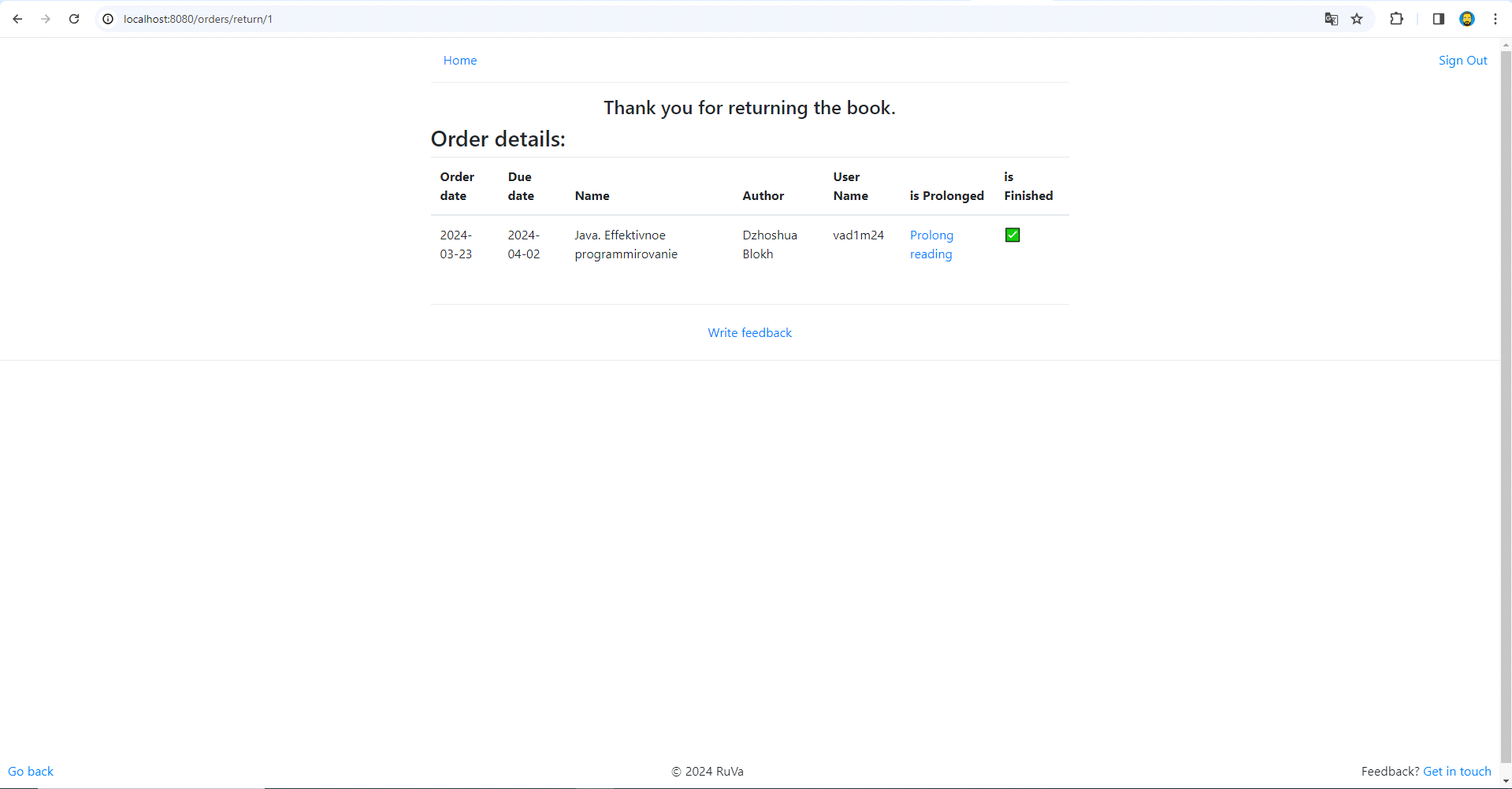
На примере первой книги, сделаем ее заказ.

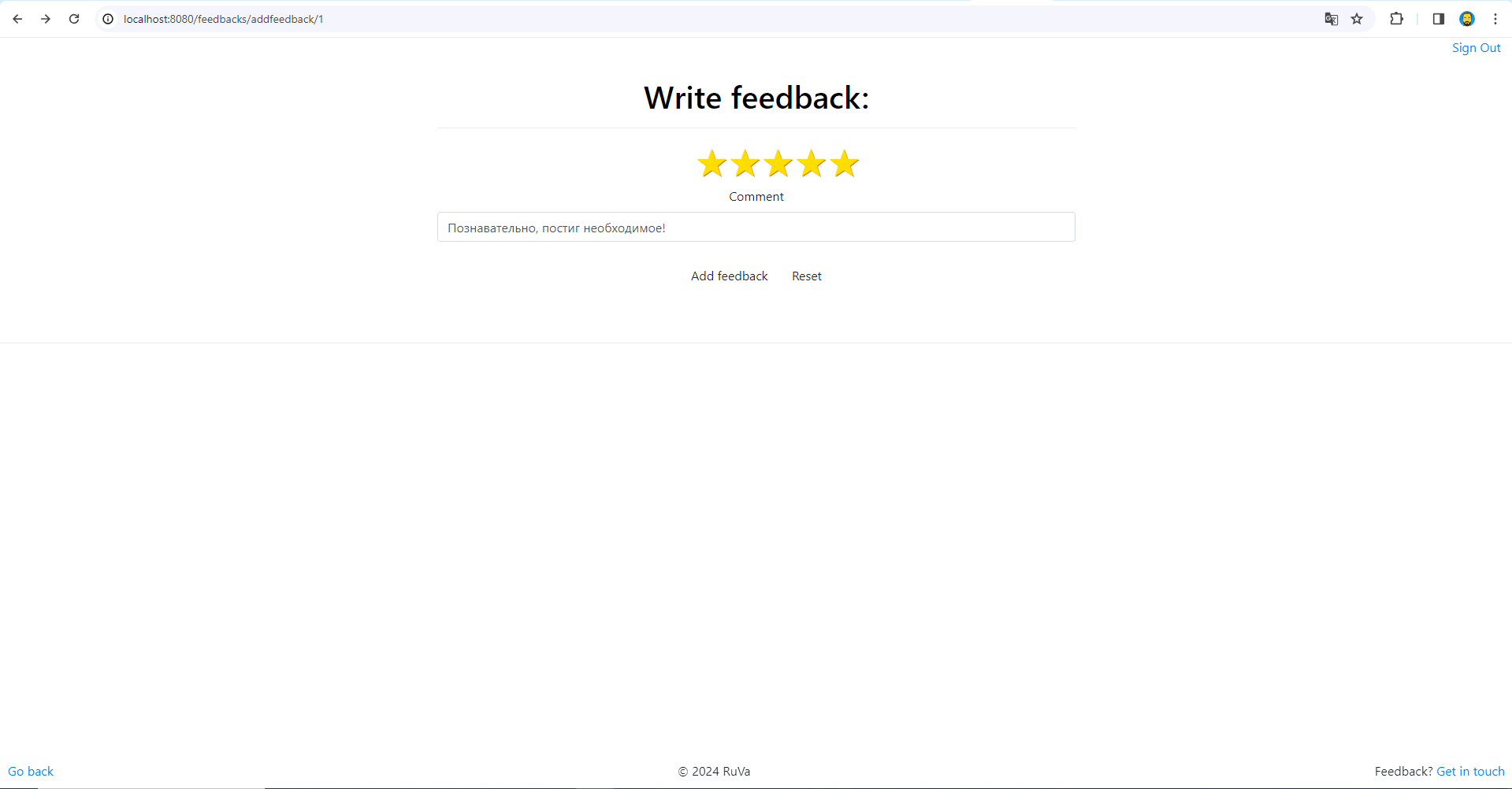


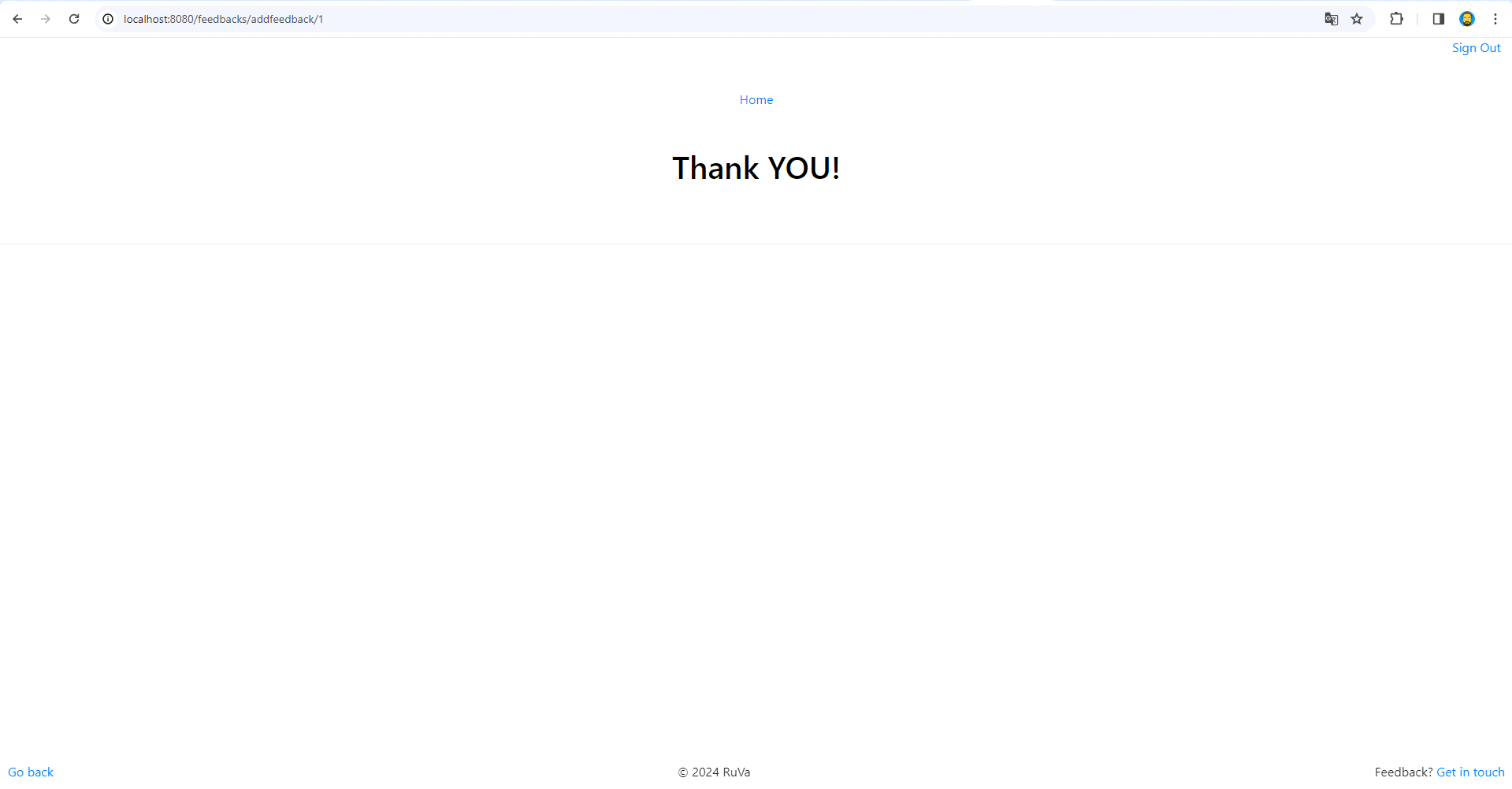
Подтвердив наше согласие, мы получаем информацию о том, что срок сдачи книги подходит через 10 дней, где конкретно указана дата сдачи, а также статус.



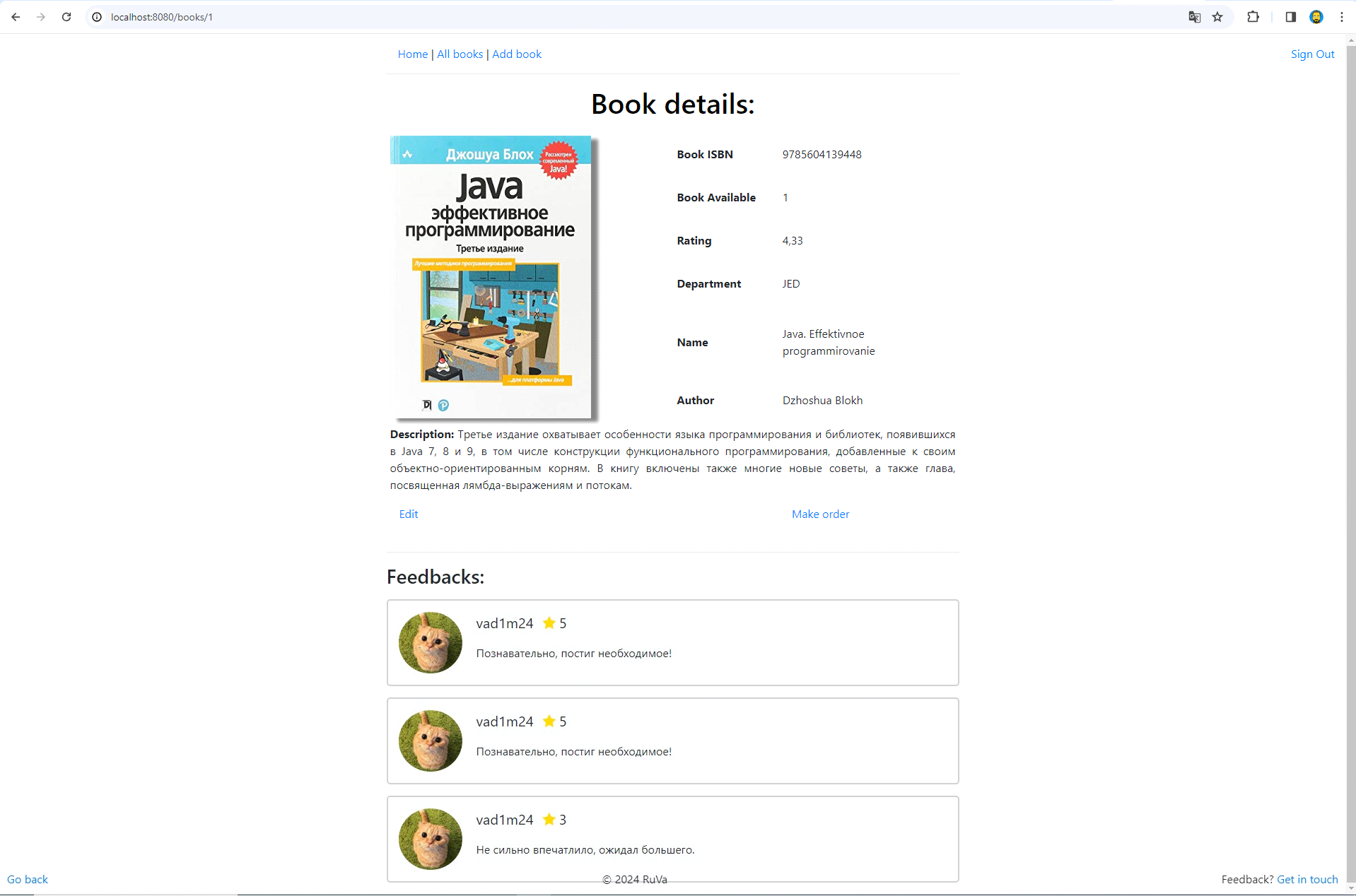
Вернув книгу, мы можем оставить полноценный отзыв для информативности и удобства пользователей.







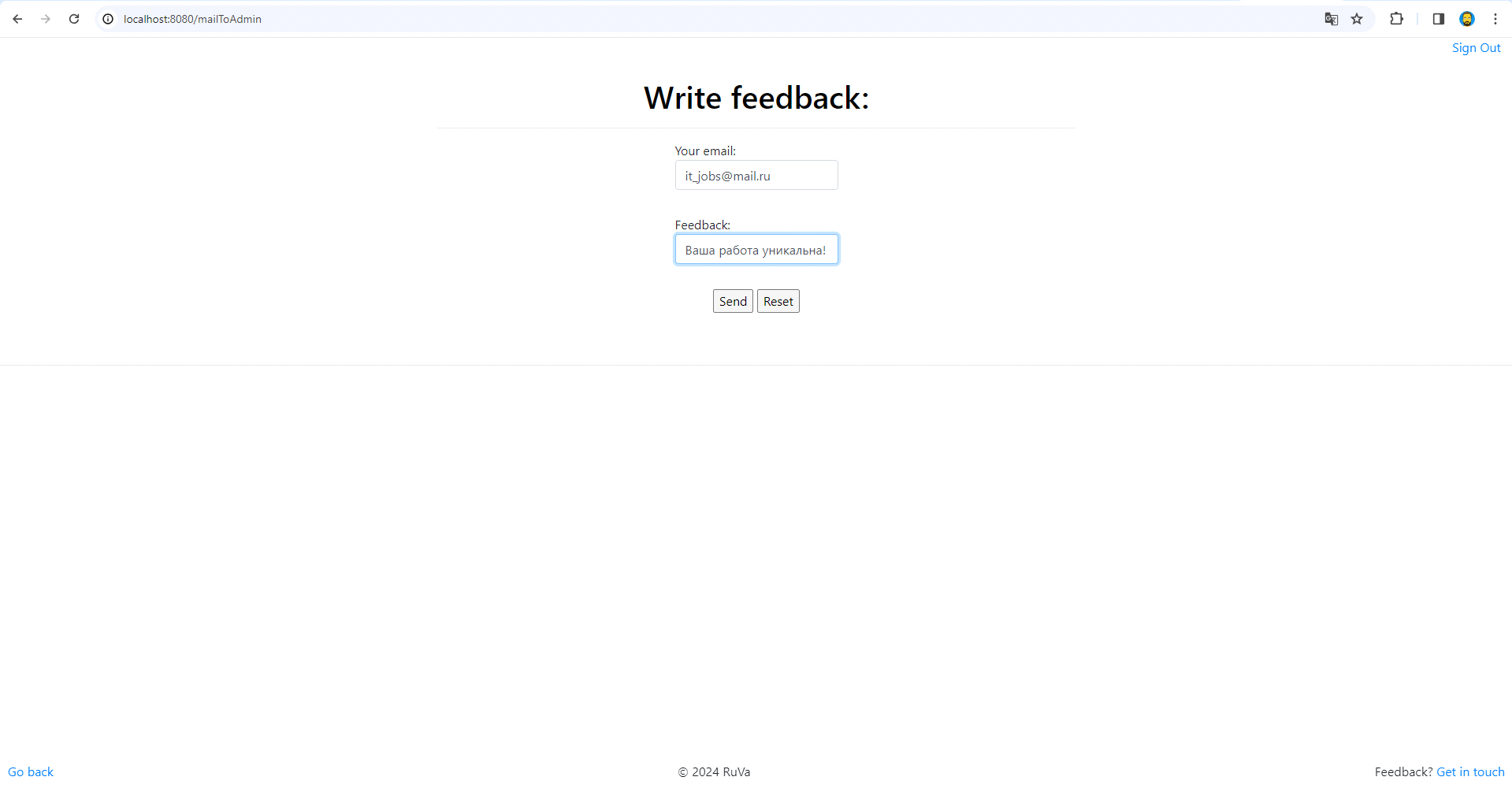
На основании всех оценок в отзывах формируется рейтинг книги.

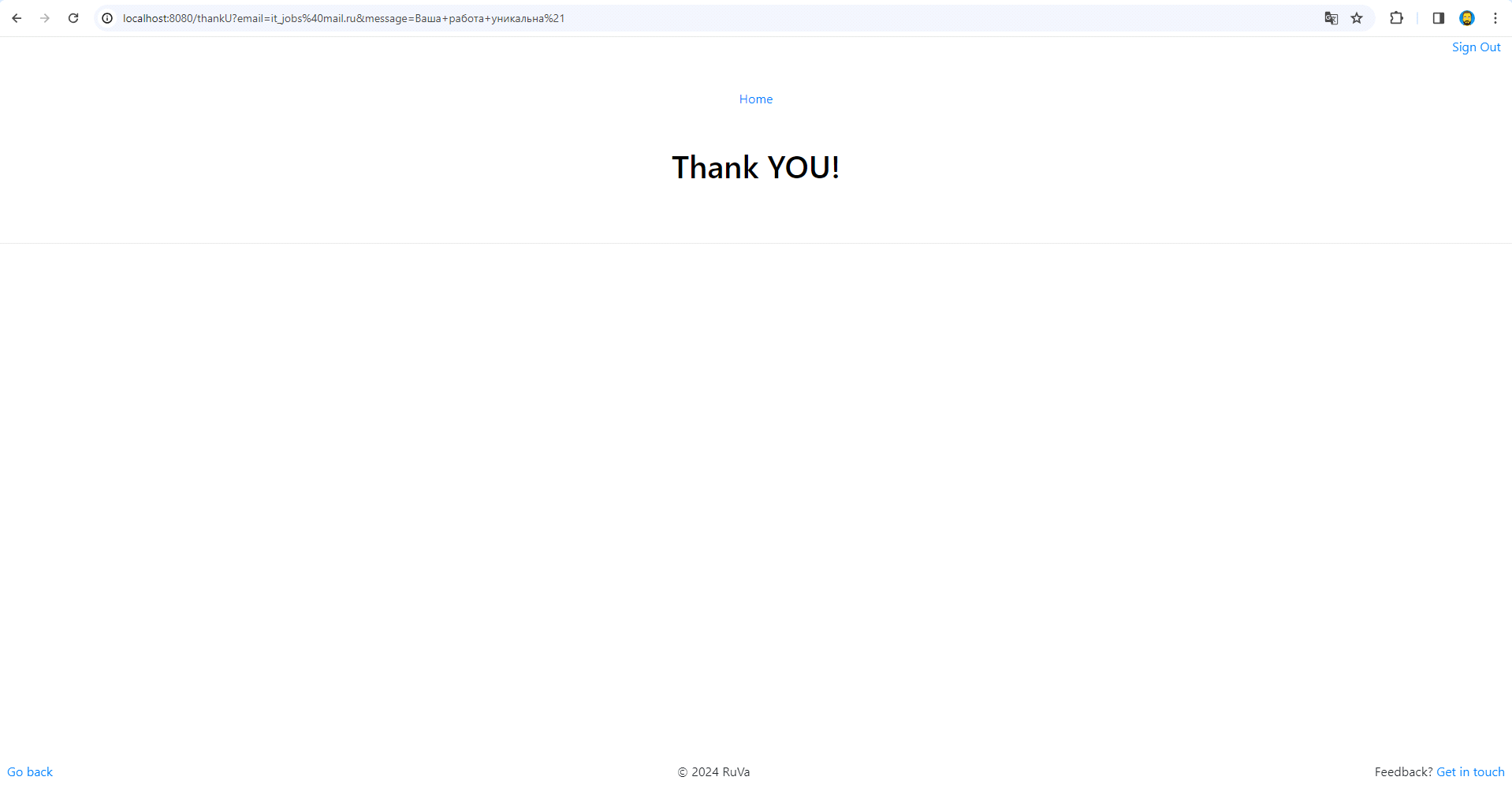


Для пользователей сайта также предусмотрена возможность дать обратную связь и оценку администратору, в случае возникновения претензий или пожеланий. Справа внизу экрана есть для этого ссылка.

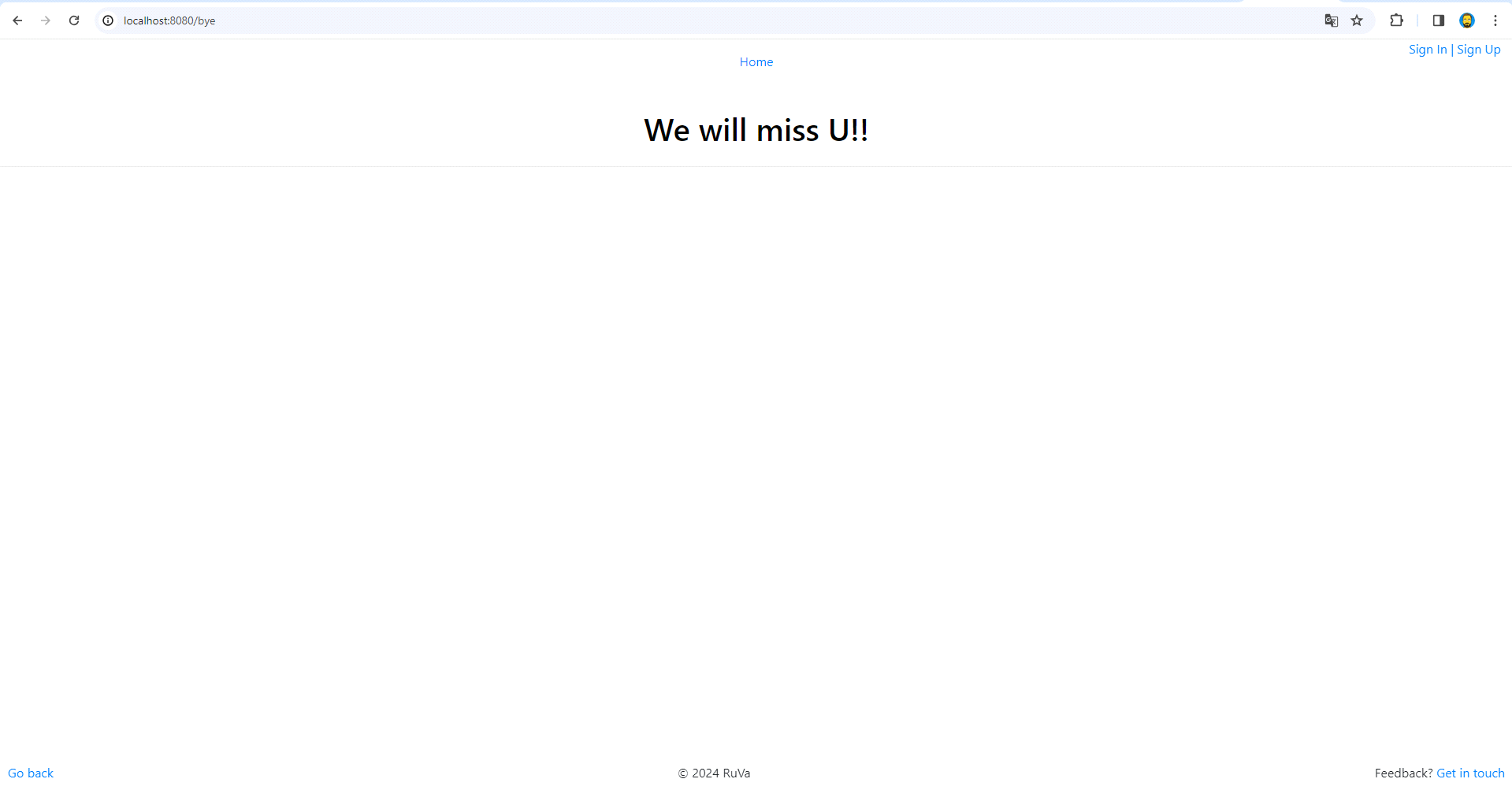


В появившемся окне необходимо указать вашу электронную почту для получения ответа на ваше письмо и заполнить собственно сам коментарий.





При выходе пользователя из своей учетной записи, на экран будет выведено следующее сообщение:



Заключение

В ходе выполнения дипломной работы было разработано многофункциональное веб-приложение для автоматизации работы библиотеки на базе Java Spring Boot. Проект реализован в соответствии с современными требованиями к программному обеспечению, что подтверждается использованием передовых технологий и подходов в разработке.

Основной упор в проекте был сделан на модульность и расширяемость системы. Благодаря чёткому разделению функционала по модулям, достигнута высокая степень взаимодействия и низкая связанность компонентов, что позволяет легко масштабировать приложение и внедрять новые функции без риска нарушения работы существующего функционала.

Модуль lib-api обеспечивает взаимодействие между слоями DAO и сервисов, а lib-controller отвечает за обработку пользовательских запросов. В модуле lib-dao реализован доступ к данным, а lib-entity содержит сущности системы. Точка входа приложения находится в модуле lib-main, а бизнес-логика сосредоточена в lib-service. Модуль lib-utils включает в себя вспомогательные инструменты, такие как почтовая рассылка и мапперы. Наконец, lib-web отвечает за взаимодействие с веб-ресурсами, в частности, за добавление информации о книгах по их ISBN.

Проект использует такие библиотеки и фреймворки, как Spring Boot, Hibernate, Liquibase, HTMLUnit, Thymeleaf и другие, что обеспечивает его высокую производительность, безопасность и удобство в использовании.

В ходе работы над проектом были решены следующие задачи:

* Разработка архитектуры приложения с учётом требований к модульности и расширяемости.
* Реализация пользовательского интерфейса с использованием HTML и CSS.
* Создание серверной части приложения с использованием Java Spring Boot.
* Интеграция с базой данных MySQL и настройка миграций через Liquibase.
* Обеспечение взаимодействия с внешними веб-ресурсами для получения данных о книгах.

Результаты работы над проектом показывают, что поставленные цели были достигнуты, а задачи выполнены полностью. Приложение успешно функционирует и готово к дальнейшему развитию и внедрению.

Перспективы развития проекта включают в себя:

* Добавление новых модулей для расширения функционала, например, модуля рекомендательной системы для пользователей библиотеки.
* Интеграция с системами аналитики для сбора статистики использования приложения.
* Разработка мобильного приложения для удобства пользователей.

Таким образом, дипломная работа является важным шагом в развитии информационных систем для библиотек и может служить основой для дальнейших исследований и разработок в данной области.

Список используемой литературы:

1. <https://ru.wikipedia.org/>

2. <https://www.wikiwand.com/>

3. <https://www.java.com/>

4. <https://skillbox.ru/>

5. <https://igrushkiland.ru/>

6. <https://ru.education-wiki.com/>

7. <https://habr.com/>

8. <https://javastudy.ru/>

9. <https://highload.today/>

10. <https://highload.today/>

11. <https://docs.liquibase.com/>

12. <http://www.liquibase.org/>

13. <https://www.thymeleaf.org/>