**Федеральное агентство морского и речного транспорта**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Волжский государственный университет водного транспорта»**

**Институт экономики, управления и права**

**Кафедра систем информационной безопасности, управления и телекоммуникаций**

**ОТЧЕТ**

**ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ (ЭКСПЛУАТАЦИОННОЙ)**

обучающегося 4 курса очной формы обучения группы ОИСТ(б)-4 по

направлению подготовки 09.03.02 «Информационные системы и технологии»

|  |
| --- |
| Пчелкина Дениса Андреевича |
| (ФИО обучающегося) |
| *Волжский государственный университет водного транспорта* |
| (наименование места прохождения практики) |

|  |
| --- |
| Руководители практики: |
| **от университета** |
| доцент к.т.н. Павлова Н.Г. |
| Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
| **От профильной организации** |
| доцент к.т.н. Павлова Н.Г. |
| Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
|  |
| Обучающийся: |
| Пчелкин Д.А.  Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

Нижний Новгород

2024

**совместный Рабочий график (план)  
проведения ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ (ЭКСПЛУАТАЦИОННОЙ)**

|  |  |
| --- | --- |
| в | *Волжский государственный университет водного транспорта* |
|  | (наименование организации) |

|  |
| --- |
| Пчелкина Дениса Андреевича |
| (ФИО обучающегося) |

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование работы | Место практики (структурное подразделение, рабочее место обучающегося) | Дата  (сроки) |
| 1. | 1 этап **Подготовительный**  Инструктаж по охране труда, технике безопасности и пожарной безопасности; ознакомление с правилами внутреннего распорядка организации | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024 |
| 2. | 2 этап **Основной** | Кафедра «Информатики, систем управления и телекоммуникаций» |  |
| 2.1. | Изучение:  - специфики деятельности организации, в которой проходит практика (далее – организация);  - соответствующих нормативных правовых и локальных актов, регламентирующих деятельность организации;  - нормативных правовых актов, регламентирующих деятельность предприятий (самостоятельная работа) | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024 |
| 2.2. | Ознакомление со структурой организации, ее целями и задачами | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024  -  02.09.2024 |
| Ознакомление с типовой структурой предприятия (самостоятельная работа) |
| Анализ бизнес-процессов организации и информационного взаимодействия структурных подразделений внутри организации и с контрагентами | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024 |
| Анализ бизнес-процессов и информационных потоков предприятия (самостоятельная работа) | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024 |
| Ознакомление с основными подходами к разработке планов и программ по совершенствованию деятельности и дальнейшему развитию организации | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024 |
| Выработка мероприятий по совершенствованию деятельности предприятия (самостоятельная работа) | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024 |
| Изучение и описание функций структурного подразделения, в котором проходит практика, его места и роли в реализации целей и задач организации | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024 |
| 2.3. | Анализ использования информационных систем организации | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024  -  02.09.2024 |
| Определение рисков, связанных с функционированием информационных систем и технологий организации, выработка направлений снижения негативного влияния рисков (при их наличии) в рамках поддержания безопасности деятельности организации, достижения оптимальности условий труда персонала, сохранения природной среды, обеспечения устойчивого развития общества |
| Информационные системы предприятий (самостоятельная работа) |
| 2.4. | Постановка основных целей и задач практики | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024  -  02.09.2024 |
| Изучение: Java и Spring Boot для разработки серверной части приложений» |
| 2.5. | Решение (участие в решении):  Разработка базы данных с frontend и backend на основе Spring Boot | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024  -  01.11.2024 |
| 2.6. | Выполнение индивидуального задания на практику | Кафедра «Информатики, систем управления и телекоммуникаций» | 02.09.2024  -  01.11.2024 |
| 3. | 3 этап **Заключительный**  Подготовка отчета по практике | Кафедра «Информатики, систем управления и телекоммуникаций» | 01.11.2024 |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Руководитель практики от университета |  | Н.Г. Павлова |
|  | *(подпись)* |  |
| Руководитель практики  от профильной организации |  | Н.Г. Павлова |
|  | *(подпись)* |  |
| Обучающийся |  | Д.А. Пчелкин |
|  | *(подпись)* |  |

**Дата составления**: \_\_\_\_\_\_\_\_\_\_\_\_\_

**индивидуальноЕ ЗАДАНИЕ**

**НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ (ЭКСПЛУАТАЦИОННУЮ)**

*Пчелкина Дениса Андреевича*

**Форма обучения:** *очная*

**Направление подготовки:** *09.03.02 «Информационные системы и технологии»*

|  |  |
| --- | --- |
| **Направляется для прохождения практики** в | *ВГУВТ* |
|  | (наименование организации) |

**Сроки прохождения практики:** с *02.09.2024* по *01.11.2024*

**Планируемые результаты практики:**

По результатам прохождения практики обучающийся должен обладать следующими компетенциями:

| Шифр | Наименование |
| --- | --- |
| ПК-2 | Понимание основных модулей и компонентов Spring. |
| ПК-3 | Умение конфигурировать и использовать Spring Data JPA для работы с реляционными базами данных например PostgreSQL, |
| УК-8 | Понимание работы с Maven или Gradle для сборки проектов, управления зависимостями и конфигурации. |

**Содержание задания на практику (перечень подлежащих рассмотрению вопросов):**

1. Инструктаж по охране труда, технике безопасности и пожарной безопасности; ознакомление с правилами внутреннего распорядка организации.

2. Характеристика места прохождения практики:

2.1. Изучение специфики деятельности организации, в т.ч. ознакомление с основными нормативными правовыми и локальными актами, регламентирующими ее деятельность.

2.2. Ознакомление со структурой организации, ее целями и задачами.

2.3. Изучение и описание функций структурного подразделения, в котором проходит практика, его места и роли в реализации целей и задач организации.

2.4. Изучение нормативного, организационного, информационного обеспечения деятельности предприятия (самостоятельная работа).

3. Постановка основных целей и задач практики.

4. Изучение вопросов, направленных на достижение целей практики.

5. Решение (участие в решении) задач, направленных на достижение целей практики.

6. Подготовка отчета по практике.

|  |  |  |
| --- | --- | --- |
| Руководитель практики от университета |  | Н.Г. Павлова |
|  | *(подпись)* |  |

**Согласовано:**

|  |  |  |
| --- | --- | --- |
| Руководитель практики от профильной организации |  | Н.Г. Павлова |
| МП | *(подпись)* |  |

**Ознакомлен:**

|  |  |  |
| --- | --- | --- |
| Обучающийся |  | Д.А. Пчелкина |
|  | *(подпись)* |  |

**Дата выдачи**: 02.09.2024

**ДНЕВНИК ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ (ЭКСПЛУАТАЦИОННОЙ)**

Обучающегося: *Пчелкина Дениса Андреевича,* 4 курс, очная форма обучения, группа ОИСТ(б)-4

Направление подготовки: *09.03.02 «Информационные системы и технологии»*

Место прохождения практики: *Волжский государственный университет водного транспорта, г. Н.Новгород*

Сроки практики: с *02.09.2024* по *01.11.2024*

Руководитель практики от профильной организации: Руководитель сервисной службы Павлова Н.Г.

Руководитель практики от университета: Павлова Н.Г.

| №п/п | Дата  (сроки) | Место практики (структурное подразделение, рабочее место обучающегося) | Краткое содержание работы, выполненной обучающимся | Отметка о выполнении руководителя практики от профильной организации, подпись | Контроль руководителя практики от университета, подпись |
| --- | --- | --- | --- | --- | --- |
| 1. | 02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | 1 этап **Подготовительный**  Инструктаж по охране труда, технике безопасности и пожарной безопасности; ознакомление с правилами внутреннего распорядка организации |  |  |
| 2. |  | Кафедра «Информатики, систем управления и телекоммуникаций» | 2. этап Основной |  |  |
| 2.1. | 02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Изучение:  - специфики деятельности организации, в которой проходит практика (далее – организация);  - соответствующих нормативных правовых и локальных актов, регламентирующих деятельность организации;  - нормативных правовых актов, регламентирующих деятельность предприятий (самостоятельная работа) |  |  |
| 2.2. | 02.09.2024  -  02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Ознакомление со структурой организации, ее целями и задачами |  |  |
|  |  | Кафедра «Информатики, систем управления и телекоммуникаций» служба (техотдел) | Ознакомление с типовой структурой предприятия (самостоятельная работа) |  |  |
|  | 02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Анализ бизнес-процессов организации и информационного взаимодействия структурных подразделений внутри организации и с контрагентами |  |  |
|  | 02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Анализ бизнес-процессов и информационных потоков предприятия (самостоятельная работа) |  |  |
|  | 02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Ознакомление с основными подходами к разработке планов и программ по совершенствованию деятельности и дальнейшему развитию организации |  |  |
|  | 02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций») | Выработка мероприятий по совершенствованию деятельности предприятия (самостоятельная работа) |  |  |
|  | 02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Изучение и описание функций структурного подразделения, в котором проходит практика, его места и роли в реализации целей и задач организации |  |  |
| 2.3. | 02.09.2024  -  02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Анализ использования информационных систем организации |  |  |
| Определение рисков, связанных с функционированием информационных систем и технологий организации, выработка направлений снижения негативного влияния рисков (при их наличии) в рамках поддержания безопасности деятельности организации, достижения оптимальности условий труда персонала, сохранения природной среды, обеспечения устойчивого развития общества |  |  |
| Информационные системы транспортных (транспортно-логистических) предприятий (самостоятельная работа) |  |  |
| 2.4. | 02.09.2024  -  02.09.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Постановка основных целей и задач практики |  |  |
| Изучение: Изучение: Java и Spring Boot для разработки серверной части приложений» |  |  |
| 2.5. | 02.09.2024  -  01.11.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Решение (участие в решении):  Разработка базы данных с frontend и backend на основе Spring Boot |  |  |
| 2.6. | 02.09.2024  -  01.11.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | Выполнение индивидуального задания на практику |  |  |
| 3. | 01.11.2024 | Кафедра «Информатики, систем управления и телекоммуникаций» | 3 этап Заключительный  Подготовка отчета по практике |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель практики от университета |  | Н.Г. Павлова |  |
|  | *(подпись)* |  | *(дата)* |
| Руководитель практики от профильной организации |  | Н.Г. Павлова |  |
| МП | *(подпись)* |  | *(дата)* |
| Обучающийся |  | Д.А. Пчелкина |  |
|  | *(подпись)* |  | *(дата)* |

**ХАРАКТЕРИСТИКА**

**на обучающегося 4 курса, очной формы обучения,  
по направлению подготовки *09.03.02 «Информационные системы и технологии»***

**института экономики, управления и права  
ФГБОУ ВО «Волжский государственный университет водного транспорта»**

***Пчелкина Дениса Андреевича***

В период с *02.09.2024* по *01.11.2024* обучающийся *Пчелкин Денис Андреевич*

проходил (а) производственную практику (эксплуатационную) в *Волжский государственный университет водного транспорта, г. Н.Новгород.*

За время прохождения практики изучил (а) вопросы, связанные с Java и Spring Boot для разработки серверной части приложений», Разработка базы данных с frontend и backend на основе Spring Boot.

При прохождении практики проявил (а) себя с положительной стороны. Ознакомился (лась)

со структурой организации, с нормативно-правовой и учредительной документацией. Место проведения практики посещал (а) регулярно в соответствии с графиком и программой прохождения практики. К должностным обязанностям и поставленным задачам относился (лась) с вниманием, проявляя интерес к работе.

*(отношение к делу; реализация умений и навыков)*

Все необходимые по программе практики компетенции освоены в полном объеме.

Оценка прохождения практики обучающимся по пятибалльной системе\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель практики от профильной организации |  | Н.Г. Павлова |  |
| МП | *(подпись)* |  | *(дата)* |

**Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Волжский государственный университет водного транспорта»**

**АТТЕСТАЦИОННЫЙ ЛИСТ**

***Пчелкина Дениса Андреевича***

Обучающийся 4 курса очной формы обучения по направлению подготовки *09.03.02 «Информационные системы и технологии»*

успешно прошел производственную практику (эксплуатационную)

с *02.09.2024* по *01.11.2024*

в организации: *Волжский государственный университет водного транспорта, г. Н.Новгород*

| **Компетенция** | **Результат освоения** |
| --- | --- |
| ПК-2. Понимание основных модулей и компонентов Spring и их роли в создании приложений. | освоена в полном объеме |
| ПК-3. Умение конфигурировать и использовать Spring Data JPA для работы с реляционными базами данных например PostgreSQL, | освоена в полном объеме |
| УК-8. Понимание работы с Maven или Gradle для сборки проектов, управления зависимостями и конфигурации. | освоена в полном объеме |

|  |  |  |
| --- | --- | --- |
| Руководитель практики  от профильной организации  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Г. Павлова  *(подпись)* |  | Руководитель практики  от ФГБОУ ВО «ВГУВТ»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Г. Павлова  *(подпись)* |

|  |
| --- |
|  |
|  |

Этот код конфигурирует безопасность веб-приложения с использованием Spring Security.

1. **Аннотации**:
   * @Configuration, @EnableWebSecurity, и @EnableGlobalMethodSecurity(prePostEnabled = true) обозначают, что класс является конфигурацией безопасности, включающей глобальное управление доступом по аннотациям (например, @PreAuthorize).
2. **SecurityFilterChain:**
   * Настраивает правила безопасности для HTTP-запросов:
     + Разрешает доступ всем пользователям к корневой странице ("/"), странице регистрации ("/registration") и стилям ("/style/\*\*").
     + Требует авторизации для всех остальных запросов.
     + Настраивает форму входа с пользовательской страницей /login и разрешает всем доступ к ней.
     + Настраивает выход (logout), доступный всем.
3. **AuthenticationManager:**
   * Настраивает аутентификацию, используя userService для загрузки данных о пользователях.
   * Использует NoOpPasswordEncoder, который не шифрует пароли (что обычно не рекомендуется для продакшн-окружения).

|  |
| --- |
|  |

Этот класс конфигурирует MVC-настройки для приложения:

1. **Аннотация @Configuration**:
   * Указывает, что этот класс является конфигурацией Spring.
2. **Поле uploadPath**:
   * Загружает путь для сохранения файлов из свойства upload.path, указанного в настройках (например, в application.properties).
3. **Метод addViewControllers**:
   * Регистрирует контроллер для маршрута /login, который отображает шаблон login (используется для страницы входа).
4. **Метод addResourceHandlers**:
   * Настраивает доступ к статическим ресурсам:
     + Ресурсы из директории, указанной в uploadPath, доступны по URL /img/\*\*.
     + Ресурсы стилей из папки classpath:/style/ доступны по URL /style/\*\*.

Эти настройки позволяют системе отображать изображения и стили, а также назначить шаблон для страницы логина.

|  |
| --- |
|  |

Контроллер **AdminController** управляет доступом к административным функциям приложения:

1. **Аннотации**:
   * @Controller указывает, что этот класс является контроллером Spring MVC.
   * @RequestMapping("/admin") назначает базовый URL для всех маршрутов внутри контроллера — /admin.
   * @PreAuthorize("hasAuthority('ADMIN')") ограничивает доступ ко всем методам класса, позволяя его только пользователям с правом ADMIN.
2. **Поле userRepository**:
   * UserRepository инжектируется через @Autowired, чтобы контроллер мог взаимодействовать с базой данных для получения пользователей.
3. **Метод viewUsers**:
   * Обрабатывает GET-запрос по адресу /admin/users.
   * Загружает всех пользователей из базы данных, используя userRepository.findAll().
   * Передает список пользователей в модель, добавляя его как атрибут users.
   * Возвращает шаблон admin/users для отображения списка пользователей.

Этот контроллер служит для отображения страницы с пользователями, доступной только администраторам.

|  |
| --- |
|  |
|  |

**CourseController** управляет операциями, связанными с курсами, такими как просмотр курсов, регистрация и отмена регистрации на курс:

1. **Поле courseRepository и userRepository**:
   * Эти репозитории, инжектируемые через @Autowired, обеспечивают доступ к данным о курсах и пользователях в базе данных.
2. **Метод viewCourses**:
   * Обрабатывает GET-запросы по /courses.
   * Получает текущего пользователя, используя principal для поиска пользователя по имени.
   * Загружает все курсы и передает их в модель, а также добавляет в модель текущего пользователя как currentUser.
   * Возвращает шаблон course\_select для отображения списка курсов.
3. **Метод enrollInCourse**:
   * Обрабатывает POST-запросы по адресу /courses/enroll/{courseId}, регистрируя пользователя на курс.
   * Находит курс и пользователя по идентификаторам. Если пользователь еще не зарегистрирован на курсе, добавляет его и сохраняет изменения.
   * Перенаправляет обратно на страницу /courses после регистрации.
4. **Метод unenrollFromCourse**:
   * Обрабатывает POST-запросы по адресу /courses/unenroll/{courseId}, позволяя пользователю отменить регистрацию на курсе.
   * Находит курс и пользователя по идентификаторам. Если пользователь зарегистрирован на курсе, удаляет его из списка участников и сохраняет изменения.
   * Перенаправляет обратно на страницу /courses после отмены регистрации.

Этот контроллер позволяет пользователю регистрироваться и сниматься с курсов, а также просматривать доступные курсы.

|  |
| --- |
|  |
|  |
|  |

**MessageController** управляет отображением и добавлением сообщений пользователей, а также загрузкой файлов.

1. **Поле messageRepository**:
   * Репозиторий, инжектируемый через @Autowired, обеспечивает доступ к данным о сообщениях в базе данных.
2. **Поле uploadPath**:
   * Указывает путь к директории для загрузки файлов.
3. **Метод greeting**:
   * Обрабатывает GET-запрос на корневой URL (/), отображая страницу приветствия.
   * Проверяет, вошел ли пользователь в систему (user != null). Если да, добавляет имя пользователя в модель для отображения, а также устанавливает атрибут isLoggedIn для проверки состояния входа.
   * Возвращает шаблон greeting.
4. **Метод main**:
   * Обрабатывает GET-запросы по адресу /main, отображая страницу с комментариями.
   * Добавляет в модель информацию о состоянии входа пользователя и его имя, если он авторизован.
   * Загружает все сообщения из базы данных и добавляет их в модель для отображения на странице.
   * Возвращает шаблон comment.
5. **Метод add**:
   * Обрабатывает POST-запросы по адресу /main для добавления нового сообщения.
   * Создает новое сообщение UserMessage с текстом, пользователем и рейтингом.
   * Если загружен файл, генерирует уникальное имя файла и сохраняет его в директорию uploadPath.
   * Сохраняет сообщение в базе данных и добавляет все сообщения в модель для отображения.
   * Возвращает шаблон comment для отображения обновленного списка сообщений.

Этот контроллер позволяет пользователям просматривать и добавлять комментарии с рейтингом и загружать файлы, которые привязываются к сообщениям.

|  |
| --- |
|  |
|  |

Этот **RegistrController** управляет регистрацией новых пользователей. Основные компоненты:

1. **Поле userRepository**:
   * Репозиторий, инжектируемый через @Autowired, обеспечивает доступ к данным о пользователях в базе данных.
2. **Метод registration**:
   * Обрабатывает GET-запрос на /registration, чтобы отобразить страницу регистрации.
   * Добавляет в модель доступные роли (Role.values()) и CSRF-токен для защиты от CSRF-атак.
   * Возвращает шаблон registration.
3. **Метод addUser**:
   * Обрабатывает POST-запрос на /registration для добавления нового пользователя.
   * Проверяет, существует ли пользователь с таким именем в базе данных. Если да, добавляет сообщение об ошибке в модель и перенаправляет обратно на страницу регистрации.
   * Устанавливает роли для нового пользователя:
     + Если роли выбраны, они преобразуются в Set<Role> и устанавливаются для пользователя.
     + Если роли не выбраны, назначается роль USER по умолчанию.
   * Активирует учетную запись пользователя (setActive(true)) и добавляет пароль с {noop}, чтобы не использовать шифрование.
   * Сохраняет нового пользователя в базе данных и перенаправляет на страницу входа (/login).

Этот контроллер управляет регистрацией и назначением ролей новым пользователям.

|  |
| --- |
|  |
|  |

**UserController** управляет пользователями и предоставляет административные функции, такие как изменение имени пользователя и назначение ролей.

### Основные компоненты:

1. **Аннотации класса**:
   * @Controller — указывает, что этот класс является контроллером Spring MVC.
   * @RequestMapping("/user") — устанавливает префикс URL /user для всех маршрутов в контроллере.
   * @PreAuthorize("hasAuthority('ADMIN')") — ограничивает доступ ко всем методам этого контроллера только пользователям с ролью ADMIN.
2. **Метод getUsers**:
   * Обрабатывает GET-запрос по адресу /user/users.
   * Загружает всех пользователей из базы данных и добавляет их в модель для отображения на странице users.
3. **Метод userSave**:
   * Обрабатывает POST-запрос на /user, позволяя обновить имя пользователя и его роли.
   * Принимает username (новое имя пользователя), form (содержащий роли, выбранные на форме), и userId, указывающий на редактируемого пользователя.
   * Очищает текущий набор ролей пользователя и обновляет его на основании выбранных ролей из формы, затем сохраняет пользователя.
4. **Методы getAllRoles и updateUserRolesFromForm**:
   * getAllRoles() возвращает все роли из перечисления Role в виде множества строк.
   * updateUserRolesFromForm() проверяет роли в form и добавляет их пользователю, если они совпадают с ролями из Role.

Контроллер предназначен для управления данными пользователей и их ролями, но доступ к нему имеют только администраторы.

|  |
| --- |
|  |

Класс Course представляет собой сущность курса с полями для хранения информации о курсе, а также списком пользователей, записанных на него.

1. **Аннотации класса**:
   * @Entity и @Table(name = "course") — указывают, что этот класс является сущностью JPA, которая будет отображена в таблицу course в базе данных.
2. **Поля**:
   * id: указывает уникальный идентификатор курса, автоматически генерируемый (@GeneratedValue(strategy = GenerationType.IDENTITY)).
   * name и description: текстовые поля для хранения названия и описания курса.
   * users: список пользователей, записанных на курс, с аннотацией @ManyToMany, что означает, что связь между Course и User является многозначной. Связь реализована через таблицу course\_user, где course\_id и user\_id указывают на идентификаторы курсов и пользователей соответственно.
3. **Методы доступа (геттеры)**:
   * Позволяют получить значение полей name, description, id и users.
4. **Методы isUserEnrolled и addUser**:
   * isUserEnrolled(User user): проверяет, записан ли конкретный пользователь на курс, возвращая true, если пользователь содержится в списке users.
   * addUser(User user): добавляет пользователя в список users (записывает пользователя на курс).

Этот класс представляет курс и его связь с пользователями, позволяя легко проверять и управлять списком пользователей, записанных на конкретный курс.

|  |
| --- |
|  |

Role является перечислением (enum), представляющим роли пользователей в приложении, такие как USER и ADMIN. Он также реализует интерфейс GrantedAuthority из Spring Security для интеграции с системой безопасности.

### Основные компоненты:

1. **Роли USER и ADMIN**:
   * Роли перечислены как константы и используются для разграничения прав пользователей в приложении.
2. **Реализация интерфейса GrantedAuthority**:
   * Интерфейс GrantedAuthority требует метод getAuthority(), который возвращает имя роли как строку.
   * Метод getAuthority() переопределен для возврата name(), что возвращает строковое представление роли, например, "USER" или "ADMIN".

Эти роли могут использоваться для определения доступа к различным частям приложения. Интеграция с GrantedAuthority позволяет Spring Security распознавать роли при проверке авторизации пользователя.

|  |
| --- |
|  |
|  |

Класс User представляет сущность пользователя и реализует интерфейс UserDetails для интеграции с системой безопасности Spring Security. Рассмотрим его основные компоненты:

### Основные поля:

1. **Идентификатор**:
   * id — уникальный идентификатор пользователя, автоматически генерируется с помощью SequenceGenerator.
2. **Имя пользователя и пароль**:
   * username — имя пользователя для аутентификации.
   * password — пароль пользователя.
3. **Статус активности**:
   * active — флаг, указывающий, активен ли пользователь в системе.
4. **Роли пользователя**:
   * roles — набор ролей пользователя, отображаемый с помощью @ElementCollection и @Enumerated(EnumType.STRING). Роли хранятся в отдельной таблице user\_role с внешним ключом user\_id.

### Методы интерфейса UserDetails:

* **getAuthorities()** — возвращает коллекцию прав пользователя, на основе его ролей.
* **isAccountNonExpired(), isAccountNonLocked(), isCredentialsNonExpired(), isEnabled()** — методы, определяющие статус аккаунта пользователя (не истекший, не заблокированный и т. д.). Эти методы возвращают true, позволяя контролировать доступ на основе активности пользователя.

### Дополнительные методы:

* **toString()** — переопределенный метод для удобного отображения информации о пользователе, включая его ID, имя пользователя, роли и статус активности.

Этот класс позволяет Spring Security управлять аутентификацией и авторизацией, определяя роли и статус пользователя в системе.

|  |
| --- |
|  |

Класс UserMessage представляет собой сущность, которая хранит сообщение пользователя, связанное с его учетной записью.

### Поля:

1. **text** — текст сообщения, которое отправляет пользователь.
2. **filename** — имя файла, прикрепленного к сообщению (если есть).
3. **id** — уникальный идентификатор сообщения, генерируемый автоматически.
4. **author** — связь с сущностью User, представляющая автора сообщения. Эта связь устанавливается через аннотацию @ManyToOne с параметром fetch = FetchType.EAGER, что означает загрузку пользователя вместе с сообщением.
5. **rating** — рейтинг сообщения, присвоенный пользователем (например, лайки или оценки).

### Конструкторы:

* **UserMessage()** — конструктор по умолчанию.
* **UserMessage(String text, User user, Integer rating)** — конструктор для создания нового сообщения с текстом, автором и рейтингом.

### Методы:

1. **getAuthorName()** — возвращает имя пользователя-автора сообщения. Если автор не установлен (например, при удалении пользователя), возвращается строка "<none>".
2. **getUserRole()** — возвращает роль пользователя-автора сообщения. Если роли пользователя нет, возвращается строка "Без роли". Если роли присутствуют, берется первая роль.

Класс UserMessage используется для хранения сообщений пользователей в базе данных с дополнительной информацией о пользователе (авторе), прикрепленных файлах и рейтинге. Это позволяет легко управлять комментариями и оценками, а также ассоциировать сообщения с учетными записями пользователей.

|  |
| --- |
|  |

Интерфейс CourseRepository расширяет JpaRepository, что позволяет использовать все стандартные операции для работы с сущностью Course в базе данных. Он автоматически предоставляет методы для выполнения базовых CRUD (создание, чтение, обновление, удаление) операций, таких как:

* **save(Course course)** — для сохранения или обновления объекта курса.
* **findById(Long id)** — для поиска курса по его идентификатору.
* **findAll()** — для получения всех курсов.
* **deleteById(Long id)** — для удаления курса по его идентификатору.

Использование JpaRepository позволяет избежать необходимости вручную реализовывать методы для этих операций, что значительно упрощает работу с базой данных.

|  |
| --- |
|  |

Интерфейс **MessageRepository** расширяет CrudRepository, что позволяет использовать стандартные CRUD операции для работы с сущностью UserMessage. Он предоставляет методы для:

* **save(UserMessage message)** — для сохранения нового сообщения или обновления существующего.
* **findById(Long id)** — для поиска сообщения по его идентификатору.
* **findAll()** — для получения всех сообщений.
* **deleteById(Long id)** — для удаления сообщения по его идентификатору.

Преимущество использования CrudRepository заключается в автоматическом предоставлении этих операций без необходимости вручную их реализовывать. Это упрощает взаимодействие с базой данных и уменьшает объем кода.

|  |
| --- |
|  |

Интерфейс UserRepository расширяет JpaRepository, что позволяет использовать стандартные операции с сущностью User и добавляет несколько пользовательских методов для работы с пользователями:

* **findByUsername(String username)** — ищет пользователя по имени пользователя (username). Возвращает Optional<User>, чтобы избежать ошибки в случае отсутствия пользователя с данным именем.
* **findById(Long id)** — ищет пользователя по его идентификатору (id). Этот метод также возвращает Optional<User>, что позволяет безопасно работать с отсутствующими данными.

Использование JpaRepository предоставляет все базовые CRUD методы (например, save, findAll, deleteById), а добавленные методы позволяют легко искать пользователей по имени и идентификатору, что полезно при аутентификации

|  |
| --- |
|  |

**UserService** реализует интерфейс UserDetailsService, что позволяет интегрировать его с механизмом аутентификации Spring Security.

* В методе **loadUserByUsername(String username)** сервис пытается найти пользователя в базе данных по имени пользователя, используя метод findByUsername из UserRepository.
* Если пользователь не найден, выбрасывается исключение RuntimeException с сообщением "User not found".
* Если пользователь найден, он возвращается как объект, который реализует интерфейс UserDetails. Это необходимо для того, чтобы Spring Security мог работать с данными о пользователе (например, для аутентификации и авторизации).

Таким образом, этот сервис отвечает за загрузку данных пользователя для системы безопасности и интегрируется с механизмами аутентификации Spring Security.

|  |
| --- |
|  |

Этот фрагмент представляет собой макрос в шаблонизаторе FreeMarker, который генерирует базовую структуру HTML-страницы. Он включает в себя подключение необходимых стилей и скриптов, а также подгрузку навигационной панели через navbar.ftl. Также есть контейнер для основной части страницы, в который будет вставляться динамическое содержимое через <#nested>.

1. **HTML-шаблон**:
   * Заголовок страницы с мета-информацией, настройками для мобильных устройств и подключением Bootstrap.
   * Ссылка на внешний CSS файл для стилей страницы (например, style.css).
2. **Подключение навигационной панели**:
   * Используется <#include "navbar.ftl">, что подключает внешний файл для навигации. Это шапка сайта, которая будет отображаться на всех страницах.
3. **Основной контейнер**:
   * Вставляет содержимое в контейнер с классом mt-5 (вставка с отступом сверху). Это место, где будет динамически вставляться контент, переданный в макрос через директиву <#nested>.
4. **Подключение JavaScript**:
   * Включены скрипты для работы с Bootstrap, такие как jQuery, Popper.js и сам Bootstrap JS для обеспечения функциональности, например, раскрывающихся меню или модальных окон.

Это основной шаблон, который может быть использован для создания других страниц. Он задает общую структуру, а конкретное содержимое будет подставляться через другие шаблоны.

|  |
| --- |
|  |
|  |

Этот код представляет два макроса в шаблонизаторе FreeMarker: один для страницы логина/регистрации, а другой — для отображения кнопки выхода (логаута).

### 1. login ****макрос:****

Этот макрос отображает форму для входа или регистрации в зависимости от значения параметра isRegisterForm.

* **Структура:**
  + Форма с двумя полями: "Имя пользователя" и "Пароль".
  + Скрытое поле для CSRF-токена, чтобы предотвратить атаки CSRF.
  + В зависимости от значения isRegisterForm отображается либо форма для логина, либо для регистрации.
  + Кнопка отправки формы меняет текст в зависимости от того, зарегистрироваться ли пользователю (Register) или войти в систему (Login).
  + Если это не форма регистрации, то добавляется ссылка для перехода на страницу регистрации нового пользователя.

### 2. logout ****макрос:****

Этот макрос отображает кнопку выхода для пользователя, если он авторизован (isLoggedIn == true).

* **Структура:**
  + Если переменная isLoggedIn истинна, то показывается имя пользователя, а также форма для выхода из системы с кнопкой "Разлогиниться".
  + Форма отправляет POST-запрос на /logout для выхода из системы, при этом CSRF-токен включен для защиты от атак.

|  |
| --- |
|  |

Этот код представляет собой шаблон для навигационной панели (navbar) с использованием Bootstrap 4 и FreeMarker. Он включает ссылки на различные страницы веб-приложения и кнопку для выхода пользователя.

### 1. ****Подключение внешних шаблонов****

* <#include "security.ftl"> — включает шаблон security.ftl, возможно, для обработки безопасности, как проверка CSRF-токенов или аутентификации.
* <#import "login.ftl" as l> — импортирует макросы из файла login.ftl и позволяет использовать их через префикс l. В этом коде используется @l.logout, что предполагает использование макроса для отображения кнопки выхода.

### 2. ****Навигационная панель (Navbar)****

* Панель выполнена с использованием классов Bootstrap для навигации с темной темой (bg-primary).
* Панель имеет логотип (ссылку на главную страницу /), несколько пунктов меню и выпадающий список для мобильной версии.

**Ссылки на страницы:**

* Главная — ссылка на главную страницу сайта.
* Коментарии — ссылка на страницу комментариев.
* Пользователи — ссылка на страницу пользователей (/user/users).
* Мои курсы — ссылка на страницу с курсами пользователя.

### 3. ****Выход пользователя (Logout)****

* В правой части панели есть вызов макроса <@l.logout/>, который, проверяет, авторизован ли пользователь. Если да, то он показывает кнопку для выхода, а также имя пользователя (из предыдущего кода для макроса logout).

|  |
| --- |
|  |

### 1. ****Проверка наличия данных в сессии****

<#if Session?? && Session.SPRING\_SECURITY\_CONTEXT??>

* Проверка наличия объекта Session и объекта SPRING\_SECURITY\_CONTEXT в сессии.
* Session — это встроенная переменная в FreeMarker, которая ссылается на сессию пользователя.
* SPRING\_SECURITY\_CONTEXT — это объект, предоставляемый Spring Security, который содержит информацию об аутентификации.

### 2. ****Получение аутентификации пользователя****

<#assign auth = Session.SPRING\_SECURITY\_CONTEXT.authentication>

* Если SPRING\_SECURITY\_CONTEXT существует, то из него извлекается объект authentication, который содержит данные о текущем аутентифицированном пользователе.

### 3. ****Скрытое отображение данных аутентификации****

<div style="display: none;">${auth?string}</div>

* Выводит объект аутентификации в скрытом блоке на странице. Это может быть полезно для отладки, чтобы увидеть содержимое объекта аутентификации.

### 4. ****Проверка аутентификации пользователя****

<#if auth?? && auth.isAuthenticated()>

<#assign user = auth.principal>

</#if>

* Если объект auth существует и пользователь аутентифицирован (auth.isAuthenticated()), то присваивается объект пользователя (auth.principal) переменной user.
* auth.principal обычно представляет собой объект, содержащий данные пользователя, такие как имя пользователя, роль и другие атрибуты.

### 5. ****Использование переменной**** user

После выполнения этого кода переменная user будет содержать данные о текущем пользователе, если он аутентифицирован. Эти данные можно использовать для отображения информации о пользователе, например, имени, ролях или других атрибутов.

|  |
| --- |
|  |
|  |

Этот фрагмент кода представляет собой страницу для добавления и отображения комментариев с использованием шаблонизатора FreeMarker. В нем реализована функциональность для создания формы для отправки комментариев и отображения списка уже добавленных сообщений.

1. **Добавление комментария**: в верхней части страницы есть кнопка "Добавить комментарий", при нажатии на которую открывается форма. Форма позволяет пользователю ввести текст комментария, выбрать рейтинг (от 1 до 5), прикрепить файл и отправить данные на сервер.
2. **Отображение сообщений**: внизу страницы отображается список комментариев. Каждый комментарий представлен в виде карточки, в которой показываются:
   * Текст комментария.
   * Рейтинг, если он был выбран (если не установлен, показывается "Не оценено").
   * Роль пользователя, который оставил комментарий.
   * Имя пользователя, автор комментария, отображается в подвале карточки.
   * Если комментарий включает файл, он показывается как изображение (если файл существует).

Код использует стандартные возможности Bootstrap для стилизации и отображения элементов на странице, а также FreeMarker для динамического вывода данных, таких как список сообщений, информация о пользователе и файлы.

|  |
| --- |
|  |
|  |

Этот код представляет страницу с доступными курсами, где пользователи могут просматривать курсы, записываться на них или отказываться от записи, если уже являются участниками курса.

Вот как работает страница:

1. **Структура страницы**:
   * Заголовок "Доступные курсы" в центре страницы.
   * Таблица с курсами, содержащая следующие столбцы:
     + **Название курса** — название каждого доступного курса.
     + **Описание** — описание курса.
     + **Записанные пользователи** — список пользователей, которые уже записаны на курс. Для каждого пользователя отображается его имя.
     + **Действие** — кнопка для записи на курс или для отказа от записи с учетом того, записан ли текущий пользователь на курс.
2. **Запись на курс**:
   * Если пользователь еще не записан на курс, отображается кнопка "Записаться на курс", которая отправляет POST-запрос на сервер по маршруту /courses/enroll/${course.id}. При этом отправляется скрытое поле с ID текущего пользователя и CSRF-токен для безопасности.
3. **Отказ от курса**:
   * Если пользователь уже записан на курс, отображается кнопка "Отказаться от курса". При нажатии на эту кнопку отправляется POST-запрос на сервер по маршруту /courses/unenroll/${course.id}, чтобы удалить пользователя из списка записанных.
4. **Динамическое заполнение**:
   * В блоке <#list courses as course> происходит перебор всех доступных курсов, и для каждого курса выводится его название, описание, а также список пользователей, которые на него записаны.
   * В блоке <#if course.users?seq\_contains(currentUser)> проверяется, записан ли текущий пользователь на курс, и в зависимости от этого отображается соответствующая кнопка.
5. **Использование шаблонизатора FreeMarker**:
   * Для отображения данных используются выражения ${course.name}, ${course.description}, и ${user.username}, которые заменяются на реальные значения при рендеринге страницы.
   * CSRF-защита обеспечивается путем добавления скрытых полей с токеном и параметром в форму.

Это позволяет пользователю взаимодействовать с курсами на платформе, записываться на них или отказываться от участия.

|  |
| --- |
|  |

Этот код представляет страницу приветствия на сайте, с призывом войти в аккаунт или зарегистрироваться.

Вот основные элементы страницы:

1. **Заголовок и текст**:
   * Заголовок "Добро пожаловать на наш сайт!" в центре.
   * Под заголовком текст, предлагающий пользователю войти в систему или зарегистрироваться.
2. **Кнопки**:
   * Две кнопки: одна ведет на страницу входа (/login), другая — на страницу регистрации (/registration).
   * Кнопки имеют класс btn btn-primary, что придает им стиль Bootstrap и делает их привлекательными для пользователя.
3. **Расположение элементов**:
   * Используется Bootstrap для центрирования содержимого на странице. Кнопки расположены в горизонтальном ряду с помощью d-flex justify-content-around, что распределяет их по центру и по краям экрана.

|  |
| --- |
|  |

Этот код представляет страницу входа, используя шаблоны, импортированные из других частей проекта. Вот разбор:

1. **Импорты**:
   * parts/layout.ftl: импортирует макет страницы.
   * parts/login.ftl: импортирует макрос, который определяет форму для входа пользователя.
2. **Основной контент**:
   * <@c.page>: вызывает макрос из layout.ftl, который оборачивает контент в общий шаблон страницы (например, добавляет заголовок, меню и прочее).
   * Внутри страницы вызывается макрос login из login.ftl, который создает форму для входа на сайт. Передаются два аргумента:
     + "/login": это путь, на который отправляется форма для входа.
     + false: указывает, что это форма для входа (не для регистрации), так как в шаблоне формы логина есть различие между этими состояниями.
   * При рендеринге страницы будет отображена форма входа, с полями для имени пользователя и пароля, а также кнопкой для отправки формы. Эта форма отправляется на URL /login.

|  |
| --- |
|  |

Этот код представляет страницу регистрации с формой для создания нового пользователя, используя шаблоны FreeMarker.

Вот разбор работы кода:

1. **Импорты**:
   * <#import "parts/layout.ftl" as c>: импортирует макросы из шаблона layout.ftl, который содержит структуру страницы.
   * <#import "registration.ftl" as r>: импортирует макросы из шаблона registration.ftl, который содержит саму форму для регистрации.
2. **Основной контент**:
   * <@c.page>: вызывает макрос, который оборачивает текущий контент в структуру страницы, как это определено в layout.ftl.
   * Внутри <@c.page> проверяется, есть ли сообщение (через message?exists). Если оно существует, то выводится блок с сообщением в виде информационного алерта: <div class="alert alert-info" role="alert">${message}</div>.
   * Затем добавляется блок с формой регистрации, которая передается в макрос registrationForm из registration.ftl. Для макроса передаются параметры:
     + action="/registration" — путь для отправки формы.
     + roles=roles — список ролей, которые могут быть выбраны пользователем.
3. **Форма регистрации**: В макросе registrationForm создается форма для регистрации:
   * action="${action}": указывает на URL, куда отправляется форма.
   * input type="hidden" name="\_csrf" value="${csrfToken}": скрытое поле для защиты от CSRF атак.
   * **Поле для имени пользователя**: <input type="text" id="username" name="username" class="form-control" required>.
   * **Поле для пароля**: <input type="password" id="password" name="password" class="form-control" required>.
   * **Выбор роли**: создается радиокнопки для выбора роли пользователя. Список ролей передается в макрос через параметр roles. Каждый элемент списка отображается как радиокнопка с меткой.

* **Кнопка отправки формы**: <button type="submit" class="btn btn-primary">Register</button>.

Страница будет отображать форму регистрации с полями для ввода имени пользователя, пароля и выбора роли.

Если есть сообщение (например, об ошибке или успехе), оно будет отображено вверху страницы.

После отправки формы данные отправляются на сервер по пути /registration методом POST.

|  |
| --- |
|  |

Этот код представляет собой страницу, которая отображает список пользователей и их роли в виде таблицы. Вот разбор работы кода:

1. **Импорт макроса**:
   * <#import "parts/layout.ftl" as c>: импортирует макросы из шаблона layout.ftl, который, вероятно, содержит общую структуру страницы (шапка, меню, футер и т.д.).
2. **Основной контент**:
   * <@c.page>: вызывает макрос page из шаблона layout.ftl, что позволяет использовать общую структуру страницы, в которую будет вложен текущий контент.
3. **Заголовок**:
   * <h1>Список пользователей</h1>: отображается заголовок страницы с текстом "Список пользователей".
4. **Таблица пользователей**:
   * **Имя пользователя**.
   * **Роль**.

* Таблица имеет thead, который описывает заголовки столбцов, и tbody, в котором будут отображаться данные о пользователях.

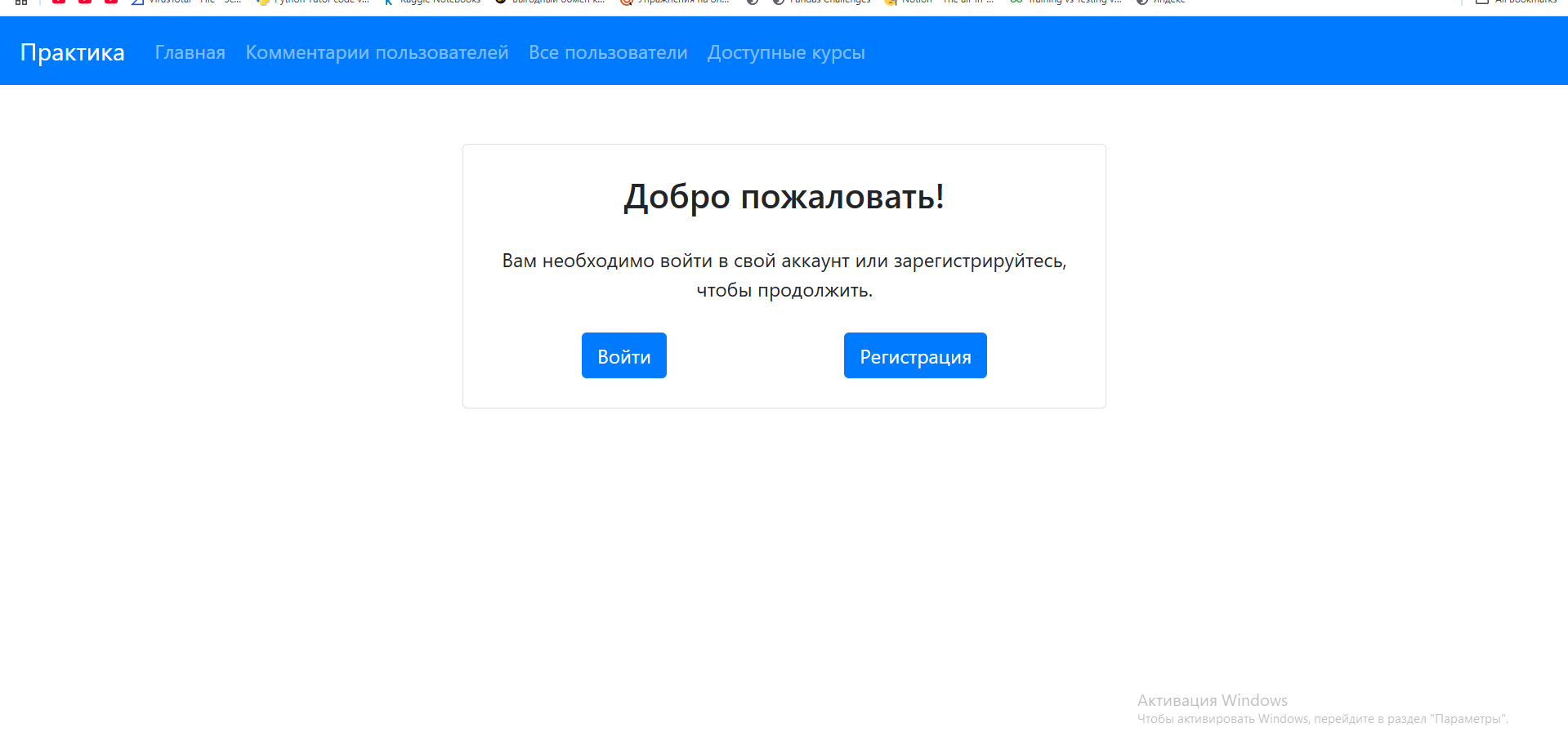
1. **Цикл по пользователям**:

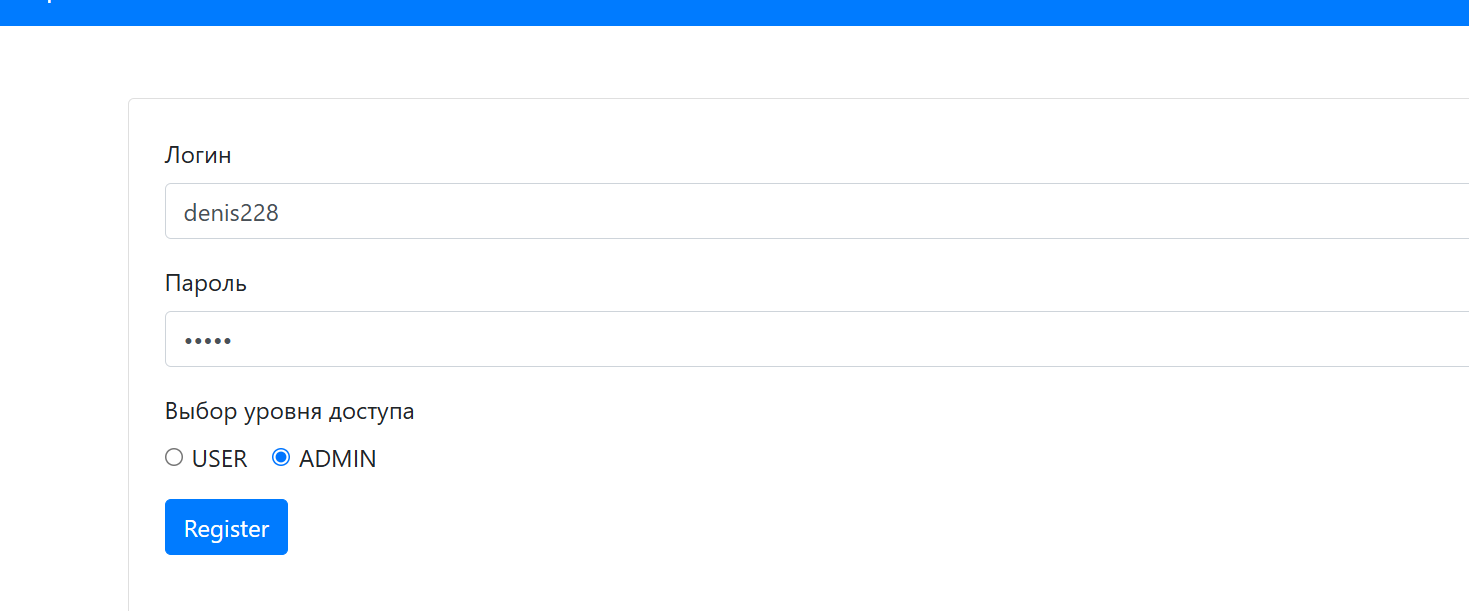
* Внутри тега <tbody> используется цикл FreeMarker (<#list users as user>), чтобы перебрать список пользователей и создать строку для каждого пользователя:

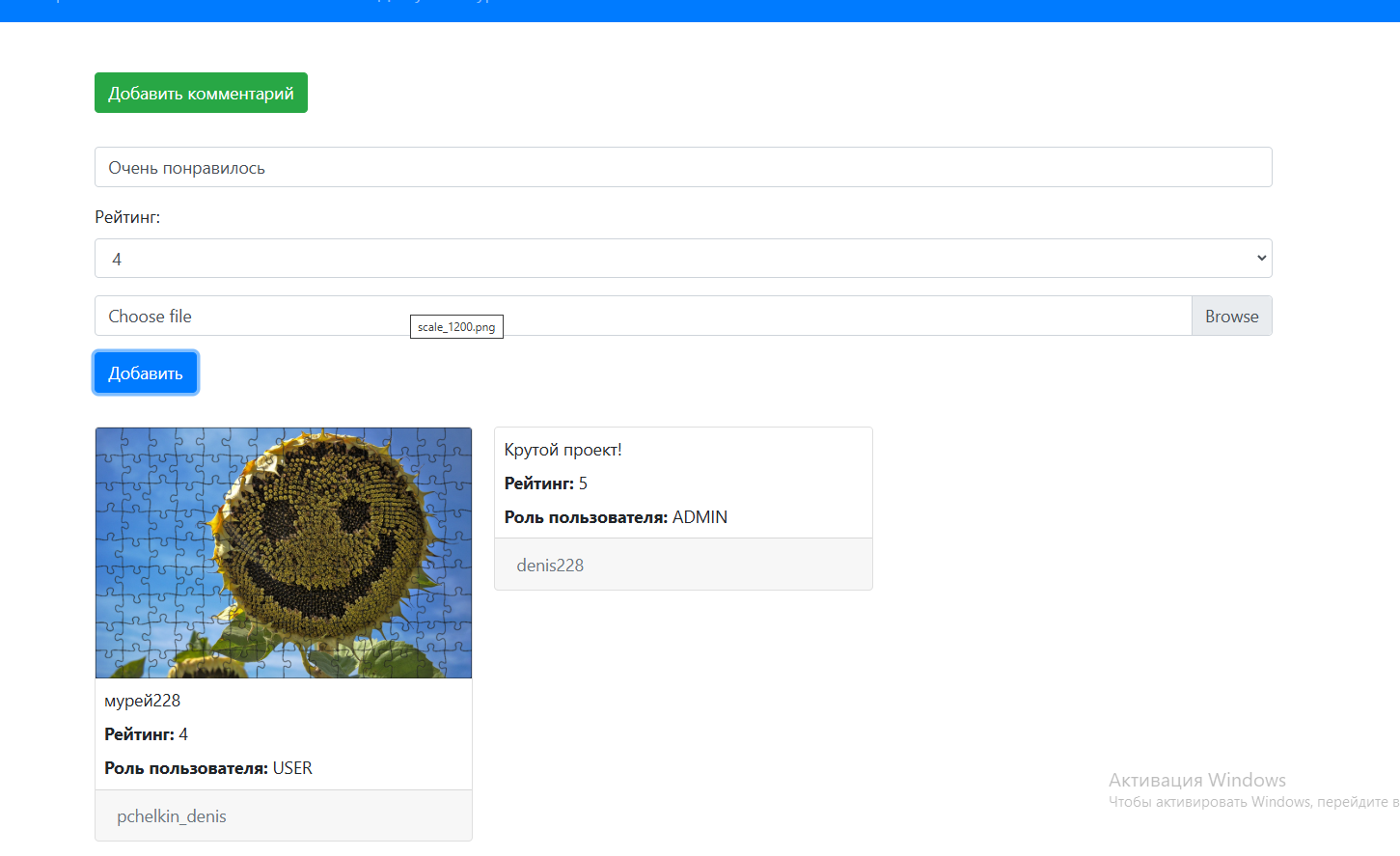
Для каждого пользователя отображается его username (имя пользователя).

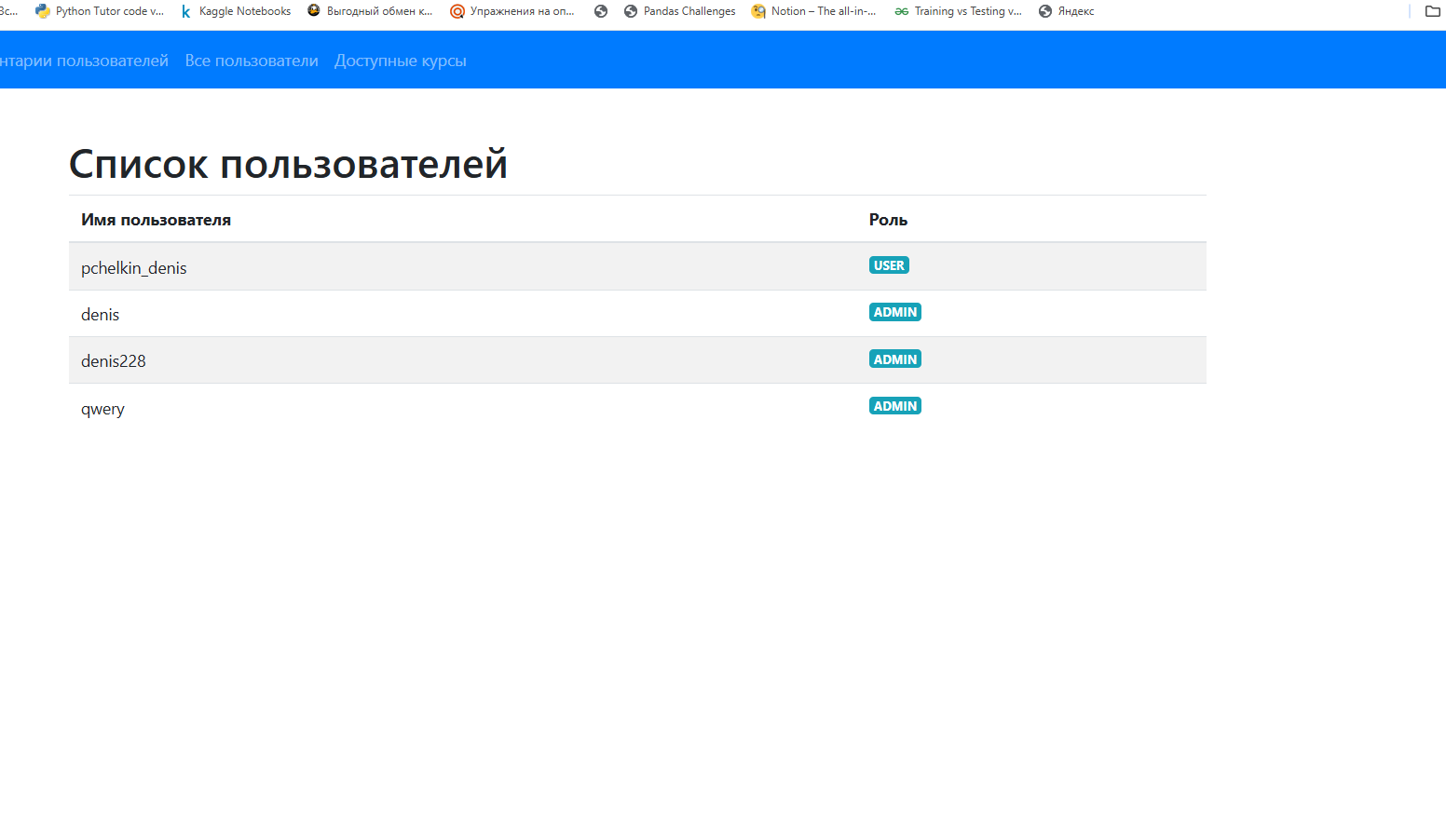
Для каждого пользователя отображаются его роли (список roles), и каждая роль выводится в виде элемента с классом badge badge-info, что стилизует роли как метки.

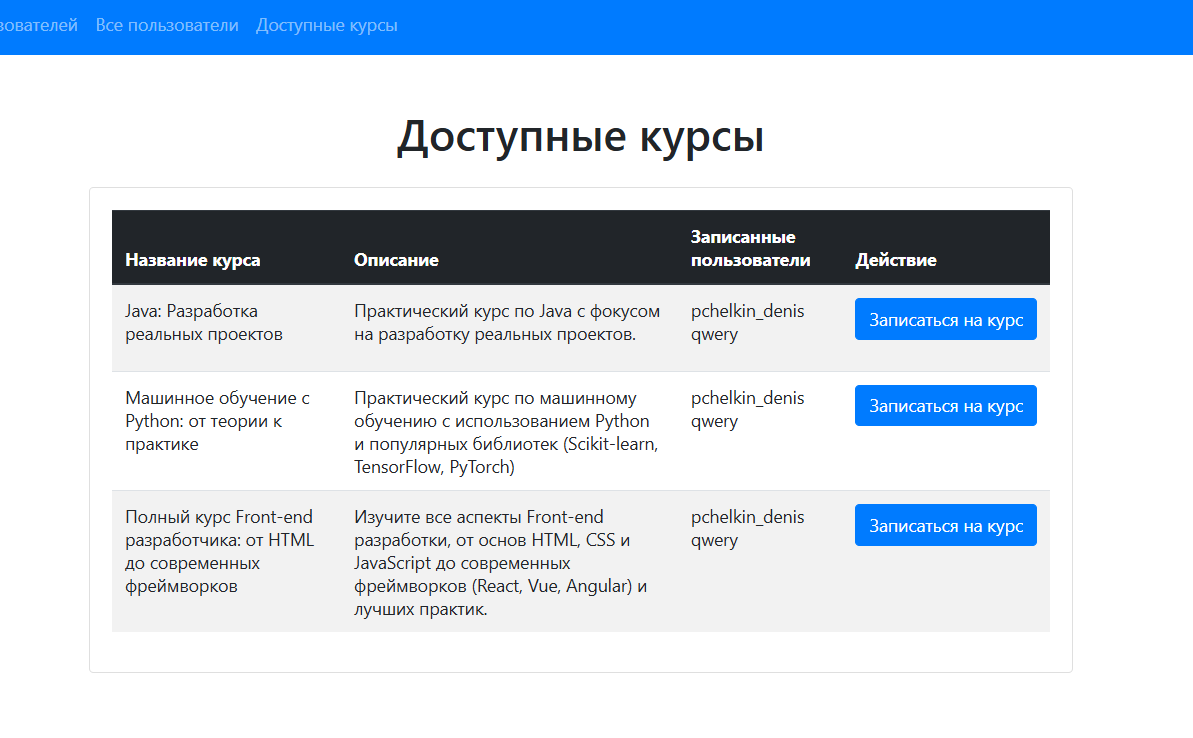
### Интерфейс

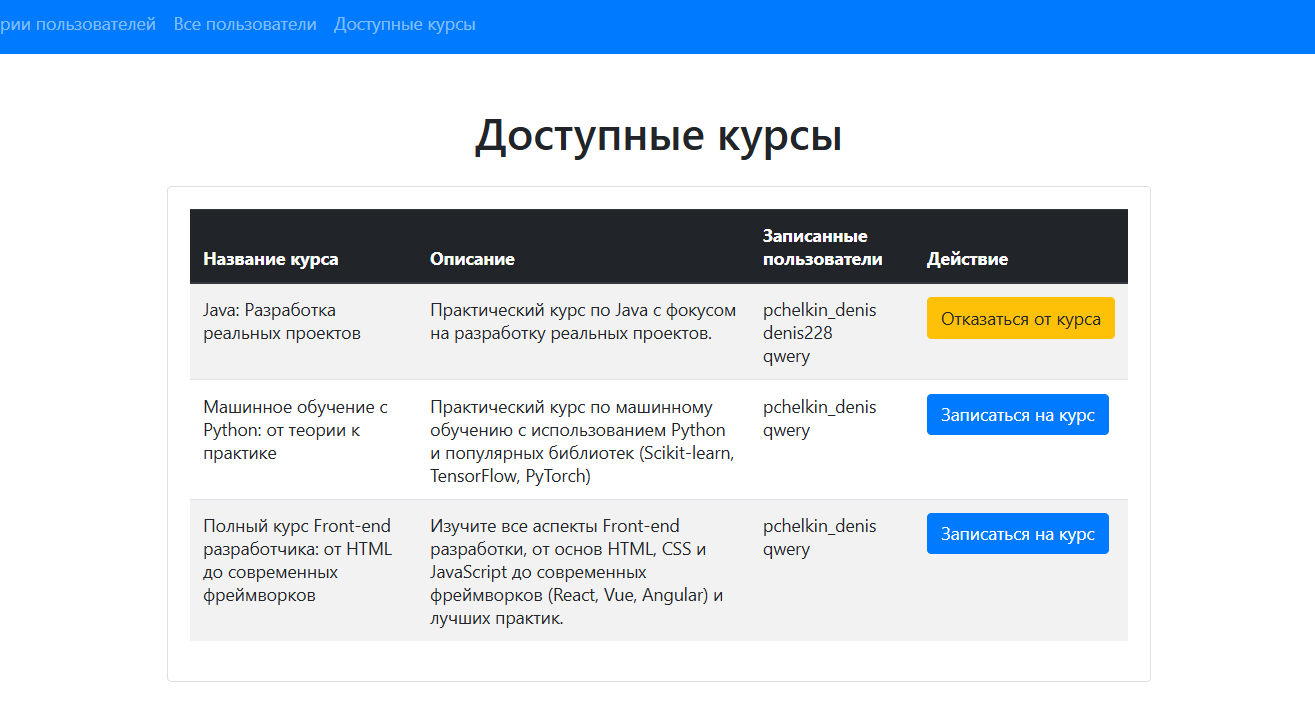






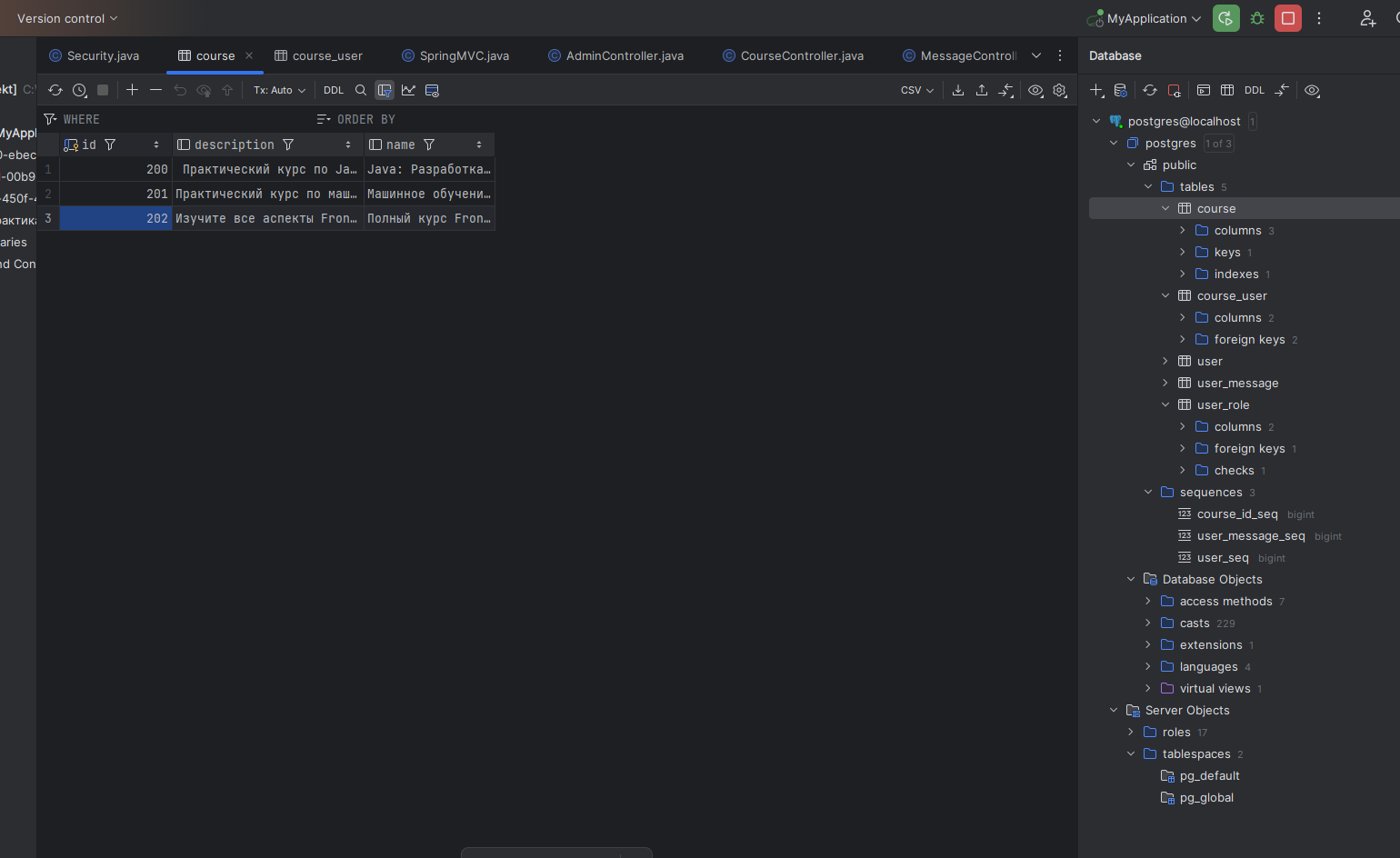




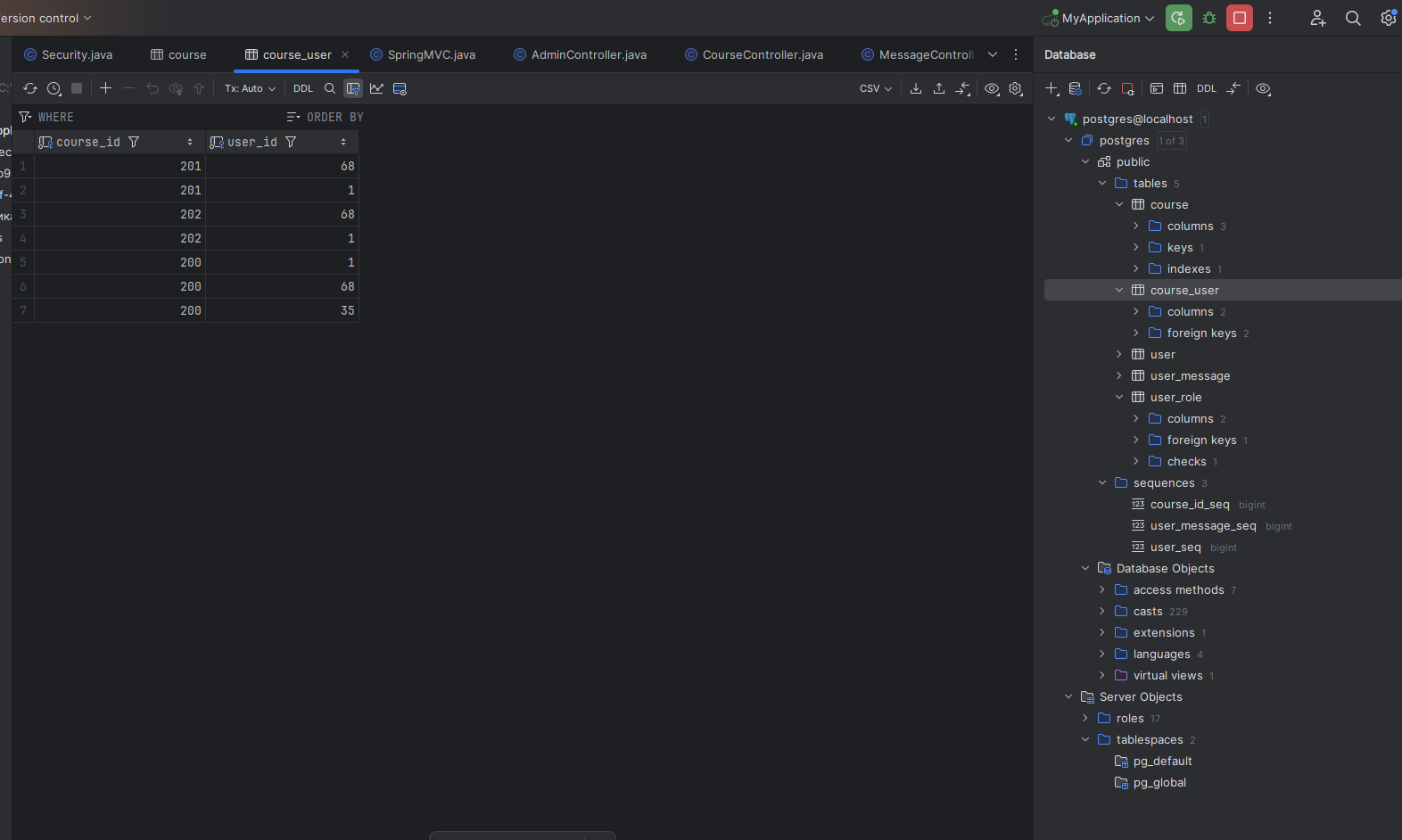


### База данных

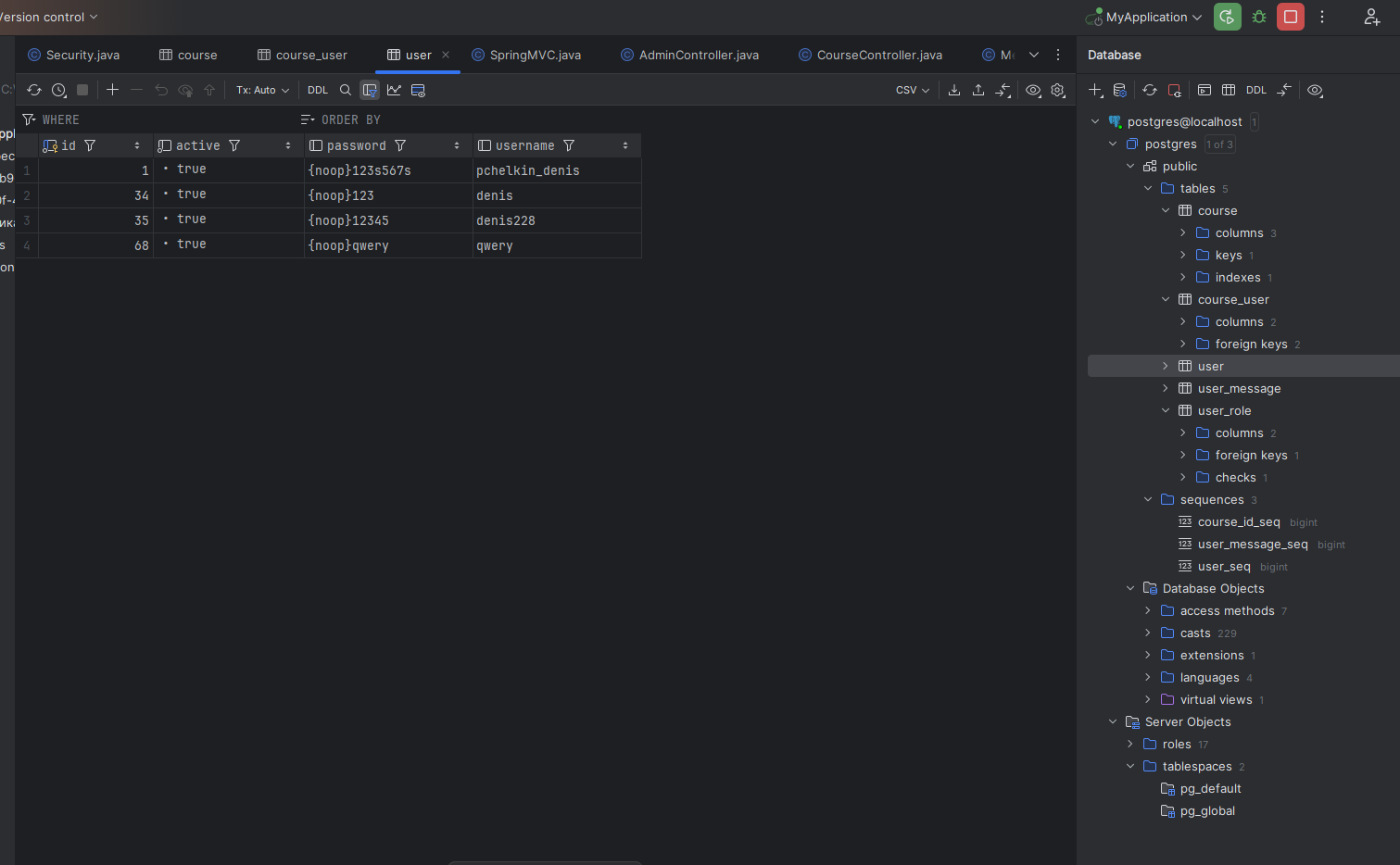
**Таблица course**



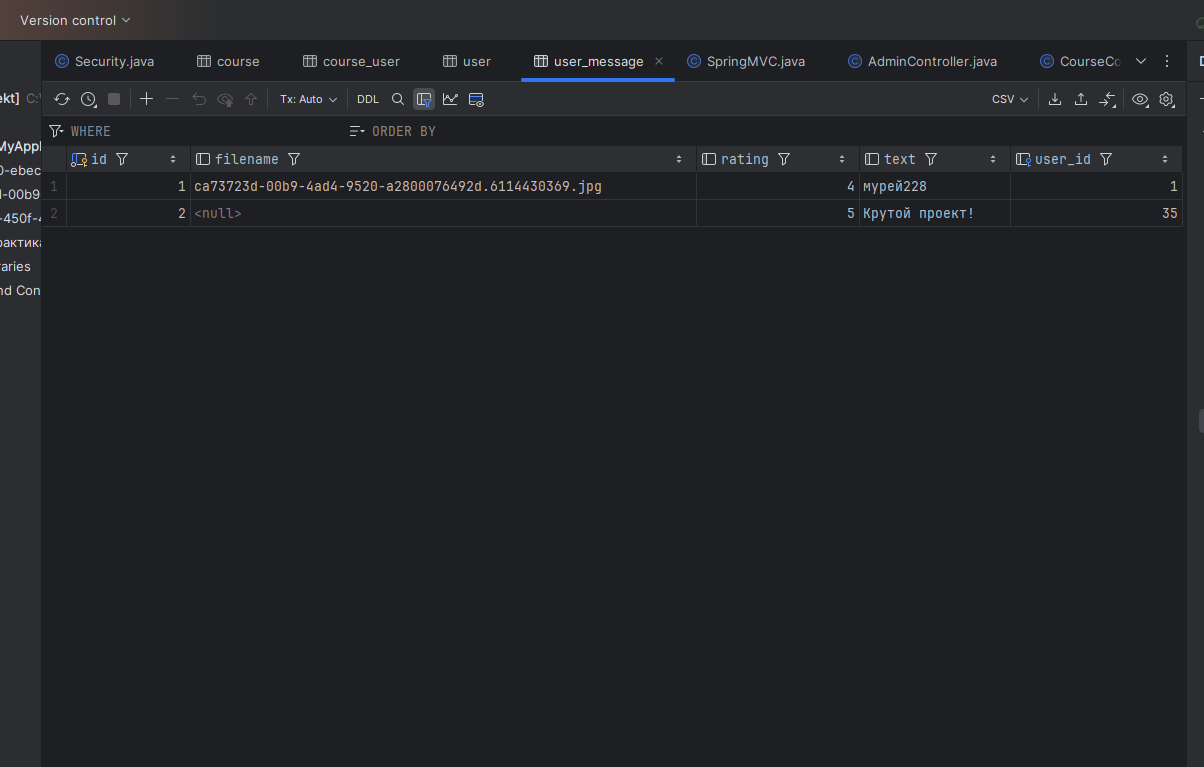
**Таблица course\_user**



**Таблица user**



**Таблица user\_message**



**Таблица user\_role**

