ФГБОУ ВПО Уральский государственный горный университет

Инженерно-экономический факультет

Кафедра информ@тики

**Курсовой проект**

По дисциплине «Технологии программирования»

На тему «Разработка модуля отправки отзывов для рекомендательной системы CS»

Выполнил:

Студент гр. АУБП-21-2

Лаврентьев Р.В.

Проверила:

ст. преп. каф. информатики Волкова Е.А.

Екатеринбург, 2024г.

Оглавление

[1. Постановка задачи 3](#_Toc187368049)

[1.1. Характеристика объекта и проблемной области 3](#_Toc187368050)

[1.2. Постановка целей и задач автоматизации/информатизации 3](#_Toc187368051)

[1.3. Краткое техническое задание 4](#_Toc187368052)

[2. Проектирование системы 6](#_Toc187368053)

[2.1. Моделирование системы 6](#_Toc187368054)

[2.2. Проектирование БД 7](#_Toc187368055)

[2.3. Проектирование интерфейса 8](#_Toc187368056)

[3. Разработка системы 11](#_Toc187368057)

[3.1. Выбор средств разработки 11](#_Toc187368058)

[3.2. Структура проекта 12](#_Toc187368059)

[3.3. Реализация 13](#_Toc187368060)

# Постановка задачи

Целью разработки модуля в рамках системы "Рекомендательная система для игроков CS2" является создание функционала, обеспечивающего сбор, хранение и отображение пользовательских отзывов о работе системы. Это позволит учитывать обратную связь для дальнейшего улучшения качества рекомендаций и удобства системы.

## Характеристика объекта и проблемной области

**Объект автоматизации**

Объектом автоматизации является процесс сбора и отображения пользовательских отзывов о работе рекомендательной системы для игроков CS2.

**Проблемная область**

Современные автоматизированные системы, включая рекомендательные системы, активно внедряются в различные сферы, включая киберспорт. Однако, для повышения качества и эффективности таких систем важно учитывать обратную связь от пользователей, которая позволяет улучшать алгоритмы и интерфейс. В рамках рекомендательной системы для игроков CS2 возникает необходимость в модуле, предоставляющем игрокам возможность оставлять отзывы о работе системы и видеть отзывы других пользователей. Такой модуль позволяет не только собирать обратную связь, но и формировать доверие к системе за счет прозрачности и открытости.

**Существующие решения**

На текущий момент аналогичные системы обратной связи могут быть реализованы через сторонние платформы (например, форумы, социальные сети или сторонние виджеты отзывов). Однако такие подходы не интегрированы непосредственно в систему и не позволяют использовать данные отзывов для улучшения работы алгоритмов.

## Постановка целей и задач автоматизации/информатизации

**Цель автоматизации**

Основной целью автоматизации является разработка модуля сбора и отображения пользовательских отзывов о работе рекомендательной системы для игроков CS2.

**Задачи автоматизации**

* Создание формы для ввода отзыва, включающей следующие поля: текст отзыва, оценка по шкале от 1 до 5.
* Отображение списка отзывов пользователю, с возможностью оценивания полезности отзыва с помощью кнопок лайка и дизлайка.
* Создание отдельной страницы для отображения всех отзывов этого пользователя.

## Краткое техническое задание

**Функциональные требования**

1. Сбор отзывов.

* Форма для добавления отзыва должна включать:
  + Текстовое поле для ввода комментария.
  + Поле для выбора оценки по шкале от 1 до 5.

1. Отображение отзывов

* Отзывы должны быть отображены в виде списка с указанием: текста отзыва, имени автора, оценки.
* Поддержка постраничного отображения списка (пагинация).

1. Хранение данных
   * База данных должна поддерживать следующие параметры:
     + Уникальный идентификатор отзыва.
     + Поля: текст отзыва, оценка, автор.
   * Формат хранения данных: реляционная база данных (PostgreSQL)

**Нефункциональные требования:**

**Требования к показателям назначения:**

* Обработка запроса на добавление отзыва в БД должна занимать не более 3 секунд.
* Загрузка списка отзывов из БД должна занимать не более 5 секунд

**Требования к техническому обеспечению:**

Требования к пользовательским устройствам: должно корректно работать в актуальных версиях браузеров на базе Chromium (Google Chrome, Microsoft Edge, Opera), а также базовая функциональность в Mozilla Firefox и Safari. (Актуальные версии, а также на 2 версии старше)

**Требования к лингвистическому обеспечению АС:**

Языки интерфейса: русский (в дальнейшем добавление английского языка)

**Требования к интерфейсу:**

Интерфейс должен быть выполнен в фирменных цветах: #0F141A, #2C3F52, #CC0000, #ECECEC, # FBFCFC

**Ограничения**

Модуль должен быть разработан с использованием языка программирования Java, фреймворка Spring Boot

# Проектирование системы

## Моделирование системы

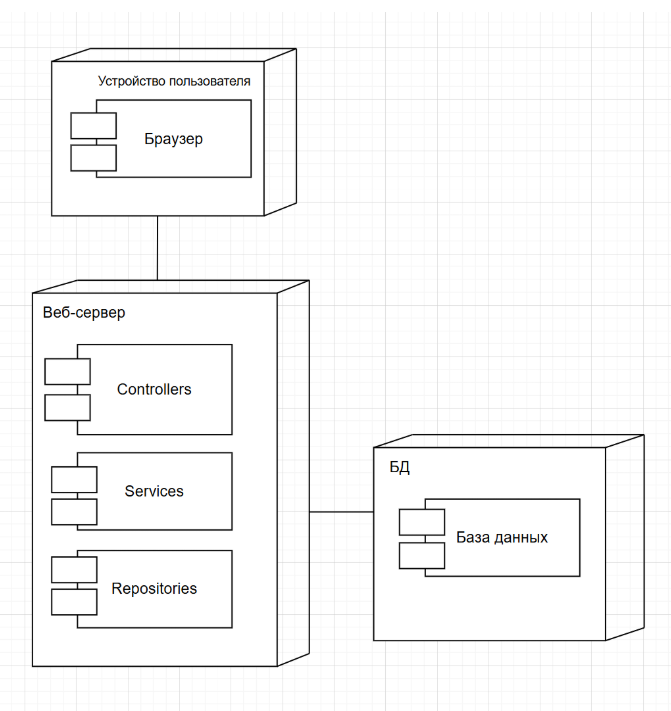
Для реализации системы выбран архитектурный паттерн MVC. Этот паттерн подразумевает создание контроллеров, сервисов и сущностей (Entity).

UML Диаграмма Use Case для данного модуля представлена на рисунке 1:

Изображение выглядит как текст, диаграмма, круг, линия

Автоматически созданное описание

«Рис.1. UML Диаграмма Use Case»



«Рис.2. UML Диаграмма развертывания»

## Проектирование БД

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

«Рис.3. ER-диаграмма базы данных»

Таблица Users: Таблица хранит данные о пользователях системы

* Nickname – уникальное имя пользователя.
* Email – Уникальный адрес электронной почты пользователя.
* Password – пароль, который используется для входа в систему.
* Created\_at – дата и время регистрации пользователя в системе.

Таблица Reviews: Таблица хранит отзывы пользователей о работе системы

* User\_id – Внешний ключ, связывающий отзыв с конкретным пользователем из таблицы users
* Content – Содержание отзыва
* Rating – Рейтинг отзыва в диапазоне от 1 до 5
* Created\_at – дата и время создания отзыва. Устанавливается автоматически при добавлении записи

Таблица Reaction

* Review\_id – внешний ключ, связывающий реакцию с конкретным отзывом из таблицы reviews
* User\_id - внешний ключ, связывающий реакцию с конкретным пользователем из таблицы users
* Reaction\_type – тип реакции, который определяет, является реакция положительной (like) или отрицательной (dislike)
* Created\_at – Дата и время добавления реакции.

Таблица Tags: хранит теги для отзывов, которые может выбрать пользователь для своего отзыва.

* Name – хранит тег для отзыва.

Таблица Review\_tags: промежуточная таблица, которая связывает отзывы и теги через связь “многие ко многим”.

* Review id – внешний ключ, связывающий отзыв с конкретным тегом
* Tag id – внешний ключ, связывающий тег с конкретным отзывом

## Проектирование интерфейса

Скетчи интерфейса приведены на рисунках 4.1. и 4.2.

Изображение выглядит как текст, снимок экрана, Прямоугольник, дизайн

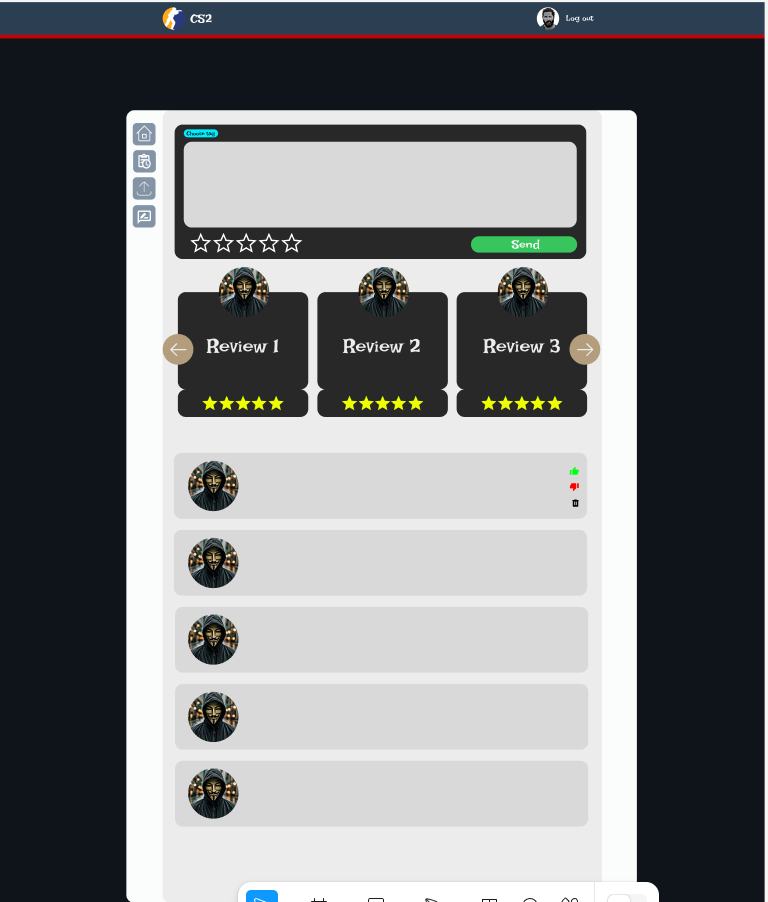
Автоматически созданное описание

«Рис.4.1. Скетч Balsamiq, страница отправки отзывов»

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

«Рис.4.2. Скетч Balsamiq, страница отзывов пользователя»

Дизайн страниц сделан в Figma и представлен на рисунках 5.1 и 5.2. 

«Рис.5.1. Figma, страница отправки пользователя»

Изображение выглядит как снимок экрана, программное обеспечение, текст, Значок на компьютере

Автоматически созданное описание

«Рис.5.2. Figma, страница отзывов пользователя»

# Разработка системы

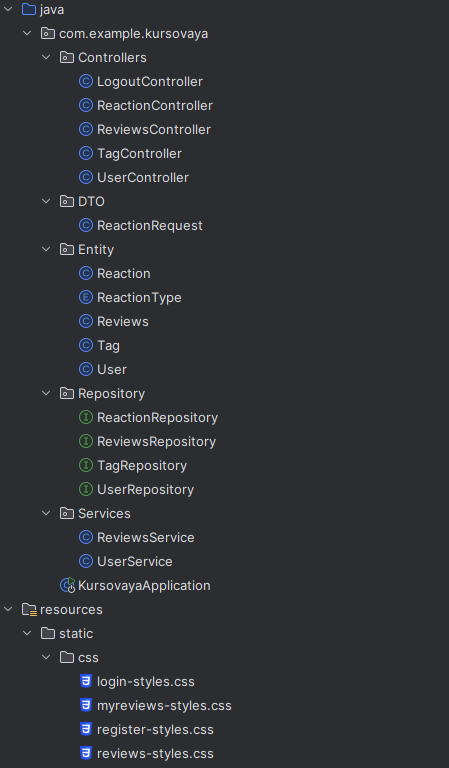
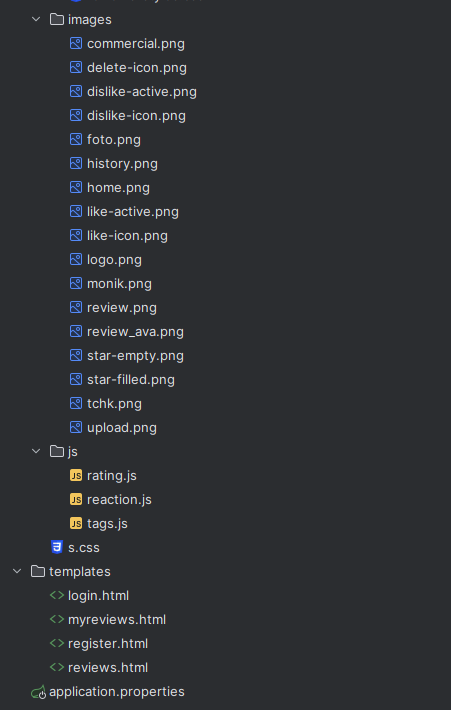
## Выбор средств разработки

Для разработки системы был выбран язык программирования Java с использованием фреймворка Spring Boot и вспомогательных библиотек (Spring Data JPA, Thymleaf, Hibernate). Выбор был обоснован тем, что раннее был опыт разработки систем на этом языке программирования с использованием этого фреймворка. Также Spring Boot упрощает настройку проектов за счет предоставления встроенных зависимостей и шаблонов. Это позволяет быстро разрабатывать RESTful API и веб приложения. Также поддерживает многослойную архитектуру

База данных, используемая в системе – PostgreSQL. Выбор обусловлен тем, что это реляционная база данных с высоким уровнем производительности, безопасности и надежности.

Вспомогательные библиотеки, такие как Spring Data JPA позволяют удобно работать с базой данных, Thymleaf позволяет легко внедрять динамические данные в HTML шаблоны, что идеально подходит для разработки пользовательского интерфейса.

## Структура проекта

**** Дерево проекта представлено на рисунке 6.

«Рис.6. Дерево проекта»

Диаграмма классов представлена на рисунке 7.

Изображение выглядит как снимок экрана, текст, 3D-моделирование, Цифровая сборка

Автоматически созданное описание

«Рис.7. Диаграмма классов»

## Реализация

|  |
| --- |
| @Controller public class ReviewsController {   @Autowired  private ReviewsRepository reviewsRepository;   @Autowired  private ReactionRepository reactionRepository;   // Метод для отображения отзывов текущего пользователя  @GetMapping("/myreviews")  public String showUserReviews(  HttpSession session,  Model model) {  User loggedInUser = (User) session.getAttribute("user"); // Получение текущего пользователя из сессии  if (loggedInUser == null) {  return "redirect:/login"; // Перенаправление на страницу входа, если пользователь не авторизован  }  List<Reviews> userReviews = reviewsRepository.findByUserId(loggedInUser.getId()); // Получение отзывов пользователя из базы данны  model.addAttribute("user", loggedInUser);  model.addAttribute("userReviews", userReviews);  return "myreviews";  }   // Метод для отображения страницы с отзывами  @GetMapping("/reviews")  public String showReviewPage(  @RequestParam(value = "page", required = false, defaultValue = "0") int page,  HttpSession session,  Model model) {  User loggedInUser = (User) session.getAttribute("user"); // Получение текущего пользователя из сессии   if (loggedInUser == null) {  return "redirect:/login"; // Перенаправление на страницу входа, если пользователь не авторизован  }   Pageable topThreePageable = PageRequest.*of*(0, 3, Sort.*by*(Sort.Direction.*DESC*, "createdAt")); // Получение топ-3 отзывов (по дате создания, по убыванию)  List<Reviews> topThreeReviews = reviewsRepository.findTop3Reviews(topThreePageable).getContent();  Pageable pageable = PageRequest.*of*(page, 5, Sort.*by*(Sort.Direction.*DESC*, "createdAt")); // Получение пагинированного списка отзывов (по 5 отзывов на страницу)  Page<Reviews> paginatedReviews = reviewsRepository.findAllWithPagination(pageable);   model.addAttribute("user", loggedInUser);  model.addAttribute("topReviews", topThreeReviews);  model.addAttribute("paginatedReviews", paginatedReviews.getContent());  model.addAttribute("currentPage", page);  model.addAttribute("totalPages", paginatedReviews.getTotalPages());   return "reviews";  }   // Метод для добавления нового отзыва  @PostMapping("/reviews")  public String submitReview(  @RequestParam String content,  @RequestParam int rating,  @RequestParam(required = false) String selectedTag,  HttpSession session,  Model model) {  User loggedInUser = (User) session.getAttribute("user"); // Получение текущего пользователя из сессии  if (loggedInUser == null) {  return "redirect:/login"; // Перенаправление на страницу входа, если пользователь не авторизован  }  if (rating < 1 || rating > 5) { // Проверка корректности рейтинга  model.addAttribute("error", "Rating must be between 1 and 5!");  return "reviews";  }  Reviews review = new Reviews(); // Создание нового отзыва  review.setUser(loggedInUser);  review.setContent(content);  review.setRating(rating);  review.setCreatedAt(LocalDateTime.*now*());  review.setUpdatedAt(LocalDateTime.*now*());  if (review.getTags() == null) {  review.setTags(new HashSet<>());  }   if (selectedTag != null && !selectedTag.isEmpty()) {  Tag tag = tagRepository.findByName(selectedTag)  .orElseGet(() -> {  Tag newTag = new Tag();  newTag.setName(selectedTag);  return tagRepository.save(newTag);  });  review.getTags().add(tag); // Связываем отзыв с тегом  }  reviewsRepository.save(review); // Сохранение отзыва в базе данных  model.addAttribute("message", "Review submitted successfully!");  return "redirect:/reviews";  }   // Метод для удаления отзыва  @PostMapping("/reviews/delete/{id}")  public String deleteReview(@PathVariable Long id, HttpSession session) {   User loggedInUser = (User) session.getAttribute("user"); // Получение текущего пользователя из сессии  if (loggedInUser == null) {  return "redirect:/login"; // Перенаправление на страницу входа, если пользователь не авторизован  }  Reviews review = reviewsRepository.findById(id) // Получение отзыва по его ID или выброс исключения, если он не найден  .orElseThrow(() -> new IllegalArgumentException("Отзыв с ID " + id + " не найден."));   if (!review.getUser().getId().equals(loggedInUser.getId())) { // Проверка, является ли пользователь владельцем отзыва  throw new SecurityException("Вы не можете удалить чужой отзыв.");  }  reviewsRepository.delete(review); // Удаление отзыва из базы данных  return "redirect:/myreviews";  }  } |

«Листинг 1. Контроллер отзывов.»

|  |
| --- |
| // REST-контроллер для управления реакциями (лайки и дизлайки на отзывы) @RestController @RequestMapping("/reactions") public class ReactionController {   private final ReactionRepository reactionRepository;  private final ReviewsRepository reviewsRepository;  private final UserRepository userRepository;   public ReactionController(ReactionRepository reactionRepository, ReviewsRepository reviewsRepository, UserRepository userRepository) {  this.reactionRepository = reactionRepository;  this.reviewsRepository = reviewsRepository;  this.userRepository = userRepository;  }   // Метод для получения всех реакций текущего пользователя  @GetMapping  public ResponseEntity<?> getUserReactions(HttpSession session) {  User loggedInUser = (User) session.getAttribute("user"); // Получение текущего пользователя из сессии  if (loggedInUser == null) {  return ResponseEntity.*status*(401).body(Map.*of*("message", "User not authenticated")); // Возвращение ошибки 401, если пользователь не авторизован  }  List<Reaction> reactions = reactionRepository.findByUser(loggedInUser); // Получение списка реакций пользователя  Map<Long, String> userReactions = reactions.stream() // Преобразование списка реакций в Map, где ключ - ID отзыва, значение - тип реакции  .collect(Collectors.*toMap*(  reaction -> reaction.getReview().getId(),  reaction -> reaction.getReactionType().name()  ));  return ResponseEntity.*ok*(userReactions); // Возвращение списка реакций пользователя  }   // Метод для добавления или обновления реакции  @PostMapping  public ResponseEntity<?> addReaction(@RequestBody ReactionRequest request, HttpSession session) {  User loggedInUser = (User) session.getAttribute("user"); // Получение текущего пользователя из сессии  if (loggedInUser == null) {  return ResponseEntity.*status*(401).body(Map.*of*("message", "User not authenticated")); // Возвращение ошибки 401, если пользователь не авторизован  }  Reviews review = reviewsRepository.findById(request.getReviewId()) // Получение отзыва по ID или выброс исключения, если отзыв не найден  .orElseThrow(() -> new RuntimeException("Review not found"));  Reaction existingReaction = reactionRepository.findByReviewAndUser(review, loggedInUser); // Проверка, существует ли уже реакция на данный отзыв от текущего пользователя  if (existingReaction != null) {  if (existingReaction.getReactionType() == request.getReactionType()) { // Если тип реакции совпадает с новым запросом, удаляем существующую реакцию  reactionRepository.delete(existingReaction);  return ResponseEntity.*ok*().build();  } else {  existingReaction.setReactionType(request.getReactionType()); // Если тип реакции отличается, обновляем её  reactionRepository.save(existingReaction);  return ResponseEntity.*ok*().build();  }  }  Reaction reaction = new Reaction(); // Если реакция отсутствует, создаем новую  reaction.setReview(review);  reaction.setUser(loggedInUser);  reaction.setReactionType(request.getReactionType());  reactionRepository.save(reaction); // Сохраняем новую реакцию в базе данных  return ResponseEntity.*ok*().build();  } } |
|  |

«Листинг 2. Контроллер реакций.»

|  |
| --- |
| public interface ReviewsRepository extends JpaRepository<Reviews, Long> {  @Query("SELECT r FROM Reviews r WHERE r.user.id = :userId ORDER BY r.createdAt DESC")  List<Reviews> findByUserId(@Param("userId") Long userId);  @Query("SELECT r FROM Reviews r ORDER BY r.createdAt DESC")  Page<Reviews> findTop3Reviews(Pageable pageable);  @Query("SELECT r FROM Reviews r ORDER BY r.createdAt DESC")  Page<Reviews> findAllWithPagination(Pageable pageable);  @Query("SELECT r FROM Reviews r LEFT JOIN FETCH r.tags WHERE r.id = :id")  Optional<Reviews> findByIdWithTags(@Param("id") Long id);  } |

«Листинг 3. Репозиторий отзывов.»

Листинги создания таблиц базы данных представлены на листинг 4,5,6,7,8.

CREATE TABLE users (

id SERIAL PRIMARY KEY, -- Уникальный идентификатор пользователя

nickname VARCHAR(50) NOT NULL UNIQUE, -- Имя пользователя

email VARCHAR(100) NOT NULL UNIQUE, -- Электронная почта

password\_hash VARCHAR(255) NOT NULL, -- Пароль

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Дата создания пользователя

);

«Листинг 4. Создание таблицы users в БД.»

CREATE TABLE reviews (

id SERIAL PRIMARY KEY, -- Уникальный идентификатор отзыва

user\_id INT NOT NULL, -- Внешний ключ на пользователя

content TEXT NOT NULL, -- Содержание отзыва

rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5), -- Рейтинг отзыва (1-5)

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Дата создания отзыва

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Дата последнего обновления

FOREIGN KEY (user\_id) REFERENCES users (id) ON DELETE CASCADE -- Связь с таблицей users

);

«Листинг 5. Создание таблицы reviews в БД.»

CREATE TABLE reactions (

id SERIAL PRIMARY KEY, -- Уникальный идентификатор реакции

review\_id INT NOT NULL, -- Внешний ключ на отзыв

user\_id INT NOT NULL, -- Внешний ключ на пользователя

reaction\_type VARCHAR(10) NOT NULL CHECK (reaction\_type IN ('like', 'dislike')), -- Тип реакции

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Дата реакции

FOREIGN KEY (review\_id) REFERENCES reviews (id) ON DELETE CASCADE,

FOREIGN KEY (user\_id) REFERENCES users (id) ON DELETE CASCADE,

UNIQUE (review\_id, user\_id) -- Один пользователь может оставить только одну реакцию на конкретный отзыв

);

«Листинг 6. Создание таблицы reactions в БД.»

CREATE TABLE tags (

id SERIAL PRIMARY KEY, -- Уникальный идентификатор тега

name VARCHAR(50) NOT NULL UNIQUE – Название тега

);

«Листинг 7. Создание таблицы tags в БД.»

CREATE TABLE review\_tags (

review\_id INT NOT NULL, -- Идентификатор отзыва.

tag\_id INT NOT NULL, -- Идентификатор тега

PRIMARY KEY (review\_id, tag\_id), -- Композитный первичный ключ: гарантирует уникальность пар "отзыв-тег".

FOREIGN KEY (review\_id) REFERENCES reviews (id) ON DELETE CASCADE,

FOREIGN KEY (tag\_id) REFERENCES tags (id) ON DELETE CASCADE

);

«Листинг 8. Создание таблицы review\_tags в БД.»

**Скриншоты приложения, демонстрирующие основной функционал приложения:**

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

«Рис.8. Страница отзывов»

Изображение выглядит как снимок экрана, программное обеспечение, текст, Операционная система

Автоматически созданное описание

«Рис.9. Страница отзывов пользователя»