**Поволжский Государственный Университет Телекоммуникаций и Информатики**Кафедра «Программная инженерия»

|  |  |
| --- | --- |
| Сдана на проверку  «\_\_» \_\_\_\_2023 г. | Допустить к защите  «\_\_» \_\_\_\_\_ 2023 г.  Защищена с оценкой  «\_\_\_» \_\_\_\_\_\_\_\_ 2023 г. |

**КУРСОВАЯ РАБОТА**По дисциплине: «Прикладное программирование»  
На тему: «Разработка клиент-серверного web-приложения OpenBoard»  
  
Пояснительная записка

|  |  |
| --- | --- |
| Студент группы ИВТ-11\_\_\_\_\_\_\_ Симонова Л. С.  (роспись) (ФИО)  Руководитель \_\_\_\_\_\_\_\_ к.т.н., доц. Ахметшина Э. Г.  (роспись) (ФИО) | 0 200588 0  (№ зачетной книжки) |

Самара 2023

# Рецензия

# **Содержание**

[Описание предметной области. Актуальность 4](#_Toc3)

[Описание программы. Общие сведения 6](#_Toc4)

[Функциональное назначение 6](#_Toc5)

[Логическая модель базы данных 8](#_Toc6)

[Физическая модель базы данных 10](#_Toc7)

[Диаграмма классов 12](#_Toc8)

[Диаграмма компонентов 14](#_Toc9)

[Диаграмма вариантов использования 16](#_Toc10)

[Диаграмма последовательности 18](#_Toc11)

[Демонстрация работы приложения 20](#_Toc12)

[Список используемых источников 21](#_Toc13)

[Приложение А – Листинг программного кода 22](#_Toc14)

# Описание предметной области. Актуальность

Создание и использование публичных досок объявлений в современном обществе имеет высокую актуальность и оправдание в условиях цифровой эпохи. Этот раздел обсудит несколько ключевых аспектов, делающих публичные доски объявлений, такие как "OpenBoard", востребованными и необходимыми.

Одним из фундаментальных аспектов актуальности публичных досок объявлений является создание механизма для повсеместного доступа к информации. "OpenBoard" предоставляет платформу, где пользователи могут размещать и просматривать объявления без необходимости регистрации. Это важно в современном обществе, где быстрый и неограниченный доступ к информации становится стандартом.

Публичные доски объявлений становятся эффективным инструментом для формирования сообществ и социального взаимодействия. "OpenBoard" создает пространство, где люди с общими интересами или нуждами могут объединяться, обмениваться информацией и поддерживать взаимодействие в онлайн-среде.

Актуальность публичных досок объявлений также заключается в разнообразии контента и событий, предоставляемых пользователями. "OpenBoard" позволяет размещать объявления о различных категориях, включая события, товары, услуги и многое другое. Это открывает возможности для широкого круга пользователей и разнообразия представленной информации.

Создание открытой платформы для размещения объявлений стимулирует обмен мнениями и формирование общественного мнения. "OpenBoard" предоставляет возможность оценивать и комментировать объявления, что способствует общественной дискуссии и помогает пользователям принимать осознанные решения.

Публичные доски объявлений, такие как "OpenBoard", остаются актуальными в условиях современного общества благодаря своей способности обеспечивать доступ к информации, формировать сообщества, разнообразить контент и стимулировать обмен мнениями. Это инструмент, способствующий взаимодействию и обмену информацией в открытом формате.

# Описание программы. Общие сведения

Приложение – веб-ресурс. Установка не требуется.

Язык программирования: JavaScript, Java.

Среда разработки: Intellij IDEA, VS Code.

Объем проекта: 681 МБ (вместе с исходным кодом).

Исходный код (классы и код форм): 1346 строк.

# Функциональное назначение

Функциональное назначение платформы "OpenBoard" охватывает широкий спектр возможностей, предоставляя пользователям удобные инструменты для размещения, просмотра и взаимодействия с объявлениями. Ниже представлены ключевые функциональные аспекты системы:

* **размещение объявлений:** пользователи могут легко создавать и публиковать объявления, предоставляя подробную информацию о событиях, товарах, услугах и других сущностях. Заголовок, описание, категория и местоположение обеспечивают полноту информации.
* **просмотр объявлений:** все пользователи могут свободно просматривать объявления без ограничений доступа. Интуитивный интерфейс позволяет удобно фильтровать и искать объявления по различным параметрам.
* **категории и местоположения:** Объявления организованы по категориям и местоположениям, что облегчает навигацию пользователя и нахождение интересующей информации.
* **комментарии:** Пользователи имеют возможность оставлять комментарии к объявлениям, обсуждать детали или предлагать дополнительную информацию, что способствует взаимодействию на платформе.
* **оценки:** Для выражения своего мнения о полезности или интересности объявлений введена система оценок. Пользователи могут выставлять оценки и оценивать другие объявления.
* **управление категориями и местоположениями:** Пользователи могут предлагать новые категории и местоположения. Система автоматически адаптируется к предложенным данным, обеспечивая широкий спектр категорий для объявлений.
* **поиск и фильтрация:** реализован мощный механизм поиска и фильтрации объявлений, что позволяет пользователям быстро находить интересующую информацию.

Функциональное назначение "OpenBoard" ориентировано на предоставление максимальной функциональности для удобного и эффективного обмена информацией через объявления. Система обеспечивает разнообразные инструменты и возможности взаимодействия на платформе, не требуя при этом создания пользовательских аккаунтов или управления администраторами.

# Логическая модель базы данных

Сущности базы данных "OpenBoard" и их атрибуты:

1. объявление (Announcement):
   * ID (Primary Key): уникальный идентификатор объявления.
   * заголовок (Title): заголовок объявления.
   * описание (Description): подробное описание содержания объявления.
   * дата создания (Creation Date): дата и время создания объявления.
   * категория (Category): категория, к которой относится объявление.
   * местоположение (Location): местоположение, связанное с объявлением.
   * оценка (Rating): оценка пользователей, выставленная объявлению.
2. категория (Category):
   * ID (Primary Key): уникальный идентификатор категории.
   * наименование (Name): название категории.
3. местоположение (Location):
   * ID (Primary Key): уникальный идентификатор местоположения.
   * Город (City): город, в котором расположено местоположение.
   * Регион (Region): регион или район, к которому относится местоположение.
4. оценка (Rating):
   * ID (Primary Key): уникальный идентификатор оценки.
   * Значение (Value): значение оценки.

Связи между сущностями:

* сущность "Объявление" связана с сущностью "Категория" отношением "Many-to-One", так как одно объявление может принадлежать только к одной категории, но у одной категории может быть множество объявлений.
* сущность "Объявление" также связана с сущностью "Местоположение" отношением "Many-to-One", поскольку одно объявление может быть связано только с одним местоположением, но у одного местоположения может быть множество объявлений.
* сущность "Объявление" также связана с сущностью "Оценка" отношением "Many-to-One", так как одно объявление может иметь одну или более оценок, но каждая оценка принадлежит только к одному объявлению.

Таким образом, модель данных "OpenBoard" обеспечивает структурированное хранение информации о объявлениях, их категориях, местоположениях и оценках, а связи между сущностями позволяют эффективно организовывать и извлекать данные для обеспечения функциональности системы.

# Физическая модель базы данных

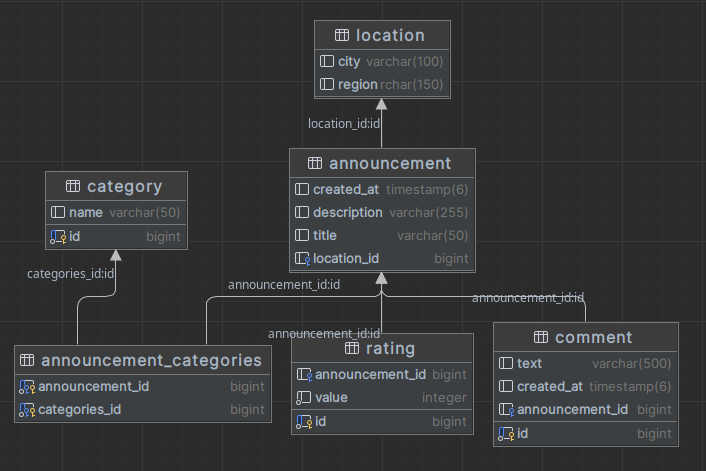


рис. 1– ER диаграмма базы данных

Физическая модель базы данных системы OpenBoard описывается созданными таблицами, их полями и связями. Ниже представлено подробное описание каждой таблицы:

1. таблица "category" (Категории):
   * id (bigint, primary key): уникальный идентификатор категории.
   * name (varchar(50)): название категории.
2. таблица "location" (Местоположения):
   * id (bigint, primary key): Уникальный идентификатор местоположения.
   * city (varchar(100)): Город.
   * region (varchar(150)): Регион.
3. таблица "announcement" (Объявления):
   * id (bigint, primary key): Уникальный идентификатор объявления.
   * created\_at (timestamp(6)): Дата и время создания объявления.
   * description (varchar(255)): Описание объявления.
   * title (varchar(50)): Заголовок объявления.
   * location\_id (bigint): Внешний ключ, связанный с таблицей "location".
4. таблица "comment" (Комментарии):
   * id (bigint, primary key): Уникальный идентификатор комментария.
   * text (varchar(500)): Текст комментария.
   * created\_at (timestamp(6)): Дата и время создания комментария.
   * announcement\_id (bigint): Внешний ключ, связанный с таблицей "announcement".
5. таблица "rating" (Оценки):
   * id (bigint, primary key): Уникальный идентификатор оценки.
   * announcement\_id (bigint): Внешний ключ, связанный с таблицей "announcement".
   * value (integer, not null): Значение оценки.
6. таблица "announcement\_categories" (Связь между объявлениями и категориями):
   * announcement\_id (bigint): Внешний ключ, связанный с таблицей "announcement".
   * categories\_id (bigint): Внешний ключ, связанный с таблицей "category".
   * Определен как составной primary key (announcement\_id, categories\_id).

Эта физическая модель представляет собой структуру базы данных, в которой объявления могут принадлежать к разным категориям, имеют местоположение, к ним могут быть оставлены комментарии и оценки. Модель позволяет эффективно хранить и связывать данные для функционирования платформы OpenBoard.

# Диаграмма классов

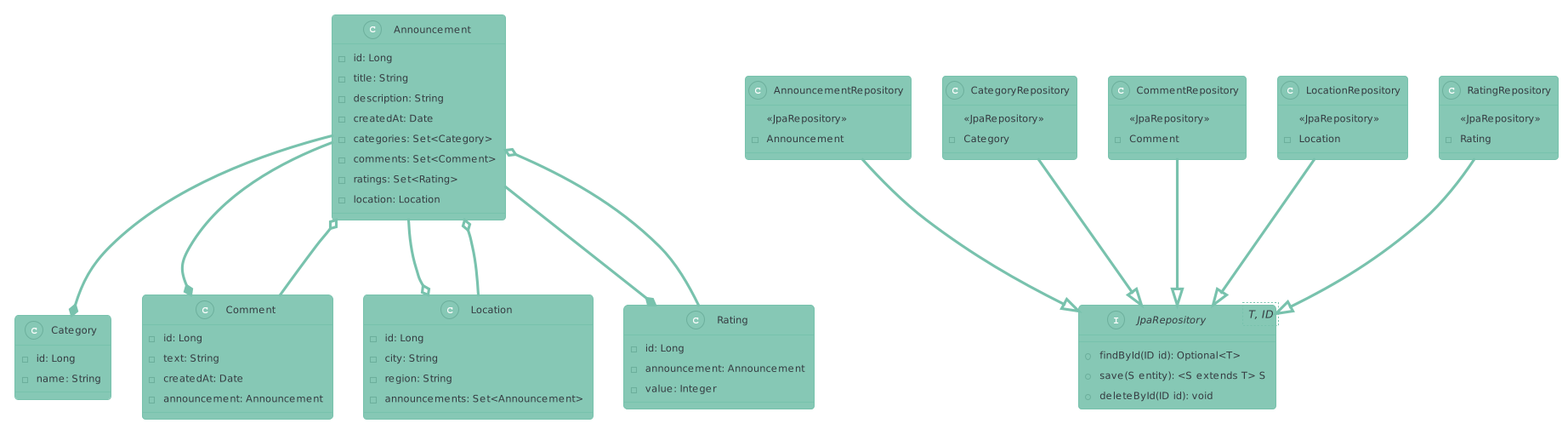


рис. 2.1 – диаграмма классов сущностей

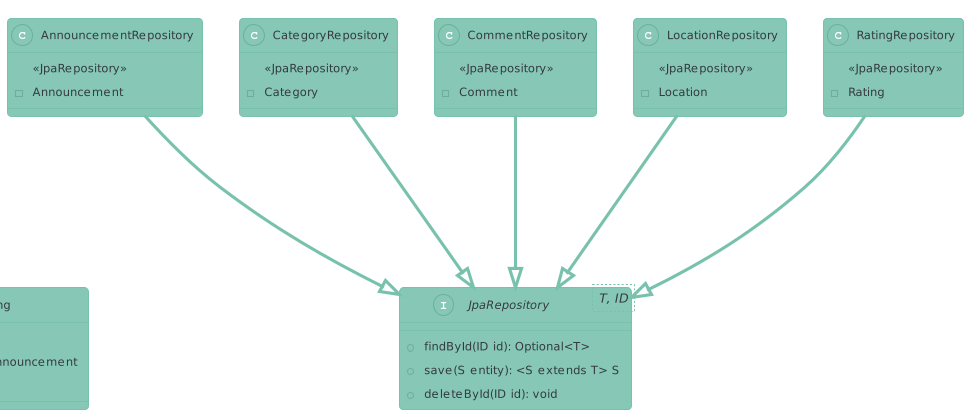


рис. 2.2 – диаграмма классов JPA-репозиториев

Представленная на рис. 2.1, 2.2 диаграмма классов представляет собой визуализацию структуры классов системы OpenBoard и их взаимосвязей.

Ниже приведено описание основных классов системы:

1. Класс `Announcement` (Объявление):
   * `id` (Long): Уникальный идентификатор объявления.
   * `title` (String): Заголовок объявления (максимум 50 символов).
   * `description` (String): Описание объявления (максимум 255 символов).
   * `createdAt` (Date): Дата и время создания объявления.

Отношения:

* Отношение "ManyToOne" с классом `Location`.
* Отношение "OneToMany" с классами `Comment` и `Rating`.
* Отношение "ManyToMany" с классом `Category`.

1. Класс `Category` (Категория):
   * `id` (Long): Уникальный идентификатор категории.
   * `name` (String): Название категории (максимум 50 символов).

Отношения:

* Отношение "ManyToMany" с классом `Announcement`.

1. Класс `Location` (Местоположение):
   * `id` (Long): Уникальный идентификатор местоположения.
   * `city` (String): Город (максимум 100 символов).
   * `region` (String): Регион (максимум 150 символов).

Отношения:

* Отношение "OneToMany" с классом `Announcement`.

1. Класс `Comment` (Комментарий):
   * `id` (Long): Уникальный идентификатор комментария.
   * `text` (String): Текст комментария (максимум 500 символов).
   * `createdAt` (Date): Дата и время создания комментария.

Отношения:

* Отношение "ManyToOne" с классом `Announcement`.

1. Класс `Rating` (Оценка):
   * `id` (Long): Уникальный идентификатор оценки.
   * `value` (Integer): Значение оценки (не может быть пустым).

Отношения:

* Отношение "ManyToOne" с классом `Announcement`.

Данная диаграмма классов отражает основные сущности системы OpenBoard и их взаимосвязи. Классы `Announcement`, `Category`, `Location`, `Comment`, и `Rating` формируют основу для хранения и взаимодействия данных на платформе объявлений.

# Диаграмма компонентов

Диаграмма компонентов представляет из себя средство для визуализации, позволяющее получить представление об организации компонентов системы и взаимосвязей между ними.

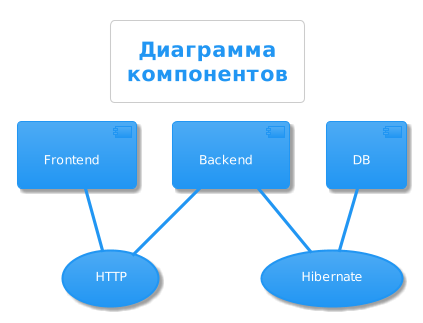


рис. 3– Диаграмма компонентов системы

Представленная на рис. диаграмма компонентов представляет архитектурный обзор системы "OpenBoard". На диаграмме выделены следующие компоненты и их взаимосвязи:

* Frontend: компонент Frontend представляет пользовательский интерфейс системы. Этот компонент взаимодействует напрямую с конечным пользователем, отображая информацию и обрабатывая пользовательский ввод.
* Backend: представляет серверную часть приложения. Он обрабатывает запросы от Frontend, содержит бизнес-логику системы и управляет взаимодействием с базой данных.
* Database (DB): компонент Database отражает хранилище данных системы. В данном случае, это может быть база данных, где хранится информация о объявлениях, категориях, местоположениях и других сущностях.
* Hibernate: Hibernate представляет собой фреймворк для работы с базой данных, который упрощает взаимодействие Backend с Database. Он предоставляет маппинг объектов приложения на таблицы базы данных.
* HTTP: Компонент HTTP обозначает протокол передачи данных, используемый для связи между Frontend, Backend и Hibernate. Frontend отправляет HTTP-запросы Backend, и обратно получает HTTP-ответы с необходимой информацией.

Данная диаграмма демонстрирует взаимосвязь компонентов системы и подчеркивает их роли в обеспечении функциональности "OpenBoard". Frontend и Backend взаимодействуют для обеспечения пользовательского опыта, а Hibernate упрощает доступ к данным в базе данных, обеспечивая структурированное хранение информации. Коммуникация между компонентами осуществляется посредством протокола HTTP, обеспечивая эффективное взаимодействие системы в целом.

# Диаграмма вариантов использования



рис. 4 – Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой графическую модель взаимодействия пользователей с системой OpenBoard. В следующем описании представлены ключевые варианты использования, выделенные на основе данной диаграммы:

просмотр всех данных:

* лицо: Пользователь
* основное действие: пользователь просматривает список всех объявлений на главной странице.
* взаимодействие: пользователь имеет возможность просматривать информацию о каждом объявлении, включая заголовок, описание, дату создания и другие связанные данные.

создание данных:

* лицо: Пользователь
* основное действие: пользователь создает новое объявление, указывая заголовок, описание и другие необходимые данные.
* взаимодействие: система обеспечивает пользователю возможность создания объявлений с различными характеристиками, такими как категория, местоположение и оценка.

удаление данных:

* лицо: пользователь
* основное действие: пользователь удаляет выбранное объявление или другие элементы данных.
* взаимодействие: система подтверждает удаление, после чего выбранный элемент данных удаляется, и пользователь видит обновленный список объявлений.

редактирование данных:

* лицо: пользователь
* основное действие: пользователь вносит изменения в информацию о выбранном объявлении или других элементах данных.
* взаимодействие: система сохраняет обновленные данные, и пользователь видит изменения в списке объявлений.

# Диаграмма последовательности

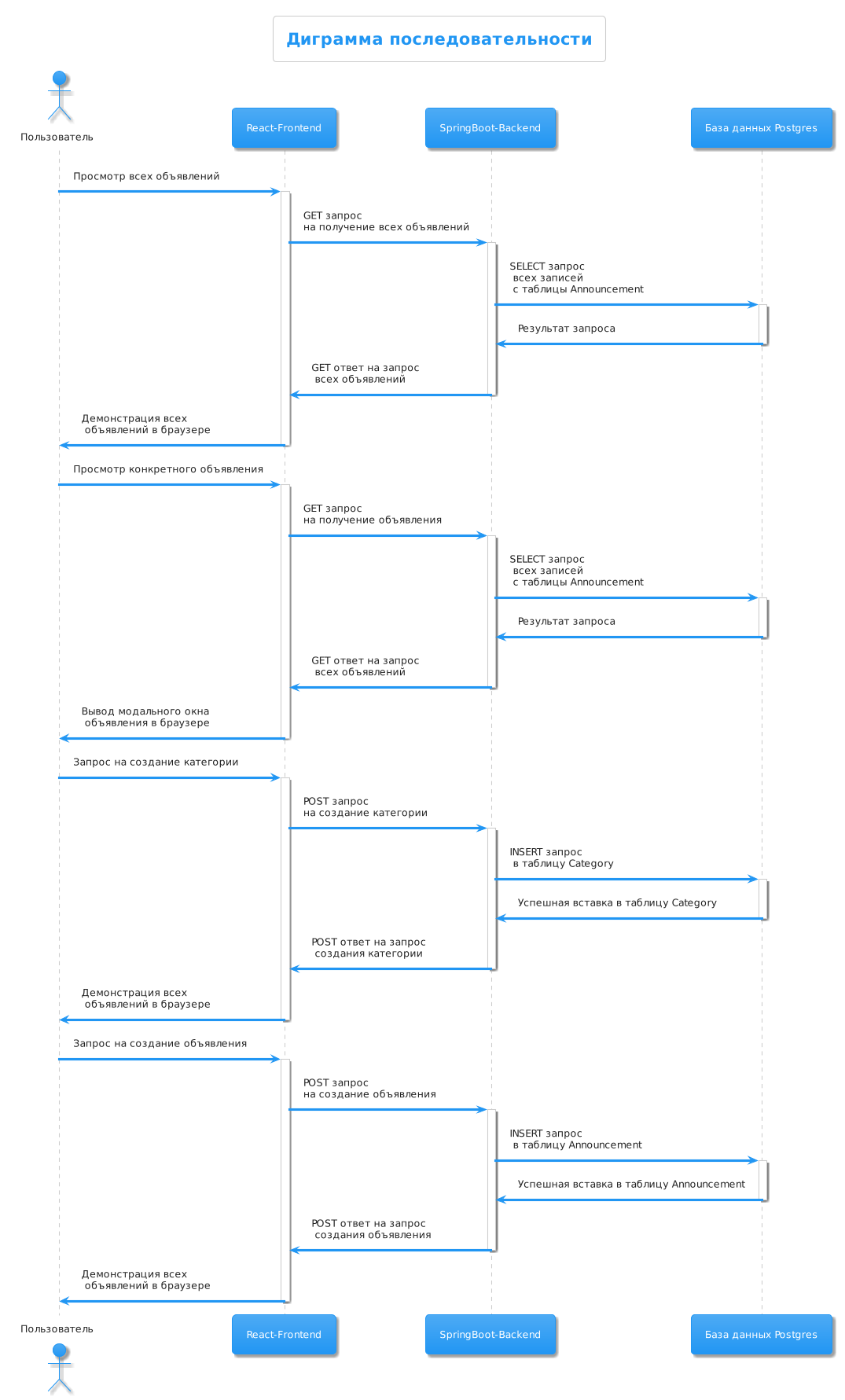


рис. 5 – диаграмма последовательности

Диаграмма последовательности представляет визуальное изображение взаимодействия между компонентами системы OpenBoard при выполнении пользователем определенных действий. Рассмотрим каждый этап взаимодействия:

* просмотр всех объявлений:
  1. пользователь инициирует запрос на просмотр всех объявлений.
  2. frontend отправляет GET-запрос backend для получения всех объявлений.
  3. backend выполняет запрос к базе данных для извлечения всех записей из таблицы announcement.
  4. результат запроса из базы данных возвращается Backend.
  5. backend отправляет GET-ответ с данными всех объявлений frontend.
  6. frontend демонстрирует все объявления в браузере.
* просмотр конкретного объявления: аналогичен шагам предыдущего этапа, за исключением того, что запрос направлен на получение конкретного объявления.
* создание категории:
  1. пользователь запрашивает создание категории.
  2. frontend отправляет POST-запрос Backend для создания категории.
  3. backend выполняет запрос в базу данных для вставки новой записи в таблицу Category.
  4. результат успешной вставки возвращается Backend.
  5. backend отправляет POST-ответ Frontend.
  6. frontend демонстрирует все объявления в браузере.
* создание объявления: аналогичен предыдущему этапу, за исключением того, что запрос направлен на создание объявления.

Данная диаграмма последовательности обеспечивает обзор взаимодействия между компонентами системы OpenBoard при выполнении основных операций.

# Демонстрация работы приложения

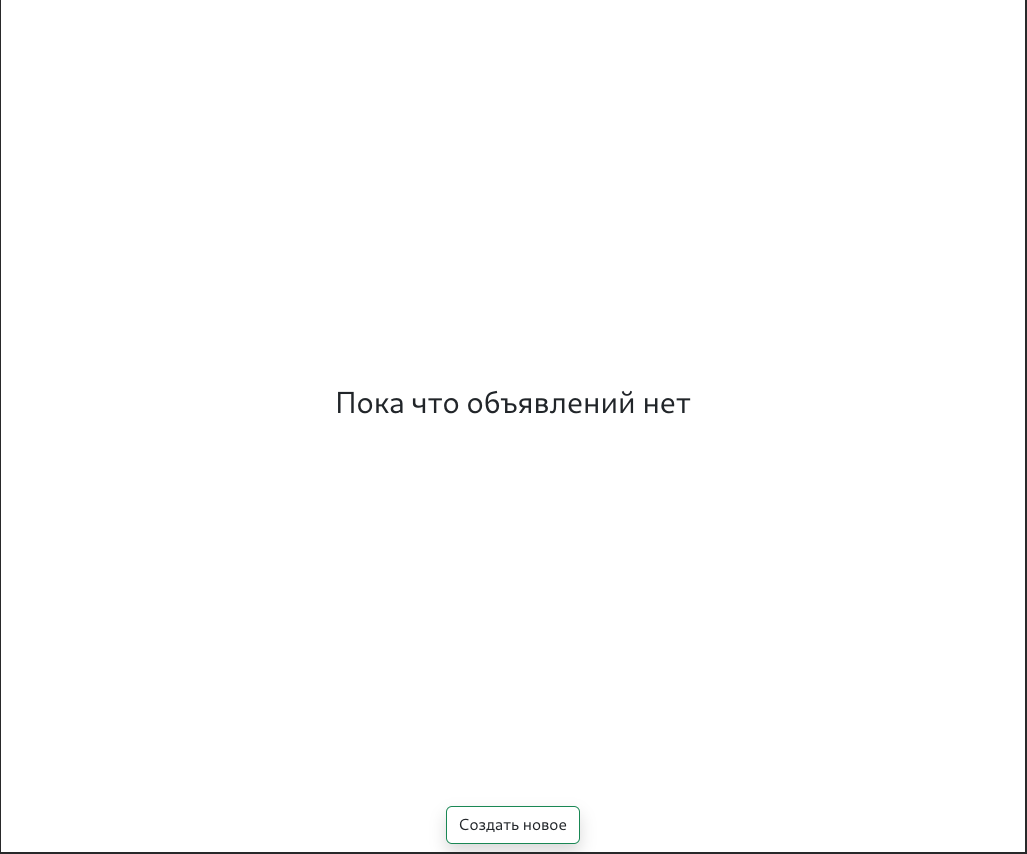


рис. 6.1 – Главная страница приложения

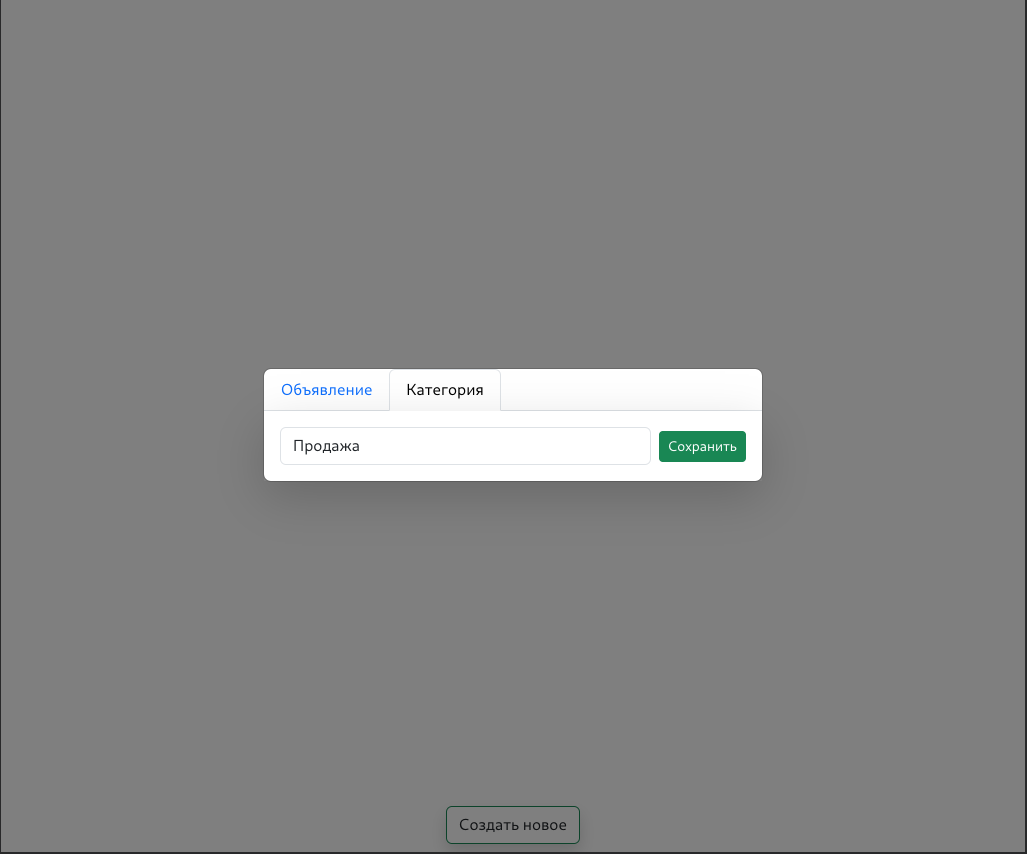


рис. 6.2 – Окно создания категории

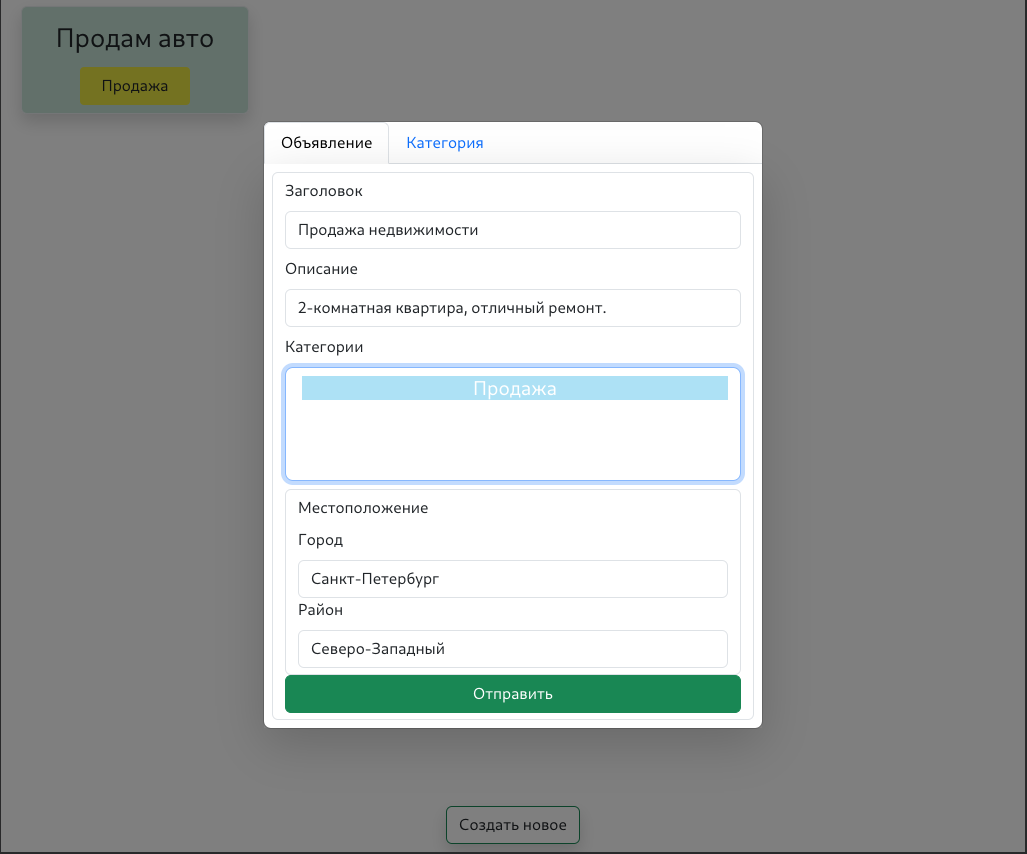


рис. 6.3 – окно создания объявления

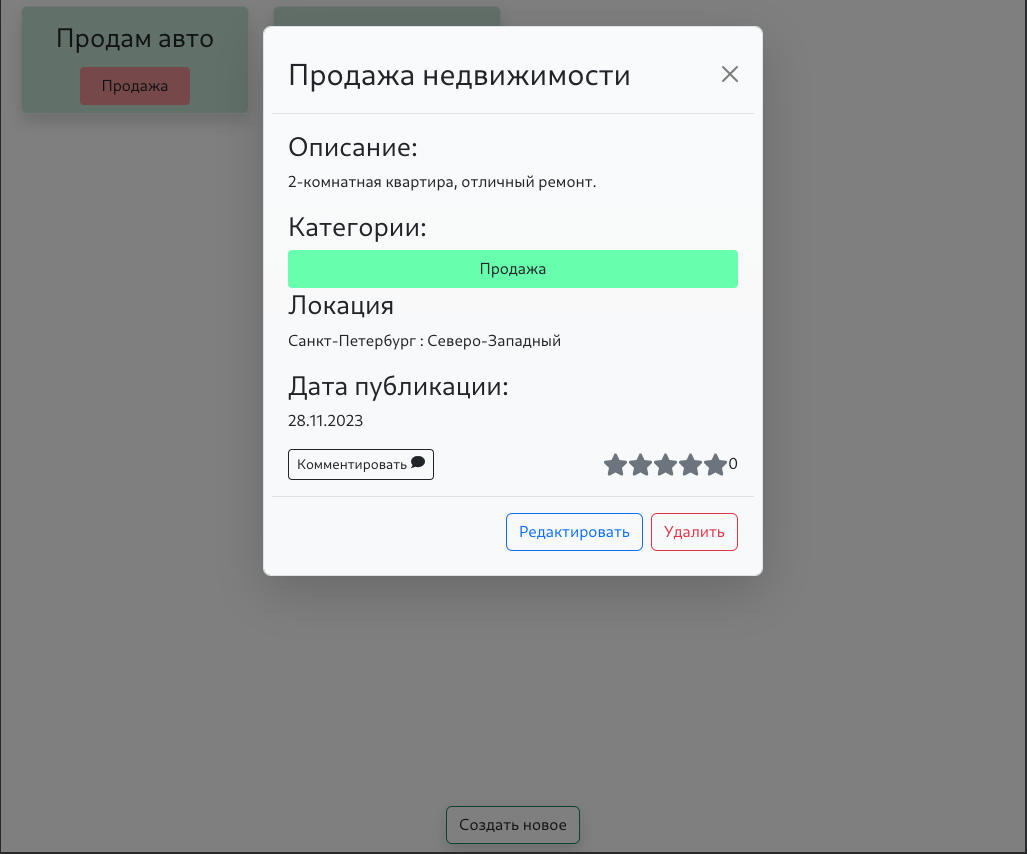


рис. 6.4 – окно просмотра объявления

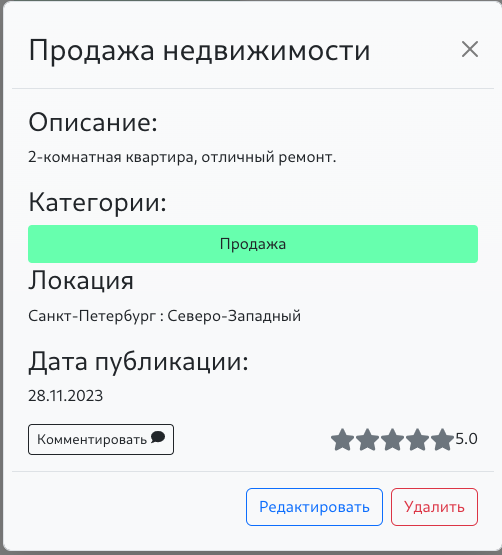


рис. 6.5 – установка оценки

После установки оценки, на рис.6.5 мы можем увидеть среднее значение с учетом всех оценок. В данном случае оценка всего одна и это 5 звезд.

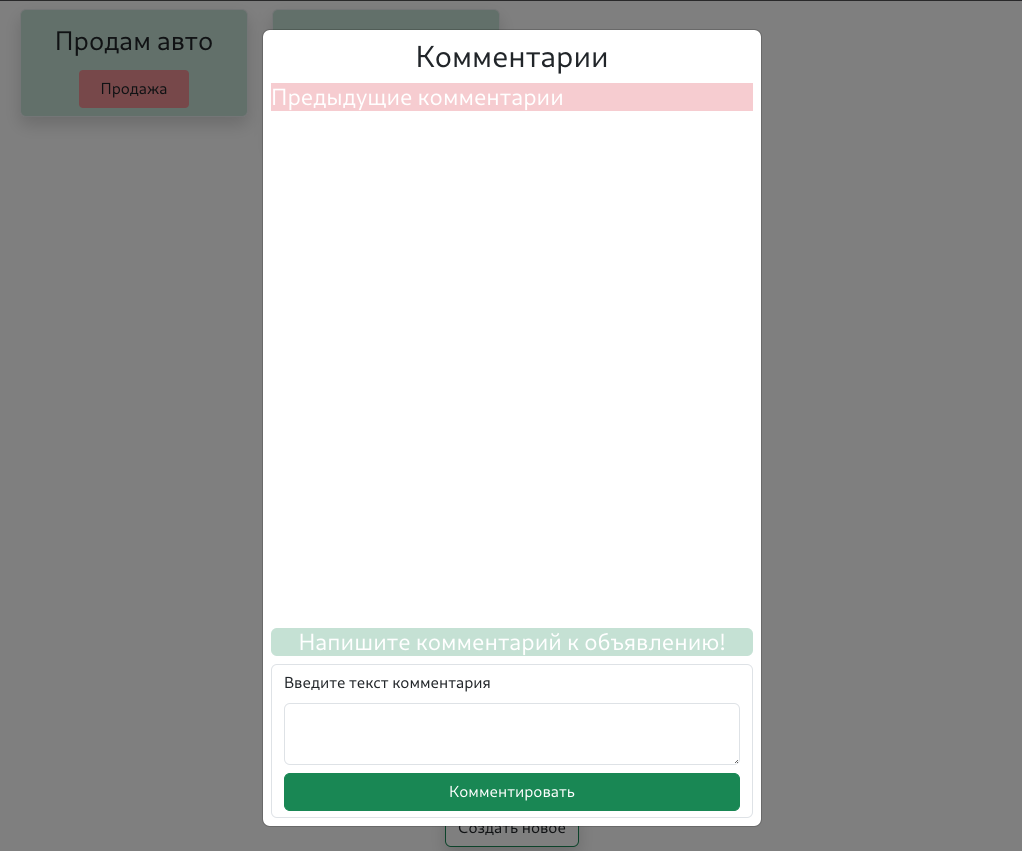


рис. 6.6 – окно комментариев

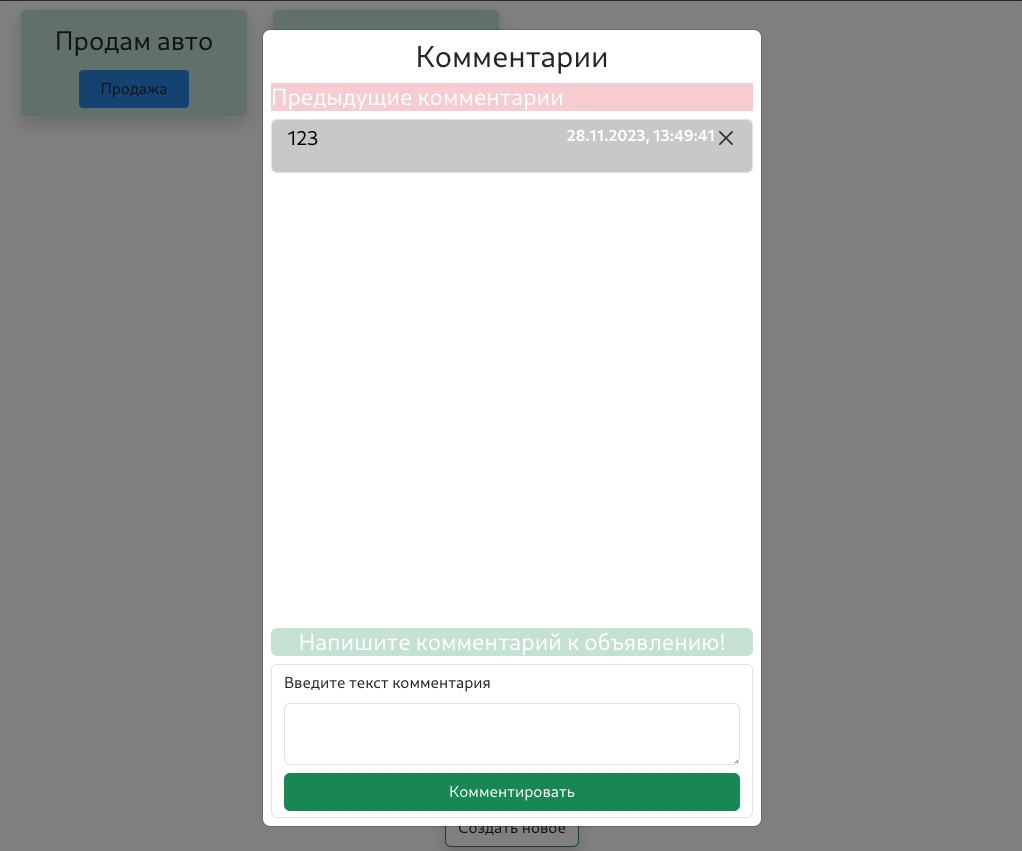


рис. 6.7 – новый комментарий

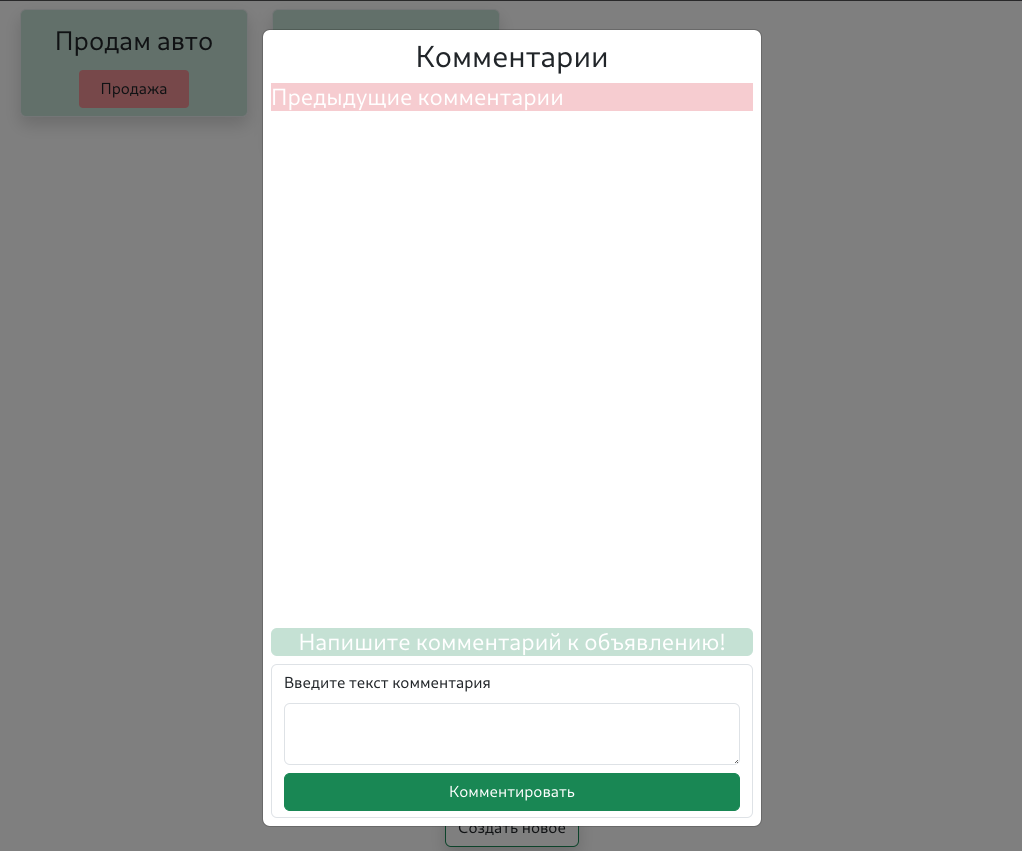


рис. 6.8 – удаление комментария

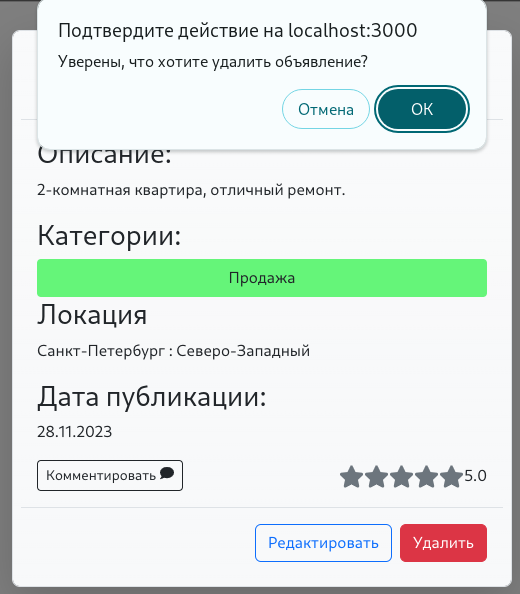


рис. 6.9 – удаление объявления



рис. 6.10 – страница объявлений после удаления

# Список используемых источников

1. Дэвид Геккер, Джон Оспорн - "React: Быстрый старт", Питер, 2021, 320 стр. (ориг. название: "Learning React: Modern Patterns for Developing React Apps", O'Reilly Media)
2. Эрик Фримен, Элизабет Робсон - "JavaScript. Подробное руководство", Вильямс, 2022, 992 стр. (ориг. название: "JavaScript: The Definitive Guide", O'Reilly Media)
3. Крейг Уоллс - "Spring Boot в действии", Питер, 2020, 616 стр. (ориг. название: "Spring Boot in Action", Manning Publications)
4. Крейг Уоллс - "Spring в действии", Питер, 2020, 768 стр. (ориг. название: "Spring in Action", Manning Publications)
5. Герберт Шилдт - "Java: Руководство для начинающих", ДМК Пресс, 2021, 704 стр. (ориг. название: "Java: A Beginner's Guide", McGraw-Hill)
6. Эрик Фримен, Элизабет Робсон - "HTML и CSS. Дизайн и верстка сайтов", Вильямс, 2021, 720 стр. (ориг. название: "Head First HTML and CSS", O'Reilly Media)
7. Джон Дакетт - "HTML5: Руководство разработчика", Питер, 2021, 944 стр. (ориг. название: "HTML5 Developer's Guide", McGraw-Hill)
8. Дэвид Флэнаган - "JavaScript: Подробное руководство", Вильямс, 2022, 1224 стр. (ориг. название: "JavaScript: The Definitive Guide", O'Reilly Media)
9. Крис Шейфер, Джон Уайсс - "Spring Framework. Практика разработки Enterprise-приложений", Питер, 2022, 784 стр. (ориг. название: "Pro Spring 5", Apress)
10. Бен Фрейн - "CSS3 в примерах", Питер, 2020, 288 стр. (ориг. название: "CSS in Depth", Manning Publications)

# Приложение А – Листинг программного кода

package ru.lea.openboardbackend.entity;  
import jakarta.annotation.Nullable;  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
import org.hibernate.annotations.CreationTimestamp;  
import org.springframework.data.annotation.CreatedDate;  
import java.util.Date;  
import java.util.Set;  
@Getter  
@Setter  
@Entity  
@Table(name = "announcement")  
public class Announcement {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name = "id", nullable = false)  
 private Long id;  
 @Column(name = "title", length = 50)  
 private String title;  
 @Column(name = "description")  
 private String description;  
 @CreationTimestamp  
 @CreatedDate  
 private Date createdAt;  
 @ManyToMany(targetEntity = Category.class)  
 private Set<Category> categories;  
 @OneToMany(targetEntity = Comment.class, mappedBy = "announcement", cascade = {CascadeType.REMOVE})  
 private Set<Comment> comments;  
 @OneToMany(targetEntity = Rating.class, mappedBy = "announcement", cascade = {CascadeType.REMOVE})  
 private Set<Rating> ratings;  
 @ManyToOne(targetEntity = Location.class)  
 private Location location;  
}

package ru.lea.openboardbackend.entity;  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
@Getter  
@Setter  
@Entity  
@Table(name = "category")  
public class Category {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name = "id", nullable = false)  
 private Long id;  
 @Column(name = "name", length = 50)  
 private String name;  
}

package ru.lea.openboardbackend.entity;  
import com.fasterxml.jackson.annotation.JsonProperty;  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
import org.hibernate.annotations.CreationTimestamp;  
import java.util.Date;  
@Getter  
@Setter  
@Entity  
@Table(name = "comment")  
public class Comment {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name = "id", nullable = false)  
 private Long id;  
 @Column(name = "text", length = 500)  
 private String text;  
 @JsonProperty(access = JsonProperty.Access.READ\_ONLY)  
 @CreationTimestamp  
 @Temporal(TemporalType.TIMESTAMP)  
 @Column(name = "created\_at")  
 private Date createdAt;  
 @ManyToOne(targetEntity = Announcement.class)  
 @JoinColumn(name = "announcement\_id")  
 private Announcement announcement;  
}

package ru.lea.openboardbackend.entity;  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
import java.util.Set;  
@Getter  
@Setter  
@Entity  
@Table(name = "location")  
public class Location {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name = "id", nullable = false)  
 private Long id;  
 @Column(name = "city", length = 100)  
 private String city;  
 @Column(name = "region", length = 150)  
 private String region;  
 @OneToMany(targetEntity = Announcement.class, mappedBy = "location", orphanRemoval = true)  
 private Set<Announcement> announcements;  
}

package ru.lea.openboardbackend.entity;  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
@Getter  
@Setter  
@Entity  
public class Rating {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name = "id", nullable = false)  
 private Long id;  
 @ManyToOne(targetEntity = Announcement.class)  
 @JoinColumn(name = "announcement\_id")  
 private Announcement announcement;  
 @Column(nullable = false)  
 private Integer value;  
}  
  
  
 const [loadState, setLoadState] = useState({  
 items: [],  
 isLoaded: false  
 })  
  
 useEffect(() => {  
 fetchItems().then(resp => {  
  
 if (resp.hasOwnProperty("error")) {  
 setLoadState(() => ({  
 error: resp.error  
 }));  
 } else {  
 setLoadState(() => ({  
 items: resp,  
 isLoaded: true  
 }))  
 }  
 })  
 }, []);  
 if (loadState.error) {  
 alert(loadState.error.message);  
 } else if (!loadState.isLoaded) {  
 return (  
 <div className="position-absolute  
 d-flex flex-column justify-content-center align-items-center  
 z-1 bg-opacity-25  
 bg-dark w-100 h-100">  
 <Loading/>  
 </div>  
 );  
 }  
  
 return (  
 <div style={{backgroundColor: "#ffffff"}} className="h-100 min-vh-100 d-flex flex-column justify-content-between  
 p-2">  
 <ul className="row row-gap-1 m-0 p-0 h-100 flex-grow-1">  
 {  
 loadState.items.length  
 ? loadState.items.map(i => <AnnouncementItem item={i} key={i.\_links.self.href}  
 onEdit={()=>{  
 fetchItems().then(resp=>{  
 setLoadState(prev=>  
 ({  
 ...prev,  
 items: [...resp]  
 }));  
 // window.location.reload();  
 })  
 }}/>)  
 :  
 <li className="d-flex flex-column align-items-center justify-content-center">  
 <h2>Пока что объявлений нет</h2>  
 </li>  
 }  
 </ul>  
 <button className="btn border border-success shadow align-self-center"  
 data-bs-target="#createForm"  
 data-bs-toggle="modal"  
 >  
 Создать новое  
 </button>  
 <div className="modal fade" id="createForm">  
 <div className="modal-dialog modal-dialog-centered">  
 <div className="modal-content">  
 <ul className="nav nav-tabs">  
 <li className="nav-item">  
 <a className="nav-link active" data-bs-toggle="tab" href="#announcement">Объявление</a>  
 </li>  
 <li className="nav-item">  
 <a className="nav-link" data-bs-toggle="tab" href="#category">Категория</a>  
 </li>  
 </ul>  
 <div className="tab-content shadow-lg">  
 <div className="tab-pane active" id="announcement">  
 <AnnouncementCreator onCreate={  
 ()=> {  
 fetchItems()  
 .then(json => {  
 setLoadState(prevState => ({  
 isLoaded: prevState.isLoaded,  
 items: json  
 }  
 )  
 );  
 }  
 )  
 }  
 }/>  
 </div>  
 <div className="tab-pane" id="category">  
 <CategoryForm onSubmit={data=>{  
 fetch("http://localhost:8080/categories",{  
 method:"POST",  
 headers:{  
 accept: "application/json",  
 "content-type":"application/json"  
 },  
 body:JSON.stringify(data)  
 })  
 .then(resp=>{  
 })  
 .catch(error=>alert(error));  
 }}/>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 );  
}  
  
const fetchItems = ()=>{  
 return fetch("http://localhost:8080/announcements")  
 .then(resp=>{  
 if(!resp.ok){  
 throw new Error(resp.statusText);  
 }  
 return resp.json();  
 })  
 .then(json=>{  
 console.log(json);  
 return json.\_embedded.announcements;  
 })  
 .catch(error=>{  
 return {error};  
 });  
}