

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Web-портал для информационной системы
экологического волонтерского движения**

Студент

(подпись, дата)

Бессонов Никита Олегович

(ФИО полностью)

Руководитель

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Нормоконтролер

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 9.06.2025 г. № 12)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Бессонов Никита Олегович гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Web-портал для информационной системы экологического
волонтерского движения

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: организация экологических мероприятий для волонтеров и добровольцев
Портал предоставляет удобный интерфейс для создания и редактирования
мероприятий, публикации новостей, ведения учета участников и подачи заявок на
участие в волонтерском движении

Основные функции приложения:

- 1) регистрация и авторизация пользователей (волонтеры, организаторы, администраторы);
- 2) создание мероприятий с описанием, местом и временем;
- 3) подача заявок на участие с подтверждением;
- 4) просмотр истории участия волонтером;
- 5) модерация мероприятий (администратор);
- 6) отправка уведомлений об изменениях в событиях.

Технологии разработки: Spring Boot, React

Языки программирования: Java, JavaScript

Среды программирования: IntelliJ IDEA, Visual Studio Code

5. Перечень подлежащих разработке вопросов:

- 1) анализ предметной области и требований к системе;
- 2) проектирование программных средств;
- 3) реализация программных средств;
- 4) тестирование программных средств;
- 5) оформление ПЗ.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан Ф _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 115 листов, 19 рисунков, 5 таблиц, 22 листинга, 20 использованных источников и 2 приложения.

ЭКОЛОГИЧЕСКОЕ ВОЛОНТЕРСТВО, МЕРОПРИЯТИЯ, НОВОСТИ,
МОДУЛЬНЫЙ МОНОЛИТ, MVC, АВТОРИЗАЦИЯ, JWT, REST API

Объект разработки – клиент-серверное приложение для информационной системы экологического волонтерства, предоставляющее функционал по управлению мероприятиями, ведению и просмотру новостного контента, а также подаче заявок.

Цель работы: разработать web-портал для информационной системы, предназначенной для поддержки деятельности экологического волонтерского движения, включая планирование, координацию и информирование участников, а также отслеживание активности и достижений волонтеров.

Результаты разработки: исследована и проанализирована предметная область, выделены основные требования, спроектирована архитектура и графический интерфейс программного обеспечения, проведено планирование разработки и оценка бюджета, программный продукт полностью реализован и отлажен.

Средства разработки:

- система управления базами данных PostgreSQL;
- языки программирования Java, JavaScript (фреймворки Spring Boot и React);
- среды разработки IntelliJ IDEA 2024 CE, Visual Studio Code.

ПГУ 09.03.04 – 08БР211.02 ПЗ

					ПГУ 09.03.04 – 08БР211.02 ПЗ						
Изм.	Лист	№ докум.	Подпись	Дата							
Разраб.		Бессонов Н.О.			Web-портал для информационной системы экологического волонтерского движения			Лит.	Лист	Листов	
Провер.		Такташкин Д.В.								4	115
Реценз.								Группа 21ВП1			
Н. Контр.		Такташкин Д.В.									
Утверд.		Козлов А.Ю.									

Содержание

Введение.....	7
1 Анализ предметной области и требований	10
1.1 Анализ экологического волонтерства	10
1.2 Анализ существующих программных продуктов-аналогов	11
1.3 Постановка задачи на разработку	14
1.4 Анализ требований.....	15
1.4.1 Анализ функциональных требований.....	15
1.4.2 Анализ нефункциональных требований.....	18
2 Проектирование приложения	20
2.1 Выбор технологий разработки	20
2.2 Проектирование архитектуры приложения	22
2.3 Проектирование графического интерфейса	27
2.4 Планирование разработки и оценка бюджета	31
3 Реализация web-портала для информационной системы экологического волонтерского движения	37
3.1 Реализация базы данных.....	37
3.2 Реализация архитектуры программного обеспечения.....	38
3.2.1 Реализация механизма аутентификации и авторизации на основе JWT	38
3.2.2 Реализация модуля мероприятий	42
3.3 Реализация пользовательского интерфейса.....	48
3.3.1 Реализация взаимодействия с сервером через REST API.....	48
3.3.2 Реализация личного кабинета волонтера	51
4 Контроль качества программного обеспечения	55
4.1 Функциональное тестирование на основе классов эквивалентности.....	55
4.2 Анализ сложности кода модуля новостей	56
4.3 Автоматизированное тестирование модуля мероприятий.....	59
Заключение	62

Список использованных источников	63
Приложение А (обязательное) Исходный код программного обеспечения.....	65
Приложение Б (рекомендуемое) Исходный код создания таблиц базы данных	112

Введение

В данный момент вопросы охраны окружающей среды приобретают все большую значимость. Экологические вызовы, такие как изменение климата, загрязнение окружающей среды, потеря биоразнообразия, требуют не только глобальных решений, но и активного участия граждан. Волонтерские инициативы играют ключевую роль в решении экологических проблем на локальном уровне, объединяя неравнодушных людей, стремящихся внести вклад в сохранение природы и улучшение качества жизни.

Организация экологических мероприятий, таких как субботники, посадка деревьев, просветительские акции, требует тщательного планирования, координации участников и эффективного обмена информацией. В условиях высокой загруженности как у организаторов, так и у волонтеров, эти задачи становятся особенно трудоемкими. Информационные системы приходят на помощь, предлагая централизованные и автоматизированные решения для управления мероприятиями, взаимодействия между участниками и формирования устойчивого сообщества.

Автоматизация процессов в экологическом движении способствует не только снижению организационной нагрузки, но и повышению вовлеченности граждан, прозрачности мероприятий и своевременной обратной связи. Информационная система позволяет учитывать интересы волонтеров, отслеживать их участие и достижения, а также оперативно информировать о новостях и изменениях в событиях. Это делает участие более осознанным, а работу организаторов – более эффективной.

Актуальность разработки подобной системы определяется возрастающей потребностью в современных цифровых инструментах поддержки экологического волонтерства. Такие инструменты могут быть востребованы как общественными организациями, так и инициативными группами, желающими наладить устойчивое взаимодействие с

единомышленниками. В условиях цифровизации социальной сферы, автоматизация становится неотъемлемым фактором успешной реализации экологических инициатив.

Информационная система также способствует развитию экологической культуры, формированию чувства ответственности за окружающую среду и стимулирует социальную активность. Это особенно важно в условиях, когда устойчивое развитие должно быть не только задачей государства, но и результатом совместных усилий общества.

Тем не менее, важно понимать, что цифровые технологии служат лишь инструментом. Главным остается человек – его мотивация, осознанность и стремление к позитивным изменениям. Информационная система лишь облегчает путь к этим изменениям, предоставляя удобную и эффективную среду для их реализации.

Целью данной работы является разработка информационной системы, предназначенной для поддержки деятельности экологического волонтерского движения, включая планирование, координацию и информирование участников, а также отслеживание активности и достижений волонтеров.

Для достижения поставленной цели требуется решить следующие задачи:

- выполнить анализ предметной области;
- провести анализ существующих программных продуктов-аналогов;
- выделить функциональные требования к разрабатываемой системе;
- разработать необходимые диаграммы в нотации UML;
- спроектировать архитектуру приложения;
- спроектировать пользовательский интерфейс;
- провести планирование и оценку бюджета разработки системы;
- разработать спроектированную систему;

- протестировать разработанную систему и выполнить оценку качества кода.

В результате выполнения указанных задач необходимо провести анализ проделанной работы и сделать выводы о достижении поставленной цели.

1 Анализ предметной области и требований

1.1 Анализ экологического волонтерства

Экологическое волонтерство – это социально-ориентированная деятельность, направленная на улучшение состояния окружающей среды и формирование экологической культуры. В рамках таких инициатив проводятся мероприятия по уборке территорий, сбору вторсырья, посадке деревьев, экопросвещению и другим формам участия.

Если говорить о цифровой среде экологического волонтерства и о web-портале в частности, то здесь можно выделить несколько ключевых категорий участников: гости портала, волонтеры, организаторы и администраторы.

Гости портала – незарегистрированные пользователи, имеющие возможность просматривать общедоступную информацию: новости, мероприятия и призыв к участию.

Волонтеры – физические лица, желающие участвовать в мероприятиях, отслеживать свою активность и оставлять отзывы. Они подают заявки на участие и получают уведомления о событиях и изменениях.

Организаторы – представители организаций или инициативных групп, создающие и управляющие мероприятиями, публикующие новости и взаимодействующие с волонтерами.

Администраторы системы – модераторы, контролирующие корректность публикуемой информации, подтверждающие или отклоняющие заявки на регистрацию организаторов. Также они управляют статусом пользователей (например, блокировка за нарушение правил).

Основными объектами предметной области экологического волонтерства являются:

- 1) мероприятия – структурированные события с описанием, временем, местом проведения и списком участников;
- 2) заявки на участие – регистрационные формы волонтеров, желающих принять участие в конкретном мероприятии;

3) новости – информационные сообщения от организаторов о текущей деятельности, изменениях, инициативах, отчеты о прошедших мероприятиях;

4) достижения – система мотивации и геймификации, позволяющая волонтерам получать награды за активность;

5) уведомления – автоматические оповещения о подтверждениях, изменениях, отменах и других событиях.

Вся информация в системе должна быть актуальной, проверенной и доступной пользователям с учетом их роли. В основе работы системы – прозрачное взаимодействие между организаторами и волонтерами с обеспечением административного контроля и удобства использования.

1.2 Анализ существующих программных продуктов-аналогов

В процессе анализа существующих программных продуктов-аналогов разрабатываемого веб-портала было выявлено несколько альтернатив. Ниже представлен обзор недостатков найденных в поисковой системе аналогичных платформ.

Платформа «Добро.РФ». Интерфейс главной страницы представлен на рисунке 1.

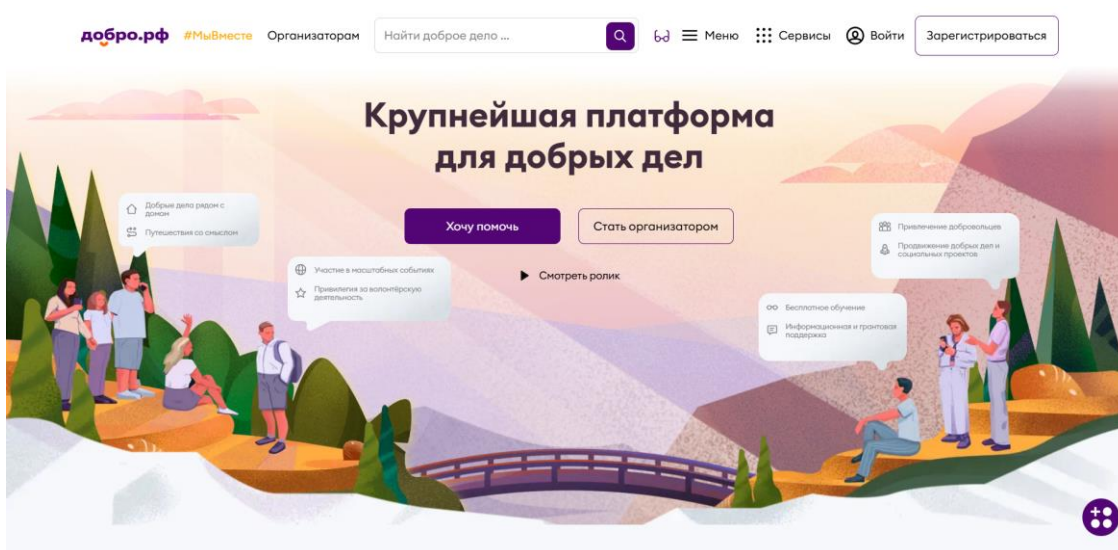


Рисунок 1 – Главная страница платформы «Добро РФ»

Данная платформа имеет следующие недостатки:

- экологические мероприятия представлены лишь как одна из многочисленных категорий, без акцента на экологическую тематику;
- сложный и перегруженный пользовательский интерфейс, особенно для новых участников;
- ограниченные возможности аналитики для организаторов и волонтеров;
- централизованная структура управления (слабая поддержка локальных инициатив) [1].

Платформа «Движение ЭКА». Интерфейс главной страницы представлен на рисунке 2.

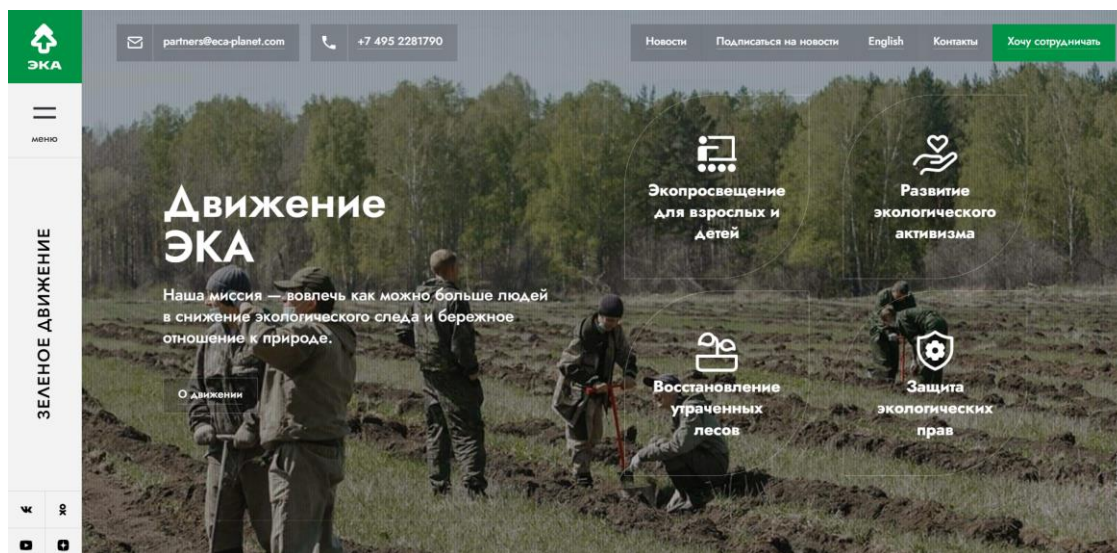


Рисунок 2 – Главная страница платформы «Движение ЭКА»

Данная платформа имеет следующие недостатки:

- устаревший дизайн и ограниченный функционал для интерактивного взаимодействия;
- отсутствует полноценная система регистрации волонтеров и учета их активности;
- мероприятия публикуются в основном централизованно, что ограничивает вовлеченность местных организаторов;

– недостаточная интеграция системы отзывов и оценки эффективности мероприятий [2].

Платформа «Мосволонтер». Интерфейс главной страницы представлен на рисунке 3.

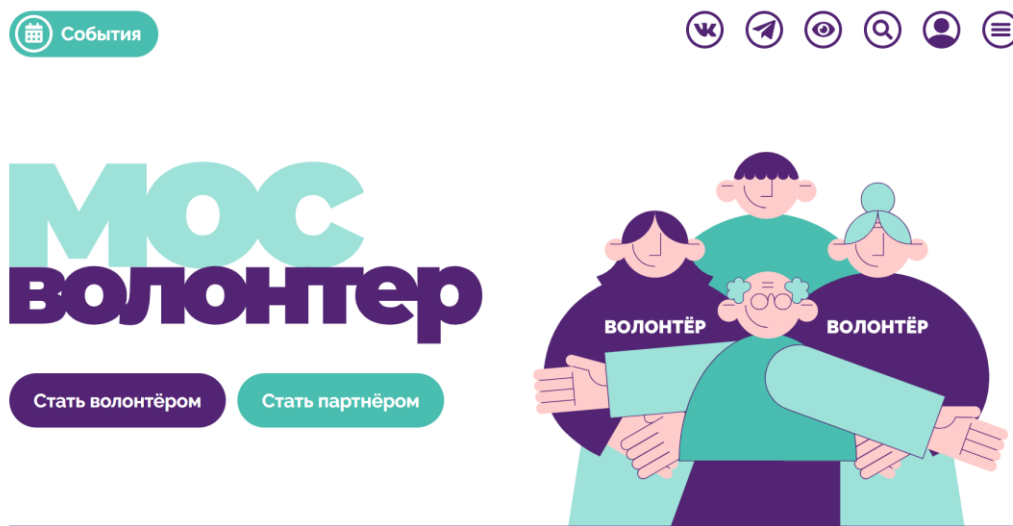


Рисунок 3 – Главная страница платформы «Мосволонтер»

Данная платформа имеет следующие недостатки:

- ограниченная география охвата, фокус только на московский регион;
- слабая поддержка экологических инициатив как отдельного направления;
- закрытая модель взаимодействия между организациями и волонтерами, отсутствие гибкой системы фильтрации мероприятий по интересам [3].

Разрабатываемая платформа отличается от аналогов следующими преимуществами:

- полная специализация на экологическом волонтерстве;
- интуитивно понятный и легкий интерфейс для всех категорий пользователей;
- возможность локального управления мероприятиями со стороны общественных организаций;

- система отзывов для повышения эффективности мероприятий;
- мотивация волонтеров через геймификацию: начисление достижений за участие.

Таким образом, разрабатываемое приложение учитывает все недостатки существующих на данный момент аналогов и предлагает их решение. Оно предоставит пользователям систему, нацеленную именно на экологическую тематику с более понятным и простым интерфейсом. Также у пользователей (в частности, волонтеров) появится система поощрения в виде получения достижений, мотивирующая их участвовать в мероприятиях.

1.3 Постановка задачи на разработку

В результате анализа предметной области была сформулирована и поставлена задача на разработку программного продукта. В данной работе предстоит реализовать web-портал для информационной системы экологического волонтерского движения. Разрабатываемый продукт должен предоставлять функционал по созданию мероприятий и управлению ими, а также возможность принять в них участие. Помимо раздела мероприятий, необходимо реализовать новостной раздел, где пользователи смогут отслеживать активность компаний и организаций, комментировать новостные статьи и оставлять отзывы к отчетам о мероприятиях, а также иметь доступ к актуальной информации по соответствующей тематике. На портале будут взаимодействовать две основные категории пользователей: организаторы экологических мероприятий и волонтеры, участвующие в них. Необходимо организовать процесс аутентификации и авторизации для четкого распределения ролей пользователей в системе и доступа к соответствующим ресурсам и функциям.

Разрабатываемое приложение должно предоставлять пользователям простой интуитивно понятный интерфейс, позволяющий легко и эффективно использовать приложение.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

В результате анализа предметной области и поставленной задачи были выделены основные категории пользователей портала и соответствующие для каждой категории функциональные требования.

Основные функциональные требования по категориям пользователей:

- 1) функциональные требования для категории «Гость»:
 - просмотр новостей и информации о предстоящих мероприятиях;
 - возможность регистрации в качестве волонтера;
- 2) функциональные требования для категории «Волонтер»:
 - просмотр предстоящих мероприятий и истории поданных заявок на участие;
 - подача заявок на участие в мероприятиях;
 - просмотр и комментирование новостей на портале;
 - отслеживание личного прогресса (достижения);
 - возможность подать заявку на регистрацию в качестве организатора;
- 3) функциональные требования для категории «Организатор»:
 - создание и редактирование собственных мероприятий;
 - просмотр предстоящих мероприятий;
 - управление списком участников на предстоящие мероприятия;
 - написание и публикация новостных статей и отчетов о мероприятиях;
- 4) функциональные требования для категории «Администратор»:
 - модерация контента новостей;
 - управление пользователями и организациями.

На основе выдвинутых к системе функциональных требований была создана диаграмма вариантов использования, изображенная на рисунке 4,

наглядно представляющая возможности портала для каждой категории пользователей [4].

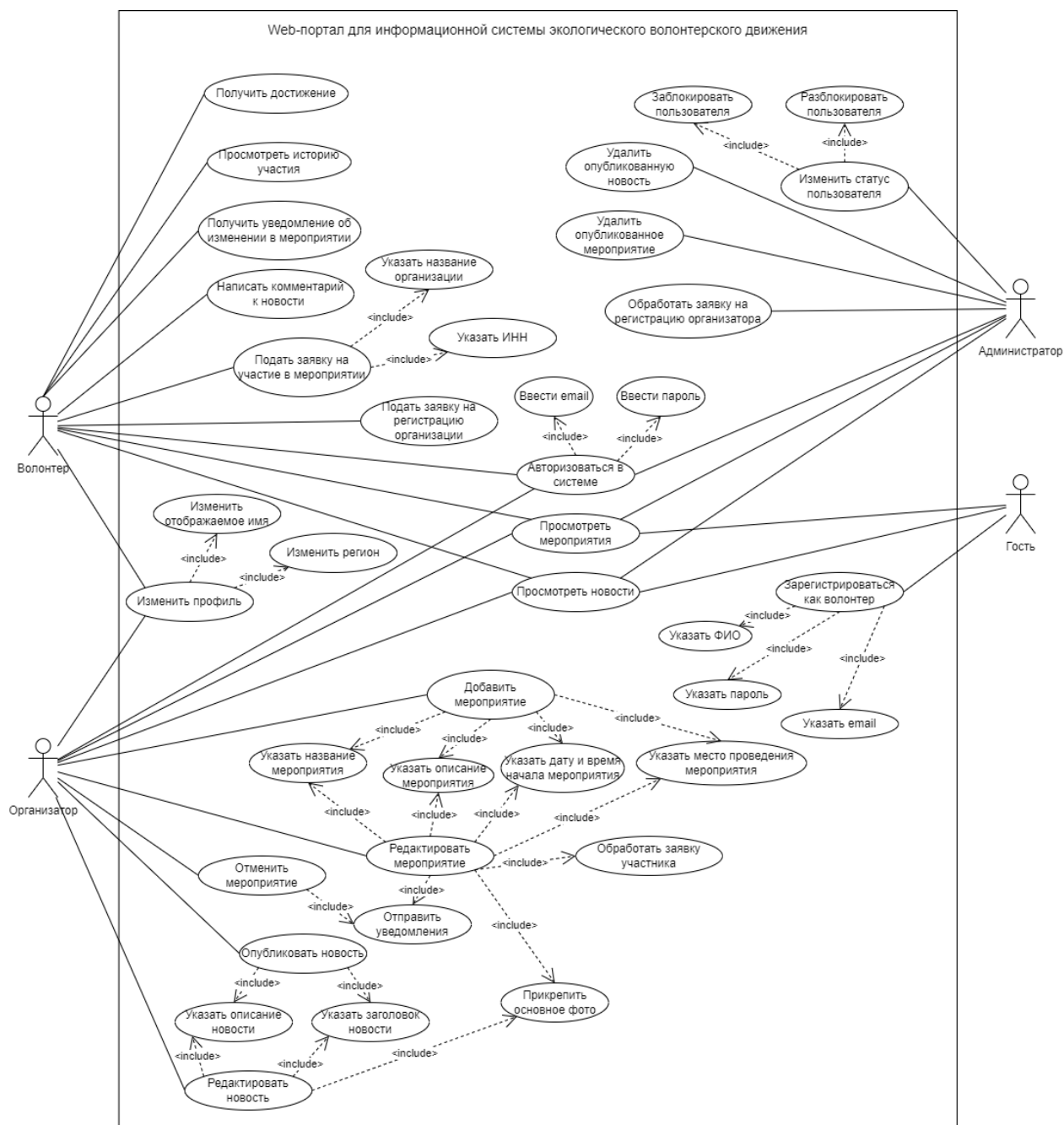


Рисунок 4 – Диаграмма вариантов использования

Ниже представлены спецификации основных вариантов использования для актера «Организатор».

Спецификация варианта использования «Авторизоваться в системе» представлена на рисунке 5.

Вариант использования	Авторизоваться в системе
ID	1
Краткое описание	Пользователь вводит учетные данные (логин, пароль) и авторизуется на портале
Действующие лица	Организатор
Предусловие	Пользователь зарегистрирован
Основной поток	<ol style="list-style-type: none"> 1. Пользователь переходит на страницу входа на портал 2. Пользователь вводит данные для входа в существующий аккаунт (email и пароль) 3. Пользователь получает доступ к соответствующему функционалу портала
Постусловие	Пользователь авторизовался в системе, ему доступен функционал организатора

Рисунок 5 – Спецификация варианта использования «Авторизоваться в системе»

Спецификация варианта использования «Добавить мероприятие» представлена на рисунке 6.

Вариант использования	Добавить мероприятие
ID	2
Краткое описание	Пользователь добавляет новое мероприятие в систему с соответствующими данными
Действующие лица	Организатор
Предусловие	Пользователь авторизован
Основной поток	<ol style="list-style-type: none"> 1. Пользователь переходит на страницу добавления мероприятия 2. Пользователь вводит название мероприятия 3. Пользователь вводит описание мероприятия 4. Пользователь вводит дату и время начала мероприятия 5. Пользователь вводит место проведения мероприятия 6. Пользователь публикует мероприятие
Постусловие	Мероприятие добавлено в список предстоящих мероприятий

Рисунок 6 – Спецификация варианта использования «Добавить мероприятие»

Спецификация варианта использования «Редактировать новость» представлена на рисунке 7.

Вариант использования	Редактировать новость
ID	3
Краткое описание	Пользователь редактирует новость
Действующие лица	Организатор
Предусловие	Пользователь авторизован, новость создана и не опубликована
Основной поток	<ol style="list-style-type: none"> 1. Пользователь переходит на страницу управления новостями 2. Пользователь выбирает «Редактировать» у соответствующей новости 3. Пользователь вводит заголовок 4. Пользователь вводит текст новости 5. Пользователь прикрепляет фотографию 6. Пользователь сохраняет новость
Постусловие	Новость сохранена, обновленная информация отображается на портале

Рисунок 7 – Спецификация варианта использования «Редактировать новость»

В результате анализа функциональных требований была создана диаграмма вариантов использования разрабатываемого продукта, а также были описаны спецификации основных вариантов использования для актера «Организатор».

1.4.2 Анализ нефункциональных требований

При анализе требований важно учитывать не только функциональные, но также и нефункциональные требования. При этом нефункциональные требования напрямую влияют на пользовательский опыт, стабильность и безопасность системы. Они определяют, насколько удобно и надежно будет работать программный продукт в реальных условиях использования. Игнорирование этих аспектов может привести к снижению эффективности использования системы и неудовлетворенности пользователей [5].

Нефункциональные требования к web-порталу для информационной системы экологического волонтерского движения:

- удобный и понятный пользовательский интерфейс, обеспечивающий интуитивную навигацию по разделам портала;

- поддержка корректной работы во всех современных версиях популярных браузеров, таких как Opera, Google Chrome, Mozilla Firefox и Safari;
- надежность и устойчивость к длительной нагрузке – система должна сохранять работоспособность при продолжительном использовании;
- стабильное функционирование при параллельной работе нескольких пользователей;
- защита данных и обеспечение безопасного доступа к функциям портала в зависимости от прав пользователя (просмотр, редактирование и пр.);
- возможность масштабирования, позволяющая расширять систему при увеличении числа пользователей и объема обрабатываемой информации.

В результате анализа требований к создаваемому web-порталу были сформулированы как функциональные, так и нефункциональные требования, которые лягут в основу проектирования и реализации информационной системы и web-портала в целом.

2 Проектирование приложения

2.1 Выбор технологий разработки

При разработке информационной системы для экологического волонтерского движения особое внимание было уделено выбору технологий. Необходимо выбрать такие технологии, которые обеспечивают стабильную работу, масштабируемость и простоту дальнейшего сопровождения программного продукта. На основе анализа требований были выбраны инструменты, охватывающие все ключевые аспекты системы – от хранения и обработки данных до построения удобного пользовательского интерфейса. Основной акцент был сделан на проверенные, активно поддерживаемые и широко распространенные технологии с открытым исходным кодом.

Первостепенное значение носит выбор способов хранения данных. Выбор хранилища данных напрямую влияет на масштабируемость, гибкость и производительность системы. Так как для разработки информационной системы веб-портала необходимо поддерживать ограничения целостности данных, ссылочные ограничения, а также валидацию данных, было решено использовать реляционную модель данных.

В качестве СУБД будет использоваться PostgreSQL – современная система управления базами данных, обеспечивающая надежное хранение и обработку структурированных данных.

PostgreSQL относится к классу объектно-реляционных СУБД и отличается выраженной гибкостью и надежностью. Он обеспечивает поддержку сложных структур данных – включая пользовательские типы, массивы, JSON-структуры, геометрические типы и диапазоны – что делает его универсальным инструментом для самых разных сценариев, от простого хранения до аналитики. Кроме того, PostgreSQL реализует устойчивую систему репликации (и асинхронную, и синхронную), а также ведет журнал транзакций (WAL), что обеспечивает высокую отказоустойчивость и безопасность при аварийных ситуациях. Наконец, благодаря адаптации к

многоядерным процессорам и возможностям горизонтального масштабирования (например, через Postgres-XL и другие расширения), PostgreSQL является отличным выбором для проектов с растущими нагрузками и объемами данных [6].

Предварительно, архитектура разрабатываемого продукта является клиент-серверной, что означает наличие, как минимум, двух единиц развертывания: «сервер» и «клиент».

Архитектура «клиент-сервер» представляет собой модель, в которой несколько клиентов отправляют запросы на централизованный сервер, а тот обрабатывает их и возвращает ответы. Сервер способен обслуживать множество подключений одновременно, распределяя ресурсы посредством процессов или потоков, что обеспечивает масштабируемость и устойчивость системы при параллельной нагрузке. Такая модель позволяет централизованно управлять данными, безопасностью и ресурсами, одновременно делегируя клиентам задачи пользовательского интерфейса и взаимодействия с пользователем [7].

Для организации обработки данных на серверной части был выбран фреймворк Spring Boot. Spring Boot обеспечивает быстрое и удобное построение серверной части благодаря встроенной поддержке шаблонов проектирования, модульности и хорошей документации, возможности масштабирования и интеграции с другими сервисами. Он также имеет встроенные модули: ORM Hibernate для более надежной работы с базой данных и Spring Security, который позволяет настроить механизмы аутентификации и авторизации пользователей, а также фильтры и параметры доступа к различным ресурсам [8].

Для клиентской части был выбран React – современная JavaScript-библиотека, предназначенная для создания динамических и интерактивных пользовательских интерфейсов. React обеспечивает высокую скорость работы, поддержку одностраничных приложений (SPA) и удобную компонентную

архитектуру, что делает его оптимальным решением для разработки сложных веб-приложений.

React использует декларативный подход к описанию интерфейсов, позволяя разработчикам сосредоточиться на логике приложения, а не на низкоуровневых операциях с DOM (Document Object Model). Благодаря синтаксису JSX, код становится более читаемым и удобным для работы. Кроме того, библиотека поддерживает виртуальный DOM, что значительно ускоряет рендеринг и повышает производительность. Активное сообщество и большое количество готовых компонентов делают React гибким инструментом для создания масштабируемых веб-приложений [9].

Таким образом, выбранные технологии позволят реализовать надежную, гибкую и масштабируемую информационную систему для экологического волонтерского движения.

2.2 Проектирование архитектуры приложения

Архитектура приложения представляет собой совокупность решений, определяющих структуру системы, способы взаимодействия ее компонентов и принципы их организации. Она играет ключевую роль в обеспечении надежности, масштабируемости и удобства сопровождения программного продукта. Грамотно спроектированная архитектура позволяет разделить систему на логически обоснованные части, определить потоки данных и взаимодействие между уровнями, а также заложить основу для будущего расширения и модификации функциональности. В современных веб-приложениях архитектура часто строится на основе клиент-серверной модели и ориентируется на принципы модульности, повторного использования и устойчивости к изменениям.

Разрабатываемое приложение следует трехуровневой архитектуре, предполагающей разделение на клиентскую часть, серверную часть и слой хранения данных. Такая структура способствует изоляции ответственности, снижает связанность компонентов и упрощает поддержку проекта. Связь

между клиентом и сервером осуществляется через REST API, построенный на протоколе HTTP и использующий формат JSON для обмена данными. Такой подход обеспечивает совместимость, масштабируемость и независимость клиентской и серверной части.

Таким образом, имеются три единицы развертывания: компьютер клиента, сервер приложения и сервер базы данных. Для лучшего понимания на рисунке 8 приведена диаграмма развертывания.

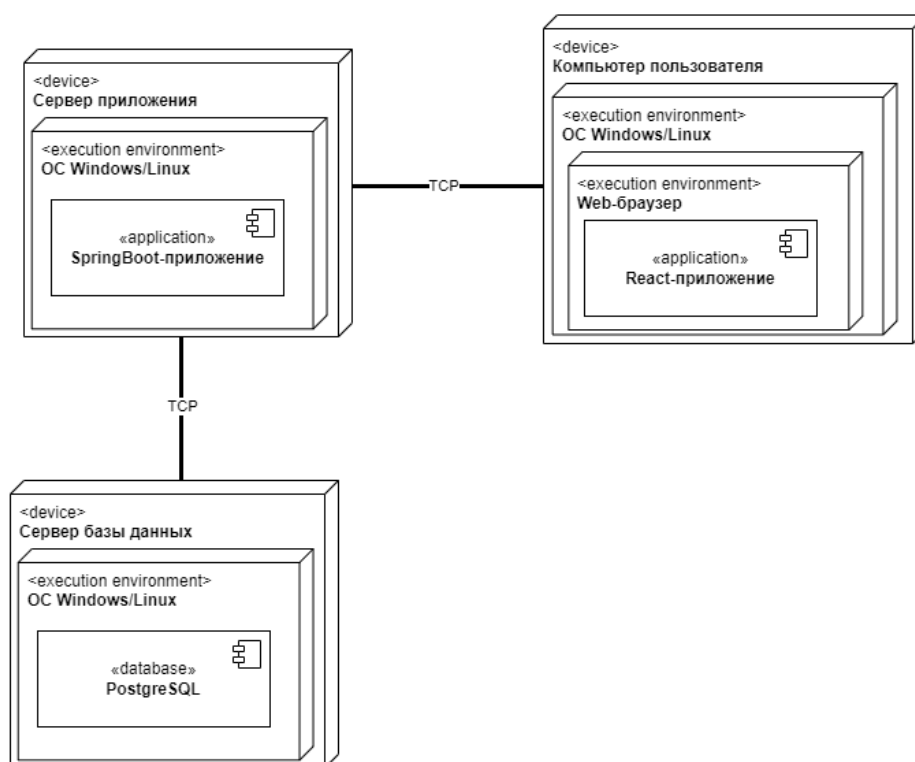


Рисунок 8 – Диаграмма развертывания

Диаграмма развертывания отражает физическое распределение компонентов системы в процессе выполнения. В ней представлены три основных узла: сервер базы данных, сервер приложения и клиентское устройство пользователя. Каждый узел содержит соответствующие исполняющие среды и размещенные на них программные компоненты.

Сервер базы данных представляет собой отдельное устройство, на котором развернута реляционная система управления базами данных PostgreSQL. Этот узел обеспечивает надежное хранение и обработку

информации, необходимой для функционирования веб-приложения. В качестве операционной системы на данном сервере может использоваться Windows или Linux.

Сервер приложения размещает основную бизнес-логику системы. На нем функционирует Spring Boot-приложение, предоставляющее REST API для обработки клиентских запросов и взаимодействия с базой данных. Приложение исполняется в окружении операционной системы Windows или Linux и взаимодействует с базой данных посредством протокола TCP.

Клиентская часть системы представлена пользовательским компьютером, на котором осуществляется взаимодействие с системой через современный веб-браузер. В браузере выполняется React-приложение, обеспечивающее динамический пользовательский интерфейс и отправку HTTP-запросов к серверной части.

Взаимодействие между компонентами осуществляется по протоколу TCP. Клиентское приложение отправляет запросы к серверу приложения через протокол HTTP, а сервер приложения, в свою очередь, обращается к базе данных через соединение по JDBC.

Таким образом, представленная архитектура обеспечивает четкое разделение ответственности, позволяет обеспечить масштабируемость, простоту развертывания и поддержку, а также отвечает современным требованиям к клиент-серверным веб-приложениям.

С точки зрения организации серверного кода, выбрана архитектура модульного монолита, сочетающая преимущества монолитных приложений с высокой степенью внутренней модульности. В рамках одного исполняемого приложения логика разделяется на изолированные модули (например: управление пользователями, работа с мероприятиями, заявками, уведомлениями и достижениями). Это позволяет изолировать бизнес-логику по доменам, уменьшить связанность компонентов, упростить тестирование, сопровождение и масштабирование, в перспективе – выделить отдельные модули в сервисы без полной переработки архитектуры.

Модульный монолит представляет собой архитектурный стиль, который объединяет преимущества монолитных и микросервисных систем. Он позволяет структурировать приложение на независимые модули, обеспечивая четкое разделение ответственности и упрощая поддержку. Такой подход снижает сложность интеграции, устраняет проблемы распределенных систем, но при этом сохраняет гибкость и масштабируемость. Благодаря модульности, разработчики могут внедрять новые функции без риска нарушить работу всей системы, а также легче управлять зависимостями и тестированием [10].

Наглядное представление архитектуры серверной части представлено на диаграмме компонентов (рисунок 9).

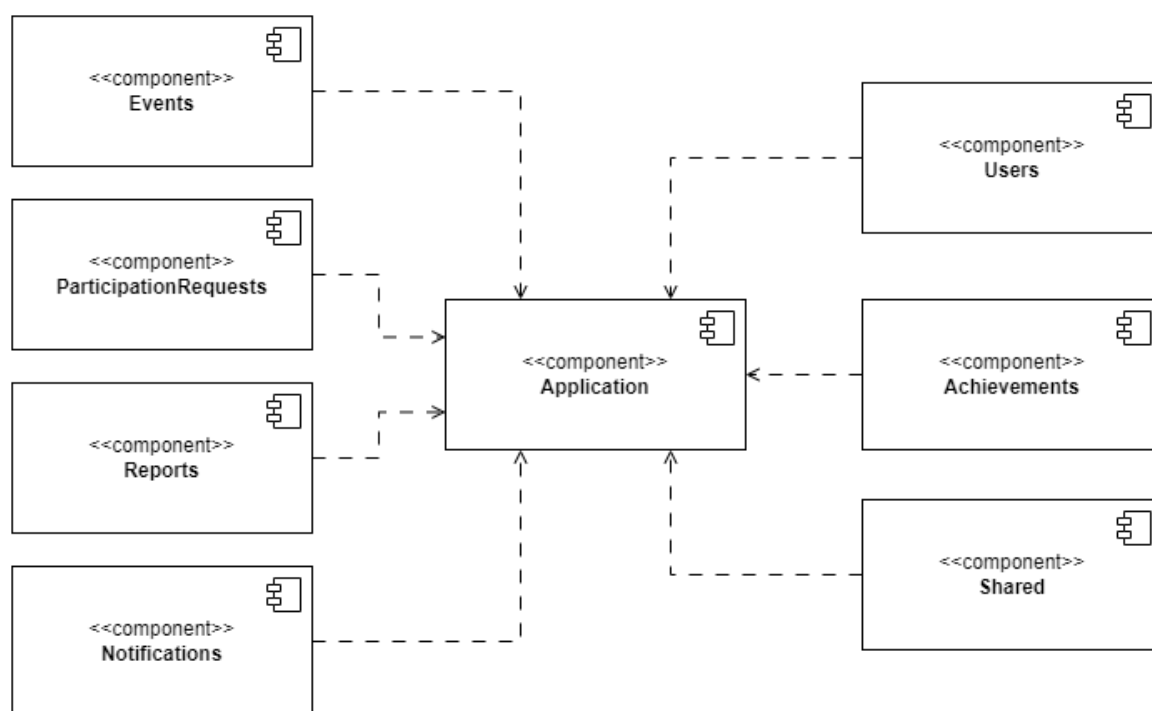


Рисунок 9 – Диаграмма компонентов серверной части приложения

Краткое описание представленных на рисунке 9 модулей приведено ниже.

Точка входа программы (Application) – компонент, который выполняет запуск программы и подключает все остальные компоненты.

Модуль пользователей (Users) содержит всю логику, связанную с управлением пользователями системы: регистрацией, авторизацией,

хранением и редактированием профиля, ролями (гость, волонтер, организатор, администратор). Также он реализует аутентификацию и авторизацию, взаимодействует с системой безопасности.

Модуль мероприятий (Events) отвечает за публикацию, редактирование, отмену мероприятия, просмотр основной информации о мероприятии (название, описание, дату, регион, место проведения); включает в себя логику фильтрации списка мероприятий по различным параметрам.

Модуль заявок на участие (ParticipationRequests) реализует подачу, просмотр и обработку заявок на участие волонтеров в мероприятиях; поддерживает смену статуса заявки (например, на «подтверждена», «отклонена»), а также позволяет организатору управлять списком участников каждого события.

Модуль уведомлений (Notifications) обеспечивает генерацию и доставку уведомлений пользователям по действиям, связанным с изменениями в мероприятиях или статусе заявок; поддерживает хранение и просмотр уведомлений.

Модуль достижений (Achievements) ведет учет активности волонтеров и определяет текущий уровень пользователя в зависимости от количества участия в мероприятиях; включает функциональность по присвоению достижений за выполнение определенных условий.

Модуль новостей (Reports) позволяет организаторам публиковать новости, связанные с деятельностью или мероприятиями; предоставляет функционал для создания, редактирования, удаления и получения новостей; используется для информирования сообщества о важных событиях.

Модуль вспомогательных компонентов (Shared) содержит в себе утилиты и классы для совместного использования модулями, а также фильтры и настройки безопасности.

Каждый модуль (кроме главного модуля) включает в себя слои:

- слой «Controller» – обеспечивают прием HTTP-запросов и возврат HTTP-ответов;

- слой «Service» – содержит основную бизнес-логику;
- слой «Data Access» («Repository» и «Model») – реализуют доступ к данным и взаимодействие с СУБД;
- слой «DTO» и «Mapper» – обеспечивают передачу данных между слоями.

Такую структуру еще называют «MVC». Архитектура MVC (Model-View-Controller) активно используется при разработке веб-приложений благодаря своей гибкости и удобству в реализации пользовательских интерфейсов. Такой подход позволяет организовать код приложения структурированно и логично, что особенно полезно при разработке крупных и масштабируемых систем.

Одним из важных преимуществ использования MVC является повышение удобства тестирования, поскольку каждый компонент может быть проверен отдельно. Кроме того, изменение пользовательского интерфейса не требует значительных изменений в логике приложения, что ускоряет разработку и упрощает внедрение новых функций [13].

2.3 Проектирование графического интерфейса

Проектирование графического интерфейса (UI/UX) играет ключевую роль в создании удобных и интуитивно понятных интерфейсов. Оно включает в себя анализ пользовательских потребностей, разработку логики взаимодействия и визуального оформления. Основные принципы UI/UX-дизайна направлены на повышение удобства использования, минимизацию когнитивной нагрузки и обеспечение доступности. Важными аспектами проектирования являются адаптивность интерфейса, последовательность элементов и интуитивность навигации, что способствует улучшению пользовательского опыта [11].

К ключевым принципам проектирования интерфейсов относятся:

- интерфейс должен быть понятен без инструкций;

- пользователи, как правило, сначала ищут объект, а затем – действие;
- визуальный отклик на действия должен быть максимально быстрым;
- индивидуальные настройки следует предоставлять с осторожностью;
- при длительной работе с задачей пользователь концентрируется на ее выполнении и начинает игнорировать подсказки и сообщения системы.

Все элементы графического интерфейса должны соответствовать современным тенденциям дизайна, быть визуально привлекательными и легко воспринимаемыми. Это повысит удобство навигации и эффективность взаимодействия с приложением.

В результате проектирования графического интерфейса были созданы основные макеты экранов.

На рисунке 10 изображен макет экрана входа на портал.

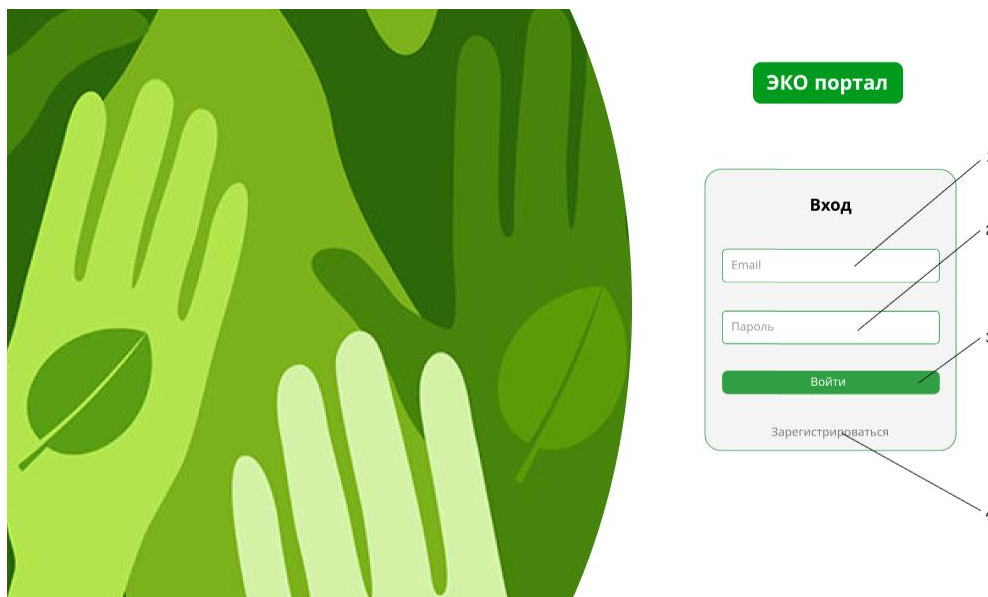


Рисунок 10 – Макет экрана входа на портал

Макет экрана входа содержит следующие элементы:

- 1 – поле ввода email;
- 2 – поле ввода пароля;

3 – кнопку «Войти» для входа в приложения;

4 – кнопку «Зарегистрироваться» для перехода на экран создания аккаунта.

На рисунке 11 изображен макет экрана личного кабинета волонтера.

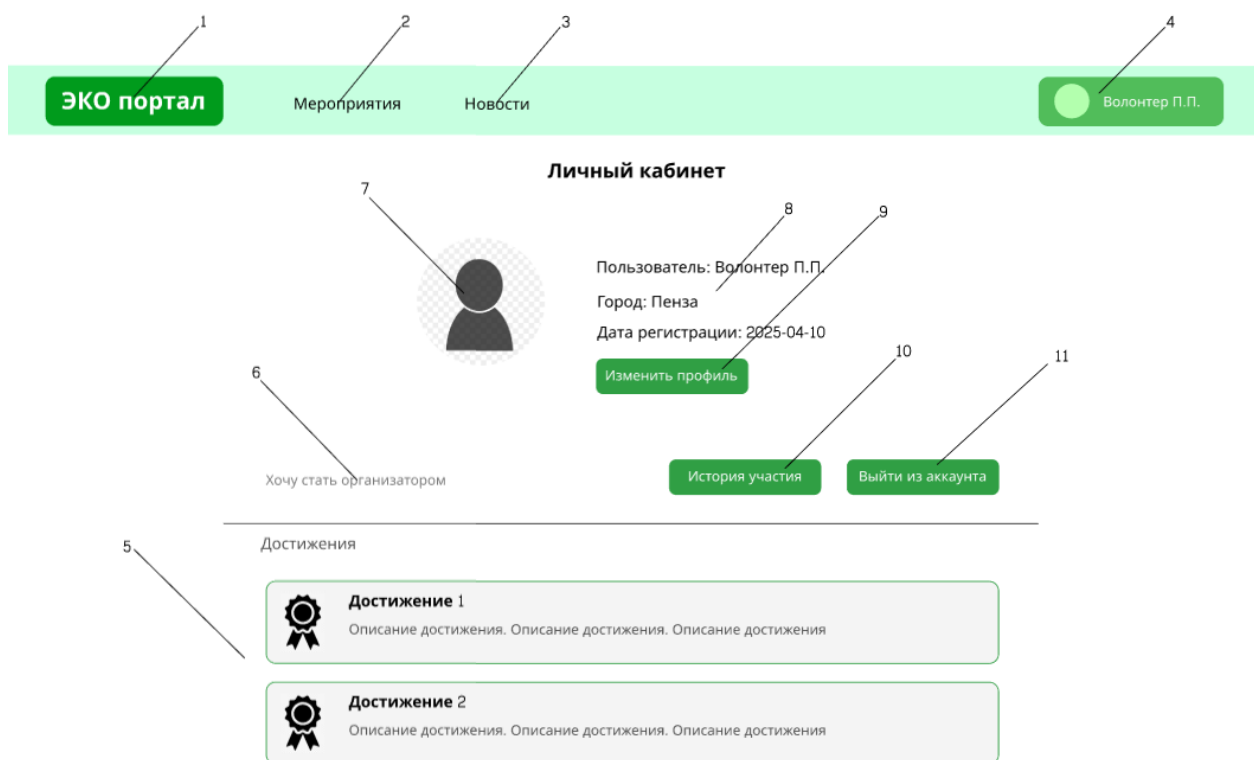


Рисунок 11 – Макет экрана личного кабинета волонтера

Макет экрана личного кабинета волонтера содержит следующие элементы:

1 – кнопка перехода на главную страницу (мероприятия);

2 – кнопка перехода на страницу мероприятий;

3 – кнопка перехода на страницу новостей;

4 – кнопка перехода на страницу личного кабинета;

5 – раздел показа достижений волонтера;

6 – кнопка подачи заявки на регистрацию организации;

7 – фото пользователя (если загружено);

8 – основная информация о волонтере;

9 – кнопка перехода на страницу редактирования профиля;

10 – кнопка перехода на страницу просмотра истории участия;

11 – кнопка инициирования выхода из аккаунта.

Личный кабинет организатора и администратора отличается только большим набором возможных действий, таких как: управление мероприятиями, управление новостями и управление пользователями (только у администратора).

На рисунке 12 изображен макет экрана ленты мероприятий.

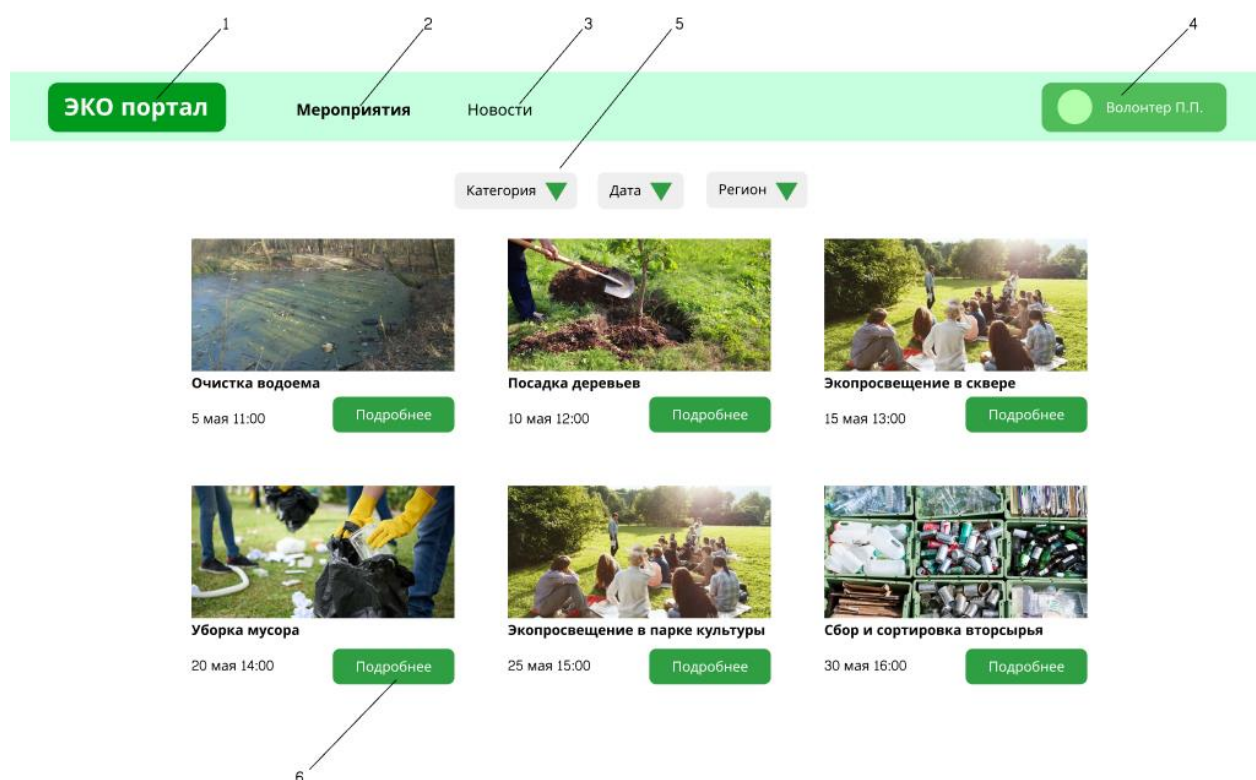


Рисунок 12 – Макет экрана ленты мероприятий

Макет экрана ленты мероприятий содержит следующие элементы:

1 – кнопка перехода на главную страницу (мероприятия);

2 – кнопка перехода на страницу мероприятий;

3 – кнопка перехода на страницу новостей;

4 – кнопка перехода на страницу личного кабинета;

5 – меню выбора фильтров поиска;

6 – кнопка перехода на страницу с описанием мероприятия.

На рисунке 13 изображен макет экрана ленты новостей.

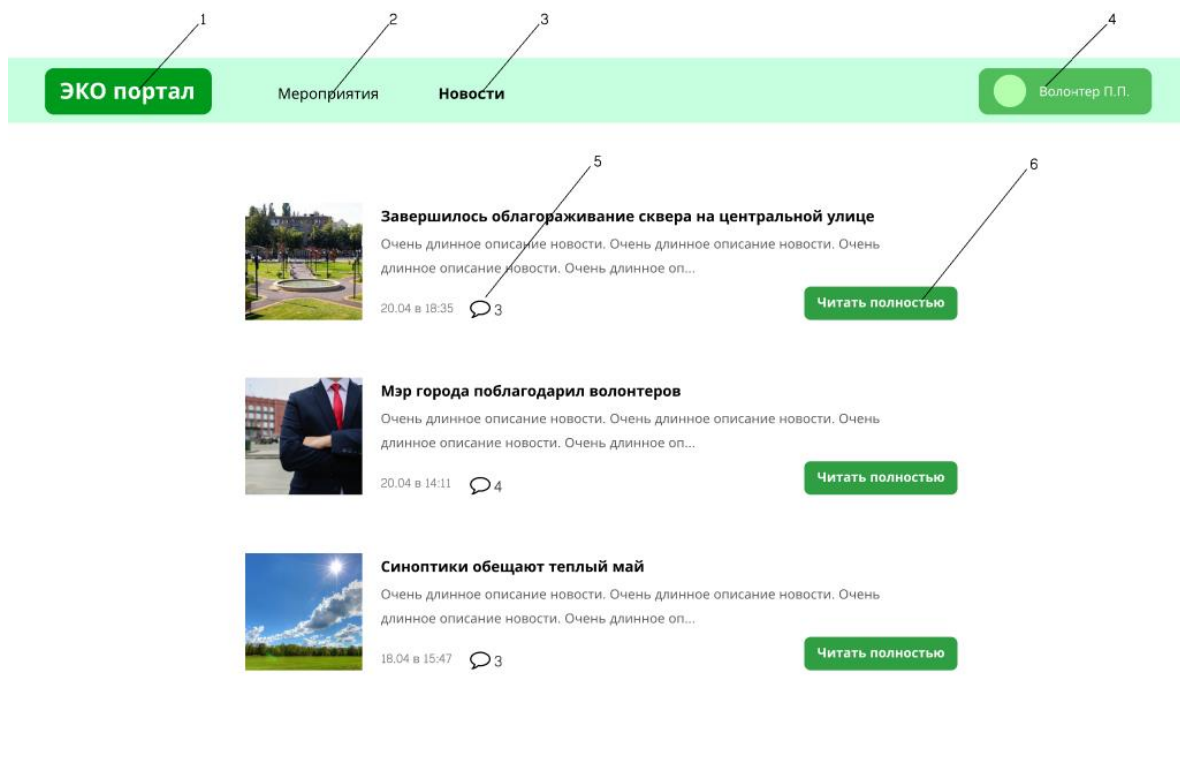


Рисунок 13 – Макет экрана ленты новостей

Макет экрана ленты новостей содержит следующие элементы:

- 1 – кнопка перехода на главную страницу (мероприятия);
- 2 – кнопка перехода на страницу мероприятий;
- 3 – кнопка перехода на страницу новостей;
- 4 – кнопка перехода на страницу личного кабинета;
- 5 – индикатор количества оставленных комментариев;
- 6 – кнопка перехода на страницу с описанием новости.

Таким образом, были спроектированы основные макеты экранов веб-портала. В дальнейшем на основе этих макетов будет разработан пользовательский интерфейс.

2.4 Планирование разработки и оценка бюджета

Разработка программного обеспечения – это трудоемкий и ресурсоемкий процесс, требующий предварительного планирования затрат и рационального распределения бюджета. Оценка бюджета позволяет определить необходимые ресурсы, продолжительность выполнения этапов и

ожидаемую стоимость проекта. Это особенно важно для эффективного управления временем, ресурсами и рисками при создании информационных систем.

Для оценки бюджета разработки web-портала для информационной системы экологического волонтерского движения использовалась система управления проектами ProjectLibre, которая позволяет построить диаграмму Ганта, распределить задачи, установить зависимости и назначить ресурсы, а также рассчитать бюджет.

Оценка бюджета проводилась по двум основным категориям ресурсов:

- трудовые ресурсы (сотрудники);
- материальные ресурсы (оборудование).

Характеристика используемых при оценке бюджета разработки ресурсов приведена в таблице 1.

Таблица 1 – Характеристика используемых ресурсов

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных	Начисление
Аналитик	Трудовой	240/час	380/час	Пропорциональное
Проектировщик	Трудовой	280/час	440/час	Пропорциональное
Backend-разработчик	Трудовой	350/час	550/час	Пропорциональное
Frontend-разработчик	Трудовой	320/час	500/час	Пропорциональное
Тестировщик	Трудовой	270/час	470/час	Пропорциональное
Компьютер	Материальный	75 000	-	В начале
Принтер	Материальный	10 000	-	В начале
Бумага	Материальный	250	-	В начале

На основании списка ресурсов и оценки сложности планируемых при проектировании работ были выделены задачи и время на их выполнение (таблица 2).

Таблица 2 – Задачи проекта

Название задачи	Длительность	Начало	Окончание
Начало проекта	41 дней	08.04.2025, 08:00	03.06.2025, 17:00
Анализ предметной области и требований	5 дней	08.04.2025, 08:00	14.04.2025, 17:00
Анализ экологического волонтерства	1 день	08.04.2025, 08:00	08.04.2025, 17:00
Анализ продуктов-аналогов	1 день	09.04.2025, 08:00	09.04.2025, 17:00
Постановка задачи на разработку	1 день	10.04.2025, 08:00	10.04.2025, 17:00
Анализ требований	2 дней	11.04.2025, 08:00	14.04.2025, 17:00
Анализ функциональных требований	1 день	11.04.2025, 08:00	11.04.2025, 17:00
Анализ нефункциональных требований	1 день	14.04.2025, 08:00	14.04.2025, 17:00
Завершение анализа предметной области и требований	0 дней	14.04.2025, 17:00	14.04.2025, 17:00
Проектирование программных средств	9 дней	15.04.2025, 08:00	25.04.2025, 17:00
Выбор технологий разработки	1 день	15.04.2025, 08:00	15.04.2025, 17:00
Проектирование архитектуры приложения	3 дней	16.04.2025, 08:00	18.04.2025, 17:00
Проектирование графического интерфейса	3 дней	21.04.2025, 08:00	23.04.2025, 17:00
Планирование разработки и оценка бюджета	2 дней?	24.04.2025, 08:00	25.04.2025, 17:00
Завершение проектирования	0 дней	25.04.2025, 17:00	25.04.2025, 17:00

Продолжение таблицы 2

Название задачи	Длительность	Начало	Окончание
Реализация web-портала для информационной системы экологического волонтерского движения	11 дней	28.04.2025, 08:00	12.05.2025, 17:00
Реализация базы данных	2 дней	28.04.2025, 08:00	29.04.2025, 17:00
Реализация архитектуры программного обеспечения	5 дней	30.04.2025, 08:00	06.05.2025, 17:00
Реализация механизма аутентификации и авторизации на основе JWT	1 день	30.04.2025, 08:00	30.04.2025, 17:00
Реализация программных модулей	4 дней	01.05.2025, 08:00	06.05.2025, 17:00
Реализация пользовательского интерфейса	4 дней	07.05.2025, 08:00	12.05.2025, 17:00
Реализация взаимодействия с сервером	1 день	07.05.2025, 08:00	07.05.2025, 17:00
Реализация основных страниц интерфейса	3 дней	08.05.2025, 08:00	12.05.2025, 17:00
Завершение реализации	0 дней	12.05.2025, 17:00	12.05.2025, 17:00
Контроль качества программного обеспечения	8 дней	13.05.2025, 08:00	22.05.2025, 17:00
Функциональное тестирование на основе классов эквивалентности	3 дней	13.05.2025, 08:00	15.05.2025, 17:00
Анализ сложности кода модуля новостей	3 дней	16.05.2025, 08:00	20.05.2025, 17:00
Автоматизированное тестирование модуля мероприятий	2 дней	21.05.2025, 08:00	22.05.2025, 17:00
Завершение контроля качества	0 дней	22.05.2025, 17:00	22.05.2025, 17:00

Окончание таблицы 2

Название задачи	Длительность	Начало	Окончание
Оформление документации	8 дней	23.05.2025, 08:00	03.06.2025, 17:00
Завершение проекта	0 дней	03.06.2025, 17:00	03.06.2025, 17:00

Для планирования этапов реализации программного обеспечения и контроля сроков выполнения задач была использована диаграмма Ганта. Этот инструмент проектного управления представляет собой график, в котором задачи отображаются в виде горизонтальных полос на временной шкале. Он позволяет легко отслеживать ход выполнения проекта, определять зависимости между задачами и своевременно корректировать план в случае отклонений.

Разработанная диаграмма Ганта приведена на рисунке 14.

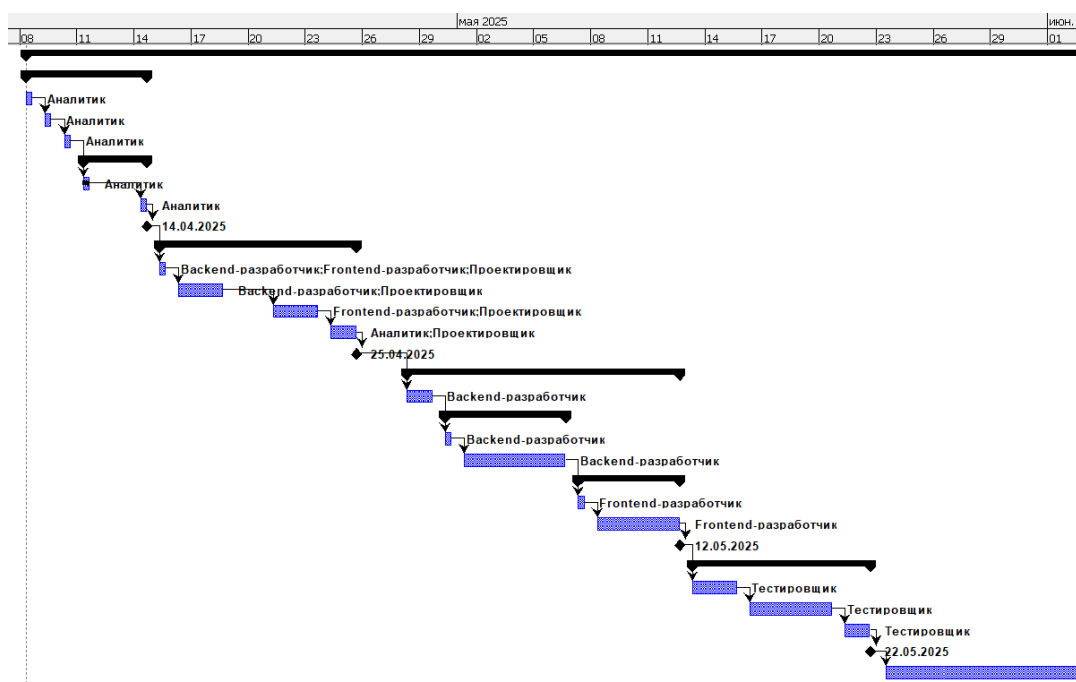


Рисунок 14 – Диаграмма Ганта

Использование диаграммы Ганта повышает прозрачность и управляемость проекта, позволяет эффективно распределять ресурсы, расставлять приоритеты и координировать работу команды. Благодаря этому

снижается вероятность пропуска сроков и повышается общая успешность проекта [12].

Таким образом, по результатам оценки бюджета разработки мобильного приложения была составлена диаграмма Ганта, рассчитано время необходимое на полный цикл разработки проекта, которое составляет 41 день, на основании необходимых ресурсов рассчитан бюджет проекта, который равен 212 116 рублей.

3 Реализация web-портала для информационной системы экологического волонтерского движения

3.1 Реализация базы данных

Для хранения и обработки информации в системе используется объектно-реляционная СУБД PostgreSQL, которая обеспечивает надежность, масштабируемость и поддержку сложных связей между сущностями.

Разработка структуры базы данных производилась на основе анализа предметной области и проектирования архитектуры. Были выделены ключевые сущности: пользователи, мероприятия, заявки на участие, новости, комментарии, уведомления и достижения. Таблицы организованы с учетом связей «один ко многим» и «многие ко многим», с использованием внешних ключей и ограничений целостности.

Диаграмма «сущность-связь», отображающая структуру таблиц в базе данных, представлена на рисунке 15.

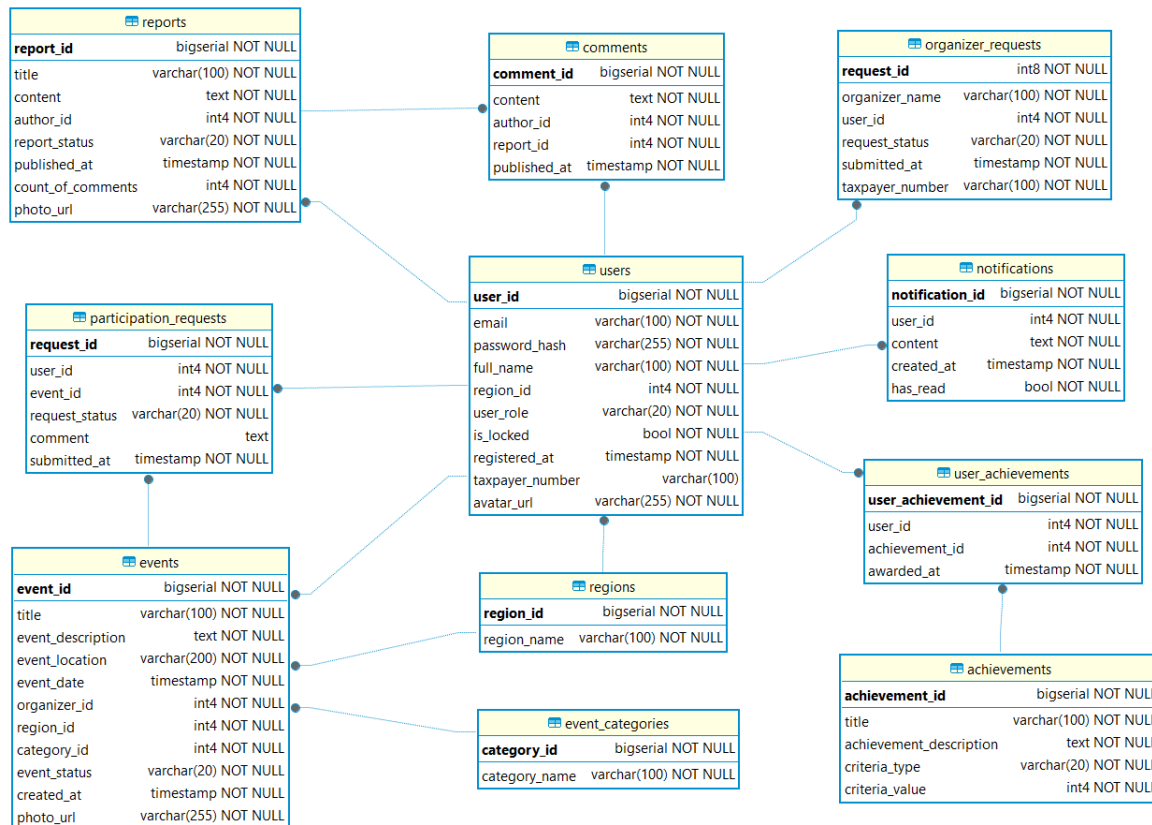


Рисунок 15 – Диаграмма «сущность-связь»

Сущности базы данных организованы с учетом типов связей: «один ко многим» и «многие ко многим». Например, сущность «users» (пользователи) связана с сущностью «regions» (регионы), а также с сущностями «organizer_requests» (заявки на регистрацию организаций) и «participation_requests» (заявки на участие в мероприятиях) через внешние ключи. В свою очередь, таблица «events» (мероприятия) включает ссылки на пользователей (организаторов), регионы и категории мероприятий, что обеспечивает структурированное хранение информации о событиях.

Для реализации механизма управления доступом и роли пользователей в системе предусмотрены ограниченные значения в поле «user_role» таблицы «users», с возможными ролями «VOLUNTEER», «ORGANIZER» и «ADMIN». Каждое событие может быть привязано к одному организатору, региону и категории, что позволяет эффективно классифицировать и управлять мероприятиями. В то же время, системы заявок (на участие и на регистрацию организации) позволяют пользователям взаимодействовать с системой через заявочные процессы, имеющие различные статусы («PENDING», «APPROVED» и «REJECTED»).

Исходный код создания таблиц базы данных представлен в приложении Б.

3.2 Реализация архитектуры программного обеспечения

3.2.1 Реализация механизма аутентификации и авторизации на основе JWT

Одним из ключевых аспектов информационной безопасности веб-приложения является надежная система аутентификации и авторизации пользователей. В разрабатываемом приложении данная функциональность реализована с использованием фреймворка «Spring Security» в сочетании с механизмом JWT (JSON Web Token). Такая архитектура позволяет обеспечить безопасный и масштабируемый способ контроля доступа на основе ролей без необходимости хранения состояния сеанса на сервере.

Процесс аутентификации и авторизации реализован через следующий набор компонентов:

- класс-фильтр «JwtAuthFilter», обрабатывающий входящие запросы и извлекающий токен;
- класс-сервис «JwtService», отвечающий за генерацию, валидацию и парсинг токенов;
- класс «AuthService», реализующий бизнес-логику регистрации, входа и обновления токена;
- класс-контроллер «AuthController», принимающий соответствующие HTTP-запросы;
- класс «SecurityConfig», содержащий конфигурацию фильтров, прав доступа и исключений.

JWT представляет собой закодированный JSON-объект, содержащий информацию о пользователе и сроке действия токена. Приложение использует два типа токенов:

- «access» токен – имеет короткий срок действия, используется при каждом запросе;
- «refresh» токен – используется для обновления «access» токена без повторной аутентификации.

Токены подписываются с помощью алгоритма «HMAC SHA-256» и секретного ключа, заданного в файле конфигурации проекта «application.yml».

Генерация «access» и «refresh» токенов производится в классе «JwtService». Метод генерации «access» токена представлен на листинге 1.

Листинг 1 – Генерация «access» токена

```
public String generateAccessToken(User user) {
    return Jwts.builder()
        .setSubject(user.getEmail())
        .claim("role", user.getRole().name())
        .setIssuedAt(new Date())
        .setExpiration(new Date(System.currentTimeMillis() +
accessExpiration))
        .signWith(accessKey, SignatureAlgorithm.HS256)
```

```
        .compact();
    }
}
```

Валидация токена осуществляется при каждом запросе, поступающем в систему, в фильтре «JwtAuthFilter», который наследуется от «OncePerRequestFilter». Он извлекает из входящего HTTP-запроса заголовок «Authorization», проверяет наличие и корректность токена, а затем – наличие соответствующего пользователя [14]. Фрагмент кода фильтра представлен в листинге 2.

Листинг 2 – Фрагмент фильтра «JwtAuthFilter»

```
if (!jwtService.isAccessTokenValid(jwt)) {
    filterChain.doFilter(request, response);
    return;
}
...
UsernamePasswordAuthenticationToken authToken =
    new UsernamePasswordAuthenticationToken(user, null,
user.getAuthorities());
SecurityContextHolder.getContext().setAuthentication(authToken);
```

Таким образом, каждый валидный токен восстанавливает контекст безопасности пользователя в рамках текущего запроса.

Логика входа, регистрации и обновления токена реализована в классе «AuthService». В листинге 3 представлен метод входа в систему.

Листинг 3 – Метод «login» в классе «AuthService»

```
public AuthResponse login(LoginRequest request) throws Exception
{
    authenticationManager.authenticate(
        new
UsernamePasswordAuthenticationToken(request.getEmail(),
request.getPassword())
    );
    var user = userRepository.findByEmail(request.getEmail())
        .orElseThrow(() -> new Exception("Пользователь не
найден"));
    return new AuthResponse(
        jwtService.generateAccessToken(user),
        jwtService.generateRefreshToken(user)
    );
}
```

Учетные данные передаются в виде объекта DTO-класса «LoginRequest», содержащего поля email и пароля пользователя. Проверка

учетных данных осуществляется через «AuthenticationManager». Он в свою очередь содержит в себе логику нахождения пользователя в базе данных и проверки учетных данных на корректность. После чего пользователю возвращаются токены доступа.

Класс-контроллер «AuthController» принимает соответствующие HTTP-запросы на вход, регистрацию и обновление токена. Все маршруты этого контроллера являются открытыми. Метод обработки запроса на вход приведен ниже (листинг 4).

Листинг 4 – Метод обработки запроса на вход в классе «AuthController»

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest
request) {
    return ResponseEntity.ok(authService.login(request));
}
```

Все настройки безопасности определены в классе SecurityConfig. Здесь задаются разрешенные и защищенные маршруты, настраивается фильтрация, CORS, поведение при ошибках и внедряется фильтр проверки JWT (листинг 5).

Листинг 5 – Настройка разрешений и фильтрации

```
http.csrf(csrf -> csrf.disable())
    .cors(Customizer.withDefaults())
    .authorizeHttpRequests(auth -> auth
        .requestMatchers("/api/auth/**").permitAll()
        .requestMatchers(HttpMethod.GET, "/api/events",
"/api/reports").permitAll()
        .anyRequest().authenticated()
    )
    .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class)
    ...
```

Для ограничения доступа к маршрутам по ролям в приложении также используется аннотация «@PreAuthorize» в методах контроллеров.

Процесс аутентификации и авторизации наглядно представлен на диаграмме последовательности (рисунок 16).

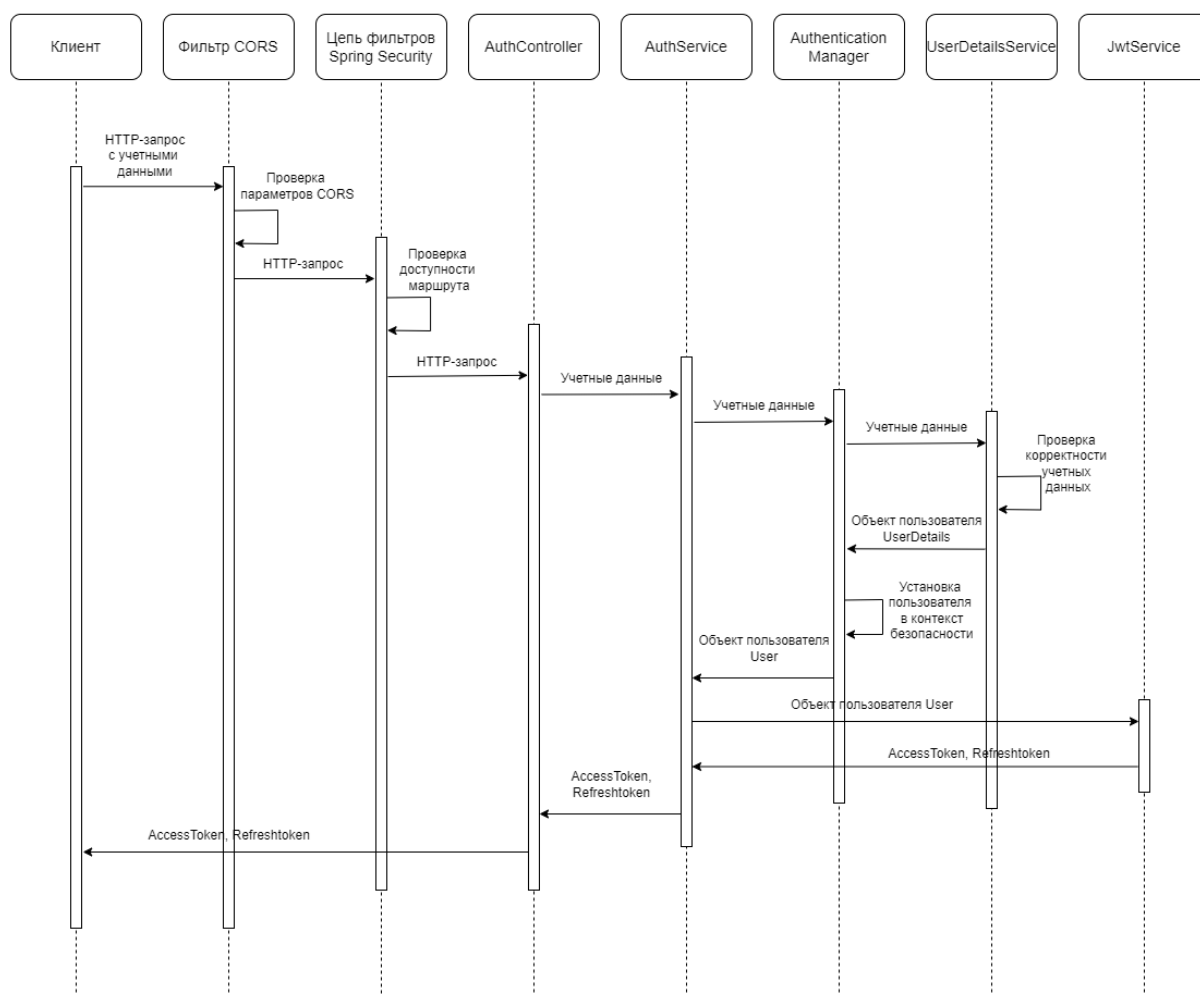


Рисунок 16 – Диаграмма последовательности для процесса аутентификации и авторизации на основе JWT

Таким образом, в приложении реализована полнофункциональная система аутентификации и авторизации, основанная на современных принципах безопасности: без сохранения сессий, с отдельным использованием «access» и «refresh» токенов, защитой маршрутов, контролем ролей и корректной обработкой исключений – это обеспечивает надежный доступ к защищенным ресурсам.

3.2.2 Реализация модуля мероприятий

Модуль мероприятий (events) реализует основную функциональность, связанную с созданием, редактированием, публикацией, отображением и управлением событиями. Он выбран в качестве примера, поскольку отражает

общую архитектурную модель всех модулей приложения: от уровня хранения данных до взаимодействия с пользователем через REST API. Архитектура модуля построена на принципах слоистой структуры и включает следующие компоненты: класс-модель, интерфейс-репозиторий, класс-сервис, DTO-классы, класс-маппер и класс-контроллер. Диаграмма классов модуля мероприятий представлена на рисунке 17.

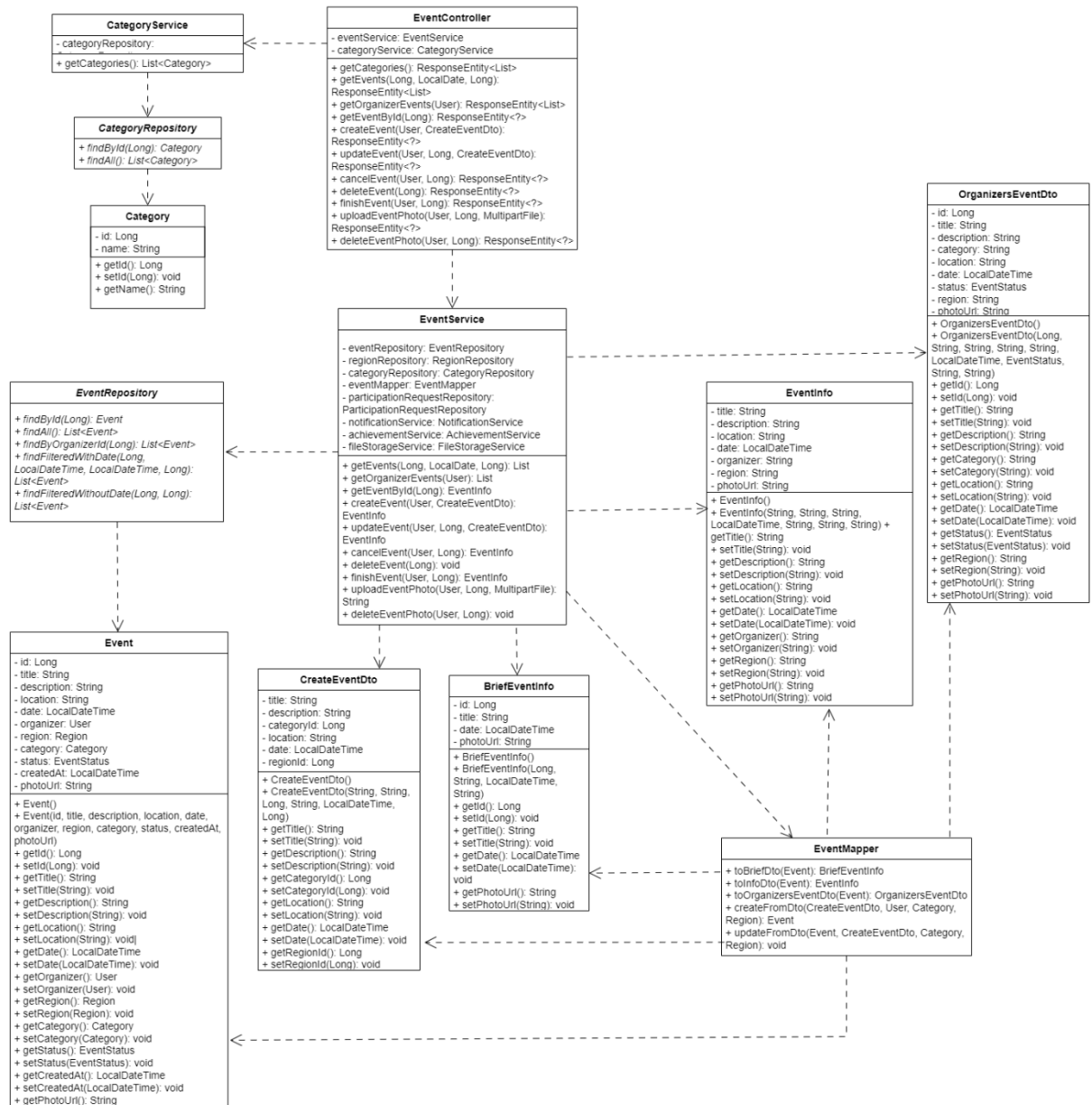


Рисунок 17 – Диаграмма классов модуля мероприятий

Класс-модель «Event» описывает одноименную сущность в базе данных. Он также связан с другими классами-моделями: «User» (организатор),

«Region» (регион проведения), «Category» (категория мероприятия). В листинге 6 представлен фрагмент класса «Event».

Листинг 6 – Класс «Event»

```
@Data
@Entity
@Table(name = "events")
public class Event {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;
    private String location;
    private LocalDateTime date;

    @ManyToOne
    private User organizer;

    @ManyToOne
    private Region region;

    @ManyToOne
    private Category category;

    @Enumerated(EnumType.STRING)
    private EventStatus status;

    private String photoUrl;
    private LocalDateTime createdAt;
}
```

Перечисление «EventStatus» отражает текущий статус мероприятия и может принимать значения: «PUBLISHED», «CANCELLED», «FINISHED» и «DELETED».

Интерфейс «EventRepository», расширяющий «JpaRepository», реализует доступ к данным. Помимо базовых методов, он содержит пользовательские запросы для фильтрации мероприятий по дате, региону и категории. В листинге 7 приведен метод нахождения мероприятий по указанным фильтрам.

Листинг 7 – Метод поиска и фильтрации мероприятий

```
@Query("""
    SELECT e FROM Event e
    WHERE e.date > CURRENT_TIMESTAMP
    """)
```

```

        AND e.status =
com.eco_portal.events.model.EventStatus.PUBLISHED
        AND (:regionId IS NULL OR e.region.id = :regionId)
        AND (:categoryId IS NULL OR e.category.id = :categoryId)
        ORDER BY e.date ASC
    """)
List<Event> findFilteredWithoutDate(@Param("regionId") Long
regionId, @Param("categoryId") Long categoryId);

```

Метод «findFilteredWithoutDate» использует аннотацию, в которой описан запрос на языке JPQL (Java Persistence Query Language) – объектно-ориентированном язык запросов, который по синтаксису близок к SQL, но оперирует сущностями и их полями, а не таблицами и столбцами. Такой подход использует встроенный в Spring Boot фреймворк «Hibernate». Он обеспечивает автоматическое сопоставление объектов Java с таблицами реляционной базы данных (ORM – Object-Relational Mapping), что позволяет разработчику работать с базой через объектную модель, не прибегая к ручному написанию SQL-запросов.

Использование Hibernate и JPQL в сочетании с JpaRepository обеспечивает высокую читаемость и переиспользуемость кода, ускоряет разработку и минимизирует количество низкоуровневых операций с базой данных [15].

Класс «EventService» реализует основную бизнес-логику модуля. Он содержит методы для:

- получения мероприятий;
- создания и редактирования мероприятий;
- публикации, отмены и завершения мероприятий;
- загрузки фото мероприятия;
- уведомления участников.

Метод создания мероприятия представлен в листинге 8.

Листинг 8 – Метод создания мероприятия

```

public EventInfo createEvent(User organizer, CreateEventDto dto)
throws Exception {
    Category category =
categoryRepository.findById(dto.getCategoryId())

```

```

        .orElseThrow(() -> new Exception("Категория не
найдена"));
        Region region = regionRepository.findById(dto.getRegionId())
        .orElseThrow(() -> new Exception("Регион не
найден"));

        Event event = eventMapper.createFromDto(dto, organizer,
category, region);
        return eventMapper.toInfoDto(eventRepository.save(event));
    }

```

Метод проверяет корректность входных данных, создает объект класса «Event», сохраняет его в базе и возвращает DTO-объект с полной информацией о созданном мероприятии.

В методе редактирования «updateEvent» дополнительно проверяется, имеет ли текущий пользователь право на изменение мероприятия (является ли он организатором и не завершено ли событие). При изменении и отмене мероприятия осуществляется отправка уведомлений участникам с помощью метода из модуля «Notifications».

В целях разделения слоев и оптимизации передачи данных между клиентом и сервером используются специализированные классы DTO:

- класс «CreateEventDto» – для создания и обновления мероприятий;
- класс «BriefEventInfo» – краткое представление о событии (для списков);
- класс «EventInfo» – полная информация;
- класс «OrganizersEventDto» – информация о событии, созданном организатором.

Установка соответствующих значений между сущностями и DTO реализован вручную в классе «EventMapper». Метод получения краткой информации о мероприятии представлен в листинге 9.

Листинг 9 – Метод «toBriefDto»

```

public BriefEventInfo toBriefDto(Event event) {
    return new BriefEventInfo(
        event.getId(),
        event.getTitle(),
        event.getDate(),
        event.getPhotoUrl()
    );
}

```

```
);  
}
```

Такой подход позволяет избежать избыточной передачи данных и повысить читаемость кода API.

Класс «EventController» содержит REST-эндпоинты (маршруты) для взаимодействия с клиентом. Все маршруты, начинающиеся с «/api/events/...», обрабатываются этим контроллером. Методы аннотированы «@PreAuthorize» для ограничения доступа по ролям. Например, метод обработки HTTP-запроса на создание мероприятия содержит аннотацию «@PreAuthorize» с указанной ролью «ORGANIZER» для предотвращения несанкционированного доступа (листинг 10).

Листинг 10 – Метод обработки HTTP-запроса на создание мероприятия

```
@PostMapping  
@PreAuthorize("hasRole('ORGANIZER')")  
public ResponseEntity<?> createEvent(@AuthenticationPrincipal  
User user, @RequestBody CreateEventDto dto) {  
    try {  
        return ResponseEntity.ok(eventService.createEvent(user,  
dto));  
    } catch (Exception e) {  
        return  
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage(  
));  
    }  
}
```

Также реализованы методы обработки запросов на публикацию, завершение и отмену мероприятия, а также обработка запросов на загрузку фотографии мероприятия.

Таким образом, модуль мероприятий реализован с соблюдением принципов модульности, повторяемости и разделения ответственности. Его архитектура и структура аналогичны другим модулям проекта, что обеспечивает единый стиль разработки, облегчает сопровождение и масштабирование. Использование DTO, сервисного слоя и контроллера позволяет эффективно обрабатывать запросы и выполнять логику на серверной стороне.

Исходный код серверных модулей представлен в приложении А.

3.3 Реализация пользовательского интерфейса

3.3.1 Реализация взаимодействия с сервером через REST API

Пользовательский интерфейс приложения реализован в виде одностраничного веб-приложения (SPA) на фреймворке React, которое осуществляет взаимодействие с серверной частью через REST API. Обмен данными между клиентом и сервером осуществляется в формате JSON с использованием асинхронных HTTP-запросов. Основным инструментом для работы с API на клиенте – библиотека «Axios».

Axios – это популярная JavaScript-библиотека для выполнения HTTP-запросов из браузера или Node.js. В рамках проекта Axios использовался в React-приложении для реализации взаимодействия клиентской части с REST API сервером. Она обеспечивает поддержку всех стандартных HTTP-методов, позволяет централизованно настраивать заголовки, перехватывать запросы и ответы, обрабатывать ошибки, а также автоматически сериализует данные в формат JSON. Использование Axios упростило реализацию аутентификации (через перехватчики), обновления «access» токенов и отправки защищенных запросов. Это позволило значительно повысить надежность и масштабируемость клиентской логики [16].

В целях централизованного управления HTTP-запросами в проекте создан специализированный модуль `axiosInstance.js`, в котором настраивается экземпляр Axios с базовым адресом API и перехватчиками запросов и ответов (листинг 11). Перехватчик запроса автоматически добавляет заголовок «Authorization», содержащий «access» токен, ко всем защищенным маршрутам, кроме авторизации.

Листинг 11 – Базовая конфигурация Axios

```
const instance = axios.create({
  baseURL: "http://localhost:8081/api"
});
instance.interceptors.request.use((config) => {
  const token = localStorage.getItem("accessToken");
  if (token && !config.url.includes("/auth")) {
```

```

    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

```

Таким образом, клиент может взаимодействовать с защищенными ресурсами без повторного ввода данных при каждом запросе.

В случае, если «access» токен истек, Axios перехватывает ответ с кодом «401 Unauthorized» и выполняет автоматическую попытку обновления токена с помощью «refresh» токена. Если обновление прошло успешно – оригинальный запрос отправляется повторно с новым токеном. Перехватчик ответа сервера, осуществляющий обновление «access» токена, представлен в листинге 12.

Листинг 12 – Перехватчик ответа, осуществляющий обновление токена

```

instance.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;

    if (error.response?.status === 401 &&
    !originalRequest._retry) {
      originalRequest._retry = true;
      const refreshToken = localStorage.getItem("refreshToken");

      const res = await axios.post("/auth/refresh-token", {
        refreshToken
      });
      const { accessToken, refreshToken: newRefresh } =
        res.data;

      localStorage.setItem("accessToken", accessToken);
      localStorage.setItem("refreshToken", newRefresh);
      originalRequest.headers.Authorization = `Bearer
      ${accessToken}`;
      return instance(originalRequest);
    }

    return Promise.reject(error);
  }
);

```

В случае, если обновление токена не удалось (например, «refresh» токен истек), пользователь перенаправляется на страницу входа, а локальное хранилище браузера очищается.

Компонент «LoginPage» содержит формы для авторизации и регистрации пользователя. После успешного входа или регистрации токены сохраняются в «localStorage» (локальное хранилище веб-браузера), и пользователь перенаправляется на основную страницу. Процесс аутентификации представлен в листинге 13.

Листинг 13 – Аутентификация пользователя

```
const response = await axios.post("/auth/login", { email,
password });
const { accessToken, refreshToken } = response.data;
localStorage.setItem("accessToken", accessToken);
localStorage.setItem("refreshToken", refreshToken);
navigate("/events");
```

Если данные введены некорректно или сервер возвращает ошибку, пользователю отображается сообщение с помощью библиотеки «react-toastify».

При загрузке личного кабинета волонтера выполняется несколько последовательных запросов:

- получение профиля пользователя (листинг 14);
- загрузка списка достижений;
- получение списка регионов.

Все запросы осуществляются через ранее настроенный экземпляр «Axios», и в случае ошибки пользователю отображается уведомление.

Листинг 14 – Загрузка профиля

```
axios.get("/users/profile")
  .then(res => setProfile(res.data))
  .catch(() => toast.error("Не удалось загрузить профиль"));
```

Также предусмотрено обновление токенов при загрузке страницы через декодирование JWT и проверку срока действия.

Интерфейс разрабатываемого программного обеспечения содержит формы, с которых данные отправляются на сервер с помощью методов POST или PUT. Примеры таких действий:

- обновление профиля (листинг 15);
- отправка заявки на получение статуса организатора;

- загрузка аватара или фотографии мероприятия.

Листинг 15 – Отправка формы профиля

```
await axios.put("/users/profile", editData);
toast.success("Профиль обновлен");
```

Во всех компонентах используется библиотека «react-toastify» для отображения сообщений. Ниже приведены основные типы выводимых сообщений:

- toast.success() – успешные действия;
- toast.error() – ошибки;
- toast.warn() – предупреждения.

Это обеспечивает наглядную обратную связь и улучшает пользовательский опыт (UX).

Таким образом, в клиентской части приложения реализована надежная и масштабируемая система взаимодействия с сервером, основанная на REST API, JWT-авторизации и централизованном управлении запросами через «Axios». Использование перехватчиков ответов сервера позволяет автоматически обрабатывать токены и ошибки, а отображение сообщений через «react-toastify» делает интерфейс понятным и удобным для пользователя. В результате обеспечивается надежная интеграция между «frontend» и «backend» частями проекта и соответствует современным требованиям к web-разработке.

3.3.2 Реализация личного кабинета волонтера

Личный кабинет пользователя является одной из ключевых страниц клиентского приложения. Он предоставляет волонтеру доступ к информации о собственном профиле, достижениям, истории участия в мероприятиях, а также к возможностям редактирования данных и подачи заявки на получение статуса организатора. В данном разделе рассматривается реализация страницы личного кабинета для роли VOLUNTEER.

Страница реализована в компоненте «VolunteerProfilePage», написанном на React с использованием функционального подхода. Интерфейс формируется через JSX-шаблон. React обрабатывает JSX как расширение JavaScript и преобразует его в чистые вызовы JavaScript-функций. Каждый фрагмент JSX-компонента транслируется в вызов «React.createElement()», создающий React-элемент – специальный объект с описанием узла интерфейса. Затем «ReactDOM.render()» анализирует эти объекты и создает реальные DOM-узлы, применяя методы вроде «document.createElement», устанавливая свойства и включая вложенные элементы, фактически формируя HTML-структуру в браузере [17].

После загрузки страницы происходит:

- проверка авторизации пользователя;
- получение данных профиля;
- загрузка достижений;
- получение списка регионов.

Эти действия выполняются в блоке «useEffect», что обеспечивает их однократное выполнение при монтировании компонента.

Данные профиля и достижений загружаются с сервера и сохраняются в локальном состоянии компонента. В листинге 16 приведен пример запроса профиля пользователя.

Листинг 16 – Получение профиля пользователя

```
axios.get("/users/profile")
  .then((res) => {
    setProfile(res.data);
    setEditData({
      fullName: res.data.fullName,
      regionId: res.data.regionId,
    });
  })
  .catch(() => toast.error("Не удалось загрузить профиль"));
```

В результате полученные данные используются для отображения имени, региона, аватара и даты регистрации пользователя. Блок отображения данных профиля представлен в листинге 17.

Листинг 17 – JSX-фрагмент отображения профиля

```
<p><strong>{profile.fullName}</strong></p>
<p><strong>Регион:</strong> {profile.regionName}</p>
<p><strong>Дата регистрации:</strong> {new
Date(profile.registeredAt).toLocaleDateString("ru-RU")}</p>
```

Формирование интерфейса происходит динамически. В зависимости от состояния данные вставляются прямо в шаблон.

Для редактирования профиля реализовано модальное окно, в котором пользователь может изменить имя и регион. Изменения отправляются на сервер методом PUT (листинг 18).

Листинг 18 – Сохранение изменений профиля

```
await axios.put("/users/profile", editData);
toast.success("Профиль обновлен");
```

Поля формы привязаны к состоянию компонента через «useState», что позволяет отслеживать изменения в реальном времени и сохранять их при подтверждении.

Пользователь может изменить фотографию профиля. Загрузка файла реализована через элемент «input type=file» и передачу данных на сервер в формате «FormData». Загрузка фото на сервер представлена в листинге 19.

Листинг 19 – Загрузка фотографии

```
const formData = new FormData();
formData.append("file", file);
await axios.post("/users/profile/avatar", formData);
```

После успешной загрузки страница перезагружается, и новое изображение подставляется автоматически. А в нижней части личного кабинета отображается список достижений пользователя. Каждый элемент включает в себя название, описание, прогресс и отметку о завершенности.

Данные запрашиваются с сервера по маршруту «/achievements/progress» и отображаются в виде карточек (листинг 20).

Листинг 20 – JSX-фрагмент отображения достижений

```
{achievements.map((a) => (
  <div key={a.id} className="achievement-card">
    <strong>{a.title}</strong>
    <p>{a.description}</p>
```

```

    <p>{a.current}/{a.required}</p>
    {a.achieved && <FaCheckCircle />}
  </div>
  )))

```

Пользователь может подать заявку на роль организатора, заполнив модальное окно с названием организации и ИНН. Данные отправляются методом POST на маршрут «/organizer-requests» (листинг 21).

Листинг 21 – Отправка заявки на регистрацию организатора

```

await axios.post("/organizer-requests", orgData);
toast.success("Заявка отправлена");

```

Интерфейс закрывается, а пользователь получает уведомление об успешной отправке.

Перед отображением страницы выполняется проверка валидности JWT-токена. Если токен истек, он автоматически обновляется, иначе пользователь перенаправляется на страницу входа. Для этого используется хук «useAuthReady», фрагмент которого приведен в листинге 22.

Листинг 22 – Фрагмент «useAuthReady»

```

const { exp } = jwtDecode(access_token);
const isExpired = exp * 1000 < Date.now();

if (isExpired) {
  const res = await axios.post("/auth/refresh-token", {
    refreshToken
  });
  ...
}

```

Таким образом, обеспечивается защита маршрута и доступ только авторизованным пользователям.

Реализация личного кабинета волонтера демонстрирует комплексный подход к разработке пользовательских интерфейсов: загрузка данных через формы, привязка к состоянию, взаимодействие с API, отображение динамического контента и работа с файлами.

Остальные страницы интерфейса приложения реализованы с использованием аналогичных подходов.

4 Контроль качества программного обеспечения

4.1 Функциональное тестирование на основе классов

эквивалентности

Функциональное тестирование является важной частью процесса разработки программного обеспечения, так как оно помогает убедиться в соответствии приложения заявленным требованиям и ожидаемым функциям. Это тестирование позволяет выявить ошибки и несоответствия на ранних стадиях разработки, что снижает риски и затраты на исправление в будущем. Кроме того, при внесении изменений в код функциональное тестирование помогает гарантировать, что новые функции не нарушают уже существующий функционал.

Для разрабатываемой программы было проведено функциональное тестирование ввода значения ИНН при отправке заявки на регистрацию организации, основанное на классах эквивалентности.

Функциональное тестирование с применением метода эквивалентного разбиения позволяет значительно оптимизировать число тестовых сценариев без потери качества проверки. Входные данные разбиваются на эквивалентные классы – группы значений, при обработке которых система ведет себя одинаково. Затем достаточно протестировать по одному представителю из каждого класса: если один элемент проходит успешно (или вызывает ожидаемую ошибку), можно считать, что поведение распространяется на весь класс [18].

Выделенные классы эквивалентности приведены в таблице 3.

Таблица 3 – Классы эквивалентности для ИНН

Свойство	Правильные КЭ	Неправильные КЭ
Длина номера	10 (1)	0 – 9 (2) 11 – ∞ (3)
Содержание	Цифры 0 – 9 (4)	Остальное (5)

Интервалы классов эквивалентности длины ИНН представлены на рисунке 18.



Рисунок 18 – Интервалы классов эквивалентности для длины ИНН

На основе выделенных классов эквивалентности были сформированы тест-кейсы для поля ввода ИНН организации (таблица 4).

Таблица 4 – Тест-кейсы для поля ввода ИНН организации

№	КЭ	Входные значения	Выходные значения
1	1 4	1234567890	Успешно. ИНН принят.
2	2 4	12345678	Ошибка: ИНН должен состоять из 10 цифр.
3	3 4	123456789012	Ошибка: ИНН должен состоять из 10 цифр.
4	1 5	12345678a0	Ошибка: ИНН может содержать только цифры.
5	1 5	12345@7890	Ошибка: ИНН может содержать только цифры.
6	2 5	abc	Ошибка: ИНН должен состоять из 10 цифр.
7	3 5	1234567890abc	Ошибка: ИНН должен состоять из 10 цифр.

В результате функционального тестирования были выделены классы эквивалентности для значения поля ввода ИНН, а также, на их основе, были сформированы тест-кейсы, показавшие корректность работы программы.

4.2 Анализ сложности кода модуля новостей

Для оценки сложности и качества кода был проведен анализ модуля новостей с использованием метрик Холстеда. В качестве объекта исследования был выбран класс бизнес-логики, так как он является самым

наполненным. Класс «ReportService» представлен в приложении А (листинг А.14).

Метрики Холстеда представляют собой формальные количественные оценки сложности программного кода, предложенные американским программистом Морисом Холстедом в 1977 году. Они основаны на подсчете количества операторов и операндов и позволяют оценить такие параметры, как объем кода, теоретическую сложность, время необходимое для разработки, а также потенциальное количество ошибок [19].

Среди метрик Холстеда можно выделить первостепенные, на основе которых вычисляются остальные метрики:

- 1) n_1 – число уникальных операторов программы, включая символы разделители, имена процедур и знаки операций (словарь операторов);
- 2) n_2 – число уникальных операндов программы (словарь операндов);
- 3) N_1 – общее число операторов в программе;
- 4) N_2 – общее число операндов в программе.

Оператор – это символ или зарезервированное слово, определяющее действие. Операнд – это объект, над которым выполняется действие. К числу операторов относятся, например, арифметические знаки, ключевые слова языка, вызовы функций, а к операндам – переменные, литералы и константы [19].

В таблице 5 приведены результаты расчета количества операторов и операндов для тестируемого блока кода.

Таблица 5 – Операторы и операнды класса «ReportService»

№ п/п	Операторы	Число операторов	Операнды	Число операндов
1	.	55	reportRepository	14
2	()	77	reportMapper	13
3	{}	13	fileStorageService	5
4	=	7	report	27
5	;	28	author	10
6	,	9	reportId	10
7	@Service	1	user	3
8	@RequiredArgsConstructor	1	dto	6

Продолжение таблицы 5

9	if	4	file	2
10	!	4	photoUrl	3
11	::	2	CreateReportDto	2
12	->	5	Report	6
13	throw	14	ReportStatus	6
14	equals()	5	BriefReportInfo	1
15	new	9	OrganizersReportDto	1
16	return	7	ReportInfo	4
17	stream()	2	User	1
18	map()	2	MultipartFile	1
19	toList()	2	«Новость не найдена»	5
20	findById()	5	«Новость была удалена»	1
21	save()	5	«Нет доступа к редактированию новости»	1
22	findPublishedAndOrderedByDate()	1	«Нет доступа к изменению фото»	2
23	findByAuthorId()	1	«/upload/reports/default.png»	1
24	createFromDto()	1	this	3
25	updateFromDto()	1	Exception	5
26	toInfoDto()	4		
27	toBriefInfoDto()	1		
28	toOrganizersReportDto()	1		
29	setPhotoUrl()	2		
30	saveReportPhoto()	1		

Ниже произведен расчет метрик Холстеда в соответствии с подсчитанным количеством операторов и операндов исследуемого класса.

Длина кода программы рассчитывается по формуле:

$$N = N_1 + N_2 = 403,$$

где N_1 – общее число операторов в программе;

N_2 – общее число операндов в программе.

Словарь программы рассчитывается по формуле:

$$n = n_1 + n_2 = 55,$$

где n_1 – число уникальных операторов программы;

n_2 – число уникальных операндов программы.

Потенциальный объем программы рассчитывается по формуле:

$$\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2 = 263.303,$$

где n_1 – число уникальных операторов программы;

n_2 – число уникальных операндов программы.

Объем кода программы рассчитывается по формуле:

$$V = N \log_2 n = 2329.88796,$$

где N – длина программы;

n – словарь программы.

Сложность кода программы рассчитывается по формуле:

$$D = \frac{n_1}{2} * \frac{N_2}{n_2} = 79.8,$$

где N_2 – общее число операндов в программе;

n_1 – число уникальных операторов программы;

n_2 – число уникальных операндов программы.

Усилие для написания и понимания кода рассчитывается по формуле:

$$E = D * V = 189526.0592,$$

Время написания кода рассчитывается по формуле:

$$T = \frac{E}{18} = 105345.23,$$

где D – усилие для написания и понимания кода;

V – объем кода программы.

Количество доставленных кодом ошибок рассчитывается по формуле:

$$B = \frac{V}{3000} = 0.77662932,$$

где V – объем кода программы.

В результате анализа сложности кода модуля новостей с использованием метрики Холстеда было выявлено, что исходный код класса «ReportService» соответствует необходимым требованиям по метрикам качества. Следовательно, реализацию программы можно считать успешной.

4.3 Автоматизированное тестирование модуля мероприятий

Автоматизированное тестирование является неотъемлемой частью процесса обеспечения качества программного обеспечения. Оно позволяет своевременно выявлять ошибки, сократить затраты на отладку и повысить надежность разрабатываемой системы. В рамках реализации web-портала для

информационной системы экологического волонтерского движения особое внимание было уделено тестированию контроллера модуля мероприятий.

В качестве инструмента для модульного тестирования использовались фреймворки JUnit 5 и Mockito, предоставляющие широкий набор аннотаций и средств для создания воспроизводимых тестов. Контроллер «EventController» тестировался в изоляции от остальной системы с помощью «@WebMvcTest», при этом зависимости подменялись через «@MockBean». Были реализованы тесты для основных операций (создание, редактирование, отмена мероприятия), обработки исключений, а также проверки недоступности операций без соответствующих ролей («403 Forbidden»). Это позволило покрыть как позитивные, так и негативные сценарии.

Автоматизация тестирования должна обеспечивать воспроизводимость проверок, контроль критериев завершения тестирования и регистрацию отклонений [20]. Использование фреймворков «JUnit» и «Mockito» полностью соответствует этим требованиям: все тест-кейсы определяются программно и сопровождаются логированием результатов. На рисунке 19 приведен результат успешного прохождения всех тестов в среде IntelliJ IDEA.

✓ EventControllerTest (com.eco_portal.events.controller)	608 ms
✓ testAccessDenied_noRole()	104 ms
✓ testDeleteEvent_error()	172 ms
✓ testUpdateEvent_exception()	114 ms
✓ testGetEvents_success()	79 ms
✓ testCancelEvent_error()	12 ms
✓ testGetOrganizerEvents_success()	15 ms
✓ testUploadPhoto_error()	25 ms
✓ testUploadPhoto_success()	11 ms
✓ testCancelEvent_success()	15 ms
✓ testGetOrganizerEvents_forbidden()	14 ms
✓ testGetEventById_notFound()	9 ms
✓ testDeleteEvent_success()	8 ms
✓ testGetCategories_success()	11 ms
✓ testUpdateEvent_success()	10 ms
✓ testGetEventById_success()	9 ms

Рисунок 19 – Результаты автоматизированного тестирования класса-контроллера модуля мероприятий

Таким образом, модульное тестирование показало корректность функционирования разработанного программного обеспечения для обработки HTTP-запросов в модуле мероприятий.

Заключение

В ходе выполнения проекта была разработана информационная система, предназначенная для поддержки деятельности экологического волонтерского движения. Система охватывает ключевые аспекты организации экологических мероприятий, включая планирование, координацию участников, информирование и отслеживание активности волонтеров. В процессе разработки был выполнен всесторонний анализ предметной области, исследованы существующие программные аналоги, определены функциональные требования, разработаны диаграммы UML, спроектированы архитектура и интерфейс системы, проведено планирование бюджета, реализован программный продукт и выполнено его тестирование.

Реализация системы позволила значительно автоматизировать процессы, связанные с организацией волонтерских мероприятий и управления новостным контентом, что, в свою очередь, способствует снижению организационной нагрузки, повышению эффективности взаимодействия между участниками и укреплению сообщества. Кроме того, система обеспечивает прозрачность процессов, способствует формированию экологической культуры и стимулирует осознанное участие граждан в решении актуальных экологических проблем.

Поставленная цель – создание эффективного цифрового инструмента для поддержки экологического волонтерства – была достигнута. Разработанная информационная система может быть успешно применена в деятельности общественных организаций и инициативных групп, заинтересованных в устойчивом развитии и охране окружающей среды. Перспективы дальнейшего развития включают расширение функциональности, интеграцию с мобильными платформами и реализацию механизмов аналитики данных для более глубокой оценки вклада волонтеров и оптимизации экологических инициатив.

Список использованных источников

1. Главная страница // Добро.рф [Электронный ресурс]. URL: <https://dobro.ru/> (дата обращения: 15.05.2025).
2. Главная страница // Движение ЭКА [Электронный ресурс]. URL: <https://ecamir.ru/> (дата обращения: 15.05.2025).
3. Главная страница // Мосволонтер [Электронный ресурс]. URL: <https://mosvolonter.ru/> (дата обращения: 15.05.2025).
4. Современные информационные технологии в проектировании программных систем и комплексов: учеб. пособие / И. Ю. Балашова, Д. В. Такташкин; под ред. П. П. Макарычева. – Пенза : Изд-во ПГУ, 2019. – 106 с.
5. Разработка требований к программному обеспечению. 3-е изд., дополненное / К. Вигерс, Д. Битти; Пер. с англ. – М. : Издательство «Русская редакция» ; СПб. : БХВ-Петербург, 2014. – 736 с.
6. Подойдет ли PostgreSQL вообще всем проектам или нужны альтернативы // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/selectel/articles/740680> (дата обращения 22.05.2025)
7. Что такое клиент-серверная архитектура? Все, что вы должны знать // Simplilearn [Электронный ресурс]. URL: <https://www.simplilearn.com/what-is-client-server-architecture-article> (дата обращения 22.05.2025)
8. Введение в Spring Boot: создание простого REST API на Java // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/435144/> (дата обращения 22.05.2025)
9. React. JavaScript-библиотека для создания пользовательских интерфейсов // React [Электронный ресурс]. URL: <https://ru.legacy.reactjs.org/> (дата обращения 23.05.2025)
10. Архитектура Приложений: Monolith, Modular Monolith, Microservices // Alishoff [Электронный ресурс]. URL:

<https://alishoff.com/blog/483> (дата обращения 23.05.2025)

11. Что такое UX/UI дизайн на самом деле? // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/321312/> (дата обращения: 24.05.2025).

12. Сухоруков А. А. Основы управления проектами в ИТ. Учебное пособие. – М.: КноРус, 2018. – 128 с.

13. Паттерн MVC: что это и зачем он нужен // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/post/215605/> (дата обращения: 25.05.2025)

14. Сессии, JWT, Access, Refresh // Хабр [Электронный ресурс]. URL: <https://vc.ru/dev/1780474-sessii-jwt-access-refresh/> (дата обращения: 26.05.2025)

15. Фленов М. Ю. Java. Библиотека профессионала. Том 2: Технологии Java EE. – СПб.: Питер, 2020. – 800 с.

16. Axios: документация // Справочник JavaScript [Электронный ресурс]. URL: <https://learn.javascript.ru/axios> (дата обращения: 28.05.2025)

17. Игорь Шишкин. Основы JSX // Metanit [Электронный ресурс]. URL: <https://metanit.com/web/react/1.3.php> (дата обращения: 28.05.2025)

18. Классы эквивалентности в тестировании // TQuality.ru [Электронный ресурс]. URL: <https://tquality.ru/blog/klassy-ekvivalentnosti-v-testirovanii> (дата обращения: 30.05.2025)

19. Halstead, M. H. Elements of Software Science. – New York: Elsevier North-Holland, 1977. – 127 p. (русский перевод: Холстед М. Основы программной науки. – М.: Мир, 1980. – 160 с.)

20. ГОСТ 28195–89. Обеспечение качества программных средств. Общие положения. – М.: Госстандарт, 1989

Приложение А
(обязательное)

Исходный код программного обеспечения

Листинг А.1 – Файл EcoPortalApplication.java

```
package com.eco_portal;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EcoPortalApplication {

    public static void main(String[] args) {
        SpringApplication.run(EcoPortalApplication.class,
args);
    }

}
```

Листинг А.2 – Файл pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.8</version>
        <relativePath/>
    </parent>
    <groupId>com</groupId>
    <artifactId>eco-portal</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>eco-portal</name>
    <description>description</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
```

```

    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-
jpa</artifactId>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
security</artifactId>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
      </dependency>

      <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
      </dependency>
      <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.32</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
      </dependency>
      <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
      </dependency>
      <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-api</artifactId>
        <version>0.11.5</version>
      </dependency>
      <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-impl</artifactId>
        <version>0.11.5</version>
        <scope>runtime</scope>
      </dependency>
      <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-jackson</artifactId>
        <version>0.11.5</version>
        <scope>runtime</scope>

```

```

        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-
plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                    <annotationProcessorPaths>
                        <path>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>
                        <version>1.18.32</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>

```

Листинг А.3 – Файл application.yml

```

spring:
  application:
    name: eco-portal
  datasource:
    url: jdbc:postgresql://localhost:5432/eco_portal_db
    username: postgres
    password: postgres
    driver-class-name: org.postgresql.Driver
  jpa:
    database-platform: org.hibernate.dialect.PostgreSQLDialect
  web:
    resources:
      static-locations: file:../img/
  servlet:
    multipart:
      max-file-size: 15MB
      max-request-size: 15MB

server:
  port: 8081

jwt:

```

```

    access:
      secret:
qBTmv4oXFFR2GwjexDJ4t6fsIUIUhhXqlktXjXdkcyygs8nPVEwMfo29VDRRepYD
VV5IkIxBMzr70EHXEhd37w==
      expiration: 300000          # 5 minutes
    refresh:
      secret:
zL1HB3Pch05Avfynovxrf/kpF9O2m4NCWKJUjEp27s9J2jEG3ifiKCGylaZ8fDeo
ONSTJP/wAzKawB8F9rOMNg==
      expiration: 2592000000     # 30 days

```

Листинг А.4 – Файл SecurityConfig.java

```

package com.eco_portal.shared.security;

import com.eco_portal.users.repository.UserRepository;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.authentication.AuthenticationManage
r;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.authentication.co
nfiguration.AuthenticationConfiguration;
import
org.springframework.security.config.annotation.method.configurat
ion.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.Http
Security;
import
org.springframework.security.config.annotation.web.configuration
.WebSecurityCustomizer;
import
org.springframework.security.core.userdetails.UserDetailsService
;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePassword
AuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import
org.springframework.web.cors.UrlBasedCorsConfigurationSource;

@Configuration

```

```

@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    private final UserRepository userRepository;
    private final JwtAuthFilter jwtAuthFilter;

    @Bean
    public UserDetailsService userDetailsService() {
        return email -> userRepository.findByEmail(email)
            .orElseThrow(() -> new RuntimeException("User
not found"));
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(
        AuthenticationConfiguration config) throws Exception
    {
        return config.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
throws Exception {
        http.csrf(csrf -> csrf.disable())
            .cors(Customizer.withDefaults())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(
                    "/api/auth/**"
                ).permitAll()
                .requestMatchers(HttpMethod.GET,
                    "/api/reports",
                    "/api/reports/*",
                    "/api/reports/*/comments",
                    "/api/events",
                    "/api/events/*",
                    "/upload/**",
                    "/api/shared/regions",
                    "/api/events/categories"
                ).permitAll()

            .requestMatchers(HttpMethod.OPTIONS, "/*").permitAll()
                .anyRequest().authenticated()
            )
            .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class)
            .exceptionHandling(e -> e

```

```

        .authenticationEntryPoint((request,
response, authException) -> {

response.setStatus(HttpServletResponse.SC_UNAUTHORIZED); // 401

response.setContentType("application/json");

response.getWriter().write("{\"message\": \"Access token
expired\"}");

        })
        .accessDeniedHandler((request, response,
accessDeniedException) -> {

response.setStatus(HttpServletResponse.SC_FORBIDDEN); // 403

response.setContentType("application/json");

response.getWriter().write("{\"message\": \"Access denied\"}");

        })
    );

    return http.build();
}

@Bean
public WebSecurityCustomizer webSecurityCustomizer() {
    return web ->
web.ignoring().requestMatchers("/upload/**");
}

@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration config = new CorsConfiguration();
    config.addAllowedOrigin("http://localhost:3000"); //
frontend адрес
    config.addAllowedMethod("*");
    config.addAllowedHeader("*");
    config.setAllowCredentials(true); // если используешь
авторизацию (JWT)

    UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", config);
    return source;
}
}

```

Листинг А.5 – Файл JwtAuthFilter.java

```

package com.eco_portal.shared.security;

import com.eco_portal.users.repository.UserRepository;
import com.eco_portal.users.model.User;

```

```

import com.eco_portal.shared.service.JwtService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.authentication.UsernamePasswordAuth
enticationToken;
import
org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.web.authentication.WebAuthenticatio
nDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtAuthFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserRepository userRepository;

    @Override
    protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain
    ) throws ServletException, IOException {

        final String authHeader =
request.getHeader("Authorization");

        if (authHeader == null || !authHeader.startsWith("Bearer
")) {
            filterChain.doFilter(request, response);
            return;
        }

        String jwt = authHeader.substring(7);

        if (!jwtService.isAccessTokenValid(jwt)) {
            filterChain.doFilter(request, response);
            return;
        }

        if
(SecurityContextHolder.getContext().getAuthentication() == null)
{

```

```

        String email =
jwtService.extractAccessClaims(jwt).getSubject();
        User user =
userRepository.findByEmail(email).orElse(null);
        if (user != null) {
            UsernamePasswordAuthenticationToken authToken =
                new
UsernamePasswordAuthenticationToken(user, null,
user.getAuthorities());
            authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);
        }
    }

    filterChain.doFilter(request, response);
}
}

```

Листинг А.6 – Файл JwtService.java

```

package com.eco_portal.shared.service;

import com.eco_portal.users.model.User;
import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import javax.crypto.SecretKey;
import java.util.Date;

@Service
public class JwtService {

    @Value("${jwt.access.secret}")
    private String jwtAccessSecret;

    @Value("${jwt.refresh.secret}")
    private String jwtRefreshSecret;

    @Value("${jwt.access.expiration}")
    private long accessExpiration;

    @Value("${jwt.refresh.expiration}")
    private long refreshExpiration;

    private SecretKey accessKey;
    private SecretKey refreshKey;
}

```

```

@PostConstruct
public void init() {
    this.accessKey =
Keys.hmacShaKeyFor(jwtAccessSecret.getBytes());
    this.refreshKey =
Keys.hmacShaKeyFor(jwtRefreshSecret.getBytes());
}

public String generateAccessToken(User user) {
    return Jwts.builder()
        .setSubject(user.getEmail())
        .claim("role", user.getRole().name())
        .setIssuedAt(new Date())
        .setExpiration(new
Date(System.currentTimeMillis() + accessExpiration))
        .signWith(accessKey, SignatureAlgorithm.HS256)
        .compact();
}

public String generateRefreshToken(User user) {
    return Jwts.builder()
        .setSubject(user.getEmail())
        .setIssuedAt(new Date())
        .setExpiration(new
Date(System.currentTimeMillis() + refreshExpiration))
        .signWith(refreshKey, SignatureAlgorithm.HS256)
        .compact();
}

public boolean isAccessTokenValid(String token) {
    return isValidToken(token, accessKey);
}

public boolean isRefreshTokenValid(String token) {
    return isValidToken(token, refreshKey);
}

public boolean isValidToken(String token, SecretKey
secretKey) {
    try {
        Jwts.parserBuilder()
            .setSigningKey(secretKey)
            .build()
            .parseClaimsJws(token);
        return true;
    } catch (ExpiredJwtException expEx) {
        System.out.println("Token expired " + expEx);
    } catch (UnsupportedJwtException unsEx) {
        System.out.println("Unsupported jwt " + unsEx);
    } catch (MalformedJwtException mjEx) {
        System.out.println("Malformed jwt " + mjEx);
    } catch (SignatureException sigEx) {
        System.out.println("Invalid signature " + sigEx);
    }
}

```

```

        } catch (Exception e) {
            System.out.println("Invalid token " + e);
        }
        return false;
    }

    public Claims extractAccessClaims(String token) {
        return extractClaims(token, accessKey);
    }

    public Claims extractRefreshClaims(String token) {
        return extractClaims(token, refreshKey);
    }

    private Claims extractClaims(String token, SecretKey key) {
        return Jwts.parserBuilder()
            .setSigningKey(key)
            .build()
            .parseClaimsJws(token)
            .getBody();
    }
}

```

Листинг А.7 – Файл FileStorageService.java

```

package com.eco_portal.shared.service;

import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.io.UncheckedIOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Optional;
import java.util.stream.Stream;

@Service
public class FileStorageService {

    private static final Path BASE_DIR =
        Paths.get("../img/upload").toAbsolutePath().normalize();

    // ===== USER =====

    public String saveUserAvatar(Long userId, MultipartFile
file) throws IOException {
        return saveFile("users", userId, "avatar", file);
    }
}

```

```

    public void deleteUserAvatar(Long userId) throws IOException
    {
        deleteFile("users", userId, "avatar");
    }

    // ===== EVENT =====

    public String saveEventPhoto(Long eventId, MultipartFile
file) throws IOException {
        return saveFile("events", eventId, "eventPhoto", file);
    }

    public void deleteEventPhoto(Long eventId) throws
IOException {
        deleteFile("events", eventId, "eventPhoto");
    }

    // ===== REPORT =====

    public String saveReportPhoto(Long reportId, MultipartFile
file) throws IOException {
        return saveFile("reports", reportId, "reportPhoto",
file);
    }

    public void deleteReportPhoto(Long reportId) throws
IOException {
        deleteFile("reports", reportId, "reportPhoto");
    }

    // ===== INTERNAL =====

    private String saveFile(String type, Long id, String
baseFileName, MultipartFile file) throws IOException {
        String extension =
Optional.ofNullable(file.getOriginalFilename())
        .filter(f -> f.contains("."))
        .map(f ->
f.substring(file.getOriginalFilename().lastIndexOf('.')))
        .orElse(".png");

        Path targetDir = BASE_DIR.resolve(type + "/" + id);
        Files.createDirectories(targetDir);

        String fileName = baseFileName + extension;
        Path filePath = targetDir.resolve(fileName);
        file.transferTo(filePath.toFile());

        return "/upload/" + type + "/" + id + "/" + fileName;
    }

    private void deleteFile(String type, Long id, String
baseFileName) throws IOException {

```

```

        Path targetDir = BASE_DIR.resolve(type + "/" + id);
        if (Files.exists(targetDir)) {
            try (Stream<Path> files = Files.list(targetDir)) {
                files.filter(p ->
                    p.getFileName().toString().startsWith(baseFileName))
                    .forEach(p -> {
                        try {
                            Files.deleteIfExists(p);
                        } catch (IOException e) {
                            throw new
UncheckedIOException(e);
                        }
                    });
            }
        }
    }
}

```

Листинг А.8 – Файл AuthService.java

```

package com.eco_portal.users.service;

import com.eco_portal.shared.service.JwtService;
import com.eco_portal.users.dto.RefreshAccessTokenRequest;
import com.eco_portal.users.dto.RegisterRequest;
import com.eco_portal.shared.model.Region;
import com.eco_portal.users.model.Role;
import com.eco_portal.users.model.User;
import com.eco_portal.users.repository.RegionRepository;
import com.eco_portal.users.repository.UserRepository;
import com.eco_portal.users.dto.LoginRequest;
import com.eco_portal.users.dto.AuthResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;

@Service
@RequiredArgsConstructor
public class AuthService {

    private final UserRepository userRepository;
    private final RegionRepository regionRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;

```

```

        private final AuthenticationManager authenticationManager;

        public AuthResponse login(LoginRequest request) throws
        Exception {
            authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                    request.getEmail(),
                    request.getPassword()
                )
            );

            User user =
            userRepository.findByEmail(request.getEmail())
                .orElseThrow(() -> new Exception("Пользователь
            не найден"));

            String accessToken =
            jwtService.generateAccessToken(user);
            String refreshToken =
            jwtService.generateRefreshToken(user);

            return new AuthResponse(accessToken, refreshToken);
        }

        public AuthResponse register(RegisterRequest request) throws
        Exception {
            if
            (userRepository.findByEmail(request.getEmail()).isPresent()) {
                throw new Exception("Email уже занят");
            }

            Region region =
            regionRepository.findById(request.getRegionId())
                .orElseThrow(() -> new Exception("Регион не
            найден"));

            User user = User.builder()
                .email(request.getEmail())

                .password(passwordEncoder.encode(request.getPassword()))
                .fullName(request.getFullName())
                .taxpayerNumber(null)
                .region(region)
                .role(Role.VOLUNTEER)
                .isLocked(false)
                .registeredAt(LocalDateDateTime.now())
                .avatarUrl("/upload/users/default.png")
                .build();

            userRepository.save(user);

            String accessToken =
            jwtService.generateAccessToken(user);

```

```

        String refreshToken =
jwtService.generateRefreshToken(user);

        return new AuthResponse(accessToken, refreshToken);
    }

    public AuthResponse
refreshAccessToken(RefreshAccessTokenRequest request) throws
Exception {
        String refreshToken = request.getRefreshToken();
        if (!jwtService.isRefreshTokenValid(refreshToken)) {
            throw new Exception("Некорректный refresh token");
        }

        String email =
jwtService.extractRefreshClaims(refreshToken).getSubject();
        User user = userRepository.findByEmail(email)
            .orElseThrow(() -> new Exception("Пользователь
не найден"));
        String newAccessToken =
jwtService.generateAccessToken(user);

        return new AuthResponse(newAccessToken, refreshToken);
    }
}

```

Листинг А.9 – Файл AuthController.java

```

package com.eco_portal.users.controller;

import com.eco_portal.users.dto.LoginRequest;
import com.eco_portal.users.dto.RefreshAccessTokenRequest;
import com.eco_portal.users.dto.RegisterRequest;
import com.eco_portal.users.service.AuthService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {

    private final AuthService authService;

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody
RegisterRequest request) {
        try {
            return
ResponseEntity.ok(authService.register(request));
        } catch (Exception e) {

```

```

        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest
request) {
    try {
        return
ResponseEntity.ok(authService.login(request));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@PostMapping("/refresh-token")
public ResponseEntity<?> refreshAccessToken(@RequestBody
RefreshAccessTokenRequest request) {
    try {
        return
ResponseEntity.ok(authService.refreshAccessToken(request));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}
}

```

Листинг А.10 – Файл UserService.java

```

package com.eco_portal.users.service;

import com.eco_portal.notifications.service.NotificationService;
import com.eco_portal.shared.model.Region;
import com.eco_portal.shared.service.FileStorageService;
import com.eco_portal.users.dto.*;
import com.eco_portal.users.mapper.OrganizerRequestMapper;
import com.eco_portal.users.mapper.UserMapper;
import com.eco_portal.users.model.*;
import
com.eco_portal.users.repository.OrganizerRequestRepository;
import com.eco_portal.users.repository.RegionRepository;
import com.eco_portal.users.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

```

```

import java.util.List;

@Service
@RequiredArgsConstructor
public class UserService {

    private final OrganizerRequestRepository
organizerRequestRepository;
    private final OrganizerRequestMapper organizerRequestMapper;
    private final UserMapper userMapper;
    private final UserRepository userRepository;
    private final RegionRepository regionRepository;
    private final NotificationService notificationService;
    private final FileStorageService fileStorageService;

    public UserProfile getProfile(User user) {
        return userMapper.toUserProfile(user);
    }

    public UserProfile changeProfile(User user,
ChangeProfileRequest request) throws Exception {
        Region region =
regionRepository.findById(request.getRegionId())
                .orElseThrow(() -> new Exception("Регион не
найден"));
        user.setFullName(request.getFullName());
        user.setRegion(region);
        return
userMapper.toUserProfile(userRepository.save(user));
    }

    public void sendOrganizerRequest(User user,
CreateOrgRequestDto dto) {

organizerRequestRepository.save(organizerRequestMapper.createFro
mDto(user, dto));
    }

    public List<UserData> getAllUsers() {
        return userRepository.findAll().stream()
                .map(userMapper::toData)
                .toList();
    }

    public List<OrgRequestDto> getPendingOrganizerRequests() {
        return
organizerRequestRepository.findByStatus(OrgRequestStatus.PENDING
).stream()
                .map(organizerRequestMapper::toDto)
                .toList();
    }
}

```

```

    public void approveOrganizerRequest(Long requestId) throws
Exception {
        OrganizerRequest request =
organizerRequestRepository.findById(requestId)
                                .orElseThrow(() -> new Exception("Заявка не
найдена"));

        User user = request.getUser();
        user.setRole(Role.ORGANIZER);
        userRepository.save(user);

        request.setStatus(OrgRequestStatus.APPROVED);
        organizerRequestRepository.save(request);

        String notificationContent = "Ваша заявка от " +
            request.getSubmittedAt().toLocalDate() +
            " на регистрацию организации '" +
            request.getOrganizerName() +
            "' подтверждена!";
        notificationService.sendNotification(user,
notificationContent);
    }

    public void rejectOrganizerRequest(Long requestId) throws
Exception {
        OrganizerRequest request =
organizerRequestRepository.findById(requestId)
                                .orElseThrow(() -> new Exception("Заявка не
найдена"));

        request.setStatus(OrgRequestStatus.REJECTED);
        organizerRequestRepository.save(request);

        String notificationContent = "Ваша заявка от " +
            request.getSubmittedAt().toLocalDate() +
            " на регистрацию организации '" +
            request.getOrganizerName() +
            "' отклонена!";
        notificationService.sendNotification(request.getUser(),
notificationContent);
    }

    public void lockUser(Long userId) throws Exception {
        User user = userRepository.findById(userId)
                                .orElseThrow(() -> new Exception("Пользователь
не найден"));

        if (user.isLocked()) {
            throw new Exception("Пользователь уже
заблокирован");
        }

        if (user.getRole().equals(Role.ADMIN)) {

```

```

        throw new Exception("Невозможно заблокировать
Администратора");
    }

    user.setLocked(true);
    userRepository.save(user);
}

public void unlockUser(Long userId) throws Exception {
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new Exception("Пользователь
не найден"));

    if (!user.isLocked()) {
        throw new Exception("Пользователь не является
заблокированным");
    }

    user.setLocked(false);
    userRepository.save(user);
}

public String uploadAvatar(User user, MultipartFile file)
throws IOException {
    String avatarUrl =
fileStorageService.saveUserAvatar(user.getId(), file);
    user.setAvatarUrl(avatarUrl);
    userRepository.save(user);
    return avatarUrl;
}

public void deleteAvatar(User user) throws IOException {
    fileStorageService.deleteUserAvatar(user.getId());
    user.setAvatarUrl("/upload/users/default.png");
    userRepository.save(user);
}
}

```

Листинг A.11 – Файл UserController.java

```

package com.eco_portal.users.controller;

import com.eco_portal.users.dto.*;
import com.eco_portal.users.model.User;
import com.eco_portal.users.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;

```

```

import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.List;

@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    @GetMapping("/profile")
    public ResponseEntity<UserProfile>
    getProfile(@AuthenticationPrincipal User user) {
        return ResponseEntity.ok(userService.getProfile(user));
    }

    @PutMapping("/profile")
    public ResponseEntity<?>
    changeProfile(@AuthenticationPrincipal User user, @RequestBody
    ChangeProfileRequest request) {
        try {
            return
            ResponseEntity.ok(userService.changeProfile(user, request));
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
        }
    }

    @PostMapping("/organizer-requests")
    @PreAuthorize("hasRole('VOLUNTEER')")
    public ResponseEntity<?>
    sendOrganizerRequest(@AuthenticationPrincipal User user,
    @RequestBody CreateOrgRequestDto dto) {
        userService.sendOrganizerRequest(user, dto);
        return ResponseEntity.ok().build();
    }

    @GetMapping
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<List<UserData>> getAllUsers() {
        return ResponseEntity.ok(userService.getAllUsers());
    }

    @GetMapping("/organizer-requests")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<List<OrgRequestDto>>
    getPendingOrganizerRequests() {

```

```

        return
        ResponseEntity.ok(userService.getPendingOrganizerRequests());
    }

    @PutMapping("/organizer-requests/{requestId}/approve")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<?>
    approveOrganizerRequest(@PathVariable Long requestId) {
        try {
            userService.approveOrganizerRequest(requestId);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.NOT_FOUND)
                .body(e.getMessage());
        }
    }

    @PutMapping("/organizer-requests/{requestId}/reject")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<?>
    rejectOrganizerRequest(@PathVariable Long requestId) {
        try {
            userService.rejectOrganizerRequest(requestId);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.NOT_FOUND)
                .body(e.getMessage());
        }
    }

    @PutMapping("/{userId}/lock")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<?> lockUser(@PathVariable Long userId)
    {
        try {
            userService.lockUser(userId);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.NOT_FOUND)
                .body(e.getMessage());
        }
    }

    @PutMapping("/{userId}/unlock")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<?> unlockUser(@PathVariable Long
    userId) {
        try {
            userService.unlockUser(userId);
            return ResponseEntity.ok().build();
        }
    }

```

```

        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.NOT_FOUND)
                .body(e.getMessage());
        }
    }

    @PostMapping("/profile/avatar")
    @PreAuthorize("hasAnyRole('VOLUNTEER', 'ORGANIZER',
'ADMIN')")
    public ResponseEntity<String> uploadAvatar(
        @AuthenticationPrincipal User user,
        @RequestParam("file") MultipartFile file
    ) throws IOException {
        String avatarUrl = userService.uploadAvatar(user, file);
        return ResponseEntity.ok(avatarUrl);
    }

    @DeleteMapping("/profile/avatar")
    @PreAuthorize("hasAnyRole('VOLUNTEER', 'ORGANIZER',
'ADMIN')")
    public ResponseEntity<Void>
deleteAvatar(@AuthenticationPrincipal User user) throws
IOException {
        userService.deleteAvatar(user);
        return ResponseEntity.ok().build();
    }
}

```

Листинг А.12 – Файл EventService.java

```

package com.eco_portal.events.service;

import com.eco_portal.achievements.service.AchievementService;
import com.eco_portal.events.dto.BriefEventInfo;
import com.eco_portal.events.dto.CreateEventDto;
import com.eco_portal.events.dto.EventInfo;
import com.eco_portal.events.dto.OrganizersEventDto;
import com.eco_portal.events.mapper.EventMapper;
import com.eco_portal.events.model.Category;
import com.eco_portal.events.model.Event;
import com.eco_portal.events.model.EventStatus;
import com.eco_portal.events.repository.CategoryRepository;
import com.eco_portal.events.repository.EventRepository;
import com.eco_portal.notifications.service.NotificationService;
import
com.eco_portal.participation_requests.model.ParticipationRequest
;
import
com.eco_portal.participation_requests.model.RequestStatus;

```

```

import
com.eco_portal.participation_requests.repository.ParticipationRe
questRepository;
import com.eco_portal.shared.service.FileStorageService;
import com.eco_portal.shared.model.Region;
import com.eco_portal.users.model.User;
import com.eco_portal.users.repository.RegionRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class EventService {

    private final EventRepository eventRepository;
    private final RegionRepository regionRepository;
    private final CategoryRepository categoryRepository;
    private final EventMapper eventMapper;
    private final ParticipationRequestRepository
participationRequestRepository;
    private final NotificationService notificationService;
    private final AchievementService achievementService;
    private final FileStorageService fileStorageService;

    public List<BriefEventInfo> getEvents(Long regionId,
LocalDate eventDate, Long categoryId) {
        List<Event> events;

        if (eventDate != null) {
            LocalDateTime start = eventDate.atStartOfDay();
            LocalDateTime end = eventDate.atTime(LocalTime.MAX);
            events =
eventRepository.findFilteredWithDate(regionId, start, end,
categoryId);
        } else {
            events =
eventRepository.findFilteredWithoutDate(regionId, categoryId);
        }
        return events.stream()
            .map(eventMapper::toBriefDto)
            .toList();
    }

    public List<OrganizersEventDto> getOrganizerEvents(User
user) {

```

```

        return
eventRepository.findById(user.getId()).stream()
            .map(eventMapper::toOrganizersEventDto)
            .toList();
    }

    public EventInfo getEventById(Long eventId) throws Exception
    {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Мероприятие не
найдено"));
        if (event.getStatus().equals(EventStatus.DELETED)) {
            throw new Exception("Мероприятие было удалено");
        }
        return eventMapper.toInfoDto(event);
    }

    public EventInfo createEvent(User organizer, CreateEventDto
dto) throws Exception {
        Category category =
categoryRepository.findById(dto.getCategoryId())
            .orElseThrow(() -> new Exception("Категория не
найдена"));
        Region region =
regionRepository.findById(dto.getRegionId())
            .orElseThrow(() -> new Exception("Регион не
найден"));

        Event event = eventMapper.createFromDto(dto, organizer,
category, region);
        return
eventMapper.toInfoDto(eventRepository.save(event));
    }

    public EventInfo updateEvent(User organizer, Long eventId,
CreateEventDto dto) throws Exception {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Мероприятие не
найдено"));
        Category category =
categoryRepository.findById(dto.getCategoryId())
            .orElseThrow(() -> new Exception("Категория не
найдена"));
        Region region =
regionRepository.findById(dto.getRegionId())
            .orElseThrow(() -> new Exception("Регион не
найден"));

        if
(!organizer.getId().equals(event.getOrganizer().getId()) ||
!event.getStatus().equals(EventStatus.PUBLISHED)) {

```

```

        throw new Exception("Нет доступа к редактированию
мероприятия");
    }

    String notificationContent = "Мероприятие '" +
event.getTitle() + "' было изменено";
    sendNotificationsToParticipants(eventId,
notificationContent);

    eventMapper.updateFromDto(event, dto, category, region);
    return
eventMapper.toInfoDto(eventRepository.save(event));
}

    public void sendNotificationsToParticipants(Long eventId,
String notificationContent) {
        List<User> participants = participationRequestRepository
            .findByEventIdAndStatuses(
                eventId,
                List.of(RequestStatus.APPROVED)
            )
            .stream()
            .map(ParticipationRequest::getUser)
            .toList();
        for (User p : participants) {
            notificationService.sendNotification(p,
notificationContent);
        }
    }

    public EventInfo cancelEvent(User organizer, Long eventId)
throws Exception {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Мероприятие не
найденно"));

        if
(!organizer.getId().equals(event.getOrganizer().getId()) ||
            event.getStatus().equals(EventStatus.DELETED) ||
            event.getStatus().equals(EventStatus.FINISHED))
        {
            throw new Exception("Мероприятие невозможно
отменить");
        }
        event.setStatus(EventStatus.CANCELLED);

        String notificationContent = "Мероприятие '" +
event.getTitle() + "' было отменено";
        sendNotificationsToParticipants(eventId,
notificationContent);

        return
eventMapper.toInfoDto(eventRepository.save(event));
    }

```

```

    }

    public void deleteEvent(Long eventId) throws Exception {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Мероприятие не
найдено"));

        if (!event.getStatus().equals(EventStatus.PUBLISHED)) {
            throw new Exception("Мероприятие невозможно
удалить");
        }
        event.setStatus(EventStatus.DELETED);

        String notificationContent = "Мероприятие '" +
event.getTitle() + "' было отменено";
        sendNotificationsToParticipants(eventId,
notificationContent);

        eventRepository.save(event);
    }

    public EventInfo finishEvent(User organizer, Long eventId)
throws Exception {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Мероприятие не
найдено"));

        if
(!organizer.getId().equals(event.getOrganizer().getId()) ||
!event.getStatus().equals(EventStatus.PUBLISHED)) {
            throw new Exception("Мероприятие невозможно
завершить");
        }
        if (LocalDateTime.now().isBefore(event.getDate())) {
            throw new Exception("Мероприятие невозможно
завершить, так как оно еще не началось");
        }
        event.setStatus(EventStatus.FINISHED);

        eventRepository.save(event);
        achievementService.checkAndAwardByFinishedEvent(event);

        return eventMapper.toInfoDto(event);
    }

    public String uploadEventPhoto(User user, Long eventId,
MultipartFile file) throws Exception, IOException {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Мероприятие не
найдено"));

```

```

        if (!user.getId().equals(event.getOrganizer().getId()))
        {
            throw new Exception("Нет доступа к изменению фото");
        }

        String photoUrl =
fileStorageService.saveEventPhoto(eventId, file);
        event.setPhotoUrl(photoUrl);
        eventRepository.save(event);
        return photoUrl;
    }

    public void deleteEventPhoto(User user, Long eventId) throws
Exception, IOException {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Мероприятие не
найденно"));

        if (!user.getId().equals(event.getOrganizer().getId()))
        {
            throw new Exception("Нет доступа к изменению фото");
        }

        fileStorageService.deleteEventPhoto(eventId);
        event.setPhotoUrl("/upload/events/default.png");
        eventRepository.save(event);
    }
}

```

Листинг A.13 – Файл EventController.java

```

package com.eco_portal.events.controller;

import com.eco_portal.events.dto.BriefEventInfo;
import com.eco_portal.events.dto.CreateEventDto;
import com.eco_portal.events.dto.OrganizersEventDto;
import com.eco_portal.events.model.Category;
import com.eco_portal.events.service.CategoryService;
import com.eco_portal.events.service.EventService;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.time.LocalDate;

```

```

import java.util.List;

@RestController
@RequestMapping("/api/events")
@RequiredArgsConstructor
public class EventController {

    private final EventService eventService;
    private final CategoryService categoryService;

    @GetMapping("/categories")
    public ResponseEntity<List<Category>> getCategories() {
        return
ResponseEntity.ok(categoryService.getCategories());
    }

    @GetMapping
    public ResponseEntity<List<BriefEventInfo>> getEvents(
        @RequestParam(required = false) Long regionId,
        @RequestParam(required = false) @DateTimeFormat(iso
= DateTimeFormat.ISO.DATE) LocalDate eventDate,
        @RequestParam(required = false) Long categoryId
    ) {
        return
ResponseEntity.ok(eventService.getEvents(regionId, eventDate,
categoryId));
    }

    @GetMapping("/my")
    @PreAuthorize("hasRole('ORGANIZER')")
    public ResponseEntity<List<OrganizersEventDto>>
getOrganizerEvents(@AuthenticationPrincipal User user) {
        return
ResponseEntity.ok(eventService.getOrganizerEvents(user));
    }

    @GetMapping("/{eventId}")
    public ResponseEntity<?> getEventById(@PathVariable Long
eventId) {
        try {
            return
ResponseEntity.ok(eventService.getEventById(eventId));
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
        }
    }

    @PostMapping
    @PreAuthorize("hasRole('ORGANIZER')")

```

```

        public ResponseEntity<?>
createEvent(@AuthenticationPrincipal User user, @RequestBody
CreateEventDto dto) {
    try {
        return
ResponseEntity.ok(eventService.createEvent(user, dto));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@PutMapping("/{eventId}")
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<?>
updateEvent(@AuthenticationPrincipal User user, @PathVariable
Long eventId, @RequestBody CreateEventDto dto) {
    try {
        return
ResponseEntity.ok(eventService.updateEvent(user, eventId, dto));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@PutMapping("/{eventId}/cancel")
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<?>
cancelEvent(@AuthenticationPrincipal User user, @PathVariable
Long eventId) {
    try {
        return
ResponseEntity.ok(eventService.cancelEvent(user, eventId));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@PutMapping("/{eventId}/delete")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<?> deleteEvent(@PathVariable Long
eventId) {
    try {
        eventService.deleteEvent(eventId);
        return ResponseEntity.noContent().build();
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)

```

```

        .body(e.getMessage());
    }
}

@PutMapping("/{eventId}/finish")
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<?>
finishEvent(@AuthenticationPrincipal User user, @PathVariable
Long eventId) {
    try {
        return
ResponseEntity.ok(eventService.finishEvent(user, eventId));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@PostMapping("/{eventId}/photo")
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<?> uploadEventPhoto(
    @AuthenticationPrincipal User user,
    @PathVariable Long eventId,
    @RequestParam("file") MultipartFile file
) {
    try {
        String photoUrl =
eventService.uploadEventPhoto(user, eventId, file);
        return ResponseEntity.ok(photoUrl);
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@DeleteMapping("/{eventId}/photo")
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<?> deleteEventPhoto(
    @AuthenticationPrincipal User user,
    @PathVariable Long eventId
) {
    try {
        eventService.deleteEventPhoto(user, eventId);
        return ResponseEntity.noContent().build();
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}
}

```

Листинг А.14 – Файл ReportService.java

```
package com.eco_portal.reports.service;

import com.eco_portal.reports.dto.BriefReportInfo;
import com.eco_portal.reports.dto.CreateReportDto;
import com.eco_portal.reports.dto.OrganizersReportDto;
import com.eco_portal.reports.dto.ReportInfo;
import com.eco_portal.reports.mapper.ReportMapper;
import com.eco_portal.reports.model.Report;
import com.eco_portal.reports.model.ReportStatus;
import com.eco_portal.reports.repository.ReportRepository;
import com.eco_portal.shared.service.FileStorageService;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

@Service
@RequiredArgsConstructor
public class ReportService {

    private final ReportRepository reportRepository;
    private final ReportMapper reportMapper;
    private final FileStorageService fileStorageService;

    public List<BriefReportInfo> getReports() {
        return
reportRepository.findPublishedAndOrderedByDate().stream()
                    .map(reportMapper::toBriefInfoDto)
                    .toList();
    }

    public List<OrganizersReportDto> getAuthorReports(User
author) {
        return
reportRepository.findByAuthorId(author.getId()).stream()
                    .map(reportMapper::toOrganizersReportDto)
                    .toList();
    }

    public ReportInfo getReportById(Long reportId) throws
Exception {
        Report report = reportRepository.findById(reportId)
                    .orElseThrow(() -> new Exception("Новость не
найдена"));

        if (report.getStatus().equals(ReportStatus.DELETED) ) {
            throw new Exception("Новость была удалена");
        }
        return reportMapper.toInfoDto(report);
    }
}
```

```

    }

    public ReportInfo createReport(CreateReportDto dto, User
author) {
        Report report = reportMapper.createFromDto(dto, author);
        return
reportMapper.toInfoDto(reportRepository.save(report));
    }

    public ReportInfo updateReport(User author, Long reportId,
CreateReportDto dto) throws Exception {
        Report report = reportRepository.findById(reportId)
            .orElseThrow(() -> new Exception("Новость не
найдена"));

        if (!author.getId().equals(report.getAuthor().getId())
||

!report.getStatus().equals(ReportStatus.PUBLISHED) ) {
            throw new Exception("Нет доступа к редактированию
новости");
        }

        reportMapper.updateFromDto(report, dto);
        return
reportMapper.toInfoDto(reportRepository.save(report));
    }

    public ReportInfo deleteReport(Long reportId) throws
Exception {
        Report report = reportRepository.findById(reportId)
            .orElseThrow(() -> new Exception("Новость не
найдена"));

        report.setStatus(ReportStatus.DELETED);
        return
reportMapper.toInfoDto(reportRepository.save(report));
    }

    public String uploadReportPhoto(User user, Long reportId,
MultipartFile file) throws Exception {
        Report report = reportRepository.findById(reportId)
            .orElseThrow(() -> new Exception("Новость не
найдена"));

        if (!user.getId().equals(report.getAuthor().getId())) {
            throw new Exception("Нет доступа к изменению фото");
        }

        String photoUrl =
fileStorageService.saveReportPhoto(reportId, file);
        report.setPhotoUrl(photoUrl);
        reportRepository.save(report);
    }

```

```

        return photoUrl;
    }

    public void deleteReportPhoto(User user, Long reportId)
    throws Exception {
        Report report = reportRepository.findById(reportId)
            .orElseThrow(() -> new Exception("Новость не
найдена"));

        if (!user.getId().equals(report.getAuthor().getId())) {
            throw new Exception("Нет доступа к изменению фото");
        }

        fileStorageService.deleteReportPhoto(reportId);
        report.setPhotoUrl("/upload/reports/default.png");
        reportRepository.save(report);
    }
}

```

Листинг А.15 – Файл CommentService.java

```

package com.eco_portal.reports.service;

import com.eco_portal.reports.dto.CommentData;
import com.eco_portal.reports.dto.CreateCommentDto;
import com.eco_portal.reports.mapper.CommentMapper;
import com.eco_portal.reports.model.Comment;
import com.eco_portal.reports.model.Report;
import com.eco_portal.reports.repository.CommentRepository;
import com.eco_portal.reports.repository.ReportRepository;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class CommentService {

    private final CommentRepository commentRepository;
    private final ReportRepository reportRepository;
    private final CommentMapper commentMapper;

    public List<CommentData> getReportComments(Long reportId) {
        return commentRepository.findByReportId(reportId)
            .stream()
            .map(commentMapper::toData)
            .toList();
    }

    public CommentData addComment(Long reportId,
    CreateCommentDto dto, User author) throws Exception {

```

```

        Report report = reportRepository.findById(reportId)
            .orElseThrow(() -> new Exception("Новость или
отчет не найден"));
        report.setCountOfComments(report.getCountOfComments() +
1);
        reportRepository.save(report);
        Comment comment = commentMapper.createFromDto(dto,
author, report);
        return
commentMapper.toData(commentRepository.save(comment));
    }

    public void deleteComment(Long commentId) throws Exception {
        Comment comment = commentRepository.findById(commentId)
            .orElseThrow(() -> new Exception("Комментарий не
найден"));
        Report report =
reportRepository.findById(comment.getReport().getId())
            .orElseThrow(() -> new Exception("Новость или
отчет не найден"));
        report.setCountOfComments(report.getCountOfComments() -
1);
        reportRepository.save(report);
        commentRepository.delete(comment);
    }
}

```

Листинг A.16 – Файл ReportController.java

```

package com.eco_portal.reports.controller;

import com.eco_portal.reports.dto.*;
import com.eco_portal.reports.service.CommentService;
import com.eco_portal.reports.service.ReportService;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

@RestController
@RequestMapping("/api/reports")
@RequiredArgsConstructor
public class ReportController {

```

```

private final ReportService reportService;
private final CommentService commentService;

@GetMapping
public ResponseEntity<List<BriefReportInfo>> getReports() {
    return ResponseEntity.ok(reportService.getReports());
}

@GetMapping("/{reportId}")
public ResponseEntity<?> getReportById(@PathVariable Long
reportId) {
    try {
        return
ResponseEntity.ok(reportService.getReportById(reportId));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@GetMapping("/my")
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<List<OrganizersReportDto>>
getAuthorReports(@AuthenticationPrincipal User author) {
    return
ResponseEntity.ok(reportService.getAuthorReports(author));
}

@PostMapping
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<?> createReport(@RequestBody
CreateReportDto dto, @AuthenticationPrincipal User author) {
    try {
        return
ResponseEntity.ok(reportService.createReport(dto, author));
    } catch (Exception e) {
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

@PutMapping("/{reportId}")
@PreAuthorize("hasRole('ORGANIZER')")
public ResponseEntity<?>
updateReport(@AuthenticationPrincipal User author, @PathVariable
Long reportId, @RequestBody CreateReportDto dto) {
    try {
        return
ResponseEntity.ok(reportService.updateReport(author, reportId,
dto));
    } catch (Exception e) {

```

```

        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }

    @PutMapping("/{reportId}/delete")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<?> deleteReport(@PathVariable Long
reportId) {
        try {
            return
ResponseEntity.ok(reportService.deleteReport(reportId));
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
        }
    }

    @PostMapping("/{reportId}/photo")
    @PreAuthorize("hasRole('ORGANIZER')")
    public ResponseEntity<?> uploadReportPhoto(
        @AuthenticationPrincipal User user,
        @PathVariable Long reportId,
        @RequestParam("file") MultipartFile file
    ) {
        try {
            String photoUrl =
reportService.uploadReportPhoto(user, reportId, file);
            return ResponseEntity.ok(photoUrl);
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
        }
    }

    @DeleteMapping("/{reportId}/photo")
    @PreAuthorize("hasRole('ORGANIZER')")
    public ResponseEntity<?> deleteReportPhoto(
        @AuthenticationPrincipal User user,
        @PathVariable Long reportId
    ) {
        try {
            reportService.deleteReportPhoto(user, reportId);
            return ResponseEntity.noContent().build();
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
        }
    }
}

```

```

/* -----Эндпоинты комментариев-----
----- */

    @GetMapping("/{reportId}/comments")
    public ResponseEntity<List<CommentData>>
getReportComments(@PathVariable Long reportId) {
    return
ResponseEntity.ok(commentService.getReportComments(reportId));
}

    @PostMapping("/{reportId}/comments")
    @PreAuthorize("hasAnyRole('VOLUNTEER', 'ORGANIZER')")
    public ResponseEntity<?> addComment(
        @PathVariable Long reportId,
        @RequestBody CreateCommentDto dto,
        @AuthenticationPrincipal User author) {
        try {
            return
ResponseEntity.ok(commentService.addComment(reportId, dto,
author));
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
        }
    }

    @DeleteMapping("/comments/{commentId}")
    @PreAuthorize("hasAnyRole('ADMIN')")
    public ResponseEntity<?> deleteComment(@PathVariable Long
commentId) {
        try {
            commentService.deleteComment(commentId);
            return ResponseEntity.noContent().build();
        } catch (Exception e) {
            return ResponseEntity
                .status(HttpStatus.BAD_REQUEST)
                .body(e.getMessage());
        }
    }
}

```

Листинг А.17 – Файл ParticipationRequestService.java

```

package com.eco_portal.participation_requests.service;

import com.eco_portal.events.model.Event;
import com.eco_portal.events.repository.EventRepository;
import
com.eco_portal.participation_requests.dto.EventParticipationInfo
;

```

```

import
com.eco_portal.participation_requests.dto.OrganizerComment;
import
com.eco_portal.participation_requests.dto.ParticipantInfo;
import
com.eco_portal.participation_requests.mapper.ParticipationRequestMapper;
import
com.eco_portal.participation_requests.model.ParticipationRequest;
import
com.eco_portal.participation_requests.model.RequestStatus;
import
com.eco_portal.participation_requests.repository.ParticipationRequestRepository;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class ParticipationRequestService {

    private final ParticipationRequestRepository participationRequestRepository;
    private final EventRepository eventRepository;
    private final ParticipationRequestMapper participationRequestMapper;

    public List<EventParticipationInfo>
    getParticipantHistory(User user) {
        return
        participationRequestRepository.findByUserId(user.getId())
            .stream()

        .map(participationRequestMapper::toEventParticipationInfo)
            .toList();
    }

    public List<ParticipantInfo> getEventParticipants(Long
    eventId) {
        return participationRequestRepository
            .findByEventIdAndStatuses(
                eventId,
                List.of(RequestStatus.PENDING,
                RequestStatus.APPROVED)
            )
            .stream()

        .map(participationRequestMapper::toParticipantInfo)

```

```

        .toList();
    }

    public EventParticipationInfo sendParticipationRequest(Long
eventId, User user) throws Exception {
        Event event = eventRepository.findById(eventId)
            .orElseThrow(() -> new Exception("Событие не
найден"));
        List<ParticipationRequest> requests =
participationRequestRepository.findByUser_IdAndEvent_Id(user.getId(), eventId);

        if (!requests.isEmpty()) {
            throw new Exception("На данное мероприятие уже была
подана заявка от текущего пользователя");
        }

        ParticipationRequest request =
ParticipationRequest.builder()
            .user(user)
            .event(event)
            .status(RequestStatus.PENDING)
            .comment(null)
            .submittedAt(LocalDate.now())
            .build();

        return
participationRequestMapper.toEventParticipationInfo(participatio
nRequestRepository.save(request));
    }

    public ParticipantInfo approveParticipationRequest(Long
requestId, OrganizerComment comment) throws Exception {
        ParticipationRequest request =
participationRequestRepository.findById(requestId)
            .orElseThrow(() -> new Exception("Заявка не
найдена"));
        request.setStatus(RequestStatus.APPROVED);
        request.setComment(comment.getText());
        return participationRequestMapper

.toParticipantInfo(participationRequestRepository.save(request))
;
    }

    public ParticipantInfo rejectParticipationRequest(Long
requestId, OrganizerComment comment) throws Exception {
        ParticipationRequest request =
participationRequestRepository.findById(requestId)
            .orElseThrow(() -> new Exception("Заявка не
найдена"));
        request.setStatus(RequestStatus.REJECTED);
        request.setComment(comment.getText());
    }

```

```

        return participationRequestMapper

.toParticipantInfo(participationRequestRepository.save(request))
;
    }

}

```

Листинг А.18 – Файл ParticipationRequestController.java

```

package com.eco_portal.participation_requests.controller;

import
com.eco_portal.participation_requests.dto.EventParticipationInfo
;
import
com.eco_portal.participation_requests.dto.OrganizerComment;
import
com.eco_portal.participation_requests.dto.ParticipantInfo;
import
com.eco_portal.participation_requests.service.ParticipationReque
stService;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/participation-requests")
@RequiredArgsConstructor
public class ParticipationRequestController {

    private final ParticipationRequestService
participationRequestService;

    @GetMapping("/history")
    @PreAuthorize("hasRole('VOLUNTEER')")
    public ResponseEntity<List<EventParticipationInfo>>
getParticipationHistory(@AuthenticationPrincipal User user) {
        return
ResponseEntity.ok(participationRequestService.getParticipantHist
ory(user));
    }

    @GetMapping("/event/{eventId}/participants")
    @PreAuthorize("hasRole('ORGANIZER')")

```

```

        public ResponseEntity<List<ParticipantInfo>>
getEventParticipants(@PathVariable Long eventId) {
            return
ResponseEntity.ok(participationRequestService.getEventParticipan
ts(eventId));
        }

        @PostMapping("/event/{eventId}")
        @PreAuthorize("hasRole('VOLUNTEER')")
        public ResponseEntity<?> sendRequest(@PathVariable Long
eventId, @AuthenticationPrincipal User user) {
            try {
                return
ResponseEntity.ok(participationRequestService.sendParticipationR
equest(eventId, user));
            } catch (Exception e) {
                return ResponseEntity
                    .status(HttpStatus.BAD_REQUEST)
                    .body(e.getMessage());
            }
        }

        @PutMapping("/{requestId}/approve")
        @PreAuthorize("hasRole('ORGANIZER')")
        public ResponseEntity<?> approveRequest(@PathVariable Long
requestId, @RequestBody OrganizerComment comment) {
            try {
                return
ResponseEntity.ok(participationRequestService.approveParticipati
onRequest(requestId, comment));
            } catch (Exception e) {
                return ResponseEntity
                    .status(HttpStatus.NOT_FOUND)
                    .body(e.getMessage());
            }
        }

        @PutMapping("/{requestId}/reject")
        @PreAuthorize("hasRole('ORGANIZER')")
        public ResponseEntity<?> rejectRequest(@PathVariable Long
requestId, @RequestBody OrganizerComment comment) {
            try {
                return
ResponseEntity.ok(participationRequestService.rejectParticipatio
nRequest(requestId, comment));
            } catch (Exception e) {
                return ResponseEntity
                    .status(HttpStatus.NOT_FOUND)
                    .body(e.getMessage());
            }
        }
    }
}

```

Листинг А.19 – Файл NotificationService.java

```
package com.eco_portal.notifications.service;

import com.eco_portal.notifications.dto.NotificationDto;
import com.eco_portal.notifications.mapper.NotificationMapper;
import com.eco_portal.notifications.model.Notification;
import
com.eco_portal.notifications.repository.NotificationRepository;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.time.LocalDateTime;

@Service
@RequiredArgsConstructor
public class NotificationService {

    private final NotificationRepository notificationRepository;
    private final NotificationMapper notificationMapper;

    public List<NotificationDto> getUserNotifications(User user,
boolean onlyUnread) {
        List<Notification> notifications = onlyUnread
            ?
notificationRepository.findByIdAndHasReadFalseOrderByCreated
AtDesc(user.getId())
            :
notificationRepository.findByIdOrderByCreatedAtDesc(user.get
Id());

        return
notifications.stream().map(notificationMapper::toDto).toList();
    }

    public void markAsRead(Long notificationId, User user)
throws Exception {
        Notification notification =
notificationRepository.findById(notificationId)
            .orElseThrow(() -> new Exception("Уведомление не
найдено"));

        if
(!notification.getUser().getId().equals(user.getId())) {
            throw new Exception("Нет доступа к уведомлению");
        }

        notification.setHasRead(true);
        notificationRepository.save(notification);
    }
}
```

```

        public void sendNotification(User recipient, String content)
        {
            Notification notification = Notification.builder()
                .user(recipient)
                .content(content)
                .createdAt(LocalDateTime.now())
                .hasRead(false)
                .build();

            notificationRepository.save(notification);
        }
    }
}

```

Листинг А.20 – Файл NotificationController.java

```

package com.eco_portal.notifications.controller;

import com.eco_portal.notifications.dto.NotificationDto;
import com.eco_portal.notifications.service.NotificationService;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/notifications")
@RequiredArgsConstructor
public class NotificationController {

    private final NotificationService notificationService;

    @GetMapping
    @PreAuthorize("hasAnyRole('VOLUNTEER', 'ORGANIZER', 'ADMIN')")
    public ResponseEntity<List<NotificationDto>>
        getUserNotifications(
            @AuthenticationPrincipal User user,
            @RequestParam(defaultValue = "false") boolean
            unreadOnly
        ) {
            return
            ResponseEntity.ok(notificationService.getUserNotifications(user,
            unreadOnly));
        }
    }
}

```

```

        @PutMapping("/{notificationId}/read")
        @PreAuthorize("hasAnyRole('VOLUNTEER', 'ORGANIZER',
'ADMIN')")
        public ResponseEntity<?> markAsRead(@PathVariable Long
notificationId, @AuthenticationPrincipal User user) {
            try {
                notificationService.markAsRead(notificationId,
user);

                return ResponseEntity.ok().build();
            } catch (Exception e) {
                return ResponseEntity
                    .status(HttpStatus.NOT_FOUND)
                    .body(e.getMessage());
            }
        }
    }
}

```

Листинг А.21 – Файл AchievementService.java

```

package com.eco_portal.achievements.service;

import com.eco_portal.achievements.dto.AchievementProgressDto;
import com.eco_portal.achievements.mapper.AchievementMapper;
import com.eco_portal.achievements.model.Achievement;
import com.eco_portal.achievements.model.CriteriaType;
import com.eco_portal.achievements.model.UserAchievement;
import
com.eco_portal.achievements.repository.AchievementRepository;
import
com.eco_portal.achievements.repository.UserAchievementRepository
;
import com.eco_portal.events.model.Event;
import com.eco_portal.events.model.EventStatus;
import
com.eco_portal.participation_requests.model.ParticipationRequest
;
import
com.eco_portal.participation_requests.model.RequestStatus;
import
com.eco_portal.participation_requests.repository.ParticipationRe
questRepository;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class AchievementService {

```

```

        private final AchievementRepository achievementRepository;
        private final UserAchievementRepository
userAchievementRepository;
        private final ParticipationRequestRepository
participationRequestRepository;
        private final AchievementMapper achievementMapper;

        public List<AchievementProgressDto>
getAllWithUserProgress(User user) {
            List<Achievement> all = achievementRepository.findAll();

            List<Long> userAchievementIds =
userAchievementRepository.findByUserId(user.getId()).stream()
                .map(userAch ->
userAch.getAchievement().getId())
                .toList();

            long finishedCount =
participationRequestRepository.countFinishedParticipationByUserI
d(
                user.getId(),
                RequestStatus.APPROVED,
                EventStatus.FINISHED
            );

            return all.stream().map(achievement -> {
                int required = achievement.getCriteriaValue();
                int current = switch (achievement.getCriteriaType())
{
                    case EVENT_COUNT -> (int) finishedCount;
                    // можно будет добавить другие типы достижений в
будущем
                };

                boolean achieved =
userAchievementIds.contains(achievement.getId());

                return achievementMapper.toProgressDto(
                    achievement,
                    current,
                    required,
                    achieved
                );
            }).toList();
        }

        public void checkAndAwardByFinishedEvent(Event event) throws
Exception {
            if (!event.getStatus().equals(EventStatus.FINISHED)) {
                throw new Exception("Мероприятие не завершено");
            }

            List<User> participants = participationRequestRepository

```

```

        .findByEventIdAndStatuses(
            event.getId(),
            List.of(RequestStatus.APPROVED)
        )
        .stream()
        .map(ParticipationRequest::getUser)
        .toList();

        for (User user : participants) {
            long count =
                participationRequestRepository.countFinishedParticipationByUserId(
                    user.getId(),
                    RequestStatus.APPROVED,
                    EventStatus.FINISHED
                );
            List<Achievement> achievements =
                achievementRepository.findByCriteriaType(CriteriaType.EVENT_COUNT);

            for (Achievement achievement : achievements) {
                if (count >= achievement.getCriteriaValue() &&
                    !userAchievementRepository.existsByUserIdAndAchievementId(user.getId(), achievement.getId())) {

                    UserAchievement userAch =
                        UserAchievement.builder()
                            .user(user)
                            .achievement(achievement)
                            .awardedAt(LocalDate.now())
                            .build();
                    userAchievementRepository.save(userAch);
                }
            }
        }
    }
}

```

Листинг A.22 – Файл AchievementController.java

```

package com.eco_portal.achievements.controller;

import com.eco_portal.achievements.dto.AchievementProgressDto;
import com.eco_portal.achievements.service.AchievementService;
import com.eco_portal.users.model.User;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.GetMapping;

```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
@RequestMapping("/api/achievements")
@RequiredArgsConstructor
public class AchievementController {

    private final AchievementService achievementService;

    @GetMapping("/progress")
    @PreAuthorize("hasAnyRole('VOLUNTEER', 'ORGANIZER')")
    public ResponseEntity<List<AchievementProgressDto>>
getUserProgress(@AuthenticationPrincipal User user) {
        return
ResponseEntity.ok(achievementService.getAllWithUserProgress(user
));
    }

}
```

Приложение Б
(рекомендуемое)

Исходный код создания таблиц базы данных

Листинг Б.1 – Файл create-tables.sql

```

-- Регионы
CREATE TABLE regions (
    region_id BIGSERIAL PRIMARY KEY,
    region_name VARCHAR(100) NOT NULL
);

-- Пользователи
CREATE TABLE users (
    user_id BIGSERIAL PRIMARY KEY,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    region_id INT NOT NULL,
    taxpayer_number VARCHAR(100) UNIQUE,
    user_role VARCHAR(20) CHECK (user_role IN ('VOLUNTEER',
'ORGANIZER', 'ADMIN')) NOT NULL,
    is_locked BOOLEAN DEFAULT FALSE NOT NULL,
    registered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    avatar_url VARCHAR(255) DEFAULT '/upload/users/default.png'
NOT NULL,
    FOREIGN KEY (region_id) REFERENCES regions(region_id) ON
DELETE SET NULL
);

-- Заявки на регистрацию организаций
CREATE TABLE organizer_requests (
    request_id BIGSERIAL PRIMARY KEY,
    organizer_name VARCHAR(100) NOT NULL,
    taxpayer_number VARCHAR(100) UNIQUE NOT NULL,
    user_id INT NOT NULL,
    request_status VARCHAR(20) CHECK (request_status IN
('PENDING', 'APPROVED', 'REJECTED')) DEFAULT 'PENDING' NOT NULL,
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE
CASCADE
);

-- Категории мероприятий
CREATE TABLE event_categories (
    category_id BIGSERIAL PRIMARY KEY,
    category_name VARCHAR(100) NOT NULL
);

-- Мероприятия
CREATE TABLE events (
    event_id BIGSERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    event_description TEXT,
    event_location VARCHAR(200) NOT NULL,
    event_date TIMESTAMP NOT NULL,
    organizer_id INT NOT NULL,

```

```

        region_id INT NOT NULL,
        category_id INT NOT NULL,
        event_status VARCHAR(20) CHECK (event_status IN
('PUBLISHED', 'CANCELLED', 'DELETED', 'FINISHED')) DEFAULT
'PUBLISHED' NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
        photo_url VARCHAR(255) DEFAULT '/upload/events/default.png'
NOT NULL,
        FOREIGN KEY (organizer_id) REFERENCES users(user_id) ON
DELETE CASCADE,
        FOREIGN KEY (region_id) REFERENCES regions(region_id) ON
DELETE CASCADE,
        FOREIGN KEY (category_id) REFERENCES
event_categories(category_id) ON DELETE CASCADE
);

-- Заявки на участие
CREATE TABLE participation_requests (
    request_id BIGSERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    event_id INT NOT NULL,
    request_status VARCHAR(20) CHECK (request_status IN
('PENDING', 'APPROVED', 'REJECTED')) DEFAULT 'PENDING' NOT NULL,
    comment TEXT,
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE
CASCADE,
    FOREIGN KEY (event_id) REFERENCES events(event_id) ON DELETE
CASCADE,
    UNIQUE(user_id, event_id) -- Ограничение на одну заявку на
мероприятие
);

-- Новости и отчеты о мероприятиях
CREATE TABLE reports (
    report_id BIGSERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    content TEXT NOT NULL,
    author_id INT NOT NULL,
    event_id INT, -- Может быть NULL, если новость не привязана
к мероприятию
    report_status VARCHAR(20) CHECK (report_status IN
('PUBLISHED', 'DELETED')) DEFAULT 'PUBLISHED' NOT NULL,
    published_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    count_of_comments INT DEFAULT 0 NOT NULL,
    photo_url VARCHAR(255) DEFAULT '/upload/reports/default.png'
NOT NULL,
    FOREIGN KEY (author_id) REFERENCES users(user_id) ON DELETE
CASCADE,
    FOREIGN KEY (event_id) REFERENCES events(event_id) ON DELETE
SET NULL
);

```

```

-- Комментарии к новостям
CREATE TABLE comments (
    comment_id BIGSERIAL PRIMARY KEY,
    content TEXT NOT NULL,
    author_id INT NOT NULL,
    report_id INT NOT NULL,
    published_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    FOREIGN KEY (author_id) REFERENCES users(user_id) ON DELETE
    CASCADE,
    FOREIGN KEY (report_id) REFERENCES reports(report_id) ON
    DELETE CASCADE
);

-- Уведомления
CREATE TABLE notifications (
    notification_id BIGSERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    has_read BOOLEAN DEFAULT FALSE NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE
    CASCADE
);

-- Достижения
CREATE TABLE achievements (
    achievement_id BIGSERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    achievement_description TEXT,
    criteria_type VARCHAR(20) CHECK (criteria_type IN
    ('EVENT_COUNT')) NOT NULL,
    criteria_value INT NOT NULL
);

-- Связь пользователей с достижениями
CREATE TABLE user_achievements (
    user_achievement_id BIGSERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    achievement_id INT NOT NULL,
    awarded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE
    CASCADE,
    FOREIGN KEY (achievement_id) REFERENCES
    achievements(achievement_id) ON DELETE CASCADE
);

```