МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ ДЛЯ ОРГАНИЗАЦИИ ПРОЦЕССА УБОРКИ МУСОРА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Устюшина Богдана Антоновича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ С. В. Миронов

Саратов 2025

В современном мире проблема загрязнения окружающей среды становится всё более актуальной, и мусор, оставляемый на улицах, представляет собой одну из самых серьёзных составляющих этой проблемы. Накопление отходов в общественных местах не только нарушает эстетический облик городов, но и оказывает негативное воздействие на здоровье человека и окружающую экосистему. С каждым годом количество бытового и промышленного мусора растёт, и его присутствие на улицах городов становится всё более заметным. Это вызывает множество проблем: от ухудшения качества воздуха до разрушения экосистем и негативного влияния на биоразнообразие.

Целью работы является разработка web-приложения для улучшения организации процесса нахождения и уборки мусора на улицах города. Также планируется сделать упор на создание чистой архитектуры приложения и построение масштабируемой системы.

Таким образом, для достижения цели необходимо решить следующие задачи:

1. рассмотреть используемые технологии, описать преимущества выбранного стека;
2. описать архитектуру и структуру построенной информационной системы, database-, backend- и frontend-сервисов, а также описать основные решения дизайна архитектуры;
3. реализовать это с помощью выбранных инструментов.

Структура и объём работы. Для решения поставленных задач выполнена выпускная квалификационная работа, включающая в себя введение, 3 основные главы, заключение, список использованных источников из 20 наименований и 3 приложения. Работа изложена на 48 страницах. Первая глава имеет название «Обзор используемых инструментов» и содержит основную информацию об используемых в дальнейшей работе средствах для реализации практического продуктаю. Вторая глава имеет название «Разработка приложения»: она содержит описание основных архитектурных решений, которые были приняты при проектировании приложения, и комментарии наиболее важных фрагментов программного кода. Третья глава называется «Использование приложения» и содержит демонстрацию работы приложения и показывает основной бизнес-процесс приложения. Выпускная квалификационная работа заканчивается заключением, списком использованных источников, а также приложениями А-В.

1 Обзор используемых инструментов

В первой главе представлен детальный анализ инструментов и технологий, использованных при создании web-приложения для организации процесса уборки мусора. Тщательный подбор технологий был ключевым этапом, определившим масштабируемость, отказоустойчивость и удобство сопровождения итоговой системы. Разработка охватывает как клиентскую, так и серверную части, работу с базами данных, контейнеризацию и безопасную аутентификацию пользователей.

1.1 Общие принципы подбора стека

Выбор стека производился с учётом следующих критериев:

- производительность и масштабируемость системы;
- активное сообщество и поддержка;
- наличие готовых решений и документации;
- удобство интеграции и сопровождения;
- соответствие современным практикам (clean architecture, CI/CD).

С учётом этих требований был сформирован стек, включающий Go, React, PostgreSQL, Docker и JWT. Далее каждая технология рассматривается подробно.

1.2 Go

Основу backend-приложения составляет язык Go (Golang). Он компилируемый, строго типизированный, с минималистичным синтаксисом и встроенной поддержкой конкурентности (горутины). Среди преимуществ:

- высокая производительность — близка к C/C++, без затрат на виртуальные машины;
- простота синтаксиса, исключение перегрузок и сложной иерархии;
- встроенная модель параллельного выполнения (goroutines и channels);
- мощная стандартная библиотека;
- кроссплатформенная сборка из одной команды.

Go широко используется в промышленной разработке (например, в Docker, Kubernetes, Terraform). В данном проекте Go выбран как оптимальное решение для построения производительного API с поддержкой одновременных запросов и безопасного взаимодействия с БД.

1.3 React

Фронтенд реализован на базе React — библиотеки для построения SPA-приложений. React позволяет разрабатывать гибкие пользовательские интерфейсы с разделением логики по компонентам. Среди его преимуществ:

— виртуальный DOM — уменьшает накладные расходы на рендеринг;
— компонентный подход — упрощает повторное использование кода;
— one-way data flow — делает поведение интерфейса предсказуемым;
— широкая экосистема (React Router, Redux, Radix UI, Shadcn/UI).

React был выбран по совокупности факторов: зрелость решения, поддержка TypeScript, документация, богатая экосистема и высокая производительность. Также он отлично сочетается с современными инструментами сборки (Vite) и стилизации (Tailwind CSS).

1.4 PostgreSQL

В качестве СУБД используется PostgreSQL — мощная объектно-реляционная СУБД с открытым исходным кодом. Среди причин выбора:

— полная поддержка SQL-стандарта;
— расширения (например, PostGIS, pg_trgm);
— надёжность, транзакционность (ACID), безопасность;
— встроенная поддержка JSON, индексирование, репликация;
— активное сообщество и длительная история развития.

PostgreSQL хорошо масштабируется и применима как для OLTP-сценариев (обработка событий), так и для аналитических запросов (статистика).

1.5 Docker

Для развёртывания, тестирования и CI/CD используется Docker. Контейнеризация обеспечивает изоляцию зависимостей и среды выполнения, что делает приложение переносимым и воспроизводимым. Преимущества Docker:

— независимость от хост-системы;
— быстрое масштабирование (поддержка кластеров, orchestration);
— простота создания окружений (Dockerfile, docker-compose);
— поддержка DevOps-практик: CI/CD, автоматическое тестирование, сборка.

Использование `docker-compose` позволяет разворачивать сразу несколько сервисов (БД, сервер, мигратор), снижая накладные расходы на настройку

среды.

1.6 JWT

Для управления доступом используется механизм JWT (JSON Web Token). Он обеспечивает stateless-аутентификацию: не требует хранения сессий на сервере. Особенности реализации:

- токены подписываются HMAC и содержат claims (id, роль, срок действия);
- используется пара access/refresh-токенов;
- refresh-токены хранятся в HttpOnly cookie;
- access-токены передаются в Authorization-заголовках;
- предусмотрено автоматическое продление и отзыв токенов.

Такой подход обеспечивает масштабируемость (особенно в будущем при переходе на микросервисную архитектуру) и безопасность (подпись, срок действия, поддержка blacklist/whitelist).

1.7 Вспомогательные инструменты и библиотеки

Для удобства разработки и тестирования используются:

- Postman — для тестирования API;
- DBeaver — интерфейс для визуальной работы с базой данных;
- Air — автоматическая перезагрузка сервера при изменении кода;
- Goose — миграции БД на Go и SQL;
- Swag — генерация OpenAPI-документации на основе комментариев в коде;
- Vite — быстрый инструмент сборки frontend-проекта;
- Yarn — менеджер зависимостей, используемый вместо npm.

Эти утилиты ускоряют цикл разработки и снижают вероятность ошибок, позволяя сосредоточиться на бизнес-логике.

Подбор стека технологий был основан на реальных требованиях к системе: надёжность, скорость отклика, модульность, расширяемость. Все выбранные компоненты — это зрелые, активно поддерживаемые решения, широко применяемые в промышленной разработке. Их совместное использование позволило построить удобную, масштабируемую и безопасную архитектуру, подходящую как для MVP, так и для последующего промышленного развертывания.

2 Разработка приложения

2.1 Анализ и проектирование информационной системы

Процесс проектирования начался с построения бизнес-логики, определяющей поведение пользователей и сущностей в приложении. Основной задачей системы является сбор информации о мусоре на улицах города через пользовательские заявки и содействие в оперативной уборке.

В центре бизнес-процесса — взаимодействие двух ключевых сущностей: пользователя и события. Пользователь может создавать заявки (события) на уборку мусора, прикладывая фотографии и координаты. После этого событие попадает в очередь на модерацию. Администратор (модератор) оценивает заявку и либо подтверждает её, либо отклоняет. Подтверждённое событие становится видимым другим пользователям, которые могут «принять» его в работу. Завершение уборки также требует верификации администратором.

Особое внимание было уделено безопасной, прозрачной и контролируемой логике жизненного цикла события, включающей пять ключевых состояний: создание, модерация, публикация, выполнение, подтверждение. Отдельно реализована логика отката (например, если уборка не подтверждена).

Дополнительно, в рамках проектирования описаны ограничения:

- пользователь может находиться только в одном городе (при регистрации);
- админ не может создавать события;
- каждый пользователь может иметь не более одного активного задания;
- только зарегистрированные пользователи могут просматривать карту заявок.

2.2 Архитектура приложения

Приложение построено по принципам чистой архитектуры, обеспечивающей слабую связанность между слоями. Архитектура backend-сервиса реализована в виде слоёв:

- контроллер (API) — принимает HTTP-запросы;
- сервис (бизнес-логика) — обрабатывает действия и сценарии;
- репозиторий — взаимодействует с базой данных.

Для упрощения инициализации зависимостей применяется DI-контейнер — единая точка сборки компонентов. Это позволяет легко масштабировать и переиспользовать сервисы.

Также используется Tx-manager, обеспечивающий корректную работу с транзакциями. Он позволяет в одном контексте выполнять действия, затрагивающие несколько таблиц (например, логин пользователя и сохранение сессии), сохраняя целостность данных и упрощая rollback при ошибках.

Особое внимание уделено покрытию ошибок и их типизации: каждый слой возвращает конкретные ошибки (например, `ErrNotFound`, `ErrDuplicate`), которые последовательно обрабатываются вплоть до фронтенда.

2.3 Архитектура базы данных

База данных построена на PostgreSQL. Она включает в себя следующие основные сущности:

- `user_` — данные пользователей (имя, email, город);
- `event_` — события, созданные пользователями;
- `event_data_` — координаты, описание, фото до/после;
- `session_` — информация о сессиях (время, истечение);
- `user_role_`, `role_` — ролевая система;
- `event_status_` — статус события.

База нормализована: данные разделены по смыслу, но избыточного дробления избегают (для оптимизации SQL-запросов). Используются внешние ключи с каскадным удалением, а также many-to-many связи (например, пользователи и роли).

Миграции реализованы через инструмент Goose. Миграции представляют собой SQL-файлы, которые могут применяться и откатываться в рамках CI/CD или при локальной разработке. Это даёт контроль над версионированием схемы БД.

2.4 Архитектура backend-сервиса

Сервер реализован на языке Go с использованием фреймворка Gin. Этот стек обеспечивает высокую производительность и лаконичный код. Структура проекта строго организована: папки `api`, `service`, `repository`, `model`, `dto`.

Каждое действие (например, создание события) реализовано в виде последовательности:

1. DTO;
2. сервисная сущность;
3. сущность репозитория.

Используются мапперы, чтобы отделить формат входных/выходных данных от внутренней бизнес-логики.

Для работы с HTTP-клиентом применяются middleware (например, для авторизации), кастомные обработчики ошибок, логгирование, валидация запросов (через binding и аннотации).

Особое внимание уделено работе с токенами. После логина:

1. access-токен отсылается в заголовке;
2. refresh-токен может храниться в `HttpOnly` cookie;
3. при истечении access, клиент запрашивает обновление через endpoint;
4. refresh-токен проверяется и используется для генерации новой пары.

Реализованы сценарии: невалидный токен, истёкший токен, отзыв токена (например, при выходе). Это повышает безопасность и соответствие best practices.

2.5 Контейнеризация и деплой

Для развертывания системы используется Docker. Все сервисы (PostgreSQL, backend, мигратор) запускаются через `docker-compose`. Пример сценария:

— первый запуск — сборка образов, применение миграций, запуск backend;
— обновление — остановка, пересборка, перезапуск;
— удаление — очистка контейнеров и томов.

Контейнер мигратора создаёт минимальную Alpine-сборку с утилитой Goose. Bash-скрипт извлекает переменные окружения из `.env`, после чего запускает миграции. Такой подход повышает повторяемость окружения и удобство CI/CD.

2.6 Архитектура frontend-сервиса

Клиент реализован на React. Архитектура построена по паттерну Atomic Design. Для маршрутизации используется React Router. Состояние авторизации хранится в React Context, токены извлекаются из localStorage и обновляются при необходимости.

Обмен с API реализован через Axios. Интерфейс обрабатывает:

— ошибки ввода (валидация, сообщения),
— ошибки авторизации (401, 403),
— ошибки сети (переход в offline-режим).

UI построен с использованием библиотек Shadcn/UI и Radix UI, обеспечивающих современный и адаптивный внешний вид. Стилизация ведётся через Tailwind CSS.

Таким образом, разработка приложения велась с соблюдением принципов чистой архитектуры, масштабируемости и безопасности. Проект разбит на логически изолированные слои, обеспечивающие гибкость при доработках. Использование Docker, миграций и автогенерации документации облегчает поддержку. Приложение готово к промышленному использованию и может быть расширено под новые требования.

3 Использование приложения

3.1 Регистрация пользователя

Вход в систему начинается с регистрации. Для этого пользователь указывает имя, электронную почту, город проживания и пароль. Если email совпадает с одним из заранее определённых в системе как «администраторский», пользователю автоматически присваивается роль администратора. В остальных случаях создаётся аккаунт обычного пользователя.

Процесс регистрации включает:

— проверку валидности данных на стороне клиента;
— отправку запроса на backend (endpoint `/auth/register`);
— сохранение пользователя в БД, создание сессии и генерация JWT-токенов;
— возврат токенов и редирект в личный кабинет.

После регистрации пользователь может войти в систему, используя логин и пароль. При успешной авторизации система возвращает access- и refresh-токены.

3.2 Аутентификация и авторизация

Вход в систему осуществляется через форму логина, где пользователь вводит email и пароль. При успешной проверке данных backend создаёт пару JWT-токенов:

— access-токен — используется для авторизации в большинстве запросов;
— refresh-токен — применяется для продления сессии при истечении access.

Токены хранятся на клиенте: access — в памяти приложения, refresh — в `HttpOnly` cookie. При каждом запросе access-токен передаётся в заголовке, обеспечивая проверку подлинности пользователя.

3.3 Работа с событиями

Зарегистрированные пользователи могут создавать события, указывая описание, координаты, время и фото. Событие передаётся на модерацию. Валидация происходит как на клиенте (проверка формата), так и на сервере (проверка структуры и наличия обязательных полей).

После модерации событие становится доступным другим пользователям. В интерфейсе доступен список активных заявок и карта, где каждое событие отображается в виде маркера. Пользователь может принять участие в уборке, выбрав подходящую заявку.

Процесс включает следующие шаги:

1. выбор события и нажатие кнопки «Принять»;
2. выполнение уборки;
3. загрузка фотографии «после»;
4. отправка на модерацию (подтверждение выполнения).

Модератор проверяет загруженные материалы. При подтверждении событие переводится в статус «архивировано». В случае отклонения оно возвращается в активные и доступно для других пользователей.

3.4 Личный кабинет и взаимодействие с системой

Каждый пользователь имеет доступ к личному кабинету, где отображается:

— статистика (количество выполненных заявок, участие в уборках);
— настройки профиля (смена пароля);
— список своих заявок (созданных и выполненных);

В интерфейсе доступны уведомления об изменении статуса событий, ошибки авторизации и результаты модерации.

Как результат, пользовательское взаимодействие организовано интуитивно и безопасно. Все основные функции — от регистрации до завершения заявок — реализованы с учётом UX и архитектурной строгости. Механизмы авторизации, модерации и откликов обеспечивают устойчивость системы к ошибкам и недобросовестным действиям пользователей.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были выполнены следующие задачи:

1. рассмотрены технологии, описаны преимущества выбранного стека;
2. описана архитектура и структуру построенной информационной системы;
3. разработано и продемонстрировано веб-приложение.

Таким образом, все поставленные задачи были выполнены.

Разработанное приложение представляет собой важный шаг в борьбе с загрязнением городской среды, объединяя усилия жителей для решения этой актуальной проблемы. Тем не менее, будущее данного проекта достаточно перспективно и может предполагать, например:

1. проверка другого жизненного процесса событий с меньшей модерацией;
2. расширение на другие города с возможностью разделения модерации по городам;
3. внедрение технологий глубокого обучения для отмены модерации как таковой и её делегирование нейронным сетям;
4. внедрение систем оповещения либо с помощью нативных оповещений браузера, либо с помощью телеграм-бота.

Таким образом, существуют хорошие перспективы работы над приложением и дальше для улучшения возможности решения существующих проблем общества, описанных выше. Приложение имеет потенциал для увеличения осведомлённости населения о проблемах экологии и вовлечения граждан в активные действия по улучшению своей окружающей среды.