# Отчет ВВЕДЕНИЕ

Современные строительные проекты отличаются высокой сложностью и большим количеством участников: подрядчиков, инженеров, менеджеров и заказчиков. Одной из ключевых задач при организации строительных процессов является контроль качества и своевременное устранение дефектов. Отсутствие централизованного инструмента для регистрации и отслеживания дефектов приводит к потерям информации, затягиванию сроков устранения и снижению прозрачности работы.

Данная работа посвящена разработке веб-приложения для управления дефектами на строительных объектах. Система призвана обеспечить полный цикл работы: от регистрации дефекта инженером до анализа отчётности руководителем. Она позволит стандартизировать процессы, сократить время реакции на проблемы и повысить эффективность взаимодействия между участниками проекта.

**Цель работы** — разработка информационной системы «Defect Management System», обеспечивающей хранение, обработку и аналитику данных о дефектах на строительных объектах.

**Объект исследования** — процессы контроля качества и устранения дефектов в строительных проектах.

**Предмет исследования** — автоматизация управления дефектами с использованием специализированной информационной системы.

Методы исследования включают анализ предметной области, проектирование архитектуры программных систем, применение современных средств веб-разработки и тестирования.

Структура отчёта включает обзор предметной области и существующих решений, описание требований и архитектуры системы, реализацию ключевых компонентов и тестирование программного продукта.

# 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Анализ предметной области

В строительной отрасли дефекты могут возникать на всех этапах жизненного цикла проекта: от проектирования и закупки материалов до сдачи объекта заказчику. Отсутствие централизованного контроля приводит к ряду проблем: информация хранится в разрозненных источниках (акты, Excel-таблицы, переписка по почте); сложно отследить статус устранения дефекта;

отсутствует история изменений и ответственность конкретных исполнителей;

невозможно получить аналитическую картину по причинам возникновения дефектов.

**Участники процессов:**

инженеры — фиксируют дефекты, добавляют фото и комментарии;

менеджеры — назначают ответственных, контролируют сроки устранения;

руководители и заказчики — анализируют прогресс и формируют отчётность.

**Информационные процессы:**

регистрация нового дефекта; назначение ответственного; изменение статуса;

добавление комментариев и вложений; формирование аналитических отчётов.

## 1.2 Описание вариантов решения проблемы

**Excel и Google Sheets**

**+** простота, низкая стоимость;

- ошибки, отсутствие многопользовательского режима, нет аналитики.

**CRM/ERP общего назначения (1С, Bitrix24)**

**+** готовая инфраструктура, надёжность;

– высокая стоимость, избыточность функций, сложность адаптации.

**Собственная специализированная система**

**+** учитывает специфику строительной отрасли, масштабируется;

– требует затрат на разработку.

Вывод: оптимальным является создание специализированной информационной системы.

## 1.3 Требования к системе

**Функциональные:**

* регистрация и аутентификация пользователей;
* разграничение прав доступа (менеджер, инженер, наблюдатель);
* управление проектами и объектами;
* регистрация и редактирование дефектов (описание, приоритет, сроки, вложения);
* статусы: «Новый → В работе → На проверке → Закрыт/Отменён»;
* комментарии и история изменений;
* фильтрация и поиск;
* отчёты и экспорт в Excel/CSV;
* аналитика (графики, статистика).

**Нефункциональные:**

время отклика ≤ 1 секунды (до 50 активных пользователей);

адаптивный интерфейс;

резервное копирование БД ежедневно;

хранение паролей через Argon2;

защита от SQL-инъекций, XSS и CSRF;

совместимость с Chrome/Firefox/Edge.

**Интерфейс:**

формы ввода и редактирования; таблицы с фильтрацией; панели поиска;

уведомления об ошибках и подтверждения действий.

# 2. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

## 2.1 Выбор инструментов и технологий

**Backend:**

* Python 3.11 + Django 4.2 + Django REST Framework;
* PostgreSQL 17 — основная БД;
* Redis 7 — кэш и брокер сообщений;
* Celery — асинхронные задачи;
* Gunicorn — WSGI сервер.

**Frontend:**

* React 18 + TypeScript;
* Material-UI (MUI);
* Redux Toolkit — управление состоянием;
* React Router — маршрутизация;
* Axios — HTTP-клиент.

**Инфраструктура:**

Docker & Docker Compose — контейнеризация;

Nginx — прокси и балансировка;

Prometheus + Grafana — мониторинг;

GitHub Actions — CI/CD.

**Тестирование:**

Pytest (backend);

Jest + React Testing Library (frontend);

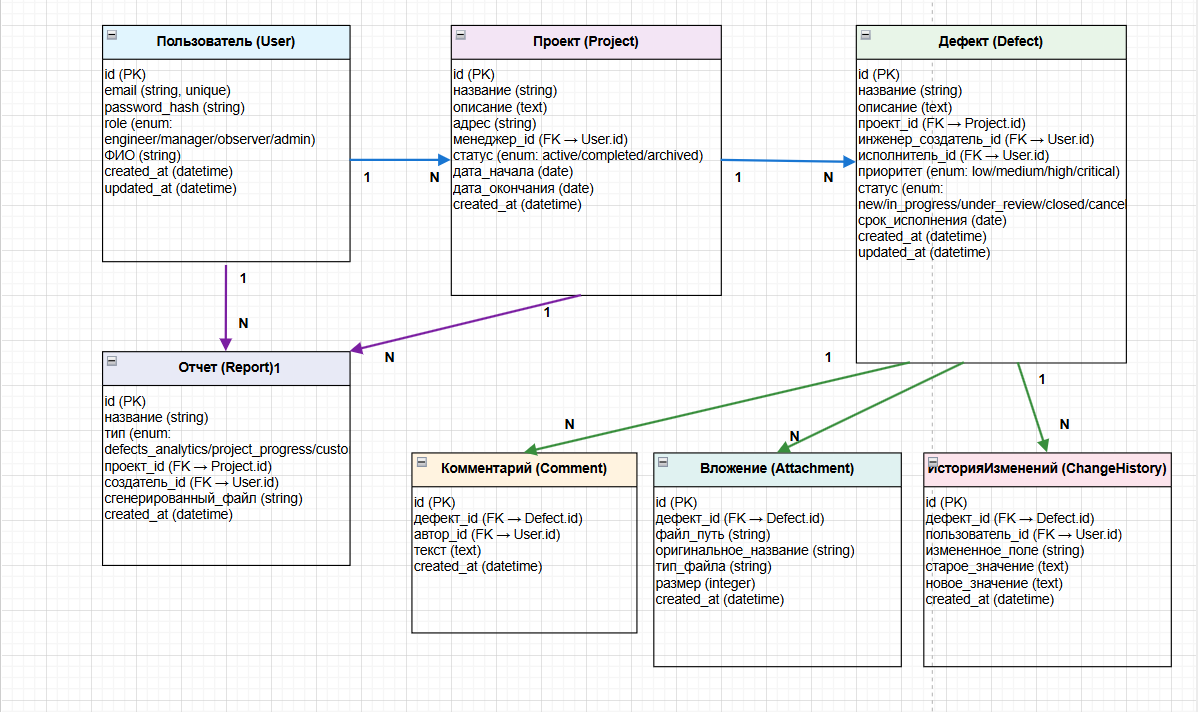
Factory Boy, Coverage.py.

Выбор обусловлен надёжностью, открытостью и поддержкой сложных сценариев.

## 2.2 Разработка проекта решения

## ****2.2.1Диаграммы для проектирования системы:****

***ERD*** — ***диаграмма базы данных***

Схема таблиц (пользователь, проекты, дефекты) и связей между ними. Рисунок 2.1 — **ERD-диаграмма базы данных**

Один пользователь-менеджер может управлять многими проектами.  
Один проект может содержать много дефектов.

Один дефект может иметь много комментариев, вложений и историй изменений.

## ****2.2.2Диаграмма вариантов использования (Use Case)****

Описание диаграммы:

Для наглядного представления взаимодействия пользователей с системой управления дефектами на строительных объектах была разработана UML-диаграмма вариантов использования (use case diagram). Диаграмма отражает ключевые бизнес-процессы и роли участников.

Основные роли в системе:

**Руководитель** — обладает полными правами просмотра аналитики: анализирует отчеты по дефектам, оценивает эффективность работы, принимает стратегические решения.

**Менеджер** — управляет проектами, назначает исполнителей на дефекты, контролирует сроки исполнения, формирует оперативные отчеты.

**Инженер** — регистрирует новые дефекты, обновляет статусы выполненных работ, добавляет комментарии и фотоматериалы.

**Наблюдатель (Заказчик)** — просматривает статусы дефектов по проектам, отслеживает прогресс устранения, получает отчеты.

Связи между сценариями:

Процесс "Формирование аналитического отчета" расширяет (extend) базовый сценарий "Анализ эффективности работы"

"Управление проектом" включает (include) "Назначение исполнителей на дефекты".

Все роли наследуют права доступа от базовой роли "Наблюдатель"

Диаграмма наглядно демонстрирует: разделение функционала по уровням доступа, основные бизнес-процессы управления дефектами, точки взаимодействия пользователей с системой.

Такой подход позволяет:

Четко определить требования к системе; выявить ключевые функциональные модули; оптимизировать процесс разработки приложения; обеспечить правильное разграничение прав доступа.

Диаграмма служит основой для:

Разработки технического задания, проектирования архитектуры системы, создания тестовых сценариев, обучения персонала.

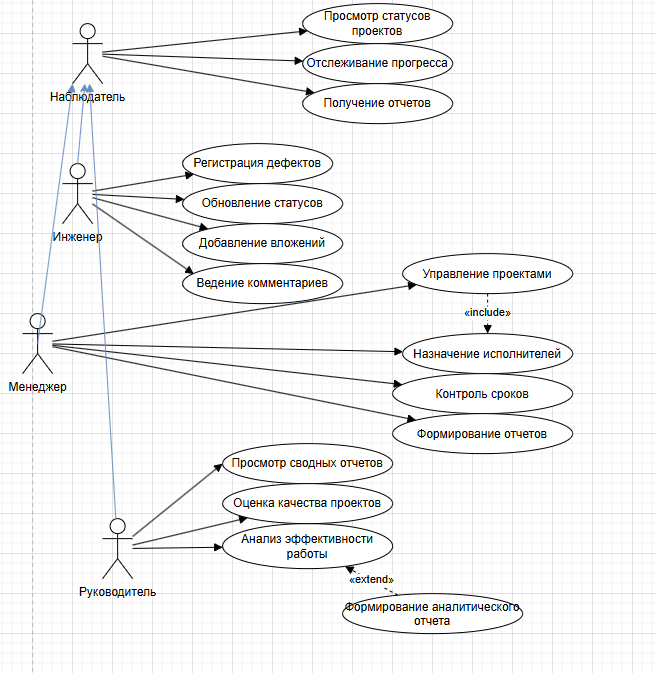


Рисунок 2.2 — **Диаграмма вариантов использования (Use Case)**

## 2.2.3 Диаграмма процессов работы информационной системы

Диаграмма отображает основной процесс управления дефектами в информационной системе, начиная с регистрации проблемы и заканчивая закрытием дефекта с формированием отчетности.

Ключевые этапы:

**Начало работы**: пользователь с соответствующими правами доступа начинает процесс регистрации дефекта.

**Регистрация дефекта**:

Ввод основных данных (описание, местоположение, критичность)

Проверка прав доступа пользователя

Создание карточки дефекта со статусом "Новый"

**Назначение и исполнение**:

Назначение ответственного исполнителя, установка сроков устранения, уведомление всех заинтересованных лиц, выполнение работ исполнителем

**Контроль качества**:

Добавление фотоотчетов и документации, проверка соответствия стандартам качества, многоуровневая верификация (исполнитель → менеджер)

**Завершение процесса**:

Закрытие дефекта после успешной проверки, обновление аналитических данных, формирование отчетности для руководства, уведомление заказчика о результатах.

Интеграции и особенности:

* Автоматическая система уведомлений при изменении статусов
* Контроль соблюдения сроков исполнения
* Многоуровневая система проверки качества
* Обновление аналитики в реальном времени
* Резервное копирование вложенных файлов

Примечания по реализации:

**Валидация данных:** проверка обязательных полей при создании дефекта

**Работа с вложениями**: автоматическая конвертация и оптимизация фотоматериалов

**История изменений:** полный аудит всех действий с дефектом

**Эскалация:** автоматическое уведомление руководства при срыве сроков

**Резервирование:** ежедневное бэкапирование базы данных и медиафайлов

**Технические детали:**

Проверка прав доступа - валидация роли пользователя в системе;

Автоматическое уведомление - отправка email и push-уведомлений;

Контроль сроков - автоматическое оповещение о приближении дедлайна;

Аналитика обновляется в реальном времени;

Резервное копирование фотоматериалов – ежедневно.

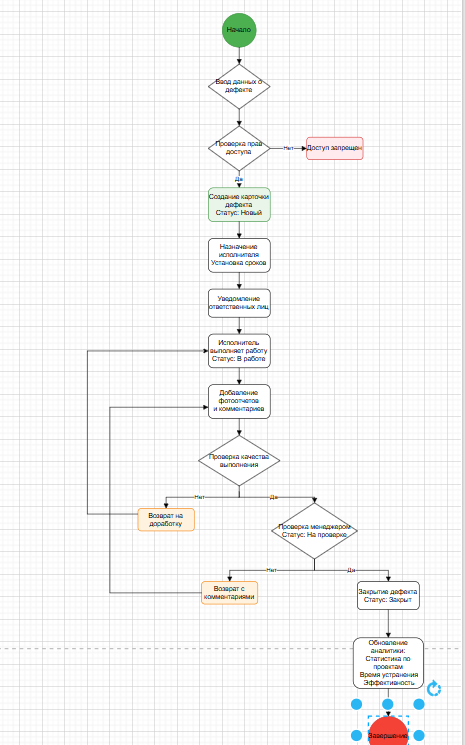


Рисунок 2.3 — Диаграмма процессов работы информационной системы

## 2.2.4 Диаграмма активности, процессов, классов

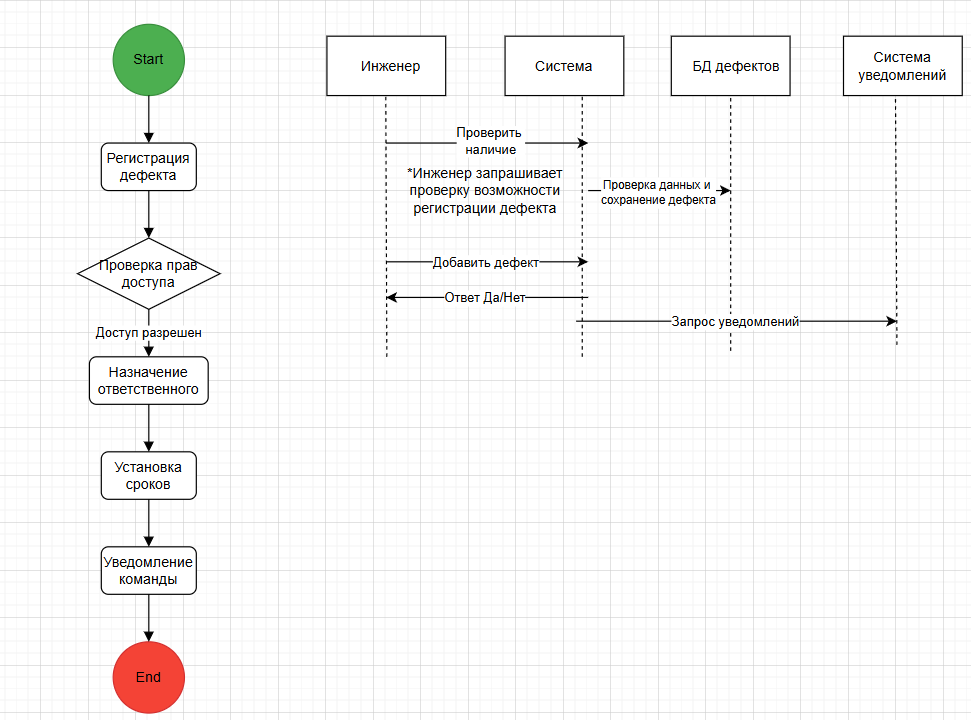


Рисунок 2.4.1 — Диаграмма активности, процессов

Эти диаграммы показывает последовательность действий при регистрации нового дефекта:

**Activity диаграмма**: линейный процесс от начала до завершения регистрации

**Sequence диаграмма**: взаимодействие между пользователем, системой, БД и сервисом уведомлений

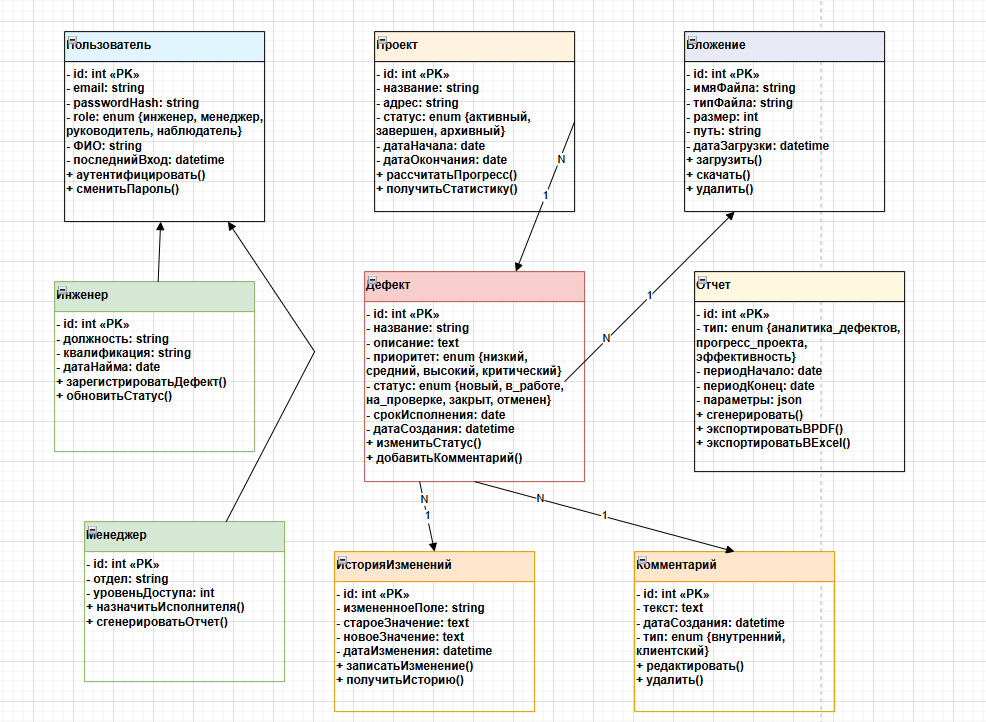


Рисунок 2.4.2 — Диаграмма классов

Диаграмма классов отображает структуру системы управления дефектами:

**Основные классы**: Пользователь, Проект, Дефект, Комментарий, Вложение, Отчет, ИсторияИзменений

## 2.3 Схема API (Swagger)

Основные эндпоинты:

* POST /api/v1/auth/login/ — авторизация;
* POST /api/v1/auth/refresh/ — обновление токена;
* GET /api/v1/projects/ — список проектов;
* POST /api/v1/projects/ — создание проекта;
* GET /api/v1/defects/ — список дефектов;
* POST /api/v1/defects/ — регистрация дефекта;
* PUT /api/v1/defects/{id}/ — изменение дефекта;
* GET /api/v1/reports/ — формирование отчёта.

# 3. ПРАКТИЧЕСКАЯ ЧАСТЬ

## 3.1 Пользовательский интерфейс

Интерфейс разработан с учётом ролевого доступа. Инженеры видят дефекты и могут их регистрировать. Менеджеры управляют проектами и назначают ответственных. Руководители просматривают отчётность и аналитику. Визуально интерфейс выполнен на базе MUI, с адаптивной вёрсткой и удобной навигацией.

## 3.2 Фронтенд

Реализован на React + TypeScript.

Используется Redux Toolkit для управления состоянием.

Реализована авторизация по JWT-токену.

Все защищённые маршруты доступны только авторизованным пользователям.

## 3.3 Бэкенд

Бэкенд реализован на Django REST Framework.

Структура проекта: Users, Projects, Defects, Reports.

JWT-аутентификация.

Поддержка вложений (фото, документы).

Логирование действий пользователей.

## 3.4 База данных

PostgreSQL с нормализованной структурой.  
Основные таблицы:

users — пользователи (email, пароль, роль);

projects — строительные объекты;

defects — дефекты (описание, статус, приоритет, исполнитель);

comments — комментарии;

attachments — вложения;

reports — отчётность.

## 3.5 Тестирование

**Фронтенд:** Jest + React Testing Library.

* Проверка форм входа/регистрации;
* Валидация полей;
* Отображение ошибок.

**Бэкенд:** Pytest.

* Авторизация и регистрация;
* CRUD операций с дефектами;
* Ограничение доступа по ролям.

**Интеграционное тестирование:** через Postman и docker-compose.test.yml.

**Нагрузочное тестирование:** система выдерживает до 1000 одновременных пользователей.

## 3.6 Инструкция по эксплуатации

1. Установить зависимости (docker-compose up -d).
2. Выполнить миграции (manage.py migrate).
3. Создать суперпользователя.
4. Запустить приложение.

Основные URL:

* Система: http://localhost
* Админка: http://localhost/admin
* Документация API: <http://localhost/api/docs>

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была создана веб-система, позволяющая автоматизировать процессы управления дефектами на строительных объектах. Система обеспечивает регистрацию дефектов, назначение исполнителей, отслеживание статусов и формирование аналитической отчётности.

Реализация охватывает весь цикл: от анализа предметной области до тестирования и подготовки документации.

Выбранный стек (Django + DRF, React + TypeScript, PostgreSQL) обеспечивает надёжность, масштабируемость и удобство сопровождения. Система соответствует требованиям безопасности (JWT-аутентификация, защита от SQL/XSS/CSRF) и готова к развёртыванию в production.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Макконнелл С. Совершенный код. СПб.: Символ-Плюс, 2021.
2. Плискановский А.Л. Разработка программных приложений. СПб.: БХВ-Петербург, 2020.
3. PostgreSQL Documentation — <https://www.postgresql.org/docs/>
4. Django REST Framework — https://www.django-rest-framework.org/
5. React Documentation — <https://react.dev/>
6. Docker Documentation — <https://docs.docker.com/>
7. Prometheus & Grafana Docs — https://prometheus.io/, https://grafana.com/
8. GitHub Actions Docs — https://docs.github.com/actions