Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Кафедра Математического обеспечения и суперкомпьютерных технологий

УЧЕБНЫЙ КУРС

«Проектирование и архитектура программных систем»

для подготовки по направлению «Программная инженерия»

ПОСТПРОЕКТНЫЙ АНАЛИЗ

Нижний Новгород
2025

Содержание

Введение

Постпроектный анализ проведен по итогам разработки веб-приложения для управления задачами и проектами в рамках учебного курса «Проектирование и архитектура программных систем».

Цель — обобщить опыт команды, оценить успехи и трудности, сформулировать выводы для будущих проектов. Анализ охватывает все этапы: от анализа требований до внедрения и тестирования. Проведение подобного анализа — важная практика для сплочения команды и повышения зрелости процессов разработки.

1 Достижения

1.1 Анализ

1. Четко сформулированы необходимость (централизация управления задачами для удаленных команд) и видение (веб-приложение с тремя ролями).
2. Проведен анализ выгод: централизация коммуникации, отечественная разработка, прозрачность выполнения задач, опыт команды.
3. Успешно определены и задокументированы границы проекта (scope) с явным указанием того, что остается за рамками (мессенджер, личные заметки, глобальный список задач).
4. Разработана и утверждена матрица компромиссов: время — фиксировано, возможности — согласуются, ресурсы — принимаются. Это помогло принимать взвешенные решения по ходу проекта.

1.2 Планирование

1. Разработан реалистичный план-график на 4 месяца с четкими вехами:
   - утверждение технического проектирования,
   - утверждение дизайна и архитектуры,
   - завершение разработки и готовность к тестированию,
   - успешный релиз и сдача проекта.
2. Четко распределены роли и ответственности (менеджер, архитектор, backend/frontend-разработчики, дизайнер, тестировщик, технический писатель).
3. Составлена смета проекта, учитывающая трудозатраты, ПО, оборудование и непредвиденные расходы.

1.3 Проектирование и архитектура

1. Выбрана и обоснована современная технологическая платформа:
   - frontend: React SPA с использованием Fetch API и React Router,
   - backend: Python + FastAPI (высокая производительность, автодокументация OpenAPI),

- база данных: PostgreSQL (надежность, транзакции, масштабируемость),
- контейнеризация: Docker, Docker Compose (гарантия идентичности окружений).

2. Разработана логическая модель данных (диаграмма классов), включающая сущности: пользователи, проекты, задачи, комментарии, встречи, уведомления.
3. Создана стратегия архитектурного дизайна (клиент-серверная архитектура) и стратегия технологического дизайна.
4. Разработаны UI/UX-прототипы всех интерфейсов в Figma, утвержденные заказчиком.

1.4 Разработка

1. Полностью реализованы три ролевых интерфейса с разделением функциональности:
   - Администратор: управление пользователями, справочниками, резервное копирование,
   - Менеджер: управление проектами, задачами, релизами, календарем, получение уведомлений,
   - Исполнитель: работа с задачами (изменение статусов, комментарии), просмотр календаря.
2. Реализована система аутентификации и авторизации на основе JWT с разграничением прав.
3. Настроен CI/CD-пайплайн (GitHub Actions/GitLab CI) для автоматической сборки, тестирования и развертывания.
4. Разработана полная API-документация с использованием Swagger/OpenAPI.

1.5 Тестирование

1. Разработаны детальные спецификации и сценарии тестов, включающие:
   - функциональное тестирование (создание задачи, изменение статуса).
2. Установлены четкие критерии производительности:
   - создание задачи — $\leqslant 0.5$ с,
   - изменение статуса задачи — $\leqslant 1$ с,

- фильтрация списка из 1000 задач — ⩽ 3 с,
- отклик интерфейса — ⩽ 400 мс.

3. Проведено успешное приемочное тестирование, подтвердившее соответствие всем критериям одобрения.

1.6 Внедрение и документирование

1. Разработан детальный план развертывания (Deployment Playbook) с процедурой отката (Rollback Plan).
2. Подготовлен комплект документации:
   - техническая: концепция, функциональная спецификация, структура проекта,
   - пользовательская: руководства для Администратора, Менеджера, Исполнителя,
   - инструкции по развертыванию (README.md, docker-compose).

2 Проблемы и ошибки

2.1 Анализ

1. Недооценка сложности интеграции модулей: изначально не в полной мере учтена сложность синхронизации данных между модулем задач и календарем, особенно в сценариях изменения статусов и дедлайнов. Это потребовало дополнительного времени на проектирование и отладку.
2. Неполный анализ смежных процессов: например, процесс отображения отпусков в календаре был проработан недостаточно детально на старте, что привело к доработкам на этапе разработки.

2.2 Планирование

1. Оптимистичная оценка трудозатрат на инфраструктуру: время на настройку CI/CD, Docker-образов и тестовых сред было заложено меньше, чем потребовалось в реальности.
2. Недостаточная детализация коммуникационных протоколов: первоначально не были четко прописаны форматы и частоту синхронизации между frontend и backend-разработчиками, что иногда приводило к задержкам в интеграции.
3. Слабая проработка рисков, связанных с внешними зависимостями: например, обновления ключевых библиотек (React, FastAPI).

2.3 Разработка

1. Отклонения от code style на ранних этапах: отсутствие строгого контроля за стилем кода в начале проекта привело к необходимости масштабного рефакторинга перед код-ревью и сдачей.
2. Трудности с организацией эффективного процесса код-ревью: из-за высокой загрузки разработчиков ревью иногда проводилось поверхностно, что позволило пропустить некоторые потенциальные уязвимости и архитектурные антипаттерны.

2.4 Тестирование

1. Недостаточное покрытие edge-кейсов в начальных тест-планах.

2. Задержки с подготовкой тестовых данных для нагрузочного тестирования: создание реалистичных наборов данных (10+ проектов, 30+ задач, календарные события) заняло больше времени, чем планировалось.
3. Ручное тестирование занимало много времени: не все сценарии были автоматизированы на раннем этапе, что замедляло регрессионное тестирование.

2.5 Внедрение и эксплуатация

1. Слабая документация по процедуре отката (Rollback): хотя план отката существовал, его шаги не были достаточно отработаны на практике, что могло привести к задержкам в критической ситуации.

3 Уроки

3.1 Анализ

1. Интеграционная карта: на будущее необходимо создавать интеграционную карту (integration map), наглядно показывающую все взаимодействия между модулями системы и потенциальные точки синхронизации данных.
2. Глубокий анализ смежных областей: при проектировании функциональности, затрагивающей несколько модулей (календарь – задачи), проводить совместные воркшопы с участием всех заинтересованных разработчиков и тестировщиков.

3.2 Планирование

1. Буфер на инфраструктуру: всегда закладывать отдельный временной буфер (10-15%) на настройку и отладку инструментов CI/CD, контейнеризации и развертывания.
2. Четкий коммуникационный план: с первого дня устанавливать регламент коммуникации: ежедневные стендапы, инструменты (VK, Яндекс Телемост), форматы отчетности и синхронизации между frontend/backend.
3. Упреждающее управление рисками: вести живой реестр рисков, особое внимание уделяя внешним зависимостям (версии ПО, библиотеки) и регулярно его актуализировать.

3.3 Разработка

1. Shift-left для качества: внедрять и строго соблюдать стандарты кодирования, проводить парное программирование на сложных участках и обязательное код-ревью для каждого PR/MR перед мерджем в основную ветку.
2. Feature flags (флаги функциональности): использовать feature flags для постепенного включения новой функциональности. Это позволяет безопасно тестировать новинки в продакшене на ограниченной аудитории и быстро откатывать при проблемах.
3. Прототипирование API на раннем этапе: создавать и согласовывать прототипы API (например, с помощью OpenAPI/Swagger) до начала активной frontend-разработки для минимизации miscommunication.

3.4 Тестирование

1. Тест-дизайн параллельно со спецификацией: начинать проектирование тест-кейсов одновременно с написанием функциональной спецификации. Это помогает выявить неоднозначности в требованиях на самом раннем этапе.
2. Приоритет автоматизации: с самого начала выделять ресурсы на написание автоматизированных end-to-end (E2E) тестов для ключевых пользовательских сценариев (создание проекта → задача → смена статуса). Это резко ускоряет регрессионное тестирование.
3. Реалистичные тестовые данные: разрабатывать скрипты для генерации реалистичных тестовых данных (с учетом связей между сущностями) одновременно с разработкой модели данных.

4 Выводы

1. Проект успешно достиг всех основных целей: разработано полнофункциональное и производительное веб-приложение для управления проектами, соответствующее всем утвержденным критериям приемки.
2. Выбранная архитектура и технологический стек (React + FastAPI + PostgreSQL + Docker) доказали свою эффективность для проектов подобного масштаба и сложности. Они обеспечили хорошую производительность, масштабируемость и удобство развертывания.
3. Качество итогового продукта напрямую зависит от качества процессов на ранних этапах. Четкий анализ, детальное планирование и строгое проектирование предотвратили множество потенциальных проблем на поздних стадиях.
4. Непрерывная коммуникация и дисциплина исполнения процессов (управление кодом, тестирование, интеграция) являются критически важными факторами успеха для распределенной команды.
5. Инвестиции в автоматизацию (CI/CD, тесты) и инфраструктуру (контейнеризация) окупаются на этапах тестирования и внедрения, значительно снижая риски и временные затраты.
6. Документирование — это не формальность, а активный инструмент управления проектом. Качественная документация (ТЗ, спецификации, руководства) служила единым источником истины для всей команды и заказчика.

Данный проект стал ценным практическим опытом для всех участников, позволив применить теоретические знания в области проектирования архитектуры, разработки полного стека, тестирования и проектного менеджмента в условиях, близких к реальным.

Рекомендация для будущих проектов: внедрить более формализованную систему сбора и анализа метрик проекта (velocity, количество дефектов, время на код-ревью) для более объективной оценки прогресса и своевременного выявления отклонений.