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»

СТРУКТУРА ПРОЕКТА

Нижний Новгород
2025

Содержание

1 В документе использованы материалы белых книг (white papers) “MSF Process Model”, “MSF Risk Management Discipline”, “MSF Team Model” (http://www.microsoft.com/msf), их переводов “Модель процессов MSF”, “Дисциплина управления рисками MSF”, “Модель проектной группы MSF” выполненных в 2003 году корпораций eLine Software (http://www.elinesoftware.com), а также официальных курсов Microsoft 2710B и 1846A.

1 Рамки проекта

1.1 Матрица компромиссов проекта

1.1.1 Треугольник компромиссов:

Ресурсы:

- команда: 1 менеджер проекта, 1 backend-разработчик, 1 frontend-разработчик, 1 дизайнер, 1 тестировщик, 1 технический писатель,
- оборудование: 6 ноутбуков с выходом в Интернет и необходимыми средами разработки и тестирования и приложениями для ведения документации.

Время:

Длительность проекта: 4 месяца.

- месяц 1: проектирование,
- месяц 2: разработка дизайна и архитектуры,
- месяц 3: разработка backend и frontend,
- месяц 4: тестирование и внедрение.

Возможности:

- управление проектами и задачами,
- система статусов и комментариев,
- календарь встреч,
- чат Исполнителей и Менеджеров,
- безопасность, аутентификация и администрирование.

1.1.2 Матрица компромиссов

Так как основная ценность проекта – это контроль и управление процессом со стороны менеджера, поэтому матрица компромиссов (таблица 1) выглядит так:

Таблица 1 – Треугольник компромиссов

	Фиксируется	Согласовывается	Принимается
Ресурсы			✓
Время	✓		
Возможности		✓	

Время фиксируется, так как рынок инструментов для удаленных команд динамичен, так что задержка с выпуском приложения может привести к потере преимущества перед конкурентами.

Возможности согласовываются, так как функциональность проекта может изменяться по согласованию с заказчиком, чтобы уложиться в сроки.

Ресурсы принимаются, так как мы готовы принять их для реализации возможностей в заданное время. Ресурсы - гибкий элемент в данной конфигурации.

1.1.3 Вехи проекта

Подход к определению вех в проекте

Для данного проекта будет использован **комбинированный подход**, основанный на:

- Завершении ключевых фаз жизненного цикла проекта. Это классический и наиболее наглядный подход, который позволяет отчитываться перед заинтересованными сторонами о переходе проекта на новый этап.
- Достижении основных содержательных результатов (Major Deliverables). Вехой считается момент, когда готов и одобрен конкретный, измеримый и значимый результат работы (например, готовый прототип или стабильная версия для тестирования).
- Критических точках принятия решений («Go/No-Go»). Некоторые вехи будут маркировать моменты, после которых проект либо продолжается с новыми ресурсами, либо пересматривается. Это точки синхронизации с заказчиком.

Такой подход идеально подходит для проекта, так как он делает прогресс осязаемым как для команды, так и для заказчика. Он фокусируется не на затратах или произвольных временных интервалах, а на реальных, работающих артефактах, что снижает субъективность в оценке прогресса и четко определяет момент достижения вехи.

Вехи проекта:

1. Утверждение технического проектирования

    1.1 Полностью готов и утвержден заказчиком документы «Концепция проекта», «Структура проекта» и «Функциональная спецификация», содержащий детальное описание всех функциональных и нефункциональных требований,

сценарии использования для трех интерфейсов (Администратора, Менеджера и Исполнителя).

1.2 Получено формальное разрешение («Go») на переход к фазе разработки дизайна и архитектуры.

2. Утверждение дизайна и архитектуры

2.1 Дизайн: утверждены полные интерактивные UI/UX прототипы всех экранов, дизайн-система и визуальный стиль.

2.2 Архитектура: утверждены документы с технической архитектурой системы (бэкенд-архитектура, схема базы данных, API спецификация).

2.3 Получено разрешение («Go») на начало фазы активной разработки (кодирования).

3. Завершение разработки и готовность к тестированию

3.1 Бэкенд: все серверные модули реализованы, API полностью готово и функционирует.

3.2 Фронтенд: все графические интерфейсы реализованы и интегрированы с бэкендом.

3.3 Сборка: приложение развернуто в тестовой среде. Создана первая интеграционная сборка (Alpha-версия), доступная для тестирования.

3.4 Значение: кодовая база проекта завершена. Продукт функционален, но может содержать ошибки.

4. Успешный релиз и сдача проекта

4.1 Тестирование: проведено полное тестирование, все критические ошибки исправлены. Составлен и утвержден отчет о качестве.

4.2 Внедрение: приложение развернуто на рабочем сервере (продакшн-среде). Проведено обучение пилотной группы пользователей, продукт передан в эксплуатацию.

4.3 Документация: сформирована финальная пользовательская и техническая документация.

4.4 Значение: финальная цель проекта достигнута. Продукт работает стабильно и готов к использованию.

1.2 Сметы проекта

Список ресурсов:

1. Команда проекта (таблица 2):

Таблица 2 – Команда проекта

Должность	Тариф (в месяц), у.е.	Общая стоимость
Менеджер проекта	700	2800
Backend-разработчик	900	3600
Frontend-разработчик	850	3400
Дизайнер	800	3200
Тестировщик	750	3000
Технический писатель	650	2600
Итого	4650	18600

2. Программное и аппаратное обеспечение (таблица 3):

Таблица 3 – Программное и аппаратное обеспечение

Ресурс	Количество	Тариф, у.е. за 1 шт	Общая стоимость, у.е.	Примечание
Виртуальный сервер	1	100	100	Для разработки, тестирования и пилотного внедрения
Лицензии ПО	5	50	1000	Лицензии на среду разработки, графические редакторы
Подписка на GitHub/GitLab Pro	1	20	20	Для приватных репозиториев и CI/CD
Итого:		370	1120	

3. Прочие расходы (накладные и непредвиденные):

Непредвиденные расходы (10% от ФОТ) - 2300 у.е.

1.3 План-график проекта

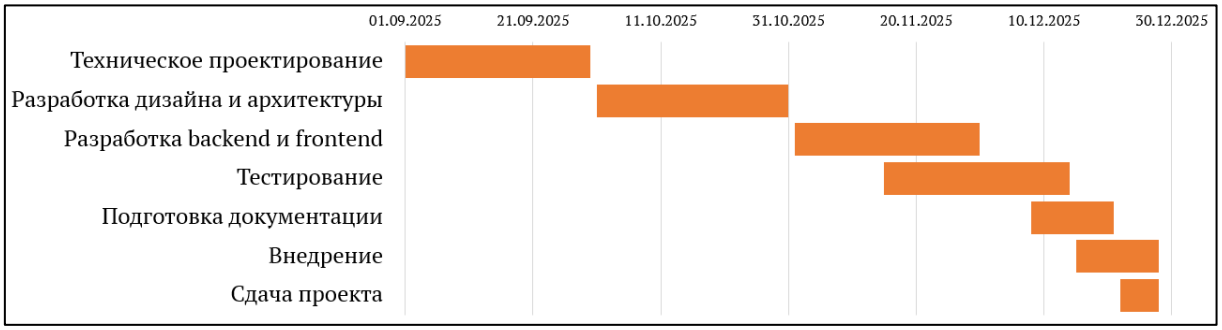

Рисунок 1 – План-график проекта

2 Роли и ответственности

2.1 Знания, умения и навыки

Таблица 4 – Необходимые знания, умения и навыки

Роль	Знания	Навыки	Умения
Менеджер проекта	Методологии управления проектами (Agile, Waterfall), основы бюджетирования и планирования	Организованность, лидерство, тайм-менеджмент, навыки ведения переговоров и презентаций	Составление и контроль плана-графика, управление рисками, ведение коммуникации с заказчиком и командой, решение конфликтных ситуаций
Архитектор проекта	Проектирование системной архитектуры, технологического стека	Коммуникация и лидерство, принятие решений и компромиссов, стратегическое мышление	Технический анализ требований, умение создавать понятную и исчерпывающую архитектурную документацию
Backend-разработчик	Python 3.x, фреймворк FastAPI, RESTful API, реляционные базы данных (SQL, PostgreSQL), ORM (SQLAlchemy), система миграций (Alembic), аутентификация (JWT)	Структурное и логическое мышление, навыки отладки и оптимизации кода	Проектирование API, разработка бизнес-логики, работа с базами данных, написание unit-тестов, контейнеризация (Docker)
Frontend-разработчик	JavaScript/TypeScript, фреймворк React, маршрутизация (React Router), HTTP-клиенты (Fetch API), HTML5, CSS3	Внимание к деталям, понимание UX-принципов	Создание адаптивных и интуитивно понятных пользовательских интерфейсов, интеграция с бэкенд-API, обеспечение кросс-браузерной совместимости
Дизайнер (UI/UX)	Принципы UI/UX дизайна, инструменты проектирования (Figma, Adobe XD), основы типографики и цветоведения	Креативность, понимание психологии восприятия, работа с фидбеком	Создание wireframes, интерактивных прототипов, дизайн-систем, адаптация дизайна под разные устройства
Тестировщик (QA-инженер)	Теория тестирования ПО, виды и уровни тестирования, инструменты для тестирования	Аналитическое мышление, внимательность, системный подход	Составление тест-планов и тест-кейсов, проведение функционального, регрессионного и usability-тестирования, документирование дефектов
Технический писатель	Принципы технического письма, нотации (UML, BPMN), инструменты (MS Word, Confluence, Markdown)	Грамотность, способность быстро осваивать новые предметные области	Структурирование и ясное изложение сложной информации, написание пользовательской и

Роль	Знания	Навыки	Умения
			технической документации

2.2 Структура команды

Таблица 5 – Структура комманды

Фамилия Имя члена команды	Роль в команде	Зона ответственности
Садиков Иван	Менеджер проекта Архитектор проекта	• Взаимодействие членов команды • Контроль за соблюдением сроков • Одобрение этапов разработки (Go/No-Go) • Разработка архитектуры приложения
Сидорина Полина	Дизайнер (UI/UX)	• Проектирование пользовательского опыта (UX) • Разработка пользовательского интерфейса (UI) • Создание интерактивных прототипов • Подготовка материалов для разработки • Контроль визуального качества
Зайцев Артем	Backend-разработчик	• Разработка серверной логики • Проектирование и реализация API • Разработка базы данных

Фамилия Имя члена команды	Роль в команде	Зона ответственности
		• Интеграция внешних сервисов • Безопасность backend
Гнитиенко Кирилл	Frontend-разработчик	• Разработка клиентской части • Интеграция с backend • Пользовательский опыт • Адаптивность • Клиентская валидация данных
Калякина Анастасия	Тестировщик (QA-инженер)	• Тест-анализ и планирование • Создание тестовой документации • Функциональное тестирование • Регрессионное тестирование • Составление баг-репортов • Приемочное тестирование
Ведерникова Ксения	Технический писатель	• Разработка технического задания • Написание пользовательской документации • Ведение и актуализация проектной документации

3 Протоколы проекта

3.1 Управление конфигурацией

Методы и средства: для контроля версий используется Git. Основной репозиторий размещен на GitHub/GitLab. Применяется модель ветвления GitFlow.

Процесс изменений: все изменения в код вносятся через Merge/Pull Request. Для слияния ветки в основную (main) требуется минимум три апрува: от создателя Merge/Pull Request, от руководителя команды (команды разработчиков, документирования или дизайна), от менеджера проекта. Редактирование и хранение документации на этапе разработки производится в Google Документы, хранение завершенной документации – в основном репозитории проекта в GitHub.

Роли и ответственности: разработчики отвечают за создание веток и MR/PR. Менеджер проекта или тимлид выполняет код-ревью и мердж.

Требования к системе контроля версий: обязательное описание коммитов и MR/PR, использование тегов для релизов.

3.2 Управление изменениями

Процесс: все запросы на изменение оформляются в единой системе (например, в Issue Tracker в GitLab). Запрос должен содержать описание изменения, обоснование и оценку влияния на сроки/бюджет/функциональность.

Форма запроса: стандартизированная Issue с пометкой "Change Request".

Роли и ответственности: менеджер проекта анализирует запрос, консультируется с командой и принимает решение об утверждении/отклонении с учетом Матрицы компромиссов. Критические изменения согласуются с заказчиком.

Влияние на контракт: изменения, влияющие на стоимость или сроки, подлежат обязательному согласованию с заказчиком и formal change order.

3.3 Управление внедрениями

Основная цель процесса внедрения — обеспечить плавный, контролируемый и безопасный переход от разработанного решения к его активному использованию заказчиком.

Процесс разделен на этапы, начиная с пилотного внедрения и заканчивая полным развертыванием.

1. Подготовка к внедрению

1.1 Формирование и тестирование релизных сборок:

1.1.1 Ведение релизного ветвления в Git: Использование модели GitFlow или аналогичной. Ветка main/master содержит только стабильные релизные версии. Ветка develop — код для следующего релиза. Релизные кандидаты помечаются тегами (например, v1.0.0-rc.1).

1.1.2 Автоматизация сборки и деплоя (CI/CD): При пуше кода в ветки develop и main автоматически запускается пайплайн в GitHub Actions/GitLab CI.

1.1.3 Чек-лист подготовки релиза.

1.2 Подготовка инфраструктуры:

1.2.1 Продакшен-окружение: Развертывание и предрелизная настройка серверов в соответствии с инфраструктурной документацией. Все изменения выполняются с помощью инструментов Infrastructure as Code (Terraform, Ansible) для обеспечения повторяемости.

1.2.2 Резервное копирование: Гарантия наличия актуальных бэкапов продакшен-базы данных и файлового хранилища перед началом развертывания.

1.3 Подготовка пользователей и документации:

1.3.1 Финальные пользовательские инструкции: Технический писатель готовит и согласовывает с заказчиком итоговые руководства пользователя для Администратора, Менеджеров и Исполнителей.

1.3.2 План коммуникации: Менеджер проекта разрабатывает план информирования будущих пользователей о предстоящем запуске, сроках и обучающих мероприятиях.

2. Стратегия внедрения: Поэтапный запуск (Pilot & Phased Rollout)

Для минимизации рисков используется стратегия поэтапного внедрения.

2.1 Пилотное внедрение (Pilot Launch) — Месяц 4:

2.1.1 Цель: Проверить работу системы в реальных условиях на ограниченной группе, выявить скрытые проблемы и собрать обратную связь перед полным запуском.

2.1.2 Масштаб: Внедрение для одной-двух команд компании-заказчика (например, 10-20 пользоваников).

2.1.3 Процесс:

2.1.3.1 Развертывание: Выпуск версии v1.0.0 на продакшен-сервер, но с ограниченным доступом только для пилотных групп.

2.1.3.2 Обучение: Проведение обучающих вебинаров для пилотных пользователей.

2.1.3.3 Поддержка: Назначение "точки контакта" из команды проекта для оперативного сбора обратной связи и решения проблем.

2.1.3.4 Сбор метрик: Мониторинг использования ключевых функций, производительности и ошибок (через Sentry).

2.2 Финальное внедрение (Full Rollout):

2.2.1 Цель: Полномасштабный запуск приложения для всех целевых пользователей.

2.2.2 Условие для старта: Успешное завершение пилотной фазы, подтвержденное подписанным Актом о успешном пилотном внедрении и отсутствием критических проблем в течение 1-2 недель.

2.2.3 Процесс:

2.2.3.1 Релиз финальной версии: Выпуск версии v1.0.0 (или v1.0.1 с учетом правок по фидбеку пилота) для всех сотрудников.

2.2.3.2 Открытие каналов поддержки: Формализация процесса обращения в поддержку для конечных пользователей.

3. Процедура развертывания (Deployment Playbook)

3.1 Четкий регламент для команды на день релиза.

3.2 День Релиза (R-Day):

3.2.1 Окно для развертывания: Назначается на вечернее/ночное время (например, с 22:00 до 02:00) для минимизации impact на пользователей.

3.2.2 Команда: Присутствуют Backend-разработчик, Frontend-разработчик, Менеджер проекта. Все на связи в Яндекс Телемосте.

3.2.3 Этапы:

3.2.3.1 Объявление начала работ. Менеджер проекта.

3.2.3.2 Остановка продакшен-приложения.

3.2.3.3 Выполнение скриптов миграции БД. (Только при наличии).

3.2.3.4 Развертывание новых версий бэкенда и фронтенда через CI/CD.

3.2.3.5 Дымовое тестирование (Smoke Test): Быстрая проверка ключевых сценариев (вход, создание задачи, смена статуса) непосредственно на продакшене.

3.2.3.6 Подтверждение успешности. Ответственные разработчики.

3.2.3.7 Объявление об окончании работ. Менеджер проекта.

4. Откат (Rollback Plan)

4.1 На случай, если после развертывания обнаружится критическая ошибка.

4.2 Критерии для отката: Падение производительности, недоступность ключевых функций, потеря/порча данных.

4.3 Процедура:

4.3.1 Принятие решения об откате Менеджером проекта по согласованию с Заказчиком.

4.3.2 Быстрый откат: Развертывание предыдущей стабильной версии приложения из заранее подготовленного Docker-образа.

4.3.3 Откат миграций БД: (Если применялись) выполнение откатывающих SQL-скриптов.

4.3.4 Информирование пользователей: Уведомление о временных неполадках и завершении работ.

5. Пострелизные мероприятия

5.1 Мониторинг: Усиленный мониторинг метрик производительности и ошибок в течение 72 часов после релиза.

5.2 Пост-мортем (Post-mortem): Проведение встречи по итогам релиза в течение недели. Обсуждение успехов, проблем и извлеченных уроков для улучшения процесса следующих внедрений.

5.3 Передача документации: Полный пакет документов (архитектура, API, руководства) передается заказчику и/или его ИТ-отделу для дальнейшего сопровождения.

3.4 Достижение качества проекта

1. Ожидания к качеству:
   - полная реализация заявленной функциональности,
   - стабильная работа системы без критических сбоев,
   - время отклика интерфейса не более 400 мс при скорости интернета от 100 Мбит/с,
   - интуитивно понятный пользовательский интерфейс,
   - соответствие техническим требованиям к развертыванию.
2. Процесс проверки качества:
   - регулярное проведение код-ревью,
   - модульное и интеграционное тестирование на этапе разработки,
   - регрессионное и приемочное тестирование перед релизом,
   - проверка соответствия критериям приемки.
3. Управление качеством:
   - еженедельные встречи по оценке качества,
   - мониторинг выполнения стандартов качества,
   - контроль соблюдения процессов тестирования,
   - анализ метрик качества проекта.
4. Распределение ответственности:
   - разработчики: качество реализуемого кода,

- тестировщик: независимая верификация и валидация,
- менеджер проекта: общее соответствие критериям приемки,
- все участники команды: соблюдение процессов обеспечения качества.

5. Используемые инструменты для разработки:
   - архитектурный стиль «клиент-сервер» обеспечивает централизованность, масштабируемость и высокую безопасность, а также упрощает обслуживание и обновление.
   - паттерн «Model-View-Controller» позволяет разделить задачи, что повышает читабельность и тестируемость и упрощает совместную разработку.
   - backend: в преимуществах фреймворка FastAPI можно выделить высокую производительность, быструю разработку и автоматическую генерацию документации,
   - frontend: SPA на React обеспечивает быструю и плавную навигацию без перегрузки страницы и улучшенный пользовательский опыт (UX), меньшую нагрузку на сервер,
   - база данных PostgreSQL – для надежного хранения данных о пользователях, проектах, задачах, комментариях и событиях календаря,
   - figma выбрана для разработки дизайна из-за понятного интерфейса, быстродействия, возможности прототипирования и гибкости фреймов,
   - документация ведется в Google Документы, так как этот сервис предоставляет возможность совместного редактирования, удобное форматирование и облачное хранение.

3.5 Рабочая среда проекта

Требования к рабочей среде проекта (Outsource)

1. Организационные требования. Размещение команды и рабочие места:
   - Команда работает удалённо из своих локаций.
   - Каждый член команды обязан обеспечить себе стационарное рабочее место с высокоскоростным и стабильным интернет-соединением (рекомендуется не менее 10 Мбит/с на загрузку и отдачу).

- Для проведения митингов требуется тихое, закрытое помещение без фоновых шумов и отвлекающих факторов.

2. Требования к аппаратному обеспечению (Hardware)

3. Требования к программному обеспечению и инструментам (Software & Tools):

3.1 Управление проектом и коммуникация:

- Коммуникация (основной канал): VK. Создаются отдельные каналы для общих обсуждений, разработки, дизайна, тестирования и срочных вопросов.
- Видосвязь и ежедневные митинги: Яндекс Телемост.
- Документация и знания: Google Диск. Единое пространство для хранения ТЗ, протоколов встреч, решений по архитектуре, пользовательских инструкций.

3.2 Разработка и дизайн:

- Система контроля версий: Git. Основной репозиторий размещается на GitHub.com, GitLab.com или Bitbucket.org (приватный).
- Среда разработки (IDE): На усмотрение разработчика (VS Code, IntelliJ IDEA, WebStorm и т.д.), но с обязательным использованием единых конфигураций и код-стайла для проекта.
- Дизайн и прототипирование: Figma. Все макеты, стили и прототипы должны быть размещены в едином Figma-файле/проекте с открытым доступом для всей команды.
- Непрерывная интеграция/непрерывное развёртывание (CI/CD): Использование встроенных в GitHub/GitLab инструментов (GitHub Actions, GitLab CI) для автоматической сборки, запуска тестов и развёртывания на тестовые стенды.

3.3 Тестирование:

- Управление тест-кейсами и багами: Jira (встроенные возможности) или TestRail.
- Автотесты: Jest / Playwright .

3.4 3.4. Безопасность:

- Двухфакторная аутентификация (2FA) должна быть включена для всех корпоративных аккаунтов (GitHub, Jira, Figma, почта).

4. Требования к тестовым и рабочим средам (Environments)

- Среда разработки (Development): Локальные машины разработчиков.
- Интеграционная/тестовая среда (Staging): Выделенный виртуальный сервер, максимально приближенный к продакшен-среде. Используется для тестирования и демонстрации функционала заказчику. Доступ по VPN.
- Производственная среда (Production): Сервер заказчика или облачная инфраструктура (например, Yandex Cloud, Selectel). Доступ строго регламентирован.