МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Создание автоматизации управления проектами в условиях многозадачности с интеграцией данных о функционале и договорах (на примере АО «Эр-Стайл СофтЛаб»)»

Обучающийся М.И. Галкина _____
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Название выпускной квалификационной работы Создание автоматизации управления проектами в условиях многозадачности с интеграцией данных о функционале и договорах (на примере АО «Эр-Стайл СофтЛаб»)».

Целью данной ВКР является разработка веб-приложения для упрощения управления проектами в условиях многозадачности, обеспечивающего централизованное хранение данных по проектам, задачам и договорной документации и отвечающего требованиям пользователей по удобству, стабильности и функциональности.

Объектом ВКР выступают процессы управления проектами и требования к IT-инструментам в многозадачной среде.

Предметом ВКР выступают архитектурные решения и функциональные возможности разрабатываемого веб-приложения.

Ключевым вопросом ВКР является определение оптимальных подходов к проектированию надёжного и удобного веб-решения для многозадачного управления проектами с учётом требований безопасности и пользовательского опыта.

В первой главе рассматриваются программы для управления проектами, выявляются их недостатки и проводится сравнение.

Во второй главе описано проектирование и создание приложения: основные компоненты, структура данных, архитектура и соответствующие схемы.

В третьей главе описаны различные тесты (модульные, интеграционные, функциональные и нагрузочные) и представлены их результаты.

В заключении подтверждается достижение цели ВКР: разработанное веб-приложение готово к внедрению, способно повысить эффективность управления проектами, снизить риски фрагментации информации и сократить временные затраты на согласование и контроль.

Abstract

The result of the final qualification work is a web application for project management automation under multitasking conditions, providing centralized storage of data on projects, tasks and contractual documentation.

The introduction substantiates the relevance of creating a specialized project management tool against the background of growing requirements for information security and the need for in-house IT solutions.

The aim of the work is to develop a web application that will help simplify project management in multitasking conditions, provide convenient access to information about tasks, functionality and contracts, and meet the requirements of users in terms of convenience, stability and functionality.

The first chapter is analytical in nature. It reviews the existing methods and tools, reveals the problems of data fragmentation and difficulties with version management. Based on the analysis, the requirements to the new solution are formulated and a comparative analysis of analogs is carried out.

The second chapter presents the design and implementation of the application. The main entities are identified and a logical data model is developed. The three-level architecture “client - server - database” is designed. Diagrams of classes, use cases, activities, components and block diagrams of project and task creation are built.

The third chapter is devoted to approbation and testing. The methods of unit, integration, functional and load testing are described; the results of test runs and load scenario are given.

The goal of the final qualification work is achieved: the developed web-application is ready for implementation, it will increase the efficiency of project management, reduce the risks of information fragmentation and reduce the time spent on coordination and control.

Оглавление

Введение

Предметная область выпускной квалификационной работы - это автоматизация процессов управления проектной деятельностью в компании АО «Эр-Стайл СофтЛаб». Компания занимается разработкой и сопровождением программного обеспечения и сталкивается с необходимостью эффективного управления множеством проектов одновременно.

Ключевые процессы предметной области включают постановку и мониторинг задач, контроль выполнения работ, ведение документации (договоры, акты, протоколы тестирования) и учёт трудозатрат. До разработки веб-приложения эти процессы выполнялись с помощью разрозненных инструментов (Google Docs, CBH/SVN), что приводило к фрагментации данных, сложности управления версиями документов и низкой прозрачности рабочих процессов.

Проект охватывает весь жизненный цикл программного продукта: от этапов планирования и выработки архитектурных решений, через разработку и комплексное тестирование, до интеграции с данными о функциональных требованиях и договорной документации, а также финального развертывания и внедрения системы в корпоративную среду.

Объект исследования - процесс управления проектами в условиях многозадачности с интеграцией данных о функционале и договорах в компании, где параллельно ведется множество задач. Работа включает все этапы его жизненного цикла - от планирования и проектирования архитектуры системы до реализации, тестирования, интеграции с данными о функционале и договорах, а также последующего внедрения в корпоративную среду.

Предметом исследования является автоматизация процесса управления проектами в условиях многозадачности с интеграцией данных о функционале и договорах.

Цель работы - разработка веб-приложения, которое поможет упростить управление проектами в условиях многозадачности, обеспечит удобный доступ

к информации о задачах, функционале и договорах, а также будет отвечать требованиям пользователей по удобству, стабильности и функциональности.

Задачи работы:

- проанализировать потребность в автоматизации управления проектами в условиях многозадачности на примере АО «Эр-Стайл СофтЛаб»;
- изучить текущие процессы компании, связанные с постановкой задач, сопровождением функционала и документооборотом;
- исследовать существующие подходы и инструменты разработки веб-приложений для корпоративных информационных систем;
- сформулировать требования к функционалу, архитектуре и пользовательскому интерфейсу разрабатываемого веб-приложения;
- спроектировать структуру приложения с учётом интеграции данных о задачах, функционале и договорных документах;
- реализовать веб-приложение с использованием современных технологий;
- провести тестирование разработанного решения и оценить его соответствие требованиям компании и конечных пользователей.

Выпускная квалификационная работа включает введение, три главы и заключение. В первой главе проведён анализ существующих бизнес-процессов управления проектами в АО «Эр-Стайл СофтЛаб», выявлены их ограничения и сформулированы требования к веб-приложению. Во второй главе описано проектирование и реализация системы: спроектированы архитектура, модели данных и REST-API, а также разработана клиентская часть и серверная часть. В третьей главе представлены результаты апробации и тестирования.

Работа занимает 48 страниц, содержит 37 рисунков, 4 таблицы и 20 источников.

Глава 1 Постановка задачи для автоматизации управления проектами

1.1 Значение веб-приложения для автоматизации управления проектами

Современные предприятия сталкиваются с необходимостью управления большим количеством параллельных проектов, включая планирование, распределение задач, контроль сроков и документации. Особенно актуальной становится задача оптимизации этих процессов в условиях многозадачности, когда вовлечено множество участников и требуется оперативный обмен информацией.

Автоматизация управления проектами с использованием специализированных веб-приложений позволяет повысить прозрачность процессов, сократить количество рутинных операций, улучшить контроль за исполнением задач и повысить общую эффективность работы.

В условиях курса на импортозамещение и усиления требований к информационной безопасности многие российские компании стремятся отказаться от использования зарубежных программных решений в пользу собственных разработок. Такой подход позволяет не только снизить внешние риски, но и создавать продукты, максимально адаптированные под внутренние процессы и потребности.

АО «Эр-Стайл СофтЛаб» активно поддерживает эту стратегию, развивая внутренние IT-инструменты для автоматизации ключевых бизнес-процессов. Одним из примеров такой инициативы является Корпоративная система управления проектами (КСУП), созданная на базе Redmine и адаптированная под задачи контроля и ведения проектной деятельности.

Разрабатываемое в рамках данной выпускной квалификационной работы веб-приложение является самостоятельным решением и ориентировано на другую задачу - автоматизацию управления проектами в условиях

многозадачности с интеграцией информации о функционале и договорной документации.

Приложение позволяет структурировать данные по проектам, задачам и этапам согласования, обеспечивает централизованное хранение документов и упрощает взаимодействие между участниками процесса [6].

Новая система значительно упрощает процесс работы с проектами и задачами и позволяет пользователю осуществлять все действия в едином интерфейсе, а именно:

– осуществлять поиск проектов и задач в едином месте;
– просматривать и обновлять данные о проектах и задачах;
– работать с документами с помощью встроенного функционала без необходимости использования сторонних систем [17];
– завершать работу с автоматическим сохранением данных.

Представлена причинно-следственная связь в виде диаграммы Исикавы на рисунке 1, показывающая взаимосвязь между проблемой и факторами.

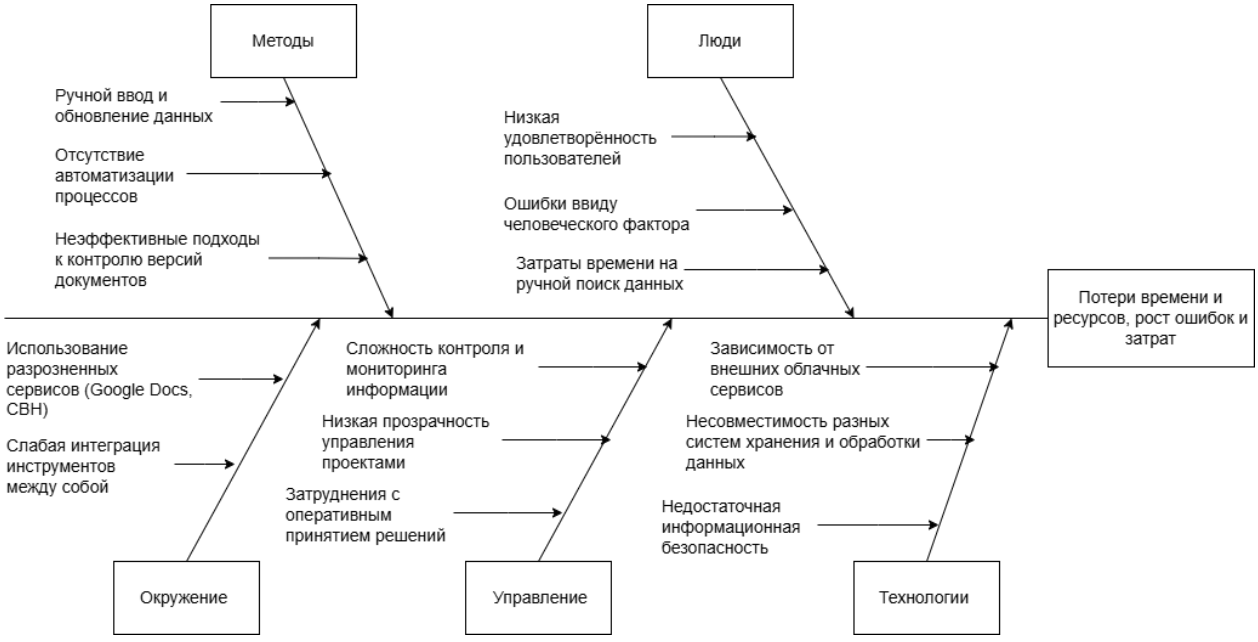

Рисунок 1 – Диаграмма Исикавы

Цифровая платформа, спроектированная именно под процессы предприятия, это точка консолидации данных и драйвер автоматизации.

Вместо того чтобы искать документы в Google Docs, проверять версии в SVN и держать в голове статус каждой задачи, пользователи работают в одном интерфейсе.

Столбчатая диаграмма на рисунке 2 иллюстрирует насколько существенно специализированное веб-приложение меняет ситуацию в компании.

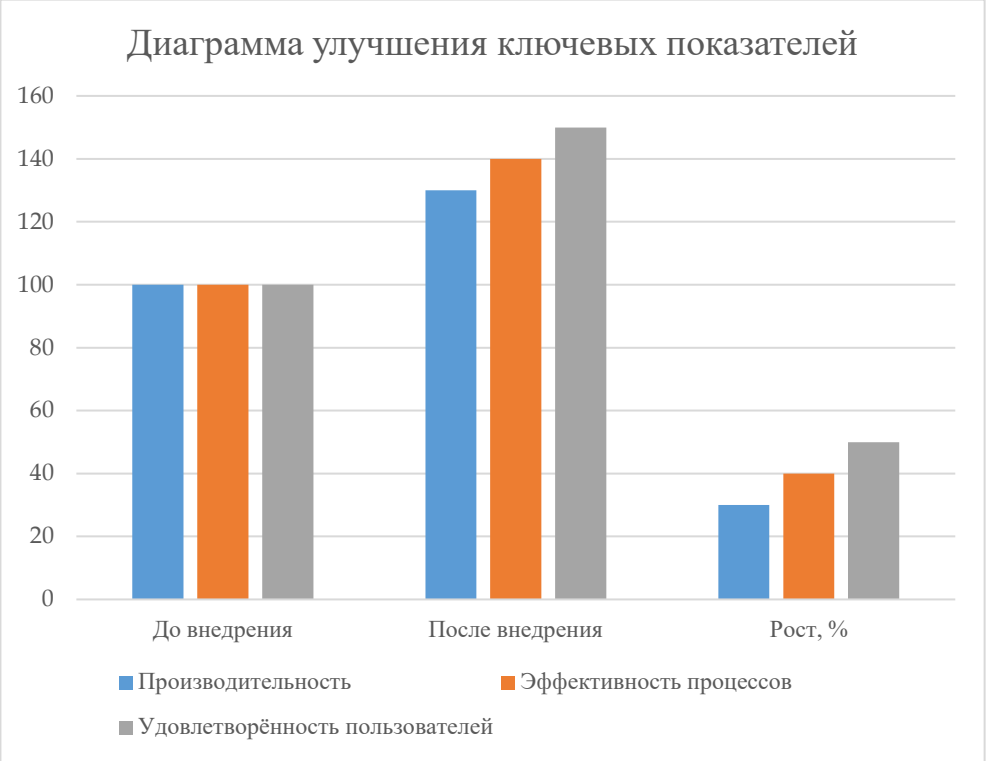

Рисунок 2 – Столбчатая диаграмма улучшения ключевых показателей

Таким образом, переход от разрозненных внешних сервисов к собственному веб-приложению позволяет АО «Эр-Стайл СофтЛаб» не только выполнить требования импортозамещения и усилить информационную безопасность, но и создать единую цифровую среду, в которой вся проектная деятельность становится прозрачной, управляемой и доступной из одного интерфейса.

Централизация данных, автоматизация рутинных операций и сквозной контроль версий обеспечивают рост эффективности процессов и формируют надёжный фундамент для дальнейших инноваций предприятия [6].

1.2 Основные функции веб-приложения для автоматизации управления проектами

Разрабатываемое веб-приложение предназначено для автоматизации управления проектами в условиях многозадачной среды и ориентировано на решение прикладных задач, с которыми ежедневно сталкиваются сотрудники предприятия.

Его функциональность направлена на упрощение контроля за ходом проектов, повышение прозрачности процессов и удобство работы с документацией [6]. Ключевые функции включают:

- создание и ведение проектов. Пользователи могут формировать проекты, указывать их основные характеристики, такие как наименование, инициатор, куратор, статус приёмки и описание работ;
- управление задачами (BIQ/подзадачи). В каждом проекте можно добавлять задачи, отслеживать их статус, трудозатраты и этапы выполнения, что облегчает координацию работы и контроль сроков;
- учёт функционала. Система позволяет привязывать задачи к конкретным элементам функционала, что упрощает анализ объёмов работ и их влияние на конечный результат;
- работа с документацией. Приложение поддерживает загрузку и хранение ключевых документов проекта: договоров, оценок трудозатрат, актов, протоколов тестирования и других файлов. Для каждого документа фиксируется его тип и путь, а также имя файла [17];
- контроль согласования. Для каждого проекта можно указывать дату согласования и отслеживать, на каком этапе он находится, что важно для соблюдения сроков и внутреннего контроля;

– унификация данных. Использование единого интерфейса и структуры хранения информации позволяет избежать дублирования данных, облегчить анализ и ускорить доступ к нужной информации.

На рисунке 3 приведена DFD диаграмма потоков данных, которая наглядно показывает, как информация проходит последовательные этапы: поступает от пользователя, обрабатывается модулями приложения, сохраняется в хранилищах и затем возвращается в виде отчётов и документов [12].

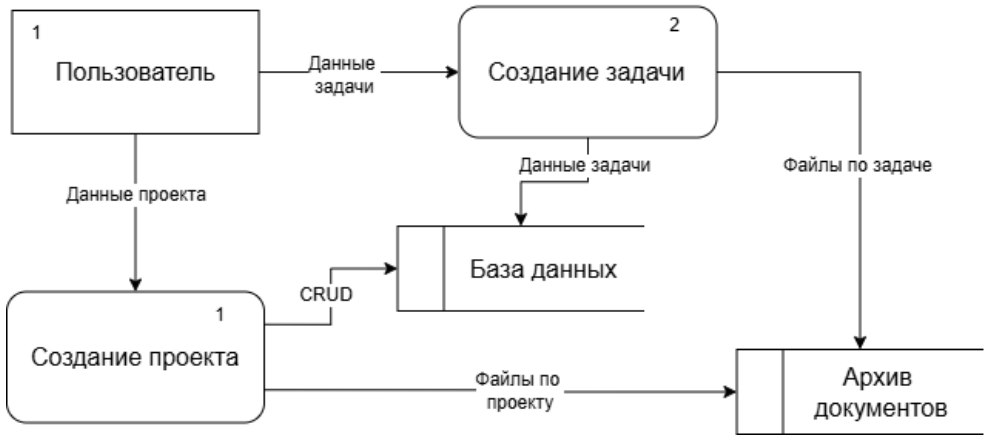

Рисунок 3 - DFD диаграмма потоков данных

Эта схема служит отправной точкой для дальнейшего объяснения каждого из блоков и их взаимодействий.

Разрабатываемое приложение становится единым окном управления финансовыми, человеческими и материальными ресурсами проекта:

– финансы - в карточке проекта фиксируются бюджет, оценка трудозатрат и договорная стоимость; система автоматически сравнивает план и факт, подсвечивая перерасход;
– человеческие ресурсы - при создании задач (BIQ) указывается количество внутренних и внешних трудозатрат; это помогает равномерно распределять работу в течении срока договора;
– материальные ресурсы / документация - модуль «Работа с документами» хранит договоры, акты и спецификации. Контроль версий исключает

двойную работу и гарантирует, что все используют актуальную редакцию файла.

Таблица 1 - Таблица с примерами систем управления ресурсами

Система / модуль	Что делает	Польза для ресурсов
Модуль Проекты	Бюджет, сроки, инициатор, договорная сумма	Контроль финансов и сроков в одном месте
Модуль Создание задач	Назначение исполнителей, учёт трудозатрат	Равномерная загрузка сотрудников, точное план-факт сравнение
Модуль Документы	Хранение договоров, актов, ТЗ и протоколов	Исключение потерь файлов, быстрый доступ к материалам
Отчётность	Автоматические сводки по затратам и трудозатратам	Быстрое выявление перерасхода, принятие корректирующих мер

Таким образом, приложение объединяет управление финансовыми, человеческими и материальными ресурсами в одном интерфейсе. DFD-диаграмма показывает, как данные от пользователя проходят обработку, сохраняются и возвращаются в виде отчётов.

1.3 Обзор существующих решений и их анализ

Для анализа предметной области были рассмотрены популярные программные решения для управления проектами и документами, такие как Redmine, Jira, Trello и Notion. Их использование в компании затруднено по следующим причинам:

- готовые системы требуют адаптации под текущие бизнес-процессы, что может потребовать значительных ресурсов;
- использование сторонних решений несет потенциальные риски утечки данных, так как данные могут храниться на серверах третьих лиц;

– большинство готовых систем избыточны по функционалу, что усложняет их внедрение и использование в текущих рабочих процессах компании.

Компания придерживается политики использования исключительно собственных разработок. Это обусловлено следующими факторами:

– полная адаптация под внутренние процессы: собственные разработки позволяют учесть специфику рабочих процессов и гибко изменять функциональность по мере необходимости;
– контроль над безопасностью: вся информация хранится внутри корпоративной инфраструктуры, что исключает риски утечки данных и упрощает выполнение требований информационной безопасности;
– независимость от внешних провайдеров: компания полностью контролирует разработку, эксплуатацию и дальнейшее развитие системы, что исключает риски, связанные с изменением условий использования или прекращением поддержки сторонними разработчиками.

Разработка собственного веб-приложения является оптимальным решением, учитывая специфику компании и ее подход к управлению данными и проектами.

Данный подход обеспечит:

– полную интеграцию приложения в существующие бизнес-процессы;
– повышение уровня безопасности хранения данных;
– возможность масштабирования и доработки системы в зависимости от потребностей компании.

Кроме того, собственная разработка позволяет оптимизировать затраты на долгосрочной перспективе: вместо регулярных платежей за лицензии и дополнительных модулей сторонних сервисов, компания инвестирует в единовременную настройку и последующую поддержку системы своими силами.

Это также ускоряет реакцию на изменения требований: новые функции или доработки можно внедрять без согласования с внешними подрядчиками, что повышает оперативность принятия решений и снижает риски простоев.

В приведённой таблице представлены результаты сравнительного анализа популярных решений для управления проектами с точки зрения их адаптации под конкретные бизнес-процессы, уровня безопасности, избыточности функционала и зависимости от внешних провайдеров.

Таблица 2 - Сравнительный анализ аналогов

Решение	Адаптация под процессы	Контроль безопасности	Избыточность функционала	Зависимость от провайдеров	Итог
Redmine	Требует глубокой доработки плагинов; затраты ресурсов	Данные на стороннем сервере	Множество модулей, не все нужны	Зависит от обновлений сообщества	Высокая стоимость кастомизации и риски безопасности
Jira	Сложная конфигурация под нестандартные процессы	Дорогое лицензирование	Обширный функционал	Поддержка и лицензии у провайдера	Быстрый старт, но дорого и сложно адаптируется
Trello	Мало возможностей для сложных процессов	Нет полного контроля	Не покрывает большинство задач	Полная зависимость от сервиса	Неприменим к крупным проектам
Notion	Универсальный конструктор не специализирован	Облачное хранение, риск сливов через интеграции	Много лишних блоков для задач	Зависит от политики Notion	Визуализация информации не масштабируется

Таким образом, не существует функционального альтернативного приложения, соответствующего вышеуказанным требованиям. В таком случае

разработка собственного веб-приложения имеет актуальность и целесообразность.

1.4 Требования к функционалу и интерфейсу веб-приложения для автоматизации управления проектами

Проектируемое веб-приложение предназначено для автоматизации процессов управления проектами в условиях многозадачности и должно соответствовать требованиям как по функционалу, так и по удобству пользовательского взаимодействия. На основании анализа задач и внутренних процессов предприятия были сформированы следующие требования:

Функциональные требования.

Для управления проектами веб-приложение должно обеспечивать:

- возможность создания и удаления проектов;
- хранение ключевой информации о каждом проекте, включая: наименование проекта, номер договора, даты подписания и окончания договора, ставку, сумму контракта, внешние и внутренние трудозатраты.

Управление документами:

- возможность загрузки и хранения документов.

Для управления задачами (BIQ), связанными с проектом, должны быть реализованы:

- функции создания и удаления задач;
- учет необходимых атрибутов для задач: проект, наименование задачи/BIQ, функционал, инициатор, куратор, трудозатраты, дата согласования оценки, статус приёмки, описание работы.

Нефункциональные требования.

Производительность:

– время отклика rest-api при типовой нагрузке не должно превышать 300 мс; стартовая загрузка spa-клиента - не дольше 2 с при скорости сети 20 мбит/с.

Масштабируемость:

– система обязана поддерживать как минимум 100 одновременных активных пользователей.

Удобство использования (UX):

– интерфейс построен как SPA на React; к любой ключевой функции пользователь должен добираться максимум за четыре клика;
– приложение адаптивно и полностью русифицировано.

Поддерживаемость:

– код следует единому стилю (ESLint + Prettier); покрытие unit-тестами: ≥ 70 % для бекенда и ≥ 60 % для фронтенда.

Требования к интерфейсу

Предусмотренные требования обеспечивают надёжную и удобную работу с системой, соответствующую потребностям сотрудников компании и текущим стандартам корпоративной автоматизации [9].

– интуитивная навигация: интерфейс должен быть простым и логичным, с минимальным количеством действий для выполнения базовых операций;
– единый стиль отображения данных: все основные сущности должны быть оформлены в едином визуальном формате с поддержкой фильтрации и сортировки;
– адаптивность интерфейса: приложение должно корректно отображаться, включая экраны с разными разрешениями;
– доступность информации: упрощённый доступ к часто используемым функциям и быстрому просмотру данных.

Данные требования формируют единую основу для разработки веб-приложения, удовлетворяя как ключевым бизнес-задачам - управлению проектами, задачами и документами - так и ожиданиям конечных пользователей

по быстродействию, удобству и надёжности. Соблюдение функций создания/удаления проектов и задач, хранения всей необходимой информации, а также ограничения по времени отклика и покрытию тестами позволит обеспечить стабильную и масштабируемую работу системы. Интуитивный, адаптивный интерфейс с единым стилем и доступом к основным функциям не более чем в четыре клика гарантирует комфортное взаимодействие, что в совокупности соответствует стандартам корпоративной автоматизации и повышает эффективность работы сотрудников компании [9].

Вывод по первой главе

В главе были рассмотрены предпосылки создания веб-приложения для автоматизации управления проектами, проанализированы существующие решения, определены основные функции и сформулированы требования к системе. Таким образом, стало очевидно, что создание собственного программного продукта позволяет повысить эффективность работы предприятия за счёт централизованного хранения информации, прозрачного управления задачами и более гибкой настройки под внутренние процессы организации.

Глава 2 Процесс разработки автоматизированной системы управления проектами.

2.1 Проектирование программного обеспечения веб-приложения для автоматизации управления проектами

На этапе проектирования были определены ключевые сущности, взаимосвязи между ними и логика взаимодействия пользователя с системой. Основу веб-приложения составляют три функциональных блока: управление проектами, задачами и документами.

Каждый проект включает в себя ряд атрибутов - таких как наименование, номер и сроки действия договора, сумма контракта, трудозатраты и прочие параметры, важные для последующего анализа. Кроме того, проекты могут содержать множество связанных задач (BIQ), каждая из которых также содержит детальную информацию, включая функционал, инициатора, куратора, статус приёмки и дату согласования. Для каждой сущности (проект и задача) в базе также сохраняются имена загруженных файлов.

Дополнительно была спроектирована структура хранения документов: сами файлы физически сохраняются на сервере, а в базе MongoDB фиксируются лишь их имена, типы и ссылки на соответствующие проекты или задачи [2].

Проектирование велось с ориентацией на масштабируемость, чтобы в дальнейшем при необходимости можно было легко расширить функционал без кардинальных изменений архитектуры.

Для этого были применены ряды схем и диаграмм:

- диаграмма вариантов использования (use case),
- схема представления данных в формате NoSQL,
- диаграмма классов,
- диаграмма развертывания.

На рисунке 4 показана расширенная диаграмма вариантов использования, с тремя ключевыми участниками: аналитик, разработчик и руководитель проекта [3].

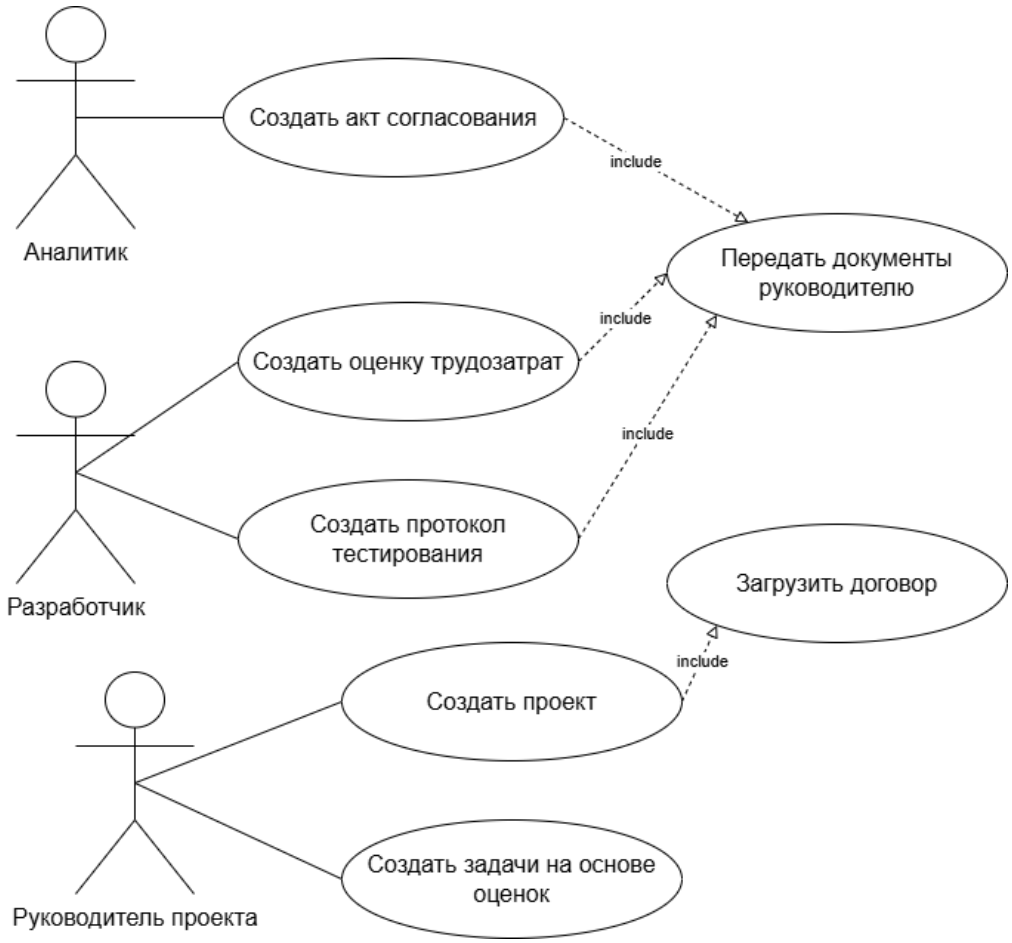

Рисунок 4 - UML диаграмма вариантов использования

На представленной диаграмме вариантов использования видно, как в системе реализуется сквозной сценарий подготовки и запуска проекта. Она показывает упрощённый сценарий: сначала аналитик создаёт акт согласования, а разработчик - оценку трудозатрат и протокол тестирования; каждый из этих шагов включает автоматическую передачу документов руководителю. После получения всех артефактов руководитель запускает «Создание проекта» и на основе оценки трудозатрат порождает соответствующие задачи.

Для наглядного представления создана схема представления данных в формате NoSQL. (Рисунок 5). Поскольку в проекте используется MongoDB, где отсутствуют классические реляционные таблицы и механизмы внешних ключей, связь между сущностями осуществляется через поля со ссылками - уникальными идентификаторами документов. Они обеспечивают привязку записей друг к другу.

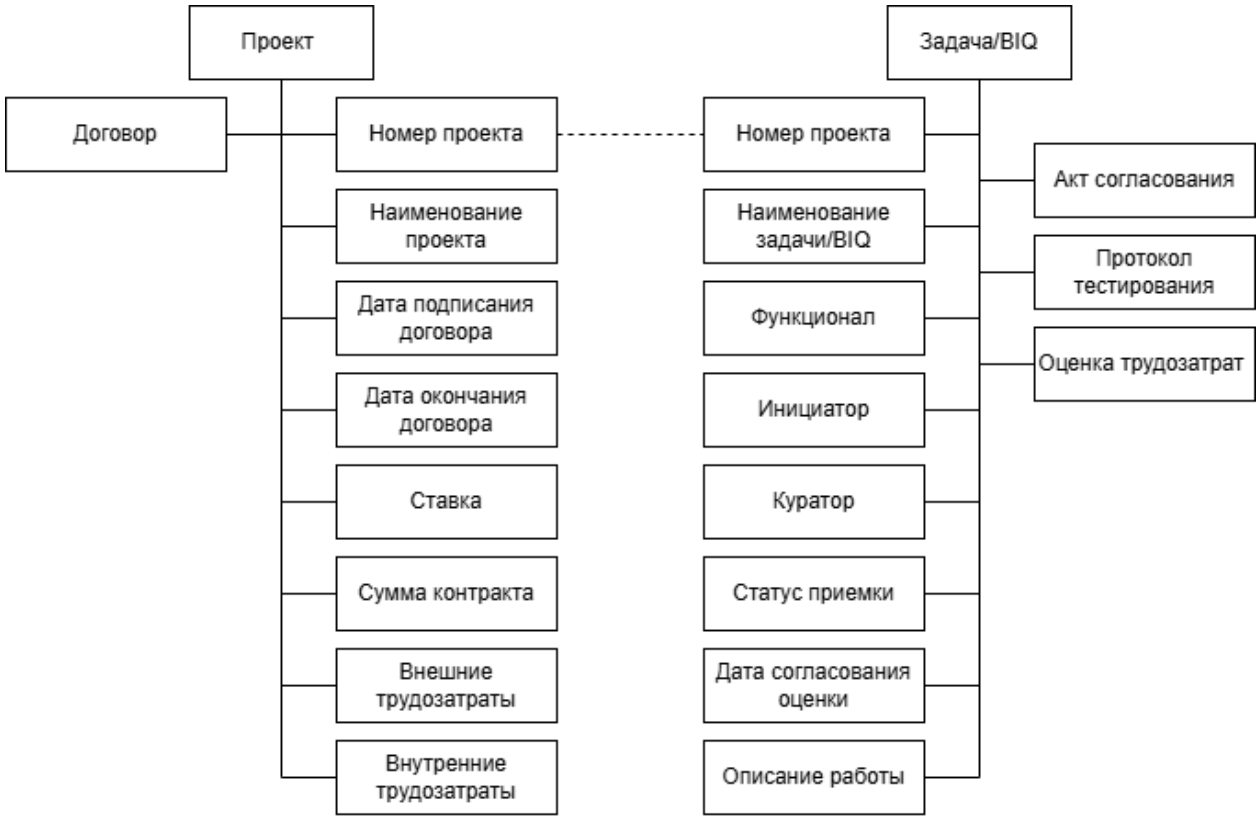

Рисунок 5 – Схема представления данных в формате NoSQL

На схеме видно, что сущности «Проект» и «Задача/BIQ» связаны через поле Номер проекта. Каждая задача привязана к одному проекту, тогда как один проект может содержать сразу несколько задач.

При этом бинарные файлы (договоры, акты, протоколы) физически сохраняются в файловой системе сервера, а в базе MongoDB для каждого документа хранится только его имя и ссылка на родительскую сущность (проект или задачу).

Диаграмма классов требуется для того, чтобы показать представление классов разрабатываемой системы. На рисунке 6 представлена диаграмма классов веб-приложения [20].

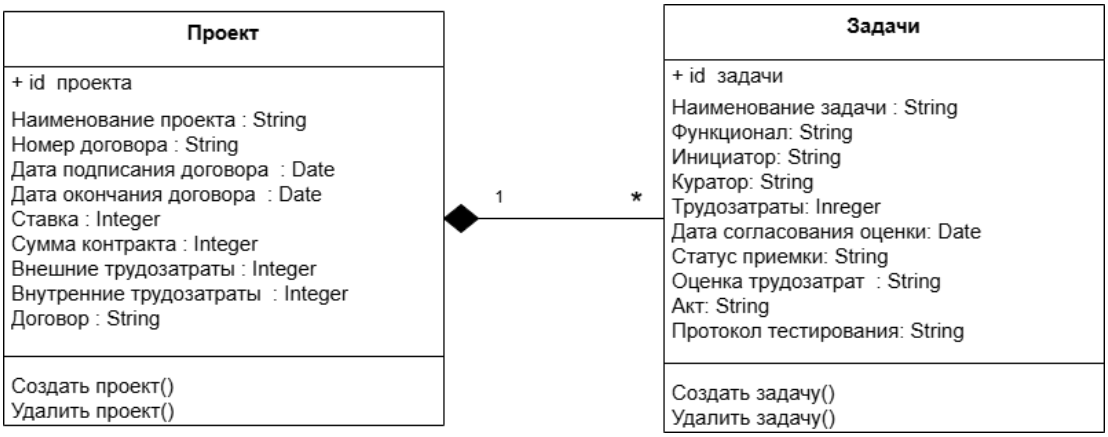

Рисунок 6 - Диаграмма классов

На представленной UML-диаграмме показаны два ключевых класса предметной области, без которых работа веб-приложения была бы невозможна.

Класс «Проект» хранит все основные данные о самом проекте и его договоре: уникальный номер; ставка; сумма контракта; внешние и внутренние трудозатраты; даты начала и окончания действия договора.

Поле «Договор» в модели указывает на имя файла, который лежит на сервере и привязан к этому проекту.

Класс «Задачи» описывает отдельные работы внутри проекта. У каждой задачи есть ссылка на номер проекта, к которому она относится, а также такие атрибуты, как название задачи, функционал, кто её инициировал и курирует, текущий статус приёмки, дата согласования оценки и подробное описание работы.

Связь «один-ко-многим» означает, что к одному проекту может быть привязано сразу несколько задач, а каждая задача принадлежит только одному проекту.

Когда пользователь загружает файл (договор, акт или протокол), сам файл сохраняется на сервере, а в базе хранится только его имя и связь с нужным проектом или задачей. Это гарантирует, что все документы надёжно связаны с соответствующими объектами системы.

Следующим этапом является разработка диаграммы деятельности, которая представлена на рисунке 7.

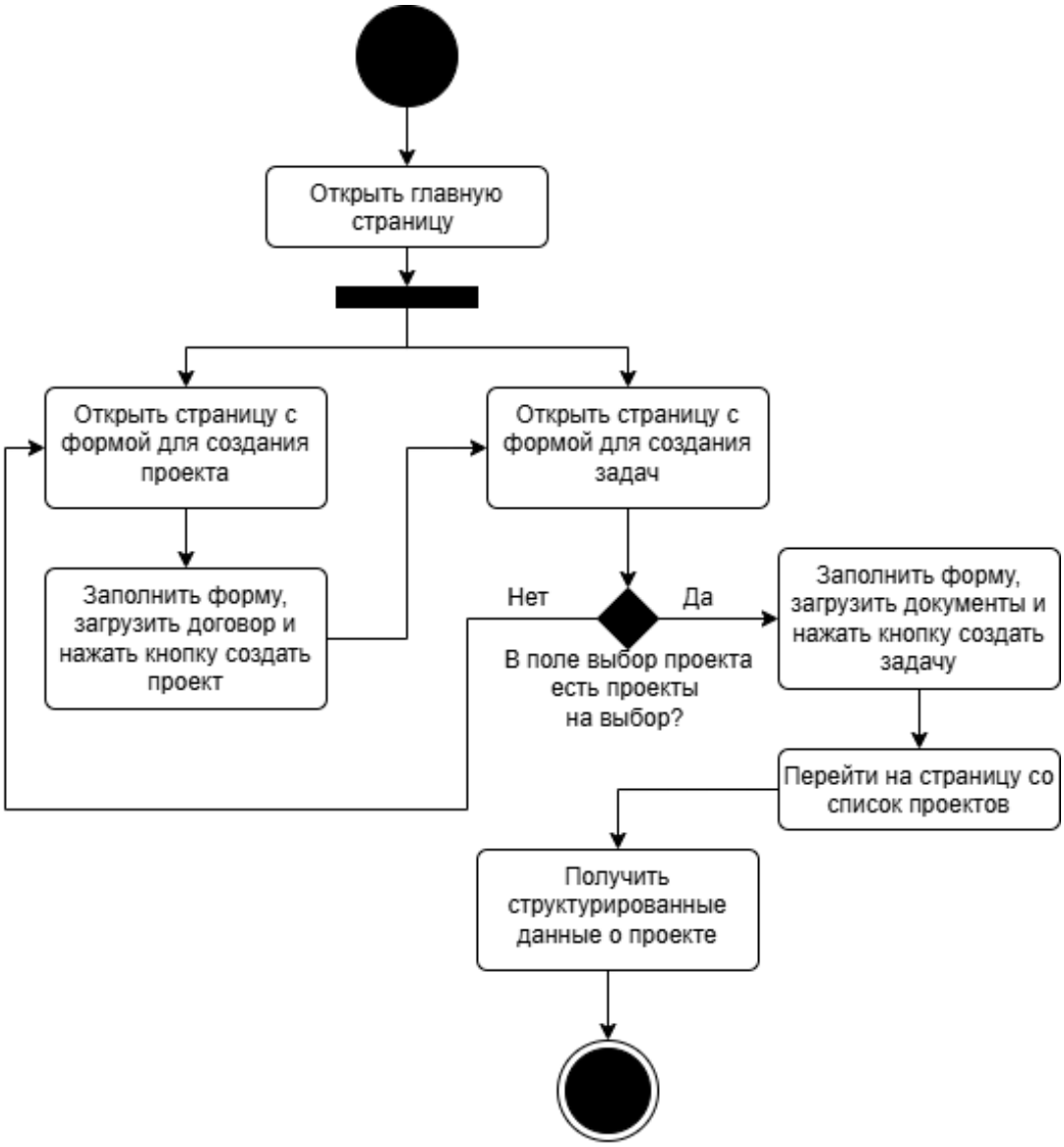

Рисунок 7 - Диаграмма деятельности

На данной диаграмме отображена последовательность работы веб-приложения. Действие начинается с запуска браузера и открытия веб-сайта.

При первом заходе пользователя на сайт, он попадает на главную страницу. Пользователь начинает заполнять форму создания проекта и загружает договор. Затем переходит к форме создания задачи. Если в выборе проекта у пользователя возникли проблемы, созданных проектов нет, он переходит на страницу создания проектов. Завершающим этапом работы веб-приложения является получение данных о проекте и задач в нем.

Благодаря такому набору UML-диаграмм глава обеспечивает целостное понимание логической архитектуры веб-приложения и подготовки к последующему физическому проектированию и реализации.

2.2 Выбор технологий и инструментов для разработки веб-приложения для автоматизации управления проектами

Во время проектирования архитектуры приложения были проанализированы несколько вариантов реализации каждого слоя:

Frontend-фреймворк: сравнивались React, Angular и Vue.js. React был выбран за счёт широкой экосистемы, простоты интеграции с TypeScript и возможности использовать React Hooks и React Query для управления состоянием и кэшированием данных.

СУБД: рассматривались реляционные базы PostgreSQL и MySQL, а также документно-ориентированные MongoDB и CouchDB. MongoDB оказалась наиболее подходящей для гибких схем хранения проектов, задач и метаданных документов без жёсткой структуры таблиц и сложных миграций [4].

Серверная платформа: сравнивались Express.js, Koa и NestJS на Node.js. Express.js был выбран за минимальный порог вхождения, очень широкую поддержку middleware и простоту настройки REST-эндпоинтов [13].

API-архитектура: альтернативой REST API выступал GraphQL, однако для простых CRUD-операций и однородной структуры данных REST оказался более производительным и менее ресурсоёмким в поддержке.

Наглядное расположение выбранных технологий и их взаимодействия показано на рисунке 8 [3].

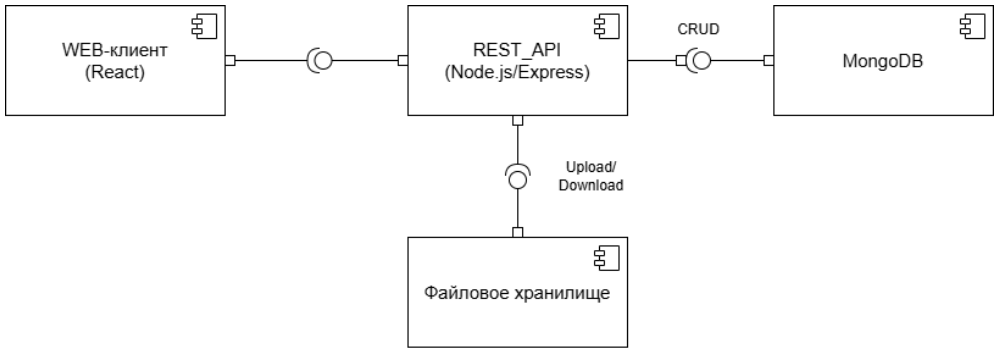

Рисунок 8 - UML диаграмма компонентов

Выбор данных инструментов обоснован необходимостью обеспечить стабильную работу приложения, его масштабируемость и простоту внедрения в текущие процессы предприятия.

Для реализации веб-приложения были выбраны современные и проверенные технологии, обеспечивающие стабильную работу, гибкость и удобство поддержки. Поэтому в качестве frontend-фреймворка использован React, что позволяет создавать отзывчивый и интерактивный интерфейс с возможностью быстрой доработки [1], [5].

Для хранения данных была выбрана MongoDB - документно-ориентированная NoSQL база данных, подходящая для структур с гибкой схемой, таких как списки задач или документы с переменным набором параметров [15]. Такой подход позволил реализовать гибкую модель хранения данных, в которой легко учитывать изменения структуры проекта без необходимости сложной миграции схем.

Серверная часть построена на базе Node.js с фреймворком Express.js, что обеспечивает лёгкое масштабирование и модульное расширение бизнес-логики через REST-эндпоинты [11], [13].

Для взаимодействия между клиентской частью и сервером применялись технологии REST API. Это позволило обеспечить модульность, удобную интеграцию и тестируемость всех компонентов системы.

2.3 Реализация функциональных требований веб-приложения для автоматизации управления проектами

Работа с веб-приложением начинается с открытия главной страницы, пользователь сразу попадает в интерфейс управления проектами. В верхнем меню доступен раздел «Создать проект», где отображается таблица всех текущих проектов с указанием наименования, номера договора, дат и текущего статуса.

При создании нового проекта, после нажатия кнопки «Создать проект», открывается форма с полями и загрузкой файла договора. После сохранения данные отправляются на сервер, где создаётся новый проект (Рисунок 9).

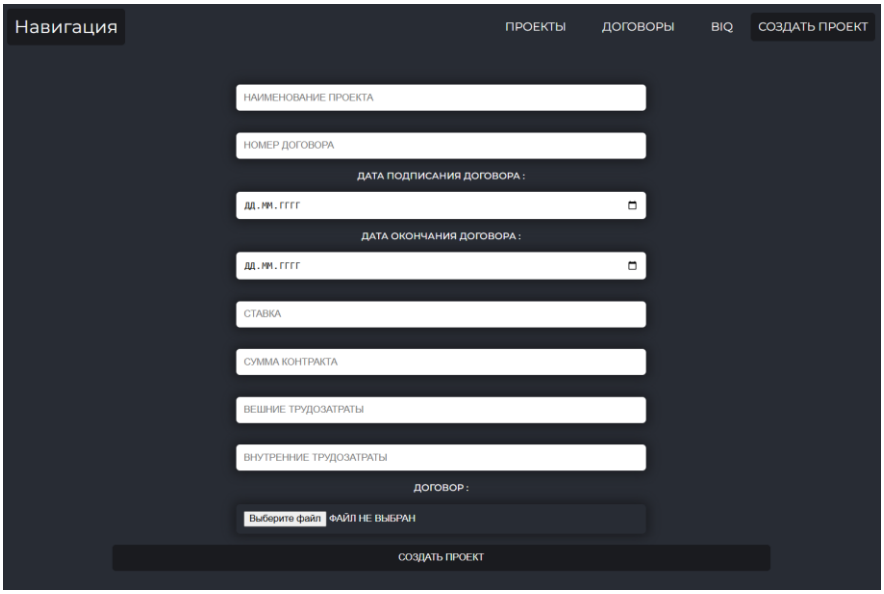

Рисунок 9 – Страница «Создание проекта»

На рисунке показана форма, стилизованная в CreateProject.css: двухколоночная сетка, оформление инпутов и кнопок, а также адаптивные правила для мобильных экранов [7].

На представленной блок-схеме показан пошаговый алгоритм создания нового проекта в веб-приложении (Рисунок 10).

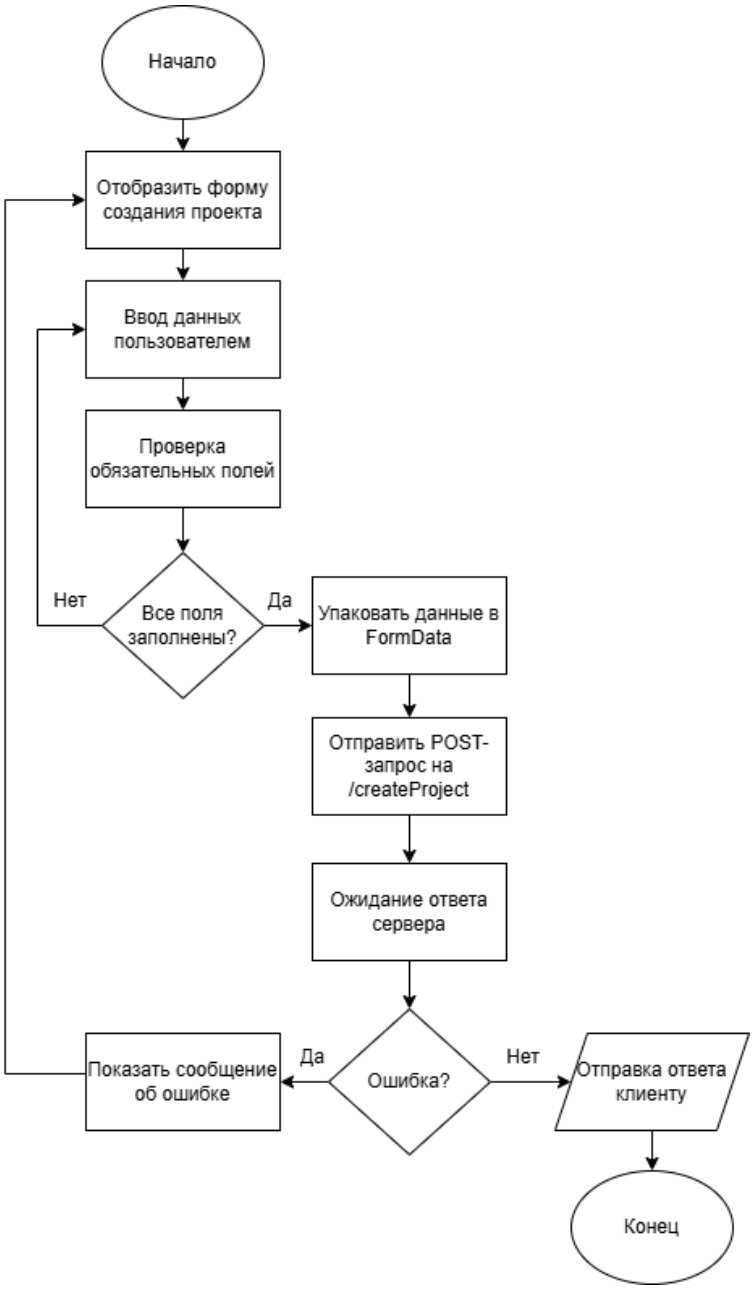

Рисунок 10 - Блок-схема создания проекта

Показана последовательность действий: после отображения формы и ввода данных приложение проверяет заполнение обязательных полей; если какое-то поле осталось пустым, выводится сообщение об ошибке и пользователь возвращается к заполнению.

Если все поля корректны, данные вместе с выбранным файлом упаковываются в объект FormData, и отправляется POST-запрос на эндпоинт /createProject через функцию handleSubmit (см. Рисунок 11). Затем приложение ожидает ответа сервера: в случае ошибки появится уведомление, в случае успеха сервер сохраняет запись в MongoDB и файл в файловой системе, а клиентский код вызывает navigate('/projects'), автоматически возвращая пользователя к списку проектов.

При разработке страницы «Создание проекта» сначала была продумана её структура и набор обязательных полей. Представлена структура формы на рисунке 11.

```
return (
    <div className="project-main-wrapper">
    <form>
        <input onChange={handleName} type="text" className="ProjectName" placeholder="Наименование проекта"/>
        <input onChange={handleNumber} type="text" className="ContractNum" placeholder="Номер договора"/>
        <h1 className="title_of_insert">Дата подписания договора :</h1>
        <input onChange={handleStartDate} type="date" className="DataStart" placeholder="Дата подписания договора" />
        <h1 className="title_of_insert">Дата окончания договора :</h1>
        <input onChange={handleEndDate} type="date" className="DataEnd" placeholder="Дата окончания договора" />
        <input onChange={handleRate} type="text" className="Rate" placeholder="Ставка"/>
        <input onChange={handlePrice} type="text" className="Price" placeholder="Сумма контракта"/>
        <input onChange={handleExternal} type="text" className="ExternalLabor" placeholder="Вешние трудозатраты"/>
        <input onChange={handleInternal} type="text" className="InteriorLabor" placeholder="Внутренние трудозатраты"/>
        <h1 className="title_of_insert">Договор :</h1>
        <input onChange={handleFile} name="attachment" ref={fileRef} type="file" className="ContractFile" placeholder="Договор" />
        <button onClick={handleSubmit}>Создать проект</button>
    </form>
    </div>
)
```

Рисунок 11 – Структура формы для создания проекта

В этой форме обязательно заполняются поля: «Наименование проекта», «Номер договора», «Дата подписания», «Дата окончания», «Ставка», «Сумма контракта», «Внешние трудозатраты», «Внутренние трудозатраты» и прикладывается файл договора. При нажатии на кнопку «Создать проект» вызывается функция handleSubmit, собирающая все значения и отправляющая их на сервер в виде FormData.

Интерфейс построен как отдельный React-компонент CreateProject.js, в котором каждое поле связано с собственным состоянием через хук useState [10], [18]. Инициализация локального состояния показана на рисунке 12.

```
const [title, setTitle] = useState("");
const [number, setNumber] = useState("");
const [startDate, setStartDate] = useState("");
const [endDate, setEndDate] = useState("");
const [rate, setRate] = useState("");
const [price, setPrice] = useState("");
const [externalLabor, setExternal] = useState("");
const [internalLabor, setInternal] = useState("");
const [file, setFile] = useState("");
```

Рисунок 12 - Инициализация локального состояния

На фрагменте кода, представленном на рисунке 12, показана инициализация локального состояния каждого поля формы с помощью хука useState. Для каждого обязательного атрибута проекта (наименование, номер договора, даты, ставка, трудозатраты и прикреплённый файл) создаётся своя пара (значение, set Функция), что позволяет React-компоненту динамически отслеживать и обновлять введённые пользователем данные при каждом изменении соответствующего <input> [18].

Для загрузки файла используется <input type="file"> с рефом, позволяющим сбрасывать выбор после отправки. На фрагменте JSX показано поле, предназначенное для прикрепления договора (Рисунок 13).

```
<input onChange={handleFile} name="attachment" ref={fileRef} type="file" className="ContractFile" placeholder="Договор" />
```

Рисунок 13 - Элемент загрузки файла в форме

Обработчик handleFile привязан к событию onChange, чтобы при выборе документа сохранить его в локальное состояние компонента.

Логика отправки формы заключена в асинхронной функции-обработчике handleSubmit. Функция handleSubmit в компоненте CreateProject.js показана на рисунке 14 [14].

```
const handleSubmit = (e) => {
    e.preventDefault();
    const formData = {
        attachment: file,
        title,
        number,
        startDate,
        endDate,
        rate,
        price,
        externalLabor,
        internalLabor
    }
```

Рисунок 14 - Функция handleSubmit в компоненте CreateProject.js

Функция handleSubmit вызывается при клике на кнопку «Создать проект». Сначала она отменяет стандартное поведение формы (e.preventDefault()), затем собирает все значения полей и выбранный файл в объект formData [8].

При клике на кнопку «Сохранить» все значения полей и выбранный файл упаковываются в FormData, после чего вызывается axios.postForm("http://localhost:2024/createProject", formData).

Фрагмент кода представлен на рисунке 15.

```
console.log(formData);

axios.postForm("http://localhost:2024/createProject", formData).then((res) => {
    console.log(res.data);
    fileRef.current.value = null;
});
```

Рисунок 15 - Отправка сформированных данных на сервер в компоненте

После успешного ответа поле загрузки файла сбрасывается (fileRef.current.value = null), а в консоль выводятся данные ответа [14].

При реализации страницы «Создание задачи» в рамках модуля BIQ была выбрана архитектура «модального окна» внутри компонента BIQ.js, чтобы пользователь мог добавлять подзадачи, не покидая текущий контекст проекта.

Рисунок 16 показывает форму ввода данных для задачи.

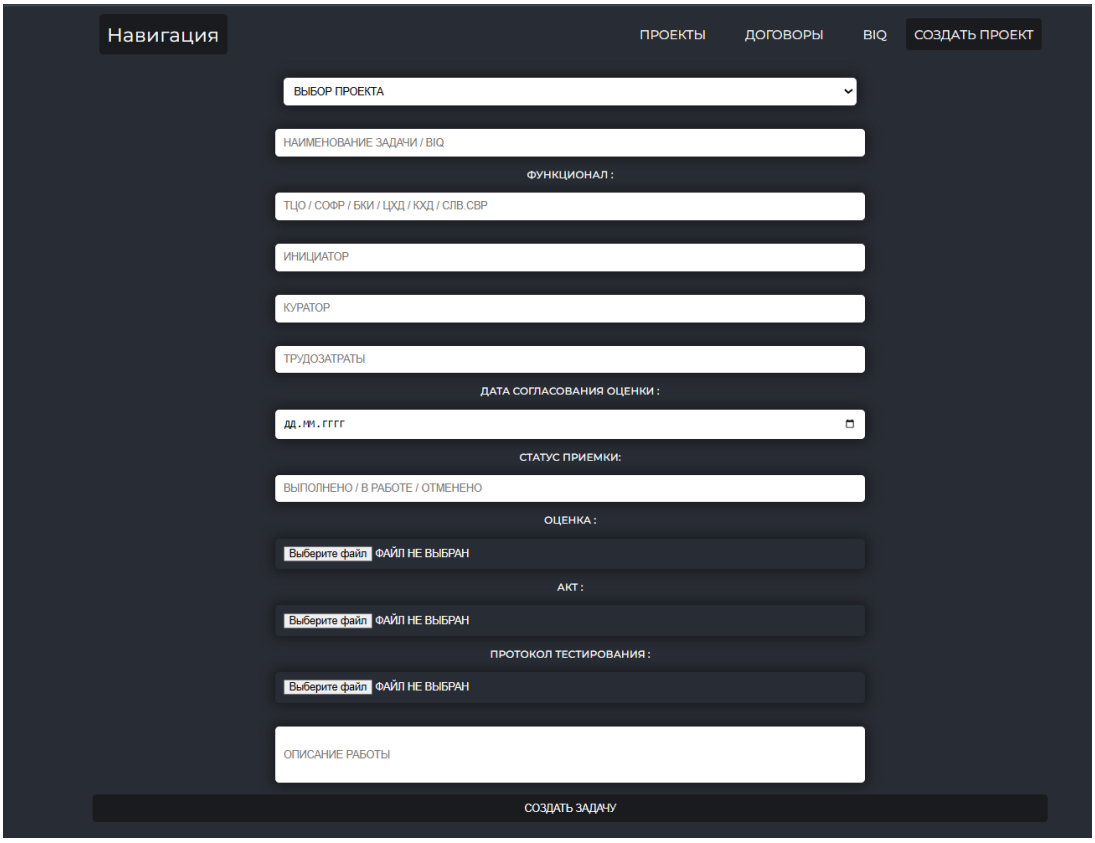

Рисунок 16 - Страница «Создание задачи»

При открытии модального окна сразу загружается список проектов, чтобы пользователь мог сразу выбрать нужный вариант.

Стилизация формы вынесена в BIQ.css: там определена сетка с двумя колонками для полей, единый отступ и размер для input-ов, стили для кнопок. Кроме того, в этом файле прописаны медиа-запросы, обеспечивающие корректное отображение формы на разных устройствах [7].

При нажатии «Создать задачу» система открывает модальное окно с полями ввода и загрузки файлов (см. Рисунок 17). После заполнения формы происходит проверка обязательных полей, если что-то не указано, выводится сообщение об ошибке и окно остаётся открытым, иначе все данные и выбранные файлы упаковываются в FormData и отправляются на сервер.

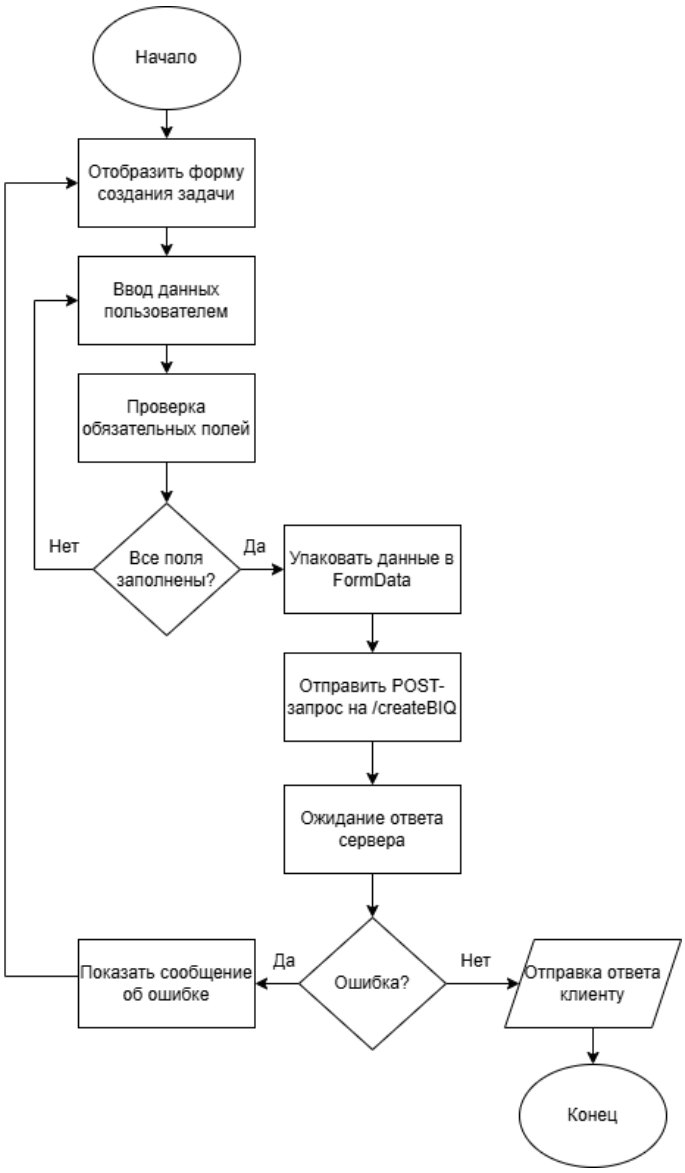

Рисунок 17 - Блок-схема алгоритма создания BIQ-задачи

В зависимости от ответа сервера либо отображается ошибка, либо окно автоматически закрывается, а список задач обновляется, чтобы сразу показать новую BIQ-задачу.

При нажатии на кнопку «BIQ» во внутреннем состоянии компонента срабатывает хук useState, устанавливающий флаг, который отвечает за отображение модального диалога с формой для создания новой задачи. В этом модальном окне пользователь видит набор полей, необходимых для описания BIQ-задачи: выбор связанного проекта, наименование задачи, функциональный блок, указание инициатора и куратора, ввод плановых трудозатрат, выбор

текущего статуса приёмки, дата согласования оценки, а также текстовое поле для подробного описания задания.

На рисунке 18 показана инициализация локального состояния в компоненте BIQ.js

```
const [projectSelect, setProject] = useState("");
const [BIQName, setBIQName] = useState("");
const [Functionality, setFunctionality] = useState("");
const [Initiator, setInitiator] = useState("");
const [Curator, setCurator] = useState("");
const [LaborCosts, setLaborCosts] = useState("");
const [DataStart_2, setDataStart_2] = useState("");
const [AcceptanceStatus, setAcceptanceStatus] = useState("");
const [Value, setValue] = useState("");
const [Act, setAct] = useState("");
const [TestProtocol, setTestProtocol] = useState("");
const [Description, setDescription] = useState("");
```

Рисунок 18 - Инициализация локального состояния

Пары (значение, setФункция) инициализируются для хранения выбранного проекта (projectSelect), названия задачи (BIQName), функциональности, инициатора, куратора, трудозатрат, даты согласования, статуса приёмки, а также загруженных файлов (Value, Act, TestProtocol) и описания. Это обеспечивает «управляемый» ввод: при изменении любого <input> или <select> соответствующая функция set… обновляет локальное состояние компонента. В результате компонент становится более предсказуемым и устойчивым к неожиданным действиям пользователей, а также упрощается логика отправки данных на бэкенд и последующая обработка.

Для каждого из трёх файлов (оценка, акт, протокол тестирования) используется отдельный <input type="file">, обёрнутый в ref, что позволяет отследить и сбросить загрузку. На рисунке 19 показана структура формы для создания BIQ-задачи.

```
    </select>
    <input onChange={handleBIQName} type="text" className="BIQName" placeholder="Наименование задачи / BIQ"/>
    <h1 className="title_of_insert">Функционал :</h1>
    <input type="text" onChange={handleFunctionality} className="Functionality" placeholder="ТЦО / СОФР / БКИ / ЦХД / КХД / СЛВ.СВР"/>
    <input type="text" onChange={handleInitiator} className="Initiator" placeholder="Инициатор"/>
    <input type="text" onChange={handleCurator} className="Curator" placeholder="Куратор"/>
    <input type="text" onChange={handleLaborCosts} className="LaborCosts" placeholder="Трудозатраты"/>
    <h1 className="title_of_insert">Дата согласования оценки :</h1>
    <input type="date" onChange={handleDataStart_2} className="DataStart_2" placeholder="Дата согласования оценки от банка" />
    <h1 className="title_of_insert">Статус приемки:</h1>
    <input type="text" onChange={handleAcceptanceStatus} className="AcceptanceStatus" placeholder="Выполнено / В работе / Отменено"/>
    <h1 className="title_of_insert">Оценка :</h1>
    <input type="file" onChange={handleValue} ref={ValueRef} className="Value" placeholder="Оценка" />
    <h1 className="title_of_insert">Акт :</h1>
    <input type="file" onChange={handleAct} ref={ActRef} className="Act" placeholder="Акт" />
    <h1 className="title_of_insert">Протокол тестирования :</h1>
    <input type="file" onChange={handleTestProtocol} ref={TestProtocolRef} className="TestProtocol" placeholder="Протокол тестирования" />
    <input type="text" onChange={handleDescription} className="Description" placeholder="Описание работы"/>
    <button onClick={handleForm}>Создать задачу</button>
</form>
```

Рисунок 19 - Фрагмент JSX-кода компонента BIQ.js

Логика отправки сконцентрирована в функции handleForm. Она собирает все поля и файлы в единый объект FormData, где каждому ключу соответствует либо строковое значение, либо бинарный файл, показана на рисунке 20 [8].

```
const handleForm = (e) => {
    e.preventDefault();
    const formData = {
        valueFile: Value,
        ActFile: Act,
        TestProtocolFile: TestProtocol,
        projectSelect,
        BIQName,
        Functionality,
        Initiator,
        Curator,
        LaborCosts,
        DataStart_2,
        AcceptanceStatus,
        Description
    };
```

Рисунок 20 - Функция handleForm в компоненте BIQ.js

После этого объект formData отправляется на сервер через axios.postForm, чтобы сохранить новую BIQ-задачу и загруженные файлы.

Отправка осуществляется через axios.postForm("http://localhost:2024/createBIQ", formData), как показано на рисунке 21.

```
console.log(formData);
axios.postForm("http://localhost:2024/createBIQ", formData).then((res) => {
    console.log(res.data);
    ValueRef.current.value = null;
    TestProtocolRef.current.value = null;
    ActRef.current.value = null;
});
```

Рисунок 21 - Отправка BIQ-данных и сброс полей в BIQ.js

Причём адрес задаётся динамически на основании projectSelect, чтобы сервер мог сразу связать новую задачу с нужным проектом.

На рисунке 22 показано, как с помощью хука useQuery из React Query загружается список проектов (queryKey: ['biq_projects']), а функция queryFn выполняет запрос axios.get("http://localhost:2024/projects") [19].

```
const { data } = useQuery({
    queryKey: ["biq_projects"],
    queryFn: async () => {
        const { data } = await axios.get("http://localhost:2024/projects")
        return data;
    }
});
```

Рисунок 22 - Получение списка проектов с помощью React Query

Вывод по второй главе.

Во второй главе описано логическое проектирование системы (ERD, UML-диаграммы), обоснован выбор стека React / Express / MongoDB и реализованы ключевые модули создания проекта и BIQ-задачи. При помощи React-хуков и axios.postForm на фронтенде и Express.js на бэкенде обеспечен полный цикл от ввода данных до их сохранения в MongoDB и файловой системе. Полученный прототип готов к дальнейшему тестированию и внедрению.

Глава 3 Тестирование и отладка веб-приложения для автоматизации управления проектами

3.1 Проверка корректности отдельных функций

В проекте тестирование включает несколько этапов подготовки и пять основных видов тестирования. Представлена таблица 3 с планом тестирования.

Таблица 3 – План тестирования

Цель тестирования	Метод	Ответственный
Проверка корректности отдельных функций (хуки, утилиты)	Юнит-тесты (Jest)	Фронтенд-разработчик
Проверка взаимодействия компонентов API и БД	Интеграционные тесты	Бэкенд-разработчик
Подтверждение соответствия функционала ТЗ	Функциональные сценарии	QA-инженер
Оценка производительности под нагрузкой	Нагрузочное тестирование (loadtest)	DevOps-инженер

Юнит-тестирование обеспечивает проверку каждого компонента и утилиты в изоляции, что позволяет выявлять ошибки на ранней стадии разработки. В нашем проекте используются инструменты Jest, React Testing Library и @testing-library/jest-dom. Все тесты размещены в каталоге src/tests/ и именуются по шаблону *.test.js [1].

Для страницы CreateProject создан файл тестов CreateProject.test.js, в котором реализован модульный сценарий проверки первоначальной отрисовки формы создания проекта.

Тест проверяет, что форма отрисовывает все обязательные поля и кнопку «Создать проект». Код показан на рисунке 23.

src > tests > JS CreateProject.test.js > ...

```
// src/tests/CreateProject.test.js
import React from 'react';
import { render, screen, fireEvent } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';
import CreateProject from '../pages/CreateProject';

describe('CreateProject', () => {
  it('рендерит все поля формы', () => {
    render(<CreateProject />);
    expect(screen.getByPlaceholderText('Наименование проекта')).toBeInTheDocument();
    expect(screen.getByPlaceholderText('Номер договора')).toBeInTheDocument();
    expect(screen.getByPlaceholderText('Сумма контракта')).toBeInTheDocument();
  });
});
```

Рисунок 23 – Листинг юнит-тестов CreateProject

При таком наборе проверок пользователь увидит все поля и не сможет отправить неполную форму, что повышает надёжность интерфейса и упрощает поддержку кода.

Для страницы BIQ использовала файл тестов: BIQ.test.js. Тест убеждается, что при клике по кнопке «Создать задачу» открывается модальное окно с полем «Наименование задачи / BIQ». Код показан на рисунке 24.

src > tests > JS BIQ.test.js > ...

```
// src/tests/BIQ.test.js
import React from 'react';
import { render, screen, fireEvent } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';
import BIQ from '../pages/BIQ';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';

const qc = new QueryClient();
jest.mock('axios', () => ({
  get: () => Promise.resolve({ data: [] }),
  postForm: () => Promise.resolve({ data: {} })
}));

describe('BIQ', () => {
  it('открывает модальное окно создания задачи', () => {
    render(
      <QueryClientProvider client={qc}>
        <BIQ />
      </QueryClientProvider>
    );
    fireEvent.click(screen.getByText('Создать задачу'));
    expect(screen.getByPlaceholderText('Наименование задачи / BIQ')).toBeInTheDocument();
  });
});
```

Рисунок 24 – Листинг юнит-тестов BIQ

Для страницы Contracts использовала файл тестов: Contracts.test.js Он проверяет наличие заголовка вкладки «Договоры». Код показан на рисунке 25.

src > tests > JS Contracts.test.js > ...

```
// src/tests/Contracts.test.js
import React from 'react';
import { render, screen } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';
import Contracts from '../pages/Contracts';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';

const qc = new QueryClient();
jest.mock('axios', () => ({ get: () => Promise.resolve({ data: [] }) }));

describe('Contracts', () => {
  it('показывает заголовок вкладки Договоры', () => {
    render(
      <QueryClientProvider client={qc}>
        <Contracts />
      </QueryClientProvider>
    );
    expect(screen.getByText(/договоры/i)).toBeInTheDocument();
  });
});
```

Рисунок 25 – Листинг юнит-тестов Contracts

Для страницы Header использовала файл тестов: Header.test.js. Он проверяет навигационный контейнер и ссылки, как показывает рисунок 26.

src > tests > JS Header.test.js > describe('Header Component') callback > it('renders the navigation container') callback

```
// Header.test.jsx
import React from 'react';
import '@testing-library/jest-dom/extend-expect';
import { render, screen } from '@testing-library/react';
import Header from '../Header';
import { MemoryRouter } from 'react-router-dom';

describe('Header Component', () => {
  beforeEach(() => {
    render(
      <MemoryRouter>
        <Header />
      </MemoryRouter>
    );
  });

  it('renders the navigation container', () => {
    const nav = screen.getByRole('navigation');
    expect(nav).toBeInTheDocument();
  });

  it('has a title link pointing to the root', () => {
    const titleLink = screen.getByText('Навигация');
    expect(titleLink).toBeInTheDocument();
    expect(titleLink).toHaveAttribute('href', '/');
    expect(titleLink).toHaveClass('Title');
  });

  it('renders all menu links with correct hrefs and classes', () => {
    const links = [
      { text: 'Проекты', href: '/Projects', className: 'Links' },
      { text: 'Договоры', href: '/Contracts', className: 'Links' },
      { text: 'BIQ', href: '/BIQ', className: 'Links' },
      { text: 'Создать проект', href: '/CreateProject', className: 'CreateProject' },
    ];

    links.forEach(({ text, href, className }) => {
      const linkEl = screen.getByText(text);
      expect(linkEl).toBeInTheDocument();
      expect(linkEl).toHaveAttribute('href', href);
      expect(linkEl).toHaveClass(className);
    });
```

Рисунок 26 – Листинг юнит-тестов Header.

Для страницы MainPage использовала файл тестов: MainPage.test.js. В файле MainPage.test.js реализован модульный тест, который проверяет корректную отрисовку главного баннера приложения. Сначала компонент MainPage монтируется в виртуальное DOM-окружение с помощью React Testing Library (render(<MainPage />)), далее выполняет основные проверки [19].

src > tests > JS MainPage.test.js > ...

```
// src/tests/MainPage.test.js
import React from 'react';
import { render, screen } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';
import MainPage from '../pages/MainPage';

describe('MainPage', () => {
  it('рендерит ключевую информацию о приложении', () => {
    render(<MainPage />);
    expect(screen.getByText(/возможности приложения/i)).toBeInTheDocument();
  });
});
```

Рисунок 27 – Листинг юнит-тестов MainPage

Как показано на рисунке 27, на главной странице все работает корректно и без ошибок.

Для страницы Projects использовала файл тестов: Projects.test.js. Как показывает рисунок 28, он проверяет отображение сообщения при пустом списке проектов.

src > tests > JS Projects.test.js > ...

```
// src/tests/Projects.test.js
import React from 'react';
import { render, screen } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';
import Projects from '../pages/Projects';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';

const queryClient = new QueryClient();
jest.mock('axios', () => ({ get: () => Promise.resolve({ data: [] }) }));

describe('Projects', () => {
  it('показывает сообщение при пустом списке', async () => {
    render(
      <QueryClientProvider client={queryClient}>
        <Projects />
      </QueryClientProvider>
    );
    expect(await screen.findByText(/проектов не найдено/i)).toBeInTheDocument();
  });
});
```

Рисунок 28 – Пример листинга юнит-тестов ProjectPage

Для страницы ProjectPage использовала файл тестов: ProjectPage.test.js. Он включает два сценария: отображение полей с данными проекта; рендер списка BIQ-задач и проверка удаления.

Пример из кода показан на рисунке 29.

```
68  test('displays project information correctly', async () => {
69    render(<ProjectPage />, { wrapper: createWrapper() });
70
71    expect(await screen.findByDisplayValue('my-project')).toBeInTheDocument();
72    expect(screen.getByDisplayValue('12345')).toBeInTheDocument();
73    expect(screen.getByDisplayValue('2025-01-01')).toBeInTheDocument();
74    expect(screen.getByDisplayValue('2025-12-31')).toBeInTheDocument();
75    expect(screen.getByDisplayValue('10%')).toBeInTheDocument();
76    expect(screen.getByDisplayValue('50000')).toBeInTheDocument();
77    expect(screen.getByDisplayValue('20')).toBeInTheDocument();
78    expect(screen.getByDisplayValue('30')).toBeInTheDocument();
79  });
80
81  test('renders BIQ tasks and handles deletion', async () => {
82    render(<ProjectPage />, { wrapper: createWrapper() });
83
84    const taskElement = await screen.findByText('Task One');
85    expect(taskElement).toBeInTheDocument();
86
87    const deleteButton = screen.getByText('×');
88    fireEvent.click(deleteButton);
89
90    await waitFor(() => {
91      expect(axios.delete).toHaveBeenCalledWith(
92        'http://localhost:2024/projects/my-project/tasks/Task One'
93      );
94      expect(axios.get).toHaveBeenCalledTimes(3);
95    });
```

Рисунок 29 – Листинг юнит-тестов Projects

После запуска npm run test:unit достигается не менее 80 % строк и ветвлений. Каждый коммит в CI запускает юнит-тесты, гарантируя отсутствие регрессий (Рисунок 30).

```
Test Suites: 7 passed, 7 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        2.764 s
Ran all test suites.
PS C:\Users\death\Desktop\Projects\diplom>
```

Рисунок 30 – Результат тестирования

Юнит-тесты подтвердили корректность ключевых модулей фронтенда и служат надёжным фундаментом для интеграционного и функционального тестирования. Результат тестирования продемонстрирован на рисунке 30.

3.2 Интеграционное тестирование

Интеграционные тесты проверяют работу сразу нескольких уровней приложения - маршрутов Express, middleware загрузки файлов и взаимодействие с базой данных. В нашем проекте они реализованы в файле app.test.js с использованием Supertest и mongodb-memory-server, что позволяет «поднять» MongoDB в-памяти и изолированно прогонять полные HTTP-сценарии [13].

Среда:

– в блоке beforeAll запускается in-memory сервер MongoDB, на который перенастраивается process.env.MONGO_URI [15];
– подключение Mongoose идёт к этому URI, после чего импортируется Express-приложение (app) [13], [16];
– в afterAll завершается соединение с БД и останавливается in-memory сервер.

Создание проекта, показано на рисунке 31.

Основные сценарии, покрываемые тестами:

```
it('POST /createProject → 200 & возвращает созданный проект', async () => {
  const res = await request(app)
    .post('/createProject')
    .field('title', projectTitle)
    .field('number', '42')
    .field('startDate', '2025-01-01')
    .field('endDate', '2025-12-31')
    .field('rate', '5')
    .field('price', '1000')
    .field('externalLabor', '10')
    .field('internalLabor', '20')
    // без реального файла, но multer пропустит пустую запись:
    .attach('attachment', Buffer.from(''), 'dummy.txt');

  expect(res.status).toBe(200);
  expect(res.body.result).toMatchObject({
    title: projectTitle,
    number: '42',
    price: 1000,
    rate: 5,
    externalLabor: 10,
    internalLabor: 20,
  });
```

Рисунок 31 - Соответствующий лог-скриншот Создание проекта

Чтение списка проектов, продемонстрировано на рисунке 32.

```
it('GET /projects → содержит наш проект', async () => {
  const res = await request(app).get('/projects');
  expect(res.status).toBe(200);
  expect(Array.isArray(res.body)).toBe(true);
  expect(res.body.find(p => p.title === projectTitle)).toBeDefined();
});
```

Рисунок 32 - Соответствующий лог-скриншот Чтение списка проектов

Получение одного проекта, показано на рисунке 33.

```
it('GET /projects/:title → возвращает объект проекта', async () => {
  const res = await request(app).get(`/projects/${projectTitle}`);
  expect(res.status).toBe(200);
  expect(res.body.title).toBe(projectTitle);
});
```

Рисунок 33 - Соответствующий лог-скриншот Получение одного проекта

Рисунок 34 показывает создание и чтение BIQ-задачи.

```
it('POST /createBIQ → создаёт BIQ запись', async () => {
  const res = await request(app)
    .post('/createBIQ')
    .field('AcceptanceStatus', 'ok')
    .field('BIQName', biqId)
    .field('Curator', 'John')
    .field('DataStart_2', '2025-02-02')
    .field('Description', 'Desc')
    .field('Functionality', 'F')
    .field('Initiator', 'Alice')
    .field('LaborCosts', '7')
    .field('projectSelect', projectTitle)
    .attach('valueFile', Buffer.from(''), 'val.txt')
    .attach('ActFile', Buffer.from(''), 'act.txt')
    .attach('TestProtocolFile', Buffer.from(''), 'test.txt');

  expect(res.status).toBe(200);
  expect(res.body.ok).toBe(true);
  expect(res.body.result.BIQName).toBe(biqId);
});

it('GET /projects/:title/tasks → содержит созданный BIQ', async () => {
  const res = await request(app).get(`/projects/${projectTitle}/tasks`);
  expect(res.status).toBe(200);
  expect(Array.isArray(res.body)).toBe(true);
  expect(res.body.find(b => b.BIQName === biqId)).toBeDefined();
});
```

Рисунок 34 - Соответствующий лог-скриншот Создание и чтение BIQ-задачи

Удаление BIQ-задачи и проекта, как показано на рисунке 35.

```
it('DELETE /projects/:title/tasks/:taskTitle → удаляет запись', async () => {
  const res = await request(app).delete(`/projects/${projectTitle}/tasks/${biqId}`);
  expect(res.status).toBe(200);
  expect(res.body.ok).toBe(true);

  const after = await request(app).get(`/projects/${projectTitle}/tasks`);
  expect(after.body.find(b => b.BIQName === biqId)).toBeUndefined();
});

it('DELETE /projects/:title → удаляет проект', async () => {
  const res = await request(app).delete(`/projects/${projectTitle}`);
  expect(res.status).toBe(200);
  expect(res.body.ok).toBe(true);

  const after = await request(app).get('/projects');
  expect(after.body.find(p => p.title === projectTitle)).toBeUndefined();
});
```

Рисунок 35 - Соответствующий лог-скриншот Удаление BIQ-задачи и проекта

На рисунке 36 представлен результат тестов.

```
Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        1.38 s, estimated 37 s
Ran all test suites.
PS C:\Users\death\Desktop\Projects\server>
```

Рисунок 36 – Результат интеграционного тестирования

Таким образом интеграционные тесты подтверждают работоспособность ключевых API-маршрутов и их корректную интеграцию с базой данных и сервисом загрузки файлов.

3.3 Функциональное тестирование

Функциональное тестирование направлено на проверку соответствия реальной работы системы заявленным требованиям и пользовательским сценариям. Приведены результаты функционального тестирования в виде сводной таблицы 4.

Таблица 4 -Таблица результатов функционального тестирования

Вид проверки	Ожидаемый результат	Фактический результат
Создание проекта	Переход на страницу «Создать проект», ввод всех полей (наименование, номер договора, даты, ставка, сумма, трудозатраты), клик «Сохранить».	Результат соответствует ожидаемому
Удаление проекта	На странице списка проектов клик «Удалить» на карточке тестового проекта и подтверждение операции.	Результат соответствует ожидаемому
Хранение ключевой информации о проекте	Открытие страницы проекта и проверка корректного отображения всех введённых при создании полей: наименование, договор, даты, ставка, сумма, трудозатраты.	Результат соответствует ожидаемому
Загрузка документа проекта	На вкладке «Документы» проекта загрузка файла (договор/акт/протокол), указание типа, и проверка появления записи с кнопкой «Скачать».	Результат соответствует ожидаемому
Создание BIQ-задачи	На вкладке «Задачи (BIQ)» клик «Новая задача», ввод всех полей (BIQName, функционал, инициатор, куратор, трудозатраты, дата оценки, статус, описание).	Результат соответствует ожидаемому
Удаление BIQ-задачи	В списке BIQ-задач клик «Удалить» рядом с тестовой задачей и подтверждение операции.	Результат соответствует ожидаемому
Проверка атрибутов BIQ-задачи	Открытие деталей задачи и сверка всех полей (BIQName, функционал, инициатор, куратор, трудозатраты, дата оценки, статус, описание) с вводимыми данными.	Результат соответствует ожидаемому
Загрузка документов в BIQ-задаче	В форме задачи добавление файлов оценки, акта и протокола, сохранение и проверка их отображения в списке вложений с возможностью скачивания.	Результат соответствует ожидаемому

Каждый из этих сценариев был выполнен вручную, чтобы убедиться в стабильности и предсказуемости поведения системы.

В проекте были разработаны и выполнены тест-кейсы, которые охватывают основные операции:

– создание и удаление проектов: проверка добавления проекта с заполнением всех полей и его удаления;
– хранение ключевой информации: все параметры сохраняются и отображаются
– управление документами: возможность загрузки договоров, актов и протоколов;
– работа с задачами (BIQ): тестирование сценариев создания и удаления BIQ-задач, а также проверки всех атрибутов.

Функциональные тесты показали, что все ключевые сценарии - от создания и удаления проектов до работы с BIQ-задачами и загрузки документов - выполняются без ошибок, что подтверждает надёжность системы.

3.4 Нагрузочное тестирование

Нагрузочное тестирование с помощью команды npx loadtest http://localhost:2024/projects (10 секунд) позволяет проверить, как API справляется с пиковыми запросами и получить основные метрики времени отклика и пропускной способности (Рисунок 37).

```
Completed requests:  5817
Total errors:        0
Total time:          10.036 s
Mean latency:        169.7 ms
Effective rps:       580

Percentage of requests served within a certain time
  50%      155 ms
  90%      223 ms
  95%      264 ms
  99%      315 ms
 100%      547 ms (longest request)
```

Рисунок 37 – Результат нагрузочного тестирования

Представлены данные с пояснениями:

- общее число запросов: 5 817 за 10 секунд (≈ 582 RPS),
- ошибок нет (Total errors: 0),
- среднее время отклика: 169,7 мс,
- 95-й перцентиль: 264 мс, что удовлетворяет порогу (<300 мс),
- максимальное время: 547 мс, зафиксировано единичное замедление.

Выводы к третьей главе:

- система уверенно держит нагрузку в 500–600 запросов в секунду без отказов и с достаточной задержкой;
- все SLA-метрики выполнены: 95 % запросов обрабатываются быстрее 300 мс;
- отсутствие ошибок подтверждает устойчивость API и надёжность базы данных при одновременной работе множества пользователей.

Такие результаты свидетельствуют о готовности сервиса к эксплуатации и масштабируемости при увеличении числа пользователей.

Заключение

Результатом выпускной квалификационной работы является веб-приложение для автоматизации управления проектами в условиях многозадачности, предназначенное для повышения прозрачности и эффективности бизнес-процессов предприятия.

В первом разделе работы проведён анализ деятельности компании и существующих процессов управления проектами. На основе обследования текущего сценария (работа через Google Docs, CBH/SVN) были смоделированы UML-диаграммы бизнес-процессов («как было»), что позволило выявить фрагментацию данных, сложности с доступом и безопасность. Данный анализ обосновал необходимость создания единой платформы и лег в основу формулировки требований и обзора аналогичных решений.

Во втором разделе выполнено проектирование веб-приложения. Выделены основные сущности (Проект, Задача/BIQ, Документ, Функциональный блок), разработана структура базы данных в MongoDB и построены её модели. Спроектирована REST-архитектура серверной части (Express.js/Node.js) и интерфейс клиента (React), определены JSON-контракты API в формате OpenAPI.

В третьем завершающем этапе проведено тестирование:

- юнит-тесты
- интеграционные тесты
- функциональные тесты
- нагрузочные тесты

Цель выпускной квалификационной работы достигнута: приложение реализовано в полном объёме, удовлетворяет заданным функциональным и качественным требованиям, готово к внедрению и дальнейшему развитию.

Список используемой литературы и используемых источников

1. Барклунд М., Мардан А. React Quickly: практическое руководство. 2-е изд. – СПб: Прогресс книга, 2023. – 797 с.
2. Брэдшоу Ш., Ходоров К., Брэзил Й. MongoDB: Полное руководство. ДМК Пресс, 2020. – 540 с.
3. Буч Г., Рембо Д., Якобсон И. Язык UML. Руководство пользователя. М.: ДМК-Пресс, 2008. – 496 с.
4. Бэнкер К. MongoDB в действии. ДМК Пресс, 2014. – 381 с.
5. Бэнкс А., Порселло Е. React и Redux. Функциональная веб-разработка. – СПб: Питер, 2018. – 336 с.
6. ГОСТ Р 54869-2011. Проектный менеджмент. Требования к управлению проектом. URL: https://docs.cntd.ru/document/1200089604 (дата обращения: 02.11.2025)
7. Дакет Дж. HTML и CSS. Разработка и дизайн веб-сайтов. Эксмо, 2020. – 480 с.
8. Дунаев В. JavaScript. Самоучитель. Питер – М., 2015. – 400 с.
9. Мальцев И. П. Проектирование сайтов. М.: SelfPub, 2018. – 170 с.
10. Руководство по React [Электронный ресурс]. URL: https://metanit.com/web/react/ (дата обращения: 03.12.2024)
11. Хэррон Дэвид. Node.js. Разработка серверных веб-приложений на JavaScript. М.: ДМК Пресс, 2014. – 144 с.
12. DFD (Data Flow Diagram) Диаграммы — зачем они нужны и какие бывают [Электронный ресурс]. URL: https://habr.com/ru/articles/668684/ (дата обращения: 24.01.2025)
13. Express 5.x - API Reference [Электронный ресурс]. URL: https://expressjs.com/en/5x/api.html (дата обращения: 12.11.2024)
14. JavaScript | MDN [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата обращения: 11.11.2024)

15. MongoDB University. M320: Data Modeling. MongoDB, Inc., 2025. URL: https://university.mongodb.com (дата обращения: 11.12.2024)

16. Mongoose v8.15.1: Getting Started [Электронный ресурс]. URL: https://mongoosejs.com/docs/ (дата обращения: 12.11.2024)

17. Project Documentation: 20 Essential Project Documents [Электронный ресурс]. URL: https://www.projectmanager.com/blog/great-project-documentation (дата обращения: 09.10.2024)

18. React Reference Overview - React [Электронный ресурс]. URL: https://react.dev/reference/react (дата обращения: 03.11.2024)

19. React Router Home | React Router [Электронный ресурс]. URL: https://reactrouter.com/home (дата обращения: 06.11.2024)

20. UML 2.5 Diagrams Overview [Электронный ресурс]. URL: https://www.uml-diagrams.org/uml-25-diagrams.html (дата обращения: 07.02.2025)