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 направление подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

На тему «Разработка веб-системы управления проектами для планирования, координации и контроля выполнения командных проектов»

Обучающийся Д.П. Пак
(Инициалы Фамилия) (Личная подпись)

Руководитель кандидат пед. наук, доцент, Е.А. Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

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

Тольятти 2025

Аннотация

Тема работы: “Разработка веб-системы управления проектами для планирования, координации и контроля выполнения командных проектов”. Работа направлена на исследование существующих систем управления проектами с целью выделения их недостатков и разработку новой системы.

Цель работы – разработка веб-системы управления проектами, обеспечивающий эффективное планирование, координацию и контроль выполнения командных проектов.

Бакалаврская работа состоит из введения, трех глав, заключения, списка используемой литературы и приложений.

Введение описывает актуальность темы, цели и задачи работы, определяет объект, предмет и методы исследования, а также практическую ценность.

В первой главе описывается компания, ее организационная структура с бизнес-процессами и проводится анализ существующих разработок.

Во второй главе проводится проектирование разрабатываемого веб-приложения, выбор технологий и реализация функциональных требований. В рамках проектирования ПО создается ряд диаграмм в нотации UML – это диаграммы развертывания, классов, компонентов и вариантов использования, а также логическая модель данных. Помимо UML диаграмм, во второй главе описываются реализованные алгоритмы с помощью блок-схем, а также демонстрируется графический интерфейс приложения.

Третья глава посвящена тестированию. Описан план тестирования, показаны тестовые сценарии для применяемых видов тестирования и составлен отчет об обнаруженных ошибках.

В заключении подводится итог работы, перечисляются выполненные задачи и описывается ценность работы.

Работа выполнена в объеме 67 страниц, содержит 37 рисунка, 7 таблиц и 7 приложений.

Abstract

The title of the graduation work is «Development of a web-based project management system for planning, coordinating, and monitoring team projects».

The graduation work consists of an introduction, three chapters, 39 figures, 7 tables, 7 appendices, a conclusion, and a list of 21 references including foreign sources.

The aim of this graduation work is to develop a new web application that provides enhances planning, coordination, and control of team projects.

The object of the graduation work is modern project management systems.

The subject of the graduation work is approaches and techniques for improving the effectiveness of project management.

The key issue of the graduation work is to analyze the drawbacks of existing project management web applications.

The graduation work may be divided into several logically connected chapters which are review of the company's business processes and analysis of application analogues with definition of requirements, application design and implementation and testing.

The first chapter describes in details the company and its business processes, analyzes existing web-based project management applications, and defines software requirements in accordance with the FURPS+ model.

The second chapter outlines the results of design and implementation of the program, as well as selected technologies and development tools. As part of the design, diagrams were drawn in UML notation explaining the structure and operation of the application. The implementation is described using flowcharts of the main algorithms and screenshots of the graphical user interface.

The third chapter consists of a test plan, functional and integration testing scenarios, and a bug report.

In conclusion we’d like to stress the key aspects of the developed application and its practical value for organizations.

Оглавление

Введение 5

Глава 1. Функциональное моделирование предметной области 7

1.1Деятельность и организационно-экономическая характеристика компании “ВорлдИнтерТех РУС” 7

1.2Концептуальное моделирование предметной области компании 9

1.3 Определение функциональных и нефункциональных требований 18

Глава 2. Процесс разработки программного обеспечения 21

2.1 Логическое и физическое проектирование 21

2.2 Выбор технологий и инструментов для разработки 35

2.3 Реализация функциональных требований 37

Глава 3. Тестирование и отладка приложения 59

3.1 План тестирования 59

3.2 Интеграционное тестирование 60

3.3 Функциональное тестирование 61

3.4 Обнаруженные ошибки 61

Заключение 66

Список используемой литературы 68

Приложение А Диаграмма “Как есть” 70

Приложение Б Фрагмент кода модуля userRouter.js 71

Приложение В Фрагмент кода модуля projectRouter.js 77

Приложение Г Блок схемы алгоритмов 82

Приложение Д Фрагмент кода модуля boardRouter.js 84

Приложение Е Фрагмент кода модуля cardRouter.js 89

Приложение Ж Тестовые сценарии 96

Введение

Современные информационные технологии играют ключевую роль в эффективном управлении проектами, особенно в эпоху цифровизации бизнеса. Веб-система управления проектами становится незаменимым инструментом для координации и контроля выполнения командных задач. В условиях стремительного развития цифровых платформ и увеличения объемов данных успешность предприятия напрямую зависит от способности быстро реагировать на изменения и эффективно управлять ресурсами.

Развитие информационных технологий обусловило необходимость внедрения специализированных инструментов для оптимизации процессов планирования, координации и контроля проектов. Наличие удобной, интуитивно понятной и функциональной веб-системы управления проектами способствует не только повышению производительности команды, но и улучшению коммуникации между участниками, сокращению сроков реализации проектов и повышению качества конечного продукта. Особую значимость приобретают облачные решения, позволяющие обеспечить масштабируемость системы и доступ к проектной информации из любой точки мира, что особенно актуально в условиях роста популярности гибридного и удаленного форматов работы.

Проблема разработки веб-систем управления проектами активно исследуется специалистами в области IT-технологий. Среди известных авторов, внесших значительный вклад в изучение этой тематики, можно отметить Томаса Демарко, Стивена Макконнелла, Эрика Райса и др., чьи труды служат основой для понимания принципов проектирования эффективных систем управления проектами.

Объектом работы являются современные веб-приложения управления проектами.

Предметом выступают механизмы и методы повышения эффективности управления проектами посредством разработки веб-приложения.

Цель работы — разработка веб-системы управления проектами, обеспечивающей эффективное планирование, координацию и контроль выполнения командных проектов. В рамках достижения цели будут решены следующие задачи:

- проведение анализа существующих веб-систем управления проектами;

- определение требований к разрабатываемой системе по модели FURPS+, исходя из потребностей пользователей и требований к современным веб-приложениям;

- реализация клиентской и серверной частей с использованием современных технологий (React.js, HTML5, CSS3, SCSS, JavaScript, Node.js, Express.js, Mongoose, MongoDB);

- тестирование разработанной системы [7].

Практическая ценность настоящего исследования заключается в предоставлении инструмента, позволяющего оптимизировать рабочие процессы, повысить эффективность коллективной работы и сократить временные издержки. Система предназначена для широкого круга пользователей, включая индивидуальных разработчиков, творческие коллективы и малые и средние предприятия, стремящиеся повысить свою конкурентоспособность на современном рынке.

Глава 1. Функциональное моделирование предметной области

1.1 Деятельность и организационно-экономическая характеристика компании “ВорлдИнтерТех РУС”

IT-компания “ВорлдИнтерТех РУС” специализируется на разработке компьютерного программного обеспечения, включая, но не ограничиваясь разработкой облачных сервисов, цифровизации бизнеса и интернет маркетинг. Офис компании находится в городе Тольятти по адресу ул. Ушакова 48, офис 41. Компания ООО "ВорлдИнтерТех РУС" образовалась относительно недавно - зарегистрирована 29 ноября 2023 года в. Чтобы поддерживать постоянный темп развития и успешно конкурировать с многими другими IT-организациями, “ВорлдИнтерТех РУС” заинтересовано в безопасном, удобном и минималистичном инструменте управления проектами, который можно использовать локально внутри компании.

Полное наименование – ООО “ВорлдИнтерТех РУС”.

На рисунке 1 проиллюстрирована организационная структура компании [8]:

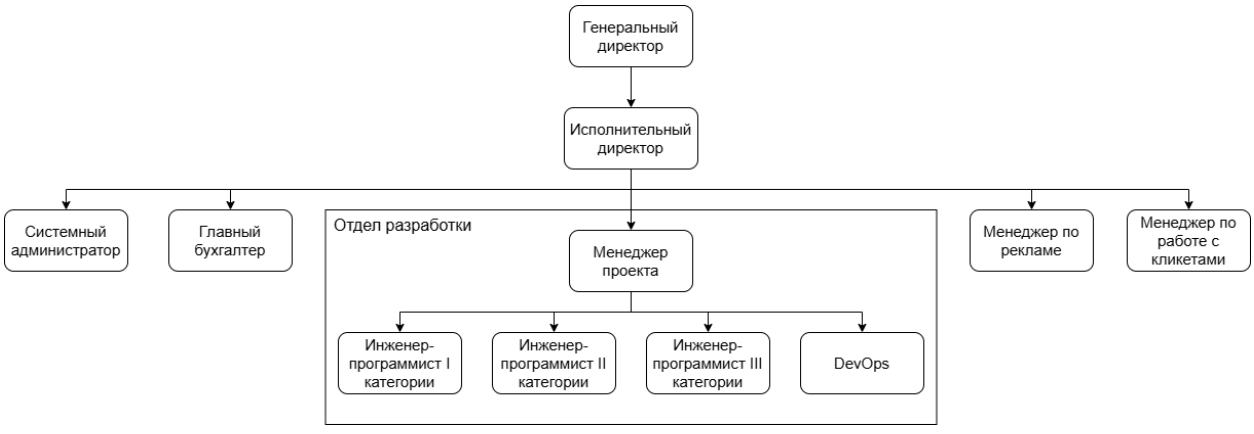


Рисунок 1 - Организационная структура предприятия “ВорлдИнтерТех РУС”

Рассмотрим организационную структуру более подробно:

- генеральный директор. Управление компанией;
- исполнительный директор. Управляет компанией, находится непосредственно в подчинении генерального директора и выполняет его указания;
- администратор. Настраивает оборудование организации;
- главный бухгалтер. Занимается бухгалтерской деятельностью.
- менеджер по рекламе. Занимается маркетингом;
- менеджер по работе с клиентами. Выполняет роль связующего звена между клиентом и менеджером проекта. Занимается с клиентами;
- инженер-программист III категории. Его сфера деятельности сильно ограничена в связи с небольшим опытом работы и ограниченностью знаний в области разработки ПО. Выполняет небольшие задачи по сопровождению и улучшению уже разработанных сервисов. Также может брать более сложные задачи с целью повышения своей квалификации. Как правило, инженер-программист III категории сильно зависит от более опытных программистов;
- инженер-программист II категории. Полностью самостоятельный. Может курировать младших программистов и помогать новым сотрудникам влиться в коллектив. Может полностью самостоятельно спроектировать и разработать небольшой модуль;
- инженер-программист I категории. Имеет большой опыт разработки, лучше всех ориентируется в предметной области. Занимается исправлением сложных багов, проектированием программного обеспечения, занимается разработкой новых сервисов. Помимо этого, занимается проверкой решений более младших программистов;
- менеджер проекта. Менеджер управляет ресурсами проекта, совместно с инженером-программистом III категории устанавливает сроки выполнения тех или иных задач и следит за их выполнением;

– DevOps – имеющий опыт разработки программного обеспечения сотрудник, занимающийся настройкой серверов и развертыванием готового программного продукта в производственной среде.

1.2 Концептуальное моделирование предметной области компании

1.2.1 Описание бизнес-процессов компании

На рисунке 2 представлена диаграмма бизнес-процессов отдела разработки [21].

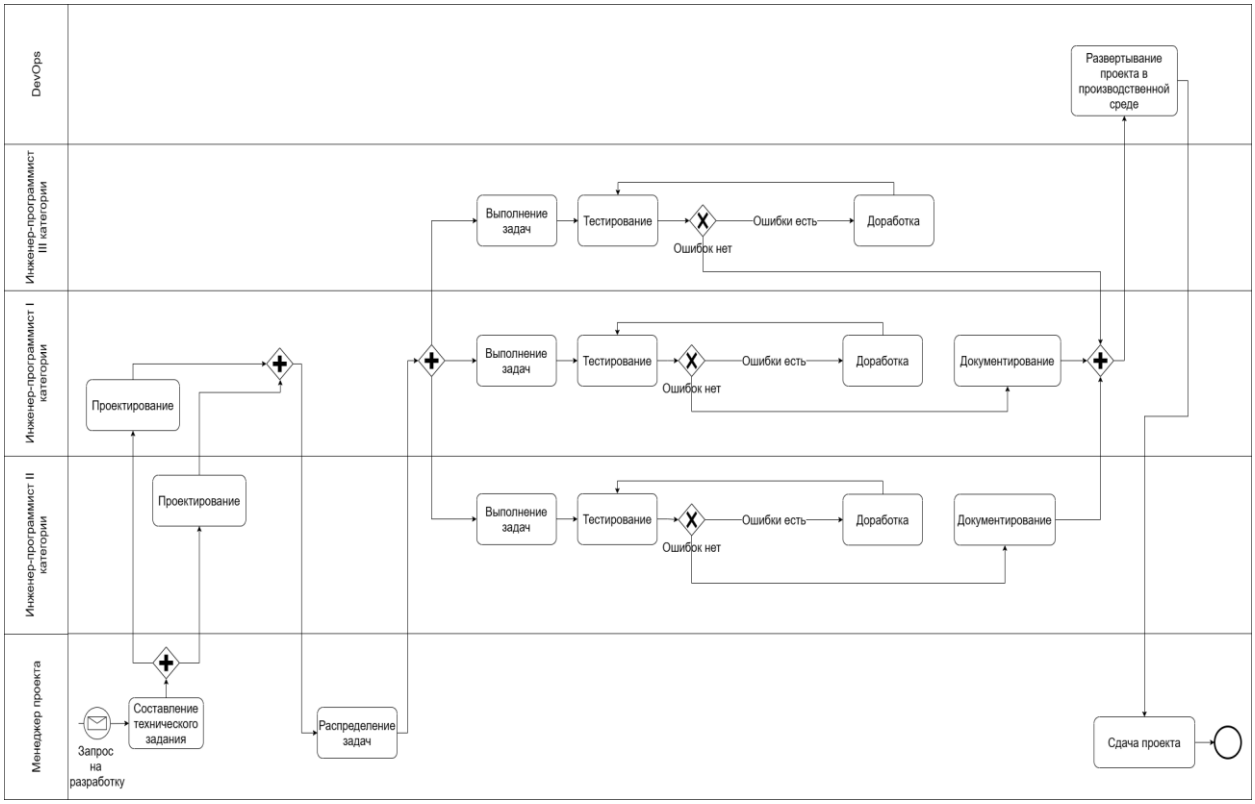


Рисунок 2 - BPMN диаграмма отдела разработки

Описание бизнес-процессов, представленных на диаграмме рисунка 2:

– составление технического задания – менеджер проекта составляет техническое задание, которое затем используется инженерами-программистами первой и второй категорий при проектировании ПО;

- проектирование ПО - инженеры-программисты первой и второй категорий выполняют проектирование программы, например, составляя диаграммы в нотации UML и описывая основные алгоритмы приложения с помощью блок-схем;
- распределение задач – после завершения этапа проектирования ПО, менеджер проекта распределяет задачи между сотрудниками в соответствии с их навыками и компетенциями;
- выполнение задач – после распределения задач инженеры-программисты выполняют их;
- тестирование –программисты проверяют реализацию своих задач. На этом этапе проверяется как работоспособность самого кода, так и соответствие разработанного функционала требованиям задачи;
- доработка – в случае, если в результате тестирования были найдены ошибки, инженеры-программисты исправляют их, после чего повторно тестируют;
- развертывание проекта в производственной среде – выполняется DevOps специалистом после того, как все программисты выполнили свои задачи, провели тестирование, исправили возникшие ошибки и задокументировали разработанное решение;
- сдача проекта – после успешного развертывания проекта, менеджер проекта передает проект заказчику.

В результате работы подразделения создается следующая документация:

- схемы и диаграммы работы ПО;
- инструкция по установке ПО;
- отчет тестирования;
- отчет об ошибках;
- руководство администратора;
- инструкция по запуску ПО;

– документация для разработчиков по внесению изменений;

– инструкция по эксплуатации ПО.

Данная документация передается менеджерам по работе с клиентами во время сдачи проекта заказчику.

На рисунках 3-4 представлено описание информационных потоков отдела разработки с помощью диаграммы IDEF0 [16].

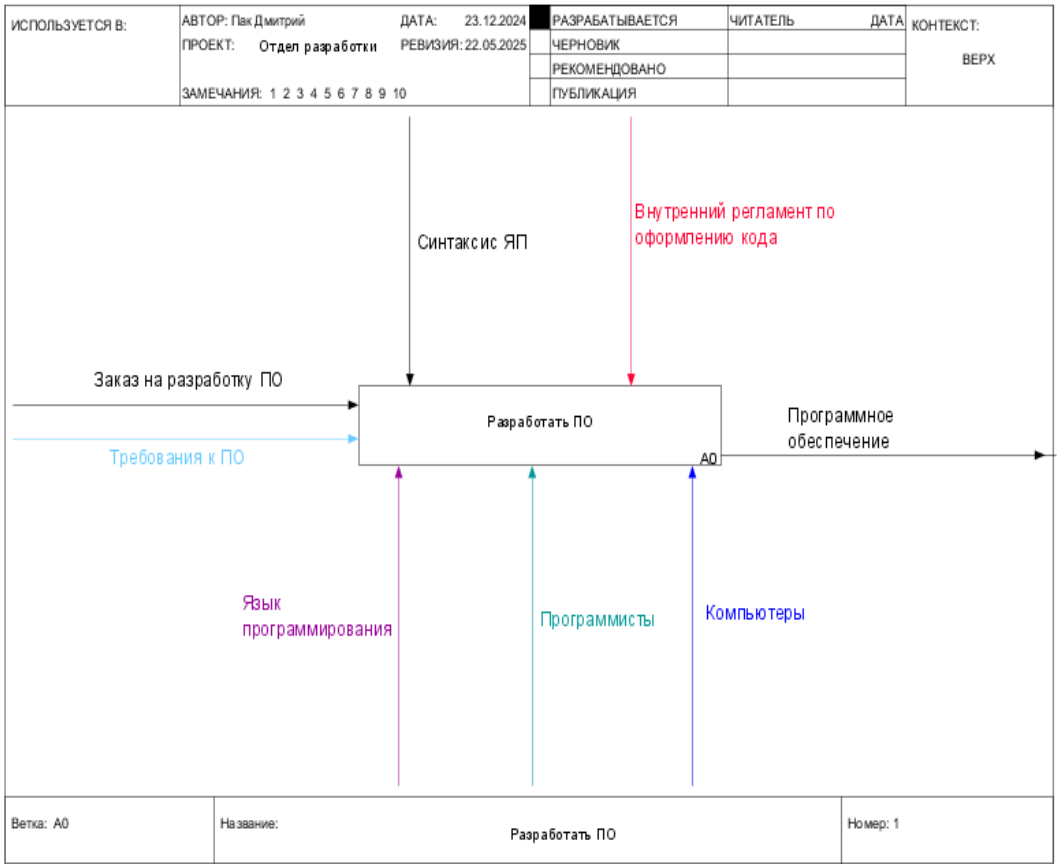


Рисунок 3 - Основный процесс отдела разработки

Основная функция отдела разработки – разрабатывать программное обеспечение по заказу. На схеме (рисунок 3) видно, что на вход идет «Заказ на разработку ПО», в роли механизмов выступают программисты с компьютерами и языком программирования, в качестве управления выступают синтаксис ЯП и внутренний регламент компании по оформлению кода. В результате на выходе появляется готовое программное обеспечение.

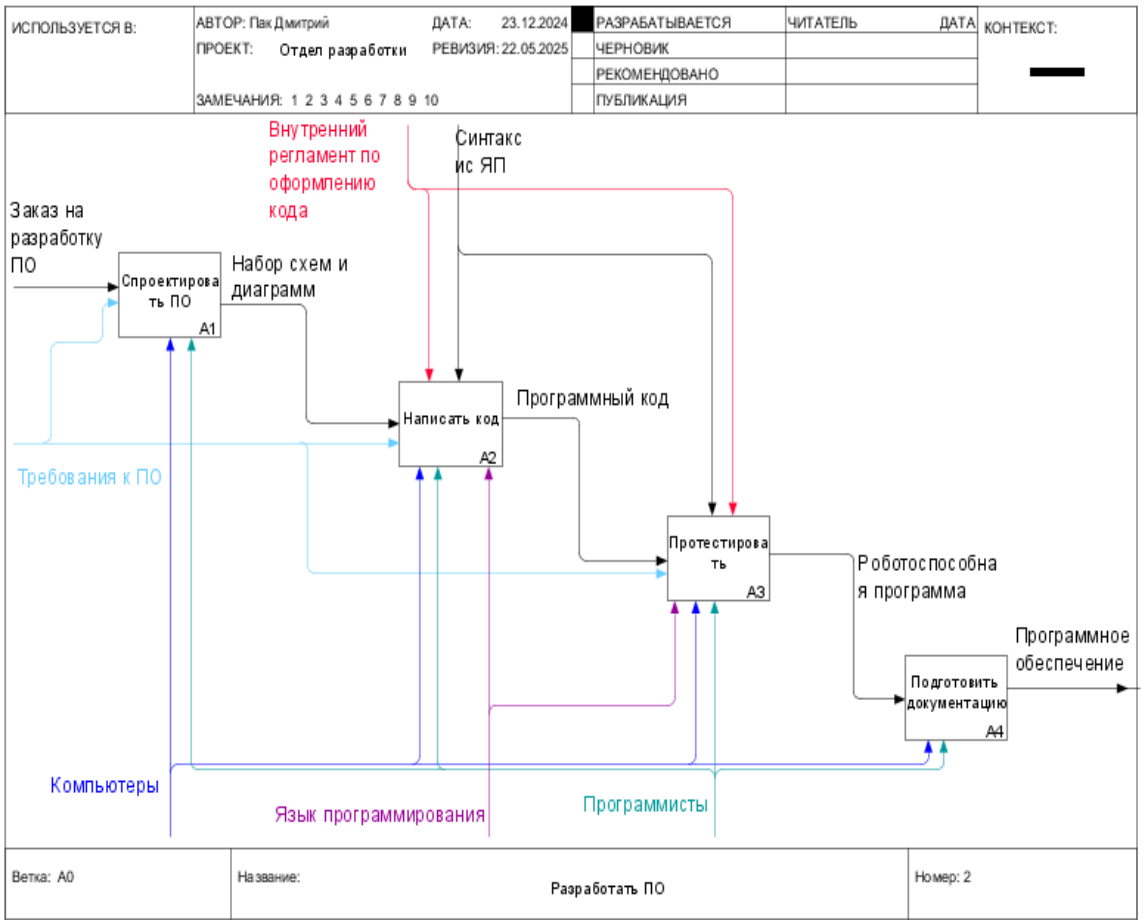


Рисунок 4 - Декомпозиция процесса “Разработать ПО”

На рисунке 4 проиллюстрирована декомпозиция основного процесса. Процесс “Разработать ПО” разбивается на следующие процессы:

- Спроектировать ПО – создание схем работы и взаимодействия компонентов ПО. Эти схемы используются для написания программного кода.
- Написать код – используя язык программирования программисты, ориентируясь на схемы, пишут программный код.
- Протестировать – используя написанный код, программисты проводят тестирование кода, в результате чего получается стабильно работающая программа.

– Подготовить документацию – основываясь на работающей программе, программисты составляют пакет документации по установке, настройке и эксплуатации ПО. В результате процесса на выходе получается готовое программное обеспечение.

1.2.2 Разработка и анализ бизнес-модели “Как есть”

Концептуальная диаграмма вариантов использования “Как есть” [9] показывает, каким образом сотрудники компании управляют проектами на данный момент, до разработки и интеграции нового решения. В Приложении А представлена диаграмма “Как есть”. На диаграмме продемонстрировано, что сотрудники предприятия используют решение по управлению проектами - сервис Kaiten - от сторонних разработчиков, соответственно данные о проектах компании, составе участников этих проектов и их конфиденциальная информация хранятся непосредственно на удаленных серверах тех же сторонних разработчиков. Это несет в себе потенциальные риски, так как сторонние разработчики могут неправильно обрабатывать конфиденциальную информацию и информацию, содержащую коммерческую тайну компании “ВорлдИнтерТех РУС”. Помимо этого, используемый ресурс может оказаться недоступен из-за внешних обстоятельств, таких как политических разногласий, войн, вооруженных конфликтов, стихийных бедствий и т.д. Отсюда появляется необходимость использовать собственно разработанную систему управления проектами [19].

1.2.3 Разработка и анализ бизнес-модели “Как должно быть”

Рассмотрим Use-case диаграмму “Как должно быть” - она позволяет детально проанализировать процесс ведения проектов, используя разработанное веб-приложение по управлению проектами, которое развернуто непосредственно в локальной сети компании. Данная “To-Be” диаграмма представлена на рисунке 5.

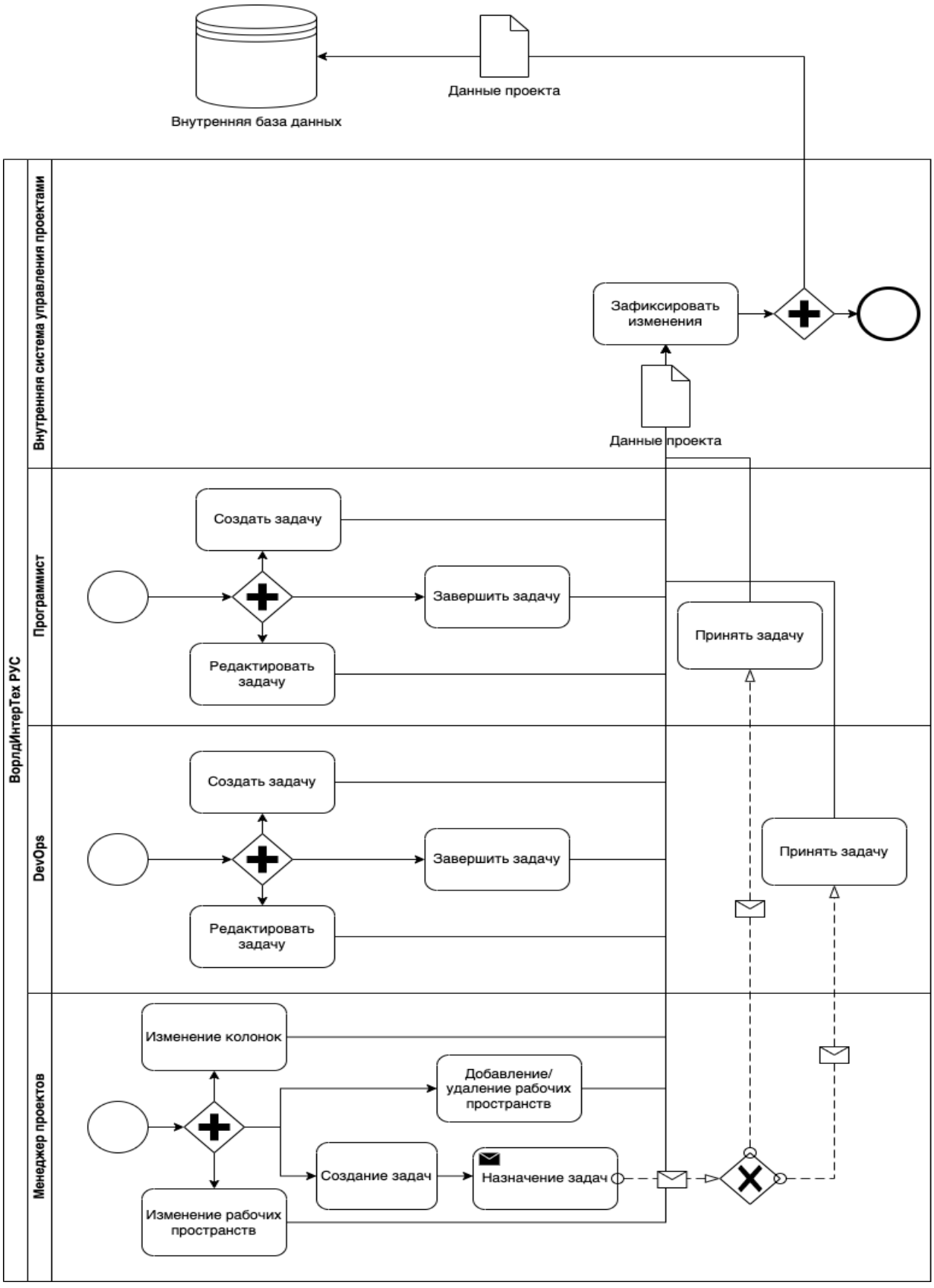


Рисунок 5 - Use-case диаграмма “Как должно быть”

Как видно из рисунка 5, теперь данные проекта вместе с базой данных находятся внутри компании на ее серверах, а не на серверах внешних поставщиков, как это было показано на диаграмме “Как есть”.

1.2.4 Анализ существующих разработок

Современные системы управления проектами позволяют отслеживать ход работ сотрудников и улучшить взаимодействие между членами команды, позволяя каждому сотруднику видеть, над чем работают его коллеги, и соответственно легко найти нужного специалиста для консультации или взаимопомощи.

Для анализа проведем сравнение шести существующих решений для управления проектами, представленные как российскими, так и зарубежными разработчиками - их сравнительная характеристика представлена в таблице 1.

Таблица 1 - Анализ конкурентов

Наименование	Преимущества	Недостатки	Стоимость и условия пользования
Kaiten	– Интеграция с GitHub, Slack; – Бесплатно для небольших команд.	– Закрытый исходный код; – Отсутствуют вложенные задачи; – Нельзя использовать локально внутри компании;	Бесплатно для небольших команд. От 420 рублей в месяц за одного пользования для расширенного функционала. Для развертывания на собственных серверах нужна Enterprise подписка, цена – договорная.
Redmine	– Открытый исходный код; – Возможно использовать локально внутри сети компании; – Есть аналитика	– Устаревший графический интерфейс; – Сложная настройка;	Полностью бесплатен

Продолжение таблицы 1

Наименование	Преимущества	Недостатки	Стоимость и условия пользования
Pyrus	– Бесплатно для небольших команд; – Автоматизация бизнес-процессов; – Можно использовать локально внутри компании.	– Закрытый исходный код; – Отсутствует аналитика;	Бесплатный для управления проектами (без аналитики, без использования на своих серверах и т.д.). 82500 рублей в месяц за тарифный план с возможностью развертывания внутри компании на своих серверах; 415 рублей в месяц за одного человека в проекте в тарифном плане с возможностью интеграции с другими сервисами и аналитикой.
Asana	– Есть аналитика; – Подзадачи; – Интеграция с Google Doc, Slack и Trello.	– Закрытый исходный код; – Нельзя использовать локально внутри компании; – Высокая стоимость	Бесплатно для личного пользования и небольших групп до 10 человек. В остальных случаях от 10.99$ до 24.49$ в год.
Jira	– Большое количество плагинов, расширяющих функционал; – Аналитические средства.	– Закрытый исходный код; – Высокая стоимость; – Сложность в освоении; – Недоступен в РФ; – Нельзя использовать локально внутри компании.	Бесплатно для команд до 10 человек, далее от 7,5 $ в месяц за одного пользователя.

Продолжение таблицы 1

Наименование	Преимущества	Недостатки	Стоимость и условия пользования
Trello	– Бесплатно для небольших команд; – Простой интерфейс; – Возможность комментировать задачи, добавлять метки к ним.	– Закрытый исходный код; – Отсутствует аналитика; – Недоступен в РФ; – Отсутствуют вложенные задачи; – Высокая стоимость – Нельзя использовать локально внутри компании.	Бесплатно для небольших команд до 10 человек (с ограничением числа Kanban-досок до 10). От 5$ до 17,50$ в месяц за пользователя.

Программные решения, представленные в таблице 1, имеют схожие преимущества и недостатки. В частности, почти все сервисы, кроме Redmine, имеют главный недостаток - закрытый исходный код и невозможность использование на серверах компании. Redmine же в свою очередь, имеет устаревший графический интерфейс и требует множества доработок.

Итого, исходя из таблицы 1, можно выделить несколько ключевых недостатков:

- стоимость;
- проприетарность систем - закрытый исходный код;
- ни одна из популярных систем не предоставляет возможности использовать ее в локальной сети;
- из-за санкций некоторые системы недоступны;
- сложность в освоении;
- перегруженность функционалом.

1.3 Определение функциональных и нефункциональных требований

Перед тем, как переходить к разработке, должны быть сформулированы функциональные и нефункциональные требования к веб-приложению.

Под функциональными требованиями понимается то, “каким должно быть поведение продукта в тех или иных условиях. Они определяют, что разработчики должны создать, чтобы пользователи смогли выполнить свои задачи” [4, с. 9].

Под нефункциональными требованиями подразумевается все то, что не делает программный продукт напрямую, но без чего не может обойтись. “т. Они могут описывать важные характеристики или свойства системы. К ним относятся доступность, легкость и простота использования, производительность и другие характеристики системы” [4, с. 10]

Определим требования к программному обеспечению по методологии FURPS+ [20].

Functionality – Функциональные требования:

- создание, редактирование и удаление проекта;
- создание, редактирование и удаление досок;
- создание, редактирование и удаление колонок;
- создание, редактирование и удаление задач;
- перемещение задач по колонкам;
- администратор может изменять и удалять доски, колонки и карточки задач;
- автор может изменять и удалять собственные проекты, а также доски, колонки и карточки своих проектов;
- рядовой участник проекта может создавать, редактировать и удалять собственные карточки задач;
- исполнитель задачи может оставлять комментарии к задаче, менять ее статус и перемещать по колонкам;

– проверяющий задачи может закрыть задачу, менять ее статус, перемещать по колонкам и комментировать.

Usability – Требования к удобству использования:

– текст не должен содержать грамматических ошибок;
– интерфейс приложения должен быть интуитивно понятным;
– текст должен быть читаемым;
– навигация в системе должна быть простой в использовании.

Reliability – Требования к надежности:

– ошибки не должны приводить к полному отказу ПО;
– доступность системы 24 часа в сутки 7 дней в неделю.

Performance – Требования к производительности:

– время запуска на серверной части < 10 секунд;
– время запуска на клиентской части < 5 секунд.

Supportability – Требования к поддержке:

– добавление нового функционала по запросу;
– исправление ошибок при обнаружении;
– развертывание на серверах компании в локальной сети;
– настройка серверной части через переменные среды.

Ограничения проектирования:

– диаграммы должны создаваться в цифровых системах проектирования (draw.io, MS Visio и другие);
– хранение реализовать в не реляционной БД.

Ограничения реализации:

– JavaScript в качестве основного языка программирования;
– MongoDB или Cassandra в качестве СУБД;
– клиентская часть должна быть реализована с помощью одного из следующих технологий: React, Vue, Angular;
– серверная часть должна быть реализована с помощью Node.js.

Безопасность:

- хеширование паролей;
- шифрование данных;
- использование шифрованного протокола передачи данных (HTTPS).

Вывод по главе 1.

В первой главе выпускной квалификационной работы был проведен анализ деятельности компании “ВорлдИнтертех РУС”. В рамках анализа смоделированы ключевые бизнес-процессы организации в нотации IDEF0, описаны бизнес-процессы и составлены диаграммы “Как есть” и “Как должно быть” по методологии BPMN. Помимо этого, дана организационно-экономическая характеристика компании.

По результатам, которые мы получили в ходе анализа, было принято решение отказаться от использования сервиса Kaiten и его аналогов в пользу разработки новой системы по управлению проектами, что обусловлено потенциально возможными проблемами с безопасностью и отсутствием контроля за исходным кодом программы.

Был проведен анализ конкурентов – уже существующих на рынке средств для управления проектами (Trello, Asana, Redmine и другие) – на основе результата анализа были сформулированы и распределены по категориям модели FURPS+ функциональные и нефункциональные требования к разрабатываемому ПО.

Глава 2. Процесс разработки программного обеспечения

“Проектирование ПО – процесс определения архитектуры, компонентов, интерфейсов, других характеристик системы и конечного результата” [10]. На этапе проектирования требуется определить архитектуру приложения, из каких частей оно будет состоять, каким образом эти части будут взаимодействовать друг с другом и внешним миром, определить, как данные буду храниться в системе. Для этого применяется ряд схем и диаграмм:

- Диаграмма вариантов использования (use case);
- Диаграмма классов;
- Диаграмма компонентов;
- Логическая модель данных;
- Диаграмма развертывания

Описанные выше диаграммы будут разобраны далее в этой главе.

2.1 Логическое и физическое проектирование

Для веб-сервиса по управлению проектами была выбрана архитектура клиент-сервер как наиболее простая и надежная архитектура для разработки веб-приложений, широко использующаяся в настоящее время.

Что же подразумевается под клиентом и сервером? Клиентом чаще всего выступает браузер конечного пользователя. Под сервером же понимается как само ПО, запущенное на компьютере и принимающее входящие запросы, так и специальный компьютер, на котором работает серверное ПО [14].

На рисунке 6 представлена диаграмма, показывающая принцип работы клиент-серверной архитектуры.

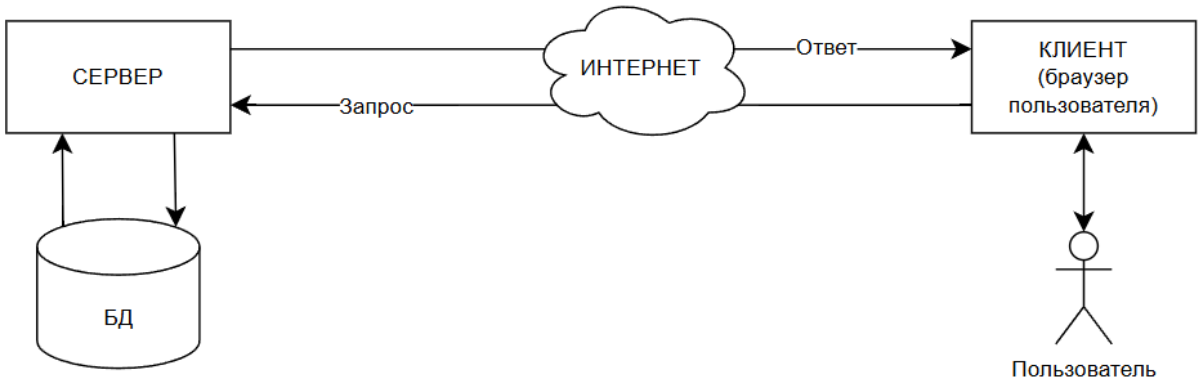


Рисунок 6 - Клиент-серверная архитектура

Клиент-серверная архитектура заключается в обмене запросами и ответами на них между клиентом и сервером, что можно видеть на рисунке 6. Клиент выступает в роли инициатора и посылает запрос на сервер, сервер обрабатывает запрос клиента и отсылает ему ответ [14].

Одной из особенности данной архитектуры является отделение бизнес логики приложения и данных от клиентской части, которая визуализирует то, что было отправлено сервером в качестве ответа.

2.1.1 Логическое проектирование системы

Логическое проектирование – это мост между концептуальной модели системы и физической ее реализации, определяющий, как она будет функционировать, не углубляясь в технические детали, делая тем самым спроектированную систему независимой от технологий и применяемых языков программирования. [11]

“ Диаграмма вариантов использования (диаграмма прецедентов, use case diagram) — это диаграмма, на которой изображаются отношения между актерами и вариантами использования” [7].

На рисунке 7 представлена диаграмма вариантов использования.

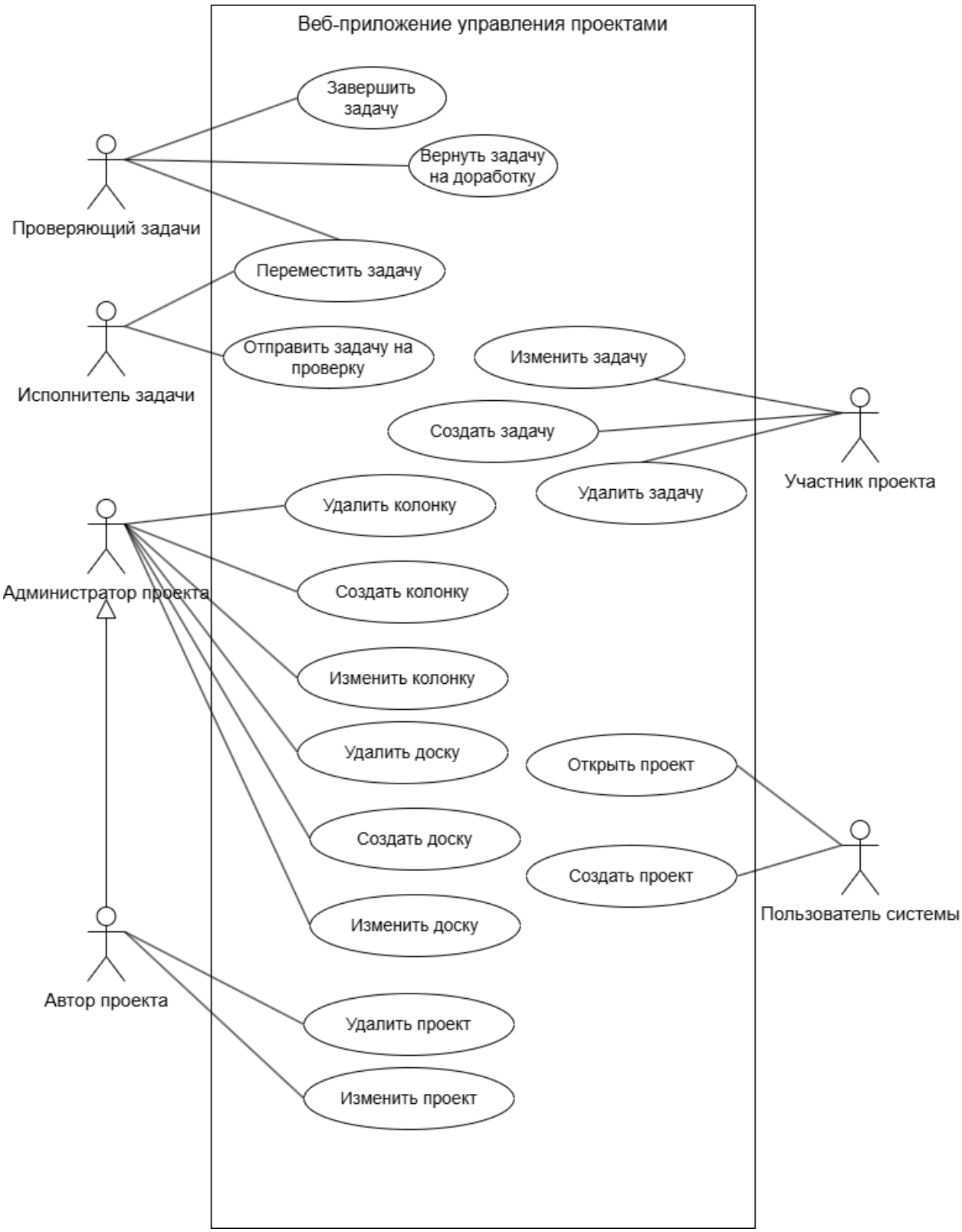


Рисунок 7 - Use-case диаграмма системы управления проектами.

Рассмотрим диаграмму на седьмом рисунке подробнее:

– Пользователь системы – любой авторизованный пользователь, который не открыл проект. Он может открыть проект, в котором он числится участником, либо создать собственный проект;
– Администратор проекта – пользователь, которого добавили в проект и наделили его правами администратора. Он может удалять, создавать и изменять доски, колонки и задачи;
– Автор проекта – пользователь, который создал проект. Связан с Администратором проекта связью обобщение (Generalization), так как выполняет те же прецеденты, что и администратор и участвует в прецедентах по удалению и изменению своих проектов;
– Участник проекта;
– Исполнитель задачи – пользователь, которого назначили исполнителем задачи. Он может отправить задачу на проверку и перемещать задачу по колонкам доски;
– Проверяющий задачи – пользователь, которого назначили проверяющим задачи. Он может отправить задачу на доработку, если задача выполнена неверно, может отметить задачу как выполненную, а также перемещать задачу по колонкам доски.

“Диаграмма классов - это самый простой, понятный и популярный тип диаграмм. Эти диаграммы предназначены для описания типов объектов и отношений между ними” [13]

На рисунке 8 продемонстрирована диаграмма классов веб-приложения по управлению проектами. На ней изображено пять ключевых классов – Project, Board, Column, Task, User - и одни интерфейс CRUD, который реализуют классы Project, Board, Column и Task. Объект класса User не зависим от остальных классов и может существовать даже при отсутствии экземпляров других классов.

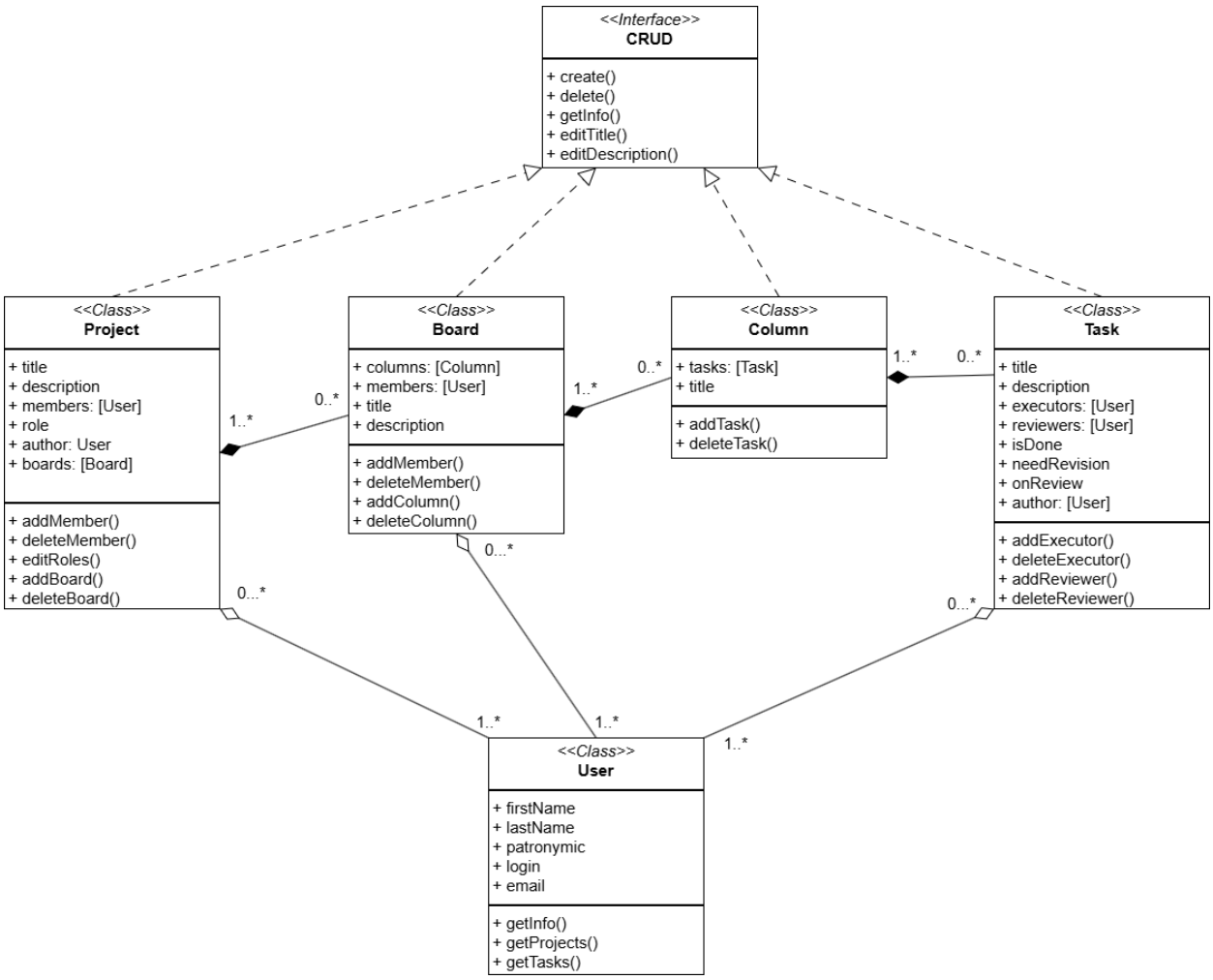


Рисунок 8 - Диаграмма классов

Рассмотрим подробнее классы, представленные на рисунке 8:

- User – класс, представляющий пользователя. Содержит ФИО, почту и логин. Имеет методы для получения информации о пользователе, доступных проектах и задачах;
- Task – класс, представляющий карточку задачи. Имеет название, описание, список исполнителей и проверяющих. Содержит методы для управления списками исполнителей и проверяющих;
- Column – класс, представляющий колонку с задачами на доске. Содержит поле списка задач и поле названия. Есть методы для добавления и удаления задачи из колонки;

– Board – класс, представляющий доску, с размещенными на ней колонками и задачами. Соответственно содержит поля для хранения информации о них и методы для их изменения;
– Project – класс, представляющий проект, с досками, колонками и задачами. В этом же классе содержится информация о пользователях и ролях. Имеет методы для управления ролями, участниками и досками.

Помимо классов, на диаграмме также представлен интерфейс CRUD (Create, Read, Update, Delete), содержащий общий набор функций, характерных для классов Project, Board, Column и Task.

“Диаграмма компонентов показывает компоненты, предоставляемые и требуемые интерфейсы, порты и отношения между ними” [2].

На рисунке 9 представлена диаграмма компонентов веб-сервиса по управлению проектами.

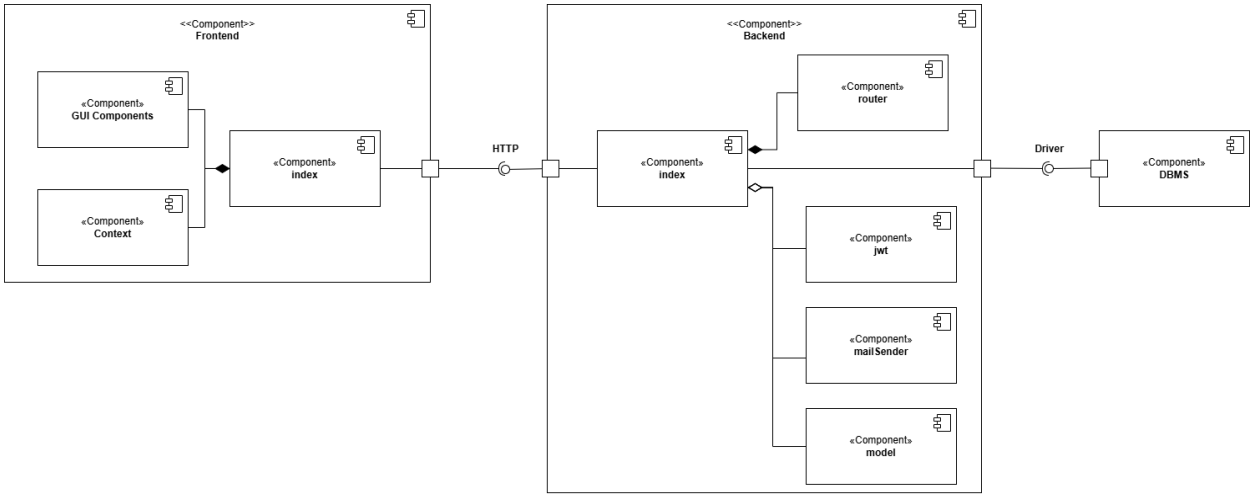


Рисунок 9 - Диаграмма компонентов

На рисунке 9 видно, что в веб-приложении имеется 3 ключевых компонентов:

– Frontend – компонент, представляющий клиентскую часть (то, что отображается у пользователя в браузере);

– Backend – компонент, представляющий серверную часть – скрытую от пользователя бизне-логику;
– DBMS – компонент, представляющий систему управления базой данных.

Можно заметить, в качестве интерфейса Frontend и Backend компонентов используется HTTP протокол, что обусловлено клиент-серверного архитектурой приложения.

Для связи компонентов СУБД и Backend используется специальный драйвер СУБД. В рамках реализации веб-системы таким драйвером является библиотека Mongoose.

“Логическая модель базы данных – представление о предметной области в виде данных и связей между ними, преобразованное для эффективной реализации в среде конкретной СУБД” [6, с. 26].

Рассмотрим модель данных для веб-приложения по управлению проектами. Так как в качестве СУБД используется MongoDB, то для хранения данных были созданы следующие коллекции:

– projects - хранение информации о проектах;
– boards - хранение информации о досках каждого из проектов;
– cards - хранение информации о карточках с задачами;
– columns - хранение информации о колонках;
– confirmationcodes - хранение кодов для подтверждения электронной почты;
– resetlinks - хранение уникальных одноразовых кодов для генерации ссылок для сброса пароля;
– users - хранение информации о пользователях.

Так как MongoDB нереляционная СУБД и в ней отсутствуют таблицы, то и такого понятия, как вторичный ключ, тоже отсутствует [15]. Вместо вторичного ключа используется ссылка на документ. На самом деле под ссылкой понимается уникально значение какого-то поля документа в

коллекции, с которой происходит связывание [3]. Рассмотрим ERD диаграмму базы данных (рисунок 10).

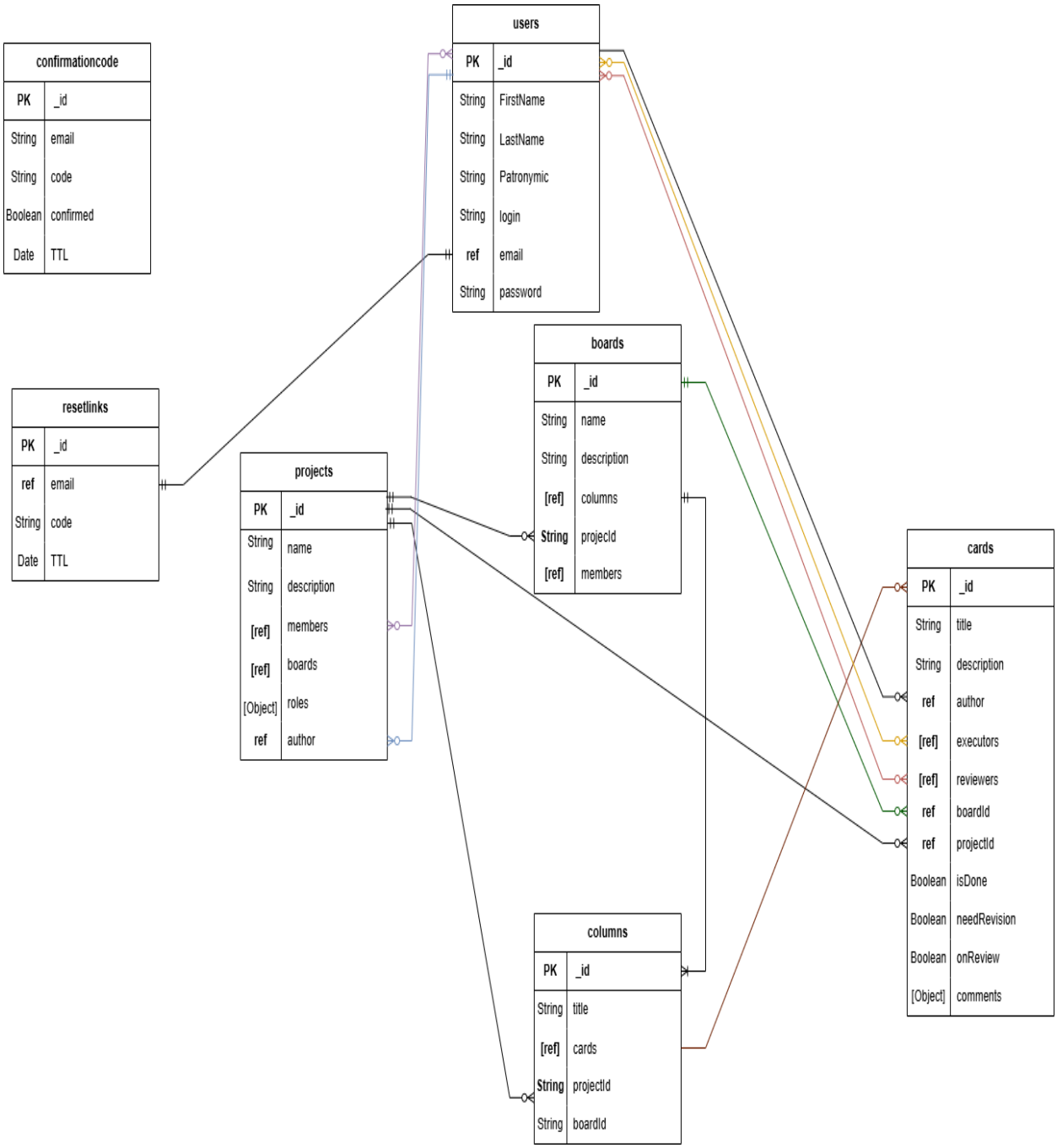


Рисунок 10 - Логическая модель данных

Как видно из рисунка 10, в БД существует несколько связей между коллекциями:

– один-ко-многим;

– один-к-одному;

– многие-ко-многим.

В таблице 2 подробно описаны связи между коллекциями базы данных.

Таблица 2 - Связи между коллекциями

Коллекция	Связанная коллекция	Тип связи и поля	Пояснение
users	cards	От одного обязательного к нескольким необязательным (_id -> author)	Один документ из коллекции users может быть связан с несколькими документами коллекции cards, либо вообще не иметь связей. Это объясняется тем, что при создании проектов, досок или колонок изначально карточек может и не быть. Их добавляют только после создания колонок.
users	cards	Многие ко многим обязательным (_id -> executors)	Множество документов из коллекции users может быть связано с множеством документов из коллекции cards – у карточки с задачей может быть множество исполнителей, в то же время у пользователя может быть множество задач (карточек)
users	cards	Многие ко многим обязательным (_id -> reviewers)	Множество документов из коллекции users может быть связано с множеством документов из коллекции cards – у карточки с задачей может быть множество проверяющих, в то же время пользователь может быть проверяющим множества задач (карточек)
users	projects	От одного обязательного к нескольким необязательным (_id -> author)	Один документ из коллекции users может быть связан с несколькими документами коллекции projects, либо вообще не иметь связей с документами этой коллекции. Это связано с тем, что, когда пользователь только зарегистрировался у него не будет проектов, в которых он состоит. Стоит отметить, что у каждого проектам может быть только один автор. При этом пользователь может создать сколько угодно проектов.

Продолжение таблицы 2

Коллекция	Связанная коллекция	Тип связи и поля	Пояснение
users	projects	От нескольких обязательных к нескольким необязательным (_id -> members)	Множество документ из коллекции users может быть связан с несколькими документами коллекции projects, либо вообще не иметь связей с документами этой коллекции. Это связано с тем, что, когда пользователь только зарегистрировался у него не будет проектов, в которых он состоит. При этом пользователь может быть участником сколь угодно проектов.
resetlinks	users	Один к одному (email -> email)	Документ из коллекции users может быть связан только с одним документом из коллекции resetlinks. Это объясняется тем, что при сбросе пароля для пользователя генерируется только одна ссылка и пока время существование не истечет, новая ссылка не может быть сгенерирована.
projects	boards	От одного обязательного к нескольким необязательным (_id -> projectId)	Один документ коллекции projects может быть связан с несколькими документами коллекции boards, либо вообще не иметь связей с документами этой коллекции. В только что созданном проекте нету досок. Однако пользователь может создать произвольное количество досок в одном проекте. При этом доска не может находится в нескольких проектах одновременно.
projects	cards	От одного обязательного к нескольким необязательным (_id -> projectId)	Один документ коллекции projects может быть связан с несколькими документами коллекции cards, либо вообще не иметь связей с документами этой коллекции. Аналогично с коллекцией boards, карточек с задачами может и не быть, но при этом их можно создать произвольно кол-во.
projects	columns	От одного обязательного к нескольким необязательным (_id -> projectId)	Один документ коллекции projects может быть связан с несколькими документами коллекции columns, либо вообще не иметь связей с документами этой коллекции. Аналогично с boards и cards, можно создать несколько columns, при этом в новом проекте может и не быть колонок вовсе.

Продолжение таблицы 2

Коллекция	Связанная коллекция	Тип связи и поля	Пояснение
boards	cards	От одного обязательного к нескольким необязательным (_id -> boardId)	Один документ коллекции boards может быть связан с несколькими документами коллекции cards, либо вообще не иметь связей с документами этой коллекции. Иными словами, при добавлении новой доски на ней нет задач, но в тоже время пользователь может задать произвольное количество карточке с задачами.
cards	columns	От одного обязательного к нескольким необязательным (_id -> cards)	Один документ коллекции cards может быть связан с несколькими документами коллекции columns, либо вообще не иметь связей с документами этой коллекции. Аналогично предыдущим связям, колонка может существовать и без карточке, однако сам пользователь может добавить несколько карточек в эту колонку.
boards	columns	От одного обязательного к нескольким необязательным (columns -> _id)	Один документ коллекции boards может быть связан с несколькими документами коллекции boards, либо вообще не иметь связей с документами этой коллекции. Пользователь добавляет доску без колонок, но потом может заполнить эту доску любым кол-вом колонок.
confirmationcode	-	-	Документы данной коллекции не ссылаются на документы других коллекций. Содержит код подтверждения почты. Документы этой коллекции создаются в процессе регистрации пользователя, до создания самого документа пользователя в коллекции users.

Подробное описание каждой коллекции приведено в следующей таблице 3.

Таблица 3 - Описание коллекций

Коллекция: users	
Поле	**Описание**
_id	Уникальный идентификатор документа (primary key)
FirstName	Имя пользователя
LastName	Фамилия пользователя
Patronymic	Отчество пользователя. Может отсутствовать.
login	Логин пользователя
email	Электронная почта пользователя
password	Пароль пользователя
Коллекция: projects	
Поле	**Описание**
_id	Уникальный идентификатор документа (primary key)
name	Название проекта
description	Описание проекта
members	Участники проекта
boards	Доски проекта
roles	Распределение ролей в проекте. Имеет вид {user_id: “role”}
author	Автор проекта. Его _id
Коллекция: boards	
Поле	**Описание**
_id	Уникальный идентификатор документа (primary key)
name	Название доски
description	Описание доски
columns	Колонки проекта
projectId	id проекта, к которой относится доска
members	Участники доски
Коллекция: columns	
Поле	**Описание**
_id	Уникальный идентификатор документа (primary key)
title	Название колонки
cards	Карточки с задачами проекта
projectId	id проекта, к которой относится колонка
Коллекция: cards	
Поле	**Описание**
_id	Уникальный идентификатор документа (primary key)
title	Название задачи
description	Описание задачи
author	Автор задачи. Его _id
executors	Исполнители задачи
reviewers	Проверяющие задачу
boardId	id доски, к которой относится карточка задачи
projectId	id проекта, к которой относится карточка задачи

Продолжение таблицы 3

Коллекция: cards	
Поле	**Описание**
isDone	Отметка о выполнении задачи
onReview	Задача выполнена исполнителем и отправлена на проверку
needRevision	Задача имеет ошибки или недочеты и отправлена на доработку исполнителям
Коллекция: resetlinks	
Поле	**Описание**
_id	Уникальный идентификатор документа (primary key)
email	Почта, на которую отправляется код для сброса пароля. Ссылается на документ в коллекции users
code	Уникальный код для сброса пароля.
TTL	Время жизни документа (Time To Live). По истечении времени документ будет уничтожен
Коллекция: confirmationcode	
Поле	**Описание**
_id	Уникальный идентификатор документа (primary key)
email	Почта, на которую высылается код подтверждения
code	Код подтверждения почты
confirmed	Является ли почта подтвержденной
TTL	Время жизни документа (Time To Live). По истечении времени документ будет уничтожен

2.1.2 Физическое проектирование

На данном этапе определяется, как будет работать программное обеспечение в производственной среде и описываются основные алгоритмы работы приложения с помощью схем или диаграмм. [10]

Физическое проектирование предполагает решение задач, связанных с адаптацией логической архитектуры к конкретным технологическим и инфраструктурным ограничениям.

“Диаграмма развертывания объединяет компоненты, артефакты и узлы для определения физической архитектуры системы” [1].

Диаграмма развертывания позволяет понять, на каком физическом оборудовании будет выполняться программное обеспечение и как модули программы будут взаимодействовать друг с другом. На рисунке 11 представлена диаграмма развертывания.

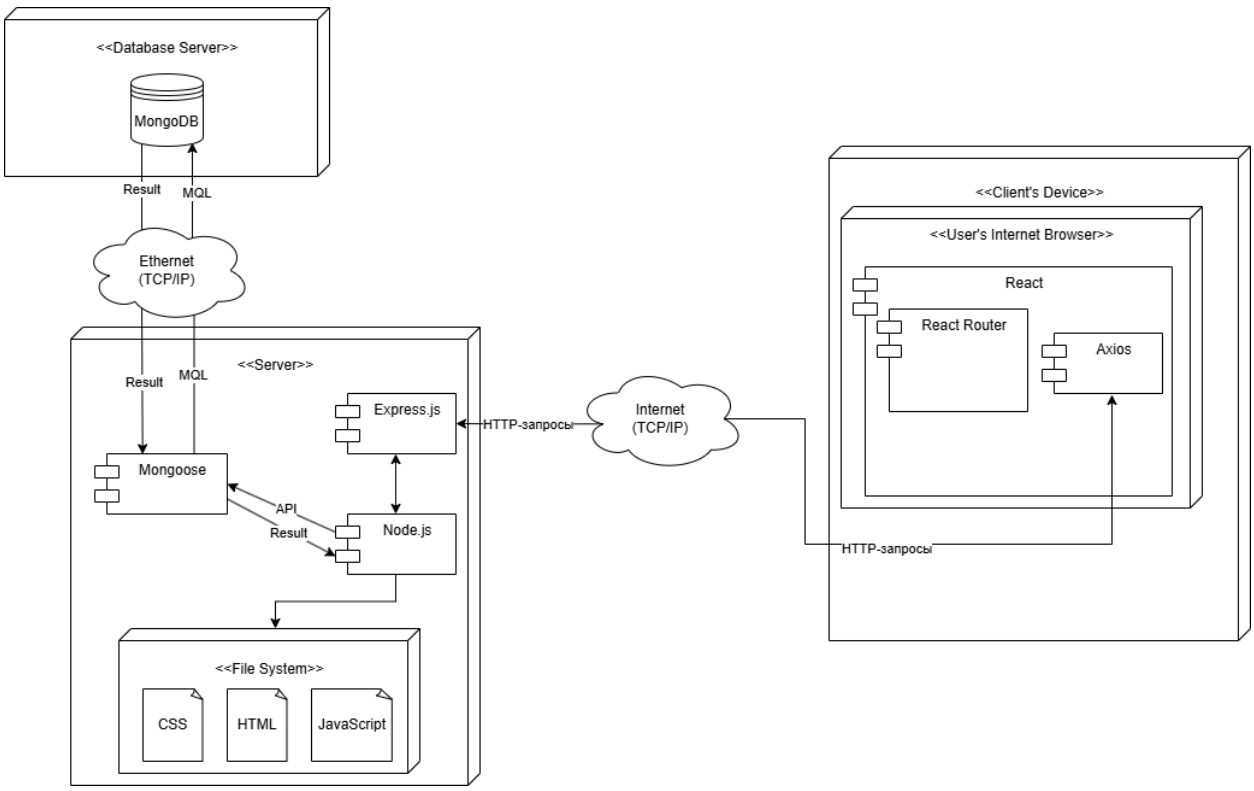


Рисунок 11 - Диаграмма развертывания

На рисунке 11 наглядно представлены следующие узлы, представляющие физическое оборудование и запущенное на них ПО:

- Database Server – сервер базы данных. На нем запущена СУБД MongoDB и хранятся данные веб-приложения.
- Server – сервер, на котором исполняется веб-приложение. Веб-приложение взаимодействует с внешним миром – БД и клиентскими устройствами - с помощью технологий Internet и Ethernet и стека протоколов TCP/IP. На сервере в его файловой системе также хранятся скомпилированные React-компоненты и SCSS файлы в виде HTML, CSS и JavaScript.
- Client’s Device – это устройство пользователя. Внутри него запущен еще один узел – приложение браузер, которое конечный пользователь использует для просмотра контента в сети интернет.

Внутри браузера выполняется React приложение со своими подмодулями.

2.2 Выбор технологий и инструментов для разработки

Используемые инструменты можно разделить на две крупные части – клиентскую и серверную. Рассмотрим каждую из этих частей более подробно.

Клиентская часть (Frontend). В качестве клиентской стороны выступает браузер с веб-страницами. Для разработки были использованы следующие средства и технологии:

- React.js;
- React Router;
- HTML;
- SCSS;
- JavaScript;
- axios - библиотека для работы с HTTP-запросами [5].

Серверная часть приложения (Backend). Не видна пользователям. Содержит основную бизнес-логику. Обработка поступающих HTTP запросов, хранение и обработка данных, работа с базой данных. Для backend были использованы следующие средства и технологии:

- MongoDB;
- Mongoose;
- Node.js;
- Express.js.
- jest
- nodemailer

Также во время разработки были использованы следующий программные средства:

- Visual Studio Code;

– MongoDB Compass;
– draw.io;
– Ramus.

Следует отметить, что React реализует технологию рендера на стороне клиента (CSR – Client Side Rendering) - сервер один раз при первом обращении к сайту отправляет конечному пользователю все компоненты, таблицы стилей и скрипты. Рендеринг самих веб-страниц происходит на стороне клиента прямо в браузере за счет выполнения скриптов. Сервер же в дальнейшем используется для получения клиентом данных или обновления данных в БД на основе информации, переданной пользователем [12]. Схематично CSR представлен на рисунках 12-13.

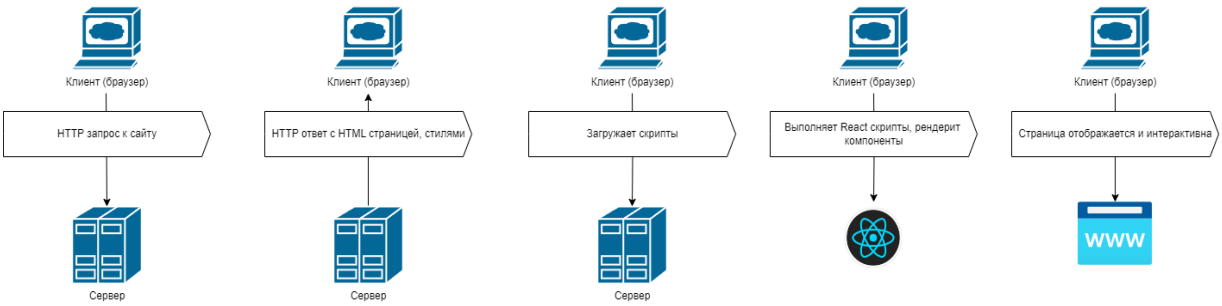


Рисунок 12 - CSR – загрузка при первом обращении к сайту

На диаграмме (рис. 12) видно, что при первом обращении к сайту, сервер отправляет все ресурсы, затем клиент скачивает необходимые скрипты и выполняет код React-фреймворка. В итоге мы получаем интерактивную веб-страницу.

При перемещении между страницами сайта, вместо запроса новой страницы, React отправляет серверу запрос на получение одних только данных, поскольку все HTML, CSS стили и скрипты были получены в полном объеме при первом обращении к веб-приложению. После получения данных, они отображаются в браузере (рисунок 13).

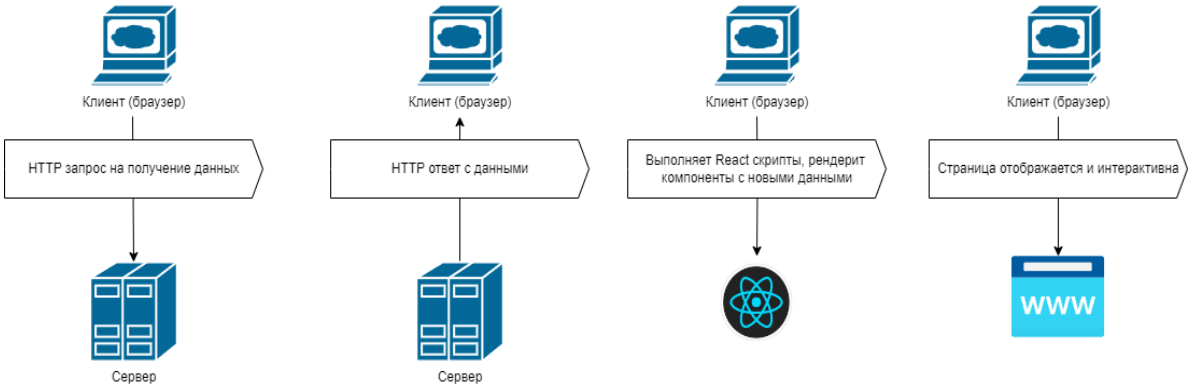


Рисунок 13 - CSR – обновление страницы новыми данными

Выбранные технологии позволяют разрабатывать современные, динамические и безопасные веб-приложения.

2.3 Реализация функциональных требований

2.3.1 Серверная часть

В данном подразделе приведено описание URL-маршрутов и алгоритмов веб-приложения по управлению проектами.

Все URL-маршруты логически объединены под общими префиксами маршрутов:

– / - общие маршруты
– /project/ - маршруты для работы с проектами
– /board/ - маршруты для работы с досками
– /column/ - маршруты для работы с колонками
– /card/ - маршруты для работы с карточками задач
– /user/ - маршруты для работы с пользователями

Рассмотрим подробнее URL-маршруты в таблице 4.

Таблица 4 - URL-маршруты веб-приложения

Маршрут	Тип	Пользователь авторизован	Только администратор проекта	Только автор проекта
/user/create	POST	Нет	Нет	Нет
/user/get	GET	Да	Нет	Нет
/user/login	POST	Нет	Нет	Нет
/user/sendConfirmationCode	POST	Нет	Нет	Нет
/user/checkConfirmationCode	GET	Нет	Нет	Нет
/user/resetPsswrd	POST	Нет	Нет	Нет
/user/sendResetLink	POST	Нет	Нет	Нет
/project/create	POST	Да	Нет	Нет
/project/update	PUT	Да	Нет	Да
/project/delete	DELETE	Да	Нет	Да
/project/getOne	GET	Да	Нет	Нет
/project/getAll	GET	Да	Нет	Нет
/board/add	POST	Да	Да	Да
/board/update	PUT	Да	Да	Да
/board/rename	PUT	Да	Да	Да
/board/delete	DELETE	Да	Да	Да
/column/add	POST	Да	Да	Да
/column/delete	DELETE	Да	Да	Да
/column/rename	PUT	Да	Да	Да
/card/add	POST	Да	Нет	Нет
/card/get	GET	Да	Нет	Нет
/card/delete	DELETE	Да	Нет	Нет
/card/archive	PUT	Да	Нет	Нет
/card/update	PUT	Да	Нет	Нет
/card/move	PUT	Да	Нет	Нет
/card/changeStatus	PUT	Да	Нет	Нет

Рассмотрим некоторые маршруты подробнее, приведя блок-схему алгоритма-обработчика пути и описание маршрута.

Маршруты группы /user/ позволяют оперировать данными пользователей. Исходный код блок-схем реализован в модуле userRouter и приведен в Приложении Б.

Маршрут /user/create нужен для регистрации новых пользователей в системе. Промежуточное ПО, обрабатывающее этот маршрут, создает документ нового пользователя и сохраняет его в коллекции users. Сохранение пользователя в БД возможна только после подтверждения электронной почты

и завершения проверок данных на корректность. Рассмотрим алгоритм регистрации пользователя на рисунке 14.

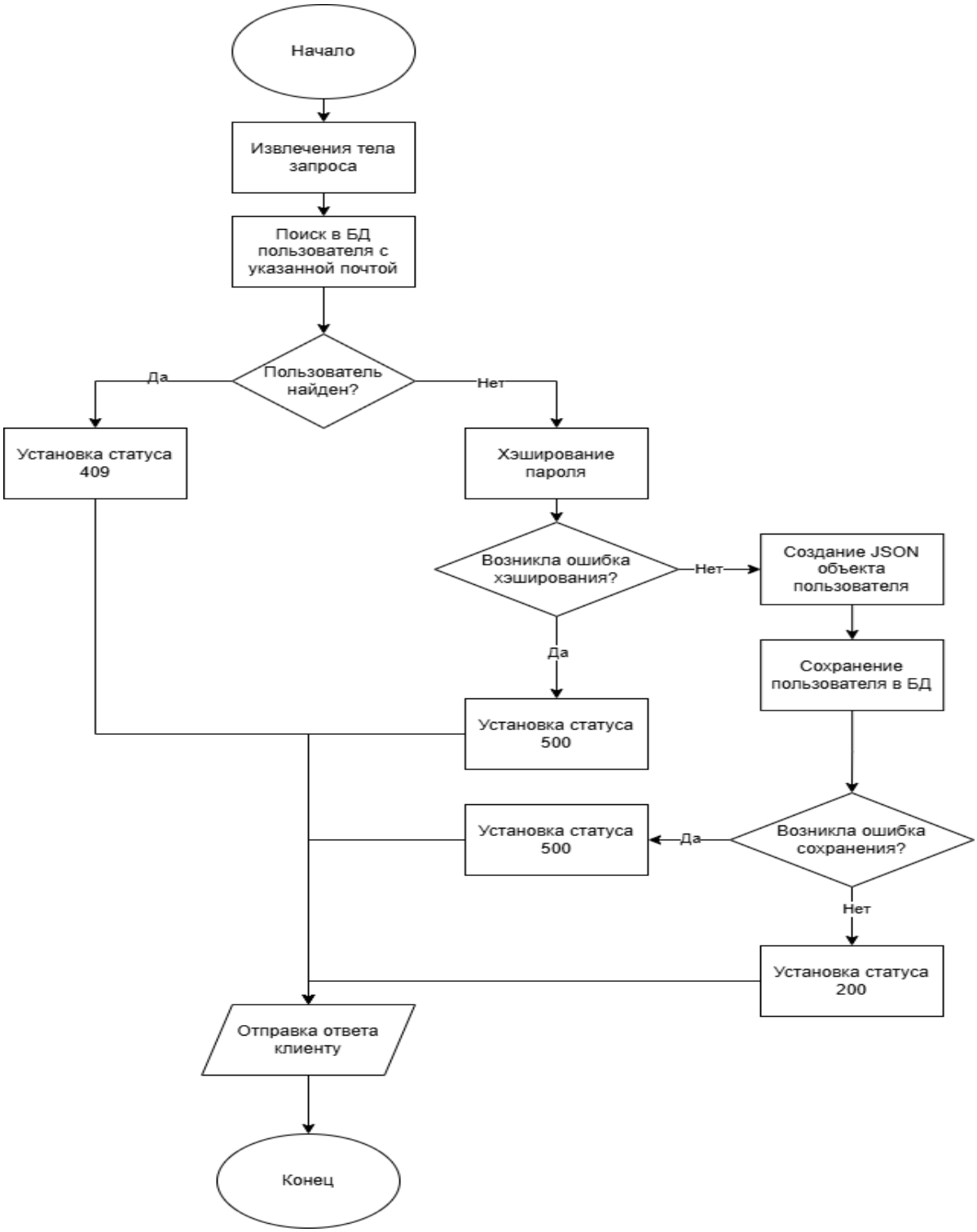


Рисунок 14 - Создание нового пользователя

Маршрут /user/get необходим для получения данных о конкретном пользователе системы. Блок-схема приведена на рисунке 15.

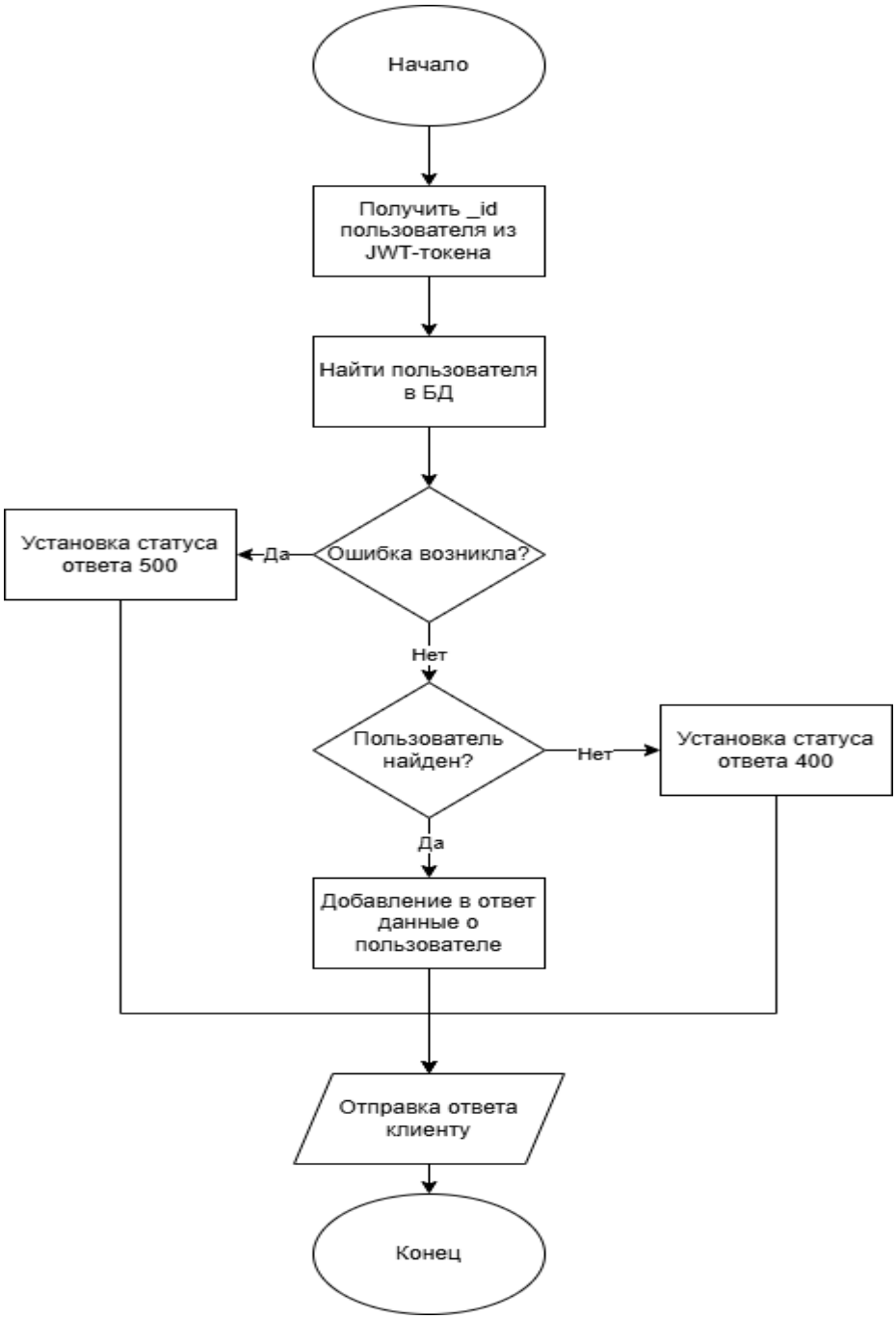


Рисунок 15 - Блок схема алгоритма получения пользователя

Запрос /user/login необходим для процедуры авторизации пользователя в системе. Блок-схема алгоритма приведена на рисунке 16.

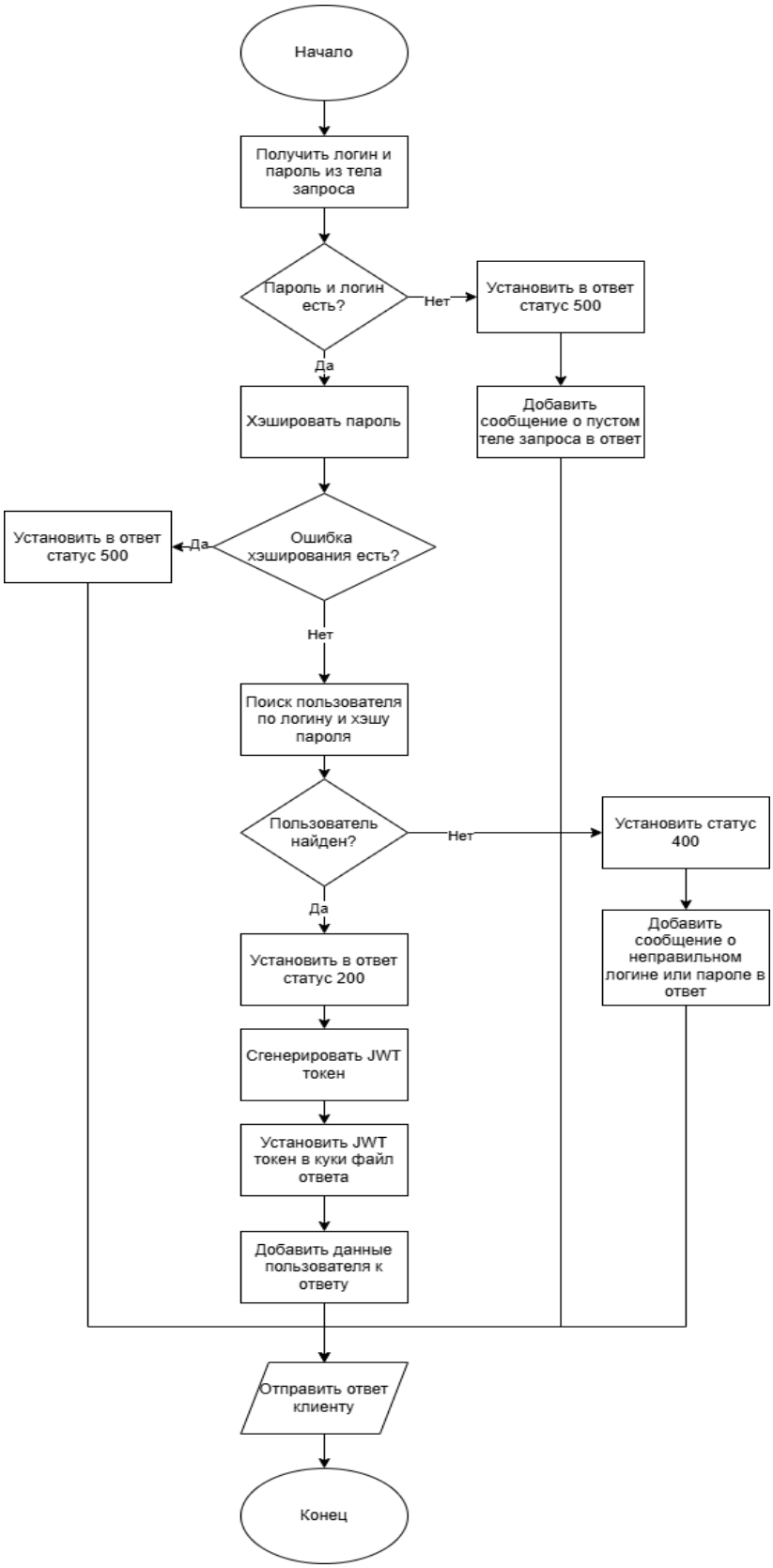


Рисунок 16 - Блок-схема алгоритма авторизации

Маршруты /project/ позволяют пользователям в системе работать с проектами – создавать, переименовывать, изменять и удалять. Обработкой запросов по данным маршрутам занимается модуль projectRouter. В Приложении В приведен исходный код модуля, блок-схемы обработчиков которого показаны на рисунках 17 – 19 и в Приложении Г.

Перейдем к блок-схеме алгоритма создания проекта, который обрабатывает путь /project/create (рисунок 17).

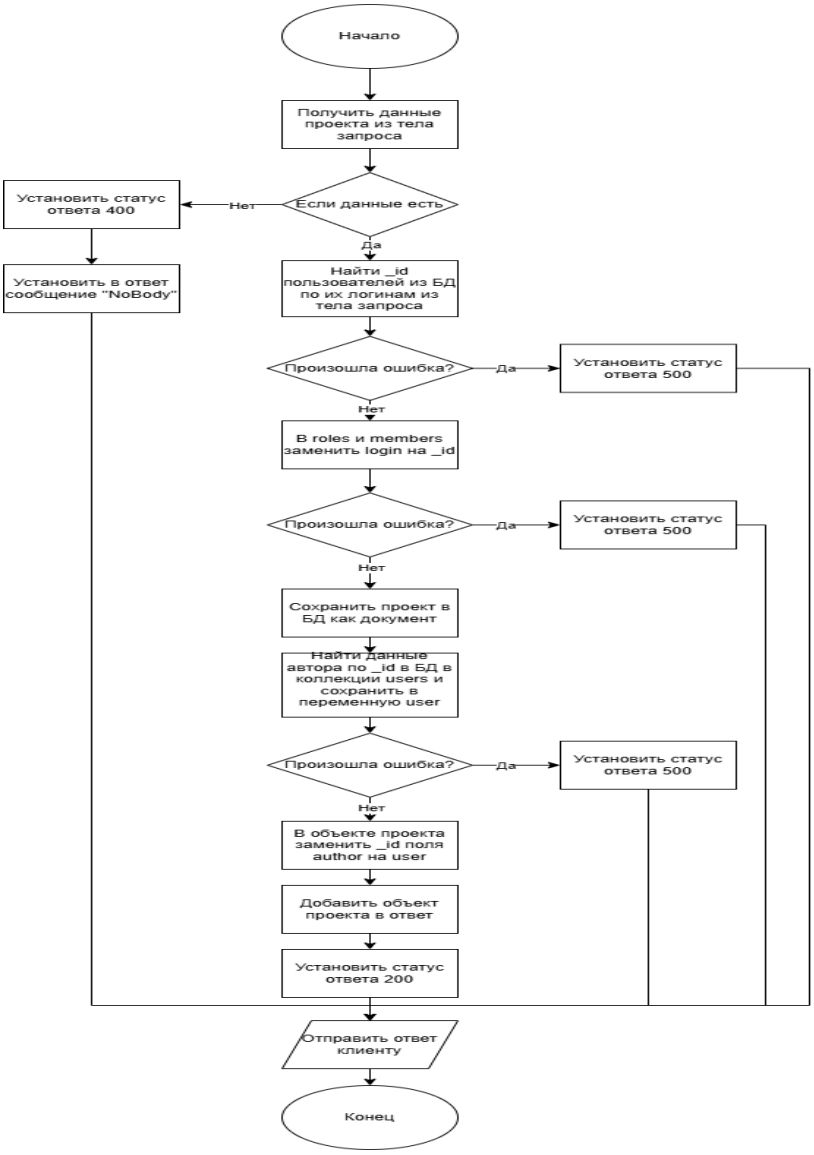


Рисунок 17 - Блок-схема алгоритма создания проекта

Перейдем к следующему маршруту - /project/update. Как следует из названия, запрос по данному маршруту позволяет обновить проект, например, сменить его название, изменить описание, добавить новых участников проекта или поменять роли уже существующих участников проекта. В Приложении Г.1 продемонстрирована блок-схема алгоритма.

На следующей блок схеме (Приложение Г.2) представлен алгоритм обработки запроса /project/delete. Данный алгоритм позволяет удалять проект.

Маршрут /project/getOne позволяет получить данные конкретного проекта. Этот запрос поступает с клиента, когда пользователь открывает проект и тем самым хочет получить данные об этом проекте. На рисунке 18 представлена блок-схема алгоритма, который обрабатывает запрос /project/getOne.

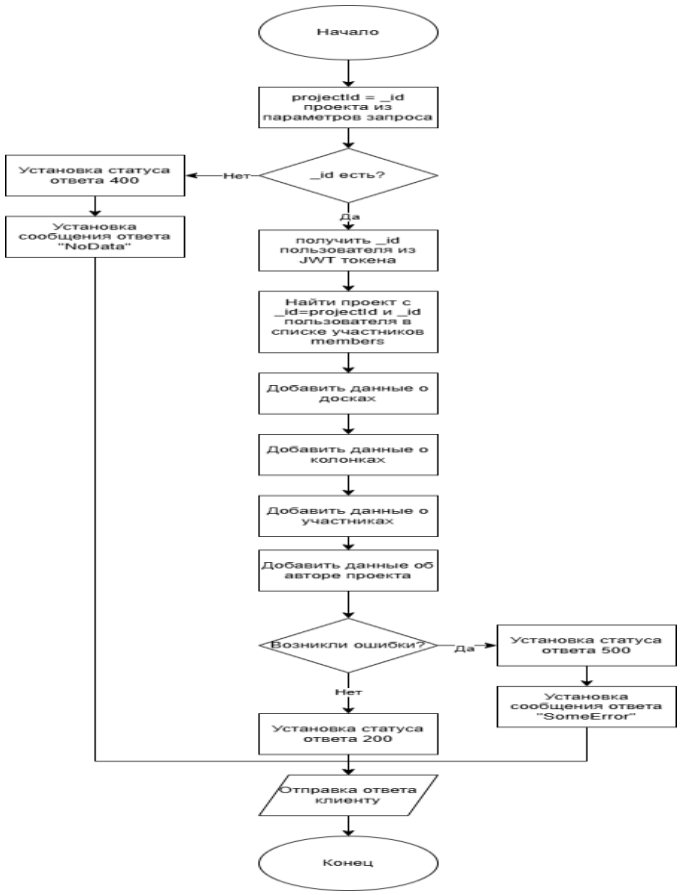


Рисунок 18 - Блок-схема алгоритма-обработчика запроса /project/geOne

На рисунке 18 показан исходный код алгоритма получения данных проекта. Для добавления данных из связанных коллекций используется метод populate.

Рассмотрим блок-схему обработчика пути /project/getAll. Данный запрос позволяет получить список всех проектов, которые создал пользователь или в которых числится участником или администратором. Блок-схема представлена на рисунке 19.

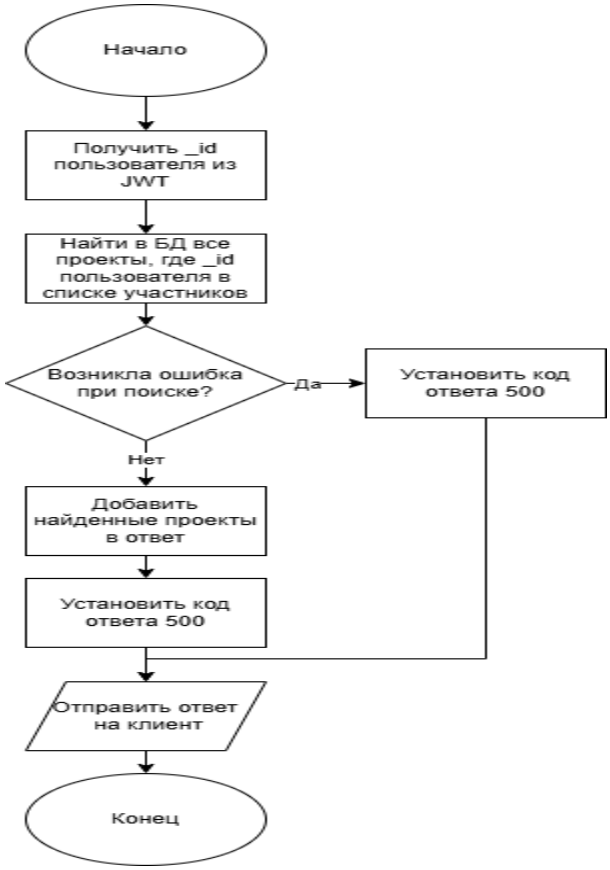


Рисунок 19 - Блок-схема алгоритма получения проектов пользователя

Перейдем к рассмотрению маршрутов группы /board/. Данные маршруты используются для управления досками (рабочими пространствами) проекта. Обработкой маршрутов /board/ занимается модуль boardRouter. Блок-схемы алгоритмов-обработчиков маршрутов представлены на рисунках 20-22. Фрагмент исходного код модуля представлен в Приложении Д.

Начнем с обработчика пути /board/add. Данный алгоритм позволяет добавлять новые доски (рабочие пространства) в проект (рисунок 20).

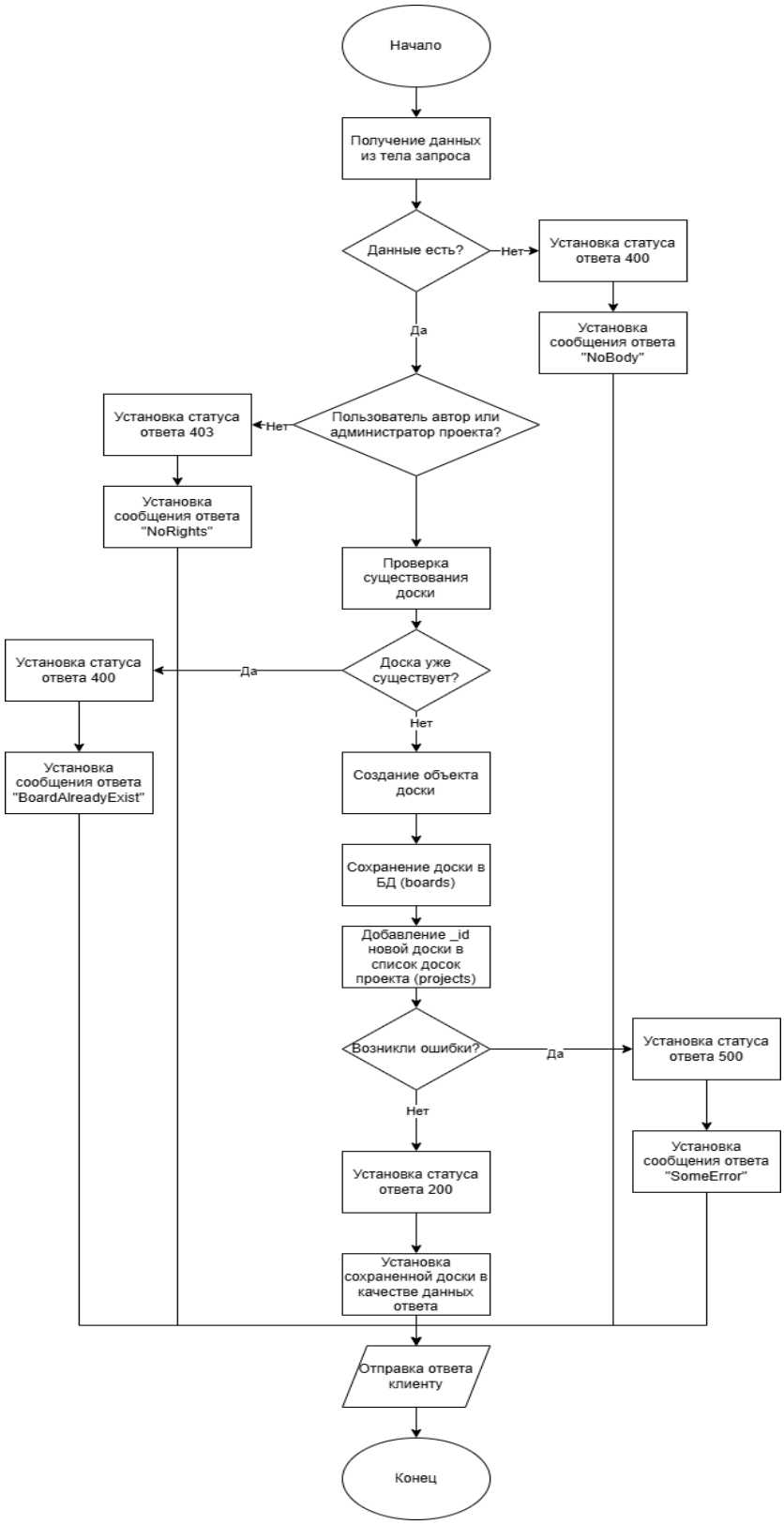


Рисунок 20 - Алгоритм по добавлению новой доски к проекту

/board/update – маршрут для обновления доски. Блок-схема обработчика данного маршрута приведена на рисунке 21.

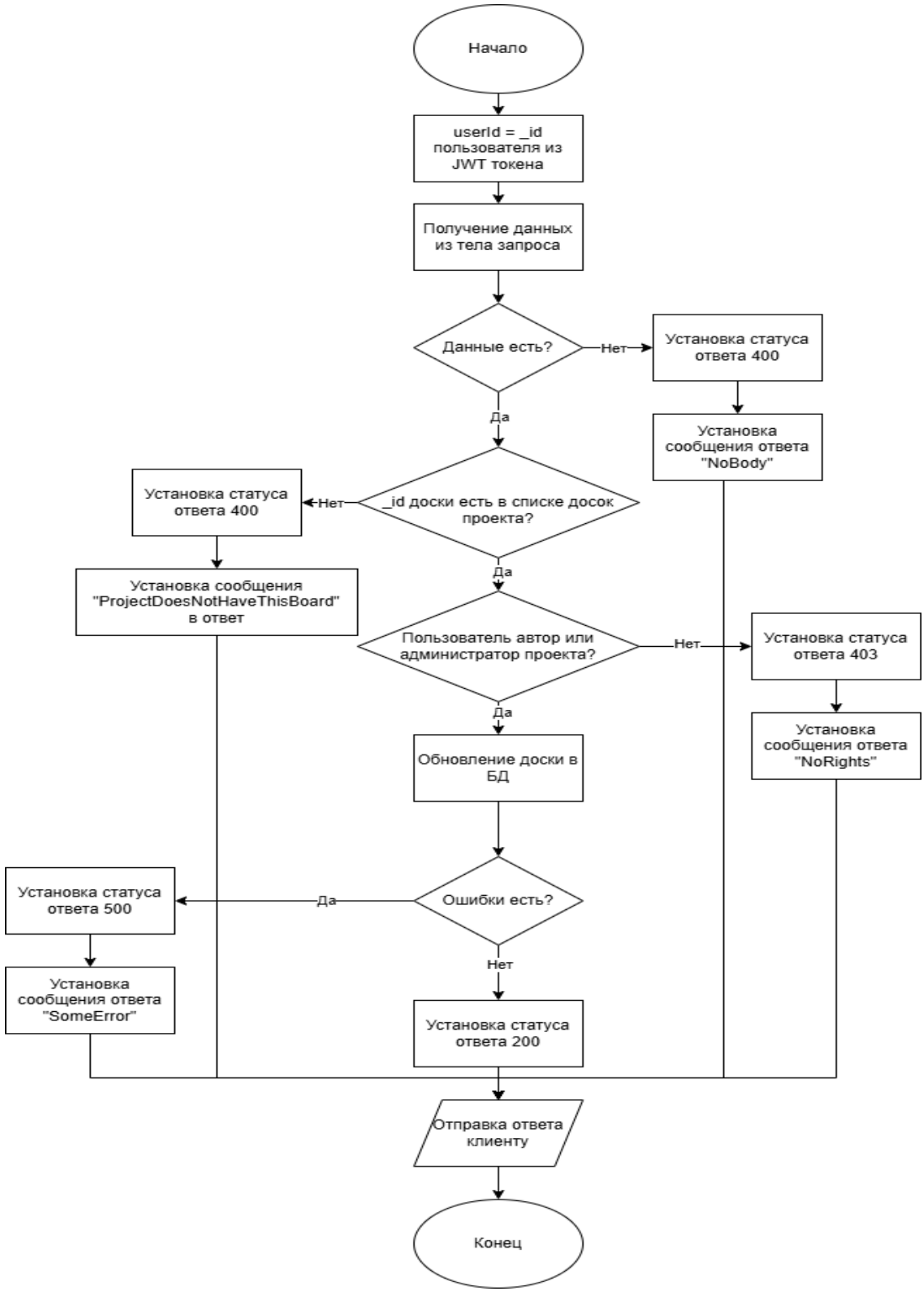


Рисунок 21 - Блок-схема обработчика пути /board/update

В качестве следующего маршрута возьмем /board/delete. Запросы на этот адрес позволяет удалять доску вместе со всем ее содержимым. Блок-схема приведена на рисунке 22.

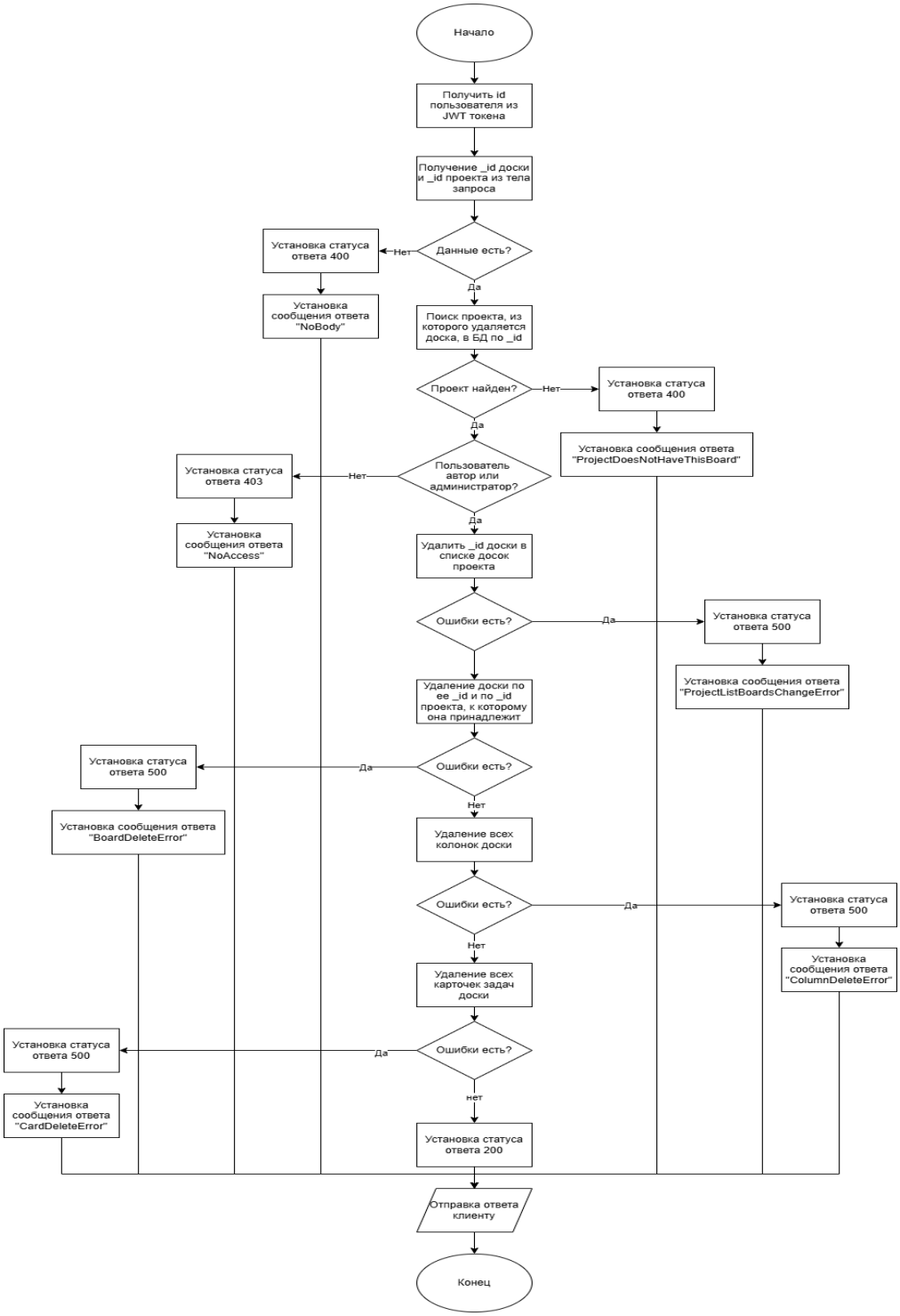


Рисунок 22 - Алгоритм удаления доски из проекта

Перейдем к рассмотрению маршрутов для управления карточками задач. Блок-схемы алгоритмов приведены на рисунках 23-26. Обрабатывает данные маршруты компонент cardRouter. Исходный код модуля по обработке запросов приведен в Приложении Е.

Маршрут /card/add. Данный маршрут позволяет добавлять новые карточки задач. Рассмотрим блок-схему обработчика маршрута на рисунке 23.

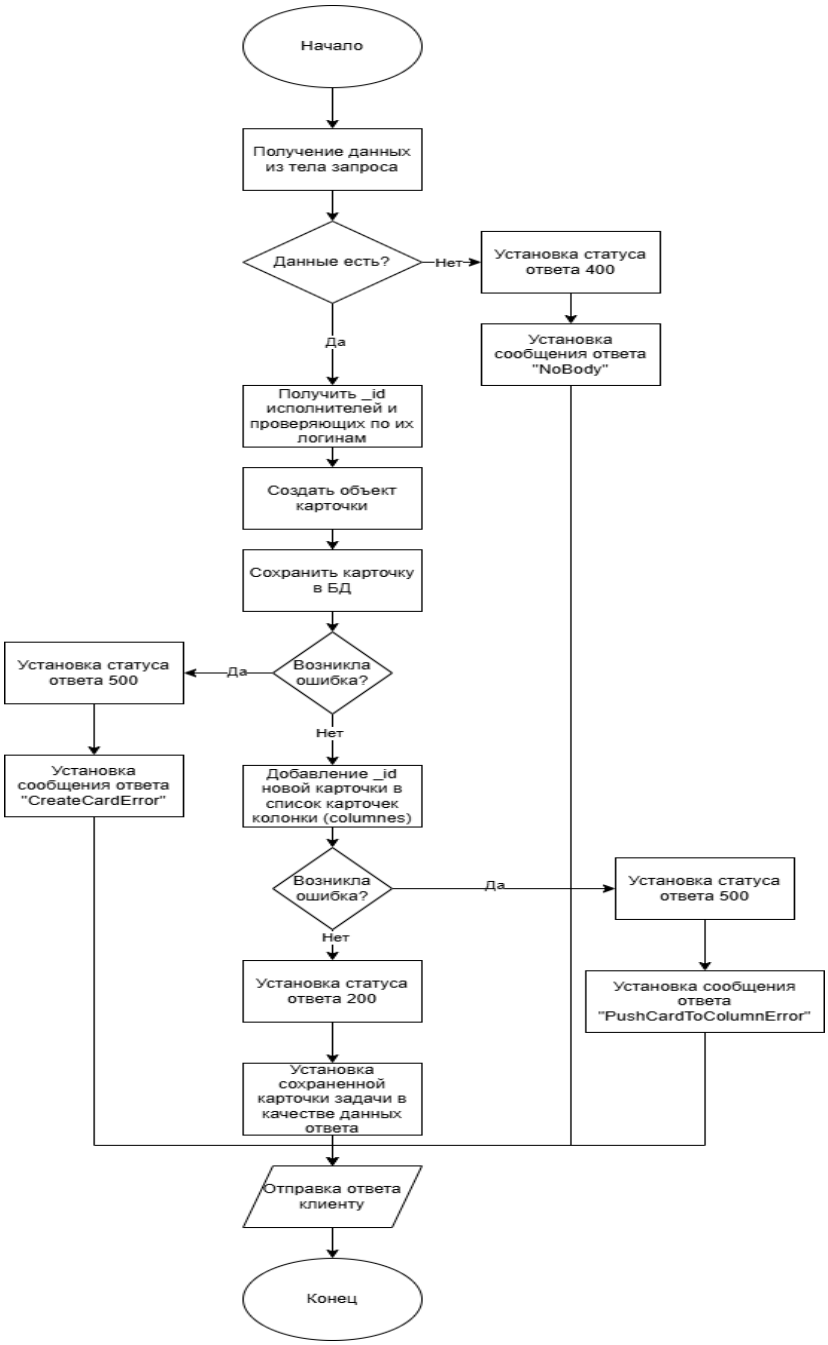


Рисунок 23 - Блок-схема алгоритма добавления новой карточки задач.

На следующей иллюстрации – рисунке 24 – показана алгоритм обновления карточки задачи.

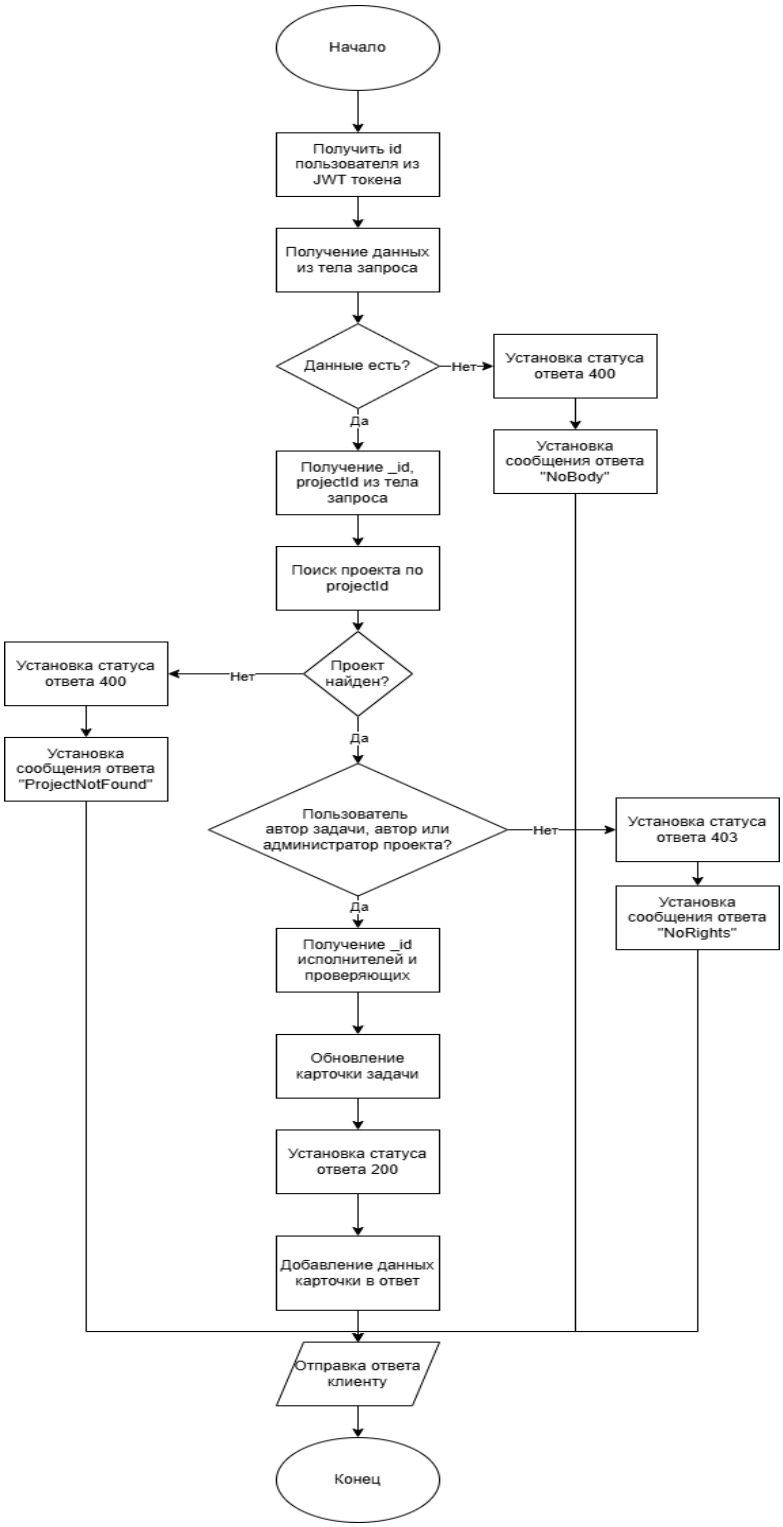


Рисунок 24 - Блок схема алгоритма обновления карточки задачи

Перейдем к рассмотрению алгоритма обработки следующего запроса - /card/changeStatus. Данный запрос позволяет изменять статус задачи. На рисунке 25 представлена схема алгоритма.

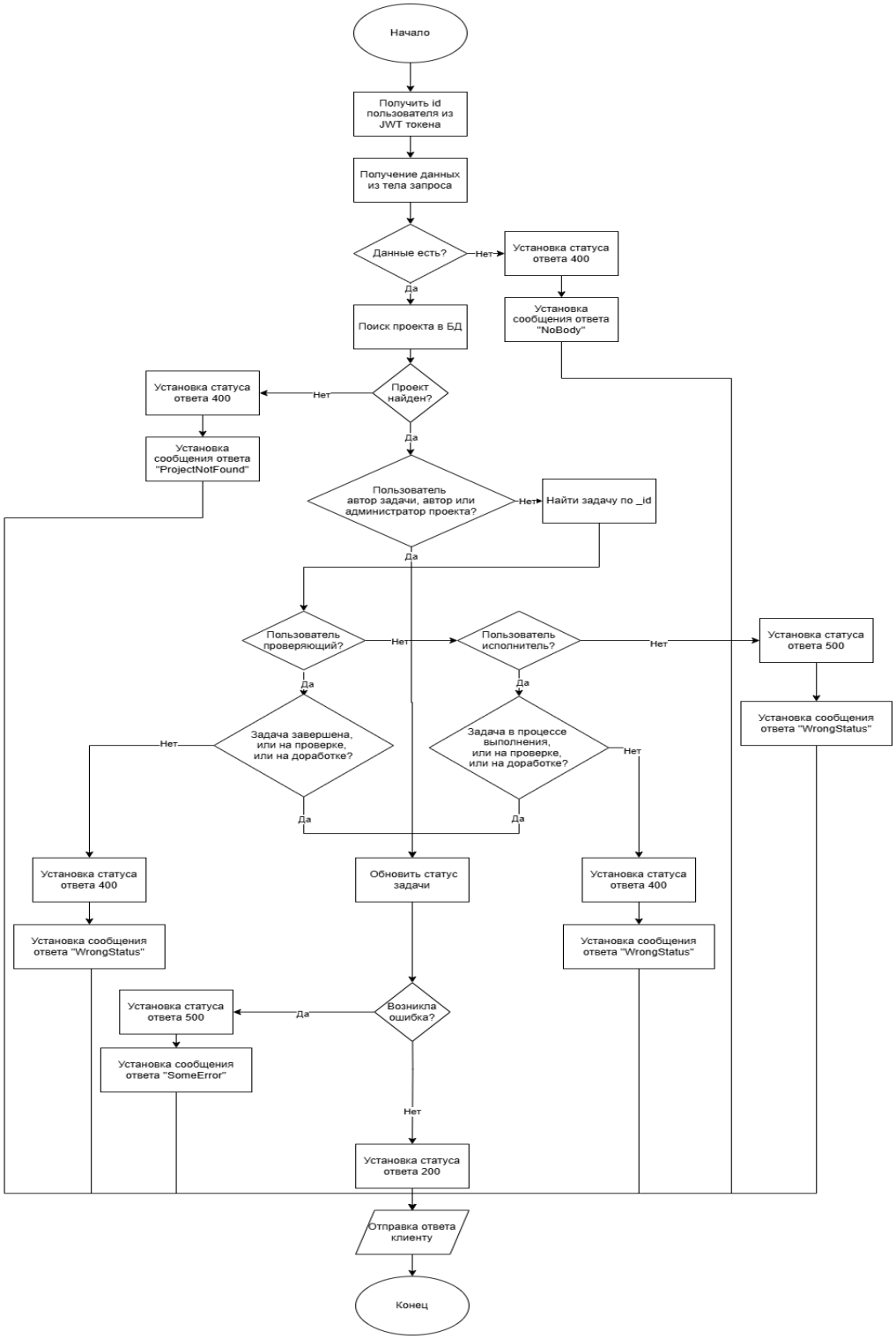


Рисунок 25 - Схема обработки смены статуса задачи

Следующий маршрут - /card/delete – предназначен для полного удаления задачи из проекта. Блок-схема алгоритма продемонстрирована на рисунке 26.

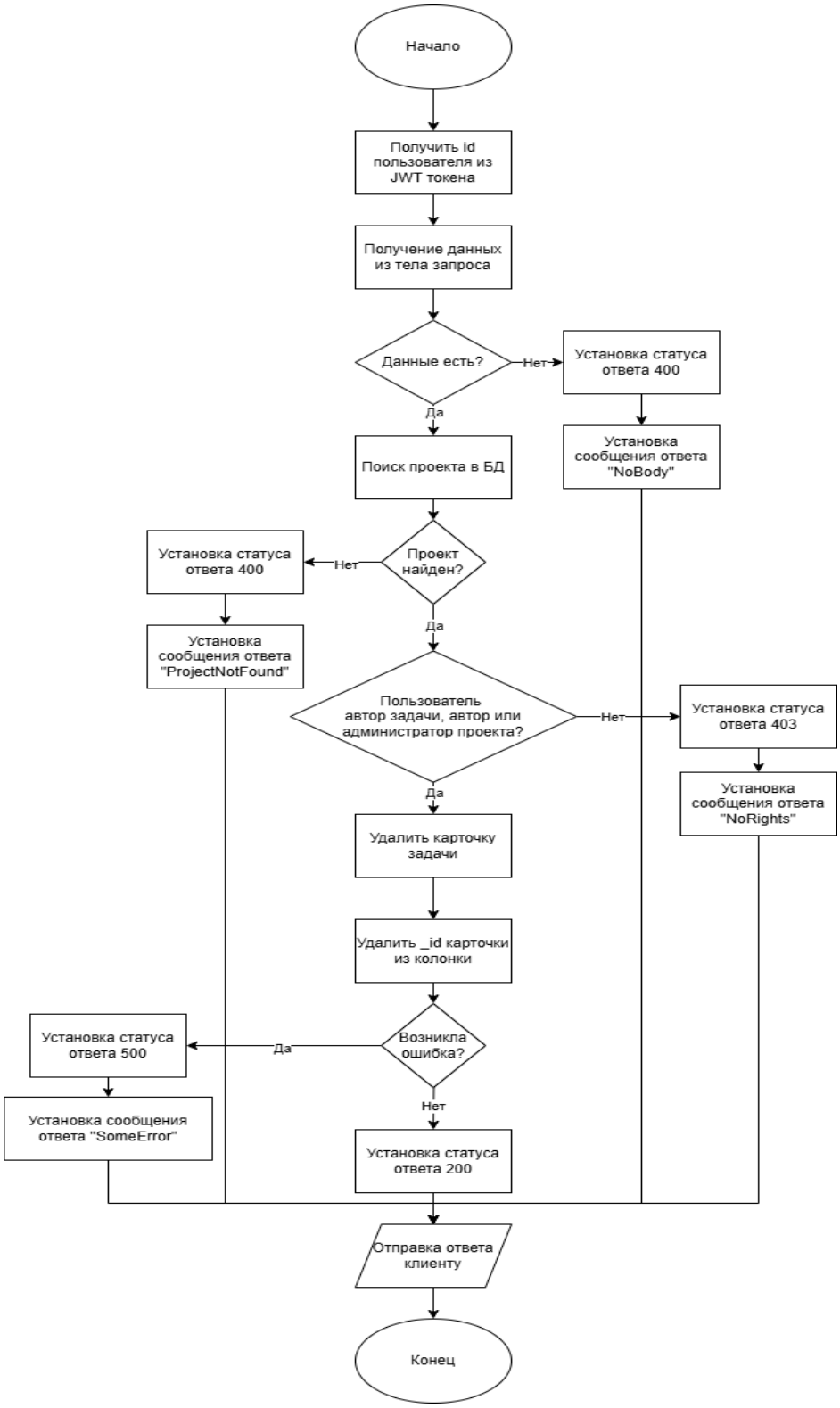


Рисунок 26 - Блок-схема алгоритма удаления задачи из проекта

Суммируя все вышесказанное в данном пункте, приведенные блок-схемы позволяют в полном объеме понять логику работы функций, не вникая в сам программный код.

2.3.2 Клиентская часть

На клиентской части веб-приложения применяется маршрутизация за счет библиотеки React Router. Рассмотрим, как реализуется маршрутизация с помощью расширения на рисунке 27.

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Main/>}/>
    <Route path="/registration" element={<Registration/>}/>
    <Route path="/auth" element={<Auth/>}/>
    <Route path="/inputEmailForResetPswrd" element={<InputEmailForResetPswrd/>}/>
    <Route path="/resetPassword" element={<ResetPassword/>}/>
  </Routes>
</BrowserRouter>
```

Рисунок 27 - Маршрутизация с помощью React Router

На рисунке 27 мы видим, что используются компоненты <BrowserRouter/>, <Routes/> и <Route/>. Компоненты <BrowserRouter/> и <Routes/> обеспечивают работу маршрутизации на стороне клиента, <Route> компонент связывает путь с конкретным компонентом, представляющим веб-страницу. [17]

Интерфейс реализован в виде пяти веб страниц. Описание каждой веб-страницы приведено в таблице 5.

Таблица 5 - Описание страниц

Страница	Адрес	Описание
Главная страница	/	Главная страница. Управление проектами. На ней можно выбрать или создать проект, добавлять доски, колонки и задачи к проектам.
Авторизации	/auth	Страница авторизации для зарегистрированных пользователей. Состоит из двух текстовых полей ввода и кнопки.

Продолжение таблицы 5

Страница	Адрес	Описание
Регистрации	/registration	Страница с формой ввода данных для регистрации. Позволяет указать ФИО, логин, почту и пароль. Для регистрации также требуется подтвердить электронную почту,.
Ввода почты для сброса пароля	/inputEmailForResetPswrd	Страница для сброса пароля. Позволяет ввести почту, на которую будет выслана ссылка для сброса пароля.
Ввода нового пароля	/resetPassword?code	Страница для ввода нового пароля. Ссылка с кодом, указанном в параметре запроса, высылается на почту, указанную пользователем на странице по пути /sendResetPasswordLink

Рассмотрим внешний вид веб-страниц. Когда пользователь переходит на главную страницу веб-приложения, появляется всплывающее окно, в котором отображаются доступные проекты и действия с ними. Пользователь может открыть проект, создать проект, удалить или изменить проект. При этом удалить или отредактировать проект может только его автор. Рассмотрим главную страницу на рисунке 28.

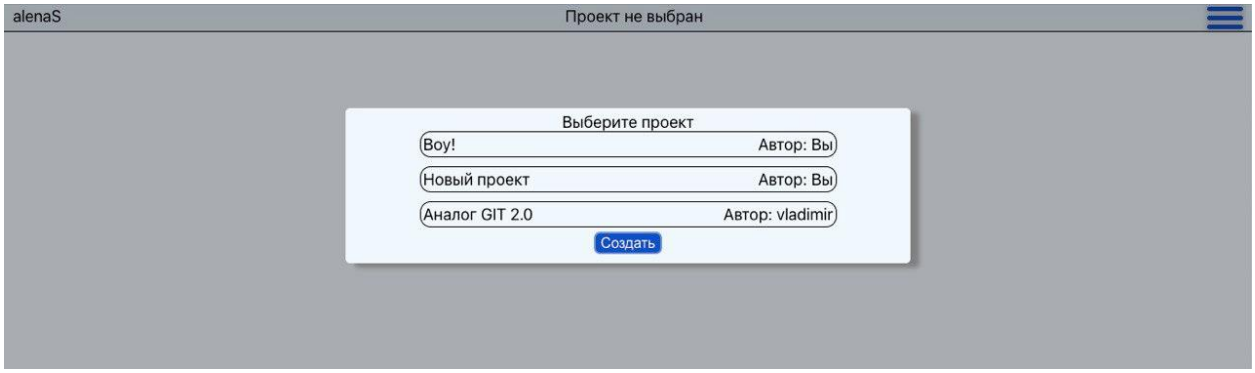


Рисунок 28. Главная страница

На 28 рисунке представлена страница по корневому адресу. На скриншоте видно, что за всплывающем окном находится имя авторизованного пользователя и некоторая рабочая область, занимающая большую часть

экрана – на ней будет располагаться доска с задачами. Рассмотрим, как выглядит окно создания нового проекта на рисунке 29.

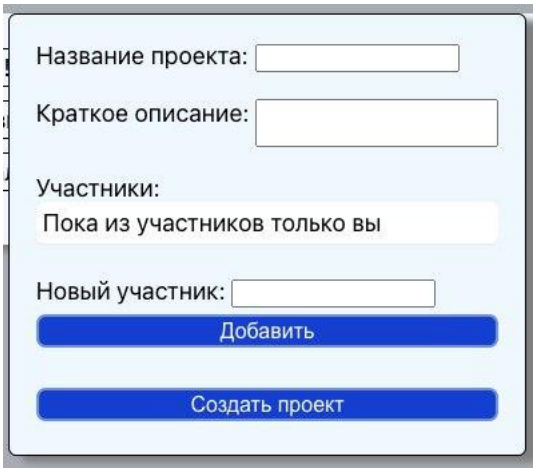


Рисунок 29. Окно создания нового проекта

Рассмотрим на рисунке 30 как будет выглядеть заполненное окно создания нового проекта.

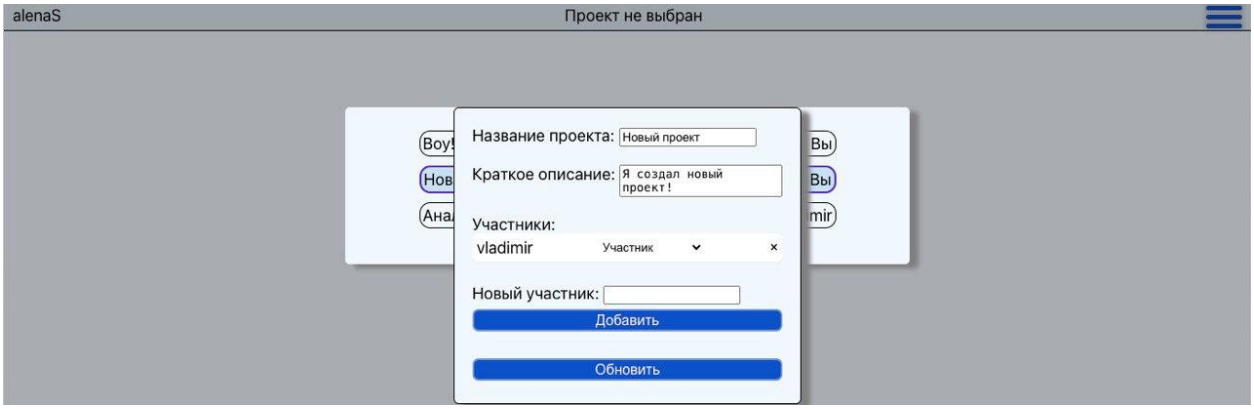


Рисунок 30. Заполненное окно создания нового проекта

На рисунке 30 можно увидеть, что при создании проекта можно определять роли участникам. Например, на картинке добавлен участник с логином “vladimir”, которому назначена роль “Участник”.

Окно редактирования проекта выглядит аналогично окну создания проекта. На рисунке 31 продемонстрирован снимок экрана с окном редактирования проекта.

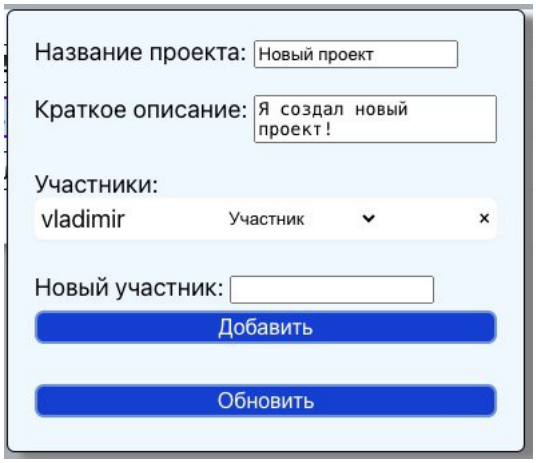


Рисунок 31. Редактирование проекта

После создания или открытия проекта, пользователь попадает в рабочее пространство проекта. Сразу открывается боковое меню – в нем содержится список доступных досок и функционал для их изменения (см. рисунок 32)

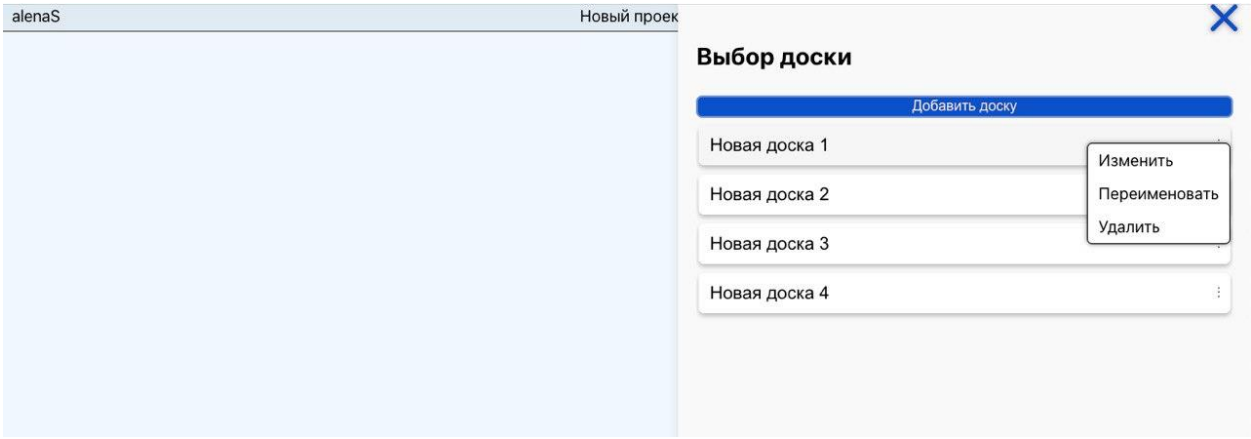


Рисунок 32 - Проект и список досок

На рисунке 32 продемонстрировано меню выбора доски. Помимо самого списка досок, есть также кнопка для их создания, а также выпадающее меню,

с помощью которого можно отредактировать название доски или удалить ее. Выпадающее меню можно открыть по нажатию на “три точки” справа от названия доски.

Доску можно редактировать – менять ее название, описание и список участников. При этом участником доски может быть только пользователь, являющийся участником проекта. Редактировать доску могут только администраторы и автор проекта. На рисунке 33 представлено всплывающее окно редактирования доски.

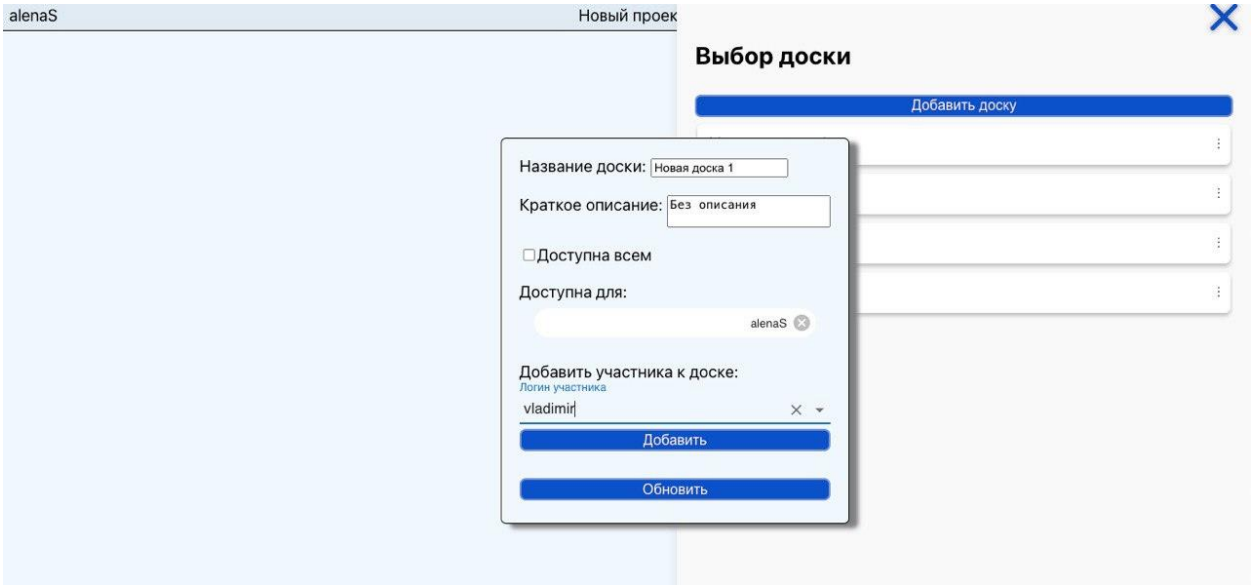


Рисунок 33 - Редактирование доски

После выбора доски загружаются колонки и доступные задачи. На рисунке 34, изображена пустая доска.

Рисунок 34 - Список колонок без задач и колонок

На рисунке 35 продемонстрирована доска с несколькими колонками.

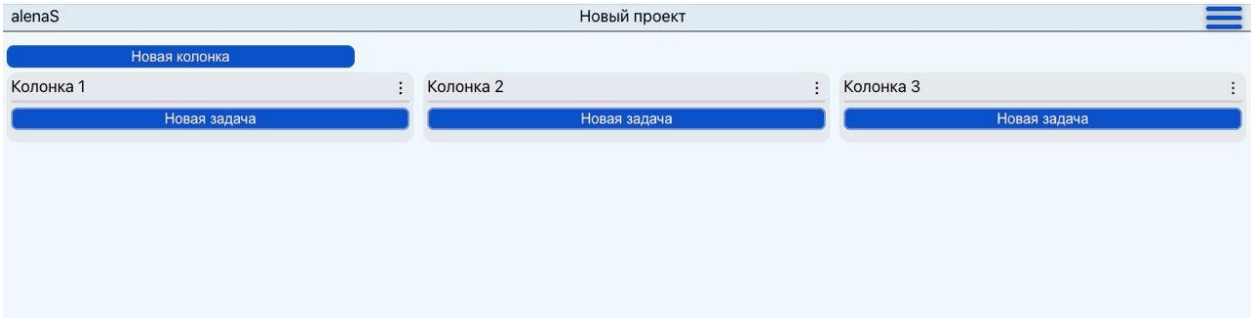


Рисунок 35 - Доска с пустыми колонками

После перехода к доске и загрузки колонок, можно создать карточки с задачами. Для этого в соответствующей колонке требуется нажать кнопку ”Новая задача”. После этого появится всплывающее окно создания новой карточки задачи. Оно показано на рисунке 36.

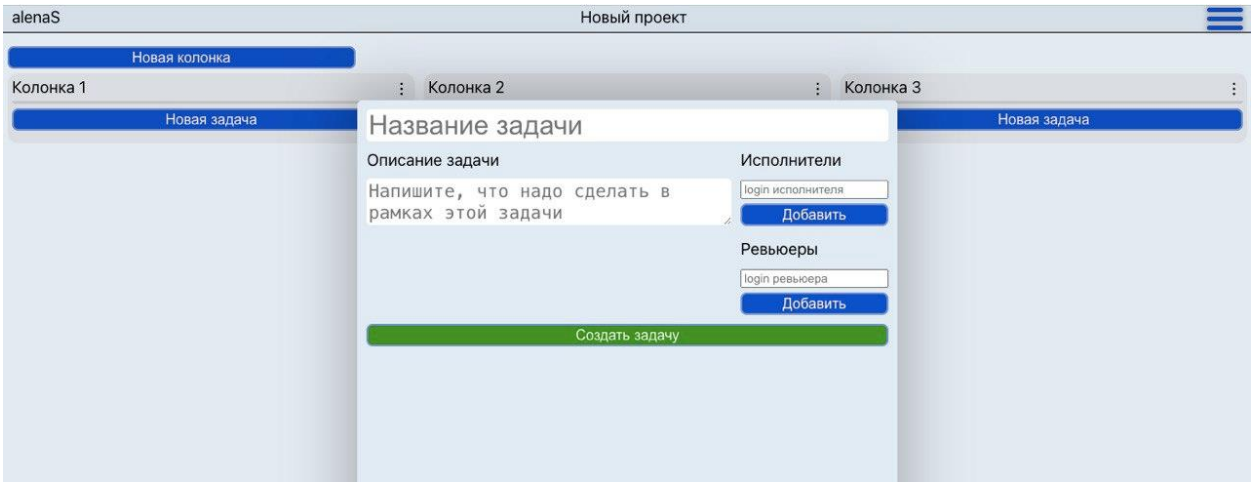


Рисунок 36 - Окно создания задачи

На рисунке 36 мы видим окно для создания задачи. В нем пользователю предлагается указать название задачи, описать, что нужно выполнить в рамках этой задачи, а также указать исполнителей и проверяющих.

На следующем рисунке (рисунок 37) наглядно показано, как будет выглядеть доска проекта с несколькими задачами.

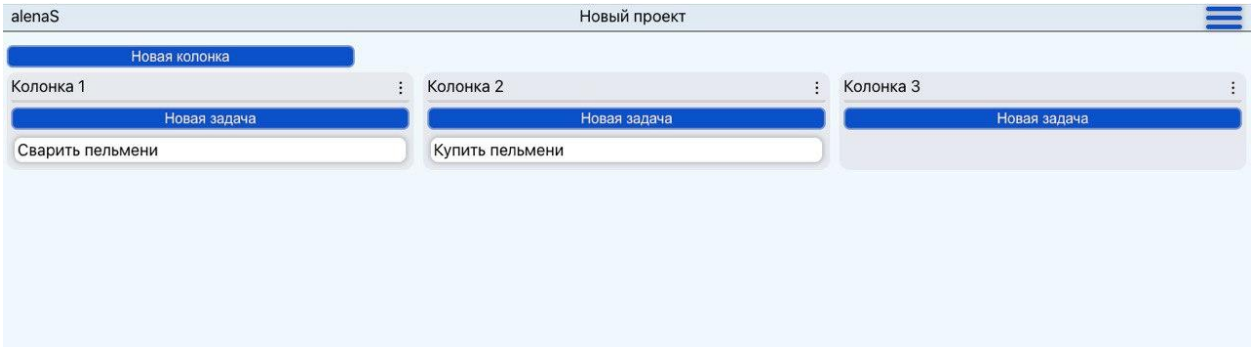


Рисунок 37 - Доска с задачами

На рисунке 37 обозначена доска (рабочее пространство) с тремя колонками, при этом в двух колонках находятся по одной задачи.

Вывод по главе 2.

Во второй главе было произведено проектирование программного обеспечения, описана реализация функциональных требований и выбраны средства и технологии разработки.

В результате проектирование веб-приложения были созданы диаграммы классов, компонентов, развертывания и реализована концептуальная модель базы данных. Описание реализованных функциональных требований заключалось в предоставлении блок-схема алгоритмов, их исходного кода, а также в демонстрации скриншотов графического интерфейса пользователя.

Глава 3. Тестирование и отладка приложения

3.1 План тестирования

Цель плана – определить объем и методы проведения функционального и интеграционного тестирования программного продукта [18].

Методы тестирования:

- функциональное тестирование – проверка соответствия требованиям, изложенным в первой главе;
- интеграционное тестирование – проверка взаимодействия компонентов веб-приложения [9].

Область интеграционного тестирования:

- тестирование серверной и клиентской частей приложения. Тестирование проводится вручную, по заготовленным сценариям. С клиента выполняются действия с веб-приложением через GUI. Затем, также через GUI, отправляются запросы на сервер, после чего проверяется ответ сервера с помощью консоли разработчика и ручной проверки документов в базе данных;
- тестирования работы сервера с базой данных MongoDB. Для этого тестирования были написаны несколько автоматических тестов;
- тестирование сервера веб-приложения с внешним сервисом – почты Рег.ру. Данный тест также автоматический.

Область функционального тестирования:

- проверка всей функциональности приложения с точки зрения конечного пользователя путем ручного взаимодействия со всеми элементами графического интерфейса.

График тестирования представлен в таблице 6.

Таблица 6 – График тестирования

Метод тестирования	Длительность	Ответственный
Интеграционный	2 недели	QA-инженер
Функциональный	2 недели	QA-инженер

Окружение и ресурсы:

- серверы – Node.js с Express.js, MongoDB с Mongoose;
- инструменты – jest, Яндекс Браузер, Google Chrome;
- данные – тестовые данные, добавленные вручную через графический интерфейс пользователя.

Критерии завершения:

- все критические ошибки исправлены;
- все недоработки устранены;
- автоматические тесты завершаются успешно;
- тестовые сценарии успешно пройдены.

3.2 Интеграционное тестирование

Рассмотрим автоматические тесты серверной части приложения:

- Project.test.js – тестирование работы с коллекцией базы данных Project;
- Board.test.js – тестирование работы с коллекцией базы данных Board;
- Column.test.js – тестирование работы с коллекцией базы данных Column;
- Card.test.js – тестирование работы с коллекцией базы данных Card;
- mailSender.test.js – тестирования модуля работы с почтой. В данном тесте проверяется корректность использования почтовой службы Рег.ру.

Тестовые сценарии представлены в таблице Ж.1 Приложения Ж. Данные сценарии покрывают большую часть приложения и позволяют обнаружить большинство ошибок.

3.3 Функциональное тестирование

Переходя к функциональному тестированию, стоит помнить, что цель данного тестирования – определить, насколько разработанное ПО соответствует завяленным требованиям. Здесь проверяется, что именно должно происходить, когда пользователь взаимодействует с системой. Поэтому все тестирования происходит вручную, без использования дополнительных инструментов.

Чтобы проводить тестирование эффективно, требуется создать тестовые сценарии – шаги тестирования тех или иных функций приложения. Таким образом можно однозначно определить, какой функционал работает в соответствии с требованиями, что работает некорректно или с ошибками. Рассмотрим тестовые сценарии для проведения функционального тестирования. Для этого обратимся к таблице Ж.2 Приложения Ж. Исходя из таблицы, можно заметить, что в колонке “Результат” мы рассматриваем, что должен получить, увидеть пользователь, например, при создании проекта пользователь должен увидеть созданный проект в списке доступных проектов.

Таким образом, функциональное тестирование позволяет точно определить, что разработанное приложение в полном объеме соответствует заявленным функциональным требованиям.

3.4 Обнаруженные ошибки

Рассмотрим некоторые ошибки, которые были выявлены и исправлены в результате тестирования приложения по тестовым сценариям. Для этого обратимся к таблице 7.

Таблица 7 - Отслеживание ошибок

№	Описание ошибки	Шаги воспроизведения	Серьезность	Приоритет
1	У пользователя, не являющегося автором проекта, отображаются кнопки “Удалить” и “Изменить”	1. Войти в аккаунт 2. Перейти на домашнюю страницу 3. Выбрать проект, автор которым пользователь не является	Значительная	Средний
2	У пользователя, не являющегося автором или администратором проекта, есть кнопка “Добавить доску”	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект, автор которым пользователь не является. 4. Нажать на кнопку “Открыть”. 5. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана.	Значительная	Средний
3	Пользователь, не являющийся автором задачи или проекта, исполнителем или проверяющим задачи, имеет интерфейс для изменения задачи.	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект, автор которым пользователь не является. 4. Нажать на кнопку “Открыть”. 5. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 6. Открыть любую задачу, где пользователь не числиться автором, исполнителем или проверяющим.	Значительная	Средний

Продолжение таблицы 7

№	Описание ошибки	Шаги воспроизведения	Серьезность	Приоритет
4	Ошибка при создании доски.	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект, в котором автор является автором или администратором. 4. Нажать на кнопку “Открыть”. 5. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 6. Нажать на кнопку “Добавить доску”	Критическая	Высокий
5	Ошибка при добавлении исполнителей или проверяющих к задачи.	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект, в котором пользователь является автором или администратором. 4. Нажать на кнопку “Открыть”. 5. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 6. Нажать на любую задачу. 7. Изменить список проверяющих или исполнителей. 8. Нажать на кнопку “Обновить задачу”	Критическая	Высокий

Продолжение таблицы 7

№	Описание ошибки	Шаги воспроизведения	Серьезность	Приоритет
6	При создании карточки нет кнопок и полей ввода	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект. 4. Нажать на кнопку “Открыть”. 5. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 6. Выбрать доску 7. Нажать на кнопку “Создать задачу”	Значительная	Средний
7	Ошибка при смене названия только что созданной доски	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект. 4. Нажать на кнопку “Открыть”. 5. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 6. Нажать на кнопку “Добавить доску” 7. Нажать на кнопку “Три точки” у только что добавленной доски 8. В появившемся меню выбрать пункт “Переименовать”	Значительная	Средний

Продолжение таблицы 7

№	Описание ошибки	Шаги воспроизведения	Серьезность	Приоритет
8	Ошибка при попытке добавления колонки к только что созданной доске	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект. 4. Нажать на кнопку “Открыть”. 5. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 6. Нажать на кнопку “Добавить доску” 7. Нажать на только что добавленную доску 8. Нажать на кнопку “Новая колонка”	Значительная	Средний

Из таблицы 7 можно сделать вывод, что большинство ошибок связаны с графическим интерфейсом и отображением графических элементов, которые не должны отображаться для пользователя с данной ролью. Также стоит отметить, что в процессе тестирования не были обнаружены блокирующие ошибки, парализующие работу всей системы.

Вывод по главе 3.

В третьей главе было проведено интеграционное и функциональное тестирование разработанного веб-приложения для управления проектами. Результаты тестирования показали стабильность и надежность функционирования приложения в различных сценариях использования. Помимо этого, тесты показали, что приложение соответствует заявленным требованиям. Был составлен отчет по обнаруженным ошибкам (таблица 7), которые были успешно устранены.

Заключение

Разработано веб-приложение по управлению проектами. Практическая ценность обосновывается ростом числа IT компаний и цифровизацией бизнеса. Это приводит к росту числа проектов внутри компаний, что приводит к необходимости управления проектами – определения числа участников в проекте, распределения задач между сотрудниками и отслеживания этапов выполнения задач. Традиционное управление проектами с использованием журналов, блокнотов и ручек с карандашами является неэффективным. К тому же такой подход усложняет коммуникацию между сотрудниками, которые работают дистанционно. Использование облачных платформ, например, Google Drive или Яндекс 360, тоже является малоэффективным, так как данные инструменты не предназначены для управления проектами и являются офисным программным обеспечением. Поэтому компаниям приходится использовать уже существующие средства для управления проектами – Kaiten, Jura, Pyrus и многие другие, но все они имеют ряд недостатков – высокая стоимость, перегруженность функционала, закрытый исходный код и ограниченность или полное отсутствие возможности развертывания на серверах компании. Это все несет в себе проблемы, например, утечка информации. Как итог, появляется потребность в разработке нового решения на рынке средств управления проектами – каким и является разработанное веб-приложение.

В рамках реализации веб-приложения, были изучены представленные на рынке конкуренты и приведен их сравнительный анализ для выявления закономерных недостатков. Далее, было спроектировано приложение – для этой задачи были определены инструменты и технологии, применяемые в процессе разработки, и нарисованы диаграммы, раскрывающие сущность ПО. Диаграммы реализовывались с помощью UML нотации и специализированного ПО для их рисования – Ramus и draw.io.

Следующий этап посвящен проектированию базы данных веб приложения. Для этого была реализована логическая модель базы данных, отражающая все ее элементы.

После этапа проектирования описана реализация функциональных требований - нарисованы блок-схемы ключевых алгоритмов и представлены скриншоты веб-интерфейса программы.

Заключительным этапом стало тестирование. В рамках тестирования был описан общий план, реализованы автоматические тесты и написаны тестовые сценарии, по которым проводилось ручное тестирование. Помимо этого, важной частью является отслеживание ошибок – собраны все обнаруженные в процессе тестирования проблемы.

Таким образом, было получено новое, качественное программное обеспечение, лишенное перегруженного интерфейса, имеющее открытый исходный код и позволяющее разворачивать ПО прямо на серверах компаний. Это позволило значительно улучшить безопасность и качество работ по управлению проектами.

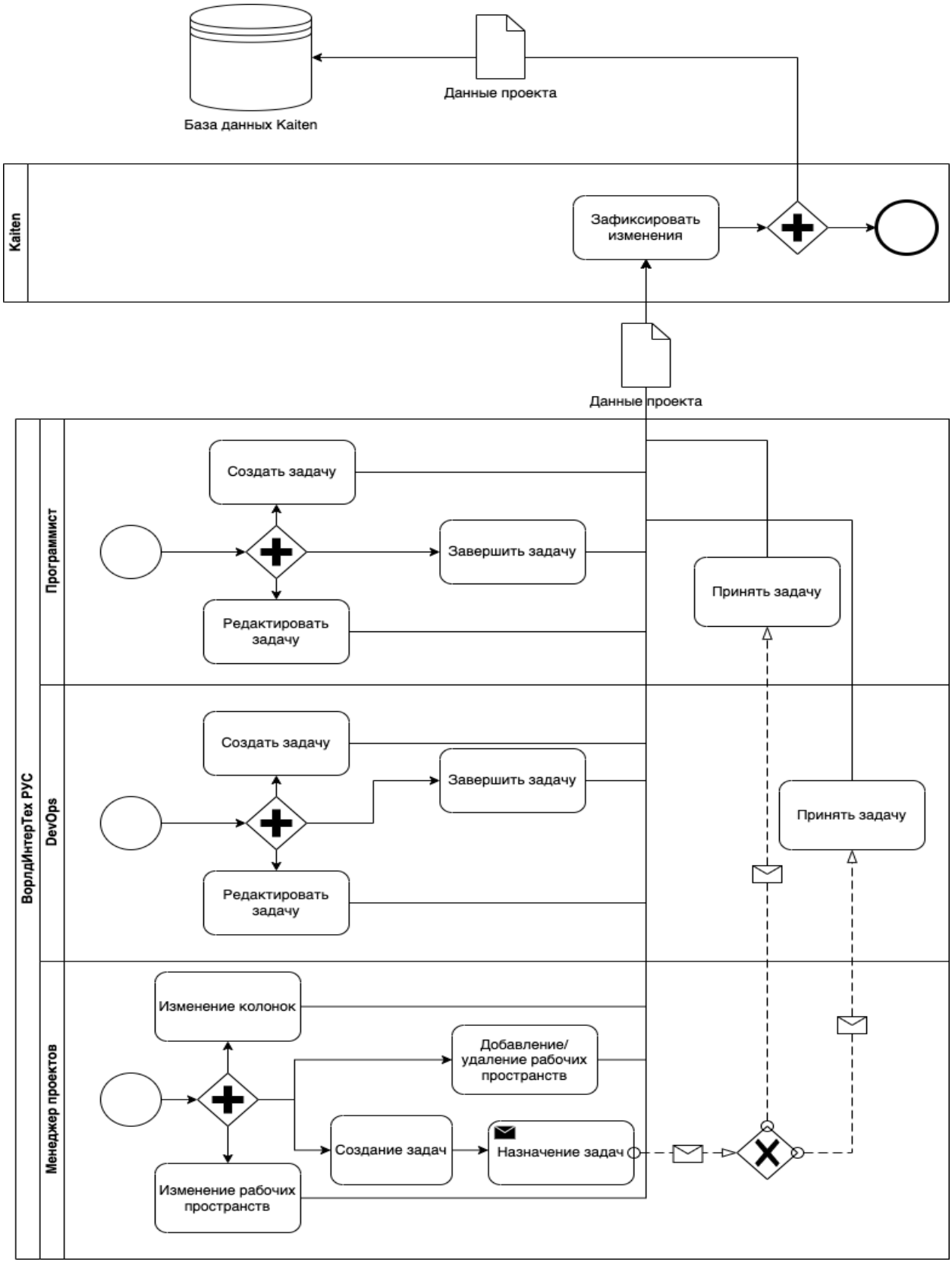
Список используемой литературы

1. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. - 2-е изд. - Санкт-Петербург: Символ-Плюс, 2007. - 624 с.
2. Бойцов А.К., Колмогорова С.С. Практическое моделирование на UML. - Санкт-Петербург: Реноме, 2023. - 210 с.
3. Бэнкер К. MongoDB в действии. - Москва: ДМК Пресс, 2012. - 368 с.
4. Вигерс К., Битти Д. Разработка требований к программному обеспечению. - 3-е изд. - Санкт-Петербург: БХВ-Петербург, 2014. - 576 с.
5. Городко С.И., Снисаренко С.В. Современные технологии программирования. - Минск: БГУИР, 2017. - 320 с.
6. Жилиндина О.В. Проектирование баз данных. - Благовещенск: АмГУ, 2023. - 41 с.
7. Каюмова А.В. Визуальное моделирование систем в StarUML. - Казань: Казанский федеральный университет, 2013. - 120 с.
8. Ковалев С., Ковалев В. Настольная книга аналитика. Практическое руководство по проектированию бизнес-процессов и организационной структуры. - Москва: 1С-Паблишинг, 2020. - 384 с.
9. Куликов С.С. Тестирование программного обеспечения. Базовый курс. - 3-е изд. - Минск: Четыре четверти, 2020. - 432 с.
10. Лаврищева Е.М., Петрухин В.А. Методы и средства инженерии программного обеспечения. - Москва: МФТИ, 2007. - 256 с.
11. Макконел С. Совершенный код. Практическое руководство по разработке программного обеспечения. - Т. 2. - Санкт-Петербург: БХВ-Петербург, 2018. - С. 70-80.
12. Мардан А., Барклунд М. React быстро. - Санкт-Петербург: Питер, 2024. - 288 с.

13. Моисеев А.Н., Литовченко М.И. Основы языка UML. - Томск: Издательство Томского государственного университета, 2023. - 150 с.
14. Хэррон Д. Node.js. Разработка серверных веб-приложений на JavaScript. - Москва: ДМК Пресс, 2016. - 354 с.
15. Ходорова К., Брэзил Й., Брэдшоу Ш. MongoDB. Полное руководство. - Москва: ДМК Пресс, 2020. - 518 с.
16. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. - Boston: Addison-Wesley, 2005. - 475 p.
17. Jin F. Designing React Hooks the Right Way. - Birmingham: Packt, 2022. - 310 p.
18. Jorgensen P.C., DeVries B. Software Testing: A Craftsman's Approach. - Boca Raton: CRC Press, 2019. - 522 p.
19. Podeswa H. UML for the IT Business Analyst. - 3rd ed. - Boston: Course Technology, 2009. - 408 p.
20. Scott T. Systems Analysis and Design. - Boston: Cengage Learning, 2020. - 480 p.
21. Silver B. BPMN Method and Style. - Altadena: Cody-Cassidy Press, 2019. - 322 p.

Приложение А

Диаграмма “Как есть”



Приложение Б

Фрагмент кода модуля userRouter.js

```
userRouter.post("/create", (request, response) => {
    console.log("Обработка запроса по маршруту /create")
    const body = request.body
    ConfirmationCode.findOne({email: body.email, confirmed: true}).then(codeData => {
        if(codeData){
            pbkdf2(body.password, "HelloFromTogliatty", 1020107, 64, 'sha512', (err, hash)
=> {
                if (err)
                    response.status(500).send()
                else{
                    const user = new User({
                        FirstName: body.FirstName,
                        LastName: body.LastName,
                        Patronymic: body.Patronymic,
                        email: body.email,
                        login: body.login,
                        password: hash.toString('base64'),
                    })

                    user.save().then(async newUserDoc => {
                        console.log("Ok");
                        console.log(newUserDoc)
                        response.status(200).cookie("token",                    await
generateJWT(newUserDoc._id), {expires: new Date(Date.now() + 7*24*60*60*1000), httpOnly:
true, secure: true, sameSite: "none"}).send("Ok")
                    }).catch(e => {console.log(`${e}`)})
                }
            })


        }
        else
            response.status(500).json({"message": "ПОЧТА НЕ ПОДТВЕРЖДЕНА"})
```

```
    })

})
userRouter.get("/get", (request, response) => {
    console.log(request.cookies)
    const _id = request.cookies.token.payload.data
    User.findOne({_id}, {password: 0})
        .then(async data => {
            response.status(200).json(data)
        })
        .catch(error => {
            // console.log(error)
            response.status(500).json({message: "something Error"})
        })
})

userRouter.post("/login", (request, response) => {
    const body = request.body
    pbkdf2(body.password, "HelloFromTogliatty", 1020107, 64, 'sha512', (err, hash) => {
        if (err)
            response.status(500).send()
        else{
            body.password = hash.toString("base64")
            if (body){
                User.findOne(body, {password: 0})
                .then(async data => {
                    response.status(200).cookie("token", await generateJWT(data._id), {expires: new Date(Date.now() + 7*24*60*60*1000), httpOnly: true, secure: true, sameSite: "none"}).json(data)
                })
                .catch(error => {
                    response.status(500).json({message: "something Error"})
                })
```

```
            }
            else
                response.status(400).json({"message": "body EMPTY!!!!!"})
        }
    })

})

userRouter.post("/sendConfirmationCode", async (request, response) => {
    const body = request.body
    if (body.email){
        if (!/\S+@\S+\.ru/.test(body.email))
            response.status(500).json({"type": "NotRuDomen"})
        else
            ConfirmationCode.findOne({email: body.email, confirmed: true}).then(codeData
=> {
                if (codeData)
                    response.status(500).json({"type": "EmailConfirmed"})
                else{
                    const code = Math.floor(Math.random()*1000000)
                    new ConfirmationCode({
                        email: body.email,
                        code: code
                    }).save().then(data => {
                        console.log(data)
                        sendEmail(body.email, "Подтверждение электронной почты", `Ваш код
подтверждения: ${code}`)
                            .then(dataSended => {
                                console.log(dataSended)
                                response.status(200).json({"message": "Confirmation code send!"})
                            }).catch(error => {
                                console.log(error)
                                response.status(400).json({"message": "ERROR!"})
                            })
```

```
                    })
                }
            })

        }
        else
            response.status(400).send("Some error")
    })

    userRouter.get("/checkConfirmationCode", (request, response) => {
        const {code, email} = request.query
        console.log(email)
        ConfirmationCode.findOne({email: email, confirmed: true}).then(data => {
            if (data)
                response.status(500).json({"message": "ПОЧТА УЖЕ ПОДТВЕРЖДЕНА"})
            else
                ConfirmationCode.findOneAndUpdate({email: email, code: code}, {confirmed:
true, TTL: null}, {new: true}).then(uData => {
                    console.log(uData);
                    response.status(200).json({"confirm": true})
                }).catch(uErr => console.log(uErr))
        })
        .catch(error => response.status(500).json({"message": error}))
    })

    userRouter.post("/resetPsswrd", (request, response) => {
        const {password, code} = request.body
        if                  (/(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*><%\\|/-><-
_`~§±|\\\]\[\{\}\(\)-]).{8,}/g.test(password))
            ResetLink.findOne({code: code}).then(linkData => {
                console.log(linkData)
                pbkdf2(password, "HelloFromTogliatty", 1020107, 64, 'sha512', (err, hash) => {
                    if(err)
```

```
                response.status(500).send()
            else{
                User.findOneAndUpdate({email: linkData.email}, {password: hash.toString("base64")}).then(result => {
                    console.log(result)
                    response.status(200).json({message: "Password reset"})
                }).catch(updateError => {
                    console.log(updateError)
                    response.status(500).json({message: "Passwod reset FAILED"})
                })
            }
        })
    }).catch(error => {
        console.log(error)
        response.status(500).json({message: "Passwod reset FAILED"})
    })
else
    response.status(500).json({message: "Passwod is not correct"})
})

userRouter.post("/sendResetLink", (request, response) => {
    const {email} = request.body

    const code = uuidv4()
    User.findOne({email: email}).then(async data => {
        if(data !== null){
            const linkData = await ResetLink.findOne({email: email, code: code})
            if (linkData === null)
                new ResetLink({email: email, code: code}).save().then(linkData => {
                    sendEmail(email, "Сброс пароля", `Для сброса пароля перейдите по ссылке: http://127.0.0.1:3000/resetPassword?code=${code}`)
                    response.status(200).json({message: "Server is running"})
                }).catch(errorLink => {
                    console.log(errorLink)
```

```
                response.status(500).json({message: "Error created link"})
            })
        else
            response.status(500).json({message: "Error link  already exists"})
    }
    else
        response.status(500).json({message: "Email not found"})
}).catch(error => console.log(error))

})
module.exports = userRouter
```

Приложение В

Фрагмент кода модуля projectRouter.js

```
projectRouter.get('/getAll', (req, res) => {
   const userId = req.cookies.token.payload.data
   if (userId){
      Project.find({members: {$in: [userId]}}, {name: 1, author: 1, _id: 1}).populate({path: "author", model: User}).then(data => {
         console.log(data)
         res.status(200).json({projects: data})
      }).catch(error => {
         console.log(error)
         res.status(500).json({message: "Something went wrong"})
      })
   }
})

projectRouter.post('/create', (req, res) => {
   const data = req.body
   console.log(data)

   if(data){
      const memberLogins = data.roles.map(role => role.login)
      User.find({login: {$in: memberLogins}}, {_id: 1, login: 1}).then(users => {
         data.members = users.map(user => user._id.toString())
         data.roles = data.roles.map(role => {
            const u = users.find(us => us.login == role.login)
            if (u)
               return ({_id: u._id.toString(), role: role.role}) // userLogin: u.login,
         })
         new Project({...data, author: req.cookies.token.payload.data}).save()
            .then(projectData => {
               User.findOne({_id: projectData.author}, {password: 0})
               .then(user => {
                  const ObjectProjectData = projectData.toObject()
                  ObjectProjectData.author = {...user.toObject()}
```

```
                res.status(200).json({message: "ProjectCreated", project: ObjectProjectData})
            })
            .catch(err => res.status(500).json({err: err, type: "SomeError"}))


        })
      .catch(err => res.status(500).json({err: err, type: "SomeError"}))
    })

  }
  else
    res.status(500).json({type: "NoBody"})
})

projectRouter.post('/update', (req, res) => {
  const data = req.body

  if (data){
    if (data._id){
      const userId = req.cookies.token.payload.data
      Project.findOne({_id: data._id, author: userId}).then(projectData => {
        if(projectData){
          const memberLogins = data.roles.map(role => role.login)

          User.find({login: {$in: memberLogins}}, {_id: 1, login: 1}).then(users => {
            data.members = users.map(user => user._id.toString())
            // С FrontEnd приходят данные с логином и ролью,
            // но в БД надо хранить _id пользователя и роль,
            // т.к. _id постоянный, а login может меняться
            data.roles = data.roles.map(role => {
              const u = users.find(us => us.login == role.login)
              if (u)
                return ({_id: u._id.toString(), role: role.role})
            })
            Project.findOneAndUpdate({_id: data._id}, data)
```

```
                .then(pData => {res.status(200).json(pData)})
                .catch(err => res.status(400).json({err: err, type: "SomeError"}))
            })
        }
        else{
            res.status(403).json({type: "NoAccess"})
        }
    })
  }
  else
    res.status(500).json({type: "NoIdProject"})
}
else
  res.status(400).json({type: "NoBody"})
})

projectRouter.delete('/delete', (req, res) => {
  const { _id } = req.body

  if(_id){
    Project.findOneAndDelete({_id: _id, author: req.cookies.token.payload.data})
    .then(pData => {
      if(pData)
        Board.deleteMany({projectId: _id}).then(_ => {
          Column.deleteMany({projectId: _id}).then(_ => {
            Card.deleteMany({projectId: _id}).then(_ => {
              res.status(200).json({message: "ProjectDeleted"})
            })
            .catch(err => res.status(500).json({err: err, type: "CardDeleteError"}))
          })
          .catch(err => res.status(500).json({err: err, type: "ColumnDeleteError"}))
        })
        .catch(err => res.status(500).json({err: err, type: "BoardDeleteError"}))
      else
```

```
            res.status(403).json({type: "NoAccess"})
        })
        .catch(err => res.status(500).json({err: err, type: "ProjectDeleteError"}))
    }
})

projectRouter.get('/getOne', (req, res) => {
    const {projectId} = req.query

    if(projectId)
        Project.findOne({_id: projectId, members: {$in: [req.cookies.token.payload.data]}})
        .populate({
            path: 'boards',
            model: Board,
            populate: [
                {
                    path: 'columns',
                    model: Column,
                    populate: {
                        path: 'cards',
                        model: Card,
                    }
                },
                {
                    path: 'members',
                    model: User,
                    select: '-password'
                }
            ]
        })
        .populate({
            path: 'author',
            model: User,
            select: '-password'
```

```
    })
    .then(pData => {
      let projectData = {...pData.toObject()}
      User.find({_id: {$in: projectData.members}}, {_id: 1, login: 1, FirstName: 1, LastName:
1, Patronymic: 1}).then(uData => {
        uData = uData.map(element => {return(element.toObject())})
        projectData.roles = projectData.roles.map(role => {
          const u = uData.find(us => us._id.toString() === role._id.toString())
          if (u)
            return ({_id: u._id.toString(), login: u.login, FirstName: u.FirstName, LastName:
u.LastName, Patronymic: u.Patronymic, role: role.role}) // userLogin: u.login,
        })
        res.status(200).json(projectData)
      })
    }).catch(projectErr => {console.log(projectErr); res.status(400).json({type: "SomeError"})})
  else
    res.status(400).json({type: "NoData"})
})
```

Приложение Г

Блок схемы алгоритмов

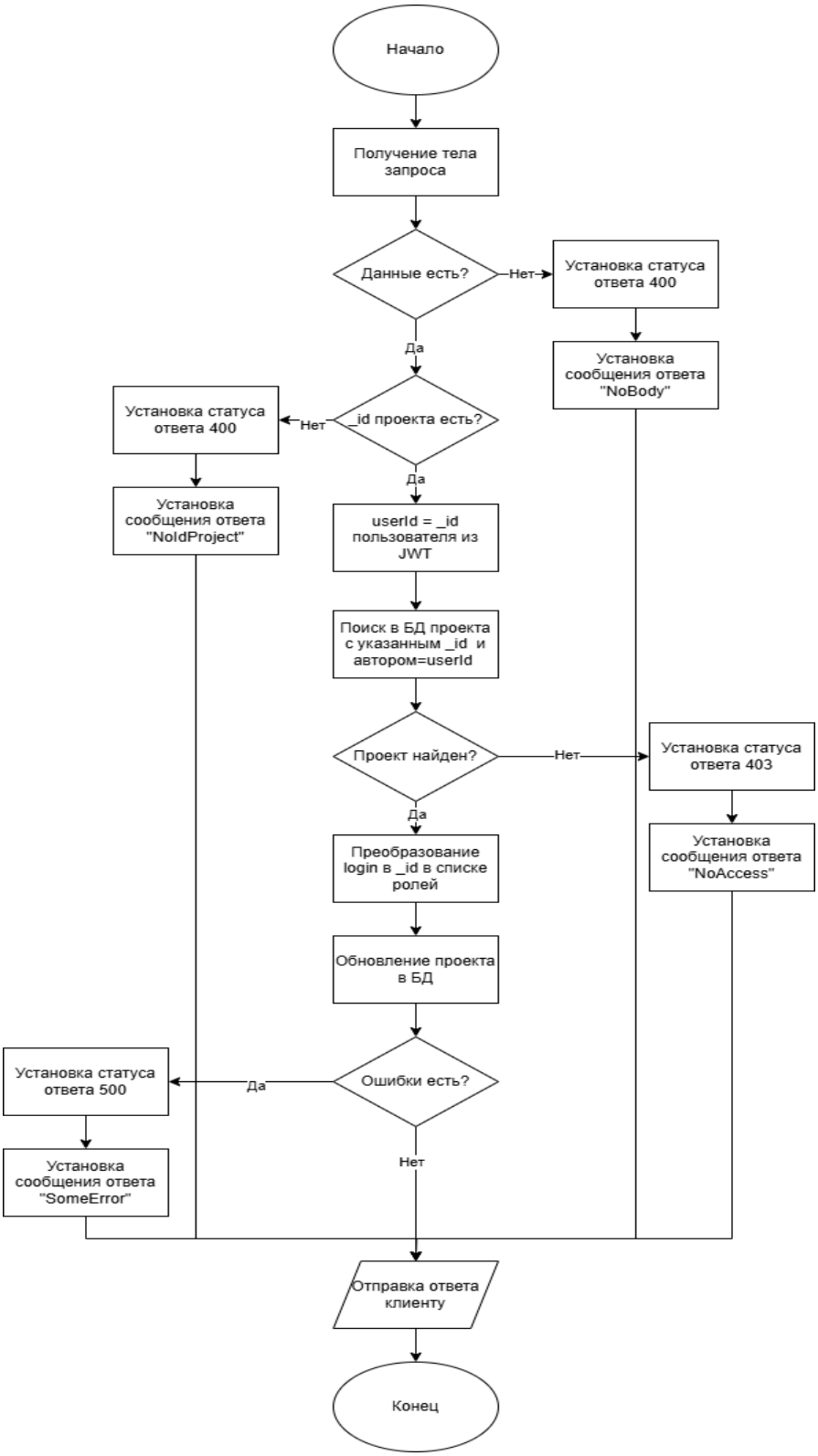


Рисунок Г.1 - Обновление проекта

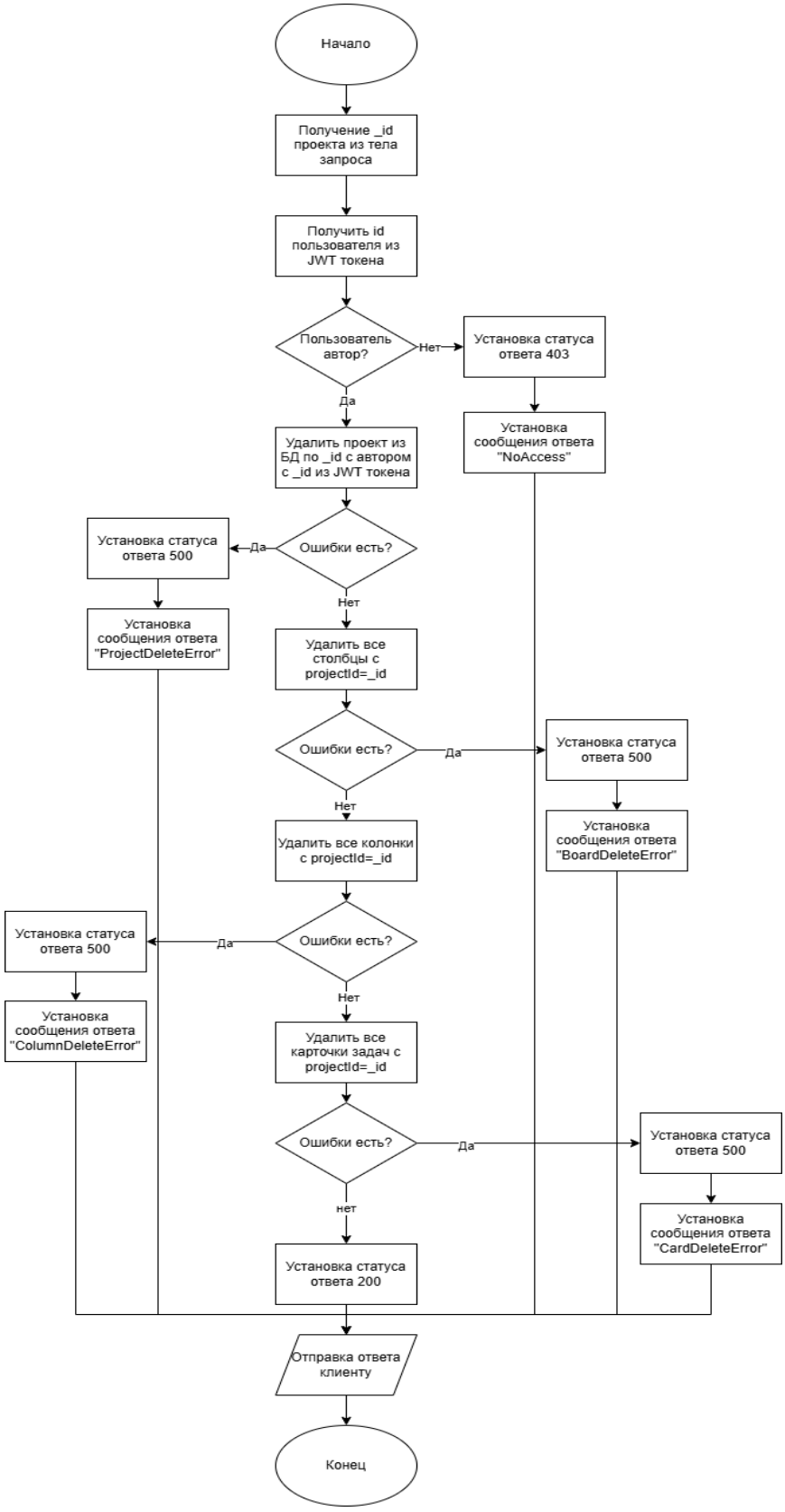


Рисунок Г.2 - Удаление проекта

Приложение Д

Фрагмент кода модуля boardRouter.js

```
boardRouter.post("/add",  (req, res) => {
   const data = req.body
   const userId = req.cookies.token.payload.data
   if(data)
      // Проверяем, не пытаются ли создать дубликат доски
      Board.findOne({projectId: data.projectId, name: data.name}).then(boardData
=> {
         if (boardData){
            console.log("Доска с таким именем уже существует")
            res.status(400).json({type: "BoardAlreadyExist"})
         }else{
            Project.findOne({_id: data.projectId}).then(projectData => {
               if(projectData.author === userId || projectData.roles.some(role =>
role._id === userId && role.role === "admin")){
                  // Создаем объект доски, сохраняем его в коллекции
                  const board = new Board(data)
                  board.save().then(bData => {
                     // Добавляем в список досок проекта только что созданную
доску
                     Project.findOneAndUpdate({_id:  data.projectId,},  {$push:
{boards: bData._id}})
                     .then(_ => res.status(200).json(bData))
                     .catch(err  =>  {console.log(err);  res.status(500).json({type:
"SomeError"})})
                  })
                  .catch(err  =>  {console.log(err);  res.status(500).json({type:
"SomeError"})})
               }
```

```
                else
                    res.status(403).json({type: "NoRights"})
            }).catch(err => {console.log(err); res.status(500).json({type:
"SomeError"})})


        }
    })
    .catch(error => {console.log(error); res.status(500).json({type:
"SomeError"})})

})

boardRouter.put("/rename", (req, res) => {
  const data = req.body
  const userId = req.cookies.token.payload.data
  if(data)
    Board.findOne({projectId: data.projectId, name: data.name}).then(boardData
=> {
        if (boardData){
            console.log("Доска с таким именем уже существует")
            res.status(400).json({type: "BoardAlreadyExist"})
        }else{
            Project.findOne({_id: data.projectId}).then(projectData => {
                if(projectData.author === userId || projectData.roles.some(role =>
role._id === userId && role.role === "admin")){
                    Board.findOneAndUpdate({_id: data._id, projectId: data.projectId},
{$set: {name: data.name}})
                    .then(_ => res.sendStatus(200))
```

```
                .catch(err => {console.log(err); res.status(500).json({type: "SomeError"})})
            }
            else
                res.status(403).json({type: "NoRights"})
        }).catch(err => {console.log(err); res.status(500).json({type: "SomeError"})})
    }
})
.catch(error => {console.log(error); res.status(500).json({type: "SomeError"})})
else
    res.status(400).json({type: "NoBody"})
})

boardRouter.put("/update", (req, res) => {
    const data = req.body
    const userId = req.cookies.token.payload.data
    if (data)
        // Ищем проект с projectId, указанным в data, и у которого в списке досок есть доска с _id, указанным в data
        Project.findOne({_id: data.projectId, boards: {$in: [data._id]}}).then(projectData => {
            if (!projectData)
                res.status(400).json({type: "ProjectDoesNotHaveThisBoard"})
            else{
                if(projectData.author === userId || projectData.roles.some(role => role._id === userId && role.role === "admin")){
```

```
                Board.findOneAndUpdate({_id: data._id, projectId: data.projectId}, data)
                .then(() => res.sendStatus(200))
                .catch(err => res.status(500).json({type: "SomeError", message: err}))
            }
            else
                res.status(403).json({type: "NoRights"})
        }
    })
    .catch(err => res.status(500).json({type: "SomeError", message: err}))
else
    res.status(400).json({type: "NoBody"})
})

boardRouter.delete("/delete", (req, res) => {
    const userId = req.cookies.token.payload.data
    const data = req.body
    if (data)
        Project.findOne({_id: data.projectId, boards: {$in: [data._id]}}).then(projectData => {
            if (!projectData)
                res.status(400).json({type: "ProjectDoesNotHaveThisBoard"})
            else
                if (projectData.author === userId || projectData.roles.some(role => role._id === userId && role.role === "admin")){
                    const updatedBoards = projectData.boards.filte(boardId => boardId !== data._id)
                    Project.updateOne({_id: data.projectId}, {boards: updatedBoards}).then(_ => {
```

```
                Board.deleteOne({_id: data._id}).then(_ => {
                    Column.deleteMany({boardId: data._id}).catch(err => res.status(500).json({type: "ColumnDeleteError", message: err}))
                    Card.deleteMany({boardId: data._id}).catch(err => res.status(500).json({type: "CardDeleteError", message: err}))
                    res.sendStatus(200)
                }).catch(err => res.status(500).json({type: "BoardDeleteError", message: err}))
            }).catch(err => res.status(500).json({type: "ProjectListBoardsChangeError", message: err}))
        }
        else
            res.status(403).json({type: "NoRights"})
    }).catch(err => res.status(500).json({type: "SomeError", message: err}))
else
    res.status(400).json({type: "NoBody"})
})
```

Приложение Е

Фрагмент кода модуля cardRouter.js

```
cardRouter.post("/add", async (req, res) => {
    const data = req.body;
    // console.log(data)
    if (data && data.executors && data.reviewers){
        data.executors = await User.find({login: {$in: data.executors}}, {_id: 1})
        data.reviewers = await User.find({login: {$in: data.reviewers}}, {_id: 1})
        new Card(data).save()
        .then(cData => {
            Column.findOneAndUpdate({_id: data.columnId}, {$push: {cards: cData._id}})
            .then(_ => res.status(200).json(cData))
            .catch(err => {console.log(err); res.status(500).json({type: "PushCardToColumnError"})})
        })
        .catch(error => {console.log(error); res.status(500).json({type: "CreateCardError"})})
    }
    else
        res.status(400).json({error: error, type: "NoBody"})
})

cardRouter.get("/get", (req, res) => {
    const _id = req.query._id
    if (_id)
        Card.findOne({_id})
        .populate({
            path: 'executors',
            model: User,
            select: 'login'
        })
        .populate(
            {
            path: 'reviewers',
            model: User,
            select: 'login',
```

```
        })
        .populate(
            {
            path: 'author',
            model: User,
            select: 'login',
        })
        .then(data => {
            res.status(200).json(data)
        })
        .catch(err => {console.log(err); res.status(500).json({type: "SomeError"})})
    else
        res.status(400).json({type: "NotId"})
})

cardRouter.put("/update", (req, res) => {
    const userId = req.cookies.token.payload.data
    const data = req.body

    if(Object.keys(data).length === 0 || !data){
        return res.status(400).json({type: "NoBody"})
    }

    const {_id, projectId} = data
    delete data["_id"]
    delete data["projectId"]
    delete data["columnId"]
    delete data["boardId"]

    Project.findOne({_id: projectId}).then(async projectData => {
        let userHaveRightForUpdateCard = false // обладает ли пользователь правами на
изменение

        if (projectData){
```

```
        // В следующем if-else блоках определяем, является ли пользователь
        // автором задачи, аввтором проекта или администратором проекта.
        if (projectData.author === userId || projectData.roles.some(role => role._id === userId &&
role.role === "admin"))
            userHaveRightForUpdateCard = true // user админ или автор проекта
        else{
            const card = await Card.findOne({_id, author: userId})
            if (card) // Если найдена задача, где Пользователь - её автор
                userHaveRightForUpdateCard = true
        }
    }else
        return res.status(400).json({type: "ProjectNotFound", message: "Something went
wrong"})


    if (userHaveRightForUpdateCard){ // если пользователь имеет права на изменение
        // обновляем задачу
        const [executorsData, reviewersData] = await Promise.all([
            User.find({ login: { $in: data.executors || [] } }, { _id: 1 }),
            User.find({ login: { $in: data.reviewers || [] } }, { _id: 1 })
        ]);
        data.executors = executorsData.map(executor => executor._id.toString())
        data.reviewers = reviewersData.map(reviewer => reviewer._id.toString())
        Card.findOneAndUpdate({_id}, data, {new: true}).then(data => {console.log(data);
res.sendStatus(200);}).catch(error => {console.log(error); res.status(500).json({type:
"UpdateCardError"})})
    }
    else // пользователь не имеет прав на изменение
        res.sendStatus(403).json({type: "NoRights"})
  })
})

cardRouter.put("/changeStatus", (req, res) => {
  const userId = req.cookies.token.payload.data
```

```
let {isDone, needRevision, onReview} = req.body

Project.findOne({_id: req.body.projectId}).then(projectData => {
    let userHaveRightForChangeStatus = false // обладает ли пользователь правами на изменение

    if (projectData){
        // В следующем if-else блоках определяем, является ли пользователь
        // аввтором проекта или администратором.
        if (projectData.author === userId || projectData.roles.some(role => role._id === userId && role.role === "admin"))
            userHaveRightForChangeStatus = true // user админ или автор проекта
    }else
        return res.status(404).json({type: "ProjectNotFound", message: "Something went wrong"})

    if (userHaveRightForChangeStatus){ //Если пользователь автор или администратор проекта
        // То сразу разрешаем ему обновлять статус задачи
        Card.findOneAndUpdate({_id: req.body._id}, {isDone, needRevision, onReview}, {new: true}).then(data => {console.log(data); res.sendStatus(200);}).catch(error => {console.log(error); res.status(500).json({type: "SomeError"})})
    } else{ // Пользователь не автор и не администратор проекта
        // Далее ищем карточку задачи, в которой пользователь либо исполнители, либо проверяющий и при этом карточка имеет конкретный статус
        Card.findOne({_id: req.body._id, $or: [{reviewers: {$in: [userId]}, $or: [{"onReview.status": true}, {"needRevision.status": true}, {"isDone.status": true}]}, {executors: {$in: [userId]}, $or: [{"needRevision.status": true}, {"needRevision.status": false, "isDone.status": false, "onReview.status": false}, {"onReview.status": true}]}]}).then(cardData => {
            if (cardData)
                Card.findOneAndUpdate({_id: req.body._id}, {isDone, needRevision, onReview}, {new: true})
                    .then(data => {console.log(data); res.sendStatus(200);})
```

```
                .catch(error => {console.log(error); res.status(500).json({type: "SomeError"})})
            else
                res.status(400).json({type: "WrongStatus", message: "you can't change the status to this one."})
        })
    }
  })
})

cardRouter.post("/move", async (req, res) => {
  const {parentColumnId, targetColumnId, card_id} = req.body
  if (!targetColumnId)
    res.status(500).json({error: "Error", type: "NoTargetColumn"})
  else if (!card_id)
    res.status(500).json({error: "Error", type: "NoCardId"})
  else if (!parentColumnId)
    res.status(500).json({error: "Error", type: "NoParentColumnId"})
  else{
    Card.findOneAndUpdate({_id: card_id}, {columnId: targetColumnId}) // Обновляем карточку
    .then(() => {
      Column.findOneAndUpdate({_id: parentColumnId}, {$pullAll: {cards: [card_id]}}) // Обновляем прошлую колонку
      .then(() => {
        Column.findOneAndUpdate({_id: targetColumnId}, {$push: {cards: card_id}}) // Обновляем новую колонку
        .then(() => res.status(200).json({message: "Successful"}))
        .catch(error => res.status(500).json({error: error, type: "SomeError"}))
      })
      .catch(error => res.status(500).json({error: error, type: "SomeError"}))
    })
    .catch(error => {
      res.status(500).json({error: error, type: "SomeError"})
    })
```

```
    }
})

cardRouter.delete("/delete", (req, res) => {
    const userId = req.cookies.token.payload.data
    const data = req.body

    if (data)
        Project.findOne({_id: data.projectId}).then(async projectData => {
            let userHaveRightForDeleteCard = false // обладает ли пользователь правами на изменение

            if (projectData){
                // В следующем if-else блоках определяем, является ли пользователь
                // автором задачи, аввтором проекта или администратором проекта.
                if (projectData.author === userId || projectData.roles.some(role => role._id === userId && role.role === "admin"))
                    userHaveRightForDeleteCard = true // user админ или автор проекта
                else{
                    const card = await Card.findOne({_id, author: userId})
                    if (card) // Если найдена задача, где Пользователь - её автор
                        userHaveRightForDeleteCard = true
                }
            }else
                return res.status(400).json({type: "ProjectNotFound", message: "Something went wrong"})

            if (userHaveRightForDeleteCard){
                Card.findOneAndDelete({_id: data._id}).then(cardData => {
                    Column.updateOne({_id: cardData.columnId}, {$pull: {cards: cardData._id}}).then(() => {
                        res.sendStatus(200)
                    }).catch(err => {console.log(err); res.status(500).json({type: "SomeError"})})
                }).catch(err => {console.log(err); res.status(500).json({type: "SomeError"})})
```

```
            }
            else
                res.sendStatus(403).json({type: "NoRights"})

        })
    else
        res.status(400).json({type: "NoBody"})
})
```

Приложение Ж

Тестовые сценарии

Таблица Ж.1 - Тестовые сценарии интеграционного тестирования

№	Название	Шаги тестирования	Результат	Статус
1	Создание проекта	1. Создать или войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Нажать кнопку “Создать”. 4. Заполнить все поля и нажать кнопку “Создать проект”.	. В качестве ответа сервер должен прислать 200 статус и объект с данными нового проекта, сохраненного в БД. В коллекции Projects должен появиться новый документ создаваемого проекта	Пройден
2	Удаление проекта	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект, в котором пользователь - автор. 4. Нажать на кнопку “Удалить”	В качестве ответа сервер должен вернуть 200 статус и сообщение “ProjectDeleted”. Из коллекции Projects БД должен исчезнуть документ удаленного проекта.	Пройден
3	Редактирование проекта	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать нужный проект, в котором пользователь - автор. 4. Нажать на кнопку “Изменить”. 5. Внести необходимые данные в форму. 6. Нажать на кнопку “Обновить”	В качестве ответа сервер должен прислать 200 статус и объект измененного проекта, сохраненного в БД. В БД документ редактируемого проекта должен быть изменен.	Пройден

Продолжение таблицы Ж.1

№	Название	Шаги тестирования	Результат	Статус
4	Добавление участника проекта	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь - автор. 4. Нажать на кнопку “Изменить”. 5. Добавить нового участника проекта. 6. Нажать на кнопку “Обновить”. 7. Войти в аккаунт участника проекта. 8. Перейти на домашнюю страницу.	В качестве ответа сервер должен прислать 200 статус и объект проекта с новым пользователем. В базе данных должен добавиться новый _id и роль в массиве roles проекта.	Пройден
5	Добавление доски	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь – автор или администратор. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Нажать на кнопку “Добавить доску”. 6. Ввести название доски. 7. Кликнуть в любое свободное место мимо доски или нажать на кнопку Enter.	В качестве ответа сервер должен прислать 200 статус и объект новой доски, сохраненной в БД. В БД в коллекции Boards должен появиться новый документ доски. В коллекции Projects базы данных должен добавиться _id новой доски в массив досок boards.	Пройден

Продолжение таблицы Ж.1

№	Название	Шаги тестирования	Результат	Статус
6	Удаление доски	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь – автор или администратор. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Нажать на три точки рядом с необходимой доской. 6. В выпадающем меню выбрать пункт “Удалить”.	В качестве ответа сервер должен прислать 200 статус и сообщение “BoardDeleted”. Из коллекции Boards должен удалиться документ доски. Из массива boards документа коллекции Projects должен удалиться _id удаляемой доски.	Пройден
7	Обновление доски	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь – автор или администратор. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Нажать на три точки рядом с необходимой доской. 6. В выпадающем меню выбрать пункт “Изменить”. 7. Внести изменения в форму. 8. Нажать на кнопку “Обновить”	В качестве ответа сервер должен прислать 200 статус и объект новой измененной доски, сохраненной в БД. В БД документ редактируемой доски должен быть изменен.	Пройден

Продолжение таблицы Ж.1

№	Название	Шаги тестирования	Результат	Статус
8	Добавление колонки	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь автор или администратор проекта. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску, нажав на название ЛКМ. 6. Нажать “Новая колонка”.	В качестве ответа сервер должен прислать 200 статус и объект новой доски, сохраненной в БД. В коллекцию Columns базы данных должен появиться новый документ. В коллекции Boards в документ доски, к которой добавляется доска, в массив columns должен быть добавлен _id новой колонки.	Пройден
9	Изменение названия колонки	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь автор или администратор проекта. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску, нажав на название ЛКМ. 6. Нажать на три вертикальные точки у любой колонки 7. Выбрать пункт “Переименовать” 8. Ввести новое название	В качестве ответа сервер должен прислать статус 200 и сообщение Ok. В БД в коллекции columns в документе обновляемой колонки поле title должно быть изменено на требуемое название.	Пройден

Продолжение таблицы Ж.1

№	Название	Шаги тестирования	Результат	Статус
10	Удаление колонки	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь автор или администратор проекта. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску, нажав на название ЛКМ. 6. Нажать на три вертикальные точки у любой колонки 7. Выбрать пункт “Удалить” 8. Подтвердить удаление	В качестве ответа сервер должен прислать 200 статус и сообщение “ColumnDeleted”. Из коллекции Columns должен удалиться документ доски. Из массива columns документа коллекции Boards должен удалиться _id удаляемой колонки.	Пройден
11	Добавление задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Нажать на кнопку “Новая задача”. 7. Заполнить форму необходимыми данными. 8. Нажать на кнопку “Создать задачу”	В качестве ответа сервер должен прислать 200 статус и объект новой задачи, сохраненной в БД. В базе данных в коллекции Cards должен появиться новый документ с новой задачей, в коллекции Columns в массив cards должен добавиться _id новой задачи.	Пройден

Продолжение таблицы Ж.1

№	Название	Шаги тестирования	Результат	Статус
12	Изменение задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Нажать на задачу, автором которой пользователь является автором или любую задачу, если пользователь является автором или администратором проекта 7. Изменить данные в форме. 8. Нажать на кнопку “Обновить задачу”	В качестве ответа сервер должен прислать статус 200 и сообщение Ok. В БД в коллекции cards в документе обновляемой задачи данные поля должны измениться.	Пройден
13	Удаление задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Нажать на задачу, автором которой пользователь является автором или любую задачу, если пользователь является автором или администратором проекта 7. Нажать на кнопку “Удалить задачу” 8. Подтвердить удаление	В качестве ответа сервер должен прислать 200 статус и сообщение “CardDeleted”. Из коллекции Cards должен удалиться документ доски. Из массива cards документа коллекции Cards должен удалиться _id удаляемой задачи.	Пройден

Таблица Ж.2 - Сценарии для функционального тестирования

№	Название	Шаги тестирования	Результат	Статус
1	Создание проекта	1. Создать или войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Нажать кнопку “Создать”. 4. Заполнить все поля и нажать кнопку “Создать проект”.	На домашней странице в списке проектов появился созданный проект	Пройден
2	Удаление проекта	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать проект, в котором пользователь - автор. 4. Нажать на кнопку “Удалить”	На домашней странице в списке проектов должен исчезнуть удаленный проект как у автора, так и у остальных участников проекта	Пройден
3	Редактирование проекта	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать нужный проект, в котором пользователь - автор. 4. Нажать на кнопку “Изменить”. 5. Внести необходимые данные в форму. 6. Нажать на кнопку “Обновить”	На домашней странице в списке проектов проект должен быть изменен как у автора проекта, так и у всех участников проекта	Пройден

Продолжение таблицы Ж.2

№	Название	Шаги тестирования	Результат	Статус
4	Добавление участника проекта	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь - автор. 4. Нажать на кнопку “Изменить”. 5. Добавить нового участника проекта. 6. Нажать на кнопку “Обновить”. 7. Войти в аккаунт участника проекта. 8. Перейти на домашнюю страницу.	В списке проектов у добавленного как участник пользователя должен появится новый проект, в который его добавили.	Пройден
5	Добавление доски	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь – автор или администратор. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Нажать на кнопку “Добавить доску”. 6. Ввести название доски. 7. Кликнуть в любое свободное место мимо доски или нажать на кнопку Enter.	В списке досок должна появиться новая доска	Пройден

Продолжение таблицы Ж.2

№	Название	Шаги тестирования	Результат	Статус
6	Удаление доски	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь – автор или администратор. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Нажать на три точки рядом с необходимой доской. 6. В выпадающем меню выбрать пункт “Удалить”.	В списке досок должна исчезнуть доска, которую удалили	Пройден
7	Обновление доски	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь – автор или администратор. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Нажать на три точки рядом с необходимой доской. 6. В выпадающем меню выбрать пункт “Изменить”. 7. Внести изменения в форму. 8. Нажать на кнопку “Обновить”	Доска должна успешно обновиться. Изменения должны отобразиться у всех пользователей, имеющих доступ к доске.	Пройден
8	Добавление колонки	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь автор или администратор проекта. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску, нажав на название ЛКМ. 6. Нажать “Новая колонка”.	На доске должна появится новая колонка у всех участников проекта, которые имеют доступ к доске, к которой добавлена новая задача.	Пройден

Продолжение таблицы Ж.2

№	Название	Шаги тестирования	Результат	Статус
9	Изменение названия колонки	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь автор или администратор проекта. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску, нажав на название ЛКМ. 6. Нажать на три вертикальные точки у любой колонки 7. Выбрать пункт “Переименовать” 8. Ввести новое название	Название колонки должно отобразиться у всех участников проекта, добавленных к доске, на которой есть данная колонка.	Пройден
10	Удаление колонки	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект, в котором пользователь автор или администратор проекта. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску, нажав на название ЛКМ. 6. Нажать на три вертикальные точки у любой колонки 7. Выбрать пункт “Удалить” 8. Подтвердить удаление	Колонка должна удалиться у всех пользователей, имеющих доступ к доске, с которой удаляется колонка.	Пройден

Продолжение таблицы Ж.2

№	Название	Шаги тестирования	Результат	Статус
11	Добавление задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Нажать на кнопку “Новая задача”. 7. Заполнить форму необходимыми данными. 8. Нажать на кнопку “Создать задачу”	На доске, в выбранной колонке, должна появиться задача у всех участников проекта, которые имеют доступ к доске.	Пройден
12	Изменение задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Нажать на задачу, автором которой пользователь является автором или любую задачу, если пользователь является автором или администратором проекта 7. Изменить данные в форме. 8. Нажать на кнопку “Обновить задачу”	Задача должна измениться у всех пользователей, которые имеют доступ к проекту и доске.	

Продолжение таблицы Ж.2

№	Название	Шаги тестирования	Результат	Статус
13	Удаление задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Нажать на задачу, автором которой пользователь является автором или любую задачу, если пользователь является автором или администратором проекта 7. Нажать на кнопку “Удалить задачу” 8. Подтвердить удаление	Задача должна исчезнуть из колонки у всех пользователей, которые имеют доступ к проекту и доске.	Пройден
14	Перемещение задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Зажать ЛКМ и удерживать на задачи, автором или исполнителем которой пользователь является автором, или любую задачу, если пользователь является автором или администратором проекта 7. Переместить курсор мыши с задачей на новую позицию 8. Отпустить ЛКМ	Карточка задачи должна переместиться в новую позицию у всех участников доски и проекта.	Пройден

Продолжение таблицы Ж.2

№	Название	Шаги тестирования	Результат	Статус
15	Смена статуса задачи	1. Войти в аккаунт. 2. Перейти на домашнюю страницу. 3. Выбрать любой проект. 4. Раскрыть боковое меню, нажав на “бутербродную” кнопку в правой верхней части экрана. 5. Выбрать любую доску. 6. Выбрать любую задачу, автором которой или исполнителем которой является пользователь, или любую задачу, если пользователь является автором или администратором проекта 7. Выбрать статус задачи 8. Нажать на кнопку “Сменить статус”	Карточка задачи должна сменить цвет на соответствующий.	Пройден