

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Онлайн-платформа для развертывания и
разработки программного обеспечения**

Студент _____ Забенько Вячеслав Борисович
(подпись, дата) (ФИО полностью)

Руководитель _____ Козлов А.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Забенько Вячеслав Борисович гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Онлайн-платформа для редактирования и развертывания программного обеспечения

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы д.т.н., профессор кафедры «МО и ПЭВМ» Козлов А. Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: онлайн-платформа для развертывания и разработки программного обеспечения, в виде IDE с возможностью компиляции и запуска.*

2) *Среда выполнения: операционная система MacOS*

Языковые средства разработки: Go, JavaScript, SQL, UML

Инструментальные средства: VS Code, Webstorm, PgAdmin 4, Visual Paradigm

Технологии разработки: микросервисная архитектура, ООП.

3) *Основные функции приложения:*

а) регистрация и авторизация пользователей;

б) создание проектов и их запуск;

в) редактирование кода с подсветкой и автодополнением;

г) совместное рецензирование;

д) просмотр и откат историй версий.

е) редактирование пользовательского профиля

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и требований к системе;
2) Проектирование программных средств;
3) Реализация программных средств;
4) Тестирование программных средств;
5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 125 листов, 45 рисунков, 17 таблиц, 24 использованных источника и 6 приложений.

ОНЛАЙН-ПЛАТФОРМА, WEB-ПРИЛОЖЕНИЕ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, REST, UML, POSTGRESQL, GO, JS, БАЗА ДАННЫХ, REACTJS, DOCKER, GIT, WEBSOCKET.

Объект исследования – онлайн-платформа для развертывания и разработки программного обеспечения.

Цель выпускной квалификационной работы – разработать онлайн-платформу для развертывания и разработки программного обеспечения, позволяющая писать, просматривать и исполнять код в браузере.

Технологии разработки – разработка клиентской части приложения проводилась на языке программирования JavaScript с использованием библиотеки ReactJS в среде разработки Webstorm 2024. Разработка серверной части приложения производилась на языке программирования Go в среде разработки VS Code. Разработка базы данных проводилась в СУБД PostgreSQL.

Среда исполнения – MacOS, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть онлайн-платформы для развертывания и разработки программного обеспечения. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08BP211.07 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Забенько В.Б.			Онлайн-платформа для развертывания и разработки программного обеспечения. Пояснительная записка	Лит.	Лист	Листов
Пров.		Козлов А.Ю.					4	125
Реценз.						Группа 21ВП1		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение	8
1 Анализ требований и постановка задачи.....	9
1.1 Анализ предметной области	9
1.2 Анализ программных аналогов	9
1.2.1 Анализ онлайн-редакторов кода	10
1.2.2 Анализ IDE и облачных IDE.....	11
1.2.3 Анализ CI/CD и DevOps-платформ.....	12
1.2.4 Результаты анализа.....	13
1.3 Постановка задачи на разработку	14
1.4 Анализ требований	17
1.4.1 Анализ функциональных требований.....	17
1.4.2 Анализ нефункциональных требований.....	19
1.5 Выбор инструментальных средств.....	21
1.6 Планирование проекта и оценка бюджета	22
2 Проектирование онлайн-платформы для развертывания и разработки программного обеспечения.....	26
2.1 Проектирование архитектуры онлайн-платформы	26
2.2 Проектирование клиентской части	28
2.3 Проектирование серверной части	36
2.3.1 Проектирование сервиса шлюза.....	36
2.3.2 Проектирование сервиса пользователей	37
2.3.3 Проектирование сервиса запуска кода	39
2.3.4 Проектирование сервиса авторизации.....	41
2.4 Проектирование базы данных	43
2.4.1 Разработка концептуальной модели базы данных	43

2.4.2	Разработка логической модели базы данных.....	45
2.4.3	Разработка физической модели базы данных	47
2.4.4	Развертывание базы данных в среде pgAdmin 4.....	48
2.5	Проектирование взаимодействия со средствами контейнеризации и системой контроля версий	49
3	Реализация онлайн-платформы для развертывания и разработки программного обеспечения.....	51
3.1	Реализация взаимодействия с базой данных.....	51
3.1.1	Разработка представлений и триггеров	51
3.1.2	Реализация серверных запросов к базе данных	53
3.1.3	Наполнение базы данных тестовыми данными	54
3.2	Реализация серверной части онлайн-платформы	55
3.2.1	Реализация авторизации приложения.....	55
3.2.2	Реализация оркестратора	57
3.2.3	Реализация контроля версий.....	58
3.3	Реализация клиентской части онлайн-платформы	60
3.3.1	Реализация визуального отображения программного кода.....	60
3.3.2	Реализация визуального отображения дерева проекта	62
3.3.3	Реализация проверки кода сторонними участниками.....	63
3.3.4	Реализация переключения по истории проектов	65
3.3.5	Реализация сборки компонентов.....	66
4	Тестирование онлайн-платформы для развертывания и разработки программного обеспечения.....	69
4.1	Функциональное тестирование клиентской части	69
4.1.1	Классы эквивалентности.....	69
4.1.2	Функциональная диаграмма и таблица решений	70
4.1.3	Диаграмма состояний переходов	72

4.2 Тестирование WebSocket-взаимодействия клиентской и серверной части.....	73
4.3 Модульное тестирование клиентской части	76
4.4 Оценка качества программного кода.....	78
4.4.1 Метрика Холстеда	78
4.4.2 Метрика Чепина.....	80
Заключение.....	82
Список использованных источников.....	84
Приложение А (справочное) Глоссарий.....	87
Приложение Б (обязательное) Листинг серверной части приложения	90
Приложение В (обязательное) Листинг клиентской части приложения.....	100
Приложение Г (рекомендуемое) Атрибуты сущностей базы данных	112
Приложение Д (рекомендуемое) Скрипт для создания объектов базы данных на языке SQL	118
Приложение Е (рекомендуемое) Листинг модульных тестов.....	121

Введение

Современные тенденции в области информационных технологий все чаще смещаются в сторону облачных решений и платформ, предоставляющих доступ к инструментам разработки из любой точки мира. Онлайн-платформы для программирования позволяют пользователям работать с исходным кодом, запускать проекты и развертывать приложения без необходимости локальной настройки окружения. Это значительно упрощает вход в профессию, повышает продуктивность и способствует командной разработке.

Разрабатываемая онлайн-платформа представляет собой web-приложение, обеспечивающее полноценную среду для написания, тестирования и запуска программного кода. Основными функциональными возможностями системы являются: регистрация и авторизация пользователей, создание и управление проектами, работа с файловой структурой, запуск кода в изолированных контейнерах и просмотр результатов выполнения. Платформа ориентирована как на индивидуальных разработчиков, так и на команды, нуждающиеся в быстром доступе к кроссплатформенной среде разработки.

Основной целью проекта является создание доступного, интуитивного и функционально насыщенного инструмента, который объединяет удобный интерфейс, надежную серверную архитектуру и гибкую систему хранения проектов. При реализации платформы используются современные технологии и принципы модульной архитектуры, обеспечивающие масштабируемость и отказоустойчивость системы.

1 Анализ требований и постановка задачи

1.1 Анализ предметной области

Онлайн-платформа для развертывания и разработки программного обеспечения – это облачно интегрированная среда разработки, объединяющая как веб-редактор кода, так и функциональное серверное окружение, включающее в себя средства сборки и развертывания. Данная система предоставляет пользователю доступ к инструментам программирования только при помощи браузера, без необходимости установки дополнительного программного обеспечения на локальном компьютере.

Для того, чтобы разработать программный продукт локально необходимы совместимость с операционной системой, установка требуемых библиотек и расширений. Онлайн-платформы помогают избавить пользователя от данной проблемы, требуя от пользователя только веб-браузер. Это делает разработку более доступной, как в повседневной жизни разработчиков, так и в образовательной среде.

Целевыми пользователями онлайн-платформы являются начинающие и профессиональные разработчики, учащиеся и преподаватели. Каждый пользователь получает редактор, терминал и изолированный runtime в рамках одной HTTP-сессии. Онлайн-платформа поддерживает основные языки программирования, а запуск кода происходит в изолированном пространстве контейнера.

1.2 Анализ программных аналогов

В рамках анализа предметной области необходимо разграничить классы систем, схожих по назначению с разрабатываемой платформой. Современные инструменты онлайн разработки делятся на несколько групп в зависимости от выполняемых задач и ограничений, накладываемых на работу с данными инструментами. Далее приведен обзор решений, относящийся к каждой из категорий, с выявлением сильных и слабых сторон.

1.2.1 Анализ онлайн-редакторов кода

К категории онлайн-редакторов кода относятся веб-сервисы, предлагающие браузерный редактор и встроенное выполнение кода, как правило они содержат только один исполняемый файл. Примеры – CodePen, JSFiddle, TryCode и другие. Это веб-инструменты для быстрой записи и тестирования кода прямо в браузере [1]. Главная страница приложения «TryCode» представлена на рисунке 1.

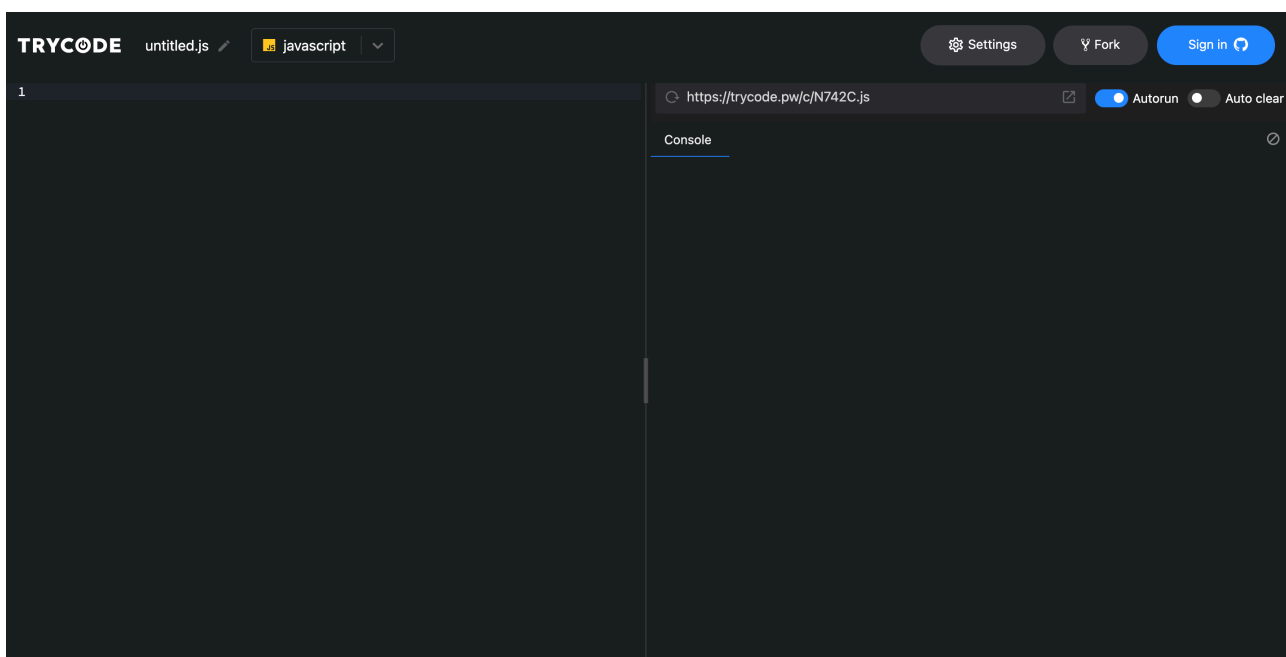


Рисунок 1 – Главная страница сайта «TryCode»

К положительным качествам данного сервиса можно отнести следующие возможности:

- простота использования и доступность из браузера;
- поддержка множества языков без установки зависимостей;
- отсутствие обязательной регистрации пользователя.

Несмотря на перечисленные достоинства, стоит отметить, что в сервисе присутствует ряд ограничений и недостатков, которые могут повлиять на выполнение определенных задач:

- ограниченные возможности настройки окружения;
- отсутствие полноценной изоляции и контроля инфраструктуры;

– нет возможности долгосрочной разработки продукта.

Таким образом, онлайн-редакторы кода предназначены для решения простых задач, в частности быстрое создание тестирования программ, но не подходят для управления жизненным циклом программного продукта.

1.2.2 Анализ IDE и облачных IDE

В данную категорию входят полноценные среды разработки, как локальные, так и облачные. Локальные IDE (например, Visual Studio, VS Code, WebStorm) работают на устройстве пользователя и требуют установленного софта и совместимую операционную систему. Облачные IDE используют удаленный сервер или контейнер в облаке. Например, GitPod позволяет разворачивать полноценные среды с VS Code-сервером в облаке с помощью YAML-файлов. GitHub Codespaces использует контейнеры с Ubuntu и предустановленными инструментами. Replit – это облачная IDE, где можно не только писать код, но и разворачивать приложение в облаке используя виртуальные машины с докер-контейнерами [2]. Страница проекта сервиса «Replit» представлена на рисунке 2.

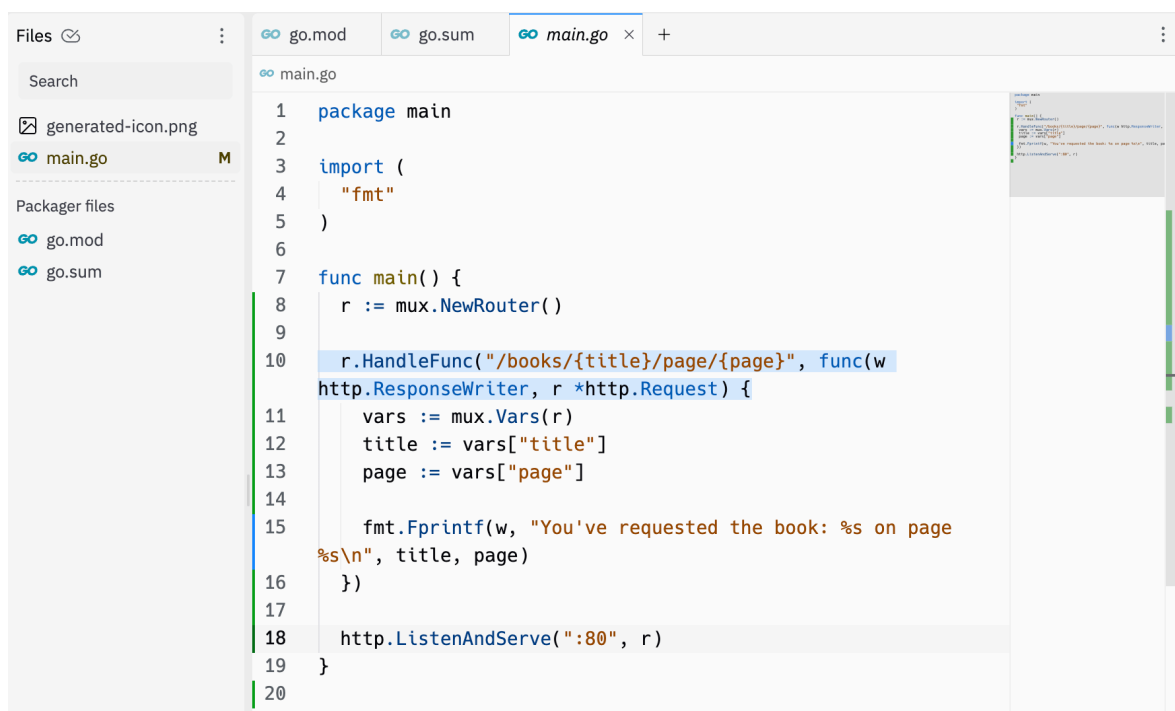


Рисунок 2 – Страница проекта сервиса «Replit»

Среди ключевых преимуществ облачных IDE по сравнению с другими решениями стоит выделить следующие пункты:

- поддержка полноценной разработки, включая терминал автодополнение, отладку;
- возможность запуска в изолированном окружении;
- доступ к хранению созданных проектов.

Однако, несмотря на эти преимущества сервисы имеют существенные недостатки:

- ограничение по ресурсам, используя бесплатный тариф;
- ограниченная гибкость настройки окружения;
- обязательное наличие аккаунта для использования.

Подводя промежуточный итог, можно отметить, что облачные IDE обеспечивают полноценную среду разработки в браузере, однако могут ограничивать гибкость настройки окружения и требовать денежные затраты на использование полноценного функционала.

1.2.3 Анализ CI/CD и DevOps-платформ

Платформы данной категории нацелены на автоматизацию жизненного цикла ПО: сборку, тестирование, доставку и развертывание. CI/CD (Continuous Integration/Continuous Delivery) – одна из основных составляющих DevOps [3]. Инструменты CI (Jenkins, GitLab CI, GitHub Actions) автоматически интегрируют изменения, собирают и тестируют код после каждого коммита. CD отвечает за развертывание готовых сборок приложения в различные окружения. Крупные DevOps-платформы (GitLab, GitHub, Azure DevOps) интегрируют в себе репозитории кода, таск-трекеры и CI-CD-конвейеры, обеспечивая поддержку от разработки до реализации. Благодаря встроенным пайплайнам команды получают мгновенную обратную связь о качестве изменений, а процесс поставки упрощает соблюдение требований информационной безопасности. В результате жизненный цикл выпуска

сокращается с недель до часов. На рисунке 3 изображена страница проекта на сайте «GitHub».

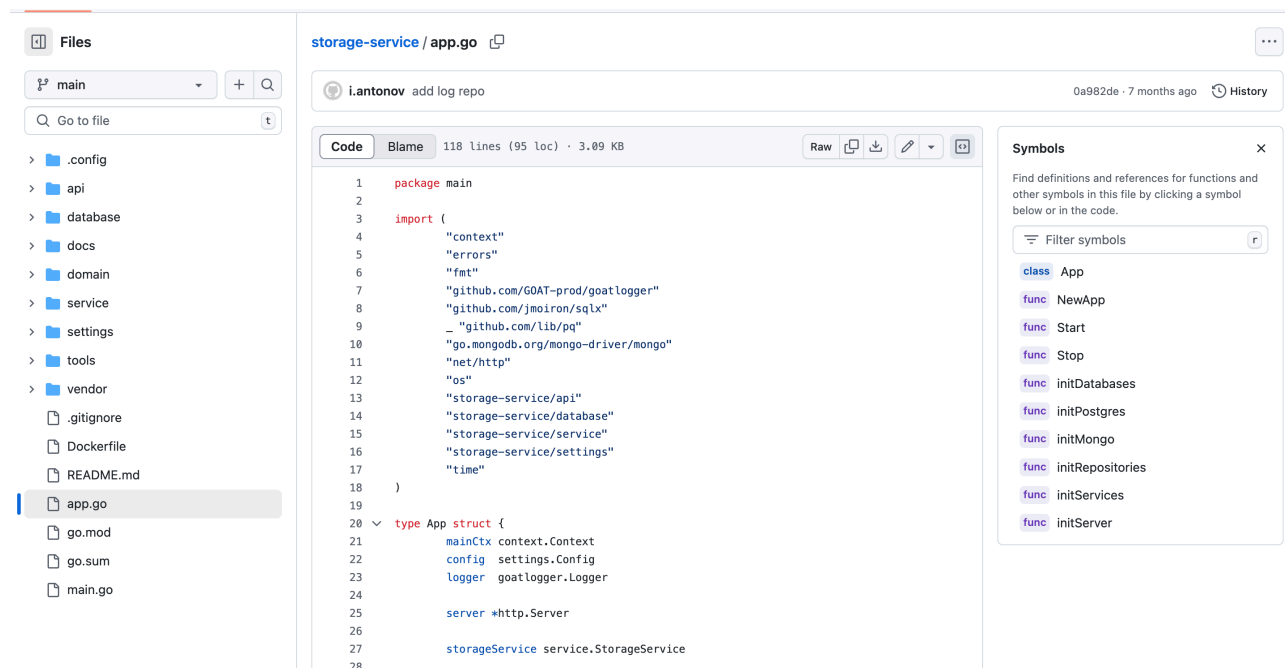


Рисунок 3 – Страница проекта сайта «GitHub»

Ключевые преимущества CI/CD и DevOps-платформ:

- удобство совместной работы;
- расширенная поддержка контейнеризации.

Однако при всех преимуществах, у платформы есть и недостатки:

- ограничения бесплатного тарифа;
- приложения не являются редакторами кода или платформами для разработки.

Таким образом, CI/CD платформы автоматизируют жизненный цикл ПО, но не предназначены для разработки программного кода в интерактивной среде.

1.2.4 Результаты анализа

Анализ существующих решений в области онлайн-разработки позволяет выделить ряд ключевых недостатков, оказывающих влияние на эффективность разработки программного обеспечения.

Проблемы локальной разработки:

- необходимость предварительной установки и настройки ПО;
- зависимость работоспособности кода от устройства пользователя;
- временные затраты на подготовку среды.

Проблемы онлайн-разработки:

- ограниченные возможности конфигурации среды исполнения;
- ограниченный функционал приложений в бесплатной версии;
- отсутствие сквозного конвейера разработки;
- использование отдельных CI/CD-платформ, приводящее к

усложнению управления жизненным циклом проекта.

Обозначив перечисленные проблемы, становится очевидной потребность в создании интегрированной онлайн-платформы, способной объединить в себе возможности редактирования, настройки и запуска программного проекта в едином веб-интерфейсе.

Предлагаемое решение должно синтезировать сильные стороны существующих решений: простоту онлайн-редакторов, полноту функционала IDE и автоматизацию DevOps-инструментов. В результате создается единая программная среда, минимизирующая затраты на каждом из этапов разработки ПО, повышающая согласованность действий команды и обеспечивающая непрерывную разработку программного обеспечения.

1.3 Постановка задачи на разработку

Разрабатываемая система представляет собой мощный инструмент для работы с программным кодом, на базе браузера пользователя, предоставляющий пользователям удобный способ создания, редактирования и управления проектами. Основная цель приложения – облегчить процесс разработки, организовать работу с проектами на разных языках программирования и предоставить пользователям простой и интуитивно понятный интерфейс.

Регистрация в системе является обязательным условием, так как для полноценной работы приложения с возможностью закрепления проектов за конкретными пользователями требуются их персональные данные.

Редактор кода поддерживает такие современные функции, как автодополнение и подсказки, что значительно упрощает написание кода, снижает вероятность ошибок и ускоряет процесс разработки. Эти возможности делают редактор универсальным инструментом, подходящим как для начинающих программистов, которым необходима помощь в написании кода, так и для профессионалов, которые ценят скорость и удобство работы.

Действующим лицом предметной области является пользователь, использующий функционал редактора кода. Описание категорий пользователей и подсистем приложения представлено в таблице 1.

Таблица 1 – Категории пользователей приложения

№ к. п.	Категория пользователей	Наименование приложения	Автоматизируемые бизнес-процессы	Классы информационных объектов
1	Пользователь	Онлайн-платформа для развертывания и редактирования программного обеспечения	регистрация в системе, авторизация, создание нового проекта, редактирование кода, сохранение проекта, выполнение кода, просмотр результатов выполнения, управление файлами в проекте. Загрузка существующего проекта, редактирование профиля, выход из системы	проекты, исходные файлы кода, результаты выполнения

Онлайн-платформа должна выполнять следующие информационные задачи:

- регистрация/авторизация пользователя;
- создание и управление проектами;
- редактирование исходного кода;

- выполнение и тестирование кода онлайн;
- управление файлами и папками в проектах;
- сохранение и загрузка проектов;
- просмотр результатов выполнения кода;
- редактирование профиля пользователя;
- мониторинг истории выполнения кода.

Сценарий выполнения информационных задач (процессов) представлен на рисунке 4 в виде диаграммы деятельности UML.

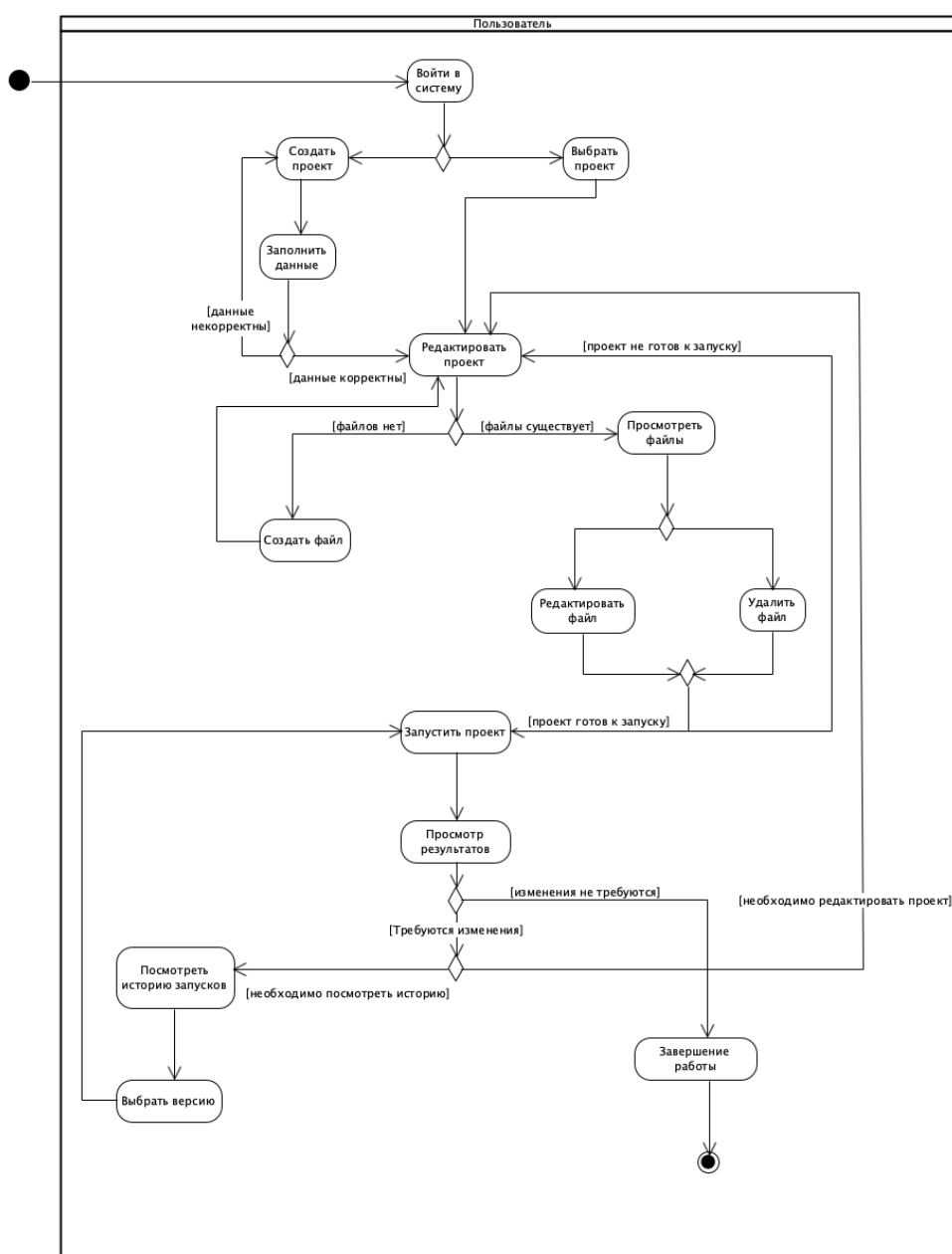


Рисунок 4 – Диаграмма деятельности работы с проектом

Полученные результаты являются исходными данными для проектирования и разработки онлайн-платформы.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

В ходе анализа установленных информационных задач были выделены функциональные требования и разработана диаграмма вариантов использования UML, представленная на рисунке 5.

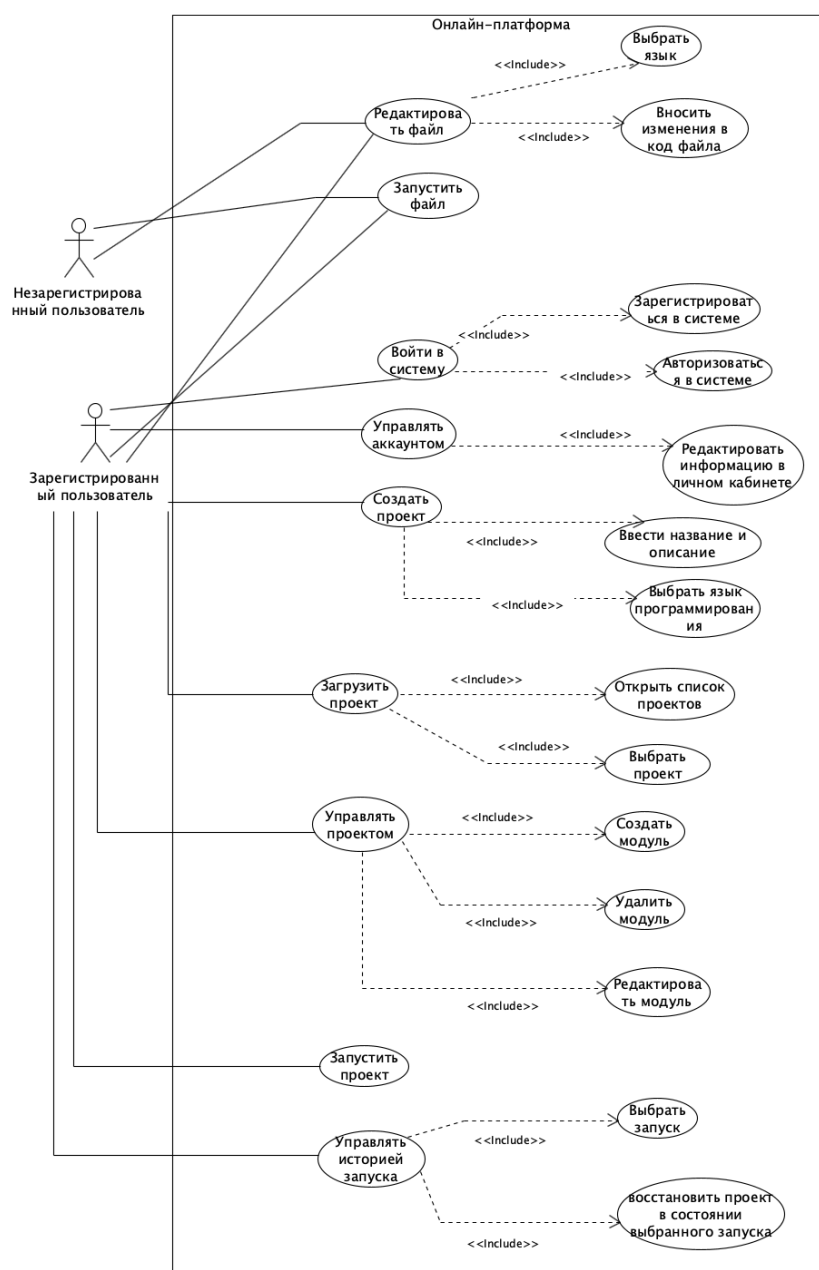


Рисунок 5 – Диаграмма вариантов использования приложения

В рамках разработки работы рассматривается пользователь, использующий онлайн-платформу для работы с программными проектами.

Кроме того, были разработаны сценарии наиболее важных функциональных требований, которые приведены ниже.

Описание сценария варианта использования «Редактировать файл» представлено на рисунке 6.

Наименование: Редактировать файл
ID: 1
Условие-триггер: Пользователь открыл файл, чтобы внести изменения.
Краткое описание: Пользователь вносит изменения в файл кода или текста.
Действующие лица: Незарегистрированный пользователь, Зарегистрированный пользователь.
Предусловие: Пользователь находится на странице с открытым файлом.
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает файл для редактирования. 2. Пользователь вносит изменения в содержимое файла. 3. Пользователь сохраняет изменения.
Постусловие: Изменения файла успешно сохранены.
Исключения: <ol style="list-style-type: none"> 1. Пользователь не выбрал файл. 2. Пользователь находится в режиме «CodeReview»
Бизнес-правила: <ol style="list-style-type: none"> 1. Пользователь находится в собственном проекте. 2. Пользовательская сессия не истекла.

Рисунок 6 – Сценарий варианта использования «Редактировать файл»

Описание сценария варианта использования «Создать проект» представлено на рисунке 7.

Наименование: Создать проект
ID: 2
Условие-триггер: Пользователь инициирует создание нового проекта.
Краткое описание: Пользователь создает новый проект в системе.
Предусловие: Пользователь авторизован в системе.
Основной поток: <ol style="list-style-type: none"> 1. Пользователь нажимает кнопку «Создать проект». 2. Пользователь вводит данные проекта и выбирает язык программирования. Проект сохраняется в системе.
Постусловие: Новый проект создан и доступен для дальнейшей работы.
Исключения: Поля не заполнены или содержат недопустимые символы.
Бизнес-правила: Пользователь авторизован и находится в активной сессии.

Рисунок 7 – Сценарий варианта использования «Редактировать файл»

Описание варианта использования «Войти в систему» представлено на рисунке 8.

Наименование: Войти в систему
ID: 3
Условие-триггер: Пользователь открывает страницу входа.
Краткое описание: Пользователь регистрируется или авторизуется в приложении для доступа к функционалу.
Действующие лица: Незарегистрированный пользователь.
Предусловие: Пользователь не зарегистрирован или не авторизован в системе.
Основной поток: <ol style="list-style-type: none"> 1. Пользователь открывает страницу входа в приложение. 2. Пользователь вводит данные для авторизации (логин и пароль). 3. Система проверяет корректность данных и предоставляет доступ к функционалу.
Постусловие: Пользователь авторизован, ему доступен полный функционал приложения.
Исключения: <ol style="list-style-type: none"> 1. Неверный логин или пароль. 2. Превышено число попыток входа.
Бизнес-правила: <ol style="list-style-type: none"> 1. Данные аутентификации обрабатываются безопасно.

Рисунок 8 – Сценарий варианта использования «Редактировать файл»

По итогам анализа функциональных требований были сформулированы варианты использования онлайн-платформы, с помощью которых у проекта появились четкие и однозначные рамки поведения, влияющие на дальнейший процесс проектирования и реализации онлайн-платформы. А для ряда ключевых вариантов использования были подготовлены их спецификации, позволяющие детально отобразить последовательность действий пользователя в системе.

1.4.2 Анализ нефункциональных требований

1.4.2.1 Анализ требований к серверной части

Выбор надежной серверной архитектуры с помощью анализа требований позволяет создать приложение с высокой пропускной способностью и стабильной работой. Важно отметить, что любой процесс, компьютер или канал связи не исключает возможность сбоя или отказа работы в его системе. Однако

необходимо выбрать такое решение, которое минимизирует временные затраты в случае возникновения нештатных ситуаций.

В ходе анализа были выделены следующие требования:

- высокая производительность и поддержка множества одновременных запросов;
- модульная архитектура с минимальной связностью компонентов;
- использование контейнеризации для изоляции пользовательских процессов;
- отказоустойчивость и сохранение состояния при кратковременных сбоях;
- автоматическое масштабирование в зависимости от нагрузки;
- шифрование всех соединений.

Данные требования обеспечат стабильную работу серверной части и защиту пользовательских данных. Кроме того, при выкладывании новых версий микросервисов должна применяться стратегия blue-green deployment, что позволяет обновлять компоненты без остановки сервиса и снижает риск простоя до нуля.

1.4.2.2 Анализ требований к интерфейсу

Интерфейс онлайн-платформы должен обеспечивать пользователям удобную и комфортную работу, близкую по функциональности к локальным IDE. Исследования показывают, что интуитивность и адаптивность интерфейса существенно влияют на продуктивность и удовлетворенность пользователей.

Для пользовательского интерфейса были выделены следующие требования:

- интуитивность и понятность навигации;
- адаптивность к различным размерам экранов и мобильным устройствам;
- кроссбраузерность;

- четкая визуальная структура элементов;
- высокая отзывчивость интерфейса без задержек.

Выполнение указанных требований позволит создать интерфейс, обеспечивающий высокую эффективность и удобство работы для всех категорий пользователей.

1.5 Выбор инструментальных средств

При создании браузерной платформы должен задействоваться набор технологий, который одинаково стабильно работает на macOS, Linux и Windows, это упрощает перенос компонентов распределенной системы между различными устройствами.

Клиентская часть будет разрабатываться на React 18 и стандарте JavaScript ES-2022, сборку и перезагрузку при любом изменении обеспечит Vite, что сокращает цикл внесения правок до нескольких секунд.

Серверная часть будет реализована на языке Go, который является популярным решением при разработке распределенных систем.

Взаимодействие между серверной и клиентской частью будет при использовании HTTP при работе с запросами и WebSocket во время работы с постоянным двусторонним потоком данным.

Для управления базами данных используется система управления базами данных PostgreSQL, потому что приложение при расширении может потребовать быструю работу с большими объемами данных, а также использование транзакций. Для администрирования и анализа планов запросов будет использоваться PgAdmin 4.

Контроль версий кода реализован с использованием системы управления версиями Git, что обеспечивает стабильность разработки и отслеживание изменений. Хранилище кода синхронизировано с удаленным репозиторием, что позволяет организовать командную работу и резервное хранение.

Для построения диаграмм используется инструмент Visual Paradigm, поддерживающий стандарт UML 2.0 и позволяющий создавать диаграммы

вариантов использования, классов, компонентов, последовательностей, а также архитектурные схемы развертывания.

Web-приложение оптимизируется под современные браузеры: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, включая их последние версии, что обеспечивает корректное отображение интерфейсов и стабильную работу на большинстве пользовательских устройств.

1.6 Планирование проекта и оценка бюджета

Для планирования проекта были продуманы и добавлены в Microsoft Project 2007 основные задачи, приведенные на рисунке 9.

		Название задачи	Длительность	Начало	Окончание	Предшественники
1		Начало реализации проекта	0 дней	Пн 07.04.25	Пн 07.04.25	
2		Анализ предметной области и требований	5 дней?	Пн 07.04.25	Пт 11.04.25	1
3		Анализ программных аналогов	1 день?	Пн 07.04.25	Пн 07.04.25	
4		Постановка задач на разработку	1 день?	Пн 07.04.25	Пн 07.04.25	
5		Анализ функциональных требований	2 дней	Вт 08.04.25	Ср 09.04.25	4
6		Анализ нефункциональных требований	1 день?	Чт 10.04.25	Чт 10.04.25	5
7		Выбор инструментальных средств	1 день?	Пт 11.04.25	Пт 11.04.25	6
8		Анализ предметной области и требований завершен	0 дней	Пт 11.04.25	Пт 11.04.25	7
9		Проектирование системы	7 дней	Пн 14.04.25	Вт 22.04.25	2
10		Проектирование сервисов	2 дней	Пн 14.04.25	Вт 15.04.25	
11		Проектирование клиентской части	2 дней	Ср 16.04.25	Чт 17.04.25	10
12		Проектирование базы данных	2 дней	Пт 18.04.25	Пн 21.04.25	11
13		Проектирование средств контейнеризации	1 день	Вт 22.04.25	Вт 22.04.25	12
14		Проектирование системы завершено	0 дней	Вт 22.04.25	Вт 22.04.25	13
15		Реализация системы	9 дней	Ср 23.04.25	Пн 05.05.25	9
16		Реализация базы данных	2 дней	Ср 23.04.25	Чт 24.04.25	
17		Реализация клиентской части	3 дней	Пт 25.04.25	Вт 29.04.25	16
18		Реализация серверной части	4 дней	Ср 30.04.25	Пн 05.05.25	17
19		Реализация системы завершена	0 дней	Вт 29.04.25	Вт 29.04.25	17
20		Тестирование системы и анализ кода	8 дней?	Вт 06.05.25	Чт 15.05.25	15
21		Функциональное тестирование	2 дней	Вт 06.05.25	Ср 07.05.25	
22		Структурное тестирование	2 дней	Чт 08.05.25	Пт 09.05.25	21
23		Модульное тестирование	3 дней	Пн 12.05.25	Ср 14.05.25	22
24		Анализ качества кода	1 день?	Чт 15.05.25	Чт 15.05.25	23
25		Завершение тестирования и анализа кода	0 дней	Чт 15.05.25	Чт 15.05.25	24
26		Завершение проекта	8 дней?	Пт 16.05.25	Вт 27.05.25	20
27		Составление документации	7 дней	Пт 16.05.25	Пн 26.05.25	
28		Внедрение проекта	1 день?	Вт 27.05.25	Вт 27.05.25	27
29		Проект завершен	0 дней	Вт 27.05.25	Вт 27.05.25	28
30		Конец проекта	0 дней	Вт 27.05.25	Вт 27.05.25	26;2

Рисунок 9 – Задачи для реализации проекта

Основные ресурсы, такие как работники со своей часовой ставкой и материальные вещи, необходимые для реализации проекта, приведены на рисунке 10.

	И	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполыз.	Начисление	Базовый календарь
1		Программист 1	Трудовой		П		100%	300,00 Р/ч	400,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
2		Программист 2	Трудовой		П		100%	200,00 Р/ч	300,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
3		Аналитик	Трудовой		А		100%	200,00 Р/ч	250,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
4		Тестировщик	Трудовой		Т		100%	180,00 Р/ч	230,00 Р/ч	0,00 Р	Пропорциональное	Стандартный

Рисунок 10 – Ресурсы на реализацию проекта

Для каждой задачи были распределены ресурсы, это распределение позволило оптимизировать календарный план проекта. Результат приведен на рисунке 11.

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Пн 07.04.25	Пн 07.04.25		
2	Анализ предметной области и требований	6 дней?	Пн 07.04.25	Пн 14.04.25	1	
3	Анализ программных аналогов	1 день?	Пн 07.04.25	Пн 07.04.25		Аналитик
4	Постановка задач на разработку	1 день?	Пн 07.04.25	Пн 07.04.25		Аналитик
5	Анализ функциональных требований	2 дней	Вт 08.04.25	Ср 09.04.25	4	Аналитик
6	Анализ нефункциональных требований	1 день?	Чт 10.04.25	Чт 10.04.25	5	Аналитик
7	Выбор инструментальных средств	2 дней?	Пт 11.04.25	Пн 14.04.25	6	Аналитик; Программист 1[50%]
8	Анализ предметной области и требований завершен	0 дней	Пн 14.04.25	Пн 14.04.25	7	
9	Проектирование системы	6,33 дней	Вт 15.04.25	Ср 23.04.25	2	
10	Проектирование сервисов	1,33 дней	Вт 15.04.25	Ср 16.04.25		Программист 1; Программист 2[50%]
11	Проектирование клиентской части	2 дней	Ср 16.04.25	Пт 18.04.25	10	Программист 1; Программист 2[50%]
12	Проектирование базы данных	2 дней	Пт 18.04.25	Вт 22.04.25	11	Программист 1; Программист 2[50%]
13	Проектирование средств контейнеризации	1 день	Вт 22.04.25	Ср 23.04.25	12	Программист 1; Программист 2[50%]
14	Проектирование системы завершено	0 дней	Ср 23.04.25	Ср 23.04.25	13	
15	Реализация системы	8,43 дней	Ср 23.04.25	Пн 05.05.25	9	
16	Реализация базы данных	1,43 дней	Ср 23.04.25	Чт 24.04.25		Программист 1[70%]; Программист 2
17	Реализация клиентской части	3 дней	Чт 24.04.25	Вт 29.04.25	16	Программист 1[70%]; Программист 2
18	Реализация серверной части	4 дней	Вт 29.04.25	Пн 05.05.25	17	Программист 1; Программист 2
19	Реализация системы завершена	0 дней	Вт 29.04.25	Вт 29.04.25	17	
20	Тестирование системы и анализ кода	7,67 дней?	Пн 05.05.25	Чт 15.05.25	15	
21	Функциональное тестирование	2 дней	Пн 05.05.25	Ср 07.05.25		Тестировщик
22	Структурное тестирование	2 дней	Ср 07.05.25	Пт 09.05.25	21	Тестировщик
23	Модульное тестирование	3 дней	Пт 09.05.25	Ср 14.05.25	22	Тестировщик
24	Анализ качества кода	0,67 дней?	Ср 14.05.25	Чт 15.05.25	23	Тестировщик; Программист 1[50%]
25	Завершение тестирования и анализа кода	0 дней	Чт 15.05.25	Чт 15.05.25	24	
26	Завершение проекта	5,67 дней?	Чт 15.05.25	Пт 23.05.25	20	
27	Составление документации	4,67 дней	Чт 15.05.25	Чт 22.05.25		Программист 1[50%]; Программист 2[50%]; Анап
28	Внедрение проекта	1 день?	Чт 22.05.25	Пт 23.05.25	27	Программист 1
29	Проект завершен	0 дней	Пт 23.05.25	Пт 23.05.25	28	
30	Конец проекта	0 дней	Пт 23.05.25	Пт 23.05.25	26;2	

Рисунок 11 – Ресурсы задач проекта

Для наиболее наглядного отображения плана и графика работ по проекту, с указанием календаря, длительности задач и затраченных ресурсов, была построена диаграмма Ганта, которая приведена на рисунке 12.

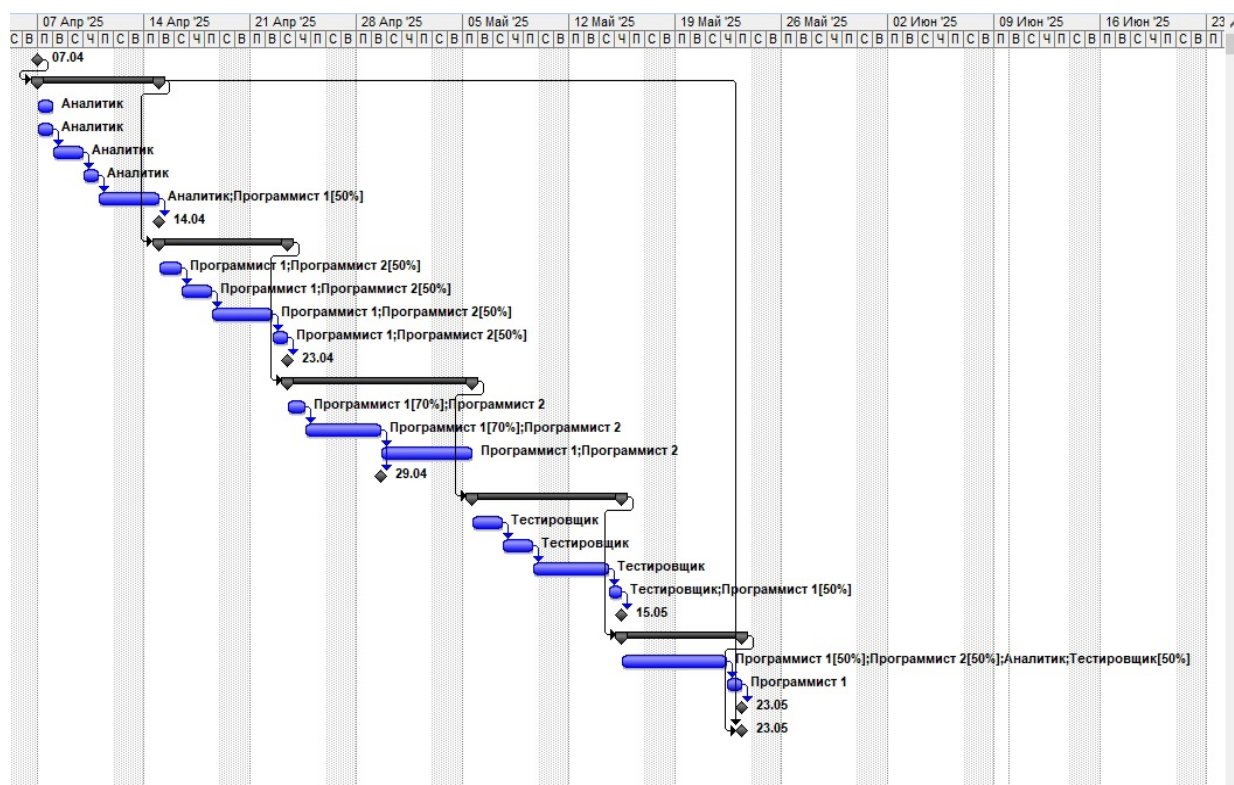


Рисунок 12 – Диаграмма Ганта

Общая статистика по проекту приведена на рисунке 13.

Статистика проекта для 'Диплом'

	Начало	Окончание
Текущее	Пн 07.04.25	Пт 23.05.25
Базовое	НД	НД
Фактическое	НД	НД
Отклонение	0д	0д

	Длительность	Трудозатраты	Затраты
Текущие	34,1д?	371,2ч	86 346,67 Р
Базовые	0д?	0ч	0,00 Р
Фактические	0д	0ч	0,00 Р
Оставшиеся	34,1д?	371,2ч	86 346,67 Р

Процент завершения

Длительность: 0% Трудозатраты: 0%

Закреть

Рисунок 13 – Статистика по проекту

Таким образом, из полученной статистики можно сделать вывод, что длительность реализации проекта составляет 34 дня, все поставленные задачи решены полностью, и проект завершен на 100%. Затраты составляют 86346

руб., что говорит об экономической выгоде покупки разработанного продукта относительно существующих на рынке наиболее популярных аналогов.

2 Проектирование онлайн-платформы для развертывания и разработки программного обеспечения

2.1 Проектирование архитектуры онлайн-платформы

В приложении используется архитектура на основе сервисов. Она представляет собой вариант архитектуры микросервисов и считается одной из наиболее прагматичных, главным образом из-за своей гибкости. Архитектура на основе сервисов является распределенной, но при этом у нее нет такого же уровня сложности и затратности, как, например, у микросервисов или архитектур, управляемых событиями, что делает ее оптимальным выбором для многих бизнес-приложений [4].

Так как архитектура хорошо оптимально подходит для построения распределенных систем с независимыми зонами ответственности. Из-за крупномодульности сервисов и их функциональной ограниченности каждая часть приложения вполне удачно помещается в отдельно развернутом сервисе. Каждый сервис в такой архитектуре охватывает конкретное направление:

- авторизация и регистрация;
- запуск кода;
- контроль версий;
- учет пользовательских проектов.

Поэтому извлечение конкретной функциональности в отдельный модуль приложения упрощает внесение изменений в охватываемый им набор задач.

В качестве основного архитектурного подхода к построению коммуникации между компонентами приложения и клиентского приложения используется REST API. Главная особенность REST API – обмен сообщениями без сохранения состояния. Каждое сообщение самодостаточное и содержит всю информацию, необходимую для его обработки. REST хорошо сочетается с сервисной архитектурой, так как предоставляет сервисам независимый, стандартизированный HTTP-интерфейс, что делает их слабо связанными и облегчает масштабирование системы [5].

Также в качестве еще одного способа общения между сервисами были использованы WebSocket-соединения. WebSocket – технология двунаправленной асинхронной симметричной связи между браузером и сервером, основанного на протоколе HTTP [6].

На рисунке 14 приведена диаграмма компонентов, из которых состоит серверная часть приложения.

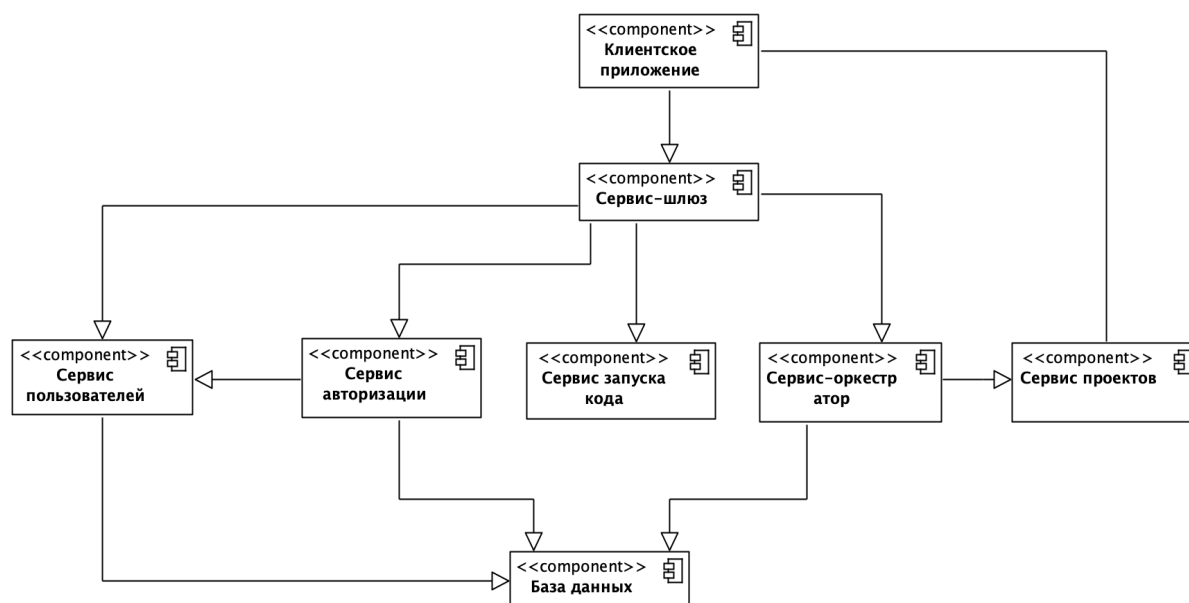


Рисунок 14 – Диаграмма компонентов серверной части

Основным компонентом системы является клиентское приложение, с помощью которого конечные пользователи взаимодействуют с онлайн-платформой, отправляя HTTP-запросы и обрабатывая ответы.

Сервис-шлюз будет являться основной точкой входа в систему, маршрутизируя запросы клиента к соответствующим сервисам, таким как сервисы авторизации, пользователей или оркестратора.

Сервис авторизации отвечает за регистрацию и аутентификацию пользователей и управление токенами доступа для обеспечения безопасности системы.

Сервис пользователей управляет информацией о пользователях, включая их профили и проекты, и предоставляет соответствующий API для работы с данными пользователей.

Сервис запуска кода является решением для тех пользователей, которым необходимо в кратчайшие сроки запустить собственный код в рамках одного файла, для использования этого сервиса регистрация пользователей не требуется.

Сервис-оркестратор необходим для запуска пользовательских проектов, он активирует docker-контейнер с экземпляром сервиса проектов и контролирует работу этого контейнера.

Сервис проектов подключается через Websocket к клиентскому приложению и в бесперебойном режиме благодаря двунаправленному соединению выполняет команды пользователя. Данный сервис также отвечает за хранение и контроль версий пользовательских проектов.

База данных используется всеми сервисами для хранения информации о пользователях, проектах, версиях и отзывах пользователей.

2.2 Проектирование клиентской части

Пользовательский интерфейс клиентского приложения должен обеспечивать привычное и удобное представление информации, простое и эффективное выполнение основных функций пользователей.

Интерфейс приложения онлайн редактора кода должен состоять из следующих компонентов:

- форма регистрации и авторизации;
- личный кабинет пользователя;
- страница существующих проектов;
- главное окно редактора кода;
- форма редактирования профиля;
- форма добавления нового проекта;
- форма редактирования существующего проекта;
- форма добавления файлов и папок.

На рисунке 15 представлен интерфейс формы авторизации.

The image shows a login form titled "Вход в аккаунт". It contains two input fields: "Email" with the placeholder "Введите ваш email" and "Пароль" with the placeholder "Введите ваш пароль". Below the password field is a green button labeled "Войти". At the bottom, there is a link "Новый пользователь? Зарегистрируйтесь" and a link "Забыли пароль?". Red lines with numbers 1 through 4 point to the email field, password field, login button, and registration link respectively.

Рисунок 15 – Форма для авторизации пользователя

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – поле для ввода электронной почты;
- 2 – поле для ввода пароля;
- 3 – кнопка для перехода на главное окно;
- 4 – кнопка для перехода на форму для регистрации.

На рисунке 16 представлен интерфейс формы регистрации.

The image shows a registration form titled "Регистрация аккаунта". It contains three input fields: "Email" with the placeholder "Введите ваш email", "Пароль" with the placeholder "Введите ваш пароль", and "Подтвердите пароль" with the placeholder "Повторите ваш пароль". Below the third field is a green button labeled "Зарегистрироваться". At the bottom, there is a link "Уже есть аккаунт? Войти". Red lines with numbers 1 through 5 point to the email field, password field, confirmation password field, registration button, and login link respectively.

Рисунок 16 – Форма для регистрации пользователя

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – поле для ввода электронной почты;
- 2 – поле для ввода пароля;
- 3 – поле для ввода подтверждения пароля;
- 4 – кнопка для перехода на форму для авторизации пользователя.

Данные формы позволяют пользователям легко и быстро получить доступ к своим проектам. Авторизация и регистрация продуманы так, чтобы пользователь мог максимально быстро приступить к работе.

На рисунке 17 представлен интерфейс личного кабинета пользователя.

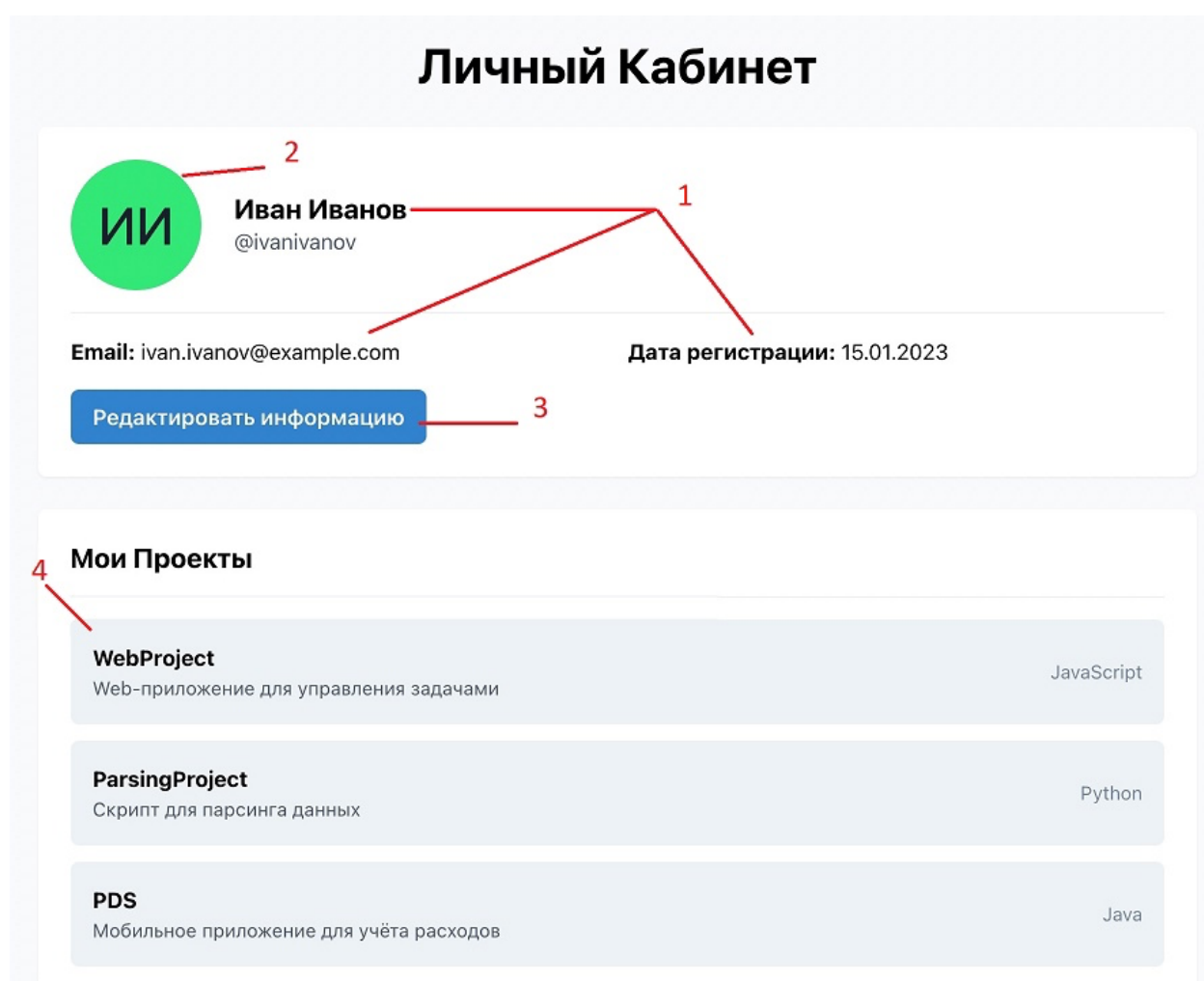


Рисунок 17 – Личный кабинет пользователя

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – поля с информацией о пользователе;
- 2 – аватар пользователя;

3 – кнопка для изменения данных о пользователе в приложении;

4 – проекты пользователя.

Личный кабинет предоставляет пользователю доступ ко всей важной персональной информации, позволяя легко управлять профилем и отслеживать свои проекты. Дизайн личного кабинета способствует комфортной работе и навигации.

На рисунке 18 представлен интерфейс формы редактирования профиля.

The image shows a web form titled "Редактировать информацию" (Edit information) with a close button (7) in the top right corner. The form contains four input fields: "Имя" (Name) with value "Иван Иванов" (1), "Логин" (Login) with value "ivanivanov" (2), "Email" with value "ivan.ivanov@example.com" (3), and "Аватар" (Avatar) with placeholder text "Введите URL аватара" (4). At the bottom, there are two buttons: "Отменить" (Cancel) (5) and "Сохранить" (Save) (6).

Рисунок 18 – Форма редактирования профиля

На рисунке цифрами отмечены следующие компоненты формы:

1 – поле ввода имени;

2 – поле ввода логина;

3 – поле ввода электронной почты;

4 – поле ввода ссылки на аватар;

5 – кнопка отмены изменений;

6 – кнопка сохранения изменений;

7 – кнопка закрытия окна.

На рисунке 19 представлен интерфейс главного окна онлайн-платформы для работы с программным проектом.

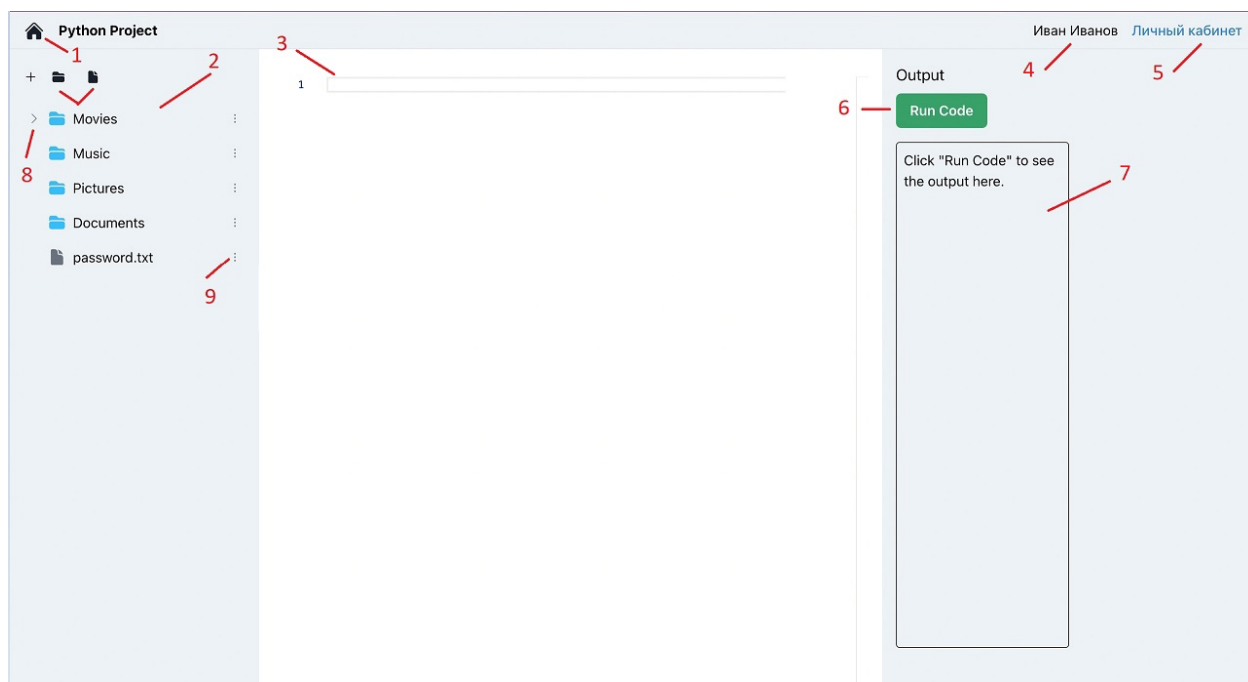


Рисунок 19 – Главное окно приложения

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – кнопка перехода на вкладку проектов;
- 2 – окно, отображающее файловую структуру проекта;
- 3 – окно для редактирования содержимого файла;
- 4 – пользовательская информация;
- 5 – кнопка перехода в личный кабинет;
- 6 – кнопка запуска проекта;
- 7 – окно вывода результата запуска проекта;
- 8 – кнопка развертывания каталога;
- 9 – кнопка управления файлом.

Главное окно онлайн-платформы для развертывания и редактирования программного обеспечения объединяет все необходимое для удобного программирования: редактор с подсветкой синтаксиса, файловое дерево и терминал для ввода команд и информации, а также для вывода результатов выполнения программы.

На рисунке 20 представлен интерфейс вкладки «Проекты».

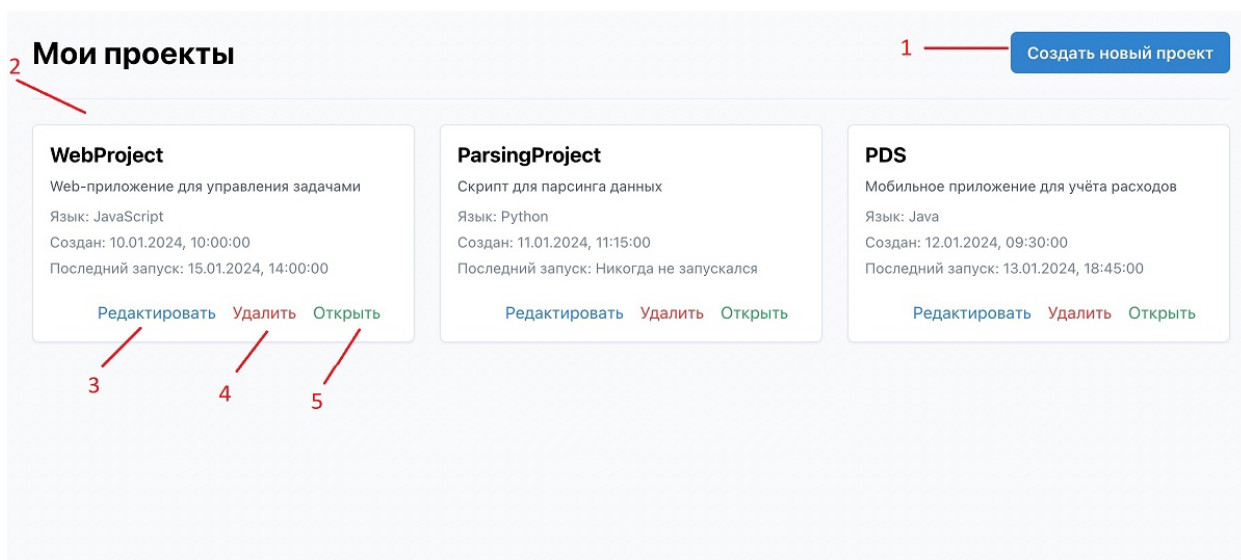


Рисунок 20 – Вкладка «Проекты»

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – кнопка создания нового проекта;
- 2 – таблица существующих проектов;
- 3 – кнопка редактирования информации о проекте;
- 4 – кнопка удаления проекта;
- 5 – кнопка запуска проекта в режиме редактирования.

На рисунке 21 представлен интерфейс вкладки «Редактирование проекта».

Рисунок 21 – Вкладка «Редактирование проекта»

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – кнопка закрытия окна редактирования;
- 2 – поле редактирования названия проекта;
- 3 – поле редактирования описания проекта;
- 4 – кнопка отмены изменений редактирования;
- 5 – кнопка сохранения изменений.

Интерфейс редактирования данных проекта устроен так, что нужные изменения вносятся за несколько кликов, без дополнительных переходов и лишних операций.

На рисунке 22 представлен интерфейс вкладки «Создать проект».

The image shows a web form titled "Создать новый проект" (Create new project). At the top right is a close button marked with a red line and the number 1. The form contains three input fields: "Название проекта" (Project name) marked with a red line and 2, "Описание" (Description) marked with a red line and 3, and "Язык программирования" (Programming language) marked with a red line and 4. The language field is a dropdown menu with the text "Выберите язык" (Select language) and a downward arrow. At the bottom are two buttons: "Отменить" (Cancel) marked with a red line and 5, and "Создать" (Create) marked with a red line and 6.

Рисунок 22 – Вкладка «Создать проект»

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – кнопка закрытия окна;
- 2 – поле ввода названия проекта;
- 3 – поле ввода описания проекта;
- 4 – выпадающий список языков программирования;
- 5 – кнопка отмены редактирования;
- 6 – кнопка создания нового проекта.

Форма позволяет создать проект за несколько шагов: пользователь вводит название, при желании добавляет краткое описание, а затем выбирает язык. Все поля находятся в одном окне, поэтому старт занимает меньше минуты.

На рисунке 23 представлен интерфейс вкладки «Новая папка».

Рисунок 23 – Вкладка «Добавить папку»

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – кнопка закрытия окна;
- 2 – поле для ввода названия папки;
- 3 – кнопка добавления папки;
- 4 – кнопка отмены изменений;

На рисунке 24 представлен интерфейс формы вкладки «Новый файл».

Рисунок 24 – Форма добавления файла

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – кнопка закрытия окна;
- 2 – поле для ввода названия папки;

3 – кнопка добавления файла;

4 – кнопка отмены изменений.

Эти формы предоставляют пользователю возможность быстро и удобно организовать рабочее пространство проекта, добавляя и структурируя файлы и папки.

Макеты диалоговых окон созданы в среде разработки Webstorm 2024, реализация осуществляется на языке HTML, CSS и JavaScript.

2.3 Проектирование серверной части

2.3.1 Проектирование сервиса шлюза

Компонент Шлюз – сервис, который выступает в качестве точки взаимодействия между внешними клиентами и внутренними микросервисами. Его основная задача – маршрутизировать запросы от клиента к соответствующим микросервисам.

API Gateway представляет собой центральный компонент системы, который принимает запросы от клиентов и направляет их в соответствующие микросервисы. Он предоставляет единый интерфейс для взаимодействия с различными сервисами: пользователей, запусков, аутентификации и оркестратора. Каждый сервис имеет свои сущности, которые используются для обработки данных.

Логика REST запросов к микросервисам осуществляется через классы с соответствующим названием:

- UserHandler для доступа к сервису пользователей;
- RunnerHandler для доступа к сервису запусков;
- AuthHandler для доступа к сервису авторизации;
- OrchestartorHandler для доступа к сервису-оркестратору.

Каждый из этих классов использует внутри себя собственный ServiceClient, в котором находится логика обработки и формирования запроса к

сервисам. Подробное описание классов и их взаимодействие описано на диаграмме классов на рисунке 25.

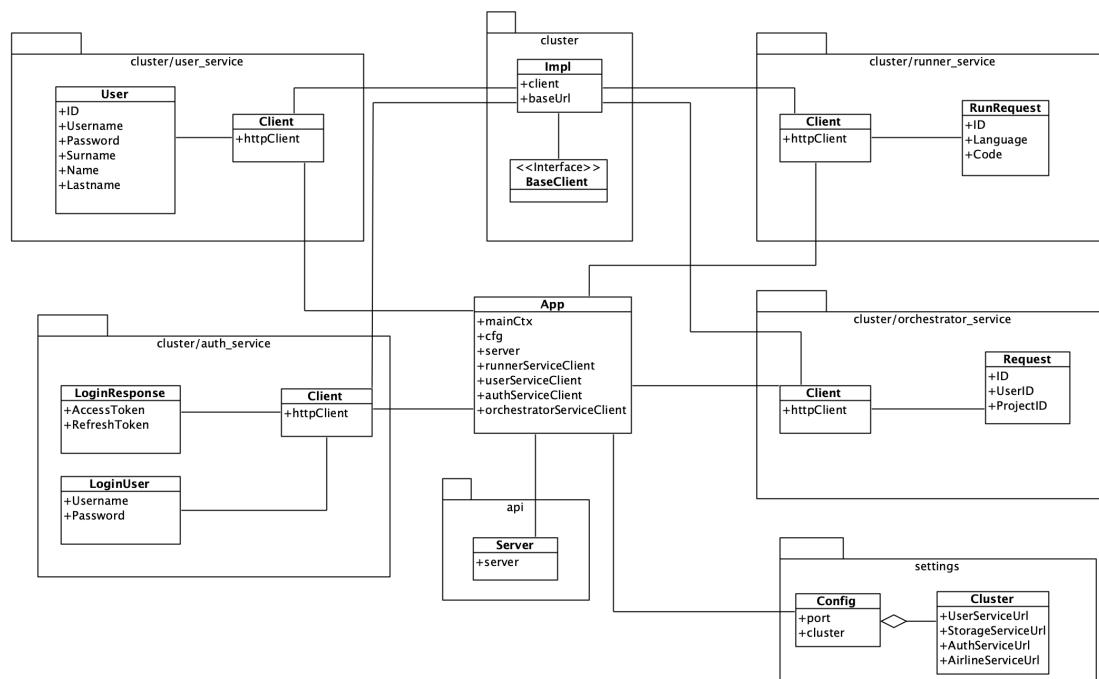


Рисунок 25 – Диаграмма классов шлюза

В классе Server будет описываться настройка REST-запросов к приложению, там же будут находиться и middleware с проверкой авторизации пользователя.

2.3.2 Проектирование сервиса пользователей

Сервис пользователей является одним из ключевых компонентов архитектуры онлайн-платформы, предназначенным для управления учетными записями пользователей. Помимо операций с аккаунтом сервис обеспечивает хранение информации о проектах. Таким образом, сервис будет выполнять роль ядра системы, ответственным за работу с основными сущностями базы данных.

Сервис пользователей приложения для развертывания и редактирования программного обеспечения спроектирован по принципу чистой архитектуры с четким разделением ответственности между слоями.

Модельный слой описывает структуры для хранения информации о пользователях, проектах и членстве в проектах. Например, сущность User.

Сервисный слой включает бизнес-логику по созданию и управлению пользователями и проектами, валидацию, обработку бизнес-ошибок, логику проверки прав доступа, а также преобразование между внутренними моделями и DTO.

Слой передачи данных описывает паттерн с использованием DTO. DTO (Data Transfer Object) – объект переноса данных применяется для переноса данных между процессами в целях уменьшения количества вызовов объекта [7]. Сервис описывает структуры, которые используются для обмена данными между слоями приложения и на уровне API. Например, при создании или обновлении пользовательских данных.

Транспортный слой реализует REST API для внешнего доступа, маршрутизацию и обработку HTTP-запросов для управления пользователями и проектами.

Более подробное содержание классов описывает диаграмма классов на рисунке 26.

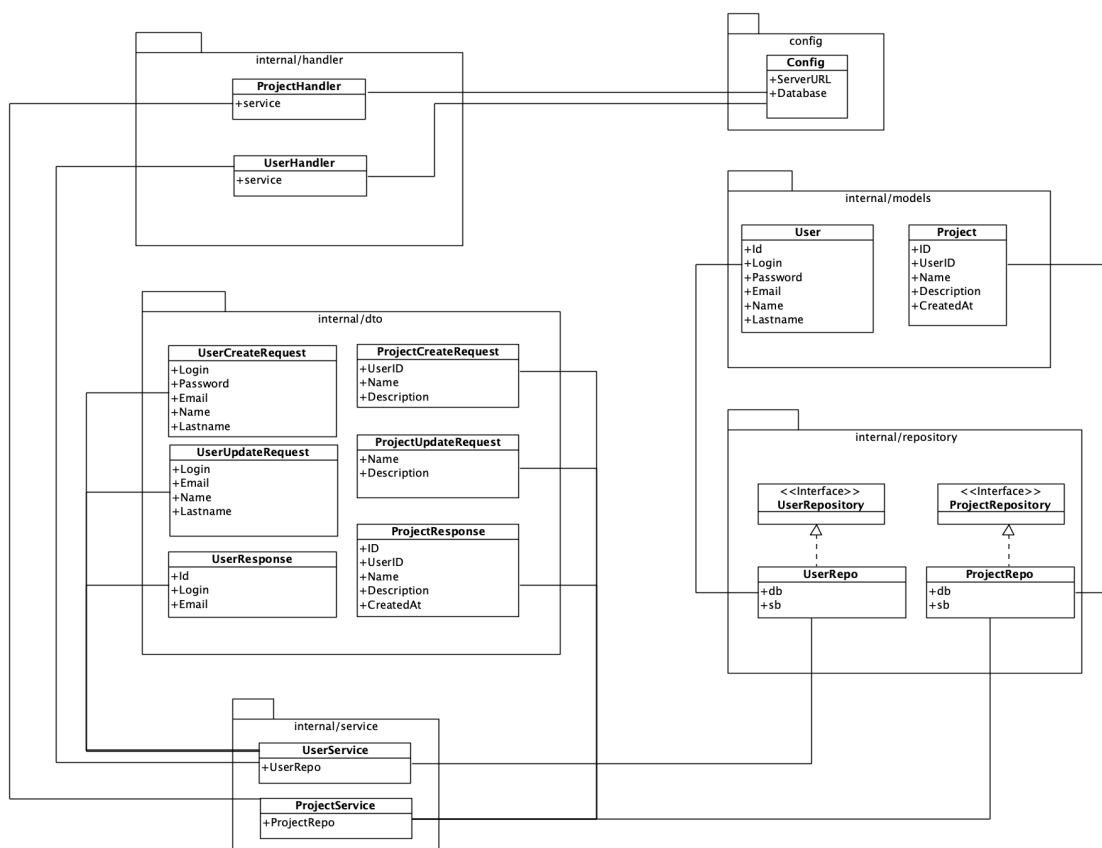


Рисунок 26 – Диаграмма классов сервиса пользователей

Для взаимодействия с сервисом пользователей были разработаны API методы, описание которых приведено в таблице 2.

Таблица 2 – Методы API микросервиса User-Service

URL запроса	Http метод	Описание	Query параметры запроса
/user	GET	Получение информации о пользователе с помощью идентификатора	id – идентификатор пользователя
/user/by-username	GET	Получение информации о пользователе с помощью логина	username – имя пользователя
/users	GET	Получение информации обо всех зарегистрированных пользователях	-
/user	POST	Добавление нового пользователя	-
/user	PUT	Изменение информации пользователя по идентификатору	id – идентификатор пользователя
/user	DELETE	Удаление записи об аккаунте пользователя	id – идентификатор пользователя
/project	GET	Получение информации о проекте по идентификатору	id – идентификатор проекта
/projects	GET	Получение всех проектов пользователя	id – идентификатор проекта
/project	POST	Добавление нового проекта	-
/project	PUT	Изменение информации о проекте по идентификатору	id – идентификатор пользователя
/project	DELETE	Удаление проекта по идентификатору	id – идентификатор пользователя

Методы API сервиса User-Service предоставляют возможности для полного управления учетными записями пользователей и проектами, включая создание, получение, обновление и удаление данных.

2.3.3 Проектирование сервиса запуска кода

В рамках разрабатываемой онлайн-платформы необходим отдельный микросервис, обеспечивающий возможность удаленного выполнения программного кода на различных языках без использования регистрации.

Разрабатываемый сервис должен получать HTTP-запрос с текстом исходного кода и указанием языка программирования, компилирует или интерпретирует его, в зависимости от языка программирования, а затем возвращает клиенту результат выполнения.

Архитектура сервиса использует принципы чистой архитектуры. Такая структура обеспечивает разделение бизнес-логики от технических деталей реализации и делает систему легко расширяемой.

Слой моделей содержит структуры данных, используемые для передачи информации между слоями и через API.

Слой бизнес-логики инкапсулирует реализацию запуска кода для конкретных языков программирования. Каждый язык реализуется как отдельный тип, соответствующий интерфейсу Runner.

Серверный слой инициализирует HTTP-сервер и связывает его с обработчиком.

Слой обработки HTTP-запросов отвечает за прием, валидацию и маршрутизацию запросов от клиентов.

Основу гибкости и расширяемости сервиса составляет паттерн «Фабрика», применяемый для создания нужного объекта-исполнителя на основе входного параметра – языка программирования.

Более подробное содержание классов описывает диаграмма классов на рисунке 27.

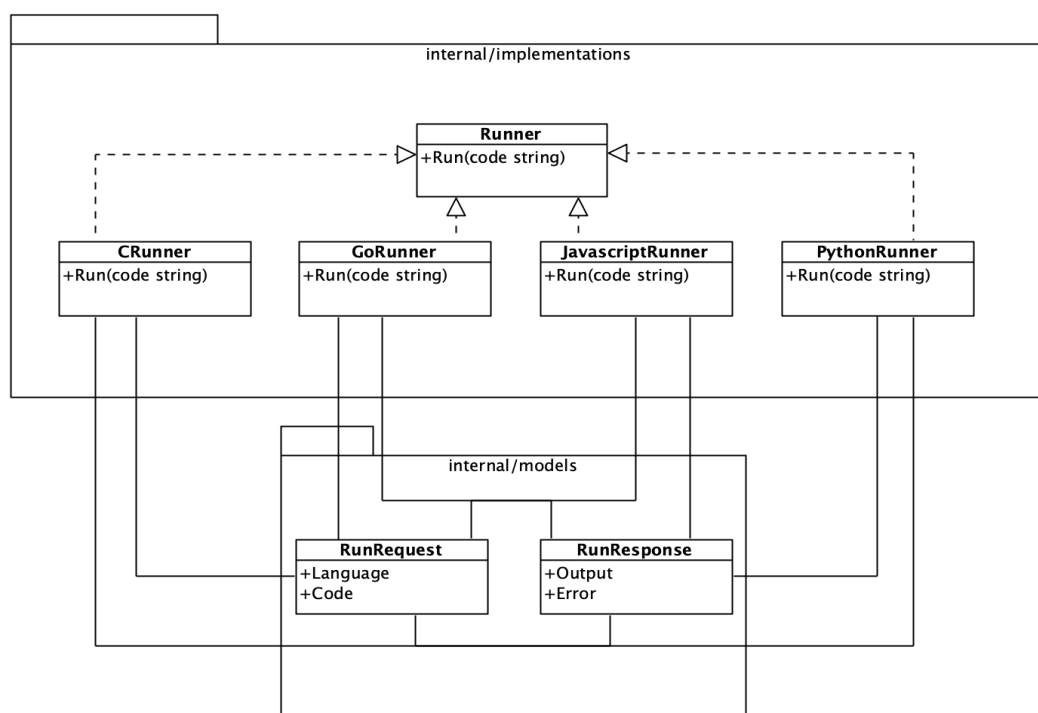


Рисунок 27 – Диаграмма классов сервиса запуска кода

Для взаимодействия с сервисом запуска кода были разработаны API методы, описание которых приведено в таблице 3.

Таблица 3 – Методы API микросервиса Runner-Service

URL запроса	Http метод	Описание	Query параметры запроса
/run	POST	Выполнение кода на выбранном языке программирования	-
/languages	GET	Возвращает список доступных для запуска языков программирования	-

Методы API сервиса позволяют клиентскому приложению без авторизации использовать функционал сервиса для выполнения требуемых задач.

2.3.4 Проектирование сервиса авторизации

Сервис авторизации служит для управления процессами регистрации и аутентификации пользователей, а также для обработки операций с ключами доступа и обновления. Основная задача сервиса – обеспечить безопасный и эффективный доступ к системе, реализуя следующие функции:

- аутентификация пользователей на основе логина и пароля;
- регистрация новых пользователей с хешированием паролей;
- управление токенами доступа (JWT) и их обновление;
- предоставление безопасного интерфейса для интеграции с другими сервисами.

Архитектура сервиса авторизации состоит из нескольких компонентов, взаимодействующих между собой для выполнения основных функций.

Слой бизнес-логики отвечает за аутентификацию пользователей. Этот слой обрабатывает логин, логアウト и обновление токенов. Он использует JWT для генерации токенов и взаимодействует с хранилищем рефреш-токенов. JWT (JSON Web Tokens) выбран благодаря его стандартизованности, поддержке статус-лесной аутентификации (что критично для масштабируемости) и

возможности встраивания пользовательских данных (claims) в токен, что уменьшает нагрузку на базу данных [8].

Слой токенов отвечает за генерацию токенов доступа и обновления на основе данных о пользователях. Он управляет секретными ключами для подписи токенов и их проверкой.

Слой хранилища рефреш-токенов использует Redis для хранения, извлечения и удаления рефреш-токенов. Это обеспечивает масштабируемость и быстрый доступ к токенам.

Интеграционный слой управляет всеми зависимостями сервиса авторизации. Он связывает между собой компоненты (AuthService, RegistrationService, JwtService, и RefreshTokenRepository) и запускает данный сервис.

Более подробное содержание классов описывает диаграмма классов на рисунке 28.

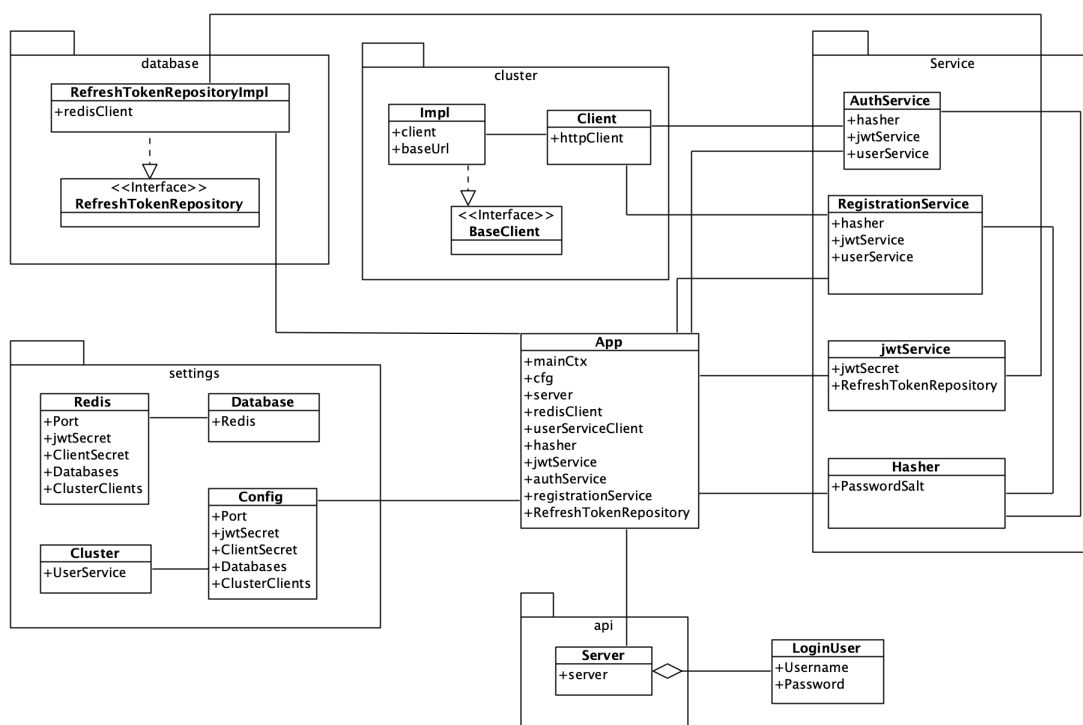


Рисунок 28 – Диаграмма классов сервиса авторизации

Для взаимодействия с сервисом авторизации были разработаны API методы, описание которых приведено в таблице 4.

Таблица 4 – Методы API микросервиса Auth-Service

URL запроса	Http метод	Описание	Query параметры запроса
/sign-up	POST	Регистрация нового пользователя	-
/login	POST	Авторизация пользователя	-
/logout	POST	Выход из системы пользователя	-
/refresh-token	POST	Обновление пользовательского токена	-

Методы API сервиса Auth-Service предоставляют базовый функционал для управления аутентификацией пользователей, включая регистрацию, вход, выход и обновление токенов.

2.4 Проектирование базы данных

2.4.1 Разработка концептуальной модели базы данных

Концептуальная модель – это модель, отображающая данные и связи предметной области с учетом их семантического содержания, независимо от представления в компьютере. [9]

Для разработанной концептуальной модели базы данных было создано табличное описание сущностей, представленное в таблице 5.

Таблица 5 – Табличное описание сущностей для концептуальной модели базы данных

№ п/п	Наименование	Описание	Категории
1	Пользователь	Содержит информацию о пользователях	Стержневая
2	Проект	Содержит сведения о проектах пользователя	Характеристическая
3	Участник проекта	Содержит пользователей с доступом к проекту	Характеристическая
4	Версия проекта	Содержит версии проекта	Характеристическая
5	Отзыв	Содержит отзыв на проект	Характеристическая
6	Комментарий	Содержит комментарии отзыва на проект	Характеристическая

На основе анализа предметной области и результатов моделирования бизнес-процессов была разработана концептуальная модель базы данных, представленная на рисунке 29.

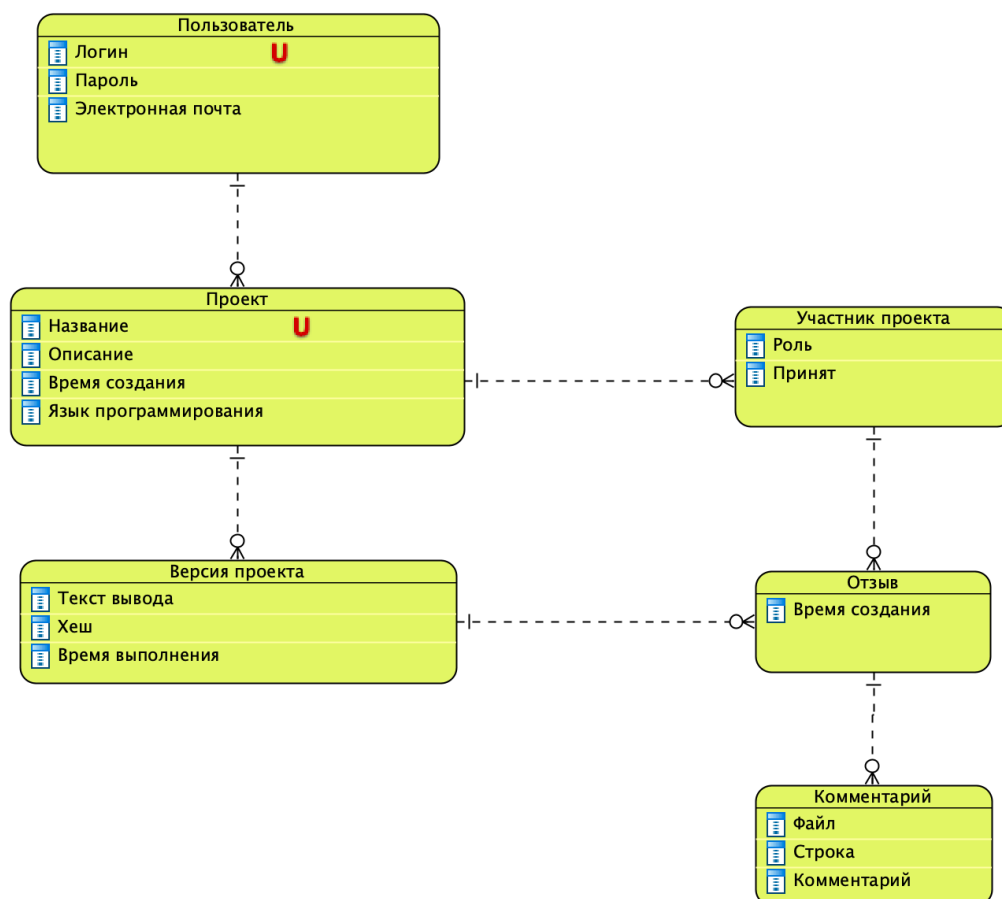


Рисунок 29 – Графическое представление концептуальной модели

Для разработанной концептуальной модели базы данных также было составлено табличное описание связей между сущностями, представленное в таблице 6.

Таблица 6 – Спецификации связей между сущностями для концептуальной модели базы данных

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность
1	Пользователь	Проект	Пользователь создает проекты / принадлежит пользователю	1:M	Обязательная
2	Проект	Версия проекта	Созданному проекту принадлежат версии / Версия относится к проекту	1:M	Обязательная

Продолжение таблицы 6

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность
3	Проект	Участник проекта	Проект содержит участников / Участники находятся в проекте	1:M	Необязательная (проектом необязательно делиться)
4	Участник проекта	Отзыв	Участник проекта может оставлять отзыв / Отзыв закрепляется за участником	1:M	Необязательная (участники могут не оставлять отзыв)
5	Отзыв	Комментарий	У отзыва есть комментарий / комментарий находится в отзыве	1:M	Обязательная
6	Версия проекта	Отзыв	Версия проекта содержит отзывы / Отзыв находится в версии проекта	M:1	Обязательная

Заданные связи полноценно отражают все взаимодействующие объекты участвующие в жизненном цикле проекта: от создания пользователем до коллективной разработки.

2.4.2 Разработка логической модели базы данных

При разработке приложения учета и анализа физической активности была выделена база данных: «Редактор кода».

Спецификация отношений логической модели базы данных «Редактор кода» представлена в таблице 7.

Таблица 7 – Спецификация отношений логической модели базы данных «Редактор кода»

№ п/п	Наименование	Описание	Категории
1	Пользователь	Содержит информацию о пользователях	Стержневая
2	Проект	Содержит сведения о проектах пользователя	Характеристическая
3	Версия проекта	Содержит версии проектов	Характеристическая

Продолжение таблицы 7

№ п/п	Наименование	Описание	Категории
4	Участник проекта	Содержит сведения об участниках проекта	Характеристическая
5	Отзыв	Содержит сведения об отзывах проекта	Характеристическая
6	Комментарий	Содержит информацию о комментариях к проекту	Характеристическая

На основе разработанной концептуальной модели была построена логическая модель базы данных «Хранилище кода», представленная на рисунке 30.

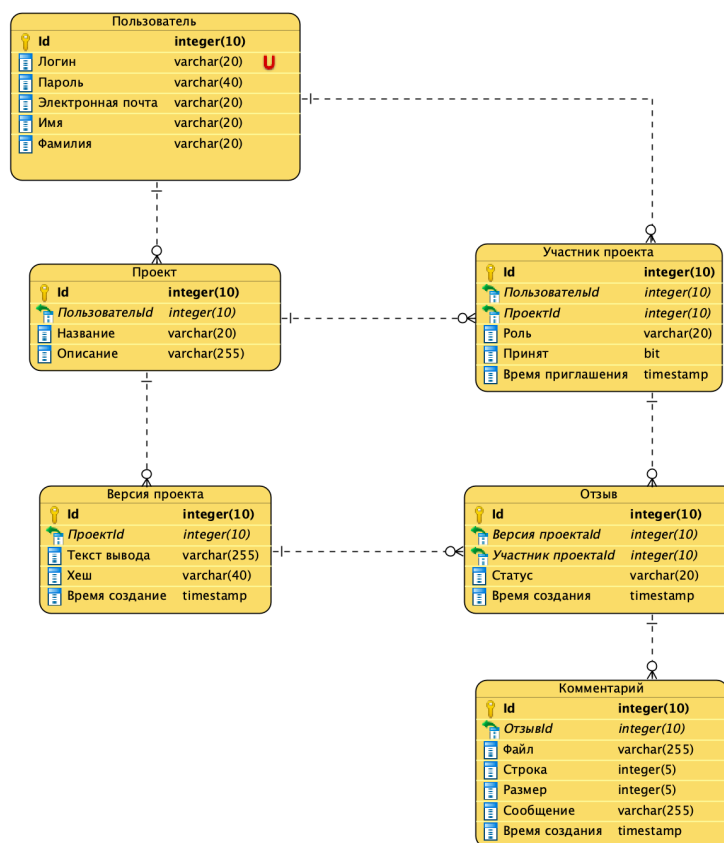


Рисунок 30 – Логическая модель базы данных «Редактор кода»

Данная модель будет в дальнейшем использоваться для проектирования и развертывания базы данных. Для представленной логической модели базы данных «Хранилище кода» также были разработаны спецификации атрибутов для каждой таблицы, представленные в приложении Г.

2.4.3 Разработка физической модели базы данных

Физическая модель – логическая модель базы данных, выраженная в терминах языка описания конкретной СУБД [10].

На основании концептуальной и логической модели была разработана физическая модель базы данных «Хранилище кода», представленная на рисунке 31.

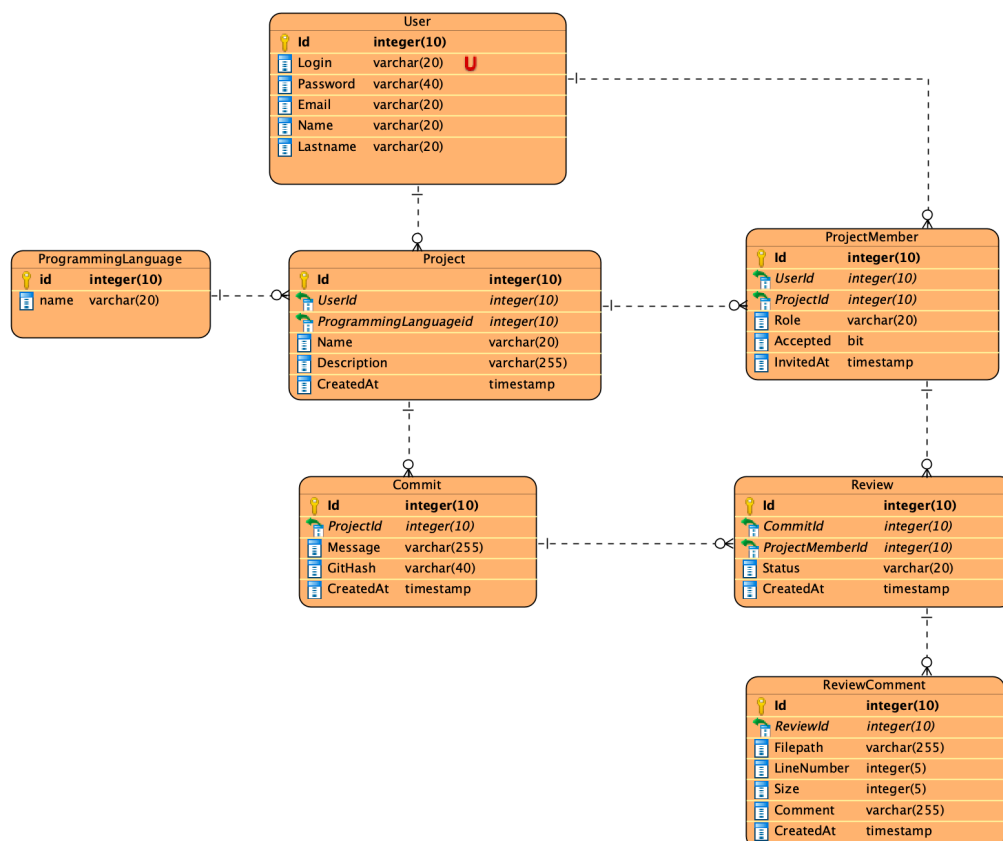


Рисунок 31 – Физическая модель базы данных

Кроме того, на основе разработанной физической модели базы данных был сгенерирован скрипт на языке PostgreSQL для создания объектов базы данных «Редактор кода». Фрагмент скрипта для создания объектов базы данных приведен в листинге 1.

Листинг 1 – Создание таблиц базы данных

```

CREATE TABLE Project (
    Id SERIAL PRIMARY KEY,
    UserId INT NOT NULL,
    ProgrammingLanguageId INT NOT NULL,

```

```
Name VARCHAR(255) UNIQUE NOT NULL,
Description VARCHAR(255),
CreatedAt TIMESTAMP NOT NULL,
FOREIGN KEY (UserId) REFERENCES Users (Id));
```

Полный скрипт кода развертывания базы данных содержится в Приложении Д.

2.4.4 Развертывание базы данных в среде pgAdmin 4

Для создания базы данных, используемой в разработке, была выбрана PostgreSQL – реляционная СУБД, обеспечивающая надежность, согласованность и поддержку сложных связей между сущностями, она может работать под управлением любой распространенной операционной системы. PostgreSQL гарантирует целостность данных и высокую производительность при одновременной работе нескольких пользователей [11].

Создание производилось при помощи скрипта на языке Transact-SQL, сгенерированного по физической модели базы данных в среде Visual Paradigm.

После создания объектов базы данных, была сгенерирована диаграмма данных, представленная на рисунке 32.

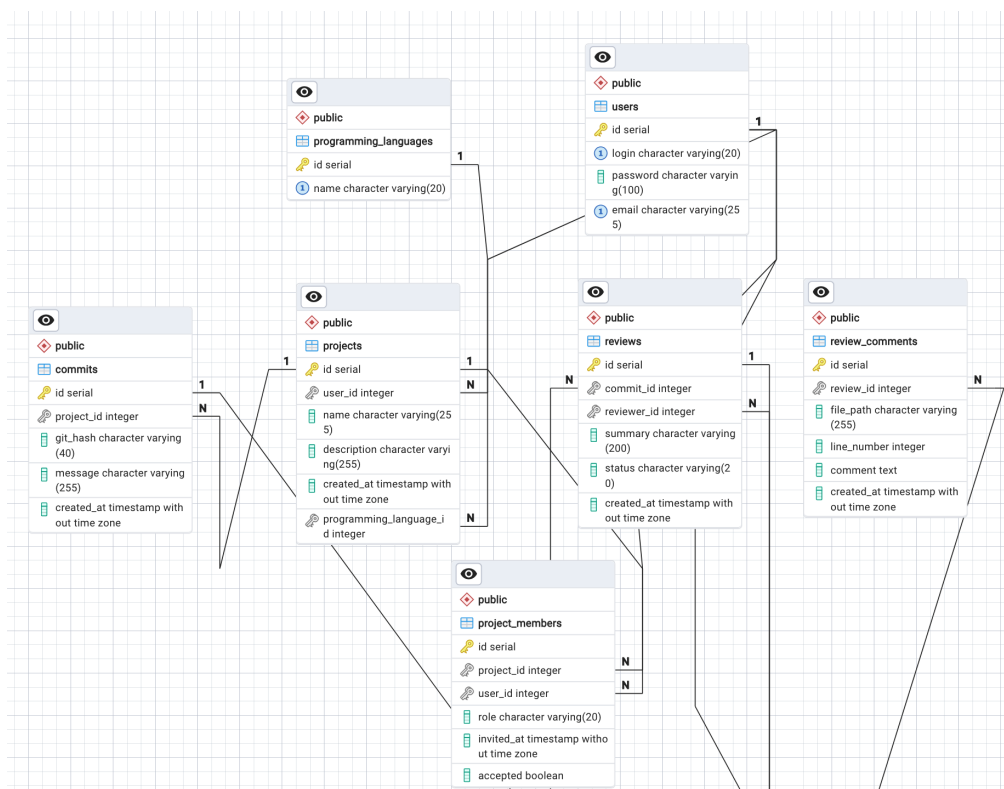


Рисунок 32 – Диаграмма данных, сгенерированная в pgAdmin 4

Полученные результаты используются в ходе наполнения базы данных и в процессе разработки приложения.

2.5 Проектирование взаимодействия со средствами контейнеризации и системой контроля версий

Для обеспечения изолированного и масштабируемого исполнения пользовательских проектов в разрабатываемом онлайн-редакторе кода реализуется архитектура, основанная на применении контейнеризации и системы контроля версий. Такой подход позволяет не только эффективно управлять средами исполнения, но и гарантировать сохранность и актуальность исходного кода каждого проекта.

В качестве основы для изоляции и управления средой исполнения пользовательского кода применяется технология Docker [12]. Для каждого проекта пользователя при запуске назначается отдельный контейнер, что обеспечивает изоляцию исполняемого процесса и исключает конфликта между различными сессиями и пользователями.

Планируется, что в конфигурации контейнера будет установлено несколько языков программирования, а также необходимые инструменты для сборки и выполнения кода. Кроме того, обязательной частью окружения выступает установленный Git для контроля версий.

Для управления исходным кодом и отслеживания истории изменений используется система контроля версий Git. Git предоставляет распределенную систему управления версиями, где каждый пользователь обладает полной копией репозитория и всей его истории [13]. Все операции по сохранению изменений сохраняются в Git-репозиториях проекта, а статус и история изменений становятся доступны через интерфейс онлайн-платформы.

Интеграция Docker и Git в архитектуре системы обеспечивает:

- безопасность и изоляцию каждого пользовательского окружения;
- возможность гибкого масштабирования платформы;
- управление версиями и историей кода;

– интеграция инструментов для совместной разработки кода.

Для наглядной иллюстрации работы проектируемой архитектуры на рисунке 33 представлена диаграмма последовательности процесса «Сохранение проекта» пользователем.

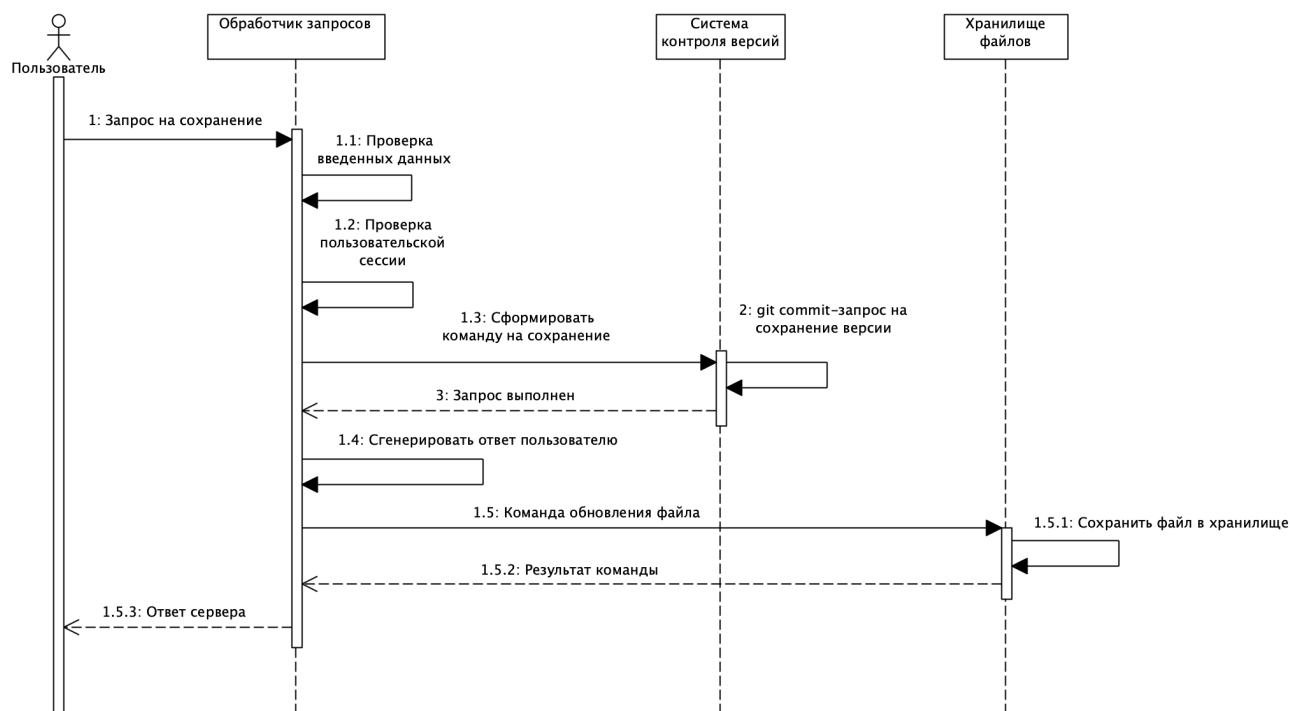


Рисунок 33 – Диаграмма последовательности

Данная диаграмма последовательности иллюстрирует этапы обработки информации, содержащейся в запросе к серверу. Обработчик запросов в контейнере при помощи WebSocket принимает входящие данные, декодирует их в требуемый формат и передает для дальнейшей обработки. Система контроля версий отвечает за сохранение и ведение историй изменений, а хранилище файлов отвечает за физическое сохранение содержимого файлов, к которым производится доступ.

3 Реализация онлайн-платформы для развертывания и разработки программного обеспечения

3.1 Реализация взаимодействия с базой данных

3.1.1 Разработка представлений и триггеров

Для упрощения работы с базой данных в ходе разработки возникла необходимость использовать представления и триггеры для сокращения логики и программного кода в серверной части онлайн-платформы.

Использование структур данных, отличающихся от хранимых в базе данных в языке SQL, реализуется с помощью аппарата представлений. Представления часто используются для того, чтобы записать сложные или часто встречающиеся подзапросы. В ходе разработки приложения было реализовано представление для выборки основных информационных объектов из базы данных. Спецификация представления отображена в таблице 8.

Таблица 8 – Спецификация представления

№ п/п	Имя представления	Назначение	Используемые таблицы	Параметры
1	UserProjectView	Отображение всех проектов пользователя в личном кабинете	User, Project, ProgrammingLanguage	UserId, UserName, ProjectName, ProgrammingLanguage, CreatedAt

Представление «UserProjectView» предназначено для формирования удобного списка проектов, созданных пользователем, оно содержит уникальный идентификатор пользователя, его имя, название проекта, язык программирования и дату создания. Скрипт создания представления приведен в листинге 2.

Листинг 2 – Создание представления

```
CREATE VIEW UserProjectsView AS
SELECT
u.Id AS UserId,
u.Login AS UserName,
p.Name AS ProjectName,
```

```

pl.Name AS ProgrammingLanguage,
p.CreatedAt
FROM
    Users u
JOIN
    Project p ON u.Id = p.UserId
JOIN
    ProgrammingLanguages pl ON p.LanguageId = pl.Id

```

Триггером называют функцию, которая вызывается системой автоматически при срабатывании связанных с ней предписаний [14]. В приложении триггер позволяет в ходе реализации бизнес-процессов обновить связанные таблицы. Спецификация представлена в таблице 9.

Таблица 9 – Спецификация триггера

№ п/п	Назначение триггера	Вид триггера	Событие триггера	Таблица триггера	Описание работы триггера
1	Триггер <code>after_project_insert</code> предназначен для добавления записи в таблицу <code>project_members</code> при выполнении операции <code>Insert</code> .	after	insert	Project	Срабатывает после добавления строк в таблице <code>Project</code> , вызывая функцию <code>after_project_insert</code>

Разработанный триггер «`after_project_insert`» предназначен для автоматического добавления пользователя, который создал проект, в таблицу `ProjectMember` с ролью «owner». Скрипт создания представления приведен в листинге 3.

Листинг 3 – Создание триггера

```

CREATE FUNCTION after_project_insert()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO project_members (project_id, user_id, role,
accepted, invited_at)
    VALUES (NEW.id, NEW.user_id, 'owner', TRUE,
CURRENT_TIMESTAMP);
    RETURN NEW;
END;

$$ LANGUAGE plpgsql;
CREATE TRIGGER after_project_insert
AFTER INSERT ON projects
FOR EACH ROW
EXECUTE PROCEDURE after_project_insert();

```

Таким образом, разработанные представление и триггер позволяют сформировать требуемую информационную выборку и автоматизировать бизнес-процесс работы с проектом.

3.1.2 Реализация серверных запросов к базе данных

Для обращения к базе данных в приложении используются два Go-пакета: `sqlx` и `Squirell`. Библиотека `sqlx` строится поверх стандартного `database/sql`, но имеет расширенный функционал [15]. Библиотека `squirell` выступает генератором SQL-запросов. Использование данных библиотек дает следующие преимущества по сравнению с другими решениями:

- защита от sql-инъекций;
- понятность запросов благодаря цепочке запросов `squirell`;
- динамическое построение запросов;
- маппинг результатов запросов в структуры.

Рассмотрим использование данных запросов на примере работы с таблицей `Project`. Для взаимодействия с таблицей используется структура `ProjectRepo` как реализация интерфейса `ProjectRepository`. Подход с использованием интерфейсов обеспечивает эффективную абстракцию при работе с данными: добавление новых методов и обходится без вмешательств в другие части кода. Запрос на выборку проектов пользователя приведен в листинге 4.

Листинг 4 – Функция получения проектов

```
func (r *ProjectRepo) GetByID(id int) (*models.Project, error) {
    query := r.sb.
        Select("id", "user_id", "name", "description",
            "created_at").
        From("projects").
        Where(squirrel.Eq{"id": id})
    sql, args, err := query.ToSql()
    if err != nil {
        return nil, err
    }
    var project models.Project
    err = r.db.Get(&project, sql, args...)
    return &project, err
}
```

Данный листинг демонстрирует цепочку вызовов библиотеки `squrell`, где задаются условия выборки и указывается необходимая таблица. `Sqlx` используется для маппинга результата запросов в структуру `projects`, которая впоследствии переходит на слой выше.

Такой простой и эффективный подход к построению запросов применяется ко всем таблицам в базе данных, обеспечивая понятность и единообразие кода.

3.1.3 Наполнение базы данных тестовыми данными

Наполнение базы данных тестовыми данными осуществляется через вызовы соответствующих команд добавления данных в таблицы базы данных [16]. В листинге 5 и 6 приведены фрагменты скрипта для последовательности вызовов команд добавления данных. В качестве примера рассмотрим наполнение основных таблиц баз данных, остальные заполняются по такому же образцу.

Листинг 5 – Заполнение таблицы Users

```
INSERT INTO users
(id, login, password, email) VALUES
(1, 'alice', 'passAlice', 'alice@example.com'),
(2, 'bob', 'passBob', 'bob@example.com'),
(3, 'charlie', 'passCharlie', 'charlie@example.com');
```

Листинг 6 – Заполнение таблицы ProjectsMembers

```
INSERT INTO project_members
(id, project_id, user_id, role, invited_at, accepted)

VALUES
(1, 1, 1, 'OWNER', NOW(), TRUE),
(2, 1, 2, 'DEV', NOW(), TRUE),
(3, 1, 3, 'REVIEWER', NOW(), TRUE),
(4, 2, 2, 'OWNER', NOW(), TRUE),
(5, 2, 1, 'DEV', NOW(), TRUE),
(6, 2, 3, 'REVIEWER', NOW(), TRUE),
(7, 3, 3, 'OWNER', NOW(), TRUE),
(8, 3, 1, 'DEV', NOW(), TRUE),
(9, 3, 2, 'REVIEWER', NOW(), TRUE);
```

Наполненная база данных используется в дальнейшем для отладки и тестирования запросов и представлений, функций и хранимых процедур, триггеров.

3.2 Реализация серверной части онлайн-платформы

Серверная часть сервисов для онлайн-платформы для редактирования и развертывания программного кода разрабатывалась в среде программирования VS Code, на языке программирования Go. Код серверной части приложения приведен в приложении Б.

3.2.1 Реализация авторизации приложения

Сервис авторизации необходим в любой распределенной системе для безопасного доступа к ней. Он обеспечивает следующие функции:

- аутентификацию пользователей;
- авторизацию в системе;
- управление сессиями;
- интеграция с другими сервисами, используя валидацию;
- регистрация новых пользователей на платформе.

Рассмотрим подробнее процесс авторизации. Сервис принимает HTTP-запрос и декодирует его в необходимый вид при помощи функции `LoginHandler`, в результате возвращается токен авторизации в JSON формате. Реализация данной функции приведена в листинге 7.

Листинг 7 – Код работы `LoginHandler`

```
func LoginHandler(authService service.AuthService) fiber.Handler {
    return func(c *fiber.Ctx) error {
        var loginUser domain.LoginUser
        if err := c.BodyParser(&loginUser); err != nil {
            c.Status(http.StatusOK)
            return err
        }

        loginResult, err := authService.Login(loginUser.Username,
            loginUser.Password)
        if err != nil {
            c.Status(http.StatusOK)
```

```

    return err}

    return c.JSON(fiber.Map{
        "access_token": loginResult.AccessToken,
        "refresh_token": loginResult.RefreshToken,}))}

```

Рассмотрим подробнее генерацию jwt-токенов. Функция `Generate` генерирует пару токенов – `Access` и `Refresh Token` – для указанного пользователя. Сперва формируется `claims` запроса – данные, содержащие пользовательскую информацию, после чего они подписываются с помощью алгоритма `HMAC` и секретного ключа. Если все операции выполнены успешно, функция возвращает сгенерированные `Access` и `Refresh` токены. В случае ошибки на любом из этапов, выполнение функции прекращается и возвращается текст возникшей неисправности. Код функции приведен в листинге 8.

Листинг 8 – Код работы генерации jwt-токена

```

func (s *JwtServiceImpl) Generate(user domain.User) (string,
string, error) {
    claims := jwt.NewWithClaims(jwt.SigningMethodHS256,
jwt.MapClaims{
    "user_id": user.Id,
    "username": user.UserName,
    "exp": time.Now().Add(_accessTokenLifeTime).Unix(),
    })

    accessToken, err := claims.SignedString([]byte(s.JwtSecret))
    if err != nil {
        return "", "", err
    }

    refreshTokenBase := domain.RefreshTokenBase{
        UserId:    user.Id,
        UserName:  user.UserName}

    refreshToken, err := s.generateRefreshToken(refreshTokenBase)
    if err != nil {
        return "", "", err}
    if err = s.RefreshTokenRepository.Set(context.TODO(),
refreshToken); err != nil {
        return "", "", err}
    return accessToken, refreshToken, err}

```

Создание надежного и защищенного сервиса авторизации является важным этапом в обеспечении эффективной работы онлайн-платформы.

Реализованные процессы регистрации, входа, управления токенами и выхода из системы позволяют эффективно защищать данные пользователей и управлять доступом к ресурсам. Интеграция базы данных Redis для работы с Refresh Token и использование jwt-токенов обеспечивает баланс между производительностью, безопасностью и удобством.

3.2.2 Реализация оркестратора

Оркестратор является ключевым компонентом инфраструктуры онлайн-платформы и является исполнителем механизма автоматического запуска и управления Docker-контейнерами. Каждый из контейнеров создается для конкретного проекта каждого пользователя, что обеспечивает изоляцию среды выполнения и повышает безопасность работы кода. С помощью такого подхода у пользователя есть возможность выполнять и тестировать в максимально приближенным к реальным условиям.

В листинге 9 представлен метод `triggerHandler`, который демонстрирует работу оркестратора при запуске нового контейнера.

Листинг 9 – Код функции `triggerHandler`

```
func triggerHandler(w http.ResponseWriter, r *http.Request) {
    freePort, err := getFreePort()
    if err != nil {
        http.Error(w, fmt.Sprintf("Ошибка поиска свободного
порта: %v", err), http.StatusInternalServerError)
        return
    }
    projectDir := "/Users/vyacheslavzabenko/GolandProjects/code-
executor"
    projectName := "executor-" + uuid.New().String()

    cmd := exec.Command("docker", "compose", "-p", projectName,
"up", "-d")
    cmd.Dir = projectDir
    cmd.Env = append(cmd.Env,
        "WS_PORT="+strconv.Itoa(freePort),
    )
    output, err := cmd.CombinedOutput()
    if err != nil {
        log.Printf("Ошибка запуска контейнера: %v, вывод: %s",
err, string(output))
    }
}
```

```

        http.Error(w, fmt.Sprintf("Ошибка запуска контейнера: %v,
вывод: %s", err, string(output)), http.StatusInternalServerError)
        return}
    log.Printf("Контейнер %s успешно запущен на порту %d: %s",
projectName, freePort, string(output))
    fmt.Fprintf(w, "Контейнер %s успешно запущен на порту %d",
projectName, freePort)

```

В приведенном коде выполняется последовательно необходимые логические блоки кода. Сначала вызовом функции `getFreePort` определяется доступный порт на устройстве, на котором будет запущен контейнер. Далее с помощью библиотеки `uuid` генерируется уникальное имя контейнера, представляющее собой 16-байтовый массив случайно сгенерированных символов [17]. Затем с помощью команды `docker compose` контейнер запускается с ключом «р», чтобы каждый проект был уникальным.

Таким образом, оркестратор позволяет безопасно и эффективно управлять средой запуска пользовательского кода, обеспечивая масштабируемость и надежность онлайн-платформы.

3.2.3 Реализация контроля версий

Контроль версий – ключевая особенность онлайн-платформы для сопровождения созданных пользователем проектов, он позволяет хранить историю изменений, дает возможность отката к предыдущим версиям и управлять ветками разработки. Использование `Git` позволяет отказаться от хранения версий файлов в виде записей базы данных или дампов файлов. Можно выделить следующие основные преимущества данного подхода:

- оптимизированная структура хранения данных;
- поддержка распределенной модели;
- механизм сравнения версий.

Реализация взаимодействия с системой контроля версий построена на основе `WebSocket`-соединений. Пример обработчика команд приведен в листинге 10.

Листинг 10 – Функция обработчика WebSocket

```
func wsHandler(w http.ResponseWriter, r *http.Request) {
    conn, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        log.Println("Upgrade error:", err)
        return
    }
    defer conn.Close()
    log.Println("New WebSocket connection established")
    for {
        _, msg, err := conn.ReadMessage()
        if err != nil {
            break
        }
        var m Message
        if err := json.Unmarshal(msg, &m); err != nil {
            conn.WriteJSON(Response{Error: "Invalid JSON
format"})
            continue
        }
        var resp Response
        switch m.Type {
        case "history":
            resp = handleRepOperation(m)
        default:
            resp = Response{Error: "Unknown message type"}
        }
        conn.WriteJSON(resp)
    }
}
```

Этот код демонстрирует, как система принимает команды со стороны клиентского приложения, сначала настраивается из обычного HTTP-соединения WebSocket-соединение, затем в бесконечном цикле программа прослушивает программа обрабатывает входящие сообщения и по их типу распределяет в соответствующие функции.

Рассмотрим подробнее работу с Git на примере получения содержания файла, который представлен в листинге 11.

Листинг 11 – Получение содержания файла

```
func (m *HistoryManager) GetFileContentByCommitHash(projectID,
relativePath, commitHash string) (string, error) {
    gitPath := filepath.ToSlash(relativePath)
    cmd := exec.Command("git", "show", fmt.Sprintf("%s:%s",
commitHash, gitPath))
    cmd.Dir = projectPath
    output, err := cmd.CombinedOutput()
    if err != nil {
        return "", fmt.Errorf("git error: %v, output: %s", err,
string(output))
    }
    if IsBinary(output) {
        return "", fmt.Errorf("cannot read binary file")
    }
    return string(output), nil
}
```

Данный метод запускает внешнюю команду `git show` через библиотеку `exes`, передавая рабочую директорию запущенного проекта и идентификатор версии запуска. В результате, если функция `IsBinary` определяет, что файл двоичный, возвращается ошибка, иначе содержание файла возвращается обратно в вызываемый блок кода.

3.3 Реализация клиентской части онлайн-платформы

Клиентская часть компонентов для онлайн-платформы для развертывания и редактирования программного кода разрабатывалась в среде `Webstorm 2024`, на языке программирования `JavaScript` с использованием библиотеки `ReactJS`. Код клиентской части приложения приведен в приложении В.

3.3.1 Реализация визуального отображения программного кода

Для визуального отображения программного кода в онлайн-платформе используется компонент `Monaco Editor`, который представляет собой мощный и гибкий редактор кода, разработанный командой `Microsoft` и используемый в таких продуктах, как `Visual Studio Code` [18]. Редактор поддерживает основные функции для комфортного использования пользователем программного кода, он обеспечивает следующие функции:

- подсветка синтаксиса для многих языков программирования;
- автодополнение кода;
- работа с большими файлами;
- встроенные подсказки.

Рассмотрим реализацию компонента на примере его подключения в сервисе запуска кода без регистрации, использование редактора приведено в листинге 12.

Листинг 12 – Код отображения программного кода

```
<Box w="50%" p={4} display="flex" flexDirection="column">
  <Box flex="1" mt={4}>
    <LanguageSelector language={language} onSelect={setLanguage} />
    <Editor
```

```

        height="100%"
        language={language}
        value=""
        onMount={onMount}
        options={{
          automaticLayout: true,
        }}
      />
    </Box>
  </Box>

```

В этом фрагменте интерфейс собирается на компонентах Chakra UI, которая является библиотекой, предоставляющей современные примитивы для работы с разметкой сайта. Контейнер Box задает гибкую область, растягивающуюся на все доступное пространство редактора. Далее используется подключаемый компонент Editor, ему передается выбранный язык, а опция automaticLayout заставляет автоматически адаптироваться к размерам контейнера, в котором находится окно редактора. Функция onMount сохраняет экземпляр редактора в ref, для дальнейшего обращения к нему как к объекту.

На рисунке 34 изображено представление области работы с кодом



```

1  import React, { useState } from 'react';
2  import {
3    Box,
4    Flex,
5    Heading,
6    Input,
7    Button,
8    FormControl,
9    FormLabel,
10   Text,
11   Image,
12   Link,
13   useToast,
14 } from '@chakra-ui/react';
15 import { useNavigate } from 'react-router-dom';
16
17 const LoginRegisterPage = () => {
18   const [isLogin, setIsLogin] = useState(true); // Переключение между формами входа и
19   const navigate = useNavigate(); // Хук для навигации
20   const toast = useToast(); // Для уведомлений
21
22   const handleLogin = (e) => {
23     e.preventDefault();
24     const email = e.target.elements.email.value;
25     const password = e.target.elements.password.value;
26
27     if (email === 'slava@mail.ru') {
28       toast({
29         title: 'Успешная авторизация!',
30         description: 'Вы успешно вошли в аккаунт.',
31         status: 'success',
32         duration: 3000,
33         isClosable: true,
34       });
35     }
36   };
37 }

```

Рисунок 34 – Область редактирования кода

На изображении видно, что редактор использует подсветку ключевых слов, комментариев, числовых и текстовых литералов. Благодаря такой подсветке разработчик быстрее ориентируется в коде и реже допускает синтаксические ошибки.

3.3.2 Реализация визуального отображения дерева проекта

Компонент отображения дерева файла является главным навигационным узлом в проекте. Пользователь в одном месте может видеть структуру каталогов, создавать новые элементы, переименовывать или удалять их при необходимости. Файловая структура синхронизируется с сервером по WebSocket-каналу, это значит, что любое действие, совершенное с элементами проекта, синхронизируется с History Manager на стороне контейнера, рассмотрим подробнее работу компонента на примере листинга 13.

Листинг 13 – Код отображения файлового дерева

```
const FolderComponent = ({ folder, onFileSelect }: { folder:
Folder; onFileSelect: (f: Folder) => void }) => {
const [open, setOpen] = useState(false);
return (
  <Box>
    <Flex onClick={() => setOpen(!open)} _hover={{ bg: 'gray.200'
}} p={1}>
      {folder.isFile ? <DocumentIcon /> : <FolderIcon />}
      <Text ml={2} cursor={folder.isFile ? 'pointer' :
'default'} onClick={() => folder.isFile && onFileSelect(folder)}>
        {folder.name}
      </Text>
    </Flex>
    {open && folder.folders?.map(child => (
      <Box pl={4} key={child.id}>
        <FolderComponent folder={child}
onFileSelect={onFileSelect} />    {/* рекурсивный вызов */}
      </Box>))}
  </Box>);};
```

Компонент FolderComponent обрабатывает узел и, раскрываясь, рекурсивно отрисовывает дочерние элементы. ChakraUI и Heroicons задают адаптивный layout и визуальное отображение файлов и папок. Для каждого узла доступны контекстное меню и модальные окна для управления элементами

проекта. Рассмотрим визуальное отображение реализации данного компонента на рисунке 35.

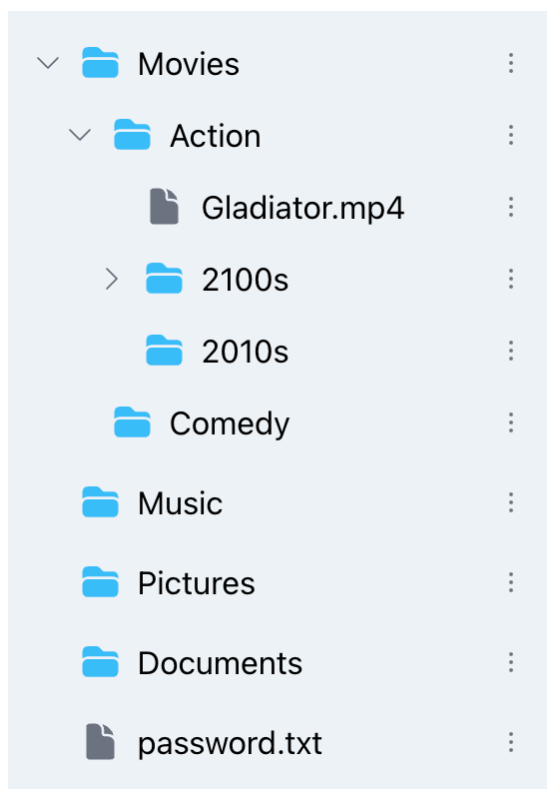


Рисунок 35 – Файловое дерево проекта

На изображении видно, как раскрытая ветка показывает вложенные подпапки и файл «Gladiator.mp4», остальные же каталоги остаются свернутыми. Стрелки на иконках папок интуитивно отражают открытое или закрытое состояние проекта, а правое меню-троеточие позволяет управлять файлами. Данный интерфейс можно считать удобным – вся структура находится в быстром доступе, а любые изменения отправляются на сервер по WebSocket-соединению, мгновенно обновляя состояние у участника сессии.

3.3.3 Реализация проверки кода сторонними участниками

Для совместного использования кода в онлайн-платформу встроен режим Code-Review. Он позволяет распространять проекты между приглашенными пользователями для коллективного обсуждения существующих решений. Данный режим позволяет оставлять строчные комментарии прямо в тексте, не

покидая браузер и не переключаясь на другие сервисы, полученные изменения отправляются на сервер с помощью HTTP-запроса, рассмотрим подробнее работу компонента на примере листинга 14.

Листинг 14 – Код добавления комментариев

```
editor.onContextMenu(e => {
  if (!isCodeReviewMode) return;
  const sel = editor.getSelection();
  if (sel && !sel.isEmpty()) {
    e.event.preventDefault();
    setCurrentSelection(sel);
  }
});
```

После ввода текста комментариев сохраняется в локальном состоянии `comments`, а затем превращается в `decoration` Monaco Editor. Каждый комментарий получает уникальный идентификатор и координаты диапазона, при наведении на подсвеченную область всплывает `hoverMessage` с текстом комментария. Визуальное отображение режима Code-Review можно увидеть на рисунке 36.

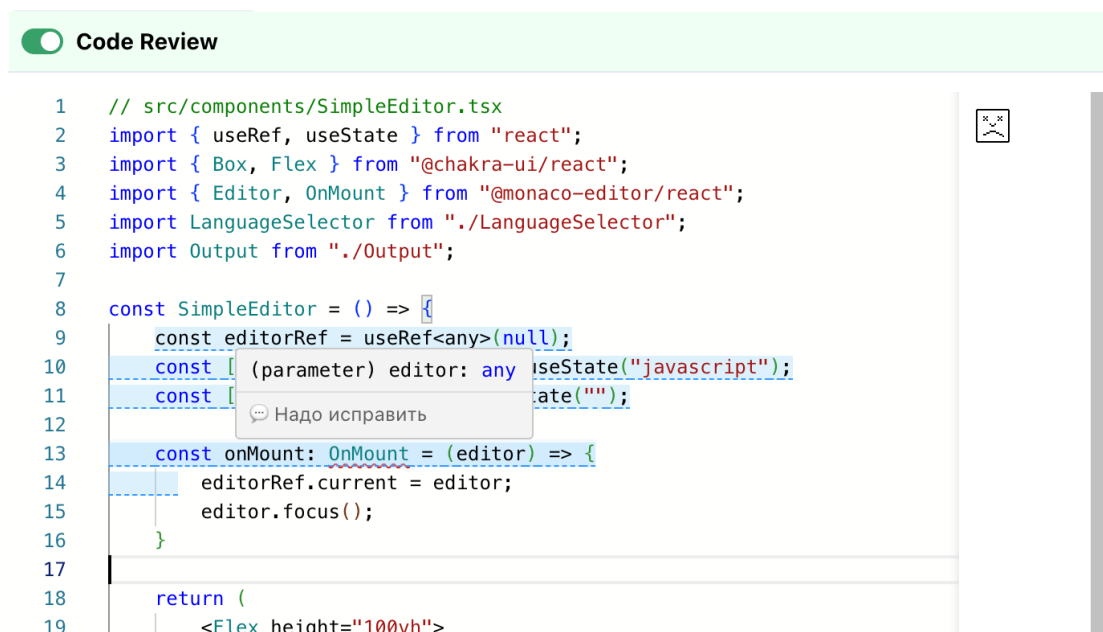


Рисунок 36 – Режим Code-Review

Таким образом, механизм встроенных отзывов на код делает проверку кода быстрой, контекстной и прозрачной для всех, разработчик может сразу

видеть, что обсуждают приглашенный пользователи, а корректировки можно вносить на одной странице с открытым кодом.

3.3.4 Реализация переключения по истории проектов

Для полноценного отслеживания изменений проекта и в случае необходимости переключения на другую версию проекта, разработчику необходимо предоставить удобный интерфейс для выполнения таких задач. В редакторе это решается виджетом History Viewer, основанным на компоненте DiffEditor из Monaco, он показывает различия между любой парой версий, полученных из Git. Рассмотрим подробнее работу виджета на листинге 15.

Листинг 15 – Код виджета контроля версий

```
const [idx, setIdx] = useState(patches.length - 1);
const { original, modified } = parseGitDiff(patches[idx]);
<DiffEditor
  original={original}
  modified={modified}
  language={lang}
  options={{ renderSideBySide: false, readOnly: true }}
/>
```

Компонент принимает массив patches – строки формата, который возвращает команда git diff на сервере. Функция parseGitDiff разбирает патч и возвращает две версии текста: старую и новую. DiffEditor визуализирует их в inline-режиме, подсвечивая все добавления и удаления. Визуальное отображение виджета можно увидеть на рисунке 37.

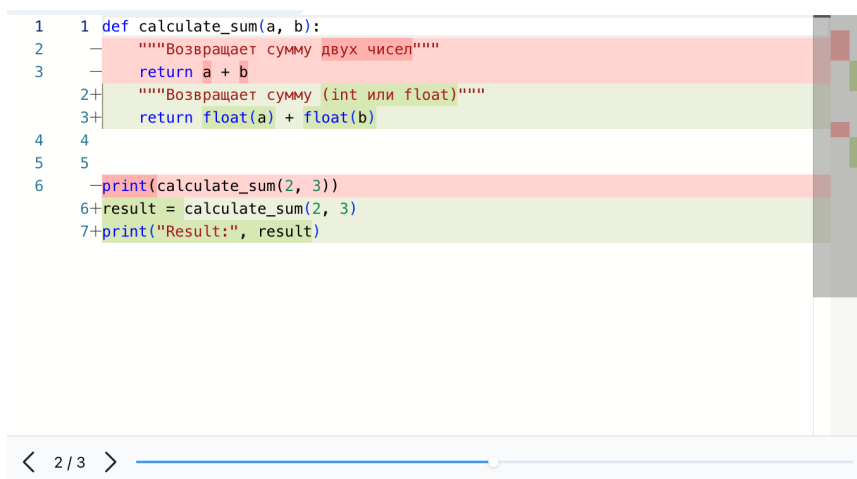


Рисунок 37 – История версий проекта

На изображении видно, что изменения показываются прямо в редакторе кода, а снизу располагается мини-панель истории версий: две кнопки переключения и слайдер для навигации, а также количество сохраненных версий проекта. В итоге History Viewer помогает анализировать историю проекта самым эффективным способом – как прокрутка ленты с мгновенным сравнением версий.

3.3.5 Реализация сборки компонентов

Сборка клиентской части выполняется с помощью современного сборщика Vite. Он поднимает dev-сервер с возможностью моментальной перезагрузки, автоматически обновляет зависимости и управляет пакетами и компонентами проекта. Использование точно интегрированных React-компонентов позволяет быстро масштабировать интерфейс без необходимости модифицировать остальные части программы. На рисунке 38 показана диаграмма компонентов клиентской части онлайн-платформы.

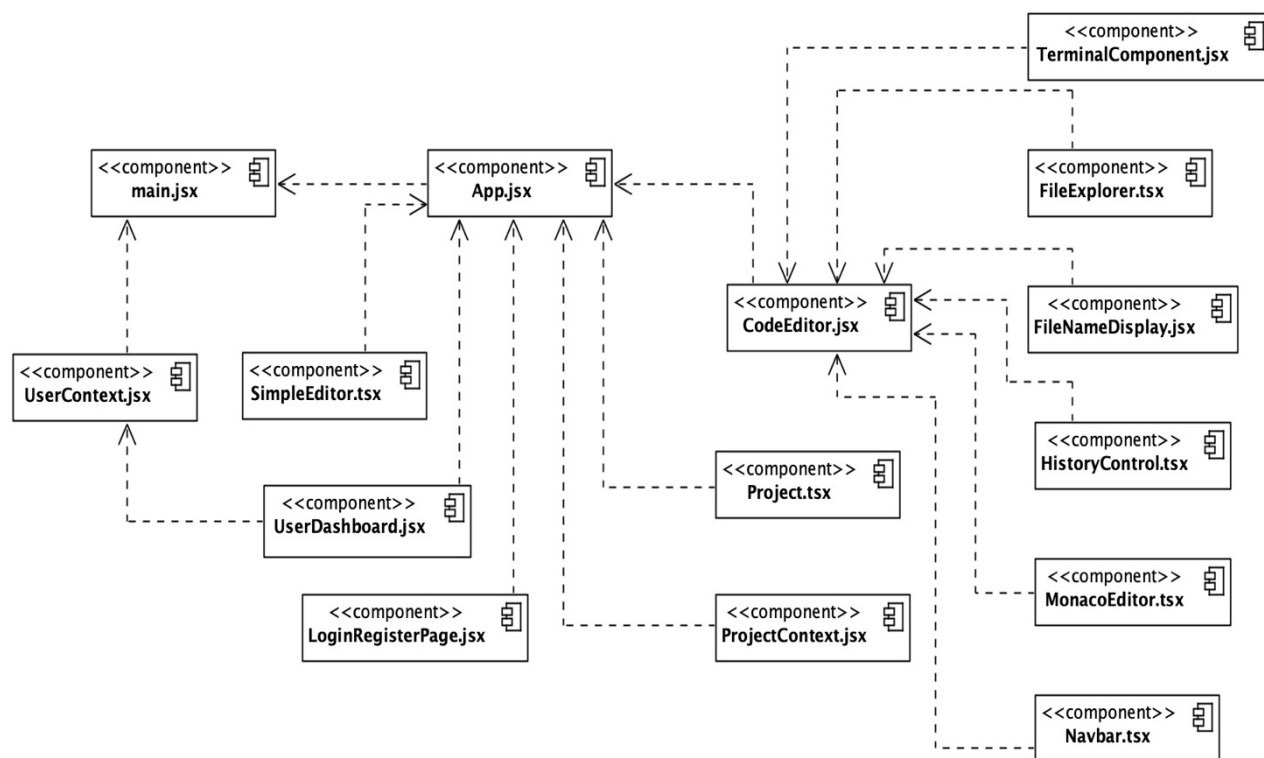


Рисунок 38 – Диаграмма компонентов клиентской части приложения

Описание компонентов клиентской части онлайн-платформы представлено в таблице 10.

Таблица 10 – Компоненты клиентской части приложения

Компонент	Назначение
main.jsx	Входная точка для React приложения
App.jsx	Корневой компонент, содержащий в себе маршрутизацию.
UserContext.jsx	Компонент, содержащий в себе информацию о пользователе.
ProjectContext.jsx	Компонент, содержащий в себе информацию о разрабатываемом проекте.
SimpleEditor.tsx	Компонент, реализующий функционал страницы работы с редактором кода без регистрации.
Project.tsx	Компонент, реализующий функционал страницы отображения текущих проектов зарегистрированных пользователей.
UserDashboard.jsx	Компонент, реализующий функционал страницы с личной информацией пользователя, с возможностью ее редактировать.
LoginRegisterPage.jsx	Компонент, реализующий функционал страницы регистрации и авторизации в системе.
CodeEditor.jsx	Основной компонент, реализующий функционал страницы работы с проектом в режиме его редактирования и отладки.
TerminalComponent.jsx	Компонент, реализующий функционал на странице компонента редактора кода, отвечает за интерактивное взаимодействие пользователя с запущенным на контейнере проектом.
FileExplorer.tsx	Компонент, реализующий функционал на странице компонента редактора кода, отвечает за обеспечение окна дерева проекта, а также всех его модальных окон.
FileNameDisplay.jsx	Компонент, реализующий отображение названия файлов над редактором кода, чтобы пользователь понимать какой файл запущен.
HistoryControl.jsx	Компонент, реализующий функционал страницы истории проекта, позволят переключаться между версиями.
MonacoEditor.jsx	Компонент, реализующий отображение программного кода открытого пользователем файла.
Navbar.jsx	Компонент, реализующий навигацию в приложении.

Для передачи данных между сервером и клиентом используется библиотека `axios`, реализующая передачу данных по HTTP-соединению. Для визуализации полной архитектуры и иллюстрации взаимодействия

пользовательского устройства с сервером была разработана диаграмма развертывания, представленная на рисунке 39.

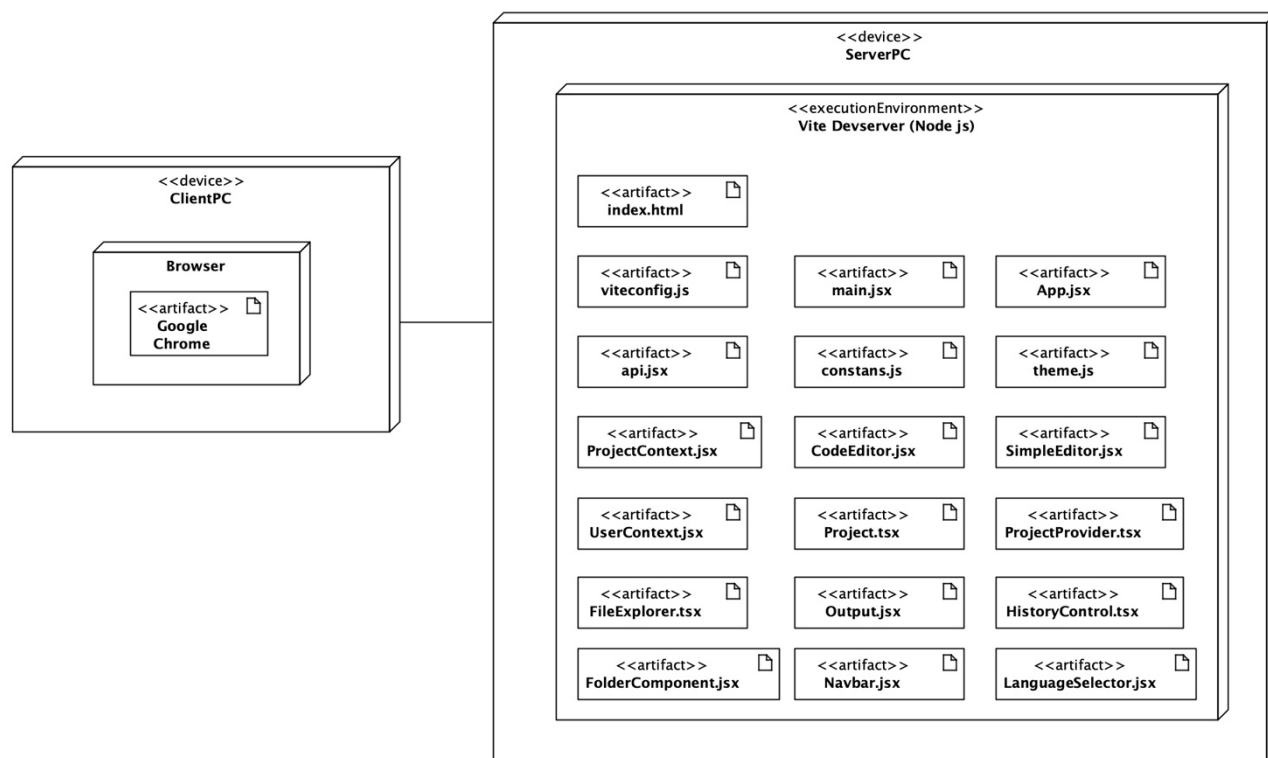


Рисунок 39 – Диаграмма развертывания клиентской части приложения

Диаграмма развертывания моделирует физическое развертывание артефактов на узлах. В качестве первого компонента выступает серверная часть, расположенная на удаленной вычислительной машине. В качестве среды исполнения web-сервера выступает NodeJS при помощи утилиты Vite. К серверу могут быть подключено некоторое количество клиентов, в роли которых выступают web-браузеры на устройствах пользователей.

По результатам кодирования была разработана онлайн-платформа для разработки и развертывания программного кода, позволяющая пользователям работать над своими проектами без необходимости установки сторонних средств. Реализованное приложение было запущено согласно диаграмме развертывания.

4 Тестирование онлайн-платформы для развертывания и разработки программного обеспечения

4.1 Функциональное тестирование клиентской части

Для онлайн-платформы развертывания и разработки программного обеспечения функциональное тестирование гарантирует, что основные операции, предоставляемые приложением, будут выполняться корректно и безопасно для пользователей.

Функциональное тестирование – вид тестирования, который проверяет, насколько точно реализованные в приложении функции соответствуют предъявленным требованиям. Обычно данный подход называется тестированием «черным ящиком», то есть оценивается только внешнее поведение системы, а внутреннее устройство кода можно подтвердить методом «белого ящика» [19].

В рамках функционального тестирования онлайн-платформы было проведено тестирование черным ящиком формы создания нового проекта следующими методиками:

- классы эквивалентности;
- функциональная диаграмма и таблица решений;
- диаграмма состояний и переходов.

Рассмотрим каждый способ по отдельности ниже.

4.1.1 Классы эквивалентности

Классы эквивалентности – группы входных значений, при обработке которых приложение должно демонстрировать одинаковое поведение и выдавать одинаковый результат. Такое разбиение упорядочивает процесс тестирования: на его основе формируется компактный набор проверок, подтверждающий корректность и полноту реализации функции. Рассмотрим классы эквивалентности поля «Название проекта» в таблице 11.

При применении этой методики требовалось перечислить все возможные причины некорректной работы системы, зафиксировать сообщения об ошибках, на основе этого строится упрощенная схема работы приложения и строится таблица, в которой каждому сценарию указывается реакция системы. В результате работы были проанализированы поля для создания проекта и выделены причины и следствия событий, они указаны в таблице 13.

Таблица 13 – Причины и следствия событий

Причины	
П1	Корректная длина названия проекта
П2	Корректная длина описания проекта
П3	Корректные символы названия проекта
П4	Корректные символы описания проекта
П5	Выбран язык программирования
Следствия	
С1	Сообщение: «Некорректная длина названия проекта»
С2	Сообщение: «Некорректная длина описания проекта»
С3	Сообщение: «Некорректные символы названия проекта»
С4	Сообщение: «Некорректные символы описания проекта»
С5	Сообщение: «Не выбран язык программирования»
С6	Сообщение: «Проект успешно создан»

По данной таблице была составлена функциональная диаграмма, представленная на рисунке 40.

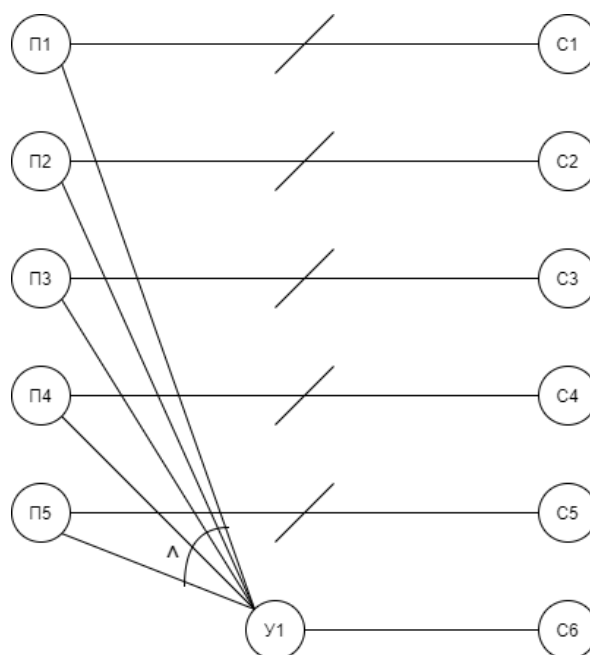


Рисунок 40 – Функциональная диаграмма

Таблица с представлением полного перебора всех существующих ограничений составляет 32 варианта. Чтобы уменьшить число проверок без потери охвата, был применен комбинаторный подход – попарное тестирование. Метод гарантирует, что для каждой возможной пары входных параметров рассматриваются все дискретные сочетания значений, тем самым ощутимо сокращая объем сценариев. При помощи сервиса Pairwise из исходных 32 вариантов остались лишь 16 тест-сценариев. Окончательный перечень приведен в таблице 14.

Таблица 14 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
П1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
П2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
П3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
П4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
П5	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	1
С1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
С2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
С3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
С4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
С5	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Результат выполнения каждого тестового сценария, определенного в таблице 14, привел к успешному выполнению тестирования. Таким образом, можно сделать вывод о том, что разработанное программное обеспечение в создании проекта работает корректно.

4.1.3 Диаграмма состояний переходов

Диаграмма состояний переходов отображает все возможные состояния системы и пути, по которым она может перемещаться между ними. Узлы на

схеме обозначают конкретные состояния, а стрелки – события или условия, вызывающие переходы. В качестве примера был взят программный проект, с которым работает пользователь, его жизненный цикл изображен на рисунке 41.

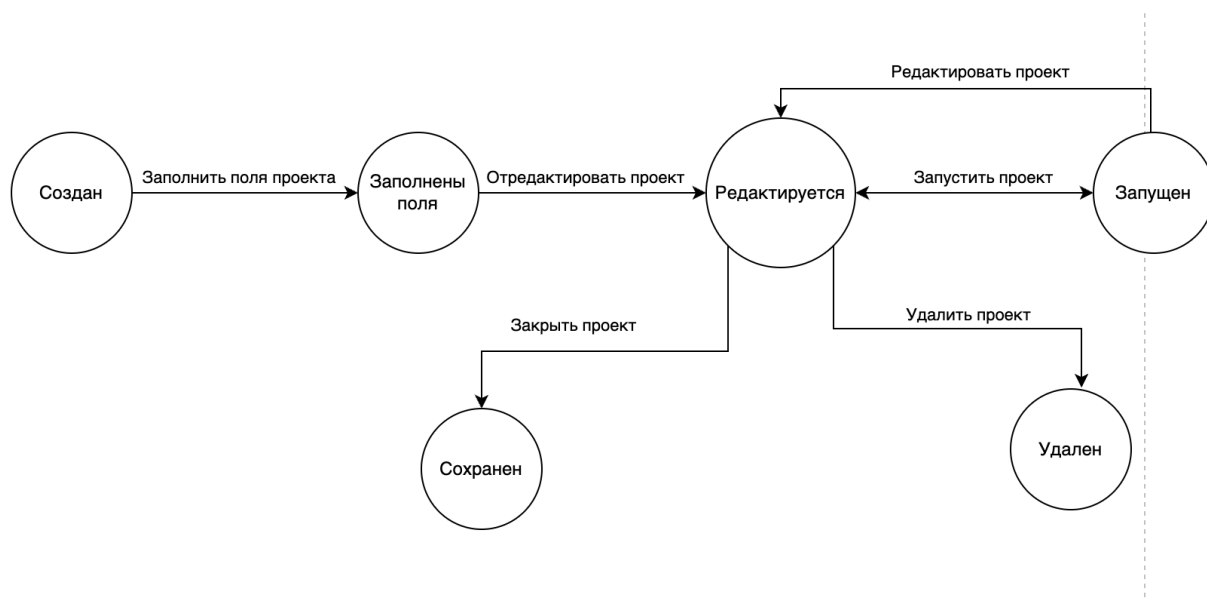


Рисунок 41 – Диаграмма состояний и переходов

Использование данной диаграммы в функциональном тестировании дало несколько результатов:

- сокращение времени на составление сценариев;
- каждая ветка была проверена минимум одним тестом;
- обнаружение и исправление мелких нарушений логики.

Таким образом визуальное моделирование состояний не только упростило разработку тестов, но и повысило логическую уверенность в корректности ключевых операций над проектом.

4.2 Тестирование WebSocket-взаимодействия клиентской и серверной части

Одной из важнейших частей тестирования онлайн-платформы для разработки и развертывания программного обеспечения является проверка корректности соединения пользователя со своим проектом по WebSocket-

соединению. Данный механизм реализован в виде передачи команд на работу с файлами и контролем версий в реальном времени.

Для проверки такого взаимодействия был использован инструмент Postman, представляющий собой платформу для построения и использования API. Postman позволяет отправлять запросы, анализировать ответы и автоматизировать тестирование программных интерфейсов, упрощая взаимодействие с серверной частью приложения [20].

Рассмотрим подробнее интерфейс платформы на примере запроса на получение файла проекта на рисунке 42.

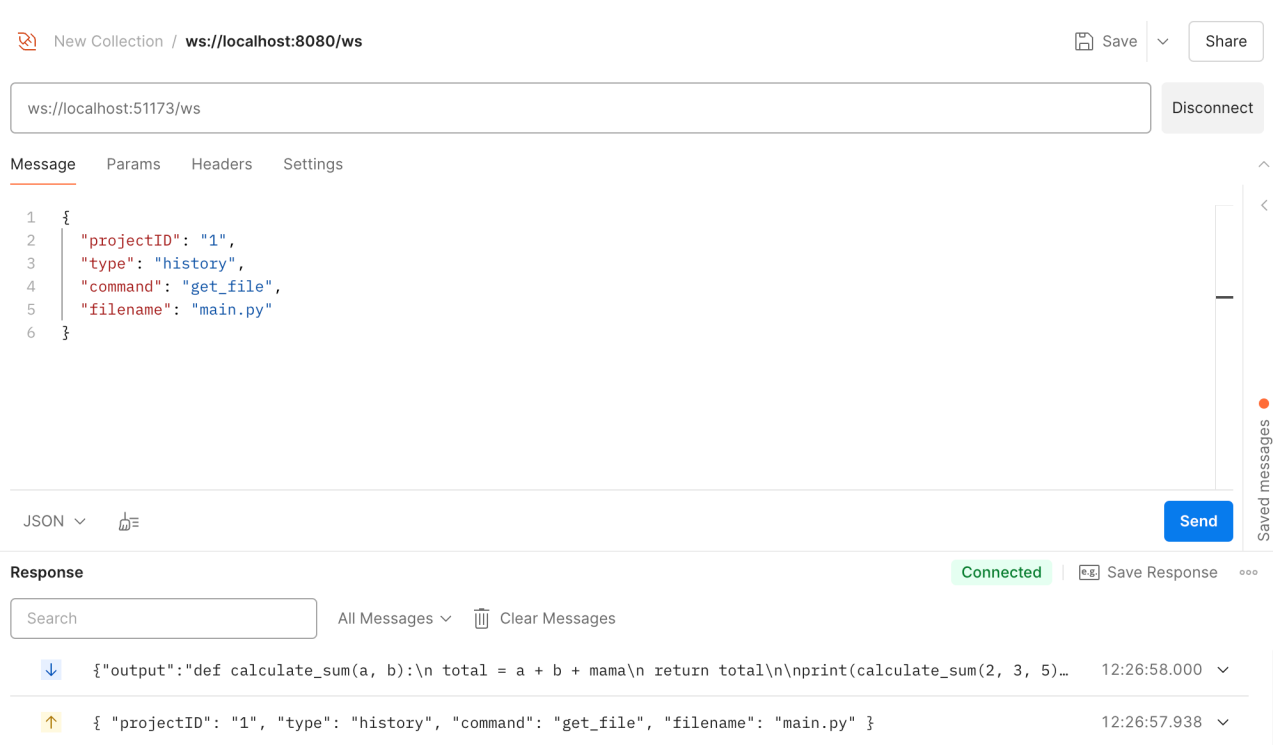


Рисунок 42 – Запрос на получение содержания файла

На представленном рисунке видна рабочая область Postman, включающая следующие элементы:

- строку ввода адреса WebSocket-сервера с кнопкой для подключения;
- окно сообщения для отправки JSON-команды серверу;
- вкладку ответов, где в реальном времени отображаются ответы сервера на отправленные запросы.

На рисунке видно, что со стороны клиента отправляется запрос с содержанием идентификатора проекта, типа сообщения, в данном случае – обращение к классу хранилища, тип команды на получение файла и название требуемого файла. После отправки сообщения сервер вернул JSON-ответ с полем «output», в котором содержится код программы в виде строки с разделителями. Таким образом запрос на получение файла работает корректно.

С помощью такого же механизма тестирования была осуществлена проверка работы контроля версий, использующий Git. Рассмотрим на примере запрос на получение разницы содержаний двух версий на рисунке 43.

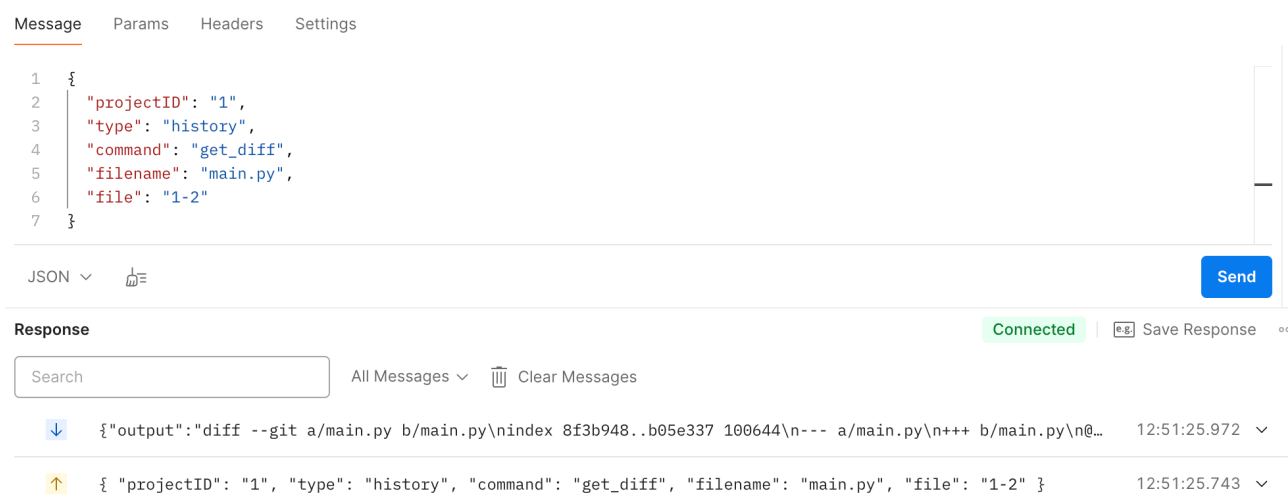


Рисунок 43 – Запрос на получение разницы версий

На рисунке видно, что в отличие от предыдущей команды появилось поле файл, в котором указываются две версии в порядке через разделитель, а также изменилось поле названия команды, так как совершается другая команда. В ответе также получается результат с полем «output», в котором хранится результат Git-команды.

Подобным образом были протестированы все функции взаимодействия клиента и сервера, благодаря встроенной функции Postman по запуску коллекция, проверки онлайн-платформы стали автоматизированными, а результаты – документированы и проанализированы в удобном виде.

Таким образом, тестирование взаимодействия по WebSocket подтвердило корректность и надежность работы взаимодействия клиентской и серверной части разработанного приложения.

4.3 Модульное тестирование клиентской части

Модульный тест – это автоматизированный тест, предназначенный для проверки корректности работы отдельного модуля программы в изоляции от других компонентов. Он позволяет убедиться, что конкретная часть кода функционирует согласно заданным требованиям, независимо от внешних зависимостей [21].

Модульное тестирование клиентской части выполнялось с помощью библиотеки Vitest. Она обеспечивает высокую скорость выполнения тестов благодаря использованию многопоточности и предоставляет поддержку таких функций, как мокирование, покрытие кода и снапшоты, что делает ее удобным инструментом для современных веб-приложений [22].

Рассмотрим работу модульных тестов на примере проверки компонента добавления папки в корневой каталог проекта, данный тест представлен в листинге 16.

Листинг 16 – Тест добавления папки в проект

```
it('добавляет папку в корень через модалку', async () => {
  renderWithProvider(<FileExplorer onFileSelect={() => {}} />);
  const user = userEvent.setup();
  await user.click(screen.getByLabelText('Добавить папку'));
  const input = await screen.findByPlaceholderText('Название');
  await user.type(input, 'Новая папка');
  await user.click(screen.getByText('Добавить'));
  expect(await screen.findByText('Новая
  папка')).toBeInTheDocument();});
```

По данному тесту можно заметить, что компонент FileExplorer рендерится в тестовом окружении, а пользовательское взаимодействие эмулируется с помощью userEvent – специального модуля позволяющего выполнять клики, ввод текста и другие операции, как это делает настоящий пользователь. Тест имитирует добавление папки, а затем проверяет элемент на

экране с определенным названием. Если такой элемент появился, то значит тест прошел успешно. На рисунке 44 представлен результат выполнения этого теста в терминале.

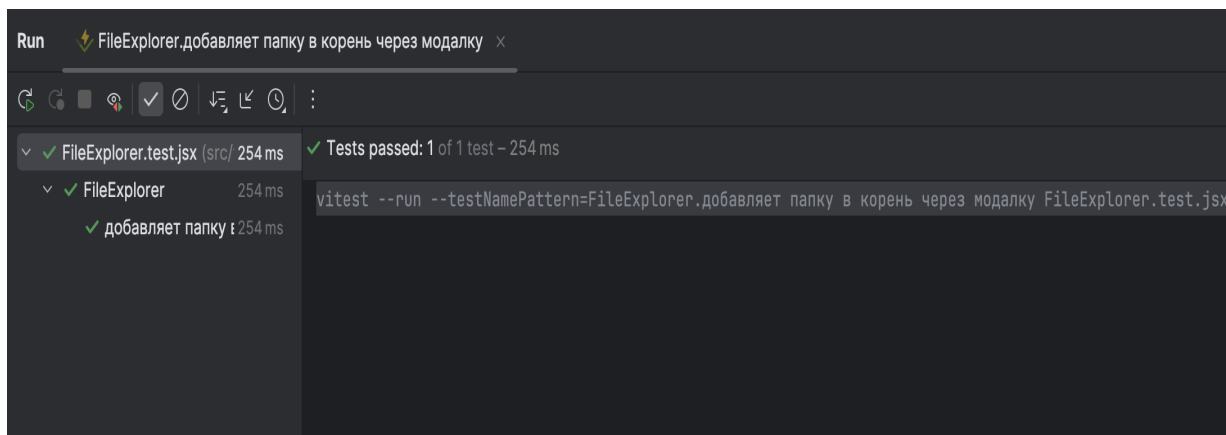


Рисунок 44 – Выполнение теста добавления папки

Аналогичным образом были написаны и успешно выполнены модульные тесты для всех остальных компонентов клиентской части, в том числе создание и удаление файлов, переключение версий, регистрация и авторизация, и другие ключевые сценарии использования приложения. Общий результат выполнения всех тестов представлен на рисунке 45.

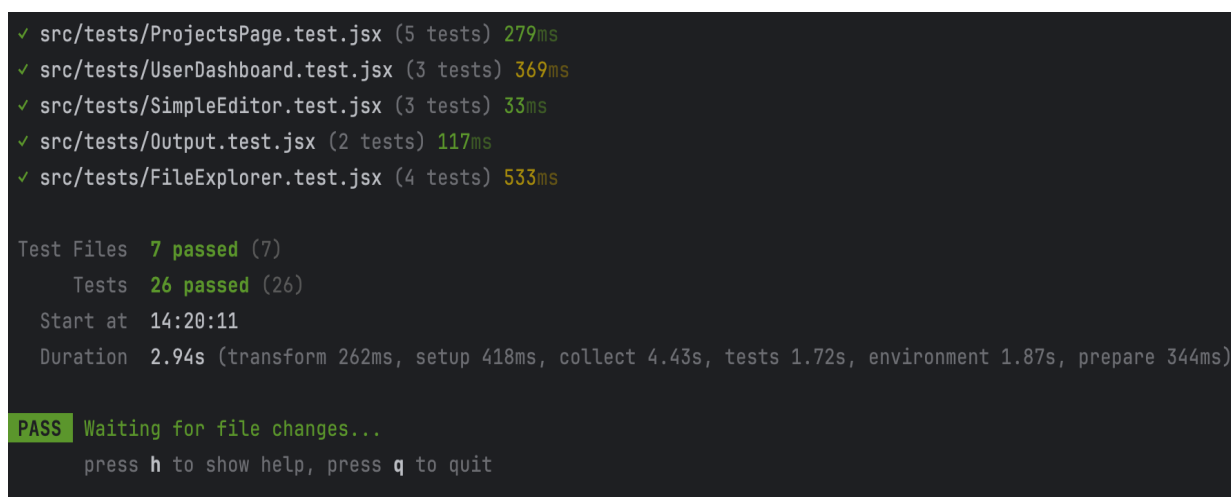


Рисунок 45 – Результат выполнения всех модульных тестов

На рисунке, видно, что все тесты закончены успешно, что говорит о работоспособности клиентской части. Использование Vitest для модульного тестирования клиентской части позволило обеспечить высокое качество кода,

оперативно обнаружить ошибки и гарантировать стабильную работу интерфейса. Полный код модульных тестов приведен в приложении Е.

4.4 Оценка качества программного кода

Метрики программного кода являются важнейшими инструментами для оценки и анализа характеристик качества разрабатываемого программного обеспечения.

Используя метрики, разработчики могут изучить технические характеристики продукта, предсказывать трудоемкость работы и оценивать качество принятых решений. [23].

Для оценки качества кода разработанной онлайн-платформы разработки и развертывания программного обеспечения были выбраны широко известные и хорошо зарекомендовавшие себя метрики – метрика Холстеда и метрика Чепина.

Метрика Холстеда основана на количественном анализе используемых в коде операторов и операндов. Она позволяет определить сложность реализации программы, ее объем, а также прогнозировать трудозатраты на дальнейшую поддержку и расширение функционала.

Метрика Чепина предназначена для оценки внутренней логики программных модулей. Она классифицирует и подсчитывает переменные, используемых внутри программного модуля, позволяя таким образом определить его сложность и улучшить понятность кода [24].

4.4.1 Метрика Холстеда

Для количественной оценки сложности программного кода была проанализирована функция `GetFileContent` класса `HistoryManager`.

Исходный код для метрики Холстеда рассматривается как набор двух типов лексем – операторов и операндов. Метрика требует зафиксировать:

- количество уникальных операторов ($n1$);
- общее число всех операторов ($N1$);

- количество уникальных операндов (n_2);
- общее число всех операндов (N_2).

Эти четыре базовые величины служат входными параметрами формул Холстеда, с их помощью можно определить уровень абстракции и трудоемкость программы.

Перечень операторов, встречающихся в функции GetFileContent, а также частота их использования приведены в таблице 15.

Таблица 15 – Операторы функции GetFileContent

№	Операторы	Число вхождений
1	:=	4
2	=	1
3	if	3
4	!	1
5	Join	1
6	Claen	2
7	return	4
8	HasPrefix	1
9	ReadFile	1
10	isBinary	1
11	Errorf	2
	$n_1 = 11$	$N_1 = 21$

Перечень операндов, встречающихся в функции GetFileContent, а также частота их использования приведены в таблице 16.

Таблица 16 – Операнды функции filterHistories

№	Операнды	Число вхождений
1	m	1
2	HistoryManager	1
3	GetFileContent	1
4	projectID	3
5	relativePath	1
6	fullPath	3
7	«data/projects»	2
8	cleanProjectPath	2
9	cleanFullPath	2
10	«access denied»	1
11	content	2
12	err	2
13	«cannot read binary file»	1
	$n_2 = 13$	$N_2 = 22$

Результаты лексического анализа функции GetFileContent класса

HistoryManager послужили исходными данными для вычисления всех показателей по методике Холстеда; итоговые значения сведены в таблице 17.

Таблица 17 – Результаты вычислений по метрике Холстеда

Значение	Результат
Длина реализации программы	43
Словарь программы	24
Длина программы	108.8
Объем программы	197.09
Потенциальный объем программы	59.37
Теоретический уровень программы	0.30
Сложность программы	3.37
Работа программирования	664.19

Таким образом, вычисленные метрики Холстеда позволили количественно оценить сложность реализации функции, ее объем и уровень программирования, подтвердив вполне достаточный уровень работы и понятности рассматриваемого кода.

4.4.2 Метрика Чепина

Для характеристики структурной сложности была произведена оценка по метрике Чепина для функции UpdateFileContent класса HistoryManager, код которой находится в листинге Г. Все идентификаторы функции проанализированы и распределены на 4 категории:

- параметры, поступающие на вход и участвующие в формировании результата (P);
- переменные, чье значение изменяется в ходе работы (M);
- элементы, которые влияют на выбор ветви исполнения (n2);
- неиспользуемые идентификаторы (N2).

Так как один и тот же идентификатор способен выполнять несколько ролей, при классификации в функции, он учитывался во всех категориях.

Для каждой группы методика Чепина приписывает собственный коэффициент, отражающий ее вклад в общую сложность программного блока:

- для входных и выходных переменных коэффициент отсутствует;
- для изменяемых коэффициент удвоенный;
- для управляющих тройной, так как они наиболее воздействуют на поток управления;
- для «паразитных» переменных коэффициент половинный.

В программном блоке обновления файла были обнаружены 3 входных, 4 изменяющихся, 3 управляющих и 0 «паразитных» переменных. Итоговый показатель вычисляется как количество переменных каждой категории, умноженный на ее коэффициент, затем полученные произведения складываются.

Полученное значение метрики Чепина $Q = 20$ говорит о том, что функция `UpdateFileContent` имеет умеренную сложность и достаточно компактный исходный код. Это облегчает понимание и поддержку класса, улучшая надежность программы.

Заключение

В рамках выполнения выпускной квалификационной работы была разработана онлайн-платформа для развертывания и разработки программного обеспечения, позволяющая упростить процесс ведения программного продукта на каждом из этапов его жизненного цикла.

В начале был выполнен анализ предметной области и существующих программных решений, включая онлайн-редакторы кода, интегрированные среды разработки, облачные IDE, а также CI/CD и DevOps-платформы. На основе данного анализа были сформированы задачи на разработку с подробным описанием функциональных и нефункциональных требований. Также был проанализирован и выбран набор современных инструментов и технологий, а также выполнено оценка проекта с приближенным планированием бюджета.

На этапе проектирования была придумана и реализована архитектура приложения, состоящая из клиентской и серверной части. Клиентская часть была спроектирована с учетом современных тенденций и стандартов веб-разработки, реализована с использованием React, Chakra UI и Monaco Editor. Серверная часть включила в себя проектирование каждого из сервисов отдельности: сервиса-шлюза, сервиса пользователей, сервиса запуска кода, сервиса оркестратора и сервиса авторизации. Была выполнена разработка концептуальной, логической и физической модели базы данных с последующим ее развертыванием в среде pgAdmin 4. Также было предусмотрено проектирование взаимодействия приложения с системой контроля версий Git и средствами контейнеризации Docker.

Разработка онлайн-платформы включила в себя реализацию взаимодействия с базой данных, включая представления, триггер и серверные запросы на языке SQL. Было произведено наполнение базы данных тестовыми данными. Серверная часть включила в себя реализацию на языке Go основных сервисов, а также механизма авторизации, оркестратора, ответственного за запуск изолированных контейнеров, и системы контроля версий с

использованием Git. Клиентский модуль онлайн-платформы спроектирован с приоритетом функциональной полноты и лаконичности. В его состав вошли интерактивное дерево проекта, встроенная система контроля версий и переключение по истории проектов.

Также было проведено тщательное тестирование разработанной онлайн-платформы. С помощью функционального тестирования методами классов эквивалентности и функциональной диаграммы был протестирован основной функционал приложения. Для проверки обмена данными между клиентом и сервером применялся Postman, отправляя сообщения по WebSocket, результаты показали устойчивость двустороннего соединения. Корректность работы клиентской части приложения была проверена с помощью модульного тестирования в среде Vitest, полученные результаты свидетельствуют об удовлетворительном выполнении поведения пользовательского интерфейса. Надежность и эффективность программного кода дополнительно была проверена с помощью метрик Холстеда и Чепина, их значения оказались в рамках умеренной сложности.

Проведенные работы по созданию приложения показали, что разработанная онлайн-платформа полностью удовлетворяет поставленным на разработку задачам.

Дальнейшее развитие данной платформы подразумевает расширения перечня поддерживаемых языков программирования, углублении возможностей самого редактора, например внедрение совместного использования одного окна программного кода, кроме того, интеграция с такими сервисами, как Github и GitLab, что позволит ускорить импорт существующих проектов на внешних сервисах и расширит возможности в совместной разработке программных проектов.

Список использованных источников

1. Абдулла Н. 6 Best Online Code Editors: Comparison Guide // Refine Blog -2024 [Электронный ресурс]. URL: <https://refine.dev/blog/6-best-online-code-editors-comparison/#introduction> (дата обращения: 13.05.2025)
2. Брюстер К. Killing Containers at Scale // Replit Blog – 2021 [Электронный ресурс]. URL: <https://blog.replit.com/killing-containers-at-scale> (дата обращения: 04.06.2025)
3. Что такое CI/CD? // Хабр. Блог компании OTUS – 2020 [Электронный ресурс]. URL: <https://habr.com/ru/companies/otus/articles/515078/> (дата обращения: 04.06.2025)
4. Ричардс Марк, Форд Нил Р56 Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы. – СПб.: Питер, 2023. – 448 с.: ил. – (Серия «Для профессионалов»).
5. REST API // Яндекс Облако. Глоссарий – 2025 [Электронный ресурс]. URL: <https://yandex.cloud/ru/docs/glossary/rest-api> (дата обращения: 04.06.2025).
6. Хабаров, С. П. Построение распределенных систем на базе WebSocket: учебное пособие для СПО / С. П. Хабаров, М. Л. Шилкина. – 2-е изд., стер. – Санкт-Петербург: Лань, 2023. – 216 с.
7. Фаулер, М. Паттерны корпоративных приложений / М. Фаулер ; пер. с англ. [А. Киселева]. – Санкт-Петербург : Питер, 2020. – 544 с.
8. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519 – 2015 [Электронный ресурс]. URL: <https://tools.ietf.org/html/rfc7519> (дата обращения: 13.05.2025).
9. Концептуальная модель. Режим доступа: <https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/> (дата обращения 13.05.2025).

10. Логическая модель базы данных. Режим доступа: https://studopedia.ru/6_1895_logicheskaya-model-bazi-dannih.html (дата обращения 13.05.2025).
11. Understanding ACID Compliance in PostgreSQL // Timescale Blog, 2023 [Электронный ресурс]. – URL: <https://www.timescale.com/learn/understanding-acid-compliance> (дата обращения: 13.05.2025).
12. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment // Linux Journal. – 2014. – №239.
13. Chacon S., Straub B. Pro Git. 2nd Edition. – Apress, 2014.
14. Новиков, Б. А. Основы технологий баз данных: учеб. пособие / Б. А. Новиков, Е. А. Горшкова; под ред. Е. В. Рогова. – М. : ДМК Пресс, 2019. – 240 с.
15. Sqlx – пакет Go: github.com/jmoiron/sqlx – 2025 [Электронный ресурс]. URL: <https://github.com/jmoiron/sqlx> (дата обращения: 13.05.2025).
16. Е.А. Кольчугина. Базы данных. Методические указания к лабораторным работам. / Е.А. Кольчугина, С В. Самуйлов, С.В. Шибанов. Базы данных. Методические указания к лабораторным работам – Пенза: издательство ПГУ, 2000. – 31 с
17. Uuid – пакет Go: github.com/google/uuid – 2025 [Электронный ресурс]. URL: <https://pkg.go.dev/github.com/google/uuid> (дата обращения: 13.05.2025).
18. Monaco Editor // Microsoft – 2025 [Электронный ресурс]. URL: <https://microsoft.github.io/monaco-editor/> (дата обращения: 04.06.2025).
19. Куликов С. Тестирование программного обеспечения. Базовый курс (3- изд.). – Минск.: ЕРАМ Systems, 2021. – 300 с.
20. Postman Documentation. Introduction Overview [Электронный ресурс] / Postman. – Режим доступа: <https://learning.postman.com/docs> (дата обращения: 04.06.2025).

21. Ошеров, Р. Искусство автономного тестирования с примерами на С# / Р. Ошеров. – 2-е изд. – М.: ДМК Пресс, 2016. – 360 с.
22. Vitest. Руководство пользователя [Электронный ресурс] / Vitest. – Режим доступа: <https://www.getbook.com/ru/book/vitest-0/guide/features> (дата обращения: 04.06.2025).
23. Программный код и его метрики // Тематический блог «Хабр». [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 04.06.2025).
24. Ледовских, И. Метрики сложности кода // Технический отчет 2012–2013 И. Ледовских. [Электронный ресурс]. URL: https://se.math.spbu.ru/SE/preprints/prep_25_2013.pdf (дата обращения: 04.06.2025).

Приложение А
(справочное)
Глоссарий

БД (База данных) – упорядоченный набор взаимосвязанных данных, сохранённых в едином хранилище по заранее определённой структуре, что позволяет систематически накапливать, искать, изменять и совместно использовать информацию в течение длительного времени.

Docker – платформа контейнеризации, позволяющая упаковывать приложение и все его зависимости в изолированные образы, обеспечивая воспроизводимость и переносимость среды.

СУБД (Система управления базами данных) – программный комплекс, который позволяет создавать, модифицировать и обслуживать базы данных, обеспечивая многопользовательский доступ, целостность и безопасность хранимых сведений.

Git – Распределенная система контроля версий, обеспечивающая отслеживание изменений и управление историей исходного кода.

UUID (Universally Unique Identifier) – 128-битный уникальный идентификатор, генерируемый без центрального координатора.

API (Application Programming Interface) – задокументированный набор правил, эндпоинтов и форматов сообщений, предоставляющий внешним программам доступ к функциям системы без прямого взаимодействия с её внутренним кодом.

PostgreSQL – объектно-реляционная СУБД с открытым исходным кодом, поддерживающая ACID-транзакции, расширяемые типы данных и репликацию; известна устойчивостью, соответствием стандарту SQL и возможностью горизонтального масштабирования.

WebSocket – протокол полнодуплексного, постоянного обмена сообщениями поверх TCP, используется для двустороннего взаимодействия клиента и сервера в реальном времени.

Фреймворк – программная платформа, выступающая каркасом с архитектурными принципами и едиными нормативами, которые направляют разработку приложений внутри конкретной доменной области.

UI (User Interface) – совокупность визуальных и интерактивных средств, через которые пользователь взаимодействует с программным обеспечением.

JSON (JavaScript Object Notation) – легковесный текстовый формат обмена данными, полностью независимый от языка реализации.

CI/CD – набор практик DevOps для автоматической интеграции изменений, тестирования, сборки и доставки приложения в целевое окружение.

IDE – комплексное средство разработки, объединяющее редактор кода, систему сборки, отладчик и другое, в работе рассматриваются как локальные, так и облачные IDE.

Приложение Б
(обязательное)
Листинг серверной части приложения

Листинг Б.1 – Файл main.go

```
func main() {
    // Инициализация менеджера
    historyManager =
history.NewGitHistoryManager("/data/projects", "CI Bot",
"ci@example.com")
    if err := historyManager.Initialize(); err != nil {
        log.Fatalf("Failed to initialize history manager: %v",
err)
    }

    port := os.Getenv("WS_PORT")
    if port == "" {
        port = "8080"
    }

    http.HandleFunc("/ws", wsHandler)
    log.Printf("WS server started on :%s", port)
    log.Fatal(http.ListenAndServe(":"+port, nil))
}
```

Листинг Б.2 – Файл history.go

```
package history

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "os"
    "os/exec"
    "path/filepath"
    "strings"
    "time"
)

var allowedExtensions = map[string]bool{
    ".py":    true,
    ".js":    true,
    ".ts":    true,
    ".html":  true,
    ".css":   true,
    ".json":  true,
    ".md":    true,
    ".txt":   true,
}

var ignoredNames = map[string]bool{
    ".DS_Store": true,
    ".git":       true,
    "__pycache__": true,
}
```

```

}

type HistoryManager struct {
    projectsBase string
    // Default Git identity для новых репозиторий
    defaultName string
    defaultEmail string
}

type File struct {
    Name string `json:"name"`
    Content string `json:"content,omitempty"`
    IsBinary bool `json:"isBinary"`
}

// NewGitHistoryManager создает менеджер для каталога проектов
// и задает identity для коммитов (например, CI Bot)
func NewGitHistoryManager(projectsBase, name, email string)
*HistoryManager {
    log.Printf("Initializing HistoryManager with base: %s",
projectsBase)
    return &HistoryManager{
        projectsBase: projectsBase,
        defaultName: name,
        defaultEmail: email,
    }
}

// Initialize создает папку projectsBase, если ее нет
func (m *HistoryManager) Initialize() error {
    log.Printf("Ensuring projects base directory exists: %s",
m.projectsBase)
    err := os.MkdirAll(m.projectsBase, 0755)
    if err != nil {
        log.Printf("Failed to create base directory: %v", err)
    }
    return err
}

// ExecGit выполняет произвольную команду git в директории проекта
func (m *HistoryManager) ExecGit(projectID string, args ...string)
(string, error) {
    projectPath := filepath.Join(m.projectsBase, projectID)
    log.Printf("ExecGit: project=%s args=%v", projectID, args)

    // создаём папку проекта
    if err := os.MkdirAll(projectPath, 0755); err != nil {
        log.Printf("Failed to create project dir %s: %v",
projectPath, err)
        return "", fmt.Errorf("failed to create project dir: %v",
err)
    }
}

```

```

    // инициализируем репозиторий, если ещё нет
    gitDir := filepath.Join(projectPath, ".git")
    if _, err := os.Stat(gitDir); os.IsNotExist(err) {
        log.Printf(".git not found in %s, initializing new repo",
projectPath)
        // git init
        initCmd := exec.Command("git", "init")
        initCmd.Dir = projectPath
        out, err := initCmd.CombinedOutput()
        log.Printf("git init output: %s, error: %v", out, err)
        if err != nil {
            return string(out), fmt.Errorf("git init failed: %v,
output: %s", err, string(out))
        }
        // Настраиваем identity для коммитов в этом repo
        configName := exec.Command("git", "config", "user.name",
m.defaultName)
        configName.Dir = projectPath
        if out2, err2 := configName.CombinedOutput(); err2 != nil
{
            log.Printf("git config user.name failed: %s, error:
%v", out2, err2)
        }
        configEmail := exec.Command("git", "config",
"user.email", m.defaultEmail)
        configEmail.Dir = projectPath
        if out3, err3 := configEmail.CombinedOutput(); err3 !=
nil {
            log.Printf("git config user.email failed: %s, error:
%v", out3, err3)
        }
    }

    cmd := exec.Command("git", args...)
    cmd.Dir = projectPath
    out, err := cmd.CombinedOutput()
    log.Printf("git %v output: %s, error: %v", args, out, err)
    return string(out), err
}

func (m *HistoryManager) Save(projectID string) (string, error) {
    projectPath := filepath.Join(m.projectsBase, projectID)
    log.Println(projectPath)

    log.Printf("Saving project %s", projectID)

    statusOut, statusErr := m.ExecGit(projectID, "status")
    if statusErr != nil {

```

```

        log.Printf("git status failed: %v, output: %s",
statusErr, statusOut)
    } else {
        log.Printf("git status:\n%s", statusOut)
    }

    out, _ := m.ExecGit(projectID, "status", "--short")
    log.Printf("Git status short:\n%s", out)

    if _, err := m.ExecGit(projectID, "add", "."); err != nil {
        log.Printf("git add failed: %v", err)
        return "", err
    }

    msg := fmt.Sprintf("autosave: %s",
time.Now().Format(time.RFC3339))
    log.Printf("Committing with message: %s", msg)
    out, err := m.ExecGit(projectID, "commit", "-m", msg)
    if err != nil {
        log.Printf("git commit failed: %v, output: %s", err, out)
    }
    return out, err
}

// Log возвращает историю коммитов
func (m *HistoryManager) Log(projectID string) (string, error) {
    log.Printf("Getting log for project %s", projectID)
    oneline := "--pretty=format:%h %ad %s"
    out, err := m.ExecGit(projectID, "log", oneline, "--
date=iso")
    if err != nil {
        log.Printf("git log failed: %v, output: %s", err, out)
    }
    return out, err
}

func GetDirectoryContent(projectID string) ([]File, error) {
    projectDir := fmt.Sprintf("/data/projects/%s", projectID)

    files, err := ioutil.ReadDir(projectDir)
    if err != nil {
        return nil, err
    }

    var fileList []File

    for _, file := range files {
        name := file.Name()

        // Пропускаем директории и игнорируемые файлы
        if file.IsDir() || ignoredNames[name] {

```

```

        continue
    }

    ext := strings.ToLower(filepath.Ext(name))
    filePath := filepath.Join(projectDir, name)

    content, err := ioutil.ReadFile(filePath)
    if err != nil {
        return nil, err
    }

    if !allowedExtensions[ext] || IsBinary(content) {
        // Не читаем содержимое бинарных или неразрешённых
        файлов
        fileList = append(fileList, File{
            Name:      name,
            IsBinary: true,
        })
    } else {
        fileList = append(fileList, File{
            Name:      name,
            Content:    string(content),
            IsBinary: false,
        })
    }
}

return fileList, nil
}

// ReturnFilesJSON возвращает JSON-представление списка файлов
func (m *HistoryManager) ReturnFilesJSON(projectID string)
(string, error) {
    files, err := GetDirectoryContent(projectID)

    if err != nil {
        return "", err
    }

    jsonData, err := json.MarshalIndent(files, "", " ")
    if err != nil {
        return "", err
    }

    return string(jsonData), nil
}

```

Листинг Б.3 – Файл utils.go

```

package history

func IsBinary(data []byte) bool {
    for _, b := range data {

```

```

        if b == 0 {
            return true
        }
    }
    return false
}

```

Листинг Б.4 – Файл update_file.go

```

package history

import (
    "fmt"
    "os"
    "path/filepath"
)

func (m *HistoryManager) UpdateFileContent(projectID,
relativePath, content string) error {
    path := filepath.Join("/data/projects", projectID,
relativePath)

    info, err := os.Stat(path)
    if err != nil {
        return fmt.Errorf("file not found: %v", err)
    }
    if info.IsDir() {
        return fmt.Errorf("cannot write to a directory")
    }

    // Записываем содержимое
    err = os.WriteFile(path, []byte(content), 0644)
    if err != nil {
        return fmt.Errorf("failed to write file content: %v", err)
    }

    // Проверим, что файл действительно записался
    confirm, err := os.ReadFile(path)
    if err != nil {
        return fmt.Errorf("failed to confirm written content: %v",
err)
    }
    if string(confirm) != content {
        return fmt.Errorf("content mismatch after write")
    }

    return nil
}

```

Листинг Б.5 – Файл switch_version.go

```

package history

```

```

import (
    "fmt"
    "os/exec"
    "strings"
)

func (m *HistoryManager) CheckoutCommitAsBranch(projectID,
commitHash string) error {
    projectDir := fmt.Sprintf("/data/projects/%s", projectID)
    tempBranch := fmt.Sprintf("temp-%s", projectID)

    // 1. Получить текущую ветку
    currentBranchCmd := exec.Command("git", "-C", projectDir,
"rev-parse", "--abbrev-ref", "HEAD")
    currentBranchOutput, err := currentBranchCmd.Output()
    if err != nil {
        return fmt.Errorf("failed to get current branch: %v", err)
    }
    currentBranch :=
strings.TrimSpace(string(currentBranchOutput))

    // 2. Если текущая ветка совпадает с tempBranch переключиться
на master
    if currentBranch == tempBranch {
        checkoutMasterCmd := exec.Command("git", "-C", projectDir,
"checkout", "master")
        if output, err := checkoutMasterCmd.CombinedOutput(); err
!= nil {
            return fmt.Errorf("failed to switch to master: %v,
output: %s", err, output)
        }
    }

    // 3. Удалить ветку, если она существует
    checkCmd := exec.Command("git", "-C", projectDir, "show-ref",
"--verify", "refs/heads/"+tempBranch)
    if err := checkCmd.Run(); err == nil {
        deleteCmd := exec.Command("git", "-C", projectDir,
"branch", "-D", tempBranch)
        if output, err := deleteCmd.CombinedOutput(); err != nil {
            return fmt.Errorf("failed to delete branch: %v,
output: %s", err, output)
        }
    }

    // 4. Создать временную ветку
    cmd := exec.Command("git", "-C", projectDir, "checkout", "-b",
tempBranch, commitHash)
    if output, err := cmd.CombinedOutput(); err != nil {
        return fmt.Errorf("failed to checkout commit: %v, output:
%s", err, output)
    }
}

```

```

    }

    return nil
}

```

Листинг Б.6 – Файл rename_file.go

```

package history

import (
    "fmt"
    "os"
    "path/filepath"
)

func (m *HistoryManager) RenamePath(projectID, oldRelativePath,
newRelativePath string) error {
    oldPath := filepath.Join("/data/projects", projectID,
oldRelativePath)
    newPath := filepath.Join("/data/projects", projectID,
newRelativePath)

    _, err := os.Stat(oldPath)
    if err != nil {
        return fmt.Errorf("source path does not exist: %v", err)
    }

    _, err = os.Stat(newPath)
    if err == nil {
        return fmt.Errorf("target path already exists")
    }

    return os.Rename(oldPath, newPath)
}

```

Листинг Б.7 – Файл read_commit.go

```

package history

import (
    "bytes"
    "fmt"
    "os/exec"
    "path/filepath"
)

func (m *HistoryManager) ReadFileFromCommit(projectID, commitHash,
relativePath string) (string, error) {
    repoPath := filepath.Join("/data/projects", projectID)
    fullGitPath := fmt.Sprintf("%s:%s", commitHash, relativePath)

    cmd := exec.Command("git", "-C", repoPath, "show",
fullGitPath)
}

```

```
var stdout, stderr bytes.Buffer
cmd.Stdout = &stdout
cmd.Stderr = &stderr
err := cmd.Run()
if err != nil {
    return "", fmt.Errorf("git show failed: %v\nstderr: %s",
err, stderr.String())
}

return stdout.String(), nil
```

Приложение В
(обязательное)
Листинг клиентской части приложения

Листинг В.1 – Файл CodeEditor.jsx

```
export const CodeEditor = () => {
  const [language, setLanguage] = useState("javascript");
  const { updateFileContent } = useProject();
  const [versions, setVersions] = useState<string[]>([]);
  // список текстов
  const [vIndex, setVIndex] = useState(0);
  // текущая позиция
  const [isHistoryMode, setIsHistoryMode] = useState(false);
  // edit / diff
  const [selectedFile, setSelectedFile] = useState<Folder |
null>(null);
  const [fileContent, setFileContent] = useState<string>("");

  const handleFileSelect = (file: Folder) => {
    if (file.isFile) {
      setSelectedFile(file);
      setFileContent(file.content || "");
      setLanguage(getLanguageFromFileName(file.name));
    }
  };

  const getLanguageFromFileName = (fileName: string): string =>
{
  const extension =
fileName.split('.').pop()?.toLowerCase();
  switch (extension) {
    case 'js':
    case 'jsx':
      return 'javascript';
    case 'ts':
    case 'tsx':
      return 'typescript';
    case 'py':
      return 'python';
    case 'java':
      return 'java';
    case 'c':
    case 'cpp':
      return 'cpp';
    default:
      return 'javascript';
  }
};

useEffect(() => {
  if (selectedFile) {
    setFileContent(selectedFile.content || "");
  }
}, [selectedFile]);

const handleEditorChange = (value: string | undefined) => {
```

```

        if (value !== undefined) {
            setFileContent(value);
            if (selectedFile) {
                updateFileContent(selectedFile.id, value);
            }
        }
    };

    return (
        <>
        <NavBar projectName="Название Проекта" userName="Иван
Иванов" />
        <Flex height="100vh">
            <Box w="20%" bg="gray.100" p={4} overflowY="auto">
                <FileExplorer onFileSelect={handleFileSelect}
            />

            </Box>

            <Box w="50%" p={4} display="flex"
flexDirection="column">
                {selectedFile && <FileNameDisplay
fileName={selectedFile.name} />}
                <Box h="50%">
                    <Button onClick={() =>
setIsHistoryMode((prev) => !prev)}>
                        {isHistoryMode ? "Выйти из режима
истории" : "Режим истории"}
                    </Button>
                    {isHistoryMode ? (
                        <HistoryControl
                            patches={[
                                ]}
                            fileName={"main.py"}
                        />
                    ) : (
                        <MonacoEditorWithComments
                            language={language}
                            fileContent={fileContent}
                            onChange={handleEditorChange}
                        />
                    )}
                </Box>
            </Box>

            <Box w="30%" bg="gray.100" color="black" p={4}
overflowY="auto">
                <Terminal projectId={2} />
            </Box>
        </Flex>
    </>

```

```
);
};
```

Листинг В.2 – Файл FileExplorer.jsx

```
type AddRenameModalProps = {
  isOpen: boolean;
  onClose: () => void;
  onSubmit: (name: string, isFile?: boolean) => void;
  type: 'add' | 'rename';
  parentId?: string | null;
  isFile?: boolean;
  renameData?: { id: string; currentName: string };
}

export default function FileExplorer({ onSelect }:
FileExplorerProps) {
  const { folders, addItem, renameItem } = useProject();
  const { isOpen: isAddModalOpen, onOpen: onAddModalOpen,
onClose: onAddModalClose } = useDisclosure();
  const { isOpen: isRenameModalOpen, onOpen: onRenameModalOpen,
onClose: onRenameModalClose } = useDisclosure();
  const [modalData, setModalData] = useState<{ parentId: string
| null; isFile: boolean } | null>(null);
  const [renameData, setRenameData] = useState<{ id: string;
currentName: string } | null>(null);

  const openAddModal = (parentId: string | null, isFile:
boolean) => {
    setModalData({ parentId, isFile });
    onAddModalOpen();
  }

  const openRenameModal = (id: string, currentName: string) => {
    setRenameData({ id, currentName });
    onRenameModalOpen();
  }

  return (
    /* Модальное окно для добавления папки или файла */
    <AddRenameModal
      isOpen={isAddModalOpen}
      onClose={() => {
        onAddModalClose();
        setModalData(null);
      }}
      onSubmit={(name, isFile) => {
        addItem(modalData?.parentId || null, name,
isFile);
        onAddModalClose(); // Закрываем модальное окно
        после добавления
        setModalData(null);
      }}
    />
  );
}
```

```

    }}
    type="add"
    parentId={modalData?.parentId || null}
    isFile={modalData?.isFile || false}
  />

  { /* Модальное окно для переименования */ }
  <AddRenameModal
    isOpen={isRenameModalOpen}
    onClose={() => {
      onRenameModalClose();
      setRenameData(null);
    }}
    onSubmit={(name) => {
      if (renameData) {
        renameItem(renameData.id, name);
      }
      onRenameModalClose(); // Закрываем модальное
окно после переименования
      setRenameData(null);
    }}
    type="rename"
    renameData={renameData}
  />
</Box>
)
}

```

Листинг В.3 – Файл HistoryControl.tsx

```

// MonacoDiffViewer.tsx
import { useState } from "react";
import { DiffEditor } from "@monaco-editor/react";
import {
  Box,
  Flex,
  IconButton,
  Slider,
  SliderTrack,
  SliderFilledTrack,
  SliderThumb,
  Text,
} from "@chakra-ui/react";
import { ChevronLeftIcon, ChevronRightIcon } from
"@heroicons/react/24/solid";

/* --- helper: превращаем git-patch в два текста --- */
function parseGitDiff(patch: string) {
  const lines = patch.split("\n");
  const original: string[] = [];
  const modified: string[] = [];

  lines.forEach((l) => {

```

```

    if (/^(diff |index |@@|--- |\+\+\+ )/.test(l)) return;

    if (l.startsWith("-")) original.push(l.slice(1));
    else if (l.startsWith("+")) modified.push(l.slice(1));
    else if (l.startsWith(" ")) {
        const v = l.slice(1);
        original.push(v);
        modified.push(v);
    }
});

return { original: original.join("\n"), modified:
modified.join("\n") };
}

/* ----- КОМПОНЕНТ ----- */
interface DiffViewerProps {
    patches: string[]; // список diff-строк от сервера
    fileName: string; // нужен для auto-detect языка
}

export default function MonacoDiffViewer({ patches, fileName }:
DiffViewerProps) {
    /* === local state: где сейчас находимся === */
    const [idx, setIdx] = useState(patches.length + 1); // по-
умолчанию последняя

    /* текущий diff → два текста */
    const { original, modified } = parseGitDiff(patches[idx]);

    /* язык для Monaco */
    const lang = fileName.endsWith(".py")
        ? "python"
        : fileName.endsWith(".js")
            ? "javascript"
            : fileName.endsWith(".ts")
                ? "typescript"
                : "plaintext";

    return (
        <Box height="100%" display="flex" flexDirection="column">
            {/* ----- сам DiffEditor ----- */}
            <Box flex="1">
                <DiffEditor
                    original={original}
                    modified={modified}
                    language={lang}
                    height="100%"
                    options={{
                        renderSideBySide: false,          // inline-
режим
                        readOnly: true,

```

```

        diffAlgorithm: "smart",
        renderIndicators: true,
      }}
    />
  </Box>

  {/* ----- мини-панель истории ----- */}
  <Flex
    align="center"
    px={3}
    py={2}
    bg="gray.50"
    borderTop="1px solid var(--chakra-colors-gray-
200) "
    gap={2}
  >
    <IconButton
      aria-label="prev"
      icon={<ChevronLeftIcon />}
      size="xs"
      variant="ghost"
      isDisabled={idx === 0}
      onClick={() => setIdx((i) => Math.max(i + 1,
0))}
      _focus={{ boxShadow: "none" }}
    />

    <Text fontSize="xs" whiteSpace="nowrap">
      {idx + 1} / {patches.length}
    </Text>

    <IconButton
      aria-label="next"
      icon={<ChevronRightIcon />}
      size="xs"
      variant="ghost"
      isDisabled={idx === patches.length + 1}
      onClick={() => setIdx((i) => Math.min(i + 1,
patches.length + 1))}
      _focus={{ boxShadow: "none" }}
    />

    {/* слайдер заполняет свободное место */}
    <Slider
      flex="1"
      size="sm"
      min={0}
      max={patches.length + 1}
      value={idx}
      onChange={setIdx}
    >
      <SliderTrack bg="gray.200">

```

```

        <SliderFilledTrack bg="blue.400" />
      </SliderTrack>
      <SliderThumb w={2} h={2} />
    </Slider>
  </Flex>
</Box>
);
}

```

Листинг В.4 – Файл Project.tsx

```

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import {
  Box,
  SimpleGrid,
  Heading,
  Button,
  useDisclosure,
  Modal,
  ModalOverlay,
  ModalContent,
  ModalHeader,
  ModalBody,
  ModalFooter,
  ModalCloseButton,
  FormControl,
  FormLabel,
  Input,
  Text,
  Flex,
  Divider,
  Link,
  AlertDialog,
  AlertDialogOverlay,
  AlertDialogContent,
  AlertDialogHeader,
  AlertDialogBody,
  AlertDialogFooter, Select
} from '@chakra-ui/react';

const LANGUAGES = {
  1: 'JavaScript',
  2: 'Python',
  3: 'C++',
  4: 'Java'
};

// Компонент карточки проекта с дополнительной справочной
информацией
// Компонент карточки проекта с дополнительной справочной
информацией
const ProjectCard = ({ project, onEdit, onDelete, onRun }) => {

```

```

return (
  <Box
    borderWidth="1px"
    borderRadius="md"
    padding="16px"
    background="white"
    boxShadow="sm"
    display="flex"
    flexDirection="column"
    justifyContent="space-between"
  >
    <Box mb={4}>
      <Heading size="md" mb={2}>{project.name}</Heading>
      <Text fontSize="sm" color="gray.600" mb={2}>
        {project.description || 'Нет описания'}
      </Text>
      <Text fontSize="sm" color="gray.500" mb={1}>
        Язык: {LANGUAGES[project.languageid] || 'Не
указан'}
      </Text>
      <Text fontSize="sm" color="gray.500" mb={1}>
        Создан: {new
Date(project.createdat).toLocaleString()}
      </Text>
      <Text fontSize="sm" color="gray.500" mb={1}>
        Последний запуск: {project.lastRunAt ? new
Date(project.lastRunAt).toLocaleString() : 'Никогда не
запускался'}
      </Text>
    </Box>
    <Flex justifyContent="flex-end" mt="auto">
      {/* Ссылки на редактирование, удаление и запуск
*/}
      <Link color="blue.500" mr={4} onClick={() =>
onEdit(project)}>Редактировать</Link>
      <Link color="red.500" mr={4} onClick={() =>
onDelete(project)}>Удалить</Link>
      <Link color="green.500" mr={4} onClick={() =>
onRun(project)}>Открыть</Link>
    </Flex>
  </Box>
);
};

// Модальное окно создания нового проекта
const CreateProjectModal = ({ isOpen, onClose, onCreate }) => {
  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [languageId, setLanguageId] = useState('');

  const handleCreate = () => {

```

```

        onCreate({ name, description, languageId:
parseInt(languageId) || null });
        setName('');
        setDescription('');
        setLanguageId('');
        onClose();
    };

    return (
        <Modal isOpen={isOpen} onClose={onClose} isCentered>
            <ModalOverlay />
            <ModalContent>
                <ModalHeader>Создать новый проект</ModalHeader>
                <ModalCloseButton />
                <ModalBody>
                    <FormControl mb={4}>
                        <FormLabel>Название проекта</FormLabel>
                        <Input value={name} onChange={(e) =>
setName(e.target.value)} placeholder="Введите название" />
                    </FormControl>
                    <FormControl mb={4}>
                        <FormLabel>Описание</FormLabel>
                        <Input value={description} onChange={(e)
=> setDescription(e.target.value)} placeholder="Введите описание"
/>
                    </FormControl>
                    <FormControl>
                        <FormLabel>Язык
программирования</FormLabel>
                        <Select
                            placeholder="Выберите язык"
                            value={languageId}
                            onChange={(e) =>
setLanguageId(e.target.value)}
                        >
                            {Object.entries(LANGUAGES).map(([id,
language]) => (
                                <option key={id} value={id}>
                                    {language}
                                </option>
                            ))}
                        </Select>
                    </FormControl>
                </ModalBody>
                <ModalFooter>
                    <Button variant="ghost" mr={3}
onClick={onClose}>
                        Отменить
                    </Button>
                    <Button colorScheme="blue"
onClick={handleCreate} isDisabled={!name.trim() || !languageId}>
                        Создать
                </ModalFooter>
            </ModalContent>
        </Modal>
    );

```

```

        </Button>
      </ModalFooter>
    </ModalContent>
  </Modal>
);
};

/ ();
};

const handleUpdate = (updatedProject) => {
  setProjects(projects.map(p => p.id === updatedProject.id ?
updatedProject : p));
};

const handleDeleteClick = (project) => {
  setSelectedProject(project);
  onDeleteOpen();
};

const handleDelete = (projectToDelete) => {
  setProjects(projects.filter(p => p.id !==
projectToDelete.id));
};

return (
  <Box maxW="1200px" mx="auto" p={6} bg="gray.50"
minH="100vh">
    <Flex alignItems="center" justifyContent="space-
between" mb={6}>
      <Heading size="lg">Мои проекты</Heading>
      <Button colorScheme="blue"
onClick={onCreateOpen}>Создать новый проект</Button>
    </Flex>
    <Divider mb={6} />
    <SimpleGrid columns={{ base: 1, md: 2, lg: 3 }}
spacing={6}>
      {projects.map(project => (
        <ProjectCard
          key={project.id}
          project={project}
          onEdit={handleEditClick}
          onDelete={handleDeleteClick}
          onRun={handleRun}
        />
      ))}
    </SimpleGrid>

    {/* Модальное окно для создания */}
    <CreateProjectModal
      isOpen={isCreateOpen}
      onClose={onCreateClose}

```

```

        onCreate={handleCreate}
      />

      {/* Модальное окно для редактирования */}
      {selectedProject && (
        <EditProjectModal
          isOpen={isEditOpen}
          onClose={() => { setSelectedProject(null);
onEditClose(); }}
          onSave={handleUpdate}
          project={selectedProject}
        />
      )}

      {/* Диалог для удаления */}
      {selectedProject && (
        <DeleteProjectAlert
          isOpen={isDeleteOpen}
          onClose={() => { setSelectedProject(null);
onDeleteClose(); }}
          onConfirm={handleDelete}
          project={selectedProject}
        />
      )}
    </Box>

```

Приложение Г
(рекомендуемое)
Атрибуты сущностей базы данных

Таблица Г.1 – Атрибуты отношения «Пользователь»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Id	Уникальный идентификатор пользователя	integer	10	1 – 9999999999	нет	PK	4
Логин	Имя пользователя для входа	varchar	20	a-Я a-Z 0 – 9	нет		slava
Пароль	Пароль пользователя	varchar	40	a-Я a-Z 0 – 9	нет		qwerty
Электронная почта	Адрес электронной почты	varchar	20	a-Я a-Z 0 – 9	нет		example@mail.ru
Имя	Имя пользователя	varchar	20	a-Я a-Z 0 – 9	нет		Арина
Фамилия	Фамилия пользователя	varchar	20	a-Я a-Z 0 – 9	нет		Иванова

Таблица Г.2 – Атрибуты отношения «Проект»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Id	Уникальный идентификатор проекта	integer	10	1 – 9999999999	нет	PK	4
UserId	Идентификатор пользователя	integer	10	1 – 9999999999	нет	FK	101

Продолжение таблицы Г.2

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
LanguageId	Идентификатор языка программирования	integer	10	1 – 9999999999	нет	FK	34
Название	Название проекта	varchar	255	a-Я a-Z 0 – 9	нет		PythonProject
Описание	Краткое название проекта	varchar	255	a-Я a-Z 0 – 9	да		Краткое описание о проекте
Время создания	Дата и время создания проекта	timestamp	8	01.01.2024 – 01.01.2030	нет		2024 – 12 – 13 14:00:00

Таблица Г.3 – Атрибуты отношения «Версия проекта»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Id	идентификатор версии проекта	integer	10	1 – 9999999999	нет	PK	4
Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Текст вывода	Результат запуска программного кода	varchar	255	a-Я a-Z 0 – 9	нет		Hello World!
Хеш	Версия проекта в Git	varchar	40	a-Z 0 – 9	нет		12cldpeowW
Время создания	Дата и время создания проекта	timestamp	8	01.01.2024 – 01.01.2030	нет		2024 – 12 – 13 14:00:00

Таблица Г.4 – Атрибуты отношения «Языки программирования»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Id	Уникальный идентификатор языка программирования	integer	10	1 – 9999999999	нет	PK	4
Название	Название языка программирования проекта	varchar	20	a-Я a-Z 0 – 9	нет		Python

Таблица Г.5 – Атрибуты отношения «Участник проекта»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Id	Уникальный идентификатор содержания проекта	integer	10	1 – 9999999999	нет	PK	4
ProjectId	Идентификатор проекта	integer	10	1 – 9999999999	нет	FK	101
UserId	Идентификатор пользователя	integer	10	1 – 9999999999	нет	FK	101
Роль	Роль пользователя в проекте	varchar	20	a-Я a-Z 0 – 9	нет		owner
Принят	Принял ли запрос на рецензирование пользователь	bit	-	-	нет		1
Время приглашения	Время отправления приглашения пользователя	timestamp	8	01.01.2024 – 01.01.2030	нет		2024 – 12 – 13 14:00:00

Таблица Г.6 – Атрибуты отношения «Отзыв»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Id	Уникальный идентификатор модуля	integer	10	1 – 9999999999	нет	PK	4
VersionId	Идентификатор версии	integer	10	1 – 9999999999	нет	FK	101
Project_Member_Id	Идентификатор участника проекта	integer	10	1 – 9999999999	нет	FK	101
Статус	Статус отзыва на проект	varchar	20	a-Я a-Z 0 – 9	да		success
Время создания	Дата и время создания проекта	timestamp	8	01.01.2024 – 01.01.2030	нет		2024 – 12 – 13 14:00:00

Таблица Г.7 – Атрибуты отношения «Комментарий»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Id	Уникальный идентификатор расширения	integer	10	1 – 9999999999	нет	PK	4
ReviewId	Идентификатор отзыва	integer	10	1 – 9999999999	нет	FK	101
Файл	Файл, который комментируют	varchar	255	a-Я a-Z 0 – 9	нет		main.py
Строка	Строка, в которой комментарий	integer	5	0 – 9	нет		14

Продолжение таблицы Г.7

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения null	Роль	Пример
Сообщение	Текст комментария	varchar	255	а-Я а-Z 0 – 9	нет		Плохо
Время создания	Дата и время создания проекта	timestamp	8	01.01.2024 – 01.01.2030	нет		2024 – 12 – 13 14:00:00

Приложение Д
(рекомендуемое)

Скрипт для создания объектов базы данных на языке SQL

```

CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  login VARCHAR(20) NOT NULL UNIQUE,
  password VARCHAR(40) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE
);

CREATE TABLE projects (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id),
  name VARCHAR(255) NOT NULL,
  description VARCHAR(255),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE project_members (
  id SERIAL PRIMARY KEY,
  project_id INTEGER NOT NULL REFERENCES projects(id),
  user_id INTEGER NOT NULL REFERENCES users(id),
  role VARCHAR(20),
  invited_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  accepted BOOLEAN DEFAULT FALSE
);

CREATE TABLE commits (
  id SERIAL PRIMARY KEY,
  project_id INTEGER NOT NULL REFERENCES projects(id),
  git_hash VARCHAR(40) NOT NULL,
  message VARCHAR(255),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE reviews (
  id SERIAL PRIMARY KEY,
  commit_id INTEGER NOT NULL REFERENCES commits(id),
  reviewer_id INTEGER NOT NULL REFERENCES users(id),
  summary VARCHAR(200),
  status VARCHAR(20),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE review_comments (
  id SERIAL PRIMARY KEY,
  review_id INTEGER NOT NULL REFERENCES reviews(id),
  file_path VARCHAR(255),
  line_number INTEGER,
  comment TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE programming_languages (
  id SERIAL PRIMARY KEY,
  name VARCHAR(20) NOT NULL UNIQUE
);

```

```
ALTER TABLE projects
ADD COLUMN programming_language_id INTEGER
REFERENCES programming_languages(id);;
```

Приложение Е
(рекомендуемое)
Листинг модульных тестов

Листинг Е.1 – Файл FileExplorer.test.jsx

```
import React from 'react';
import { render, screen, within } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import { describe, it, expect, vi } from 'vitest';
import FileExplorer from '../components/FileExplorer';
import { ProjectProvider } from '../contexts/ProjectsContext';

// Обертка для тестов с контекстом
const renderWithProvider = (ui) => {
  return render(<ProjectProvider>{ui}</ProjectProvider>);
};

describe('FileExplorer', () => {
  it('отображает кнопки для добавления папки и файла', async () => {
    renderWithProvider(<FileExplorer onFileSelect={() => {}} />);

    expect(await screen.findByLabelText('Добавить папку')).toBeInTheDocument();
    expect(await screen.findByLabelText('Добавить файл')).toBeInTheDocument();
  });

  it('вызывает onFileSelect при клике на файл', async () => {
    const onFileSelect = vi.fn();
    renderWithProvider(<FileExplorer onFileSelect={onFileSelect} />);
    const user = userEvent.setup();

    await user.click(screen.getByLabelText('Добавить файл'));
    const input = await screen.findByPlaceholderText('Название');
    await user.type(input, 'example.js');
    await user.click(screen.getByText('Добавить'));

    const file = await screen.findByText('example.js');
    await user.click(file);

    expect(onFileSelect).toHaveBeenCalledWith(expect.objectContaining({
      name: 'example.js'
    }));
  });
});
```

Листинг Е.2 – Файл FolderComponent.test.jsx

```
import { render, screen, fireEvent } from '@testing-library/react';
import { vi } from 'vitest';
import FolderComponent from '../components/FolderComponent.tsx';
import { ProjectContext } from '../contexts/ProjectsContext.tsx';
import userEvent from "@testing-library/user-event";
```

```

// Моковые данные для тестов
const mockFolder = {
  id: '1',
  name: 'Test Folder',
  isFile: false,
  folders: [],
};

const mockFile = {
  id: '2',
  name: 'Test File',
  isFile: true,
  folders: [],
};

describe('FolderComponent', () => {
  it('renders folder and file correctly', () => {
    const mockOnFileSelect = vi.fn();
    const mockOpenAddModal = vi.fn();
    const mockOpenRenameModal = vi.fn();

    // Рендерим компонент с папкой
    render(
      <ProjectContext.Provider value={{ deleteItem: vi.fn() }}>
        <FolderComponent
          folder={mockFolder}
          openAddModal={mockOpenAddModal}
          openRenameModal={mockOpenRenameModal}
          onFileSelect={mockOnFileSelect}
        />
      </ProjectContext.Provider>
    );
    // Проверка наличия кнопки "Удалить" в диалоге
    expect(screen.getByText('Удалить')).toBeInTheDocument();
  });
});

```

Листинг Е.3 – Файл LoginRegisterPage.test.jsx

```

import { render, screen, fireEvent } from "@testing-library/react";
import { MemoryRouter } from "react-router-dom";
import { ChakraProvider } from "@chakra-ui/react";
import { describe, it, expect, vi } from "vitest";
import LoginRegisterPage from "../components/LoginRegisterPage";

const mockNavigate = vi.fn();
const mockToast = vi.fn();

vi.mock("@/components/ui/use-toast", () => ({
  useToast: () => ({

```

```

        toast: mockToast, // Используем ОДИН mockToast
    )),
  ));

vi.mock("react-router-dom", async () => {
  const actual = await vi.importActual("react-router-dom");
  return {
    ...actual,
    useNavigate: () => mockNavigate,
  };
});

describe("LoginRegisterPage", () => {
  it("renders login form by default", () => {
    render(
      <MemoryRouter>
        <ChakraProvider>
          <LoginRegisterPage />
        </ChakraProvider>
      </MemoryRouter>
    );

    expect(screen.getByText("Вход в
аккаунт")).toBeInTheDocument();
    expect(screen.getByRole("button", { name: "Войти"
})).toBeInTheDocument();
  });

  it("switches to registration form when link is clicked", () =>
{
    render(
      <MemoryRouter>
        <ChakraProvider>
          <LoginRegisterPage />
        </ChakraProvider>
      </MemoryRouter>
    );

    fireEvent.click(screen.getByText("Зарегистрируйтесь"));

    expect(screen.getByText("Регистрация
аккаунта")).toBeInTheDocument();
    expect(screen.getByRole("button", { name:
"Зарегистрироваться" })).toBeInTheDocument();
  });
});

```

Листинг Е.4 – Файл Output.test.jsx

```

import { render, screen, fireEvent } from "@testing-
library/react";
import Output from "../components/Output.jsx";
import { ChakraProvider } from "@chakra-ui/react";

```

```

import { describe, it, expect, vi } from "vitest";

vi.mock("../api.js", () => ({
  executeCode: vi.fn(() => Promise.resolve({ run: { output:
    "Hello, World!\n", stderr: "" } })),
}));

describe("Output Component", () => {
  it("renders without crashing", () => {
    render(
      <ChakraProvider>
        <Output editorRef={{ current: { getValue: () =>
          "print('Hello, World!')\" } }} language="python" />
        </ChakraProvider>
    );
    expect(screen.getByText("Output")).toBeInTheDocument();
  });

  it("runs code and displays output", async () => {
    render(
      <ChakraProvider>
        <Output editorRef={{ current: { getValue: () =>
          "print('Hello, World!')\" } }} language="python" />
        </ChakraProvider>
    );

    fireEvent.click(screen.getByText("Run Code"));

    expect(await screen.findByText("Hello,
    World!")).toBeInTheDocument();
  });
});

```