

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

**Информационная система организации сбора
средств на проекты**

Студент

(подпись, дата)

Харитонов Андрей Андреевич
(ФИО полностью)

Руководитель

(подпись, дата)

Афонин А.Ю.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

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
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Харитонов Андрей Андреевич гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система организации сбора средств на проекты

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

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Афонин А.Ю.

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

1) Назначение: информационная система организации сбора средств на проекты.

2) Среда выполнения: Linux.

Языковые средства разработки: Golang, TypeScript, SQL, YAML, UML.

Инструментальные средства: VS Code, Docker, Minikube, PlantUML,

Visual Paradigm.

Технологии разработки: процедурная парадигма, микросервисная архитектура.

3) Основные функции разрабатываемого приложения:

а) регистрация и авторизация;

б) просмотр проектов и хода их сбора;

в) создание, редактирование и публикация проектов;

г) модерация проектов перед публикацией;

д) осуществление взносов в проекты;

е) продвижение проектов;

ж) изменение ролей и блокировка пользователей.

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;

2) Проектирование;

3) Реализация;

4) Контроль качества программного обеспечения;

5) Оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

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 112 листов, 39 рисунков, 7 таблиц, 46 использованных источников и 9 приложений.

КРАУДФАНДИНГ, ИНФОРМАЦИОННАЯ СИСТЕМА, ВЕБ-ПРИЛОЖЕНИЕ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, GOLANG, TYPESCRIPT, SVELTEKIT, UML, KUBERNETES, POSTGRESQL, KAFKA, REDIS, MINIO, PROMETHEUS, GRAFANA.

Объект исследования – информационные системы организации коллективного сбора средств на проекты.

Цель выпускной квалификационной работы – разработка информационной системы для организации коллективного сбора средств на проекты.

Технологии разработки – клиентская часть приложения разрабатывается на языке программирования TypeScript с использованием фреймворка SvelteKit, а серверная часть – на языке программирования Golang. В качестве редактора кода используется VS Code. Развёртывание приложения выполняется в контейнерной среде с помощью оркестратора Kubernetes.

Среда исполнения – ОС Ubuntu 24.04 LTS, веб-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были разработаны клиентская и серверная части информационной системы организации сбора средств на проекты.

					ПГУ 09.03.04 – 08BP222.30 ПЗ						
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система организации сбора средств на проекты. Пояснительная записка			Лит.	Лист	Листов	
Разраб.		Харитонов А.А.									
Провер.		Афонин А.Ю.							4	112	
Реценз.								Группа 22ВП2			
Н. Контр.		Такташкин Д.В.									
Утв.		Козлов А.Ю.									

Содержание

Введение.....	8
1 Анализ предметной области и постановка задачи на разработку.....	9
1.1 Анализ предметной области	9
1.2 Анализ аналогичных решений.....	9
1.3 Постановка задачи на разработку	13
1.4 Анализ требований к разрабатываемой системе	15
1.4.1 Анализ функциональных требований.....	15
1.4.2 Анализ нефункциональных требований.....	18
1.5 Планирование разработки и оценка бюджета.....	19
2 Проектирование информационной системы	22
2.1 Архитектура системы	22
2.2 Взаимодействие компонентов системы.....	25
2.3 Проектирование модели данных	28
2.3.1 Концептуальная модель данных	28
2.3.2 Логическая модель данных.....	30
2.4 Проектирование серверной части	32
2.5 Проектирование клиентской части	33
3 Реализация информационной системы	36
3.1 Физическая модель данных	36
3.2 Реализация серверной части	38
3.2.1 Реализация микросервиса пользователей.....	38
3.2.2 Реализация микросервиса проектов.....	39
3.2.3 Реализация микросервиса платежей	40
3.2.4 Реализация микросервиса промокодов.....	42

	6
3.3 Реализация клиентского приложения	42
3.3.1 Структура клиентского приложения	42
3.3.2 Интерфейс клиентского приложения	44
3.4 Развертывание системы.....	50
4 Контроль качества программного обеспечения.....	56
4.1 Обоснование выбора модели тестирования и метрик кода.....	56
4.2 Функциональное тестирование создания проекта.....	57
4.3 Нагрузочное тестирование.....	58
4.4 Метрики кода.....	59
4.4.1 Размер кода.....	59
4.4.2 Цикломатическая сложность	60
4.4.3 Метрики Холстеда	61
Заключение	65
Список использованных источников	67
Приложение А (справочное) Глоссарий	72
Приложение Б (обязательное) Скрипты для создания таблиц баз данных	74
Приложение В (обязательное) Фрагменты исходного кода микросервиса пользователей	82
Приложение Г (обязательное) Фрагменты исходного кода микросервиса проектов	86
Приложение Д (обязательное) Фрагменты исходного кода микросервиса платежей.....	91
Приложение Е (обязательное) Фрагменты исходного кода микросервиса промокодов	94
Приложение Ж (обязательное) Фрагменты исходного кода клиентского приложения	96

Приложение И (рекомендуемое) Диаграммы классов микросервисов	100
Приложение К (рекомендуемое) Тест-кейсы сценария создания проекта	104

Введение

В настоящее время привлечение средств на реализацию проектов всё чаще связано с использованием цифровых сервисов, позволяющих объединить авторов инициатив и заинтересованных лиц на одной платформе. Информационные системы, предназначенные для организации такого сбора средств, обычно реализуются в формате краудфандинговых платформ. Краудфандинг определяется как способ привлечения средств для финансирования проектов частных лиц и организаций от неограниченного круга лиц при посредстве специальных интернет-площадок [1]. Развитие данного направления подтверждается данными Банка России: объем рынка краудфандинга в 2023 году составил 33,4 млрд руб., а в I квартале 2024 – 11,1 млрд руб. [2].

Целью данной работы является создание информационной системы организации сбора средств на проекты. Основная задача системы – дать пользователям возможность публиковать проекты и осуществлять взносы в их поддержку.

Для достижения поставленной цели необходимо:

- провести анализ предметной области и аналогичных решений;
- определить требования к разрабатываемой системе;
- спроектировать архитектуру приложения и модели данных;
- реализовать приложение;
- провести контроль качества разработанного приложения.

В результате выполнения работы должна быть получена информационная система, обеспечивающая организацию сбора средств на проекты.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области

Предметная область данной работы связана с организацией сбора средств на реализацию проектов с помощью онлайн-платформ при поддержке заинтересованных лиц. В цифровой среде специализированные интернет-площадки, обеспечивающие реализацию данной модели, рассматриваются как краудфандинговые платформы.

Краудфандинг представляет собой способ привлечения средств для финансирования проектов частных лиц и организаций от неограниченного круга частных лиц и организаций посредством специальных интернет-площадок [1].

Основными участниками предметной области являются гость, зарегистрированный пользователь, модератор и администратор. Гость имеет доступ к открытой информации о проектах и пользователях. Зарегистрированный пользователь может работать с личным профилем, создавать проекты и поддерживать инициативы других пользователей. Модератор рассматривает заявки на публикацию проектов и принимает решение об их одобрении. Администратор управляет ролями пользователей и контролирует доступ к системе.

В рамках предметной области основными объектами являются пользователи, проекты, заявки на публикацию, взносы и выплаты. Основной процесс работы системы связан с созданием проекта автором, проверкой проекта перед публикацией, размещением проекта на платформе и дальнейшим сбором средств от заинтересованных пользователей.

1.2 Анализ аналогичных решений

Анализ аналогичных решений позволяет определить их возможности, выявить преимущества и недостатки, а также выделить особенности, которые следует учесть при разработке собственного приложения.

Planeta.ru – краудфандинговая платформа, на которой авторы собирают средства на реализацию своих проектов [3]. Пользовательский интерфейс сервиса представлен на рисунке 1.

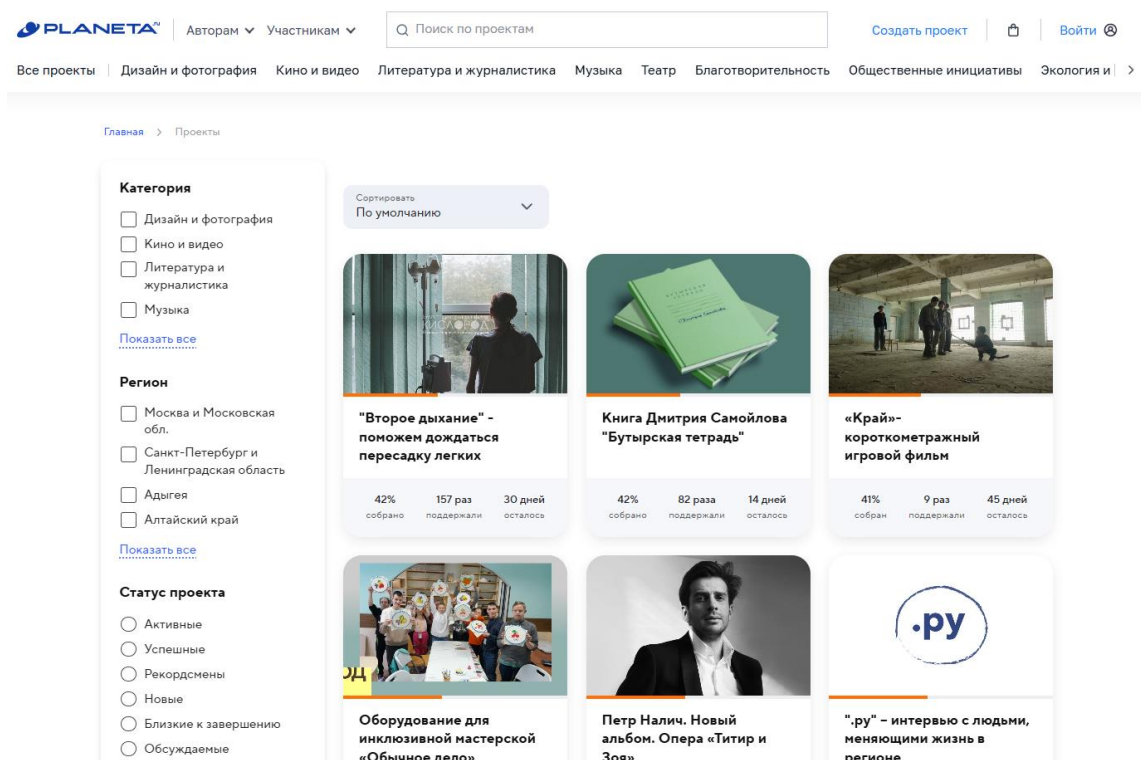


Рисунок 1 – Пользовательский интерфейс Planeta.ru

К основным преимуществам Planeta.ru можно отнести:

- большое количество обучающих и справочных материалов для авторов;
- возможность автора получить средства уже при достижении 50% от заявленной суммы;
- экосистема сервисов, благодаря которой автор может не только собрать средства, но и получить помощь с реализацией продукта;
- удобный и современный пользовательский интерфейс.

Основным недостатком Planeta.ru является отсутствие встроенного механизма, позволяющего автору самостоятельно повысить видимость своего проекта в рамках системы.

Kickstarter – платформа для финансирования проектов [4]. Пользовательский интерфейс представлен на рисунке 2.

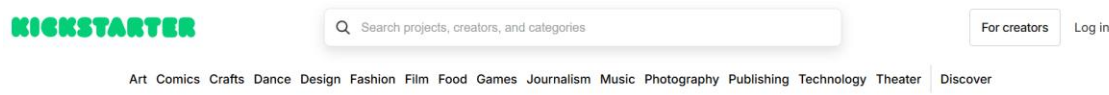


Рисунок 2 – Пользовательский интерфейс Kickstarter

К основным преимуществам Kickstarter можно отнести:

- развитую систему аналитики для авторов проектов;
- современный и понятный пользовательский интерфейс.

Явным недостатком Kickstarter является отсутствие России в перечне стран, для которых поддерживается запуск проектов [5].

Boomstarter – сайт, созданный для привлечения финансирования на реализацию проектов [6]. Пользовательский интерфейс сервиса представлен на рисунке 3.

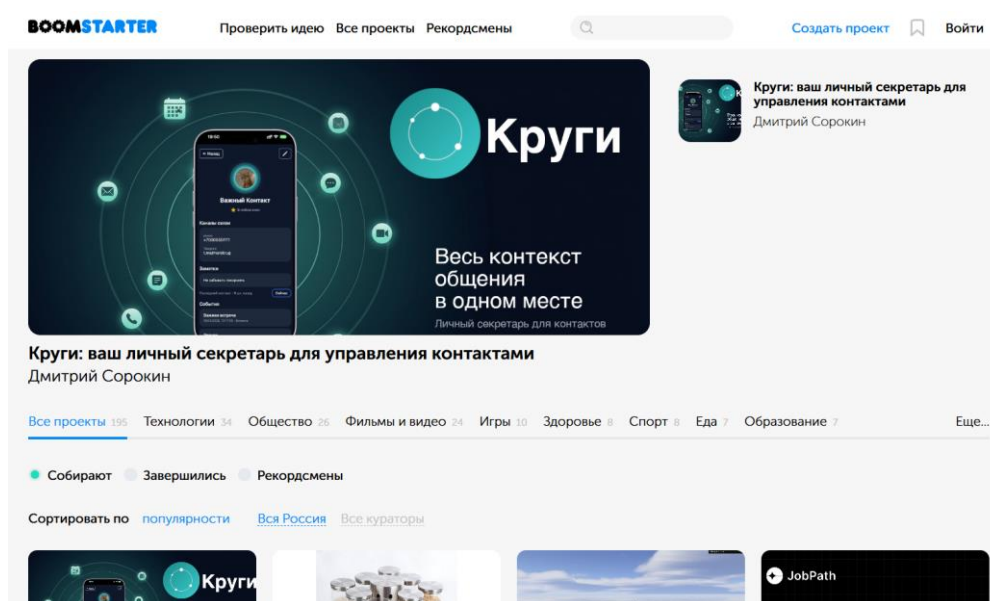


Рисунок 3 – Пользовательский интерфейс Boomstarter

К основным преимуществам Boomstarter можно отнести:

- сопровождение проекта менеджером при подготовке к запуску;
- отсутствие комиссии самой платформы за успешный сбор.

Недостатками Boomstarter являются:

- необходимость оплаты тарифного пакета для запуска проекта на платформе, из-за чего автор несёт расходы ещё до получения результата компании;
- устаревший и недоработанный в ряде разделов пользовательский интерфейс.

GoFundMe – платформа для краудфандинга и сбора средств [7]. Пользовательский интерфейс сервиса представлен на рисунке 4.

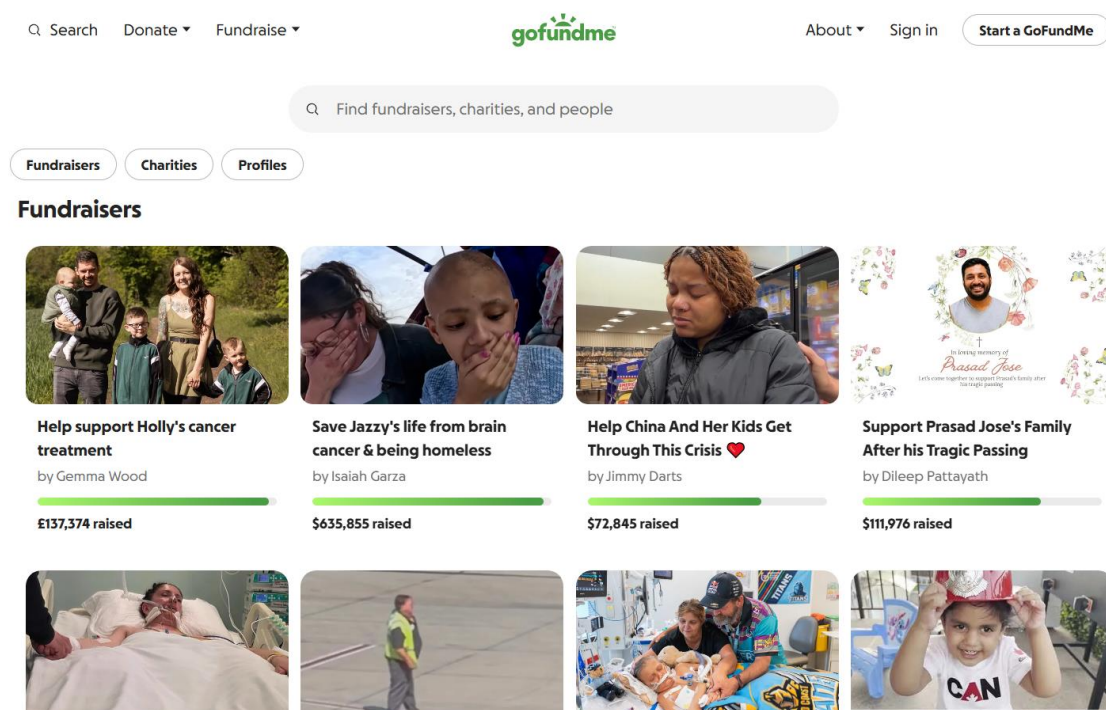


Рисунок 4 – Пользовательский интерфейс GoFundMe

К основным преимуществам GoFundMe можно отнести:

- возможность получить собранные средства даже при неполном достижении цели сбора;
- право автора самостоятельно решать, когда закрыть кампанию сбора средств;

– удобный и современный пользовательский интерфейс.

Существенным недостатком GoFundMe является отсутствие России в перечне стран, в которых поддерживается создание кампаний по сбору средств на проекты [8].

Для наглядного сравнения рассмотренных аналогов по выделенным критериям результаты анализа сведены в таблицу 1.

Таблица 1 – Сравнительный анализ аналогичных программных средств

Критерий	Planeta.ru	Boomstarter	Kickstarter	GoFundMe
Удобный и понятный интерфейс	Есть	Частично	Есть	Есть
Возможность размещения проекта для сбора средств	Есть	Есть	Есть	Есть
Проверка проекта перед публикацией	Есть	Есть	Есть	Есть
Возможность внесения средств в проект	Есть	Есть	Есть	Есть
Доступность запуска проекта для российских авторов	Есть	Есть	Нет	Нет
Возможность продвижения проекта автором	Нет	Нет	Нет	Нет

Таким образом, по результатам сравнения аналогичных решений определено, что ни одно из рассмотренных программных средств в полной мере не соответствует выделенным критериям, что подтверждает актуальность разработки собственной системы.

1.3 Постановка задачи на разработку

По результатам анализа предметной области требуется реализовать информационную систему организации сбора средств на проекты, обеспечивающую пользователям возможность как публиковать свои проекты, так и осуществлять взносы для поддержки инициатив других участников.

В разрабатываемой системе предусматриваются четыре роли: гость, пользователь, модератор и администратор.

Возможности гостя:

- просмотр проектов;
- просмотр профилей пользователей;
- регистрация и вход в систему.

Возможности пользователя:

- создание проектов и их редактирование;
- отправка проектов модерацию;
- просмотр статистики собственных проектов;
- осуществление взносов в проекты;
- продвижение собственных проектов;
- просмотр статуса продвижения своих проектов;
- просмотр статуса выплат собственных проектов;
- редактирование личного профиля;
- подписка на других пользователей.

Возможности модератора:

- просмотр проектов перед публикацией;
- одобрение и отклонение публикации проектов.

Возможности администратора:

- изменение ролей пользователей;
- блокировка пользователей.

Информационная система должна предоставлять удобный пользовательский интерфейс с понятной и логичной структурой и навигацией, не требующий от пользователя специальных знаний для работы с системой. Выполняемые действия должны сопровождаться понятной обратной связью, а при некорректном вводе данных пользователю должны выводиться информативные сообщения об ошибках.

Таким образом, результатом выполнения работы должна стать информационная система, соответствующая поставленной задаче.

1.4 Анализ требований к разрабатываемой системе

1.4.1 Анализ функциональных требований

Для разрабатываемой информационной системы выделены следующие основные функциональные требования:

- регистрация, вход в учётную запись и выход из неё;
- просмотр ленты проектов с применением фильтрации, сортировки и поиска;
- просмотр проекта;
- осуществление взноса в проект;
- просмотр профиля пользователя и созданных им проектов;
- редактирование собственного профиля;
- подписка на другого пользователя;
- создание проекта;
- редактирование собственного проекта;
- отправка собственного проекта на модерацию;
- продвижение собственного проекта в ленте посредством применения промокода;
- просмотр собственных проектов с информацией о статистике, статусах выплаты и продвижения;
- просмотр заявок на публикацию проектов;
- одобрение и отклонение заявок на публикацию проектов;
- изменение ролей пользователей;
- блокировка пользователей.

В системе должно быть предусмотрено четыре роли пользователей: гость, пользователь, модератор и администратор.

На основе анализа функциональных требований построена UML-диаграмма вариантов использования. Такой тип диаграмм применяется для описания ожидаемого поведения системы с точки зрения пользователя: он показывает, какие действующие лица взаимодействуют с системой и какие

функции им доступны [9]. Диаграмма вариантов использования разрабатываемой системы представлена на рисунке 5.

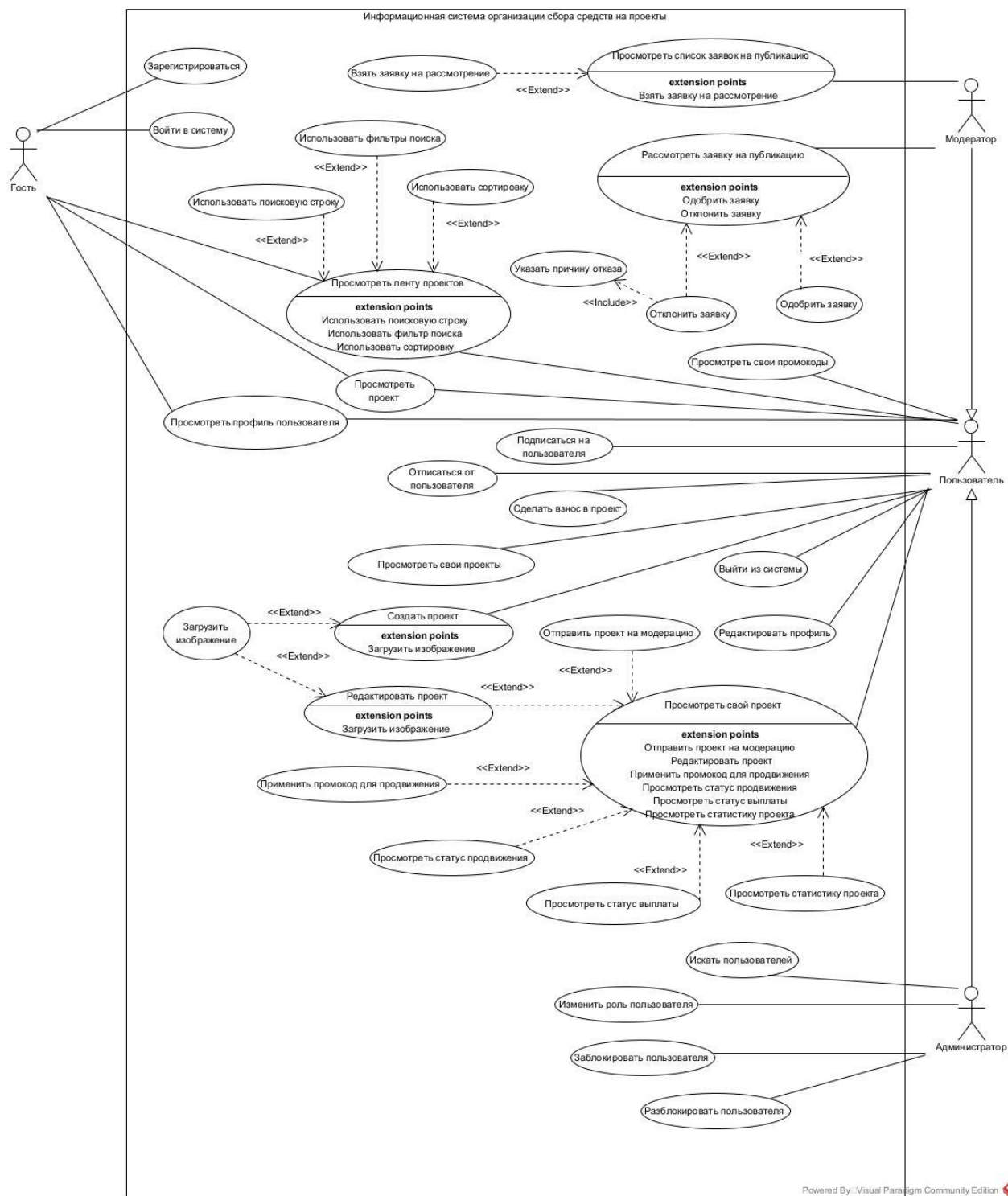


Рисунок 5 – Диаграмма вариантов использования

На диаграмме показана связь между ролями пользователей и доступными им действиями в системе. Роли модератора и администратора наследуют возможности пользователя и получают доступ к дополнительным функциям.

Ниже представлены описания наиболее важных вариантов использования разрабатываемой системы.

На рисунке 6 представлена спецификация варианта использования «Создать проект».

Наименование	Создать проект
ID	1
Краткое описание	Пользователь создает проект
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловия	1. Пользователь авторизован в системе.
Основной поток	1. Пользователь переходит к форме создания проекта. 2. Пользователь заполняет данные проекта. 3. Пользователь отправляет форму создания проекта. 4. Система проверяет корректность данных. 5. Система проверяет отсутствие у пользователя проекта с таким же названием. 6. Система создает проект со статусом «черновик». 7. Система перенаправляет пользователя на страницу созданного проекта.
Постусловия	1. В системе создан проект со статусом «черновик».
Альтернативные потоки	4а. Система сообщает о некорректности данных. 5а. Система сообщает о невозможности создания проекта с уже существующим названием.

Рисунок 6 – Спецификация варианта использования «Создать проект»

На рисунке 7 представлена спецификация варианта использования «Сделать взнос в проект».

Наименование	Сделать взнос в проект
ID	2
Краткое описание	Пользователь осуществляет взнос в проект на указанную сумму
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловия	1. Пользователь авторизован в системе. 2. Проект активен.
Основной поток	1. Пользователь открывает страницу проекта. 2. Система отображает форму внесения взноса. 3. Пользователь указывает сумму взноса. 4. Пользователь подтверждает внесение взноса. 5. Система проверяет корректность указанной суммы. 6. Система сохраняет сведения о взносе. 7. Система сообщает пользователю об успешном внесении взноса.
Постусловия	1. В системе сохранены сведения о взносе пользователя. 2. Сумма собранных средств по проекту обновляется после обработки сведений о взносе.
Альтернативные потоки	5а. Система сообщает о некорректной сумме взноса.

Рисунок 7 – Спецификация варианта использования «Сделать взнос в проект»

На рисунке 8 представлена спецификация варианта использования «Применить промокод для продвижения проекта».

Наименование	Применить промокод для продвижения проекта
ID	3
Краткое описание	Пользователь применяет промокод для продвижения собственного проекта в ленте
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловия	1. Пользователь авторизован в системе. 2. Пользователь является автором проекта. 3. Проект является активным.
Основной поток	1. Пользователь открывает страницу собственного проекта. 2. Пользователь переходит во вкладку продвижения проекта. 3. Система отображает форму ввода промокода. 4. Пользователь вводит промокод. 5. Пользователь подтверждает применение промокода. 6. Система проверяет корректность введенного промокода. 7. Система проверяет существование промокода. 8. Система применяет промокод. 9. Система отображает сообщение об успешном применении промокода.
Постусловия	1. Промокод использован. 2. Проект получает продвижение в ленте.
Альтернативные потоки	6а. Система сообщает о некорректном формате промокода. 7а. Система сообщает, что промокод не найден.

Рисунок 8 – Спецификация варианта использования «Применить промокод для продвижения проекта»

Таким образом, были определены функциональные требования к разрабатываемой системе и представлены спецификации наиболее важных вариантов использования.

1.4.2 Анализ нефункциональных требований

Для разрабатываемой системы определены нефункциональные требования, отражающие свойства, которыми должна обладать система. К ним относятся производительность, удобство и простота использования, надежность, безопасность и сопровождаемость.

Производительность характеризует работу системы при заданной нагрузке и может оцениваться по временным характеристикам, потреблению ресурсов и

уровню загруженности [10]. Для разрабатываемого приложения это свойство важно, так как при увеличении объема данных и нагрузки система должна сохранять стабильную работу без заметного ухудшения качества обслуживания пользователей.

Удобство и простота использования характеризуют, насколько просто пользователю ориентироваться в интерфейсе, осваивать возможности системы и применять их для достижения своих целей [10]. Для разрабатываемого приложения это свойство является важным, поскольку пользователи должны без лишних затруднений выполнять действия в нём. Сложный интерфейс снижает вовлеченность пользователей и делает платформу менее эффективной.

Надежность характеризует способность системы сохранять работоспособность в заданных условиях в течение определенного времени [10]. Данное свойство важно для разрабатываемого приложения, так как система должна сохранять корректное состояние при ошибках и сбоях.

Безопасность отражает способность системы защищать информацию и управлять доступом к ней в соответствии с правами пользователей [10]. Для разрабатываемого приложения данное свойство актуально, так как в системе используются учетные записи, авторизация и разделение ролей. Нарушение безопасности может привести к утечке данных или их несанкционированному изменению.

Сопровождаемость характеризует, насколько просто анализировать, модифицировать и тестировать программное обеспечение [10]. Для разрабатываемой системы это свойство важно, так как в процессе развития платформы может потребоваться добавление новых возможностей и модификация реализованного приложения.

1.5 Планирование разработки и оценка бюджета

Планирование разработки программного продукта позволяет заранее определить порядок выполнения работ, оценить сроки реализации и подготовить

данные для расчета бюджета. В рамках данной выпускной квалификационной работы для этих целей используется программное средство Microsoft Project.

На первом этапе были определены ресурсы, необходимые для выполнения проекта. В рамках работы основным и единственным трудовым ресурсом является разработчик. Заданные для него параметры в Microsoft Project представлены на рисунке 9.

Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполыз.	Начисление	Базовый календарь
Разработчик	Трудовой		P		100%	300,00 P/ч	300,00 P/ч	0,00 P	Пропорциональное	Стандартный

Рисунок 9 – Ресурсы проекта

После определения ресурсов был составлен план разработки программного продукта: выделены основные задачи проекта, определена продолжительность работ и рассчитана стоимость разработки. План проекта представлен на рисунке 10.

1	Название задачи	Длительность	Начало	Окончание	Затраты	Предшественники	Названия ресурсов
1	Разработка информационной системы	41 дней?	Вт 07.04.26	Вт 02.06.26	98 400,00 P		
2	Начало проекта	0 дней?	Вт 07.04.26	Вт 07.04.26	0,00 P		
3	Анализ предметной области и постановка задачи на разработку	5 дней?	Вт 07.04.26	Пн 13.04.26	12 000,00 P		
4	Анализ предметной области	1 день	Вт 07.04.26	Вт 07.04.26	2 400,00 P	2	Разработчик
5	Анализ аналогичных решений	1 день	Ср 08.04.26	Ср 08.04.26	2 400,00 P	4	Разработчик
6	Постановка задачи на разработку	1 день	Чт 09.04.26	Чт 09.04.26	2 400,00 P	5	Разработчик
7	Анализ требований	2 дней	Пт 10.04.26	Пн 13.04.26	4 800,00 P	6	Разработчик
8	Конец анализа предметной области и постановки задачи на разработку	0 дней?	Пн 13.04.26	Пн 13.04.26	0,00 P	7	
9	Проектирование информационной системы	7 дней?	Вт 14.04.26	Ср 22.04.26	16 800,00 P		
10	Проектирование архитектуры системы	2 дней	Вт 14.04.26	Ср 15.04.26	4 800,00 P	8	Разработчик
11	Проектирование концептуальной модели данных	1 день	Чт 16.04.26	Чт 16.04.26	2 400,00 P	10	Разработчик
12	Проектирование логической модели данных	1 день	Пт 17.04.26	Пт 17.04.26	2 400,00 P	11	Разработчик
13	Проектирование серверной части	2 дней	Пн 20.04.26	Вт 21.04.26	4 800,00 P	12	Разработчик
14	Проектирование пользовательского интерфейса	1 день	Ср 22.04.26	Ср 22.04.26	2 400,00 P	13	Разработчик
15	Конец проектирования информационной системы	0 дней?	Ср 22.04.26	Ср 22.04.26	0,00 P	14	
16	Реализация информационной системы	16 дней?	Чт 23.04.26	Чт 14.05.26	38 400,00 P		
17	Реализация физической модели данных	2 дней	Чт 23.04.26	Пт 24.04.26	4 800,00 P	15	Разработчик
18	Реализация серверной части	8 дней	Пн 27.04.26	Ср 06.05.26	19 200,00 P	17	Разработчик
19	Реализация клиентской части	4 дней	Чт 07.05.26	Вт 12.05.26	9 600,00 P	18	Разработчик
20	Развертывание приложения	2 дней	Ср 13.05.26	Чт 14.05.26	4 800,00 P	19	Разработчик
21	Конец реализации информационной системы	0 дней?	Чт 14.05.26	Чт 14.05.26	0,00 P	20	
22	Контроль качества информационной системы	3 дней	Пт 15.05.26	Вт 19.05.26	7 200,00 P		
23	Функциональное тестирование	1 день	Пт 15.05.26	Пт 15.05.26	2 400,00 P	21	Разработчик
24	Нагрузочное тестирование	1 день	Пн 18.05.26	Пн 18.05.26	2 400,00 P	23	Разработчик
25	Метрики программного кода	1 день	Вт 19.05.26	Вт 19.05.26	2 400,00 P	24	Разработчик
26	Конец контроля качества	0 дней	Вт 19.05.26	Вт 19.05.26	0,00 P	25	
27	Составление отчета	10 дней	Ср 20.05.26	Вт 02.06.26	24 000,00 P	26	Разработчик
28	Конец проекта	0 дней?	Вт 02.06.26	Вт 02.06.26	0,00 P	27	

Рисунок 10 – План проекта

На основе составленного плана проекта была построена диаграмма Ганта. Она позволяет представить задачи разработки на временной шкале, отразить сроки их выполнения и взаимосвязь между этапами работ [11]. Диаграмма Ганта представлена на рисунке 11.

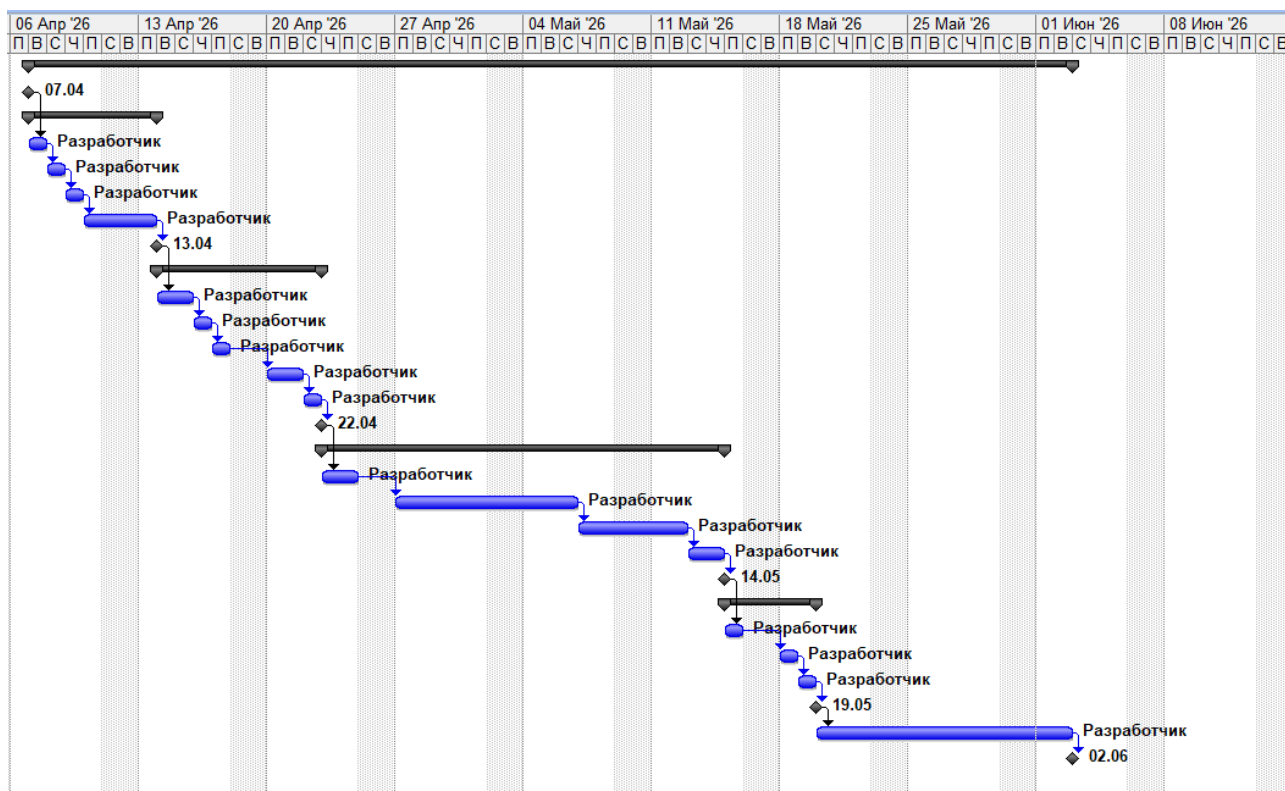


Рисунок 11 – Диаграмма Ганта проекта

По результатам планирования общая продолжительность проекта составила 328 часа, а итоговая стоимость разработки – 98 400 руб.

2 Проектирование информационной системы

2.1 Архитектура системы

В качестве архитектурного подхода для разрабатываемой системы выбрана клиент-серверная архитектура с микросервисной организацией серверной части.

В микросервисной архитектуре приложение разделяется на набор небольших независимо развертываемых сервисов, каждый из которых связан с отдельной бизнес-функцией системы и взаимодействует с другими сервисами через сетевые механизмы обмена данными [12]. Благодаря такому разделению изменение в одном сервисе требует повторного развёртывания только этого сервиса, а не всего приложения, что упрощает и ускоряет процесс выпуска обновлений [12]. Также важной характеристикой данного подхода является децентрализованное управление данными – каждый сервис управляет собственным хранилищем, при этом могут использоваться как разные экземпляры одной СУБД, так и принципиально разные системы хранения [12].

Микросервисная архитектура подходит для информационной системы организации сбора средств на проекты, так как функциональность разрабатываемого приложения предметно разделяется на отдельные компоненты, которые могут развиваться независимо друг от друга. Такой подход упрощает сопровождение приложения: при грамотном разбиении системы изменения в одном микросервисе в меньшей степени затронут остальные части системы. Кроме того, микросервисный стиль позволяет независимо масштабировать отдельные сервисы, что помогает поддерживать производительность приложения и повышает его надежность.

Для представления архитектуры приложения была разработана UML-диаграмма компонентов. Такой тип диаграмм используется для отображения архитектуры программных компонентов и зависимостей между ними [13]. Диаграмма компонентов информационной системы организации сбора средств на проекты представлена на рисунке 12.

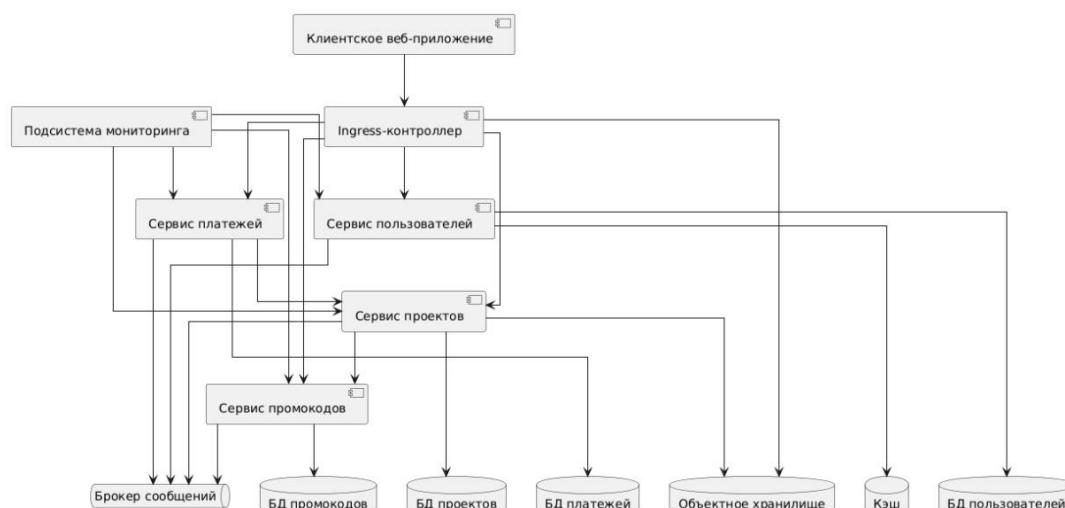


Рисунок 12 – Диаграмма компонентов

На диаграмме представлены клиентское веб-приложение, микросервисы, ingress-контроллер, хранилища данных, брокер сообщений и подсистема мониторинга.

Клиентское веб-приложение отвечает за пользовательский интерфейс. От него запросы к серверной части сначала поступают в ingress-контроллер, после чего перенаправляются к серверным компонентам.

Каждый микросервис отвечает за отдельную предметную область:

- сервис пользователей – учётные и публичные данные пользователей, а также регистрация и аутентификация;
- сервис проектов – проекты и заявки на их публикацию;
- сервис платежей – взносы и выплаты по проектам;
- сервис промокодов – информация о промокодах.

Ingress-контроллер используется для приема внешних HTTP-запросов и их маршрутизации к соответствующим компонентам системы. В качестве него выбран Traefik – обратный прокси-сервер и ingress-контроллер с открытым исходным кодом [14]. Его выбор обусловлен возможностью настройки правил маршрутизации входящих запросов, а также поддержкой динамического обновления конфигурации без прерывания обработки запросов и потери соединения [15, 16].

Для хранения структурированных данных микросервисов используются реляционные базы данных. В качестве СУБД выбрана PostgreSQL, поддерживающая транзакции и механизмы обеспечения целостности данных [17]. В рамках выбранной архитектуры каждый микросервис работает со своей базой данных, не обращая напрямую к данным других сервисов.

Кэш в разрабатываемом приложении используется для хранения часто запрашиваемых данных и снижения нагрузки на основную базу данных. В качестве него выбран Redis – хранилище данных в памяти, поддерживающее кэширование [18].

Объектное хранилище используется для хранения файловых данных. В качестве него выбран MinIO – решение с поддержкой S3-совместимого интерфейса, позволяющего работать с файлами через распространенный API объектного хранения [19]. Это удобно для разрабатываемой системы, поскольку файлы хранятся отдельно от реляционных баз данных, не смешиваясь со структурированными данными сервисов.

Брокер сообщений используется для асинхронного обмена событиями между микросервисами. В разрабатываемой системе часть действий может передаваться другим компонентам как событие без синхронной связи между сервисами. В качестве брокера сообщений выбрана Kafka – она позволяет публиковать и читать потоки событий, надёжно хранить их, а также обрабатывать по мере поступления [20]. Кроме того, в Kafka производители и потребители событий не зависят друг от друга, что важно для микросервисной архитектуры [20].

Подсистема мониторинга используется для сбора и отображения метрик работы компонентов. В качестве средства сбора метрик выбран Prometheus, поскольку он хранит метрики в виде временных рядов, поддерживает сбор данных по HTTP и подходит для мониторинга систем, состоящих из множества сервисов [21]. Prometheus позволяет получать метрики из приложений через клиентские библиотеки и из сторонних систем через экспортёры [22, 23]. Для визуализации собранных метрик удобно использовать Grafana, которая

поддерживает Prometheus как источник данных и позволяет запрашивать, визуализировать и анализировать данные мониторинга [24, 25].

Описанная архитектура может быть размещена в контейнерной среде. В качестве платформы для развёртывания выбран Kubernetes, так как он предназначен для управления контейнеризированными приложениями, а также поддерживает их масштабирование и восстановление при сбоях [26].

2.2 Взаимодействие компонентов системы

Диаграммы последовательности применяются для описания выполнения операций и обмена сообщениями между объектами с учетом порядка их взаимодействия [27]. Далее эти диаграммы представлены для основных вариантов использования системы.

На рисунке 13 изображена диаграмма последовательности процесса создания проекта.

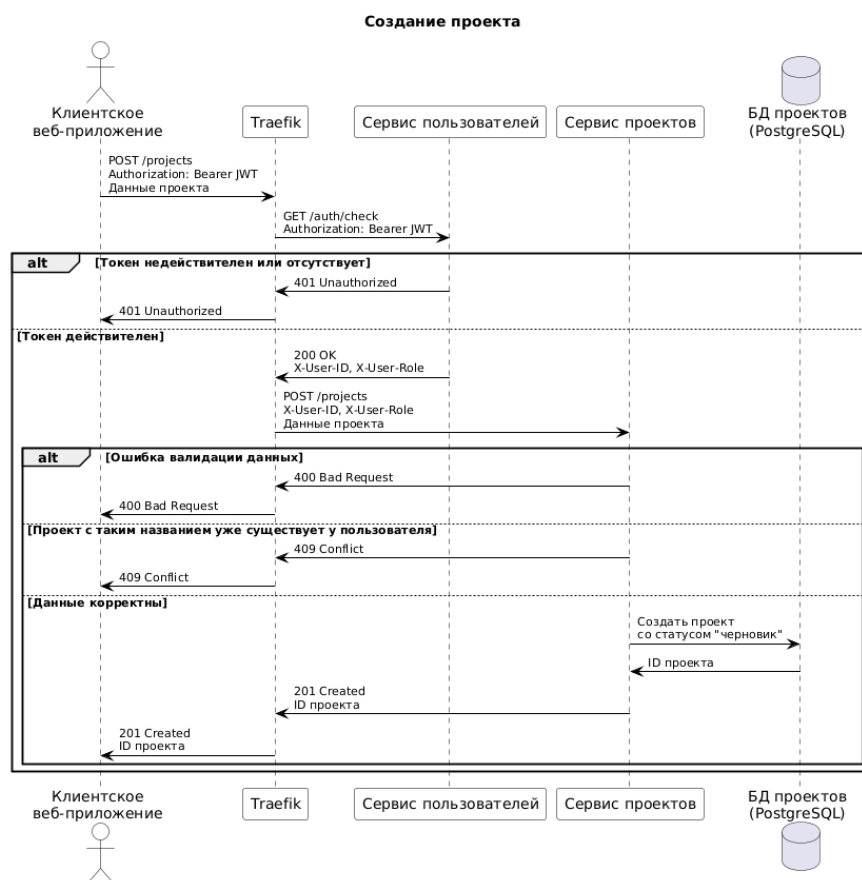


Рисунок 13 – Диаграмма последовательности процесса создания проекта

Процесс создания проекта начинается с отправки запроса клиентским веб-приложением через Traefik. Перед выполнением операции проверяется JWT-токен пользователя с помощью сервиса пользователей. После успешной проверки сервис проектов получает данные создаваемого проекта, проверяет их корректность и отсутствие у пользователя проекта с таким же названием. Если проверки пройдены, сервис проектов создает проект в базе данных со статусом «черновик» и возвращает клиентскому приложению ID созданного проекта. Если пользователь прикладывает обложку и изображения проекта, они отправляются отдельными запросами.

На рисунке 14 представлена диаграмма последовательности процесса взноса в проект.

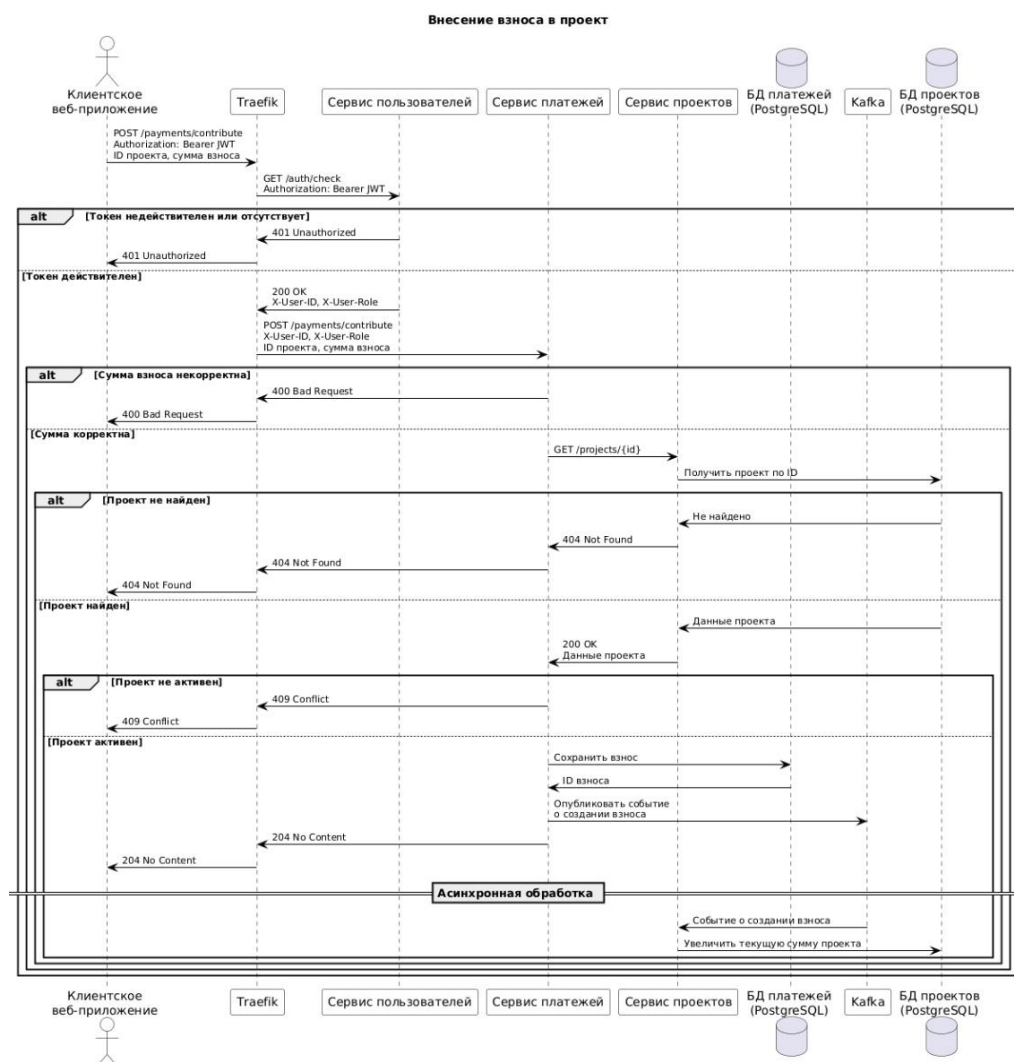


Рисунок 14 – Диаграмма последовательности процесса внесения взноса в проект

Процесс внесения взноса начинается с отправки запроса клиентским веб-приложением через Traefik. Перед выполнением операции проверяется JWT-токен пользователя. После успешной проверки сервис платежей проверяет корректность суммы взноса и запрашивает данные проекта у сервиса проектов. Если проект найден и находится в активном состоянии, сервис платежей сохраняет информацию о взносе в базе данных и публикует событие в Kafka о создании взноса. После этого клиентскому приложению возвращается успешный ответ, а текущая сумма проекта увеличивается асинхронно сервисом проектов.

На рисунке 15 представлена диаграмма последовательности процесса применения промокода для продвижения проекта.

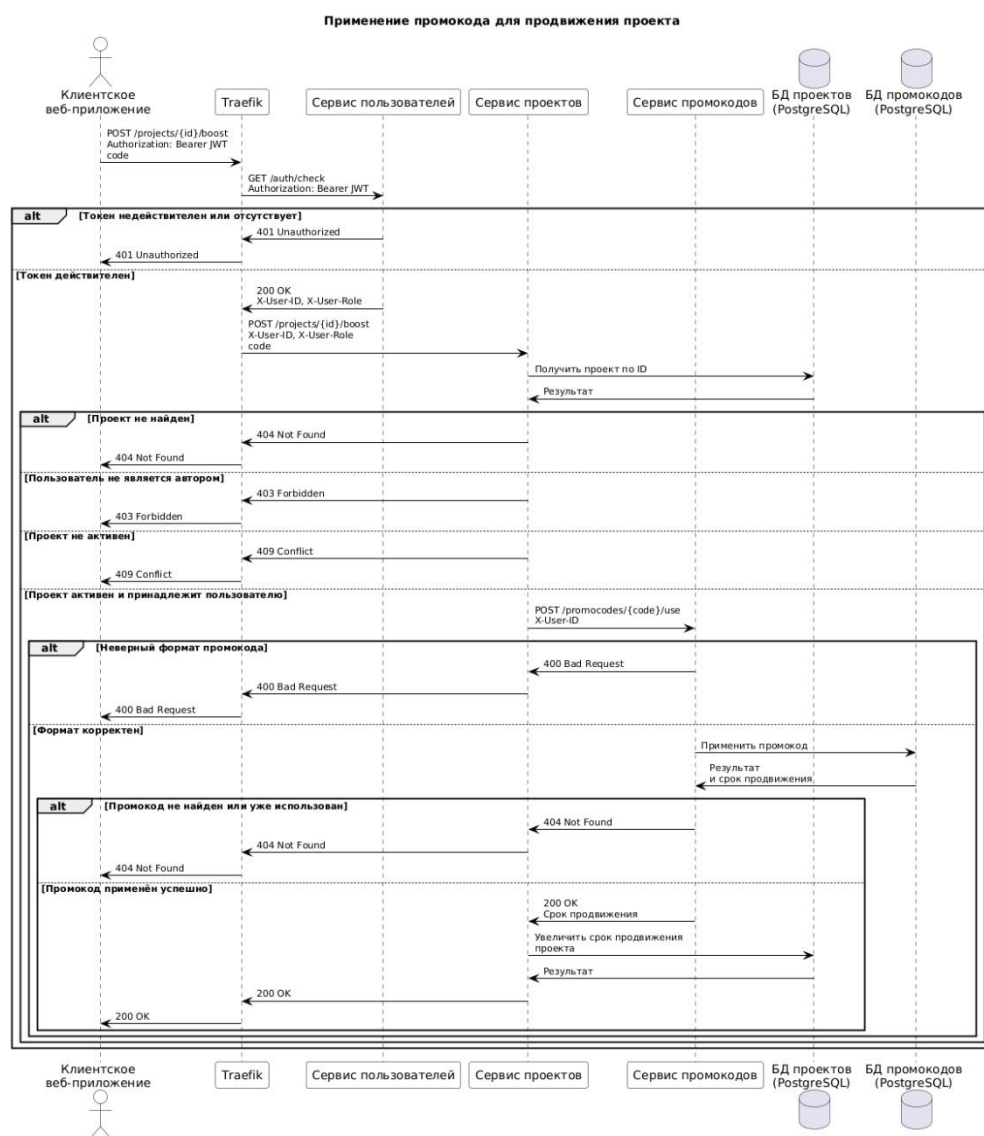


Рисунок 15 – Диаграмма последовательности процесса применения промокода для продвижения проекта

Процесс применения промокода начинается с отправки запроса клиентским веб-приложением через Traefik. Перед выполнением операции проверяется JWT-токен пользователя с помощью сервиса пользователей. После успешной проверки сервис проектов получает данные проекта из базы данных и проверяет его существование, принадлежность текущему пользователю и активный статус. Если проверки пройдены, сервис проектов обращается к сервису промокодов, который проверяет формат промокода и применяет его, возвращая срок продвижения. Затем сервис проектов увеличивает срок продвижения проекта в базе данных и возвращает ответ клиентскому приложению.

Таким образом, было выполнено проектирование взаимодействия компонентов системы и описаны основные сценарии их коммуникации с помощью диаграмм последовательности.

2.3 Проектирование модели данных

2.3.1 Концептуальная модель данных

Концептуальная модель данных используется для описания основных сущностей и связей между ними с учетом бизнес-требований, без детализации проектирования базы данных [28]. Концептуальная модель данных приложения представлена на рисунке 16.

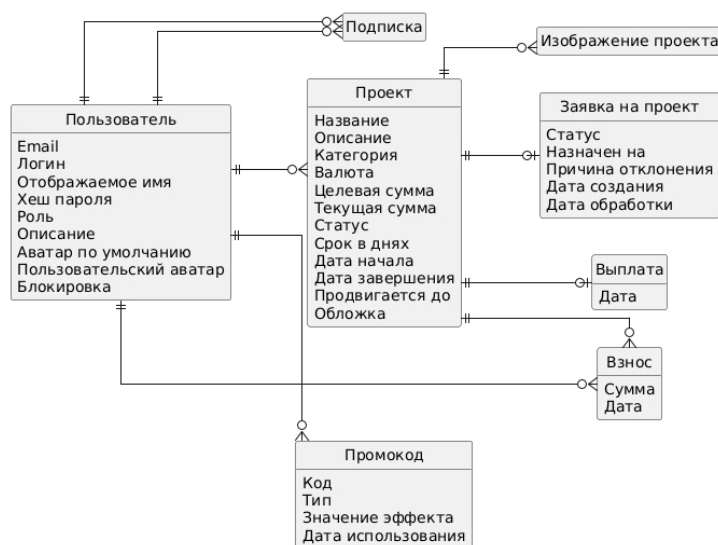


Рисунок 16 – Концептуальная модель данных

Описание всех сущностей концептуальной модели данных представлено в таблице 2.

Таблица 2 – Сущности концептуальной модели данных

Название сущности	Описание
Пользователь	Содержит сведения о пользователях системы
Подписка	Содержит информацию о подписке одного пользователя на другого
Проект	Хранит основную информацию о проекте
Изображение проекта	Содержит сведения об изображениях, прикрепленных к проекту
Заявка на проект	Хранит сведения о заявке на рассмотрение проекта для публикации
Взнос	Хранит сведения о взносах пользователей
Выплата	Хранит сведения о выплате средств по завершённому проекту
Промокод	Хранит сведения о промокодах

Описание связей между сущностями концептуальной модели приведены в таблице 3.

Таблица 3 – Связи между сущностями концептуальной модели данных

Сущность 1	Сущность 2	Семантика связи	Мощность	Обязательность
Пользователь	Проект	Пользователь является автором проекта	1 : M	Для пользователя необязательная, для проекта обязательная
Пользователь	Взнос	Пользователь вносит средства в проект	1 : M	Для пользователя не обязательная, для взноса обязательная
Проект	Взнос	Проект получает взносы пользователей	1 : M	Для проекта необязательная, для взноса обязательная

Продолжение таблицы 3

Проект	Изображение проекта	Проект содержит изображения	1 : М	Для проекта необязательная, для изображения обязательная
Проект	Заявка на проект	Проект имеет заявку на рассмотрение	1 : 0..1	Для проекта необязательная, для заявки обязательная
Проект	Выплата	Проект может иметь выплату	1 : 0..1	Для проекта необязательная, для выплаты обязательная
Пользователь	Подписка	Пользователь является подписчиком	1 : М	Для пользователя необязательная, для подписки обязательная
Пользователь	Подписка	Пользователь является целью подписки	1 : М	Для пользователя необязательная, для подписки обязательная
Пользователь	Промокод	Пользователю принадлежит промокод	1 : М	Для пользователя необязательная, для промокода обязательная

Таким образом, построенная концептуальная модель фиксирует основные сущности разрабатываемой системы и связи между ними.

2.3.2 Логическая модель данных

Логическая модель разрабатывается на основе концептуальной и задает более подробное представление данных, включая сущности, связи между ними, атрибуты, первичные и внешние ключи [28]. При этом такая модель остается независимой от конкретной базы данных [28]. Для информационной системы

организации сбора средств на проекты логическая модель разделена по зонам ответственности микросервисов: выделены базы данных пользователей, проектов, платежей и промокодов. Такой подход обеспечивает изоляцию данных между сервисами и снижает связанность компонентов системы.

На рисунке 17 представлена логическая модель базы данных пользователей.

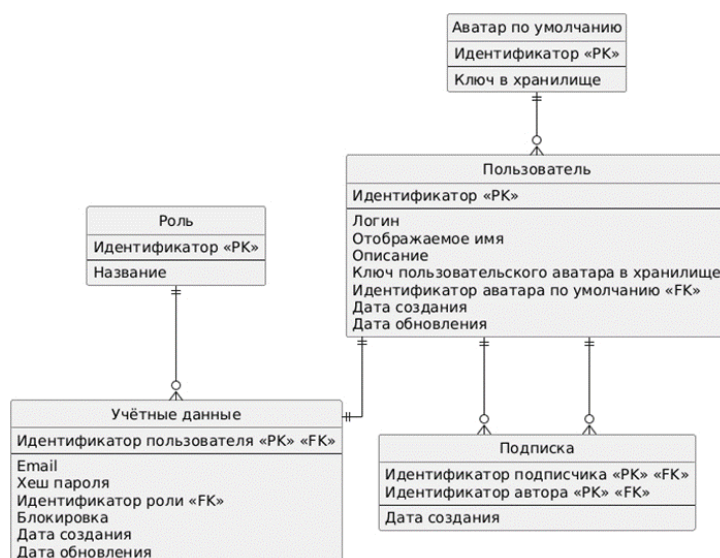


Рисунок 17 – Логическая модель базы данных пользователей

На рисунке 18 представлена логическая модель базы данных проектов.

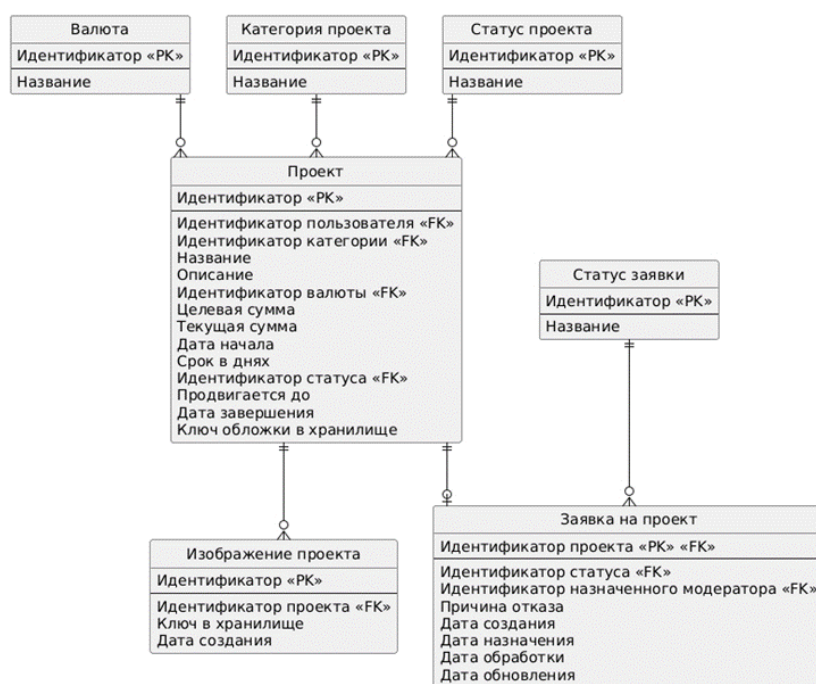


Рисунок 18 – Логическая модель базы данных проектов

На рисунке 19 представлена логическая модель базы данных платежей.

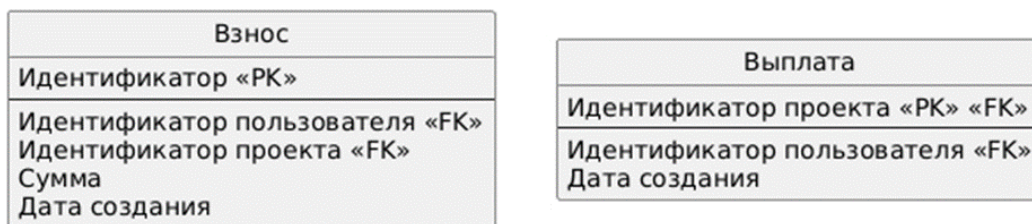


Рисунок 19 – Логическая модель базы данных платежей

На рисунке 20 представлена логическая модель базы данных промокодов.



Рисунок 20 – Логическая модель базы данных промокодов

Разработанные логические модели данных будут использованы при реализации физических моделей баз данных.

2.4 Проектирование серверной части

Для микросервисов была выбрана слоистая архитектура, при которой обработка HTTP-запросов, выполнение бизнес-логики и доступ к данным выделены в отдельные слои. Такой подход соответствует распространенному разделению веб-приложений на веб-слой, слой бизнес-логики и слой доступа к данным [29].

Слоистая архитектура для микросервисов выбрана потому, что она позволяет рассматривать обработку запросов, бизнес-логику и доступ к данным как относительно самостоятельные части приложения. Это снижает сложность внесения изменений, поскольку разработчик может сосредоточиться на отдельном слое, не смешивая его с остальными частями приложения [29]. Кроме того, такое разделение улучшает тестируемость системы [29].

На рисунке 21 представлена слоистая архитектура микросервиса разрабатываемого приложения.



Рисунок 21 – Слоистая архитектура микросервиса

В зависимости от ответственности конкретного микросервиса архитектура может дополняться дополнительными модулями:

- обработчики событий брокера сообщений (consumers);
- модули публикации событий (publishers);
- клиенты внешних сервисов.

Таким образом, для серверной части была определена типовая архитектура микросервиса, которая используется как основа при последующей реализации компонентов системы.

2.5 Проектирование клиентской части

При проектировании клиентского веб-приложения разрабатываемой системы была составлена карта сайта – визуальная схема, позволяющая отобразить основные страницы сайта и связи между ними [30]. Карта сайта представлена на рисунке 22.

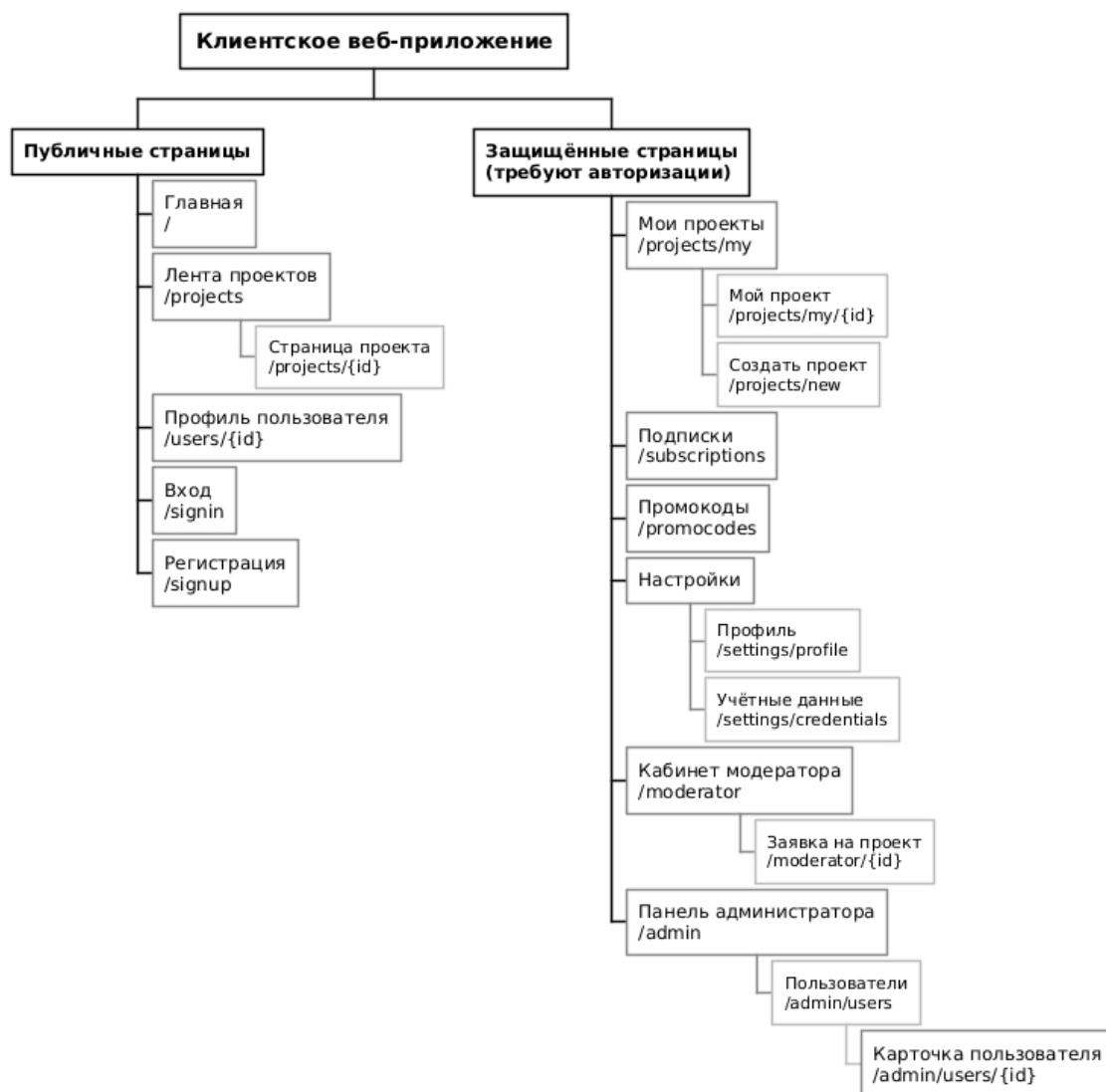


Рисунок 22 – Карта сайта

Страницы клиентского веб-приложения разделены на публичные и защищённые, которые требуют авторизации.

К публичным страницам клиентской части относятся «Главная», «Лента проектов», «Страница проекта», «Профиль пользователя», а также страницы «Вход» и «Регистрация». Эти страницы предназначены для просмотра доступных проектов, ознакомления информацией об авторах, входа в систему и создания учетной записи.

К защищённым относятся страницы «Мои проекты», «Подписки», «Промокоды», «Настройки», «Кабинет модератора», «Панель администратора» и их дочерние.

На странице «Мои проекты» автор может открыть страницу своего проекта или форму создания нового. Раздел «Подписки» отображает подписки пользователя на других зарегистрированных участников платформы. На странице «Промокоды» можно просмотреть как активные, так и использованные промокоды.

Кабинет модератора включает список заявок и страницу отдельной заявки, где содержится информация о проекте и располагаются кнопки одобрения или отклонения заявки.

Панель администратора включает список пользователей и карточку пользователя. Список используется для выбора учетной записи, а карточка – для просмотра сведений о пользователе и выполнения административных действий.

3 Реализация информационной системы

3.1 Физическая модель данных

Физическая модель данных обычно строится на основе логической и учитывает особенности конкретной СУБД [28]. В разрабатываемой информационной системе используется PostgreSQL. Согласно выбранной архитектуре, для каждого микросервиса создана отдельная база данных: users_db, projects_db, payments_db, promocodes_db.

На рисунке 23 представлена физическая модель базы данных users_db.

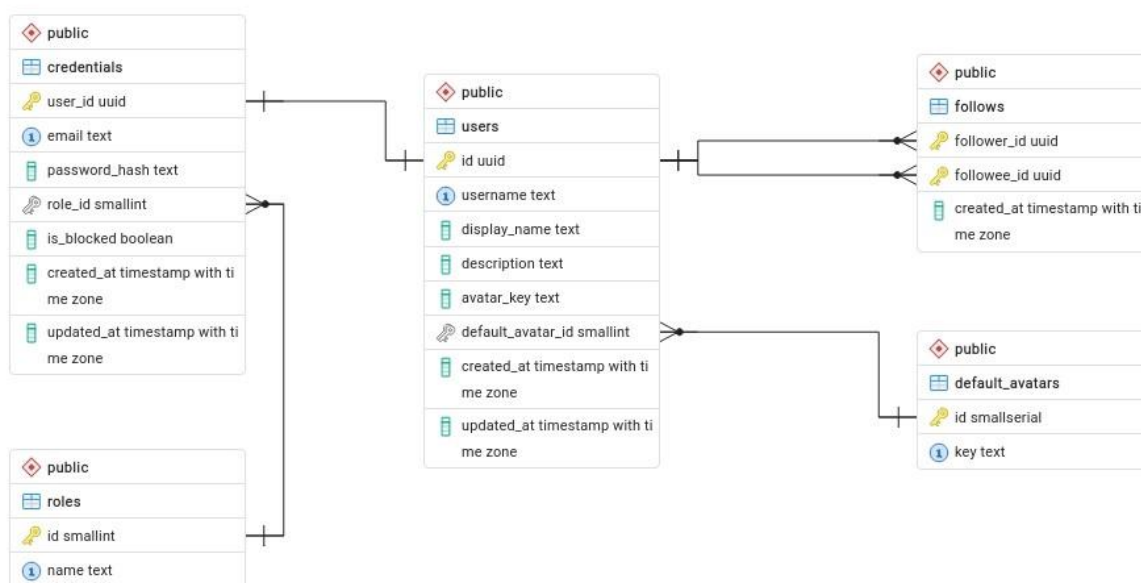


Рисунок 23 – Физическая модель базы данных users_db

На рисунке 24 представлена физическая модель базы данных payments_db.

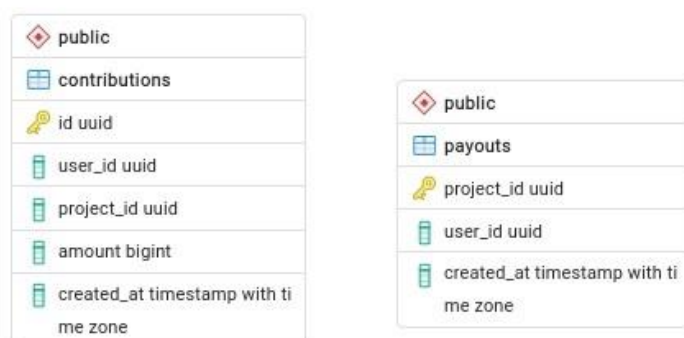


Рисунок 24 – Физическая модель базы данных payments_db

На рисунке 25 представлена физическая модель базы данных projects_db.

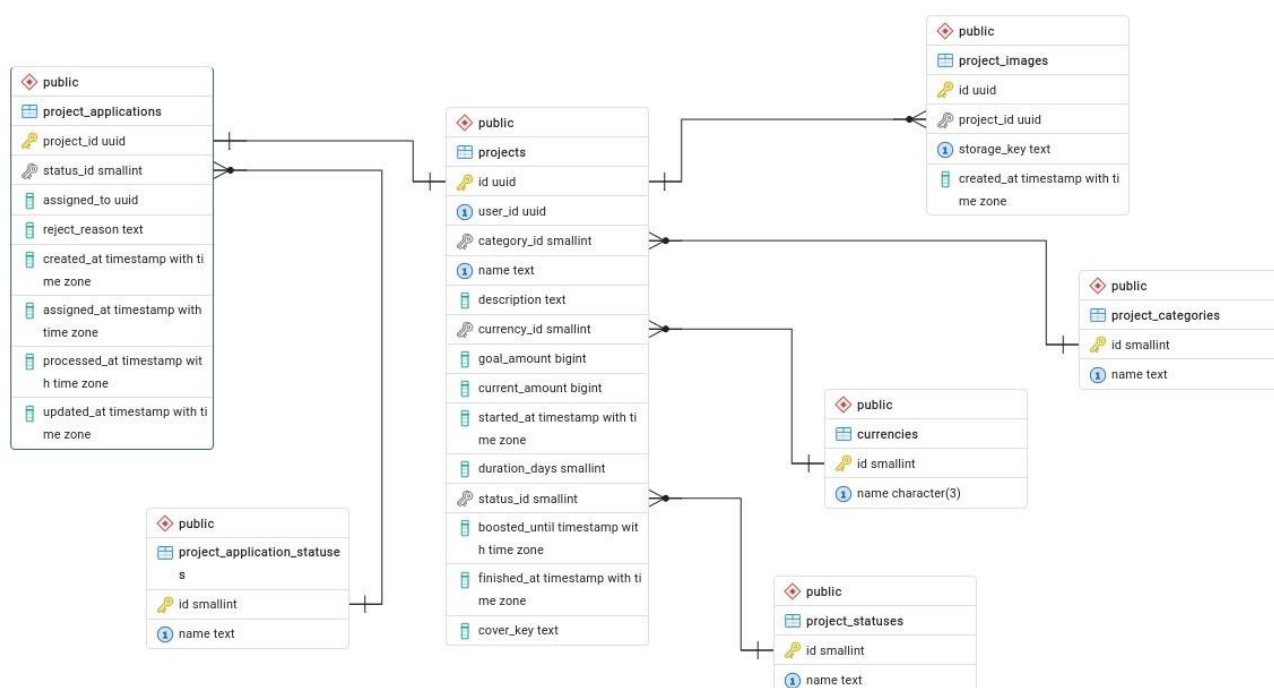


Рисунок 25 – Физическая модель базы данных projects_db

На рисунке 26 представлена физическая модель базы данных promocodes_db.

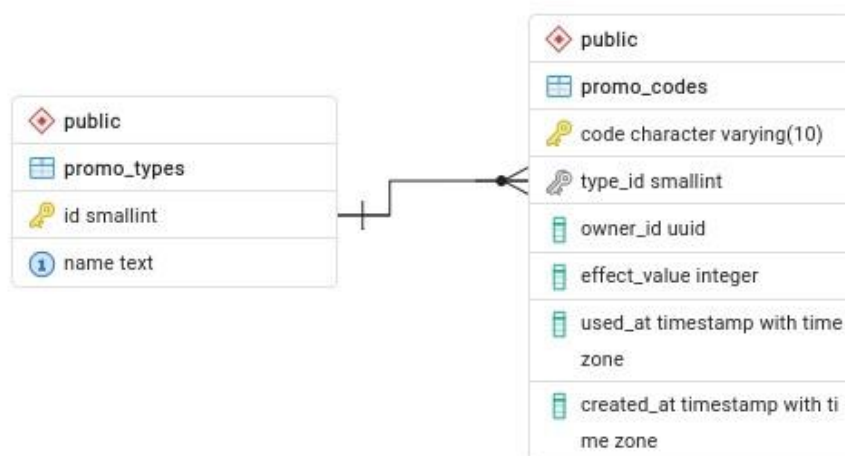


Рисунок 26 – Физическая модель базы данных promocodes_db

Таблицы в базах данных приложения создаются с помощью механизма миграций: каждый микросервис при запуске применяет SQL-скрипты, приводя структуру своей базы данных к актуальному состоянию.

SQL-скрипты миграций для создания таблиц каждой базы данных приведены в приложении Б.

3.2 Реализация серверной части

3.2.1 Реализация микросервиса пользователей

Микросервис пользователей реализован в виде двух модулей: модуль `auth` отвечает за аутентификацию пользователей с помощью JWT-токенов и управление учётными данными, а модуль `user` отвечает за профили пользователей. Диаграмма классов микросервиса представлена в приложении И (рисунок И.1).

Инициализация HTTP-сервера и подключение к нему обработчиков выполняется в методе `InitServer` приложения. На созданном экземпляре сервера вызываются методы `AddAuthHandlers` и `AddUserHandlers`, которые регистрируют эндпоинты модулей `auth` и `user` соответственно. Также с помощью метода `AddMetrics` регистрируется эндпоинт сбора метрик `Prometheus`.

Основная бизнес-логика модуля `auth` представлена структурой `auth.Service`. Для генерации и проверки JWT-токенов в модуле используется вспомогательная структура `token.Service`.

Для обращения к репозиторию `refresh`-токенов структурой `token.Service` используется интерфейс `refresh.Repo`, реализация которого работает с хранилищем `Redis`.

Доступ к учётным данным пользователей структура `auth.Service` получает через интерфейс `auth.Repo`, реализация которого предоставляется модулем `user`. Публикация событий в структуре `auth.Service` осуществляется через метод `Publish` структуры `kafka.Publisher`.

Основная бизнес-логика модуля user представлена в структуре user.Service. Для доступа к данным она использует интерфейс user.Repo, реализация которого работает с PostgreSQL.

Фрагменты исходного кода микросервиса пользователей представлены в приложении В. Полный код доступен в репозитории на GitHub [31].

3.2.2 Реализация микросервиса проектов

Микросервис проектов отвечает за управление жизненным циклом проектов: создание, модерация, публикация, сбор средств и завершение. Диаграмма классов микросервиса представлена в приложении И (рисунок И.2).

HTTP-обработчики микросервиса подключаются к серверу с помощью метода AddProjectHandlers. На сервере также регистрируется эндпоинт сбора метрик компонента с помощью Prometheus.

Основная бизнес-логика микросервиса реализована в пакете service.project. Для доступа к данным проектов структура project.Service использует интерфейс project.Repo, реализация которого работает с PostgreSQL. Загрузка и удаление изображений проектов выполняются через интерфейс storage.Client, реализация которого использует MinIO. Взаимодействие с микросервисом промокодов осуществляется через интерфейс promocode.Client, реализованный с помощью библиотеки Resty.

Публикация событий о завершении проектов в Kafka реализована по паттерну transactional outbox, что обеспечивает гарантию доставки at-least-once. За работу с outbox отвечают два фоновых обработчика: project.FinishWorker завершает проекты с истёкшим сроком сбора средств и записывает события в outbox-таблицу, а project.PublishWorker периодически читает неотправленные записи и публикует их в Kafka, помечая записи как отправленные только после успешной публикации. Функция однократного запуска обработчика project.PublishWorker приведена в листинге 1.

Листинг 1 – Функция однократного запуска обработчика project.PublishWorker

```
func (w *PublishWorker) runOnce(ctx context.Context) {
```

```

w.metrics.LastRunUnix.Set(float64(time.Now().Unix()))

projects, err := w.repo.ListPendingFinishedOutbox(ctx,
defaultPublishBatch)
if err != nil {
    if ctx.Err() != nil {
        return
    }
    log.Printf("publisher worker error: %v", err)
    return
}

for _, p := range projects {
    event := projectPublisher.Event{
        Type:    projectPublisher.EventTypeFinished,
        Project: p,
    }
    err := w.publisher.Publish(ctx, event)
    if err != nil {
        log.Printf("publisher worker publish error: %s",
err)
        continue
    }

    err = w.repo.MarkSentFinishedOutbox(ctx, p.ID)
    if err != nil {
        log.Printf("publisher worker mark sent error: %s",
err)
        continue
    }
}
}

```

Получение событий о новых взносах из брокера сообщений организовано с помощью структуры `contribution.Consumer`.

Фрагменты исходного кода микросервиса проектов представлены в приложении Г. Полный код доступен в репозитории на GitHub [32].

3.2.3 Реализация микросервиса платежей

Микросервис платежей обеспечивает приём и учёт взносов в проекты, а также формирование выплат авторам после завершения сбора средств. Диаграмма классов микросервиса представлена в приложении И (рисунок И.3).

HTTP-обработчики микросервиса подключаются к серверу с помощью метода `AddPaymentHandlers`. На сервере также регистрируется эндпоинт сбора метрик Prometheus для мониторинга состояния компонента.

Основная бизнес-логика микросервиса реализована в структуре `payment.Service`. Для доступа к данным о платежах она использует интерфейс `payment.Repo`, реализация которого работает с PostgreSQL. Данные о проектах структура `payment.Service` получает через интерфейс `project.Client`, реализованный с помощью библиотеки `Resty`. Публикация событий о платежах в брокер сообщений `Kafka` осуществляется с помощью структуры `payment.Publisher`.

Получение событий о проектах из `Kafka` реализовано с помощью структуры `project.Consumer`. При получении события `project.EventTypeFinished` инициируется выплата автору. Обработка идемпотентна: если событие уже было обработано, повторный вызов завершается без выполнения выплаты. Обработчик события приведён в листинге 2.

Листинг 2 – Обработка события `project.EventTypeFinished`

```
func (c *Consumer) handle(ctx context.Context, msg kafka.Message)
error {
    var event project.Event
    err := json.Unmarshal(msg.Value, &event)
    if err != nil {
        return fmt.Errorf("project consumer unmarshal: %w", err)
    }

    switch event.Type {
    case project.EventTypeFinished:
        err = c.svc.Payout(ctx, event.Project.ID,
event.Project.UserID)
        if errors.Is(err, domain.ErrEventProcessed) {
            return nil
        }
        if err != nil {
            return fmt.Errorf("payment service: %w", err)
        }

        log.Printf("init payout for project %s",
event.Project.ID)
    }

    return nil
}
```

Фрагменты исходного кода микросервиса платежей представлены в приложении Д. Полный код доступен в репозитории на GitHub [33].

3.2.4 Реализация микросервиса промокодов

Рассматриваемый микросервис управляет выпуском и применением промокодов. Диаграмма классов микросервиса представлена в приложении И (рисунок И.4).

HTTP-обработчики микросервиса подключаются к серверу с помощью метода `AddPromocodeHandlers`. На сервере также с помощью метода `AddMetrics` регистрируется эндпоинт сбора метрик `Prometheus`.

Основная бизнес-логика микросервиса реализована в структуре `promocode.Service`. Доступ к данным о промокодах она получает через интерфейс `promocode.Repo`, реализация которого работает с `PostgreSQL`.

Получение и обработку событий о пользователях из `Kafka` реализует структура `user.Consumer`. При получении события типа `user.EventTypeRegistered` инициируется создание промокода на продвижение проекта для нового пользователя.

Фрагменты исходного кода микросервиса промокодов представлены в приложении Е. Полный код доступен в репозитории на `GitHub` [34].

3.3 Реализация клиентского приложения

3.3.1 Структура клиентского приложения

Клиентское веб-приложение разработано с использованием фреймворка `SvelteKit` и языка программирования `TypeScript`. Сборка приложения осуществляется с помощью `Vite`.

Маршрутизация приложения строится на основе файловой структуры проекта: вложенность директорий определяет соответствующие `URL`-пути, что является особенностью фреймворка `SvelteKit`.

Диаграмма компонентов клиентской части представлена на рисунке 27.

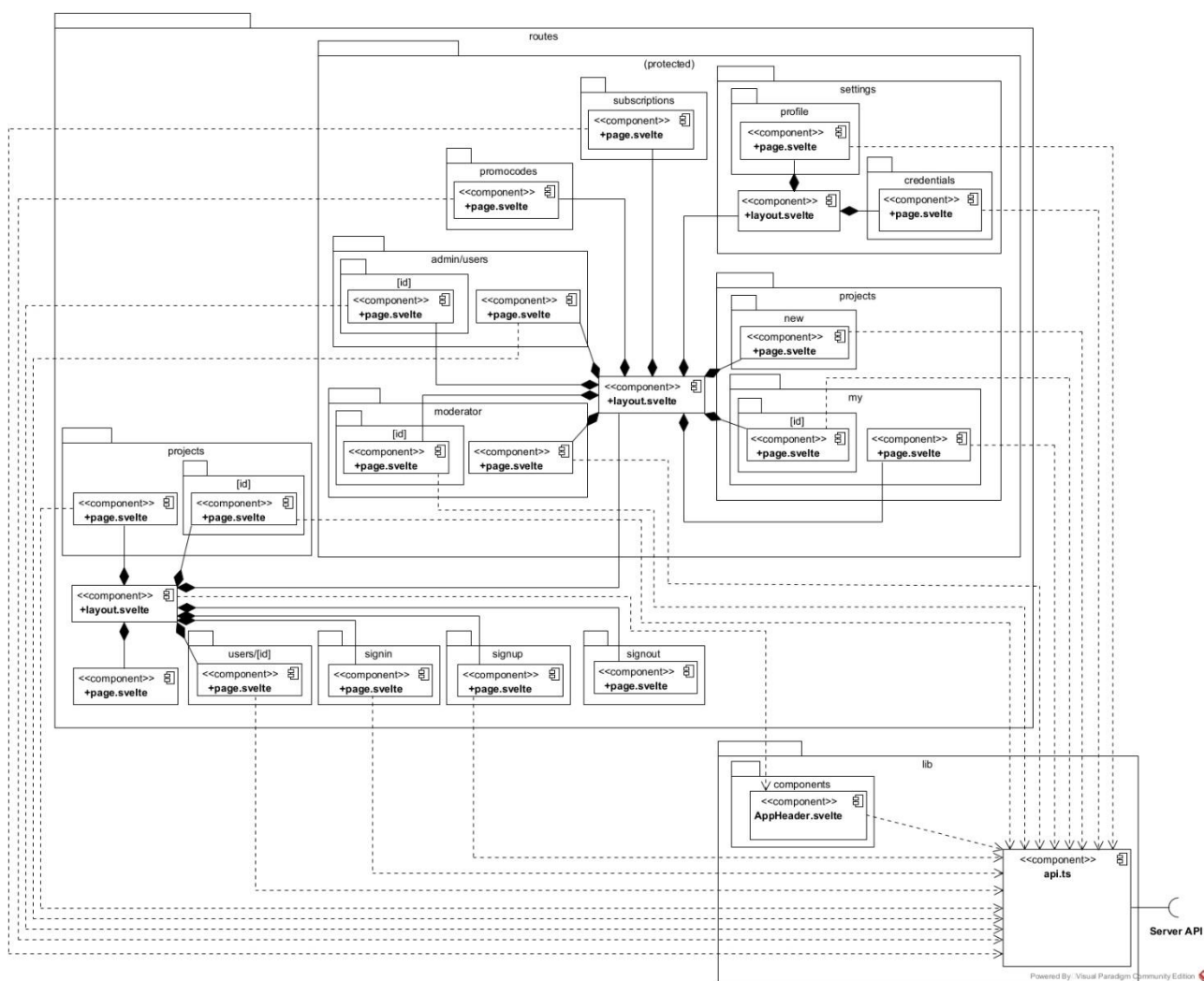


Рисунок 27 – Диаграмма компонентов клиентской части

На диаграмме компонент `+page.svelte` соответствует отдельной странице приложения и отображается при переходе по соответствующему пути. Файлы `+layout.svelte` задают общую структуру для вложенных маршрутов, а динамические сегменты в квадратных скобках, например `[id]`, позволяют обрабатывать параметризованные адреса. Компонент `AppHeader.svelte` представляет собой шапку страницы и содержит элементы навигации.

На диаграмме компонентов видно разделение маршрутов на публичные, доступные без аутентификации, и защищённые, объединенные в группу `protected`. Все запросы к серверной части централизованы в файле `api.ts`. При отправке каждого запроса, требующего аутентификации, в заголовок `Authorization` добавляется access-токен. Если сервер возвращает код 401, компонент автоматически отправляет запрос на обновление токена и повторяет

исходный запрос. При неудаче пользователь перенаправляется на страницу входа. Реализация функции, выполняющей запрос с обновлением токенов, приведена в листинге 3.

Листинг 3 – Функция выполнения запроса с обновлением токенов

```
async function authFetch(url: string, options: RequestInit,
accessToken: string): Promise<Response> {
  const res = await fetch(url, {
    ...options,
    headers: { ...options.headers, Authorization: `Bearer
${accessToken}` },
  });

  if (res.status !== 401) return res;

  const refreshToken = localStorage.getItem('refreshToken');
  if (!refreshToken) { clearAuth(); return res; }

  const refreshRes = await fetch(`${BASE}/auth/refresh`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ refreshToken }),
  });

  if (refreshRes.status === 401) { clearAuth(); return res; }
  if (refreshRes.status !== 200) return res;

  const tokens: SigninTokens = await refreshRes.json();
  localStorage.setItem('accessToken', tokens.accessToken);
  localStorage.setItem('refreshToken', tokens.refreshToken);

  // Retry original request with new token
  return fetch(url, {
    ...options,
    headers: { ...options.headers, Authorization: `Bearer
${tokens.accessToken}` },
  });
}
```

Фрагменты исходного кода клиентской части приложения представлены в приложении Ж. Полный код доступен в репозитории на GitHub [35].

3.3.2 Интерфейс клиентского приложения

В соответствии с картой сайта, представленной в пункте 2.5, разработан интерфейс клиентского веб-приложения, основные страницы которого описаны ниже.

На рисунке 28 представлена страница входа в систему.

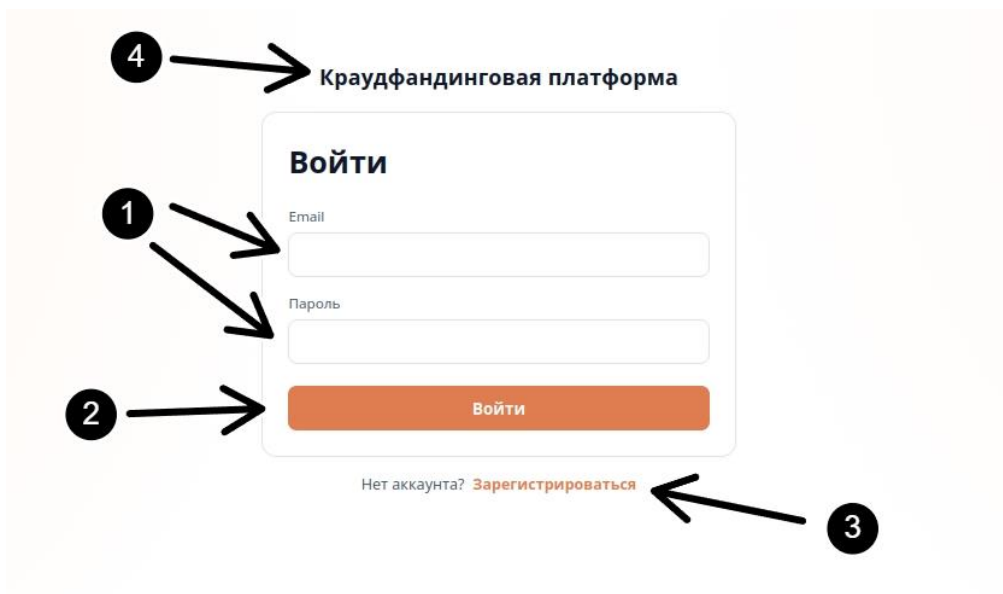


Рисунок 28 – Страница входа

Основные элементы страницы входа:

- 1 – поля ввода учётных данных;
- 2 – кнопка входа;
- 3 – переход на страницу регистрации;
- 4 – название платформы и переход на главную.

На рисунке 29 изображена страница регистрации в системе.

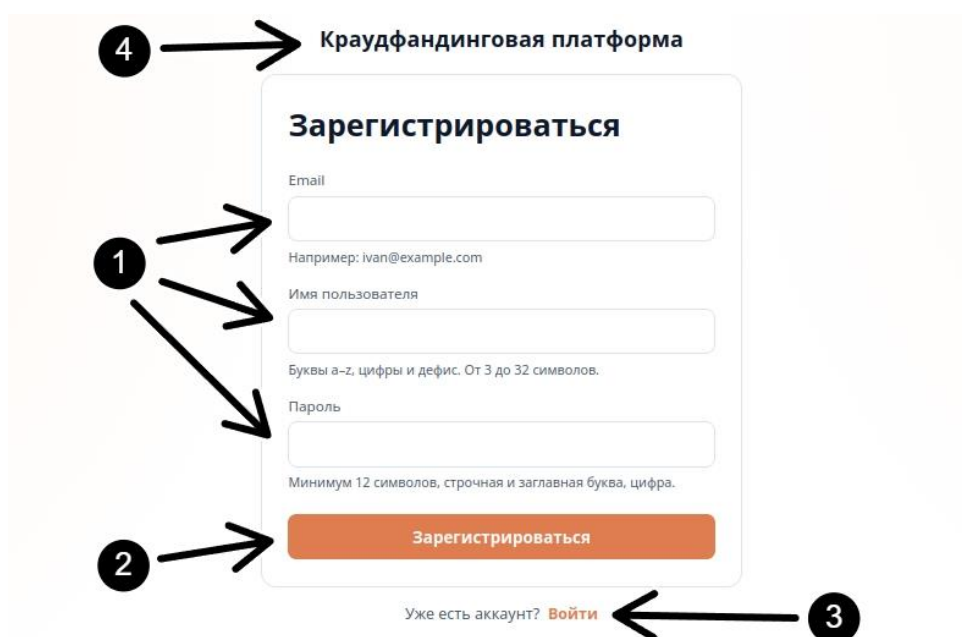


Рисунок 29 – Страница регистрации

Основные элементы страницы регистрации:

- 1 – поля ввода данных для регистрации;
- 2 – кнопка регистрации;
- 3 – переход на страницу входа;
- 4 – название платформы и переход на главную.

На рисунке 30 представлена страница ленты проектов.

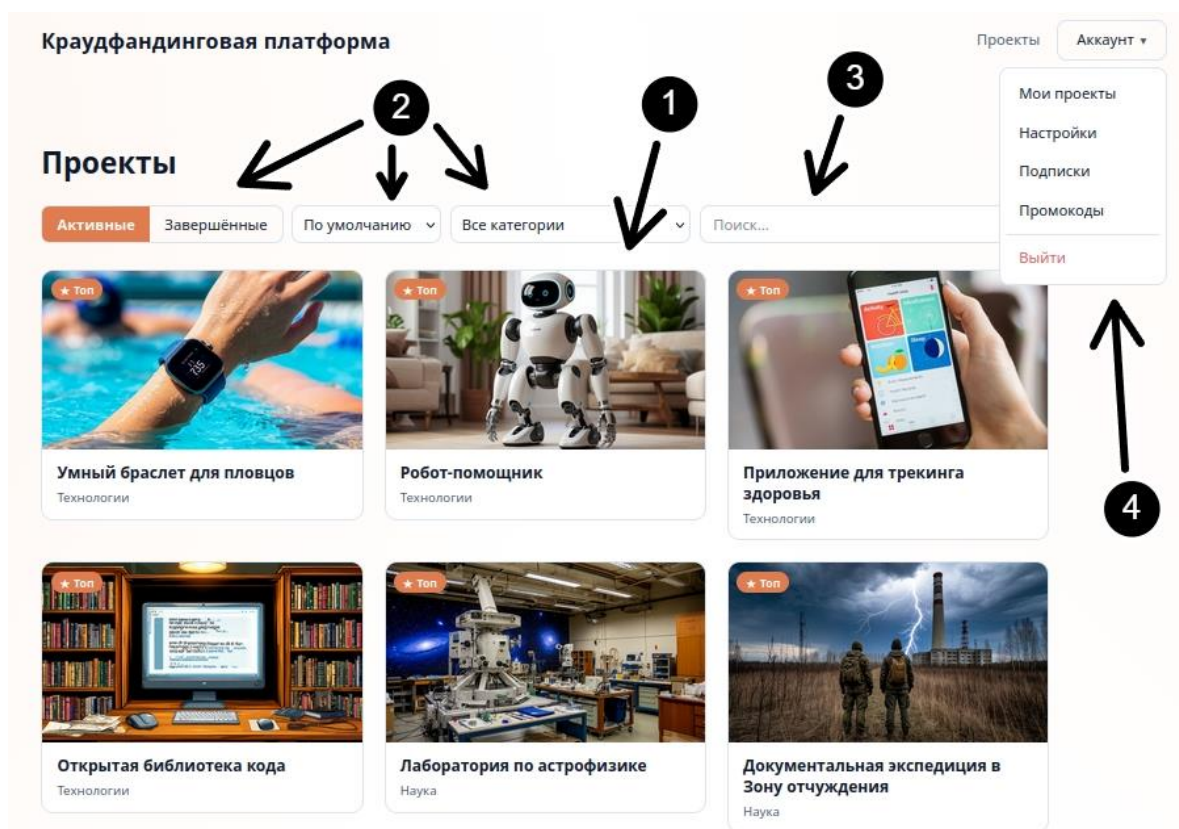


Рисунок 30 – Страница ленты проектов

Основные элементы страницы ленты проектов:

- 1 – карточка проекта;
- 2 – элементы выбора фильтрации и сортировки;
- 3 – строка поиска;
- 4 – панель аккаунта.

На рисунке 31 представлена страница проекта.

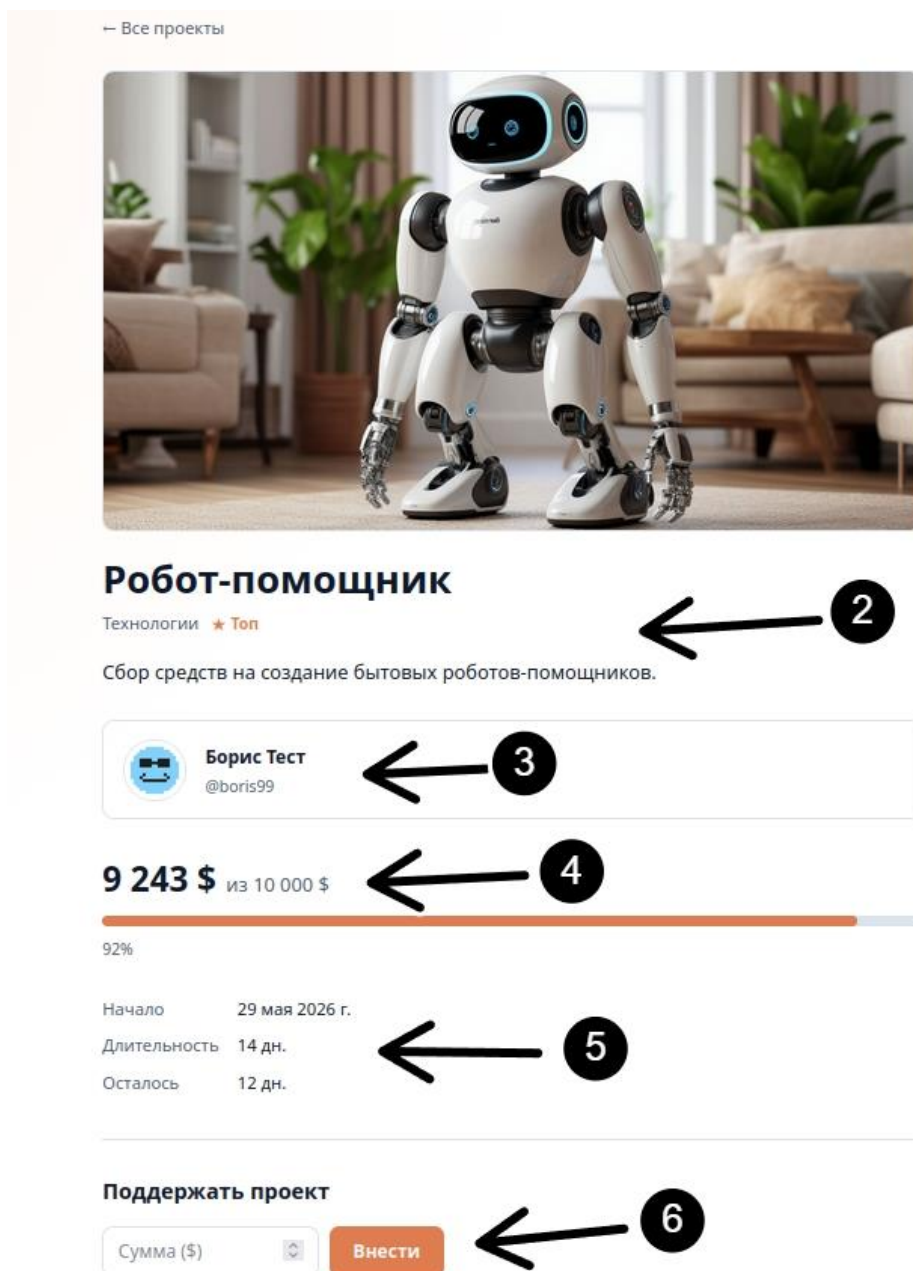


Рисунок 31 – Страница проекта

Основные элементы страницы проекта:

- 1 – обложка проекта;
- 2 – основная информация о проекте (название, категория, описание) и иконка продвижения;
- 3 – переход на страницу автора проекта;
- 4 – прогресс сбора средств;
- 5 – сроки проекта;
- 6 – элементы для осуществления взноса в проект.

На рисунке 32 отображена страница создания проекта.

← Мои проекты

Новый проект

НАЗВАНИЕ

до 80 символов

ОПИСАНИЕ

до 10 000 символов

КАТЕГОРИЯ

Выберите категорию ▾

ВАЛЮТА

Выберите валюту ▾

СУММА СБОРА

от 1 до 100 000 000

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

от 1 до 60

ОБЛОЖКА

опционально

Перетащите обложку сюда или нажмите для выбора
JPEG, PNG, GIF, WebP - до 5 МБ

ФОТОГРАФИИ

опционально

Перетащите фото сюда или нажмите для выбора
JPEG, PNG, GIF, WebP - до 5 МБ каждая

Создать проект

Diagram annotations: Arrows point from callouts 1, 2, 3, and 4 to the form elements. Callout 1 points to the Name and Description fields. Callout 2 points to the Cover upload area. Callout 3 points to the Photos upload area. Callout 4 points to the 'Создать проект' button.

Рисунок 32 – Страница создания проекта

Основные элементы страницы создания проекта:

- 1 – поля ввода информации о проекте;
- 2 – поле для добавления обложки проекта;
- 3 – поле для добавления изображений;
- 4 – кнопка создания проекта.

На рисунке 33 представлена страница модерации проекта.

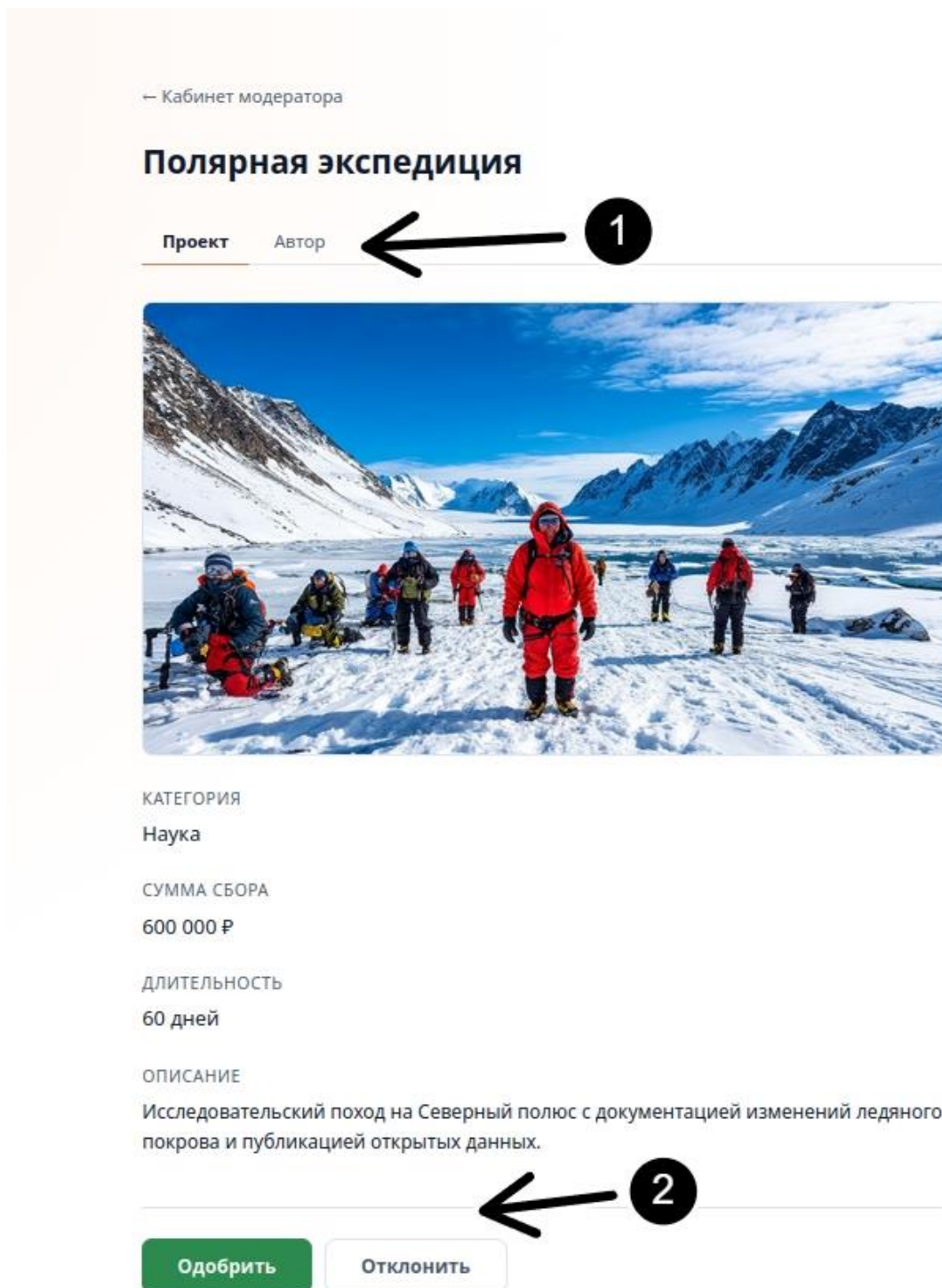


Рисунок 33 – Страница модерации проекта

Основные элементы страницы модерации проекта:

1 – переход на вкладку просмотра профиля автора;

2 – кнопки одобрения и отклонения заявки на публикацию.

На рисунке 34 отображена страница администрирования пользователя.

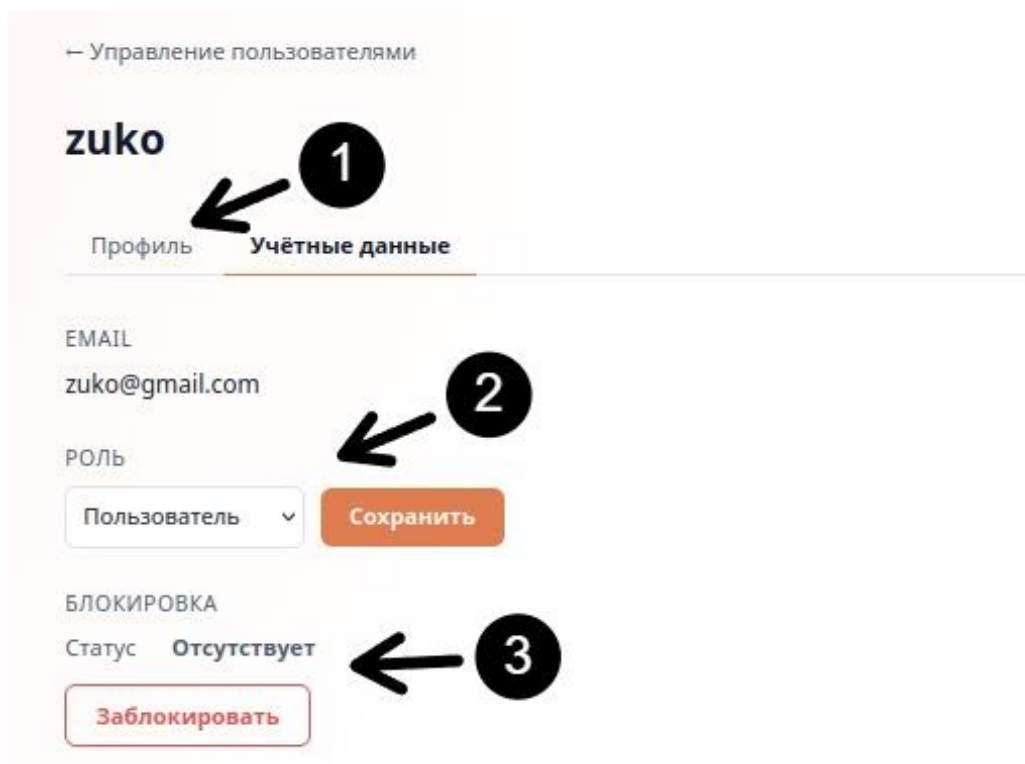


Рисунок 34 – Страница администрирования пользователя

Основные элементы страницы администрирования пользователя:

- 1 – переход на вкладку просмотра профиля пользователя;
- 2 – элементы управления для смены роли пользователя;
- 3 – статус и кнопка блокировки.

Таким образом, был реализован пользовательский интерфейс клиентского веб-приложения, предоставляющий доступ к основному функционалу разрабатываемой системы.

3.4 Развертывание системы

Развёртывание компонентов разработанной системы выполнено в контейнерной среде под управлением оркестратора Kubernetes. Диаграмма развертывания разработанной системы представлена на рисунке 35.

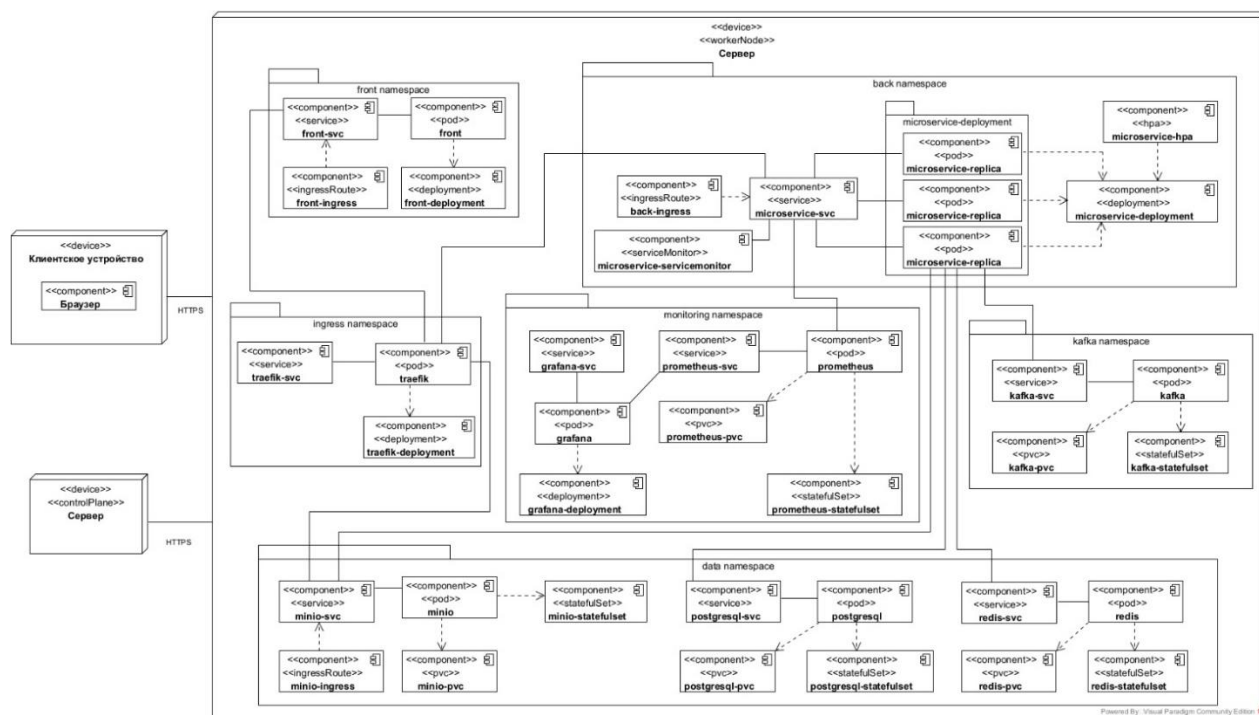


Рисунок 35 – Диаграмма развёртывания

Кластер Kubernetes делится на две части – control plane и worker nodes [36]. Узел control plane управляет состоянием всего кластера, распределяет поды между узлами и предоставляет API для взаимодействия с кластером [36]. На узлах типа worker выполняются служебные компоненты, поддерживающие работу подов и обеспечивающие среду выполнения Kubernetes [36].

Ресурсы кластера организованы в шесть пространств имён (namespace), каждое из которых объединяет компоненты по функциональному назначению.

В пространстве имён back размещены микросервисы серверной части. Для каждого из них применяется одинаковый набор ресурсов Kubernetes:

- Deployment задаёт конфигурацию подов и управление их количеством [37];
- Service типа ClusterIP предоставляет виртуальный IP-адрес для доступа к подам, по которому Kubernetes распределяет трафик между ними [38];
- HPA автоматически масштабирует число реплик подов в зависимости от нагрузки [39].

Далее перечисленные ресурсы рассмотрены на примере микросервиса промокодов.

Ресурс Deployment микросервиса промокодов представлен в листинге 4.

Листинг 4 – Ресурс Deployment микросервиса промокодов

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: promo
  namespace: back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: promo
  template:
    metadata:
      labels:
        app: promo
    spec:
      containers:
        - name: promo
          image: crowdfunding-app-promo:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
      resources:
        requests:
          cpu: 50m
          memory: 64Mi
        limits:
          cpu: 200m
          memory: 256Mi
      env:
        - name: POSTGRES_URL
          valueFrom:
            secretKeyRef:
              name: pg-secrets
              key: PROMO_DB_URL
        - name: POSTGRES_MIGRATIONS_DIR
          value: /app/migrations/postgres
        - name: KAFKA_BROKERS
          value: my-cluster-kafka-
bootstrap.kafka.svc.cluster.local:9092
        - name: USER_TOPIC
          value: user
        - name: USER_GROUP_ID
          value: promo
```

В Deployment задаётся образ контейнера и порт микросервиса, а через requests и limits ограничивается потребление процессора и памяти. Конфигурация подключения к базе данных и брокеру сообщений передаётся

через переменные окружения, причём строка подключения к БД извлекается из секрета.

Ресурс Service микросервиса промокодов представлен в листинге 5.

Листинг 5 – Ресурс Service микросервиса промокодов

```
apiVersion: v1
kind: Service
metadata:
  name: promo-svc
  namespace: back
  labels:
    app: promo
spec:
  selector:
    app: promo
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP
```

Представленный ресурс Service типа ClusterIP предоставляет точку доступа к подам микросервиса промокодов внутри кластера, которые отбираются по селектору, где ключ app имеет значение promo.

Ресурс HPA для микросервиса промокодов представлен в листинге 6.

Листинг 6 – Ресурс HPA микросервиса промокодов

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: promo-hpa
  namespace: back
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: promo
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Представленный пример НРА отслеживает среднюю загрузку процессора подов и изменяет число реплик микросервиса промокодов в диапазоне от одной до трёх: при превышении порога в 70% создаются дополнительные реплики, а при снижении нагрузки их количество уменьшается.

Сбор метрик для Prometheus обеспечивается общим для всех микросервисов ресурсом ServiceMonitor.

В пространстве имён ingress развёрнут Traefik, выполняющий роль единой точки входа внешнего трафика в кластер. Маршрутизация задаётся тремя ресурсами IngressRoute, обеспечивающими доступ к микросервисам, клиентской части и публичным изображениям в MinIO.

Пространство имён front содержит клиентское приложение, развёрнутое с помощью ресурсов Deployment и Service.

В пространстве имён data и kafka содержатся компоненты хранения данных PostgreSQL, Redis, MinIO и брокер сообщений Kafka. Они развёрнуты с использованием StatefulSet – ресурса, обеспечивающего стабильный сетевой идентификатор, что полезно, когда необходимо иметь постоянное хранилище [40].

В отдельное пространство имён monitoring вынесены Prometheus, развёрнутый через ресурс StatefulSet с персистентным томом для хранения временных рядов, и Grafana, развёрнутая с использованием ресурса Deployment.

Полный набор конфигурационных файлов ресурсов Kubernetes для развёртывания приложения доступен в репозитории на GitHub [41].

Минимальные и рекомендуемые требования к аппаратному и программному обеспечению для развёртывания системы приведены в таблице 4.

Таблица 4 – Требования к аппаратному и программному обеспечению

Характеристика	Минимальные требования	Рекомендуемые требования
Количество worker-узлов	1	3
Процессор (на узел)	x64, 2 ядра	x64, 4 ядра
Оперативная память (на узел)	8 Гб	16 Гб
Диск (на узел)	HDD, 50 Гб	SSD, 100 Гб

Продолжение таблицы 4

Пропускная способность сети	1 Гбит/с	1 Гбит/с
Операционная система	Ubuntu 22.04	Ubuntu 22.04 и выше
Версия Kubernetes	1.28	1.30 и выше
Container runtime	containerd 1.7	containerd 1.7 и выше

Таким образом, осуществлено развёртывание компонентов разработанной системы в контейнерной среде с использованием оркестратора Kubernetes и определены минимальные и рекомендуемые требования к инфраструктуре.

4 Контроль качества программного обеспечения

4.1 Обоснование выбора модели тестирования и метрик кода

Для оценки качества разработанной информационной системы организации сбора средств на проекты были выбраны функциональное и нагрузочное тестирование, а также такие метрики кода, как метрики размер кода, цикломатическая сложность и метрики Холстеда.

Функциональное тестирование выбрано для проверки соответствия реализованных функций предъявляемым требованиям и ограничениям на входные данные. Разработанная система должна корректно обрабатывать как допустимые, так и неверные значения и формировать ожидаемый результат в проверяемых сценариях.

Нагрузочное тестирование выбрано для оценки производительности при одновременной работе множества пользователей. Оно позволяет проверить, что разработанная система сохраняет работоспособность и приемлемое время отклика под нагрузкой.

Метрики размера кода выбраны для оценки общего объёма программы, что даёт представление о масштабе реализации.

Цикломатическая сложность выбрана для оценки сложности логики функций разработанной программы. В исходном коде присутствуют функции с разветвлённой структурой, содержащие множество проверок и условных переходов, и данная метрика позволяет количественно оценить, насколько сложно они устроены и какие из них требуют наибольшего внимания.

Метрики Холстеда были выбраны для оценки структурной сложности кода разработанной программы. На основе числа уникальных и общих операторов и операндов исходного кода могут быть рассчитаны длина и словарь программы, её объём сложность и трудоёмкость.

4.2 Функциональное тестирование создания проекта

Рассмотрим функциональное тестирование разработанного приложения на примере сценария создания проекта. Для генерации тест-кейсов используется библиотека `allpairs.py` языка Python, которая позволяет сократить количество комбинаций входных переменных до меньшего набора, покрывающего большинство сценариев, за счёт применения метода попарного тестирования [42].

Исходные данные для генерации комбинаций входных параметров методом попарного тестирования приведены в таблице 5.

Таблица 5 – Исходные данные для генерации комбинаций входных данных

Параметр	Корректное значение	Некорректное значение	Проверяемое ограничение
Название	«Мой проект»	«»	Длина не менее 1
		«А» * 81	Длина не более 80
Описание	«Описание»	«А» * 10 001	Длина не более 10 000
Сумма сбора	10 000	0	Значение не менее 1
		100 000 001	Значение не более 100 000 000
Длительность в днях	30	0	Значение не менее 1
		61	Значение не более 60

Результат генерации входных параметров представлен на рисунке 36.

#	(1) Название	(2) Описание	(3) Сумма сбора	(4) Длительность
1	Мой проект	Описание	10000	30
2		А * 10 001	0	30
3	А * 81	А * 10 001	100000001	0
4	А * 81	Описание	0	61
5		Описание	100000001	61
6	Мой проект	А * 10 001	10000	61
7	Мой проект	Описание	100000001	0
8		Описание	10000	0
9	А * 81	Описание	10000	30
10	Мой проект	Описание	0	0
11	Мой проект	Описание	100000001	30

Итого тест-кейсов: 11

Рисунок 36 – Результат генерации входных параметров

Сформированные на основе комбинаций тест-кейсы и результаты тестирования приведены в приложении К.

В процессе функционального тестирования отклонений от ожидаемого поведения не выявлено, что подтверждает корректную работу сценария создания проекта.

4.3 Нагрузочное тестирование

Для проведения нагрузочного тестирования разработанного приложения выбран инструмент k6. Он оптимизирован для минимального потребления ресурсов и предназначен для проведения высоконагруженных тестов [43]. Инструмент также позволяет задавать пороговые значения для измеряемых метрик, определяя тем самым критерии успешности прохождения теста [44].

В качестве примера выбран тестовый сценарий, включающий последовательное выполнение следующих шагов: регистрация пользователя, вход в систему, получение ленты проектов, внесение взноса в первый проект ленты и выход из учетной записи.

Для теста заданы следующие пороговые значения:

- для запроса GET /projects p95 должен быть меньше 200 мс, а p99 – меньше 500 мс;
- для запроса POST /payments/contribute p95 должен быть меньше 2000 мс, а p99 – меньше 3000 мс;
- доля неуспешных запросов должна быть не более 1%.

Нагрузочное тестирование разработанного приложения проводилось на стенде с операционной системой Ubuntu 24.04.4 LTS, процессором AMD Ryzen 5 PRO 5650U и оперативной памятью 16 Гб. Средствами Kubernetes для микросервисов, развернутых на тестовом стенде, было настроено горизонтальное масштабирование до трех реплик.

Результат нагрузочного тестирования разработанной системы программой k6 представлен на рисунке 37.

THRESHOLDS

```
http_req_duration{step:contribute}
✓ 'p(95)<2000' p(95)=1.57s
✓ 'p(99)<3000' p(99)=1.92s
```

```
http_req_duration{step:projects}
✓ 'p(95)<200' p(95)=1.99ms
✓ 'p(99)<500' p(99)=23.57ms
```

```
http_req_failed
✓ 'rate<0.01' rate=0.00%
```

TOTAL RESULTS

```
checks_total.....: 2044    13.322302/s
checks_succeeded...: 100.00% 2044 out of 2044
checks_failed.....: 0.00%   0 out of 2044
```

```
✓ signup 2xx
✓ signin 200
✓ projects 200
✓ contribute 204
```

HTTP

```
http_req_duration.....: avg=3s      min=680.96µs med=1.1s      max=19.26s  p(90)=9.39s p(95)=11.32s
{ expected_response:true }...: avg=3s      min=680.96µs med=1.1s      max=19.26s  p(90)=9.39s p(95)=11.32s
{ step:contribute }.....: avg=900.4ms min=26.15ms  med=924.22ms max=2.29s   p(90)=1.43s p(95)=1.57s
{ step:projects }.....: avg=2.22ms  min=766.72µs med=1.46ms    max=146.27ms p(90)=1.75ms p(95)=1.99ms
http_req_failed.....: 0.00%   0 out of 2555
http_reqs.....: 2555    16.652878/s
```

EXECUTION

```
iteration_duration.....: avg=17.03s min=3.49s   med=18.76s   max=35.33s  p(90)=26.32s p(95)=28.4s
iterations.....: 511    3.330576/s
vus.....: 1      min=1      max=100
vus_max.....: 100    min=100    max=100
```

NETWORK

```
data_received.....: 5.5 MB 36 kB/s
data_sent.....: 601 kB 3.9 kB/s
```

Рисунок 37 – Результат нагрузочного тестирования программой k6

Как видно из рисунка 30, ни одно из заданных пороговых значений не превышено. Для запроса GET /projects значение p95 составило 1,99 мс, p99 – 23,57 мс. У запроса POST /payments/contribute p95 равно 1,57 с, p99 – 1,92 с. Доля неуспешных запросов равна 0%.

Нагрузочное тестирование подтвердило стабильное выполнение проверяемого сценария в условиях заданной нагрузки.

4.4 Метрики кода

4.4.1 Размер кода

Для оценки размера исходного кода проекта используется утилита cloc, которая позволяет подсчитать пустые строки, строки с комментариями и строки исходного кода [45]. Результат работы утилиты представлен на рисунке 38.

```

261 text files.
244 unique files.
31 files ignored.

github.com/AlDanial/cloc v 1.98 T=0.08 s (2926.6 files/s, 267664.4 lines/s)
-----
Language             files      blank      comment      code
-----
Go                    96         1407         145         6376
Svelte                25          813          26         5581
JSON                  8            0            0         4506
YAML                  41           37            7         1162
SQL                   51           70           64          912
TypeScript            7           91           40          590
Bourne Shell          2           29           12          115
CSS                   1           20            1          112
Markdown              7           31            0           80
Dockerfile            4           27            0           40
HTML                  1            0            0           11
JavaScript            1            2            0            9
-----
SUM:                  244         2527         295        19494
-----

```

Рисунок 38 – Результат работы утилиты cloc

Общий объем исходного кода составляет 19 494 строки. Наибольший вклад вносят Go (6 376 строк) и Svelte (5 581 строк) – серверная и клиентская части соответственно.

4.4.2 Цикломатическая сложность

Рассмотрим в качестве примера для расчета цикломатической сложности пакет `service.project` микросервиса проектов. Подсчет значений выполнен с помощью утилиты командной строки `gocyclo`, которая вычисляет цикломатическую сложность функций в исходном коде на Go [46].

Результат работы утилиты `gocyclo` представлен на рисунке 39.

```

13 project (*Service).GetProjects service/project/service.go:71:1
10 project (*Service).UploadProjectCover service/project/service.go:284:1
8 project validateCreateProjectReq service/project/service.go:413:1
8 project (*Service).UploadProjectImage service/project/service.go:346:1
8 project (*Service).GetProjectByID service/project/service.go:46:1
6 project (*Service).DeleteProjectImage service/project/service.go:383:1
6 project (*Service).BoostProject service/project/service.go:251:1
6 project (*Service).GetMyApplications service/project/service.go:216:1
6 project (*Service).UpdateProject service/project/service.go:162:1
6 project (*PublishWorker).runOnce service/project/publisher.go:60:1
5 project imageExtension service/project/service.go:331:1
5 project (*Service).SubmitProject service/project/service.go:122:1
4 project (*Service).GetProjectsByUserID service/project/service.go:236:1
4 project (*PublishWorker).Run service/project/publisher.go:44:1
4 project (*FinishWorker).Run service/project/finisher.go:34:1
3 project (*Service).CreateProject service/project/service.go:32:1
3 project NewPublishWorker service/project/publisher.go:26:1
3 project (*FinishWorker).runOnce service/project/finisher.go:50:1
2 project (*Service).TakeApplication service/project/service.go:207:1
2 project (*Service).GetPendingApplications service/project/service.go:198:1
2 project (*Service).GetApplicationByProjectID service/project/service.go:189:1
2 project (*Service).RejectProject service/project/service.go:153:1
2 project (*Service).ApproveProject service/project/service.go:144:1
2 project (*Service).AddContribution service/project/service.go:113:1
2 project NewFinishWorker service/project/finisher.go:22:1
1 project NewService service/project/service.go:23:1
Average: 4.73

```

Рисунок 39 – Результат работы утилиты gocyclo

По результатам расчета установлено, что среднее значение цикломатической сложности пакета `service.project` составляет 4,73. Наибольшее значение имеет функция `GetProjects()` – 13, что указывает на более сложную внутреннюю логику по сравнению с остальными функциями. В целом полученные результаты показывают, что большинство функций пакета обладают невысокой сложностью.

4.4.3 Метрики Холстеда

В качестве примера для расчета метрик Холстеда выбран пакет `service.payment` микросервиса платежей, содержащий основную бизнес-логику работы со взносами и выплатами.

Для подсчета метрик Холстеда требуется определить количество уникальных операторов и операндов в анализируемом участке программы, а также общее число их вхождений в этом коде. Операторы и операнды пакета `service.payment` приведены в таблицах 6 и 7 соответственно.

Таблица 6 – Операторы пакеты `service.payment`

Оператор	Количество
if	11
return	16
:=	10
=	1
!=	11
	1
*	2
GetProjectByID	2
Errorf	8
Add	1
Duration	1
Now	1
After	1
ValidateContribAmount	1

Продолжение таблицы 6

CreateContribution	1
PublishContribution	1
Printf	1
CreatePayout	1
GetPayout	1
GetProjectStats	1
GetUserStats	1
n1 = 21	N1 = 74

Таблица 7 – Операнды пакета service.payment

Операнд	Количество
Service	1
paymentRepo	1
r	1
projectClient	1
C	1
publisher	1
p	3
Err	28
ctx	8
req.ProjectID	2
nil	13
project client: %w	2
deadline	2
p.StartedAt	1
time.Hour	1
24	1
p.DurationDays	1
p.Status	1
project.StatusActive	1
domain.ErrProjectNotActive	1
req.Amount	2
validation: %w	1

Продолжение таблицы 7

contributionID	2
userID	5
req	1
repo: %w	5
event	2
contribution.Event	1
EventID	1
Type	1
contribution.EventTypeCreated	1
ProjectID	1
Amount	1
contribution publisher: %v	1
projectID	4
payout	2
domain.Payout	1
domain.ProjectStats	3
p.UserID	1
domain.ErrNotProjectOwner	1
stats	4
domain.UserStats	1
n2 = 42	N2 = 113

Длина программы рассчитывается по формуле:

$$N = N_1 + N_2,$$

где N_1 – общее число операторов в программе;

N_2 – общее число операндов в программе.

На основе данных таблиц 6 и 7 длина программы составляет:

$$N = 74 + 113 = 187.$$

Словарь программы определяется по формуле:

$$n = n_1 + n_2,$$

где n_1 – число уникальных операторов в программе;

n_2 – число уникальных операндов в программе.

Таким образом, на основе данных из таблиц 6 и 7 словарь программы:

$$n = 21 + 42 = 63.$$

Объём программы V рассчитывается по формуле:

$$V = N * \log_2 n.$$

Исходя из рассчитанных значений N и n объём программы составляет:

$$V = 187 * \log_2 63 = 1117,7.$$

Сложность программы D рассчитывается по формуле:

$$D = (n_1 / 2) * (N_2 / n_2).$$

На основе таблиц 6 и 7 сложность программы:

$$D = (21 / 2) * (113 / 42) = 28,25.$$

Трудоёмкость разработки E определяется по формуле:

$$E = D * V.$$

Используя рассчитанные D и V , трудоёмкость разработки составляет:

$$E = 28,25 * 1117,70 = 31570.$$

Полученные значение метрик Холстеда свидетельствуют о том, что код пакета `service.payment` имеет среднюю структурную сложность и несложен для сопровождения.

Заключение

В рамках выпускной квалификационной работы спроектирована и реализована информационная система организации сбора средств на проекты, включающая серверную часть и клиентское веб-приложение.

Серверная часть разработана на языке Go и построена по принципам микросервисной архитектуры. Система включает 4 микросервиса, каждый из которых реализует отдельную бизнес-функцию: управление пользователями, проектами, платежами и промокодами. Маршрутизация входящих запросов обеспечивается ingress-контроллером Traefik, а развертывание приложения средствами Kubernetes.

В ходе работы проведён анализ предметной области, выполнен сравнительный анализ существующих решений, определены функциональные требования к системе. Определены роли пользователей и сценарии их взаимодействия с платформой.

На этапе проектирования выбран архитектурный стиль, спроектированы модели данных, определена архитектура микросервиса разрабатываемой системы, а также построена карта сайта клиентского приложения.

В процессе реализации разработаны 4 микросервиса, базы данных под управлением PostgreSQL, кэширование на основе Redis, асинхронное межсервисное взаимодействие через Apache Kafka, а также хранение изображений с использованием MinIO. Для мониторинга системы настроен стек мониторинга на базе Prometheus и Grafana. Разработано клиентское приложение на основе фреймворка Svelte, обладающее удобным пользовательским интерфейсом.

Проведён контроль качества с применением функционального тестирования на основе метода попарного тестирования, нагрузочного тестирования и определением метрик кода.

Выполнено планирование разработки и оценка затрат. Итоговые денежные затраты составили 98 400 рублей, трудозатраты – 328 часа.

Таким образом, в результате выполнения выпускной квалификационной работы разработана информационная система организации сбора средств на проекты, отвечающая предъявляемым требованиям.

Список использованных источников

1. Мотовилов О.В. Феномен краудфандинга: исследование особенностей // Вестник Санкт-Петербургского университета. Экономика. 2018. Т. 34. № 2. С. 298-316. [Электронный ресурс]. URL: <https://economicsjournal.spbu.ru/article/view/1044/886> (дата обращения: 16.05.2026)
2. Центральный банк Российской Федерации. Обзор платформенных сервисов в России: операторы инвестиционных платформ, операторы информационных систем и операторы финансовых платформ. 2023 год – I квартал 2024 года: информационно-аналитический материал. М.: Банк России, 2024. 13 с. [Электронный ресурс]. URL: https://www.cbr.ru/Collection/Collection/File/49243/platform_services_2024-1.pdf (дата обращения: 16.05.2026)
3. Planeta.ru – российская краудфандинговая платформа // Planeta.ru [Электронный ресурс]. URL: <https://planeta.ru/> (дата обращения: 18.05.2026)
4. What is Kickstarter? // Kickstarter Support [Электронный ресурс]. URL: <https://help.kickstarter.com/hc/en-us/articles/115004996453-What-is-Kickstarter> (Дата обращения: 18.05.2026)
5. Who can use Kickstarter? // Kickstarter Support [Электронный ресурс]. URL: <https://help.kickstarter.com/hc/en-us/articles/115005128594-Who-can-use-Kickstarter> (дата обращения: 16.05.2026)
6. Правила // Boomstarter [Электронный ресурс]. URL: <https://boomstarter.ru/help/guidelines> (дата обращения: 18.05.2026)
7. GoFundMe | The #1 Crowdfunding and Fundraising Platform // GoFundMe [Электронный ресурс]. URL: <https://www.gofundme.com/> (дата обращения: 16.05.2026)
8. Countries supported on GoFundMe // GoFundMe Help Center [Электронный ресурс]. URL: <https://support.gofundme.com/hc/en-us/articles/360001972748-Countries-supported-on-GoFundMe> (дата обращения: 16.05.2026)

9. What is Use Case Diagram? // Visual Paradigm [Электронный ресурс]. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/> (дата обращения: 16.05.2026)
10. Ричардс, М. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / М. Ричардс, Н. Форд – СПб.: Питер, 2023. 448 с.
11. Work with the Gantt view // Microsoft Support [Электронный ресурс]. URL: <https://support.microsoft.com/en-US/project/work-with-the-gantt-chart-view> (дата обращения: 14.05.2026)
12. Lewis J., Fowler M. Microservices // martinowler.com 25 March 2014 [Электронный ресурс]. URL: <https://martinfowler.com/articles/microservices.html> (дата обращения: 14.05.2026)
13. What is Component Diagram? // Visual Paradigm [Электронный ресурс]. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/> (дата обращения: 16.05.2026)
14. Traefik, The Cloud Native Application Proxy // Traefik Labs [Электронный ресурс]. URL: <https://traefik.io/traefik> (дата обращения: 31.05.2026)
15. Traefik HTTP Routers Rules & Priority Documentation // Traefik [Электронный ресурс]. URL: <https://doc.traefik.io/traefik/reference/routing-configuration/http/routing/rules-and-priority/> (дата обращения: 08.05.2026)
16. Traefik Configuration Documentation // Traefik [Электронный ресурс]. URL: <https://doc.traefik.io/traefik/getting-started/configuration-overview/> (дата обращения: 08.05.2026)
17. PostgreSQL: About // PostgreSQL [Электронный ресурс]. URL: <https://www.postgresql.org/about/> (дата обращения: 14.05.2026)
18. Cache-Aside Pattern with Redis: Query Caching for Microservices // Redis [Электронный ресурс]. URL: <https://redis.io/tutorials/howtos/solutions/microservices/caching/> (дата обращения: 08.05.2026)

19. Developers // MinIO AIStor Documentation [Электронный ресурс]. URL: <https://docs.min.io/aistor/developers/> (дата обращения: 08.05.2026)
20. Introduction // Apache Kafka [Электронный ресурс]. URL: <https://kafka.apache.org/intro/> (дата обращения: 08.05.2026)
21. Overview // Prometheus [Электронный ресурс]. URL: <https://prometheus.io/docs/introduction/overview/> (дата обращения: 08.05.2026)
22. Client libraries // Prometheus [Электронный ресурс]. URL: <https://prometheus.io/docs/instrumenting/clientlibs/> (дата обращения: 08.05.2026)
23. Exporters and integrations // Prometheus [Электронный ресурс]. URL: <https://prometheus.io/docs/instrumenting/exporters/> (дата обращения: 08.05.2026)
24. Introduction // Grafana documentation [Электронный ресурс]. URL: <https://grafana.com/docs/grafana/latest/fundamentals/> (дата обращения: 08.05.2026)
25. Prometheus data source // Grafana documentation [Электронный ресурс]. URL: <https://grafana.com/docs/grafana/latest/datasources/prometheus/> (дата обращения: 08.05.2026)
26. Overview // Kubernetes Documentation [Электронный ресурс]. URL: <https://kubernetes.io/docs/concepts/overview/> (дата обращения: 16.05.2026)
27. What is Sequence Diagram? // Visual Paradigm [Электронный ресурс]. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/> (дата обращения: 16.05.2026)
28. What Are Conceptual, Logical, and Physical Data Models? Redgate Blog. 9 February 2021 [Электронный ресурс]. URL: <https://www.redgate.com/blog/conceptual-logical-physical-data-model/> (дата обращения: 12.05.2026)
29. Fowler M. Presentation Domain Data Layering // martinowler.com. 26 August 2015 [Электронный ресурс]. URL: <https://martinfowler.com/bliki/PresentationDomainDataLayering.html> (дата обращения: 12.05.2026)
30. Tankala S. Information Architecture vs. Sitemaps: Whats` the Difference? // Nielsen Norman Group. 3 September 2023 [Электронный ресурс].

<https://www.nngroup.com/articles/information-architecture-sitemaps/> (дата обращения: 09.05.2026)

31. akemoon/crowdfunding-app-user // GitHub [Электронный ресурс]. URL: <https://github.com/akemoon/crowdfunding-app-user> (дата обращения: 01.06.2026)

32. akemoon/crowdfunding-app-project // GitHub [Электронный ресурс]. URL: <https://github.com/akemoon/crowdfunding-app-project> (дата обращения: 01.06.2026)

33. akemoon/crowdfunding-app-payment // GitHub [Электронный ресурс]. URL: <https://github.com/akemoon/crowdfunding-app-payment> (дата обращения: 01.06.2026)

34. akemoon/crowdfunding-app-promo // GitHub [Электронный ресурс]. URL: <https://github.com/akemoon/crowdfunding-app-promo> (дата обращения: 01.06.2026)

35. akemoon/crowdfunding-app-front // GitHub [Электронный ресурс]. URL: <https://github.com/akemoon/crowdfunding-app-front> (дата обращения: 01.06.2026)

36. Kubernetes Components // Kubernetes Documentation [Электронный ресурс]. URL: <https://kubernetes.io/docs/concepts/overview/components/> (дата обращения: 28.05.2026)

37. Deployments // Kubernetes Documentation [Электронный ресурс]. URL: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/> (дата обращения: 01.06.2026)

38. Service ClusterIP allocation // Kubernetes Documentation [Электронный ресурс]. URL: <https://kubernetes.io/docs/concepts/services-networking/cluster-ip-allocation/> (дата обращения: 01.06.2026)

39. Horizontal Pod Autoscaling // Kubernetes Documentation [Электронный ресурс]. URL: <https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/> (дата обращения: 01.06.2026)

40. StatefulSets // Kubernetes Documentation [Электронный ресурс]. URL: <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/> (дата обращения: 01.06.2026)
41. akemoon/crowdfunding-app-infra // GitHub [Электронный ресурс]. URL: <https://github.com/akemoon/crowdfunding-app-infra> (дата обращения: 01.06.2026)
42. allpairsy // PyPI [Электронный ресурс]. URL: <https://pypi.org/project/allpairsy/> (дата обращения: 21.05.26)
43. Grafana k6 // Grafana k6 documentation [Электронный ресурс]. URL: <https://grafana.com/docs/k6/latest/> (дата обращения: 23.05.2026)
44. Thresholds |// Grafana k6 documentation [Электронный ресурс]. URL: <https://grafana.com/docs/k6/latest/using-k6/thresholds/> (дата обращения: 23.05.2026)
45. cloc/README.md at master // GitHub [Электронный ресурс]. URL: <https://github.com/AIDanial/cloc/blob/master/README.md> (дата обращения: 28.05.26)
46. gocyclo package // Go Packages [Электронный ресурс]. URL: <https://pkg.go.dev/github.com/fzipp/gocyclo> (дата обращения: 21.01.26)

Приложение А
(справочное)
Глоссарий

Golang – компилируемый язык программирования с открытым исходным кодом, разработанный компанией Google и предназначенный для создания высокопроизводительных серверных приложений.

TypeScript – язык программирования, расширяющий JavaScript за счёт строгой статической типизации.

SvelteKit – фреймворк для разработки веб-приложений на основе Svelte с поддержкой серверного рендеринга.

Kubernetes (k8s) – система оркестрации контейнеров с открытым исходным кодом для автоматизации развёртывания, масштабирования и управления контейнеризированными приложениями.

Pod – минимальная единица развёртывания в Kubernetes, содержащая один или несколько контейнеров.

Deployment – ресурс Kubernetes, описывающий желаемое состояние набора подов и управляющий их обновлением.

StatefulSet – ресурс Kubernetes для управления подами с сохранением состояния и уникальными идентификаторами.

Prometheus – система мониторинга с открытым исходным кодом для сбора и хранения метрик в виде временных рядов.

Grafana – платформа для визуализации и анализа метрик, поддерживающая Prometheus в качестве источника данных.

Traefik – обратный прокси-сервер и ingress-контроллер с открытым исходным кодом.

Redis – хранилище данных в оперативной памяти, используемое для кэширования.

PostgreSQL – объектно-реляционная система управления базами данных с открытым исходным кодом.

Apache Kafka – распределённый брокер сообщений для асинхронного обмена событиями между сервисами.

MinIO – объектное хранилище с открытым исходным кодом, совместимое с Amazon S3 API.

Приложение Б
(обязательное)
Скрипты для создания таблиц баз данных

Листинг Б.1 – Файл 0001_init.sql микросервиса пользователей

```

-- +goose Up

create table if not exists roles
(
    id    smallint primary key,
    name  text not null,

    constraint roles_name_unique unique (name)
);

insert into roles (id, name) values
(1, 'user'),
(2, 'moder'),
(3, 'admin');

create table if not exists default_avatars
(
    id    smallserial primary key,
    key   text not null,

    constraint default_avatars_key_unique unique (key)
);

insert into default_avatars (key) values
('defaults/avatar-1.png'),
('defaults/avatar-2.png'),
('defaults/avatar-3.png'),
('defaults/avatar-4.png'),
('defaults/avatar-5.png'),
('defaults/avatar-6.png');

create table if not exists users
(
    id                uuid primary key default uuidv7(),
    username          text not null,
    display_name      text,
    description       text,
    avatar_key        text,
    default_avatar_id smallint not null,
    created_at        timestampz not null default now(),
    updated_at        timestampz not null default now(),

    constraint users_username_unique unique (username),
    constraint users_default_avatar_fk foreign key
(default_avatar_id) references default_avatars(id)
);

-- +goose StatementBegin
create or replace function assign_default_avatar()
returns trigger as $$
begin

```

```

        select id into new.default_avatar_id
        from default_avatars
        order by random()
        limit 1;

        return new;
end;
$$ language plpgsql;
-- +goose StatementEnd

create trigger trigger_assign_default_avatar
    before insert on users
    for each row
    execute function assign_default_avatar();

-- +goose StatementBegin
create or replace function set_updated_at()
returns trigger as $$
begin
    new.updated_at = now();
    return new;
end;
$$ language plpgsql;
-- +goose StatementEnd

create trigger trigger_users_updated_at
    before update on users
    for each row
    execute function set_updated_at();

create table if not exists credentials
(
    user_id          uuid primary key,
    email            text not null,
    password_hash    text not null,
    role_id          smallint not null default 1,
    is_blocked       boolean not null default false,
    created_at       timestampz not null default now(),
    updated_at       timestampz not null default now(),

    constraint credentials_email_unique unique (email),
    constraint credentials_role_fk foreign key (role_id)
references roles(id),
    constraint credentials_user_fk foreign key (user_id)
references users(id)
);

create trigger trigger_credentials_updated_at
    before update on credentials
    for each row
    execute function set_updated_at();

create table if not exists follows

```

```
(
    follower_id uuid,
    followee_id uuid,
    created_at timestamptz not null default now(),

    constraint follows_pk primary key (follower_id, followee_id),
    constraint follows_no_to_yourself check (follower_id <>
followee_id),
    constraint follows_follower_fk foreign key (follower_id)
references users(id),
    constraint follows_followee_fk foreign key (followee_id)
references users(id)
);

-- +goose Down

drop table if exists follows;
drop trigger if exists trigger_credentials_updated_at on
credentials;
drop table if exists credentials;
drop trigger if exists trigger_users_updated_at on users;
drop trigger if exists trigger_assign_default_avatar on users;
drop function if exists set_updated_at;
drop function if exists assign_default_avatar;
drop table if exists users;
drop table if exists default_avatars;
drop table if exists roles;
```

Листинг Б.2 – Файл 0001_init.sql микросервиса проектов

```
-- +goose Up

create table if not exists currencies
(
    id smallint primary key,
    name char(3) not null unique
);

insert into currencies (id, name) values
(1, 'RUB'),
(2, 'USD');

create table if not exists project_categories
(
    id smallint primary key,
    name text not null unique
);

insert into project_categories (id, name) values
(1, 'science'),
(2, 'tech'),
(3, 'architecture_and_urban'),
(4, 'sport');
```

```
(5, 'music'),
(6, 'art'),
(7, 'film'),
(8, 'games'),
(9, 'education'),
(10, 'food'),
(11, 'fashion'),
(12, 'health');
```

```
create table if not exists project_statuses
(
    id      smallint primary key,
    name    text not null unique
);
```

```
insert into project_statuses (id, name) values
(1, 'review'),
(2, 'active'),
(3, 'finished'),
(4, 'draft');
```

```
create table if not exists projects
(
    id                uuid          primary key default uuidv7(),
    user_id           uuid          not null,
    category_id       smallint      not null references
project_categories(id),
    name              text          not null,
    description        text          not null,
    currency_id       smallint      not null references currencies(id),
    goal_amount       bigint        not null check (goal_amount > 0),
    current_amount    bigint        not null default 0 check
(current_amount >= 0),
    started_at        timestamptz,
    duration_days     smallint      not null,
    status_id         smallint      not null default 1 references
project_statuses(id),
    boosted_until     timestamptz,
    finished_at       timestamptz,
    cover_key         text,
    constraint projects_user_id_name_unique unique (user_id, name)
);
```

```
create table if not exists project_images
(
    id                uuid          primary key default uuidv7(),
    project_id        uuid          not null references projects(id) on
delete cascade,
    storage_key       text          not null unique,
    created_at        timestamptz not null default now()
);
```

```
create table if not exists project_application_statuses
```

```

(
    id    smallint primary key,
    name  text not null unique
);

insert into project_application_statuses (id, name) values
(1, 'pending'),
(2, 'review'),
(3, 'rejected'),
(4, 'approved');

-- project_applications is a separate entity (not just a status on
projects) because:
-- 1. it carries manager-specific fields (assigned_to,
reject_reason, timestamps)
-- 2. in the future it should become 1:many - one project may
have multiple applications
--    (e.g. re-submission after rejection), each with its own
lifecycle and history
create table if not exists project_applications
(
    project_id    uuid          primary key references projects(id),
    status_id     smallint      not null default 1 references
project_application_statuses(id),
    assigned_to   uuid,
    reject_reason text          not null default '',
    created_at    timestamptz   not null default now(),
    assigned_at   timestamptz,
    processed_at  timestamptz,
    updated_at    timestamptz   not null default now()
);

create table if not exists finished_projects_outbox_statuses
(
    id    smallint primary key,
    name  text not null unique
);

insert into finished_projects_outbox_statuses (id, name) values
(1, 'pending'),
(2, 'sent');

create table if not exists finished_projects_outbox
(
    id            bigserial      primary key,
    project_id    uuid          not null references projects(id),
    status_id     smallint      not null default 1 references
finished_projects_outbox_statuses(id),
    created_at    timestamptz   not null default now(),
    locked_until  timestamptz
);

-- +goose Down

```

```

drop table if exists finished_projects_outbox;
drop table if exists finished_projects_outbox_statuses;
drop table if exists project_applications;
drop table if exists project_application_statuses;
drop table if exists project_images;
drop table if exists projects;
drop table if exists project_statuses;
drop table if exists project_categories;
drop table if exists currencies;

```

Листинг Б.3 – Файл 0001_init.sql микросервиса платежей

```

-- +goose Up

create table if not exists contributions
(
    id          uuid primary key default uuidv7(),
    user_id     uuid not null,
    project_id  uuid not null,
    amount      bigint not null,
    created_at  timestampz default now(),

    constraint contributions_amount_check check (amount > 0)
);

create table if not exists payouts
(
    project_id  uuid,
    -- user_id is denormalized from the project service so that
payouts can be filtered
    -- by owner without a cross-service call on every request
    user_id     uuid not null,
    created_at  timestampz default now(),

    constraint payouts_pk primary key (project_id)
);

-- +goose Down

drop table if exists payouts;
drop table if exists contributions;

```

Листинг Б.4 – Файл 0001_init.sql микросервиса промокодов

```

-- +goose Up

create table if not exists promo_types (
    id      smallint primary key,
    name text      not null unique
);

```

```

insert into promo_types (id, name) values (1, 'boost_project');

create table if not exists promo_codes (
    code          varchar(10),
    type_id       smallint      not null references promo_types(id),
    owner_id      uuid          not null,
    effect_value  int           not null,
    used_at       timestamptz,
    created_at    timestamptz not null default now(),

    constraint promo_codes_pk primary key (code)
);

create index if not exists promo_codes_owner_id_index on
promo_codes (owner_id);

-- tracks processed kafka events for consumer idempotency
create table if not exists processed_events (
    event_id      uuid          not null,
    created_at    timestamptz not null default now(),

    constraint processed_events_pk primary key (event_id)
);

-- +goose Down

drop table if exists processed_events;
drop table if exists promo_codes;
drop table if exists promo_types;

```

Приложение В
(обязательное)

Фрагменты исходного кода микросервиса пользователей

Листинг В.1 – Метод AddAuthHandlers

```
func (s *Server) AddAuthHandlers(svc *auth.Service, m
*metrics.AuthMetrics) {
    s.r.HandleFunc("POST /auth/signup", authHandler.SignUp(svc))
    s.r.HandleFunc("POST /auth/signin", authHandler.SignIn(svc,
m))
    s.r.HandleFunc("POST /auth/signout", authHandler.SignOut(svc))
    s.r.HandleFunc("GET /auth/check",
authHandler.CheckAccess(svc))
    s.r.HandleFunc("POST /auth/refresh", authHandler.Refresh(svc))
    s.r.HandleFunc("GET /auth/me", authHandler.GetMe(svc))
    s.r.HandleFunc("GET /auth/users/{id}",
authHandler.GetCredentialsByID(svc))
    s.r.HandleFunc("PATCH /auth/users/{id}/role",
authHandler.UpdateRole(svc))
    s.r.HandleFunc("PATCH /auth/users/{id}/blocked",
authHandler.SetBlocked(svc))
}
```

Листинг В.2 – Метод AddUserHandlers

```
func (s *Server) AddUserHandlers(svc *user.Service) {
    s.r.HandleFunc("GET /users/search",
userHandler.SearchUsers(svc))
    s.r.HandleFunc("GET /users/{id}",
userHandler.GetUserByID(svc))
    s.r.HandleFunc("GET /users/me", userHandler.GetMe(svc))
    s.r.HandleFunc("PATCH /users/me/profile",
userHandler.UpdateProfile(svc))

    s.r.HandleFunc("POST /users/{id}/follow",
userHandler.Follow(svc))
    s.r.HandleFunc("DELETE /users/{id}/follow",
userHandler.Unfollow(svc))
    s.r.HandleFunc("GET /users/me/subscriptions",
userHandler.GetMeSubscriptions(svc))
}
```

Листинг В.3 – Метод AddMetrics

```
func (s *Server) AddMetrics() {
    s.r.Handle("/metrics", promhttp.Handler())
}
```

Листинг В.4 – Файл modules/auth/repo/auth/repo.go

```
package user

import (
    "context"
```

```

        "github.com/akemoon/crowdfunding-app-user/lib/domain"
        "github.com/google/uuid"
    )

    type Repo interface {
        CreateUser(ctx context.Context, req domain.CreateUserReq)
        (uuid.UUID, error)
        GetCredentialsByEmail(ctx context.Context, email string)
        (domain.UserCredentials, error)
        GetCredentialsByID(ctx context.Context, id uuid.UUID)
        (domain.UserCredentials, error)
        UpdateRole(ctx context.Context, userID uuid.UUID, role string)
        error
        SetBlocked(ctx context.Context, userID uuid.UUID, blocked
        bool) error
    }
}

```

Листинг В.5 – Файл modules/auth/repo/refresh/repo.go

```

package refresh

import (
    "context"
    "time"

    "github.com/google/uuid"
)

type Repo interface {
    Set(ctx context.Context, hash string, userID uuid.UUID, ttl
    time.Duration) error
    Get(ctx context.Context, hash string) (uuid.UUID, error)
    Delete(ctx context.Context, hash string) error
    DeleteAllByUserID(ctx context.Context, userID uuid.UUID) error
}

```

Листинг В.6 – Файл modules/user/repo/user/repo.go

```

package user

import (
    "context"

    "github.com/akemoon/crowdfunding-app-user/modules/user/domain"
    "github.com/google/uuid"
)

type Repo interface {
    GetUserByID(ctx context.Context, id uuid.UUID) (domain.User,
    error)
    UpdateProfile(ctx context.Context, userID uuid.UUID, req
    domain.UpdateProfileReq) (domain.User, error)
}

```

```

    Follow(ctx context.Context, followerID uuid.UUID, followeeID
uuid.UUID) error
    Unfollow(ctx context.Context, followerID uuid.UUID, followeeID
uuid.UUID) error
    SearchUsers(ctx context.Context, req domain.SearchUsersReq)
([]domain.User, error)
    GetSubscriptions(ctx context.Context, userID uuid.UUID)
([]domain.User, error)
}

```

Листинг В.7 – Метод SignIn

```

func (s *Service) SignIn(ctx context.Context, req
domain.SignInReq) (domain.SignInResp, error) {
    creds, err := s.userRepo.GetCredentialsByEmail(ctx, req.Email)
    if err != nil {
        return domain.SignInResp{}, err
    }

    err = s.hasher.Compare(req.Password, creds.PasswordHash)
    if err != nil {
        return domain.SignInResp{}, lib.ErrInvalidCredentials
    }

    if creds.IsBlocked {
        return domain.SignInResp{}, domain.ErrUserBlocked
    }

    claims := domain.TokenClaims{UserID: creds.UserID, Role:
creds.Role}

    accessToken, err := s.tokenSvc.GenerateAccessToken(claims)
    if err != nil {
        return domain.SignInResp{}, fmt.Errorf("token service:
%w", err)
    }

    refreshToken, err := s.tokenSvc.GenerateRefreshToken(ctx,
creds.UserID)
    if err != nil {
        return domain.SignInResp{}, fmt.Errorf("token service:
%w", err)
    }

    return domain.SignInResp{
        AccessToken: accessToken,
        RefreshToken: refreshToken,
    }, nil
}

```

Приложение Г

(обязательное)

Фрагменты исходного кода микросервиса проектов

Листинг Г.1 – Метод AddProjectHandlers

```
func (s *Server) AddProjectHandlers(svc *project.Service) {
    s.r.HandleFunc("POST /projects", handler.CreateProject(svc))
    s.r.HandleFunc("GET /projects", handler.GetProjects(svc))
    s.r.HandleFunc("GET /projects/user",
handler.GetMyProjects(svc))
    s.r.HandleFunc("GET /projects/user/{id}",
handler.GetProjectsByUserID(svc))
    s.r.HandleFunc("GET /projects/{id}",
handler.GetProjectByID(svc))
    s.r.HandleFunc("PUT /projects/{id}",
handler.UpdateProject(svc))
    s.r.HandleFunc("POST /projects/{id}/submit",
handler.SubmitProject(svc))
    s.r.HandleFunc("POST /projects/{id}/boost",
handler.BoostProject(svc))
    s.r.HandleFunc("POST /projects/{id}/cover",
handler.UploadProjectCover(svc))
    s.r.HandleFunc("POST /projects/{id}/images",
handler.UploadProjectImage(svc))
    s.r.HandleFunc("DELETE /projects/{id}/images/{imageID}",
handler.DeleteProjectImage(svc))

    s.r.HandleFunc("GET /applications",
handler.GetPendingApplications(svc))
    s.r.HandleFunc("GET /applications/moderator",
handler.GetMyApplications(svc))
    s.r.HandleFunc("GET /applications/project/{id}",
handler.GetApplicationByProjectID(svc))
    s.r.HandleFunc("POST /applications/{id}/take",
handler.TakeApplication(svc))
    s.r.HandleFunc("POST /applications/{id}/approve",
handler.ApproveProject(svc))
    s.r.HandleFunc("POST /applications/{id}/reject",
handler.RejectProject(svc))
}
```

Листинг Г.2 – Файл repo/project.go

```
package project

import (
    "context"

    "github.com/akemoon/crowdfunding-app-project/domain"
    "github.com/google/uuid"
)

type Repo interface {
    CreateProject(ctx context.Context, userID uuid.UUID, req
domain.CreateProjectReq) (uuid.UUID, error)
```

```

    SubmitProject(ctx context.Context, id uuid.UUID, userID
uuid.UUID) error
    GetProjectByID(ctx context.Context, id uuid.UUID)
(domain.Project, error)
    GetProjects(ctx context.Context, req domain.GetProjectsReq)
([]domain.Project, error)
    GetProjectsByUserID(ctx context.Context, authorID uuid.UUID,
isOwner bool) ([]domain.Project, error)
    FinishProjects(ctx context.Context) error
    ListPendingFinishedOutbox(ctx context.Context, limit int)
([]domain.Project, error)
    MarkSentFinishedOutbox(ctx context.Context, projectID
uuid.UUID) error
    AddContribution(ctx context.Context, projectID uuid.UUID,
amount int64) error

    UpdateProject(ctx context.Context, userID uuid.UUID, id
uuid.UUID, req domain.CreateProjectReq) error

    ApproveProject(ctx context.Context, id uuid.UUID) error
    RejectProject(ctx context.Context, id uuid.UUID, reason
string) error
    GetApplicationByProjectID(ctx context.Context, projectID
uuid.UUID, userID uuid.UUID) (domain.Application, error)
    GetPendingApplications(ctx context.Context)
([]domain.Application, error)
    TakeApplication(ctx context.Context, projectID uuid.UUID,
managerID uuid.UUID) error
    GetMyApplications(ctx context.Context, managerID uuid.UUID)
([]domain.Application, error)

    BoostProject(ctx context.Context, userID uuid.UUID, projectID
uuid.UUID, days int) error

    SetProjectCover(ctx context.Context, projectID uuid.UUID,
userID uuid.UUID, key string) error
    AddProjectImage(ctx context.Context, projectID uuid.UUID,
storageKey string) (uuid.UUID, error)
    GetProjectImage(ctx context.Context, imageID, projectID
uuid.UUID) (domain.ProjectImage, error)
    DeleteProjectImage(ctx context.Context, imageID, projectID
uuid.UUID) error
}

```

Листинг Г.3 – Метод CreateProject

```

func (s *Service) CreateProject(ctx context.Context, userID
uuid.UUID, req domain.CreateProjectReq) (uuid.UUID, error) {
    err := validateCreateProjectReq(req)
    if err != nil {
        return uuid.UUID{}, err
    }
}

```

```

    id, err := s.repo.CreateProject(ctx, userID, req)
    if err != nil {
        return uuid.UUID{}, fmt.Errorf("repo: %w", err)
    }

    return id, nil
}

```

Листинг Г.4 – Метод GetProjects

```

func (s *Service) GetProjects(ctx context.Context, req
domain.GetProjectsReq) (domain.GetProjectsResp, error) {
    if req.Status != domain.StatusActive && req.Status !=
domain.StatusFinished {
        return domain.GetProjectsResp{}, domain.ErrUnknownStatus
    }

    if req.Sort == "" {
        req.Sort = domain.SortDefault
    } else {
        err := domain.ValidateSort(req.Sort)
        if err != nil {
            return domain.GetProjectsResp{}, err
        }
    }

    if req.Category != nil {
        err := domain.ValidateCategory(*req.Category)
        if err != nil {
            return domain.GetProjectsResp{}, err
        }
    }

    if req.Limit <= 0 || req.Limit > 100 {
        req.Limit = 20
    }
    if req.Offset < 0 {
        req.Offset = 0
    }

    items, err := s.repo.GetProjects(ctx, req)
    if err != nil {
        return domain.GetProjectsResp{}, fmt.Errorf("repo: %w",
err)
    }

    for i := range items {
        if items[i].CoverKey != nil {
            items[i].CoverURL = s.imagesBaseURL + "/" +
*items[i].CoverKey
        }
    }
}

```

```

    return domain.GetProjectsResp{Items: items}, nil
}

```

Листинг Г.5 – Метод BoostProject

```

func (s *Service) BoostProject(ctx context.Context, userID
uuid.UUID, projectID uuid.UUID, promoCode string) error {
    p, err := s.repo.GetProjectByID(ctx, projectID)
    if err != nil {
        return fmt.Errorf("repo: %w", err)
    }

    if p.UserID != userID {
        return domain.ErrForbidden
    }

    if p.Status != domain.StatusActive {
        return domain.ErrProjectNotActive
    }

    days, err := s.promo.UsePromoCode(ctx, userID, promoCode,
"boost_project")
    if err != nil {
        return err
    }

    err = s.repo.BoostProject(ctx, userID, projectID, days)
    if err != nil {
        return fmt.Errorf("repo: %w", err)
    }

    return nil
}

```

Приложение Д
(обязательное)

Фрагменты исходного кода микросервиса платежей

Листинг Д.1 – Метод AddPaymentHandlers

```
func (s *Server) AddPaymentHandlers(svc *payment.Service) {
    s.r.HandleFunc("POST /payments/contribute",
handler.Contribute(svc))
    s.r.HandleFunc("GET /payments/payouts/{projectID}",
handler.GetPayout(svc))
    s.r.HandleFunc("GET /payments/stats/project/{id}",
handler.GetProjectStats(svc))
    s.r.HandleFunc("GET /payments/stats/user/{id}",
handler.GetUserStats(svc))
}
```

Листинг Д.2 – Файл repo/payment/repo.go

```
package payment

import (
    "context"

    "github.com/akemoon/crowdfunding-app-payment/domain"
    "github.com/google/uuid"
)

type Repo interface {
    CreateContribution(ctx context.Context, userID uuid.UUID, req
domain.ContributeReq) (uuid.UUID, error)
    CreatePayout(ctx context.Context, projectID uuid.UUID, userID
uuid.UUID) error
    GetPayout(ctx context.Context, projectID uuid.UUID, userID
uuid.UUID) (domain.Payout, error)
    GetProjectStats(ctx context.Context, projectID uuid.UUID)
(domain.ProjectStats, error)
    GetUserStats(ctx context.Context, userID uuid.UUID)
(domain.UserStats, error)
}
```

Листинг Д.3 – Метод Contribute

```
func (s *Service) Contribute(ctx context.Context, userID
uuid.UUID, req domain.ContributeReq) error {
    err := domain.ValidateContribAmount(req.Amount)
    if err != nil {
        return fmt.Errorf("validation: %w", err)
    }

    p, err := s.projectClient.GetProjectByID(ctx, req.ProjectID)
    if err != nil {
        return fmt.Errorf("project client: %w", err)
    }
}
```

```

        deadline := p.StartedAt.Add(time.Hour * 24 *
time.Duration(p.DurationDays))
        if p.Status != project.StatusActive ||
time.Now().After(deadline) {
            return domain.ErrProjectNotActive
        }

        contributionID, err := s.paymentRepo.CreateContribution(ctx,
userID, req)
        if err != nil {
            return fmt.Errorf("repo: %w", err)
        }

        // At-most-once: event may not be delivered if publish fails.
        event := contribution.Event{
            EventID:    contributionID,
            Type:        contribution.EventTypeCreated,
            ProjectID: req.ProjectID,
            Amount:      req.Amount,
        }
        err = s.publisher.PublishContribution(ctx, event)
        if err != nil {
            log.Printf("contribution publisher: %v", err)
        }

        return nil
    }
}

```

Листинг Д.4 – Метод Payout

```

func (s *Service) Payout(ctx context.Context, projectID uuid.UUID,
userID uuid.UUID) error {
    err := s.paymentRepo.CreatePayout(ctx, projectID, userID)
    if err != nil {
        return fmt.Errorf("repo: %w", err)
    }

    return nil
}

```

Приложение Е

(обязательное)

Фрагменты исходного кода микросервиса промокодов

Листинг Е.1 – Метод AddPromocodeHandlers

```
func (s *Server) AddPromocodeHandlers(svc *promocode.Service) {
    s.router.HandleFunc("POST /promocodes/{code}/use",
        handler.UsePromocode(svc))
    s.router.HandleFunc("GET /promocodes",
        handler.GetMyPromocodes(svc))
}
```

Листинг Е.2 – Файл repo/promocode/repo.go

```
package promocode

import (
    "context"

    "github.com/akemoon/crowdfunding-app-promo/domain"
    "github.com/google/uuid"
)

type Repo interface {
    CreateWithEvent(ctx context.Context, eventID uuid.UUID, code
        domain.PromoCode) error
    UsePromocode(ctx context.Context, code string, ownerID
        uuid.UUID, expectedType domain.PromoType) (int, error)
    ListByOwner(ctx context.Context, ownerID uuid.UUID)
        ([]domain.PromoCode, error)
}
```

Листинг Е.3 – Метод UsePromocode

```
func (s *Service) UsePromocode(ctx context.Context, userID
    uuid.UUID, code string, expectedType domain.PromoType) (int,
    error) {
    if err := domain.Code(code).Validate(); err != nil {
        return 0, domain.ErrInvalidCodeFormat
    }

    effectValue, err := s.repo.UsePromocode(ctx, code, userID,
        expectedType)
    if err != nil {
        return 0, fmt.Errorf("repo: %w", err)
    }
    return effectValue, nil
}
```

Приложение Ж
(обязательное)

Фрагменты исходного кода клиентского приложения

Листинг Ж.1 – Файл src/routes/+layout.svelte

```
<script lang="ts">
  import { page } from "$app/stores";
  import AppHeader from "$lib/components/AppHeader.svelte";
  import "../app.css";
</script>

<div class="app-shell">
  {#if $page.url.pathname !== "/signin" && $page.url.pathname !==
"/signup"}
    <AppHeader />
  {/if}
  <main class="app-main">
    <slot />
  </main>
</div>
```

Листинг Ж.2 – Файл src/routes/signin/+page.svelte

```
<script lang="ts">
  import { goto } from '$app/navigation';
  import AuthFrame from '$lib/components/AuthFrame.svelte';
  import { signin } from '$lib/api';
  import { validateEmail } from '$lib/validate';

  let email = '';
  let password = '';
  let loading = false;
  let fieldErrors: Record<string, string> = {};
  let generalError = '';

  async function handleSubmit() {
    loading = true;
    fieldErrors = {};
    generalError = '';

    const emailErr = validateEmail(email);
    if (emailErr) {
      fieldErrors.email = emailErr;
      loading = false;
      return;
    }

    const result = await signin({ email, password });

    // ApiError has a 'code' field; SigninTokens does not
    if ('code' in result) {
      console.error('[signin]', result.code, result.message);
      if (result.code === 'invalid_credentials') {
        generalError = 'Неверный email или пароль';
      } else if (result.code === 'user_blocked') {
        generalError = 'Ваш аккаунт заблокирован.';
      }
    }
  }
</script>
```

```

    } else {
      generalError = 'Произошла ошибка. Попробуйте позже.';
    }
    loading = false;
    return;
  }

  localStorage.setItem('accessToken', result.accessToken);
  localStorage.setItem('refreshToken', result.refreshToken);

  goto('/projects');
}
</script>

<svelte:head>
  <title>Войти</title>
</svelte:head>

<AuthFrame title="Войти">
  <form class="form-fields"
on:submit|preventDefault={handleSubmit}>
    <div class="field">
      <label for="email">Email</label>
      <input
        id="email"
        type="email"
        bind:value={email}
        class:invalid={!!fieldErrors.email}
        autocomplete="email"
        required
      />
      {#if fieldErrors.email}
        <span class="field-error">{fieldErrors.email}</span>
      {/if}
    </div>

    <div class="field">
      <label for="password">Пароль</label>
      <input
        id="password"
        type="password"
        bind:value={password}
        class:invalid={!!fieldErrors.password}
        autocomplete="current-password"
        required
      />
    </div>

    {#if generalError}
      <p class="form-error">{generalError}</p>
    {/if}

    <button class="btn-submit" type="submit" disabled={loading}>

```

```

        {loading ? 'Загрузка...' : 'Войти'}
      </button>
    </form>

    <p class="switch-text" slot="footer">
      Нет аккаунта?
      <a href="/signup">Зарегистрироваться</a>
    </p>
  </AuthFrame>

  <style>
    .switch-text {
      margin: 0;
      text-align: center;
      color: var(--text-muted);
      font-size: 14px;
    }

    .switch-text a {
      margin-left: 4px;
      color: var(--accent);
      text-decoration: none;
      font-weight: 600;
    }

    .switch-text a:hover {
      text-decoration: underline;
    }
  </style>

```

Листинг Ж.3 – Файл src/routes/(protected)/+layout.svelte

```

<script lang="ts">
  import { onMount } from 'svelte';
  import { goto } from '$app/navigation';

  // Until the check runs, render nothing (prevents flash of
  protected content)
  let ready = false;

  onMount(() => {
    if (!localStorage.getItem('accessToken')) {
      goto('/signin');
    } else {
      ready = true;
    }
  });
</script>

{#if ready}
  <slot />
{/if}

```

Приложение И
(рекомендуемое)
Диаграммы классов микросервисов

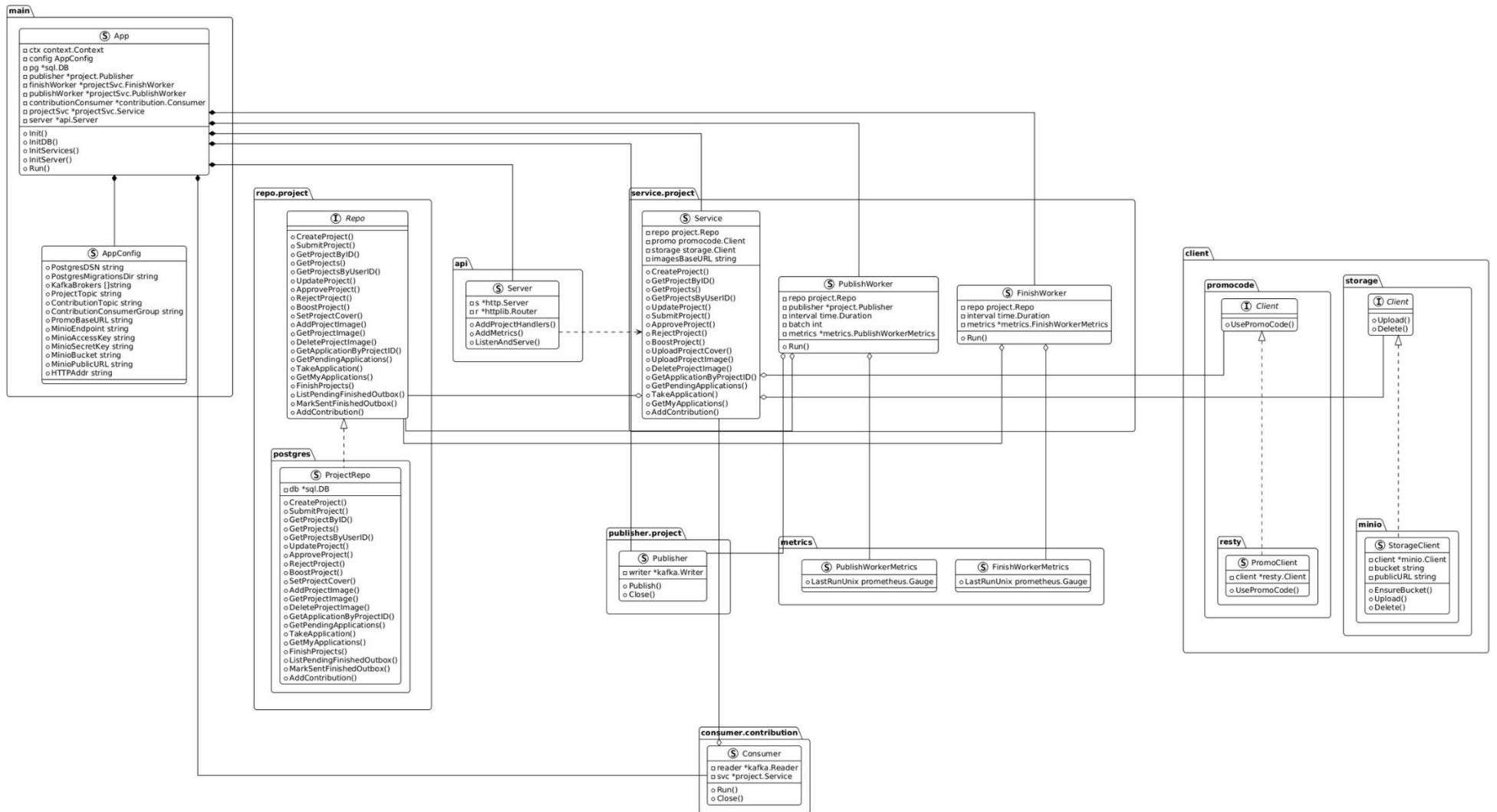


Рисунок И.2 – Диаграмма классов микросервиса проектов

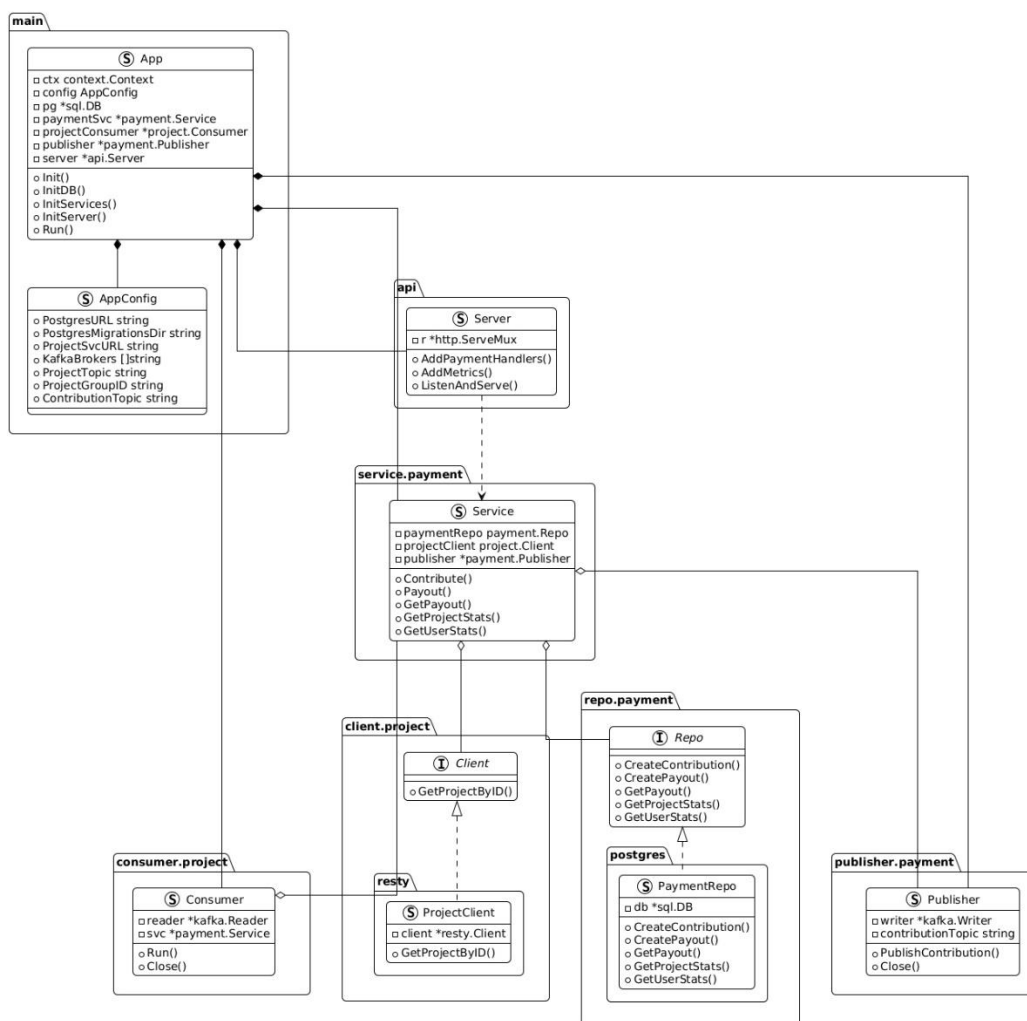


Рисунок И.3 – Диаграмма классов микросервиса платежей

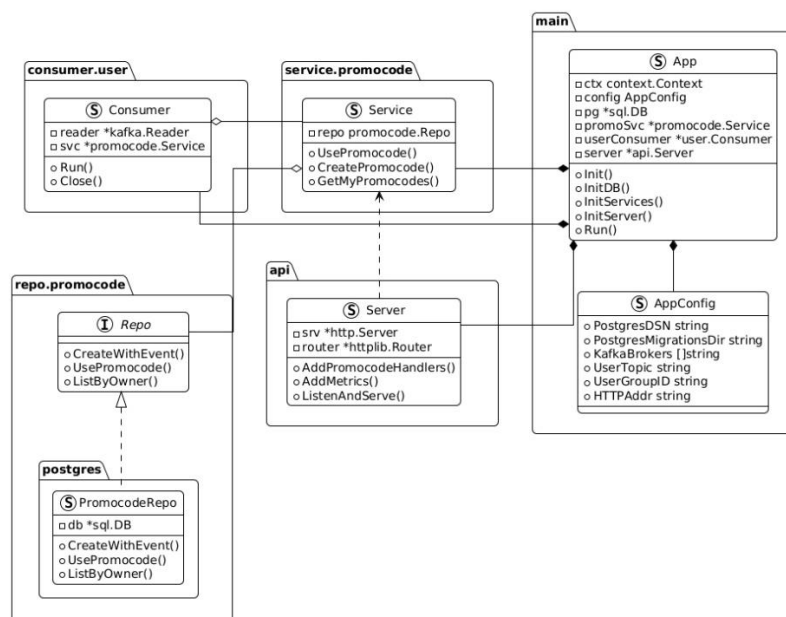


Рисунок И.4 – Диаграмма классов микросервиса промокодов

Приложение К
(рекомендуемое)
Тест-кейсы сценария создания проекта

Таблица К.1 – Тест-кейсы

№	Входные данные	Ожидаемый результат	Фактический результат	Статус теста
1	Название: «Мой проект» Описание: «Описание» Сумма сбора: 10000 Длительность: 30	Переход на страницу созданного проекта	Осуществлен переход на страницу созданного проекта (рисунок К.1)	Пройден
2	Название: «» Описание: «А» * 10001 Сумма сбора: 0 Длительность: 30	Указание на неверные поля: название, описание, сумма сбора	Помечены неверные поля: название, описание, сумма сбора (рисунок К.2)	Пройден
3	Название: «А» * 81 Описание: «А» * 10001 Сумма сбора: 100000001 Длительность: 0	Указание на неверные поля: название, описание, сумма сбора, длительность	Помечены неверные поля: название, описание, сумма сбора, длительность (рисунок К.3)	Пройден
4	Название: «А» * 81 Описание: «Описание» Сумма сбора: 0 Длительность: 61	Указание на неверные поля: название, сумма сбора, длительность	Помечены неверные поля: название, сумма сбора, длительность (рисунок К.4)	Пройден
5	Название: «» Описание: «Описание» Сумма сбора: 100000001 Длительность: 61	Указание на неверные поля: название, сумма сбора, длительность	Помечены неверные поля: название, сумма сбора, длительность (рисунок К.5)	Пройден
6	Название: «Мой проект» Описание: «А» * 10001 Сумма сбора: 10000 Длительность: 61	Указание на неверные поля: описание, длительность	Помечены неверные поля: описание, длительность (рисунок К.6)	Пройден
7	Название: «Мой проект» Описание: «Описание» Сумма сбора: 100000001 Длительность: 0	Указание на неверные поля: сумма сбора, длительность	Помечены неверные поля: сумма сбора, длительность (рисунок К.7)	Пройден

Продолжение таблицы К.1

8	Название: «» Описание: «Описание» Сумма сбора: 10000 Длительность: 0	Указание на неверные поля: название, длительность	Помечены неверные поля: название, длительность (рисунок К.8)	Пройден
9	Название: «А» * 81 Описание: «Описание» Сумма сбора: 10000 Длительность: 30	Указание на неверные поля: название	Помечены неверные поля: название (рисунок К.9)	Пройден
10	Название: «Мой проект» Описание: «Описание» Сумма сбора: 0 Длительность: 0	Указание на неверные поля: сумма сбора, длительность	Помечены неверные поля: сумма сбора, длительность (рисунок К.10)	Пройден
11	Название: «Мой проект» Описание: «Описание» Сумма сбора: 100000001 Длительность: 30	Указание на неверные поля: сумма сбора	Помечены неверные поля: сумма сбора (рисунок К.11)	Пройден

— Мои проекты

Новый проект

НАЗВАНИЕ

Мой проект

до 80 символов

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

ВАЛЮТА

RUB - рубль

СУММА СБОРА

10000

от 1 до 100 000 000

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

30

от 1 до 60

ОБЛОЖКА

опционально

Перетащите обложку сюда или нажмите для выбора

JPEG, PNG, GIF, WebP - до 5 MB

ФОТОГРАФИИ

опционально

Перетащите фото сюда или нажмите для выбора

JPEG, PNG, GIF, WebP - до 5 MB каждая

Создать проект

→

— Мои проекты

Мой проектЧерновик

Проект

Заявка

Статистика

Выплата

Категория

Наука

Цель

10 000 Р

Длительность

30 дн.

Описание

Редактировать

Загрузить обложку

Добавить фото

до 5 MB, JPEG / PNG / GIF / WebP

до 5 MB каждая, JPEG / PNG / GIF / WebP

Обложка обязательно для отправки на модерацию

Отправить на модерацию

Рисунок К.1 – Результат тест-кейса №1

← Мои проекты

Новый проект

НАЗВАНИЕ

до 80 символов

Неверное значение

ОПИСАНИЕ

до 10 000 символов

Неверное значение

КАТЕГОРИЯ

Наука

ВАЛЮТА

RUB - рубль

СУММА СБОРА

от 1 до 100 000 000

Неверное значение

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

от 1 до 60

Рисунок К.2 – Результат тест-кейса №2

← Мои проекты

Новый проект

НАЗВАНИЕ

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

до 80 символов

Неверное значение

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

ВАЛЮТА

RUB - рубль

СУММА СБОРА

0

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

61

от 1 до 100 000 000

от 1 до 60

Неверное значение

Неверное значение

Рисунок К.4 – Результат тест-кейса №4

← Мои проекты

Новый проект

НАЗВАНИЕ

до 80 символов

Неверное значение

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

ВАЛЮТА

RUB - рубль

СУММА СБОРА

100000001

от 1 до 100 000 000

Неверное значение

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

61

от 1 до 60

Неверное значение

Рисунок К.5 – Результат тест-кейса №5

[illegible]

Рисунок К.6 – Результат тест-кейса №6

← Мои проекты

Новый проект

НАЗВАНИЕ

до 80 символов

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

▼

ВАЛЮТА

RUB - рубль

▼

СУММА СБОРА

100000001

▼

от 1 до 100 000 000

Неверное значение

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

0

▼

от 1 до 60

Неверное значение

Рисунок К.7 – Результат тест-кейса №7

← Мои проекты

Новый проект

НАЗВАНИЕ

до 80 символов

Неверное значение

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

▼

ВАЛЮТА

RUB - рубль

▼

СУММА СБОРА

10000

▼

от 1 до 100 000 000

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

0

▼

от 1 до 60

Неверное значение

Рисунок К.8 – Результат тест-кейса №8

← Мои проекты

Новый проект

НАЗВАНИЕ

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

до 80 символов

Неверное значение

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

ВАЛЮТА

RUB - рубль

СУММА СБОРА

10000

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

30

от 1 до 100 000 000

от 1 до 60

Рисунок К.9 – Результат тест-кейса №9

← Мои проекты

Новый проект

НАЗВАНИЕ

Мой проект

до 80 символов

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

ВАЛЮТА

RUB - рубль

СУММА СБОРА

0

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

0

от 1 до 100 000 000

от 1 до 60

Неверное значение

Неверное значение

Рисунок К.10 – Результат тест-кейса №10

← Мои проекты

Новый проект

НАЗВАНИЕ

до 80 символов

ОПИСАНИЕ

Описание

до 10 000 символов

КАТЕГОРИЯ

Наука

ВАЛЮТА

RUB - рубль

СУММА СБОРА

100000001

от 1 до 100 000 000

Неверное значение

ДЛИТЕЛЬНОСТЬ (ДНЕЙ)

30

от 1 до 60

Рисунок К.11 – Результат тест-кейса №11