

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

|                                                                     |                                  |
|---------------------------------------------------------------------|----------------------------------|
| Факультет                                                           | Кафедра                          |
| вычислительной техники                                              | МОиПЭВМ                          |
| Сведения о направлении подготовки представлены в следующей таблице. |                                  |
| Направление подготовки                                              | 09.03.04 «Программная инженерия» |
| Направленность                                                      | Программная инженерия            |

**БАКАЛАВРСКАЯ РАБОТА**

на тему

**Система управления фриланс-проектами**

|                |                 |                           |
|----------------|-----------------|---------------------------|
| Студент        | _____           | Шмоленко Владимир Юрьевич |
|                | (подпись, дата) | (ФИО полностью)           |
| Руководитель   | _____           | Самуйлов С.В.             |
|                | (подпись, дата) | (фамилия, инициалы)       |
| Нормоконтролёр | _____           | Такташкин Д.В.            |
|                | (подпись, дата) | (фамилия, инициалы)       |

Работа допущена к защите (протокол заседания кафедры от \_\_\_\_\_ № \_\_\_\_\_)

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

Работа защищена с отметкой \_\_\_\_\_ (протокол заседания ГЭК от \_\_\_\_\_ № \_\_\_\_\_)

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

Пенза, 2026

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 Шмоленко Владимир Юрьевич \_\_\_\_ гр. \_22ВП1\_ факультета ВТ  
<Фамилия, имя, отчество полностью>

2. Тема работы Система управления фриланс-проектами \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025г.

2. Руководитель работы к.т.н, доцент, доцент кафедры «МО и ПЭВМ» Самуйлов С.В.

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

1) *Назначение: разработка системы управления фриланс-проектами*

2) *Среда выполнения: Windows 11.*

*Языковые средства разработки: C#, HTML, CSS, JavaScript, SQL, UML.*

*Инструментальные средства: Visual Studio 2022, pgAdmin 4, Visual Paradigm 17*

*Технологии разработки: ООП, микросервисная архитектура.*

3) *Основные функции приложения:*

*а) регистрация и авторизация пользователей*

*б) управление профилем;*

*в) создание, публикация, завершение и архивирование проектов;*

*г) добавление участников, вех и вложений;*

*д) создание и изменение задач;*

*е) назначение исполнителей;*

*ж) учет времени;*

*з) добавление комментариев;*

*и) фильтрация и просмотр данных;*

4. 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

| №<br>п/п | Наименование этапов работы                       | Объем<br>работы | Срок<br>выполнения         | Подпись<br>руководителя |
|----------|--------------------------------------------------|-----------------|----------------------------|-------------------------|
| 1        | Анализ предметной области и требований к системе | 10%             | 07.04.2026 –<br>13.04.2026 |                         |
| 2        | Проектирование программных средств               | 20%             | 14.04.2026 –<br>25.04.2026 |                         |
| 3        | Реализация программных средств                   | 30%             | 27.04.2026 –<br>11.05.2026 |                         |
| 4        | Тестирование программных средств                 | 20%             | 12.05.2026 –<br>21.05.2026 |                         |
| 5        | Оформление пояснительной записки                 | 20%             | 22.05.2026 –<br>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 123 страниц, 52 рисунка, 43 таблицы, 23 использованных источника и 7 приложений.

ФРИЛАНС-ПРОЕКТЫ, УПРАВЛЕНИЕ ПРОЕКТАМИ, ЗАДАЧИ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, ASP.NET CORE, POSTGRESQL, KAFKA, REST API, ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В ходе работы выполнен анализ предметной области и программ-аналогов, сформулированы функциональные и нефункциональные требования, разработана архитектура приложения, спроектированы базы данных микросервисов и реализованы серверная и клиентская части системы. Для обмена событиями между сервисами применяются брокер сообщений Kafka и шаблон outbox, для хранения данных используется PostgreSQL. Работоспособность системы подтверждена функциональным, интеграционным и модульным тестированием.

Объектом разработки является информационная система управления фриланс-проектами, предназначенная для организации работы с проектами, задачами, участниками, сроками и комментариями. Цель работы заключается в проектировании, реализации и тестировании веб-приложения, обеспечивающего централизованное управление проектной деятельностью и взаимодействие пользователей в рамках распределённой команды.

ПГУ 09.03.04 – 08ВР221.26 ПЗ

|           |      |                |         |      |                                                                   |  |  |              |      |        |     |  |
|-----------|------|----------------|---------|------|-------------------------------------------------------------------|--|--|--------------|------|--------|-----|--|
|           |      |                |         |      | ПГУ 09.03.04 – 08ВР221.26 ПЗ                                      |  |  |              |      |        |     |  |
|           |      |                |         |      |                                                                   |  |  |              |      |        |     |  |
| Изм.      | Лист | № докум.       | Подпись | Дата |                                                                   |  |  |              |      |        |     |  |
| Разраб.   |      | Шмоленко В.Ю.  |         |      | Система управления<br>фриланс-проектами.<br>Пояснительная записка |  |  | Лит.         | Лист | Листов |     |  |
| Провер.   |      | Самуйлов С.В.  |         |      |                                                                   |  |  |              |      | 4      | 123 |  |
| Реценз.   |      |                |         |      |                                                                   |  |  | Группа 22ВП1 |      |        |     |  |
| Н. Контр. |      | Такташкин Д.В. |         |      |                                                                   |  |  |              |      |        |     |  |
| Утверд.   |      | Козлов А.Ю.    |         |      |                                                                   |  |  |              |      |        |     |  |

## Содержание

|                                                                             |    |
|-----------------------------------------------------------------------------|----|
| Введение.....                                                               | 7  |
| 1 Постановка задачи и анализ требований .....                               | 9  |
| 1.1 Постановка задачи и анализ предметной области .....                     | 9  |
| 1.2 Анализ программ-аналогов .....                                          | 10 |
| 1.3 Анализ функциональных требований .....                                  | 16 |
| 1.4 Анализ нефункциональных требований .....                                | 24 |
| 1.4.1 Технические средства реализации требований.....                       | 25 |
| 2 Проектирование приложения для поддержки управления фриланс-проектами..... | 28 |
| 2.1 Архитектура приложения.....                                             | 28 |
| 2.2 Проектирование структур данных .....                                    | 31 |
| 2.2.1 Концептуальная модель базы данных.....                                | 31 |
| 2.2.2 Логическая модель базы данных .....                                   | 35 |
| 2.3 Проектирование программных средств .....                                | 43 |
| 2.5 Выбор технологий и инструментальных средств разработки приложения.....  | 51 |
| 2.5.1 Выбор технологий разработки приложения.....                           | 51 |
| 2.5.2 Выбор инструментальных средств разработки приложения.....             | 52 |
| 3 Разработка программного обеспечения .....                                 | 53 |
| 3.1 Разработка физической модели базы данных .....                          | 53 |
| 3.2 Реализация серверной части информационной системы.....                  | 57 |
| 3.3 Реализация клиентской части информационной системы.....                 | 62 |
| 3.4 Реализация интерфейса приложения .....                                  | 64 |
| 4 Отладка и тестирование программного обеспечения.....                      | 72 |

|                                                                                                 |     |
|-------------------------------------------------------------------------------------------------|-----|
|                                                                                                 | 6   |
| 4.1 Обоснование выбора режима тестирования.....                                                 | 72  |
| 4.2 Тестирование серверной части системы.....                                                   | 74  |
| 4.2.1 Функциональное тестирование серверной части .....                                         | 74  |
| 4.2.2 Интеграционное тестирование серверной части .....                                         | 77  |
| 4.2.3 Модульное тестирование серверной части .....                                              | 79  |
| 5 Контроль качества программного обеспечения.....                                               | 81  |
| 5.1 Обоснование выбора метрик кода.....                                                         | 81  |
| 5.2 Оценка качества программного кода .....                                                     | 81  |
| 5.2.1 Метрика Холстеда.....                                                                     | 81  |
| 5.2.2 Метрика Чепина .....                                                                      | 84  |
| 6 Планирование разработки и оценка бюджета.....                                                 | 86  |
| 6.1 Планирование разработки .....                                                               | 86  |
| 6.2 Расчет бюджета разработки .....                                                             | 88  |
| Заключение .....                                                                                | 92  |
| Список использованных источников .....                                                          | 93  |
| Приложение А (справочное) Глоссарий .....                                                       | 95  |
| Приложение Б (обязательное) Листинг серверной части системы.....                                | 97  |
| Приложение В (обязательное) Листинг клиентской части системы .....                              | 101 |
| Приложение Г (рекомендуемое) Листинг модульных проверок серверной части системы.....            | 104 |
| Приложение Д (рекомендуемое) Листинг интеграционных проверок межсервисного взаимодействия ..... | 107 |
| Приложение Е (рекомендуемое) Результаты функционального тестирования                            | 110 |
| Приложение Ж (рекомендуемое) Результаты модульного и интеграционного тестирования.....          | 120 |

## **Введение**

Современная организация проектной деятельности всё чаще опирается на распределённые команды, удалённое взаимодействие и гибкое распределение задач между участниками. Особенно заметно это в сфере фриланс-проектов, где заказчик и исполнитель могут находиться в разных городах, работать по индивидуальному графику и при этом должны иметь единое представление о состоянии проекта, сроках и объёме выполненных работ.

Актуальность темы выпускной квалификационной работы обусловлена необходимостью создания программного средства, позволяющего централизованно управлять проектами, задачами, участниками и сроками выполнения работ. Использование такой системы снижает риск потери информации, упрощает контроль текущего состояния проекта и повышает прозрачность взаимодействия между участниками.

Целью выпускной квалификационной работы является разработка системы управления фриланс-проектами, обеспечивающей создание и сопровождение проектов, постановку задач, назначение исполнителей, контроль сроков, ведение комментариев и фиксацию затраченного времени.

Для достижения поставленной цели необходимо решить следующие задачи: проанализировать предметную область и существующие программные аналоги; сформулировать функциональные и нефункциональные требования; разработать архитектуру приложения; спроектировать структуру данных; реализовать серверную и клиентскую части системы; выполнить тестирование и оценить качество полученного программного продукта.

Объектом исследования является процесс управления фриланс-проектами и задачами. Предметом исследования являются методы и программные средства автоматизации планирования, контроля выполнения задач и информационного взаимодействия участников проекта.

Практическая значимость работы заключается в том, что разработанное приложение может использоваться как основа для организации работы

небольшой команды, учебного проекта или индивидуального взаимодействия заказчика и исполнителя. Система предусматривает разделение данных по микросервисам, работу через веб-интерфейс и поддержку основных сценариев управления проектами.

Пояснительная записка включает анализ требований, проектирование системы, описание реализации, результаты тестирования, оценку качества программного кода, а также планирование разработки и расчёт бюджета проекта [1].

## **1 Постановка задачи и анализ требований**

### **1.1 Постановка задачи и анализ предметной области**

В рамках выпускной квалификационной работы необходимо разработать программную систему, предназначенную для управления проектами и задачами в условиях командной работы. Основной целью разработки является создание удобного и эффективного инструмента, позволяющего пользователям организовывать совместную деятельность, контролировать выполнение задач и отслеживать состояние проектов.

Предметная область включает процессы управления проектами, распределения задач между участниками, контроля сроков выполнения и взаимодействия пользователей в рамках команды. В современных условиях такие системы используются как в разработке программного обеспечения, так и в других сферах деятельности, где требуется координация работы нескольких исполнителей.

Разрабатываемая система должна обеспечивать возможность централизованного хранения информации о проектах, задачах и пользователях, а также предоставлять средства для работы с данной информацией. Важным аспектом является поддержка совместной работы, при которой несколько пользователей могут одновременно участвовать в проектах, выполнять задачи и отслеживать изменения.

Пользовательское взаимодействие с системой осуществляется через веб-интерфейс, обеспечивающий доступ к основным функциям управления проектами и задачами. Интерфейс должен быть интуитивно понятным и обеспечивать быстрый доступ к ключевым возможностям системы.

В системе предусматривается наличие различных ролей пользователей, что позволяет разграничивать доступ к функциональным возможностям. Отдельное внимание уделяется административным функциям, связанным с управлением пользователями и настройками системы.

Таким образом, разрабатываемая система ориентирована на повышение эффективности командной работы, упрощение управления проектами и обеспечение прозрачности выполнения задач.

## **1.2 Анализ программ-аналогов**

В процессе анализа предметной области были рассмотрены существующие программные решения, предназначенные для управления проектами, задачами и командной работой. Цель анализа аналогов – выявить сильные и слабые стороны существующих систем, а также определить преимущества разрабатываемого приложения.

В качестве аналогов были выбраны следующие системы:

- Redmine;
- Jira Software;
- Trello;
- Asana.

Выбор данных решений обусловлен их распространённостью, функциональной близостью к разрабатываемой системе и активным использованием в реальных проектах.

Redmine – это веб-ориентированная система управления проектами с открытым исходным кодом, широко используемая в корпоративной среде. Система предоставляет инструменты для ведения проектов, управления задачами, отслеживания ошибок и организации совместной работы команды.

На рисунке 1 представлен пример пользовательского интерфейса системы Redmine, демонстрирующий основные элементы навигации.

При рассмотрении данного аналога основное внимание уделялось не только перечню доступных функций, но и тому, насколько удобно пользователь может перейти от списка проектов к задачам и связанным рабочим материалам. Такой подход позволил определить требования к навигации разрабатываемой системы и избежать излишнего усложнения пользовательского интерфейса.

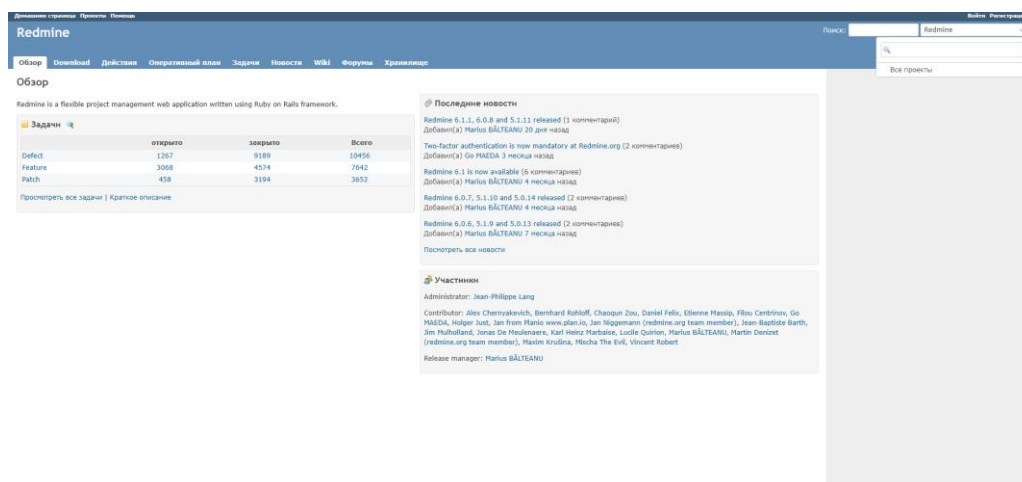


Рисунок 1 – Пользовательский интерфейс приложения «Redmine»

К основным преимуществам Redmine относятся:

- поддержка нескольких проектов и пользователей;
- гибкая система ролей и прав доступа;
- наличие встроенного трекера задач и ошибок;
- возможность расширения функционала за счёт плагинов;
- открытый исходный код и бесплатное распространение.

К недостаткам Redmine можно отнести:

- устаревший и слабо адаптивный пользовательский интерфейс;
- высокая сложность первоначальной настройки и администрирования;
- отсутствие встроенных механизмов масштабирования;
- монолитная архитектура, затрудняющая развитие системы.

Jira Software – коммерческая система управления проектами и задачами, ориентированная преимущественно на команды разработки программного обеспечения. Jira широко используется при реализации проектов по гибким методологиям управления.

На рисунке 2 показан интерфейс Jira Software, отображающий доску задач в формате Kanban.

В отличие от Redmine, данный аналог делает акцент на визуальном контроле потока задач и гибкой настройке рабочих процессов. При

проектировании разрабатываемой системы этот опыт был учтён частично: интерфейс должен оставаться достаточно простым для пользователя, но при этом давать возможность быстро определить состояние задачи и перейти к её изменению.



Рисунок 2 – Пользовательский интерфейс приложения «Jira Software»

Преимущества Jira Software:

- богатый функционал для управления задачами и проектами;
- развитые инструменты аналитики и отчётности;
- гибкая настройка рабочих процессов;
- интеграция с большим количеством сторонних сервисов.

Недостатки Jira Software:

- высокая стоимость лицензии и сопровождения;
- сложность интерфейса для новых пользователей;
- избыточный функционал для небольших команд;
- закрытый исходный код.

Trello – облачное приложение для управления задачами, основанное на канбан-подходе. Система ориентирована на визуальное представление рабочего процесса и простоту использования.

На рисунке 3 представлен пользовательский интерфейс Trello, демонстрирующий доску задач проекта.

Trello удобен для небольших команд благодаря простоте представления задач, однако при расширении предметной области становится заметным недостаток строгой структуры проектов, ролей и связанной аналитики. Поэтому в разрабатываемой системе визуальная простота должна сочетаться с более формализованной моделью данных.

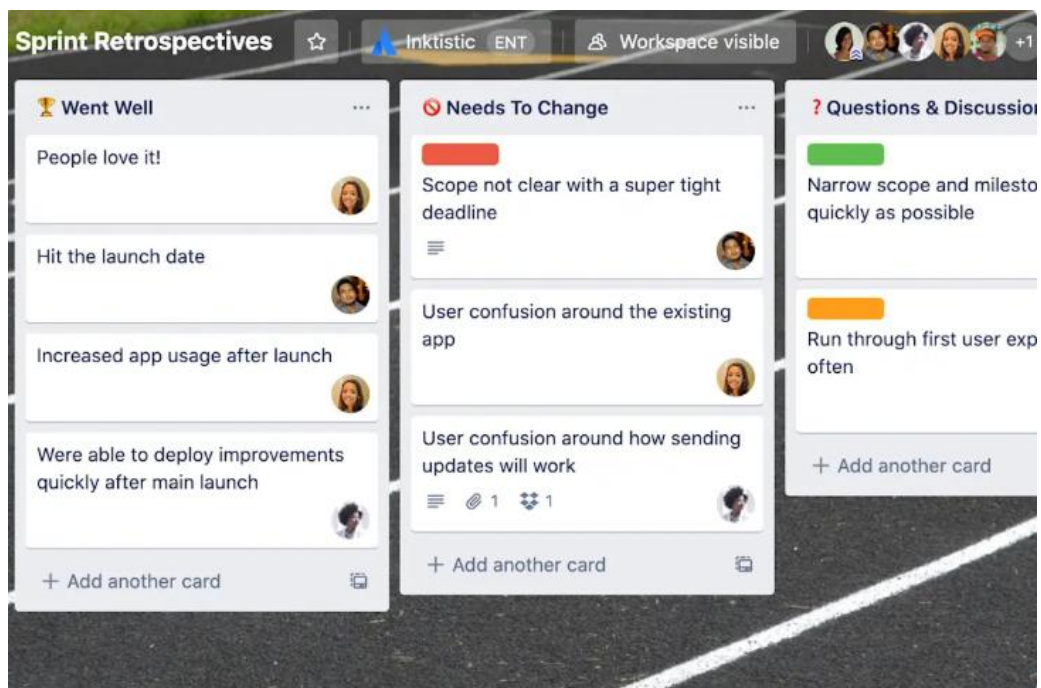


Рисунок 3 – Пользовательский интерфейс приложения «Trello»

#### Преимущества Trello:

- интуитивно понятный пользовательский интерфейс;
- минимальный порог входа;
- высокая скорость работы;
- хорошая адаптация под мобильные устройства.

#### Недостатки Trello:

- ограниченный функционал для крупных проектов;
- отсутствие развитой системы ролей и прав доступа;
- слабая поддержка сложных бизнес-процессов;
- зависимость от облачной инфраструктуры.

Asana – система управления проектами, предназначенная для организации командной работы и контроля выполнения задач. Приложение ориентировано на бизнес-пользователей и менеджеров проектов.

На рисунке 4 показан пример интерфейса Asana, иллюстрирующий представление задач в виде списка.

Asana показывает, что для пользователя важны не только сами задачи, но и контекст их выполнения: проект, срок, исполнитель, приоритет и текущее состояние. Эти элементы были включены в требования к системе, чтобы пользователь мог получать необходимую информацию без перехода между большим количеством отдельных экранов.

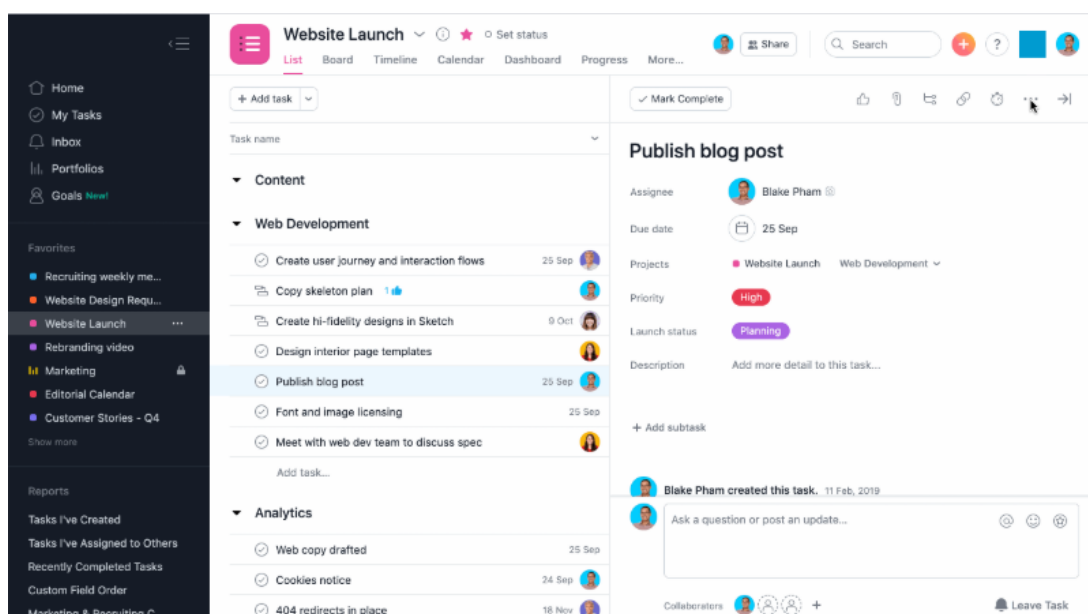


Рисунок 4 – Пользовательский интерфейс приложения «Asana»

Преимущества Asana:

- удобный и современный интерфейс;
- поддержка командной работы;
- различные представления задач;
- интеграция с популярными сервисами.

Недостатки Asana:

- ограниченный функционал в бесплатной версии;
- высокая стоимость платных тарифов;

- отсутствие возможности локального развёртывания;
- закрытый исходный код.

#### Сравнительный анализ

Для проведения сравнительного анализа аналогов были выбраны следующие критерии:

- удобство пользовательского интерфейса;
- адаптивность интерфейса;
- функциональная полнота;
- масштабируемость и производительность;
- стоимость разработки и сопровождения.

Оценка проводилась по следующей шкале:

- 0 баллов – критерий отсутствует;
- 1 балл – реализован частично;
- 2 балла – реализован полностью.

В таблице 1 представлены результаты сравнения аналогов.

Таблица 1 – Результаты сравнения аналогов

| Система                 | Удобство интерфейса | Адаптивность | Функциональность | Производительность | Низкая стоимость разработки и сопровождения | Итого |
|-------------------------|---------------------|--------------|------------------|--------------------|---------------------------------------------|-------|
| Разрабатываемая система | 2                   | 2            | 2                | 2                  | 2                                           | 10    |
| Redmine                 | 1                   | 0            | 2                | 1                  | 2                                           | 6     |
| Jira Software           | 1                   | 2            | 2                | 2                  | 0                                           | 7     |
| Trello                  | 2                   | 2            | 1                | 1                  | 1                                           | 7     |
| Asana                   | 2                   | 2            | 1                | 1                  | 0                                           | 6     |

В результате проведенного анализа было установлено, что существующие системы управления проектами либо обладают избыточным и сложным функционалом, либо не обеспечивают достаточную гибкость и масштабируемость.

Разрабатываемая система отличается:

- современной микросервисной архитектурой;
- возможностью горизонтального масштабирования;
- использованием асинхронного межсервисного взаимодействия;
- низкой стоимостью разработки и сопровождения;
- ориентацией на расширяемость и модульность.

Таким образом, разрабатываемое приложение является конкурентоспособным решением и устраняет ключевые недостатки рассмотренных аналогов.

### **1.3 Анализ функциональных требований**

В процессе анализа предметной области и задач на разработку информационной системы управления фриланс-проектами были выделены следующие основные функциональные требования:

- регистрация и авторизация пользователей;
- просмотр и редактирование профиля пользователя, восстановление пароля;
- создание, просмотр, редактирование и удаление проектов;
- добавление участников в проект;
- добавление ключевых этапов (вех) к проекту;
- добавление вложений и тегов к проекту;
- изменение статуса проекта (публикация, архивирование, завершение);
- создание, просмотр, редактирование и удаление задач;
- назначение исполнителей на задачу;
- добавление вложений и комментариев к задаче;

- изменение статуса задачи (старт, завершение, отмена);
- трекинг времени выполнения задачи;
- фильтрация задач по проекту, исполнителю, статусу и тегам.

В соответствии с указанными требованиями была построена диаграмма вариантов использования UML, отражающая взаимодействие пользователя с системой.

Диаграмма вариантов использования – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен пользователю. Диаграмма вариантов использования служит для описания функциональности и поведения проектируемой системы. На ней видно, кто взаимодействует с системой и какие действия может выполнить.

Актёром является тот, кто использует систему. В качестве актёра может выступать человек, внешняя сущность, класс или другая система. В разрабатываемой системе актёром является зарегистрированный пользователь.

Вариант использования – выполняемые системой действия, приводящие к наблюдаемым актёрами результатам. Каждый вариант использования описывает конкретный сценарий – последовательность шагов, которые система выполняет, чтобы получить нужный результат.

Пользователь может просматривать список проектов и задач, создавать проекты и задачи, редактировать их, изменять статус, добавлять участников, ключевые этапы, вложения, теги, комментарии, назначать исполнителей, вести трекинг времени, фильтровать задачи, а также редактировать данные своего профиля. Диаграмма вариантов использования для пользователя представлена на рисунке 5.

Состав вариантов использования сформирован таким образом, чтобы диаграмма отражала не отдельные элементы интерфейса, а устойчивые пользовательские сценарии. Это позволяет использовать её как основу для последующего проектирования структуры данных, серверных точек доступа и экранных форм клиентской части приложения.

При построении диаграммы варианты использования были сгруппированы по функциональным блокам: управление пользователем, управление проектами, управление задачами и фильтрация данных. Такое группирование позволяет показать границы системы и определить, какие действия должны быть доступны пользователю после авторизации.

Отдельно на диаграмме отражены операции изменения состояния проекта и задачи. Для разрабатываемой системы это важно, поскольку статус сущности определяет набор допустимых действий и влияет на дальнейшую обработку данных серверной частью.

Также в диаграмму включены сценарии добавления комментариев, вложений, участников и записей времени. Эти функции обеспечивают не только хранение основных данных, но и сопровождение рабочего процесса, что соответствует назначению системы управления фриланс-проектами.

Поскольку система ориентирована на работу с фриланс-проектами, в диаграмме не выделяются отдельные сценарии для сложных корпоративных процессов, таких как многоуровневое согласование или управление портфелями проектов. Основной акцент сделан на действиях, которые непосредственно требуются небольшой команде или отдельному исполнителю при выполнении заказа.

Полученная диаграмма используется как связующее звено между этапом анализа требований и дальнейшим проектированием. На её основе определяется состав сущностей предметной области, перечень экранов клиентской части и набор операций серверного API, которые должны быть реализованы в программной системе.

Такая детализация позволяет заранее определить границы разрабатываемой системы и исключить из проектирования функции, не относящиеся к управлению фриланс-проектами. В дальнейшем это упрощает переход от требований к описанию сущностей базы данных, поскольку каждый основной сценарий связан с конкретными объектами предметной области.

На основе диаграммы можно сопоставить функциональные требования с дальнейшими проектными решениями. Каждый выделенный сценарий связан с определенными данными, экранными формами и серверными операциями, поэтому диаграмма используется как исходная основа для проектирования структуры базы данных, интерфейса и программных модулей системы. Таким образом, диаграмма вариантов использования позволяет наглядно связать функциональные требования с действиями пользователя в системе.

Диаграмма вариантов использования системы представлена на рисунке 5.

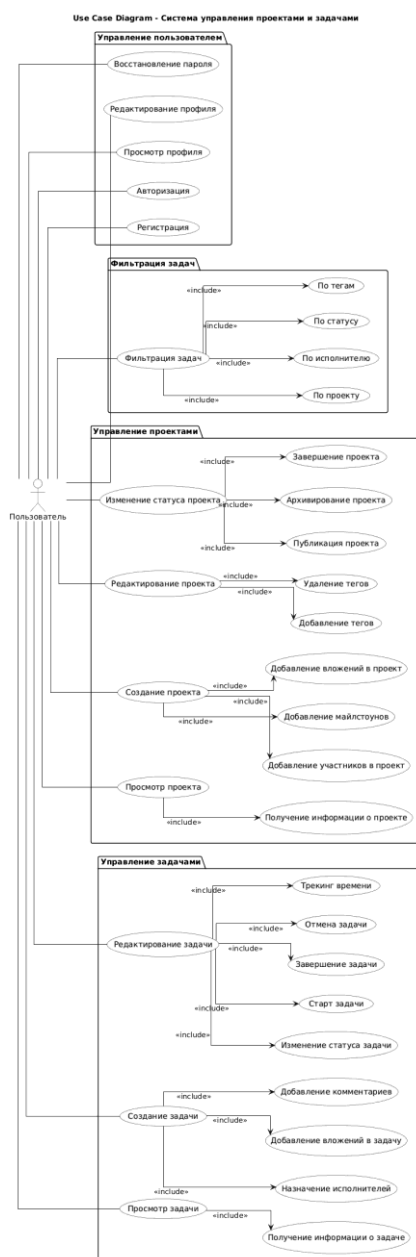


Рисунок 5 – Диаграмма вариантов использования системы управления проектами и задачами

Далее приведены спецификации наиболее важных вариантов использования, связанных с основным функционалом системы. Спецификации оформлены в виде таблиц и представлены в таблицах 2–8.

Спецификация варианта использования «Регистрация» приведена в таблице 2.

Таблица 2 – Спецификация варианта использования «Регистрация»

| Элемент              | Описание                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Актёр                | Новый пользователь (не зарегистрированный в системе)                                                                                                                                                                                                                                                                                                                                                                             |
| Предусловие          | Пользователь открыл страницу регистрации                                                                                                                                                                                                                                                                                                                                                                                         |
| Постусловие          | В системе создана новая учётная запись, пользователь авторизован                                                                                                                                                                                                                                                                                                                                                                 |
| Основной поток       | <ol style="list-style-type: none"> <li>1. Пользователь вводит логин, пароль, адрес электронной почты.</li> <li>2. Подтверждает пароль.</li> <li>3. Нажимает кнопку «Зарегистрироваться».</li> <li>4. Система проверяет уникальность логина и корректность введённых данных.</li> <li>5. Система создаёт учётную запись и автоматически выполняет вход.</li> <li>6. Пользователь перенаправляется на главную страницу.</li> </ol> |
| Альтернативный поток | Если логин уже существует или пароли не совпадают, система отображает сообщение об ошибке и предлагает исправить данные.                                                                                                                                                                                                                                                                                                         |

Спецификация варианта использования «Авторизация» приведена в таблице 3.

Таблица 3 – Спецификация варианта использования «Авторизация»

| Элемент     | Описание                                    |
|-------------|---------------------------------------------|
| Актёр       | Зарегистрированный пользователь             |
| Предусловие | Пользователь имеет учётную запись в системе |

Продолжение таблицы 3

| Элемент              | Описание                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Постусловие          | Пользователь получает доступ к системе в соответствии со своей ролью                                                                                                                                                                                                                                                                                                         |
| Основной поток       | <ol style="list-style-type: none"> <li>1. Пользователь открывает страницу входа.</li> <li>2. Вводит логин и пароль.</li> <li>3. Нажимает кнопку «Войти».</li> <li>4. Система проверяет соответствие введенных данных.</li> <li>5. При успешной проверке система создаёт сессию и перенаправляет пользователя на его рабочую страницу (список проектов или задач).</li> </ol> |
| Альтернативный поток | Если логин или пароль неверны, система отображает сообщение об ошибке и остаётся на странице входа.                                                                                                                                                                                                                                                                          |

Спецификация варианта использования «Создание проекта» приведена в таблице 4.

Таблица 4 – Спецификация варианта использования «Создание проекта»

| Элемент              | Описание                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Актёр                | Менеджер или администратор                                                                                                                                                                                                                                                                                                                                                                  |
| Предусловие          | Пользователь авторизован в системе                                                                                                                                                                                                                                                                                                                                                          |
| Постусловие          | Создан новый проект, пользователь назначен его владельцем                                                                                                                                                                                                                                                                                                                                   |
| Основной поток       | <ol style="list-style-type: none"> <li>1. Пользователь переходит на страницу «Проекты».</li> <li>2. Нажимает кнопку «Создать проект».</li> <li>3. Заполняет обязательные поля: название, описание, сроки.</li> <li>4. При необходимости добавляет вехи, вложения, теги.</li> <li>5. Нажимает «Сохранить».</li> <li>6. Система создаёт проект и отображает его в списке проектов.</li> </ol> |
| Альтернативный поток | Если не заполнено обязательное поле (название), система отображает сообщение об ошибке и не создаёт проект.                                                                                                                                                                                                                                                                                 |

Спецификация варианта использования «Создание задачи» приведена в таблице 5.

Таблица 5 – Спецификация варианта использования «Создание задачи»

| Элемент              | Описание                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Актёр                | Менеджер, администратор или исполнитель (в рамках проекта)                                                                                                                                                                                                                                                                                                                                 |
| Предусловие          | Выбран существующий проект, пользователь является участником проекта                                                                                                                                                                                                                                                                                                                       |
| Постусловие          | В проекте создана новая задача                                                                                                                                                                                                                                                                                                                                                             |
| Основной поток       | <ol style="list-style-type: none"> <li>1. Пользователь открывает страницу проекта.</li> <li>2. Нажимает кнопку «Добавить задачу».</li> <li>3. Заполняет поля: название, описание, исполнитель, срок, теги.</li> <li>4. При необходимости добавляет вложения.</li> <li>5. Нажимает «Создать».</li> <li>6. Система создаёт задачу со статусом «Открыта» и уведомляет исполнителя.</li> </ol> |
| Альтернативный поток | Если не указан исполнитель, задача сохраняется без назначения и отображается в списке нераспределённых задач.                                                                                                                                                                                                                                                                              |

Спецификация варианта использования «Назначение исполнителей» приведена в таблице 6.

Таблица 6 – Спецификация варианта использования «Назначение исполнителей»

| Элемент              | Описание                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Актёр                | Менеджер или администратор                                                                                                                                                                                                                                                                                                                                |
| Предусловие          | Задача создана, пользователь имеет право изменять задачу                                                                                                                                                                                                                                                                                                  |
| Постусловие          | Задаче назначен исполнитель                                                                                                                                                                                                                                                                                                                               |
| Основной поток       | <ol style="list-style-type: none"> <li>1. Пользователь открывает задачу.</li> <li>2. В поле «Исполнитель» начинает вводить имя или email.</li> <li>3. Система подсказывает участников проекта.</li> <li>4. Пользователь выбирает исполнителя и сохраняет изменения.</li> <li>5. Система обновляет задачу и отправляет уведомление исполнителю.</li> </ol> |
| Альтернативный поток | Если выбранный пользователь не является участником проекта, система отображает предупреждение и не назначает его.                                                                                                                                                                                                                                         |

Спецификация варианта использования «Изменение статуса задачи» приведена в таблице 7.

Таблица 7 – Спецификация варианта использования «Изменение статуса задачи»

| Элемент              | Описание                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Актёр                | Исполнитель, менеджер или администратор                                                                                                                                                                                                                                                                                                                                             |
| Предусловие          | Задача существует и не завершена                                                                                                                                                                                                                                                                                                                                                    |
| Постусловие          | Статус задачи изменён (например, «В работе» → «Готово»)                                                                                                                                                                                                                                                                                                                             |
| Основной поток       | <ol style="list-style-type: none"> <li>1. Пользователь открывает задачу.</li> <li>2. Выбирает новый статус из выпадающего списка (например, «В работе», «На проверке», «Готово»).</li> <li>3. При необходимости добавляет комментарий о выполненной работе.</li> <li>4. Подтверждает изменение.</li> <li>5. Система обновляет статус задачи и фиксирует время изменения.</li> </ol> |
| Альтернативный поток | Если пользователь пытается изменить статус задачи, на которую у него нет прав (например, исполнитель пытается завершить задачу, но требуется подтверждение менеджера), система отображает сообщение о недостаточных правах.                                                                                                                                                         |

Спецификация варианта использования «Трекинг времени» приведена в таблице 8.

Таблица 8 – Спецификация варианта использования «Трекинг времени»

| Элемент        | Описание                                                                                                                                                                                                                                                                                                                                                                         |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Актёр          | Исполнитель задачи                                                                                                                                                                                                                                                                                                                                                               |
| Предусловие    | Задача имеет статус «В работе» или «Открыта»                                                                                                                                                                                                                                                                                                                                     |
| Постусловие    | Зафиксировано время, затраченное на выполнение задачи                                                                                                                                                                                                                                                                                                                            |
| Основной поток | <ol style="list-style-type: none"> <li>1. Пользователь открывает задачу, назначенную ему.</li> <li>2. Нажимает кнопку «Начать учёт времени».</li> <li>3. Система запускает таймер.</li> <li>4. Когда работа завершена, пользователь нажимает «Остановить».</li> <li>5. Система добавляет затраченное время к общей статистике по задаче и сохраняет запись в истории.</li> </ol> |

В ходе анализа функциональных требований разрабатываемой системы были выделены ключевые сценарии взаимодействия пользователей с системой, включая регистрацию, авторизацию, управление проектами и задачами, трекинг времени. Для наиболее важных сценариев приведены спецификации, которые отображают действия пользователя и системы. Остальные варианты использования (просмотр и редактирование профиля, восстановление пароля, добавление комментариев и вложений, фильтрация задач по проекту/исполнителю/статусу/тегам) имеют типовую реализацию и не требуют детального описания на данном этапе.

#### **1.4 Анализ нефункциональных требований**

Нефункциональные требования определяют качество и характеристики работы системы, не связанные напрямую с конкретными функциями. Для информационной системы управления фриланс-проектами можно выделить следующие нефункциональные требования: производительность, безопасность, удобство использования, модульность и расширяемость, надёжность.

Производительность: система должна обеспечивать быструю реакцию на действия пользователя, включая загрузку страниц, авторизацию, отображение списков проектов и задач, создание и редактирование элементов. Время отклика интерфейса при выполнении основных операций не должно превышать 2 секунд. Приложение необходимо разработать с учётом возможного роста числа пользователей и проектов. Для этого предусмотрена возможность горизонтального масштабирования за счёт микросервисной архитектуры, а также использование кэширования для часто запрашиваемых данных.

Безопасность: система должна обеспечить надёжную аутентификацию пользователей. Пароли должны храниться в хэшированном виде. Доступ к проектам и задачам должен быть ограничен в соответствии с правами пользователя: каждый пользователь видит только те проекты, в которых он участвует, а администратор имеет полный доступ для контроля и аудита. Все

действия пользователей должны логироваться для обеспечения возможности анализа инцидентов.

Удобство использования: приложение должно обладать простым и интуитивно понятным интерфейсом, чтобы пользователь без специальной подготовки мог легко им пользоваться. Навигация по системе должна быть единообразной, основные функции (создание проекта, создание задачи, изменение статуса) должны быть доступны не более чем за три клика. Система должна предоставлять подсказки при вводе данных и валидацию заполняемых полей с выводом понятных сообщений об ошибках.

Модульность и расширяемость: архитектура системы должна позволять легко добавлять новые функции и модули без изменения уже реализованных компонентов. Использование микросервисной архитектуры и событийно-ориентированного подхода обеспечивает слабую связанность сервисов. Логика работы с базой данных должна быть инкапсулирована в репозиториях, чтобы изменения структуры БД не влияли на остальную часть системы. Также должна быть предусмотрена возможность интеграции с внешними сервисами через API.

Надёжность: система должна сохранять работоспособность при частичных сбоях. Временная недоступность одного из микросервисов (например, сервиса уведомлений) не должна приводить к остановке всей системы. Данные должны быть защищены от потери с использованием регулярного резервного копирования базы данных. Также должна быть реализована обработка ошибок на всех уровнях приложения с записью деталей в логи для последующего анализа.

#### **1.4.1 Технические средства реализации требований**

Разрабатываемая информационная система управления фриланс-проектами создаётся как веб-приложение с использованием современных технологий и архитектурных подходов.

Приложение будет построено с использованием микросервисной архитектуры. Каждый сервис отвечает за свою функциональную область

(пользователи, проекты, задачи) и взаимодействует с другими сервисами через REST API и брокер сообщений. В качестве API-шлюза используется YARP [2]. Событийное взаимодействие между сервисами организуется через Apache Kafka [3]. Такой подход соответствует принципам микросервисной архитектуры [4].

Для хранения данных будет использована реляционная система управления базами данных PostgreSQL, что обеспечит высокую производительность, надёжность, поддержку транзакций и целостность данных [5]. Доступ к данным будет организован с применением Entity Framework Core [6].

Серверная часть будет реализована на языке программирования C# с использованием фреймворка ASP.NET Core [7]. Разработка выполняется в среде Visual Studio 2022. Такой выбор позволит создать производительную, безопасную и масштабируемую серверную часть с поддержкой REST API. Описание HTTP-интерфейса выполняется с учётом спецификации OpenAPI [8].

Клиентская часть приложения будет создана с использованием JavaScript, что обеспечит обработку пользовательских действий на стороне браузера [9]. Структура экранов формируется средствами HTML [10]. Оформление интерфейса выполняется с применением CSS, что обеспечивает гибкость внешнего вида приложения [11].

Для создания диаграмм, описывающих структуру и поведение системы, используется язык UML 2.0. Диаграммы разрабатываются в среде Visual Paradigm или другом инструменте моделирования.

Веб-приложение должно быть оптимизировано для работы в популярных браузерах:

- Google Chrome (версия 90 и выше);
- Mozilla Firefox (версия 88 и выше);
- Microsoft Edge (версия 90 и выше);
- Safari (версия 14 и выше).

Преимущества выбранных инструментов и подходов:

- микросервисная архитектура позволит создать гибкое, масштабируемое и отказоустойчивое решение;
- фреймворк ASP.NET Core гарантирует высокую производительность и безопасность;
- PostgreSQL – надёжная СУБД с широкими возможностями настройки;
- Kafka – обеспечивает надёжную асинхронную доставку сообщений;
- клиент на HTML/CSS/JS – обеспечит совместимость с различными устройствами и браузерами.

## 2 Проектирование приложения для поддержки управления фриланс-проектами

### 2.1 Архитектура приложения

Архитектура разрабатываемой информационной системы управления проектами и задачами построена с использованием микросервисного подхода, что позволяет обеспечить гибкость, масштабируемость и отказоустойчивость приложения.

Система состоит из набора взаимосвязанных компонентов, каждый из которых выполняет строго определённые функции и взаимодействует с другими компонентами через стандартизированные интерфейсы. В качестве точки входа в систему используется API-шлюз, который обрабатывает входящие запросы от клиентского приложения и маршрутизирует их к соответствующим сервисам. Диаграмма компонентов системы представлена на рисунке 6.

Компонентная диаграмма позволяет показать приложение на уровне крупных программных частей, не углубляясь в реализацию отдельных классов. Такой уровень детализации удобен для обоснования выбранной микросервисной архитектуры, поскольку на схеме видно разделение клиентской части, API-шлюза, прикладных сервисов, баз данных и внешних инфраструктурных компонентов.

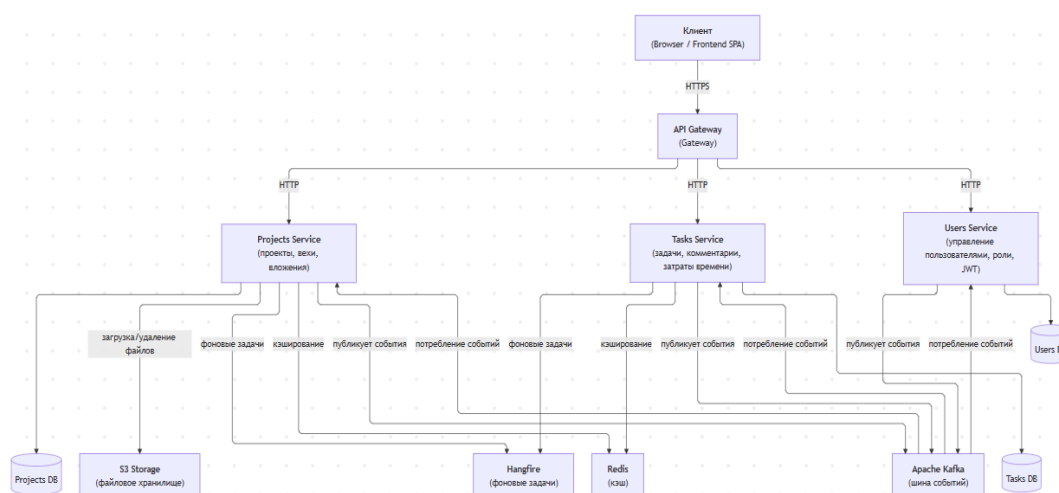


Рисунок 6 – Диаграмма компонентов

В состав системы входят следующие основные компоненты:

- клиентское приложение (браузер / одностраничное приложение);
- API-шлюз (Gateway);
- сервис пользователей (Users Service);
- сервис проектов (Projects Service);
- сервис задач (Tasks Service);
- базы данных (Users DB, Projects DB, Tasks DB);
- брокер сообщений Apache Kafka;
- кэш Redis;
- файловое хранилище S3;
- система фоновых задач Hangfire.

Клиентское приложение обеспечивает взаимодействие пользователя с системой и отправляет запросы к серверной части через API-шлюз. API-шлюз выполняет функции маршрутизации, агрегации запросов и централизованной обработки авторизации.

Сервис пользователей отвечает за регистрацию, аутентификацию и управление данными пользователей, а также за разграничение прав доступа. Сервис проектов реализует логику управления проектами и участниками. Сервис задач обеспечивает создание, изменение и контроль выполнения задач.

Взаимодействие между компонентами системы осуществляется с использованием REST API и событийной модели обмена сообщениями. Синхронное взаимодействие применяется для обработки пользовательских запросов, тогда как асинхронное используется для передачи событий между сервисами. Схема взаимодействия микросервисов представлена на рисунке 7.

Событийное взаимодействие реализуется через брокер сообщений, который обеспечивает доставку сообщений между сервисами. Это позволяет минимизировать связанность компонентов и повысить устойчивость системы к отказам.

Для повышения производительности используется кэш Redis, в котором хранятся часто запрашиваемые данные, например список проектов или задачи

пользователя [12]. Файловое хранилище S3 применяется для загрузки и хранения вложений (изображения, документы), связанных с проектами. Система фоновых задач Hangfire отвечает за выполнение фоновых операций, например архивирование просроченных проектов, без блокировки основного потока обработки запросов [13].

Диаграмма межсервисного взаимодействия уточняет порядок обмена данными между компонентами после выполнения пользовательских действий. На ней отражены как синхронные HTTP-запросы, так и асинхронная передача событий, что особенно важно для операций создания проектов, задач и обновления локальных проекций данных.

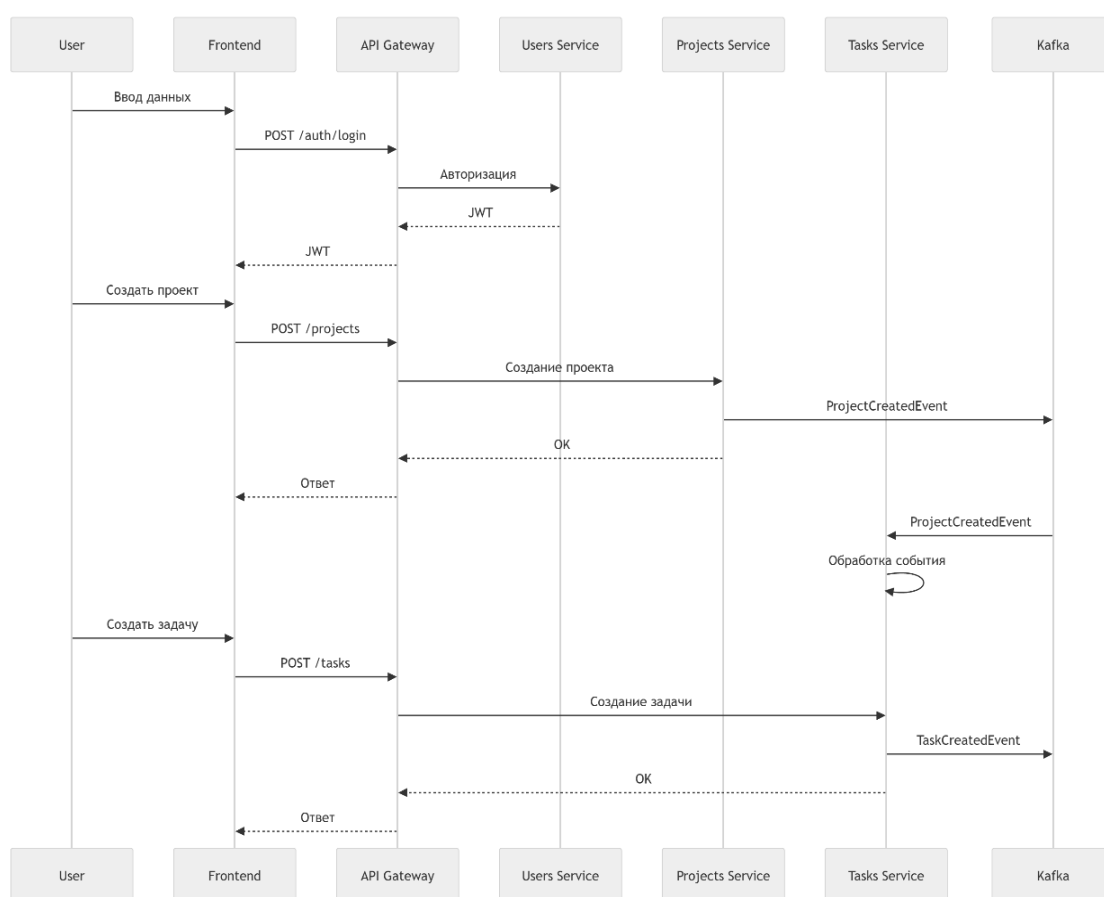


Рисунок 7 – Взаимодействие микросервисов

Каждый микросервис имеет внутреннюю многослойную структуру, включающую следующие уровни:

- уровень представления (API);
- уровень бизнес-логики;

- уровень доступа к данным;
- уровень инфраструктуры.

Такое разделение позволяет изолировать бизнес-логику от технических деталей реализации, повысить читаемость кода и упростить тестирование.

Микросервисная архитектура позволяет масштабировать отдельные компоненты системы независимо друг от друга. Это особенно важно для сервисов, испытывающих повышенную нагрузку.

Использование асинхронного взаимодействия и очередей сообщений позволяет системе продолжать работу даже при временной недоступности отдельных сервисов. Кроме того, изоляция компонентов предотвращает распространение ошибок по всей системе.

К основным особенностям архитектуры разрабатываемой системы относятся:

- разделение системы на независимые сервисы;
- использование API-шлюза как единой точки входа;
- применение событийно-ориентированного взаимодействия;
- разделение операций чтения и записи данных;
- использование кэширования для повышения производительности.

Таким образом, разработанная архитектура системы обеспечивает выполнение требований, сформулированных в разделе 1.1, и учитывает недостатки существующих решений, выявленные в разделе 1.2. Выбранный подход позволяет создать гибкую, расширяемую и устойчивую к нагрузкам систему управления проектами и задачами.

## **2.2 Проектирование структур данных**

### **2.2.1 Концептуальная модель базы данных**

Концептуальная модель описывает основные сущности предметной области и связи между ними, абстрагируясь от конкретной СУБД. Для разрабатываемой системы управления фриланс-проектами выделены следующие сущности:

- Пользователь – центральная сущность, хранящая учётные данные, личную информацию и роли;
- Проект – содержит информацию о проекте: название, описание, владелец, категория, бюджет, сроки, статус, теги;
- Задача – относится к проекту, имеет название, описание, исполнителя, постановщика, статус, приоритет, трудозатраты, срок;
- Участник проекта – связь пользователя с проектом, определяет роль участника;
- Веха проекта – ключевое событие в проекте с плановой датой и флагом выполнения;
- Вложение проекта – файлы, прикрепленные к проекту;
- Комментарий к задаче – текст, автор, дата, привязан к задаче;
- Запись времени – фиксация времени, затраченного пользователем на задачу, с почасовой ставкой;
- Роль – справочник ролей (например, «Администратор», «Менеджер», «Исполнитель»);
- Разрешение – права доступа, которые могут быть выданы роли;
- Связь пользователя с ролью (многие ко многим);
- Связь роли с разрешением (многие ко многим).

Концептуальная модель представлена на рисунках 8–10. Рисунок 8 отображает сущности и связи сервиса проектов, рисунок 9 – сервиса задач, рисунок 10 – сервиса пользователей.

Разделение концептуальной модели на несколько схем связано с микросервисной архитектурой приложения. Каждый сервис владеет собственными данными и не обращается напрямую к таблицам других сервисов, поэтому общие сведения, необходимые для выполнения запросов, передаются через события и сохраняются в виде локальных проекций.

Концептуальная модель базы данных сервиса проектов представлена на рисунке 8.



Рисунок 8 – Концептуальная модель БД сервиса проектов

Модель сервиса проектов отражает данные, которые относятся к жизненному циклу проекта: основные сведения о проекте, состав участников, плановые вехи и прикрепленные файлы. Эти сущности образуют самостоятельную область хранения, поскольку операции создания, публикации, архивирования и завершения проекта выполняются внутри сервиса проектов.

Связь проекта с участниками, вехами и вложениями показывает, что проект выступает центральной сущностью данной части системы. При этом сведения о пользователях не дублируются полностью, а используются только в объеме, необходимом для назначения ролей и отображения состава команды.

Концептуальная модель базы данных сервиса задач представлена на рисунке 9.



Рисунок 9 – Концептуальная модель БД сервиса задач

Модель сервиса задач выделена отдельно, так как задачи имеют собственный набор состояний, исполнителей, комментариев и записей времени. Такая структура позволяет сопровождать выполнение работ независимо от изменения общих параметров проекта и фиксировать историю действий по каждой задаче.

Комментарии и записи времени рассматриваются как зависимые сущности задачи. Они не используются самостоятельно, но позволяют хранить обсуждение работы, сведения о трудозатратах и признаки оплачиваемости, что важно для системы управления фриланс-проектами.

Концептуальная модель базы данных сервиса пользователей представлена на рисунке 10.



Рисунок 10 – Концептуальная модель БД сервиса пользователей

### 2.2.2 Логическая модель базы данных

Логическая модель детализирует сущности: указываются атрибуты, их типы, первичные и внешние ключи, связи. В разрабатываемой системе используется отдельная база данных для каждого микросервиса. Ниже приведены описания сущностей и логические модели для каждой базы данных.

База данных сервиса пользователей хранит учётные записи, роли и разрешения. Логическая модель представлена на рисунке 11, набор атрибутов сущностей – в таблицах 9–11. В таблице 9 представлены сущности базы данных сервиса пользователей.

Таблица 9 – Сущности базы данных сервиса пользователей

| Название сущности          | Идентификатор сущности | Назначение сущности                                                     |
|----------------------------|------------------------|-------------------------------------------------------------------------|
| Пользователь               | User                   | Содержит учётные данные, личную информацию, флаг администратора         |
| Роль                       | Role                   | Справочник ролей (например, «Администратор», «Менеджер», «Исполнитель») |
| Разрешение                 | Permission             | Справочник прав доступа (например, «project.create», «task.edit»)       |
| Связь пользователя с ролью | UserRole               | Определяет, какие роли назначены пользователю                           |
| Связь роли с разрешением   | RolePermission         | Определяет, какие права доступа предоставлены роли                      |

Атрибуты сущности «Пользователь» приведены в таблице 10.

Таблица 10 – Атрибуты сущности «Пользователь» (User)

| Атрибут           | Тип        | Описание                            |
|-------------------|------------|-------------------------------------|
| Id                | uuid       | Первичный ключ                      |
| Login             | text       | Уникальный логин                    |
| PasswordHash      | text       | Хэш пароля                          |
| Name              | text       | Полное имя                          |
| Gender            | integer    | Пол (0 – мужской, 1 – женский)      |
| Birthday          | timestampz | Дата рождения                       |
| Admin             | boolean    | Флаг администратора                 |
| Email             | text       | Адрес электронной почты             |
| CreatedAt         | timestampz | Дата создания записи                |
| CreatedBy         | text       | Кто создал                          |
| ModifiedOn        | timestampz | Дата последнего изменения           |
| ModifiedBy        | text       | Кто изменил                         |
| RevokedOn         | timestampz | Дата отзыва учётной записи          |
| RevokedBy         | text       | Кто отозвал                         |
| RegisteredObjects | integer    | Счётчик зарегистрированных объектов |

Атрибуты сущностей «Роль» и «Разрешение» приведены в таблице 11.

Таблица 11 – Атрибуты сущностей «Роль» (Role) и «Разрешение» (Permission)

| Сущность   | Атрибут | Тип  | Описание            |
|------------|---------|------|---------------------|
| Роль       | Id      | uuid | Первичный ключ      |
| Роль       | Name    | text | Название роли       |
| Разрешение | Id      | uuid | Первичный ключ      |
| Разрешение | Name    | text | Название разрешения |

Связи: пользователь может иметь много ролей (связь многие ко многим через UserRoles), роль может иметь много разрешений (связь многие ко многим через RolePermissions). Логическая модель базы данных сервиса пользователей приведена на рисунке 11.

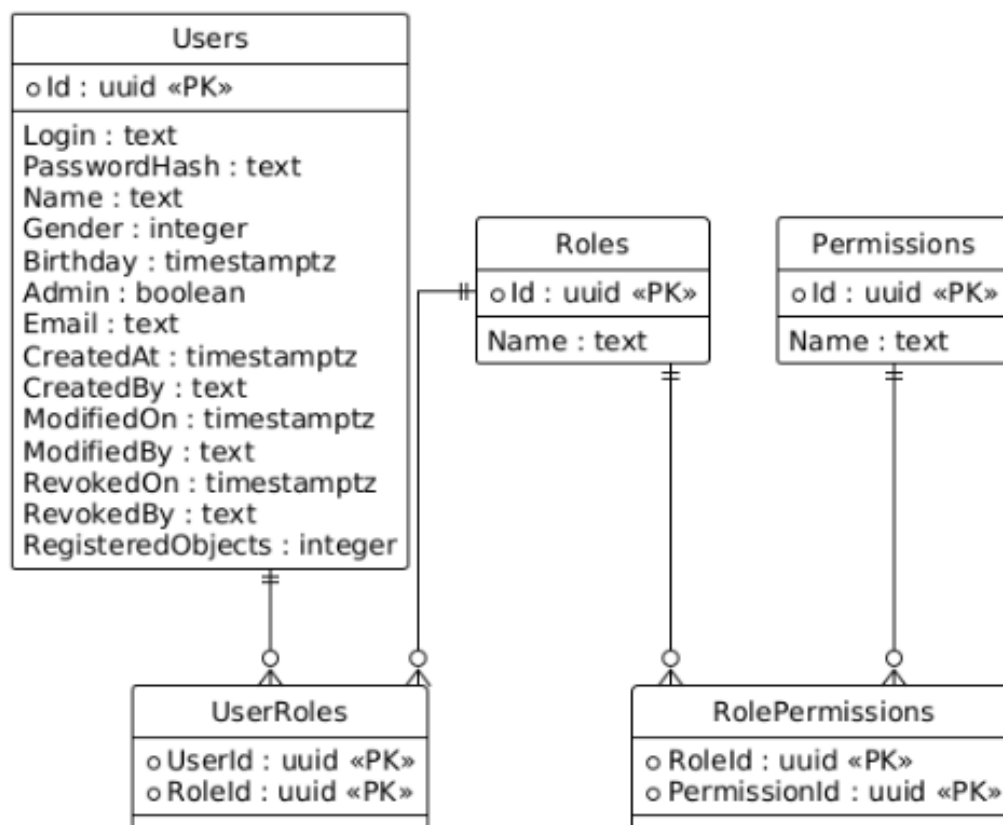


Рисунок 11 – Логическая модель базы данных сервиса пользователей

База данных сервиса проектов хранит информацию о проектах, участниках, вехах и вложениях. Логическая модель представлена на рисунке 12, набор атрибутов сущностей – в таблицах 12–15. В таблице 12 представлены сущности базы данных сервиса проектов.

Таблица 12 – Сущности базы данных сервиса проектов

| Название сущности | Идентификатор сущности | Назначение сущности                                                       |
|-------------------|------------------------|---------------------------------------------------------------------------|
| Проект            | Project                | Содержит информацию о проекте (название, описание, бюджет, сроки, статус) |
| Участник проекта  | ProjectMember          | Связывает пользователя с проектом, определяет роль                        |
| Веха              | ProjectMilestone       | Содержит ключевые события проекта (название, срок, флаг выполнения)       |
| Вложение          | ProjectAttachment      | Содержит ссылки на файлы, прикрепленные к проекту                         |
| Пользователь      | User                   | Дублирует информацию о пользователях для автономности сервиса             |

Атрибуты сущности «Проект» приведены в таблице 13.

Таблица 13 – Атрибуты сущности «Проект» (Project)

| Атрибут           | Тип           | Описание                          |
|-------------------|---------------|-----------------------------------|
| Id                | uuid          | Первичный ключ                    |
| Title             | varchar(200)  | Название проекта                  |
| Description       | varchar(2000) | Описание                          |
| OwnerId           | uuid          | Владелец (ссылка на пользователя) |
| Category          | varchar(100)  | Категория                         |
| CreatedAt         | timestampz    | Дата создания                     |
| ExpiresAt         | timestampz    | Срок истечения                    |
| Status            | integer       | Статус                            |
| BudgetMin         | numeric(18,2) | Минимальный бюджет                |
| BudgetMax         | numeric(18,2) | Максимальный бюджет               |
| CurrencyCode      | varchar(10)   | Валюта                            |
| Tags              | varchar(1000) | Теги (строкой через запятую)      |
| ConfirmationInfo  | text          | Информация о подтверждении        |
| ConfirmedAt       | timestampz    | Дата подтверждения                |
| ConfirmedByUserId | uuid          | Кто подтвердил                    |

Атрибуты сущности «Участник проекта» приведены в таблице 14.

Таблица 14 – Атрибуты сущности «Участник проекта» (ProjectMember)

| Атрибут   | Тип        | Описание               |
|-----------|------------|------------------------|
| Id        | uuid       | Первичный ключ         |
| UserId    | uuid       | Ссылка на пользователя |
| ProjectId | uuid       | Ссылка на проект       |
| Role      | text       | Роль участника         |
| AddedAt   | timestampz | Дата добавления        |

Атрибуты сущностей «Веха» и «Вложение» приведены в таблице 15.

Таблица 15 – Атрибуты сущностей «Веха» (ProjectMilestone) и «Вложение» (ProjectAttachment)

| Сущность | Атрибут     | Тип        | Описание                   |
|----------|-------------|------------|----------------------------|
| Веха     | Id          | uuid       | Первичный ключ             |
| Веха     | ProjectId   | uuid       | Ссылка на проект           |
| Веха     | Title       | text       | Название вехи              |
| Веха     | DueDate     | timestampz | Плановый срок              |
| Веха     | IsCompleted | boolean    | Флаг выполнения            |
| Веха     | IsEscalated | boolean    | Флаг эскалации             |
| Вложение | Id          | uuid       | Первичный ключ             |
| Вложение | ProjectId   | uuid       | Ссылка на проект           |
| Вложение | FileName    | text       | Имя файла                  |
| Вложение | Url         | text       | Ссылка на файл в хранилище |

Связи: проект может иметь много участников, вех и вложений (связи один ко многим). Логическая модель базы данных сервиса проектов приведена на рисунке 12. В этой схеме основная сущность Project объединяет параметры проекта, а связанные сущности хранят участников, вехи и материалы. Такое разделение упрощает поддержку данных: изменения в составе команды или прикрепленных файлах не требуют изменения базовой записи проекта. При переносе структуры в реляционную модель для каждой группы данных выделяется отдельная таблица, что делает связи явными и упрощает проверку целостности.

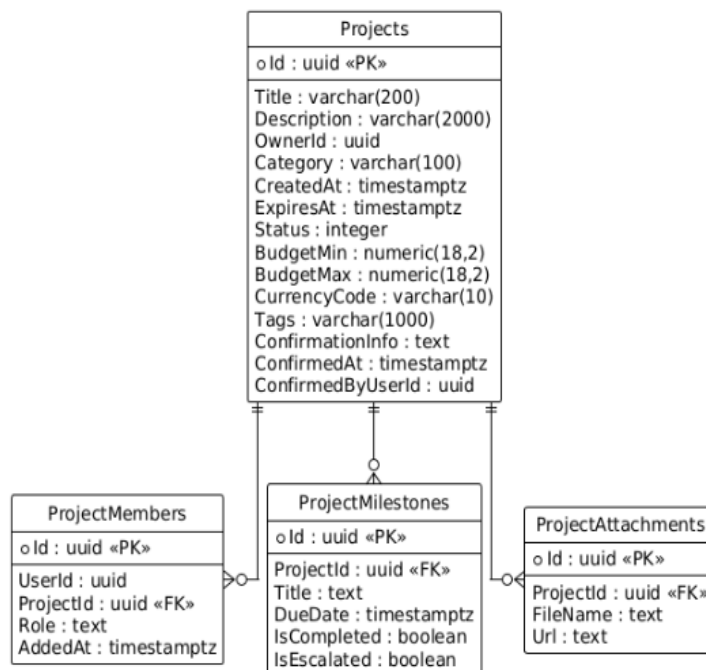


Рисунок 12 – Логическая модель базы данных сервиса проектов

База данных сервиса задач хранит информацию о задачах, комментариях и записях времени. Логическая модель приведена ниже, набор атрибутов сущностей – в таблицах 16–19. В таблице 16 представлены сущности базы данных сервиса задач.

В отличие от сервиса проектов, здесь основное внимание уделяется не описанию проекта в целом, а контролю выполнения конкретных работ. Поэтому в модель включены сведения о статусе, приоритете, сроке, исполнителе и фактически затраченном времени.

Таблица 16 – Сущности базы данных сервиса задач

| Название сущности | Идентификатор сущности | Назначение сущности                                                      |
|-------------------|------------------------|--------------------------------------------------------------------------|
| Задача            | Task                   | Содержит информацию о задаче (название, статус, приоритет, трудозатраты) |
| Комментарий       | Comment                | Содержит текст комментария к задаче, автора и дату                       |
| Запись времени    | TimeEntry              | Фиксирует время, затраченное пользователем на задачу                     |
| Проект            | Project                | Дублирует информацию о проекте для автономности сервиса                  |

Продолжение таблицы 16

| Название сущности | Идентификатор сущности | Назначение сущности                                           |
|-------------------|------------------------|---------------------------------------------------------------|
| Пользователь      | User                   | Дублирует информацию о пользователях для автономности сервиса |

Таблица 17 – Атрибуты сущности «Задача» (Task)

| Атрибут            | Тип           | Описание                                          |
|--------------------|---------------|---------------------------------------------------|
| Id                 | uuid          | Первичный ключ                                    |
| ProjectId          | uuid          | Ссылка на проект                                  |
| Title              | varchar(200)  | Название задачи                                   |
| Description        | varchar(2000) | Описание                                          |
| AssigneeId         | uuid          | Исполнитель (ссылка на пользователя)              |
| ReporterId         | uuid          | Постановщик                                       |
| Status             | integer       | Статус (0 – открыта, 1 – в работе, 2 – завершена) |
| Priority           | integer       | Приоритет (0 – низкий, 1 – средний, 2 – высокий)  |
| TimeEstimatedTicks | bigint        | Плановое время (в тиках)                          |
| TimeSpentTicks     | bigint        | Затраченное время (в тиках)                       |
| DueDate            | timestampz    | Срок выполнения                                   |
| IsBillable         | boolean       | Флаг оплачиваемости                               |
| CreatedAt          | timestampz    | Дата создания                                     |
| UpdatedAt          | timestampz    | Дата обновления                                   |

Атрибуты сущности «Комментарий» приведены в таблице 18.

Таблица 18 – Атрибуты сущности «Комментарий» (Comment)

| Атрибут   | Тип        | Описание                       |
|-----------|------------|--------------------------------|
| Id        | uuid       | Первичный ключ                 |
| TaskId    | uuid       | Ссылка на задачу               |
| AuthorId  | uuid       | Автор (ссылка на пользователя) |
| Text      | text       | Текст комментария              |
| CreatedAt | timestampz | Дата создания                  |

Атрибуты сущности «Запись времени» приведены в таблице 19.

Таблица 19 – Атрибуты сущности «Запись времени» (TimeEntry)

| Атрибут            | Тип        | Описание                        |
|--------------------|------------|---------------------------------|
| Id                 | uuid       | Первичный ключ                  |
| TaskId             | uuid       | Ссылка на задачу                |
| UserId             | uuid       | Пользователь, затративший время |
| StartedAt          | timestampz | Начало работы                   |
| EndedAt            | timestampz | Окончание работы                |
| Description        | text       | Описание выполненной работы     |
| IsBillable         | boolean    | Флаг оплачиваемости             |
| HourlyRateAmount   | numeric    | Почасовая ставка (сумма)        |
| HourlyRateCurrency | text       | Валюта ставки                   |
| Hours              | numeric    | Количество часов (вычисляемое)  |
| CreatedAt          | timestampz | Дата создания записи            |

Связи: задача относится к проекту, может иметь много комментариев и много записей времени (связи один ко многим). Логическая модель базы данных сервиса задач приведена на рисунке 13.

Такая структура позволяет отделить неизменяемые или редко изменяемые сведения о задаче от данных, которые появляются в процессе выполнения работы. Комментарии отражают обсуждение задачи, а записи времени фиксируют фактические трудозатраты исполнителя.

Использование отдельных таблиц для комментариев и временных записей также упрощает расширение системы. Например, при необходимости можно добавить модерацию комментариев, отчёты по трудозатратам или расчёт оплачиваемого времени без изменения основной таблицы задач.

Кроме того, такая модель удобна для формирования отчётности по задачам: основные сведения о задаче остаются в одной сущности, а история обсуждения и фактически затраченное время могут анализироваться отдельно. Дополнительно это позволяет быстро получать сводную информацию о ходе работ по проекту без избыточного дублирования данных. Связь с сущностью проекта сохраняет контекст выполняемой задачи, а отдельные записи позволяют

расширять функции учета и контроля. Логическая схема наглядно показывает, как эти части связаны между собой и как данные переходят от обсуждения к фиксации результата.

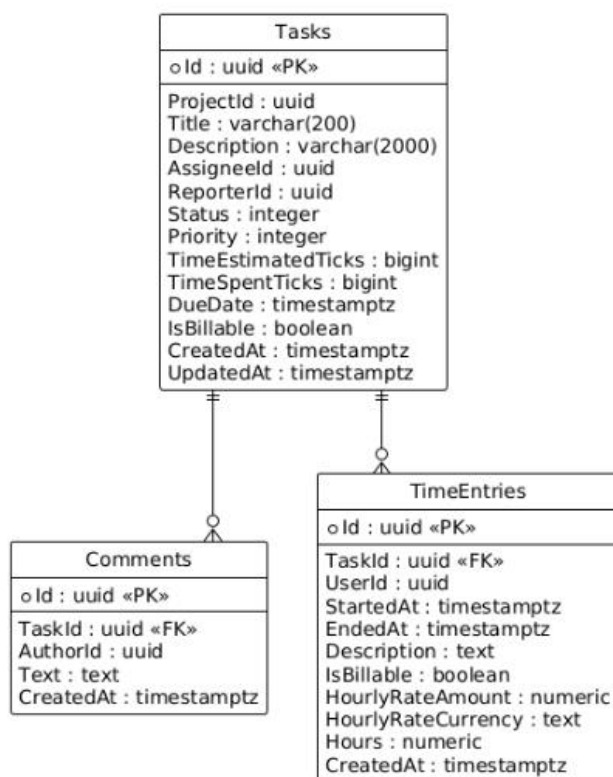


Рисунок 13 – Логическая модель базы данных сервиса задач

Таким образом, разработанные концептуальная и логическая модели полностью отражают структуру предметной области и обеспечивают основу для физической реализации трёх баз данных.

## 2.3 Проектирование программных средств

Программные средства разрабатываемой системы управления фриланс-проектами реализованы в соответствии с микросервисной архитектурой. Каждый микросервис (Users, Projects, Tasks) построен по принципам чистой архитектуры и CQRS, что обеспечивает разделение операций чтения и записи, упрощает масштабирование и поддержку системы [14].

Основные архитектурные решения, принятые при проектировании программных средств:

- каждый микросервис имеет собственную базу данных;
- взаимодействие между сервисами осуществляется асинхронно через брокер сообщений Apache Kafka (синхронные вызовы через REST API Gateway используются только для клиентских запросов);
- для обработки пользовательских запросов применён паттерн CQRS с разделением на команды (изменение состояния) и запросы (чтение данных);
- в качестве шины команд и запросов используется библиотека MediatR [15];
- доменные события публикуются через outbox и отправляются в Kafka;
- для повышения производительности часто выполняемые запросы кэшируются в Redis;
- файлы (вложения проектов) хранятся в объектном хранилище S3;
- фоновые задачи (архивация просроченных проектов) выполняются с использованием Hangfire.

Каждый микросервис структурно разделён на пять слоёв: API (контроллеры и модели запросов), Application (команды, запросы, обработчики, DTO, интерфейсы), Domain (сущности, объекты-значения, доменные события, интерфейсы репозиториев), Persistence (реализация репозиториев, конфигурации EF Core, read-модели) и Infrastructure (кэширование, файловое хранилище, Kafka, Hangfire, outbox, сервисы авторизации).

Ниже приведены описания и диаграммы классов для каждого из трёх микросервисов.

Сервис пользователей отвечает за регистрацию, аутентификацию, управление учётными записями, ролями и разрешениями. Он хранит основную информацию о пользователях, их логины, хэши паролей, а также определяет роли и права доступа.

Основные программные компоненты сервиса пользователей перечислены в таблице 20.

Таблица 20 – Основные классы и интерфейсы сервиса пользователей

| Компонент             | Тип                   | Назначение                                                                       |
|-----------------------|-----------------------|----------------------------------------------------------------------------------|
| AuthController        | Контроллер            | Обработка запросов регистрации, аутентификации, получения текущего пользователя  |
| UsersController       | Контроллер            | Управление пользователями (получение, обновление, удаление, восстановление)      |
| RolesController       | Контроллер            | Управление ролями и разрешениями                                                 |
| IUserRepository       | Интерфейс репозитория | Доступ к данным пользователей                                                    |
| IRoleRepository       | Интерфейс репозитория | Доступ к данным ролей                                                            |
| IPermissionRepository | Интерфейс репозитория | Доступ к данным разрешений                                                       |
| User                  | Доменная сущность     | Пользователь (логин, хэш пароля, имя, дата рождения, email, флаг администратора) |
| Role                  | Доменная сущность     | Роль (название)                                                                  |
| Permission            | Value object          | Разрешение (например, «project.create»)                                          |
| IJwtTokenGenerator    | Интерфейс сервиса     | Генерация JWT-токенов                                                            |
| IOutboxService        | Интерфейс сервиса     | Публикация событий через outbox                                                  |

Диаграмма классов сервиса пользователей представлена на рисунке 14.

Такое разделение компонентов позволяет отделить обработку HTTP-запросов от бизнес-логики и доступа к данным. Контроллеры принимают входные данные, команды и запросы описывают выполняемые операции, а сервисы и репозитории инкапсулируют правила работы с пользователями, ролями и разрешениями.

Благодаря этому изменения в способе хранения данных или в правилах авторизации не требуют переработки контроллеров. Компоненты сервиса остаются слабо связанными, что упрощает тестирование и дальнейшее сопровождение.

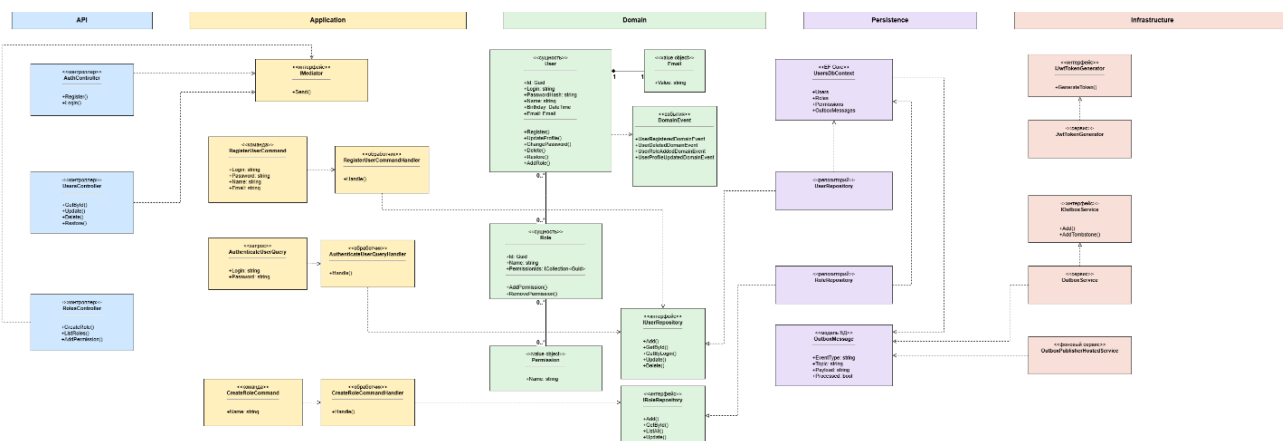


Рисунок 14 – Диаграмма классов сервиса пользователей

Бизнес-логика сервиса пользователей реализована с использованием CQRS и MediatR. Команды (регистрация, обновление профиля, изменение пароля, удаление, восстановление, создание роли, удаление роли, добавление/удаление разрешений) и запросы (аутентификация, получение пользователя по идентификатору, по логину, по email, список активных пользователей, список ролей) обрабатываются соответствующими хендлерами. Доменные события (UserRegisteredDomainEvent, UserDeletedDomainEvent, UserRoleAddedDomainEvent и др.) после сохранения изменений в базе данных публикуются через outbox и отправляются в Kafka для асинхронного уведомления других сервисов.

Сервис проектов управляет проектами, их участниками, вехами и вложениями. Он хранит информацию о проектах, бюджетах, сроках, статусах, тегах, а также обеспечивает разграничение доступа на основе ролей участников.

Основные программные компоненты сервиса проектов перечислены в таблице 21.

Таблица 21 – Основные классы и интерфейсы сервиса проектов

| Компонент          | Тип        | Назначение                                                                               |
|--------------------|------------|------------------------------------------------------------------------------------------|
| ProjectsController | Контроллер | Управление проектами (создание, обновление, удаление, публикация, архивация, завершение) |

## Продолжение таблицы 21

| Компонент             | Тип                   | Назначение                                                         |
|-----------------------|-----------------------|--------------------------------------------------------------------|
| IProjectRepository    | Интерфейс репозитория | Доступ к данным проектов, участников, вех, вложений                |
| IUserReadRepository   | Интерфейс репозитория | Чтение данных пользователей (из read-модели)                       |
| IAuthorizationService | Интерфейс сервиса     | Проверка прав доступа к проекту                                    |
| IFileStorage          | Интерфейс сервиса     | Загрузка и удаление файлов (S3)                                    |
| ICacheService         | Интерфейс сервиса     | Кэширование запросов (Redis)                                       |
| IBackgroundJobManager | Интерфейс сервиса     | Постановка фоновых задач (Hangfire)                                |
| IOutboxService        | Интерфейс сервиса     | Публикация доменных событий                                        |
| Project               | Доменная сущность     | Проект (название, описание, владелец, бюджет, сроки, статус, теги) |
| ProjectMember         | Доменная сущность     | Участник проекта (пользователь, роль, дата добавления)             |
| ProjectMilestone      | Доменная сущность     | Веха (название, плановый срок, флаги выполнения и эскалации)       |
| ProjectAttachment     | Доменная сущность     | Вложение (имя файла, ссылка)                                       |

Диаграмма классов сервиса проектов представлена на рисунке 15.

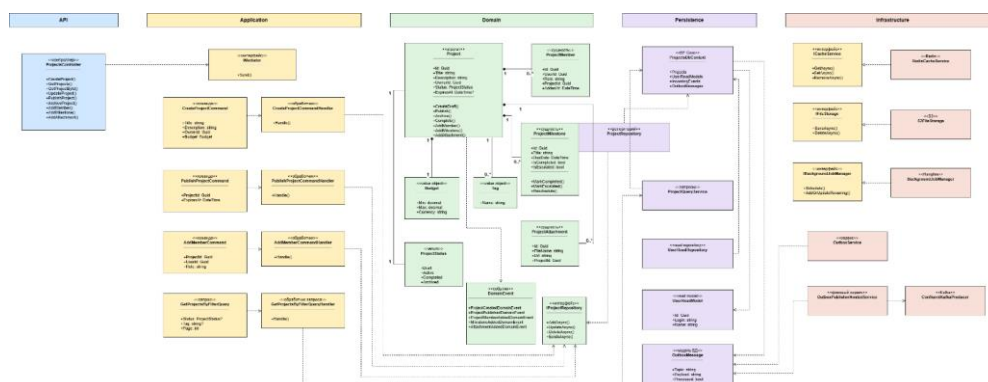


Рисунок 15 – Диаграмма классов сервиса проектов

Бизнес-логика сервиса проектов включает команды: создание, обновление, удаление, публикация, архивация, завершение проекта; добавление и удаление участников, изменение их ролей; добавление, удаление, завершение, эскалация, перенос вех; добавление и удаление вложений; добавление и удаление тегов. Запросы обеспечивают получение проектов с фильтрацией и пагинацией, детальную информацию о проекте (включая участников, вехи и вложения), список участников проекта.

Для повышения производительности часто выполняемые запросы кэшируются в Redis с использованием поведения CachingBehavior [12]. Фоновые задания запускаются по расписанию через Hangfire [13]. Доменные события (ProjectCreatedDomainEvent, ProjectPublishedDomainEvent, MemberAddedDomainEvent и др.) публикуются через outbox и отправляются в Kafka [3]. Из брокера сообщений их потребляют сервисы задач и уведомлений.

Сервис задач управляет задачами проектов, комментариями к задачам и учётом затраченного времени. Он хранит информацию о задачах (название, описание, статус, приоритет, плановое и фактическое время, срок), комментариях и временных записях. Для обеспечения автономности сервис задач поддерживает собственные read-модели пользователей и проектов, которые обновляются через входящие события из Kafka.

Основные программные компоненты сервиса задач перечислены в таблице 22.

Таблица 22 – Основные классы и интерфейсы сервиса задач

| Компонент              | Тип                   | Назначение                                                                              |
|------------------------|-----------------------|-----------------------------------------------------------------------------------------|
| ProjectTasksController | Контроллер            | Управление задачами (создание, обновление, назначение, изменение статуса, учёт времени) |
| IProjectTaskRepository | Интерфейс репозитория | Доступ к данным задач, комментариев, временных записей                                  |
| ICommentReadRepository | Интерфейс репозитория | Чтение комментариев                                                                     |

Продолжение таблицы 22

| Компонент                | Тип                   | Назначение                                                                            |
|--------------------------|-----------------------|---------------------------------------------------------------------------------------|
| IMemberReadRepository    | Интерфейс репозитория | Чтение участников проектов (read-модель)                                              |
| IProjectReadRepository   | Интерфейс репозитория | Чтение проектов (read-модель)                                                         |
| IIncomingEventRepository | Интерфейс репозитория | Хранение входящих событий из Kafka                                                    |
| IIncomingEventStore      | Интерфейс сервиса     | Обработка входящих событий                                                            |
| ICacheService            | Интерфейс сервиса     | Кэширование запросов (Redis)                                                          |
| IAuthorizationService    | Интерфейс сервиса     | Проверка прав доступа к задаче                                                        |
| ProjectTask              | Доменная сущность     | Задача (название, описание, исполнитель, постановщик, статус, приоритет, время, срок) |
| Comment                  | Доменная сущность     | Комментарий (автор, текст, дата)                                                      |
| TimeEntry                | Доменная сущность     | Запись времени (пользователь, начало, окончание, описание, почасовая ставка, часы)    |

Диаграмма классов сервиса задач представлена на рисунке 16.

Структура сервиса задач отражает несколько самостоятельных направлений бизнес-логики: управление жизненным циклом задачи, назначение исполнителей, добавление комментариев и учёт затраченного времени. Разделение этих операций по командам и обработчикам облегчает тестирование и снижает риск внесения ошибок при развитии функциональности. Команды изменения статуса, назначения исполнителей и добавления комментариев работают с одной доменной моделью, но отвечают за разные сценарии. События домена используются для уведомления других сервисов об изменениях, а

обработчики этих событий обеспечивают синхронизацию с проектами и пользователями.

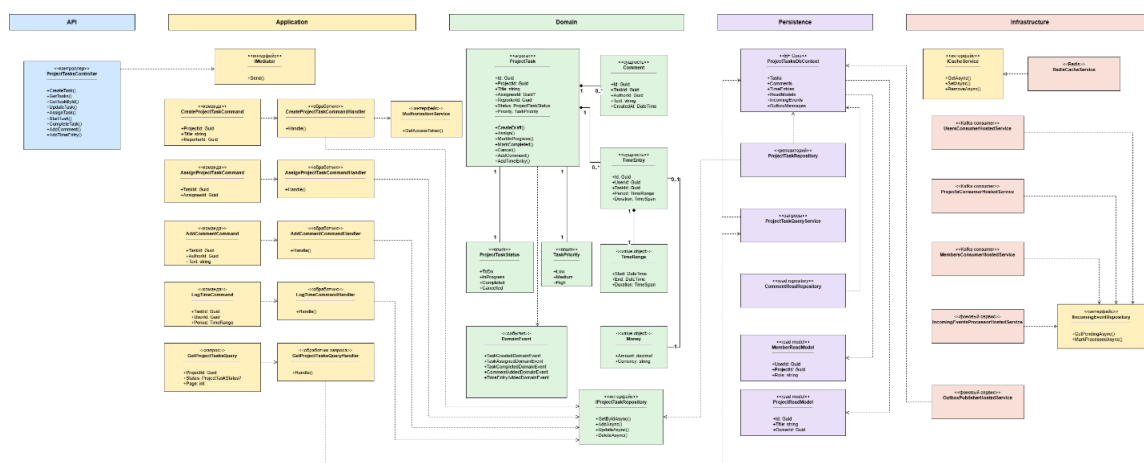


Рисунок 16 – Диаграмма классов сервиса задач

Бизнес-логика сервиса задач включает команды: создание, обновление, удаление задачи; назначение и снятие исполнителя; изменение статуса (старт, завершение, отмена); добавление комментария; добавление записи времени. Запросы обеспечивают получение задач с фильтрацией и пагинацией, детальную информацию о задаче, комментарии к задаче, список участников проекта (для назначения исполнителя).

Входящие события из Kafka (создание и обновление пользователей, создание и обновление проектов, добавление и удаление участников) обрабатываются фоновыми сервисами-потребителями (UsersConsumerHostedService, ProjectsConsumerHostedService, MembersConsumerHostedService) и сохраняются в read-модели. Это позволяет сервису задач работать автономно, не обращая синхронно к другим сервисам. Outbox-сообщения публикуются через OutboxPublisherHostedService.

Для кэширования запросов используется Redis, аналогично сервису проектов. Фоновые задания (например, обработка накопившихся входящих событий) могут быть реализованы через Hangfire при необходимости.

Таким образом, разработанные программные средства полностью соответствуют микросервисной архитектуре и требованиям к системе. Выбранные подходы (CQRS, событийное взаимодействие, отдельные базы

данных, кэширование, outbox) обеспечивают масштабируемость, расширяемость, отказоустойчивость и высокую производительность.

## **2.4 Проектирование пользовательского интерфейса**

Пользовательский интерфейс приложения проектируется с учетом основных сценариев работы участников фриланс-проектов: просмотра сводной информации, управления проектами, работы с задачами и изменения данных профиля. В качестве основных экранов выделены панель мониторинга, список проектов, карточка проекта, страница задач и страница профиля пользователя.

Навигация между разделами строится через боковое меню, что обеспечивает быстрый переход к ключевым функциям системы. Для действий, связанных с созданием и изменением данных, предусматриваются формы и модальные окна. Такой подход позволяет не перегружать основные страницы и сохранять последовательность пользовательского сценария.

При проектировании интерфейса учитывались требования к наглядности, минимальному количеству действий для выполнения типовых операций и корректному отображению состояния объектов. Пользователь должен видеть статус проекта или задачи, доступные действия и результат выполненной операции без обращения к дополнительным справочным разделам.

## **2.5 Выбор технологий и инструментальных средств разработки приложения**

Выбор технологий и инструментальных средств выполнялся с учетом архитектуры приложения, требований к масштабируемости, надежности хранения данных и удобству дальнейшего сопровождения программного продукта.

### **2.5.1 Выбор технологий разработки приложения**

Серверная часть приложения реализуется на языке C# с использованием платформы ASP.NET Core [7]. Данный стек выбран из-за поддержки разработки

REST API, встроенных механизмов внедрения зависимостей, развитой экосистемы библиотек и возможности построения масштабируемых веб-сервисов.

Для хранения данных используется PostgreSQL [5]. Доступ к данным организуется через Entity Framework Core [6]. Асинхронное взаимодействие между микросервисами обеспечивается брокером сообщений Kafka [3]. Клиентская часть реализуется с использованием JavaScript, что обеспечивает обработку пользовательских действий на стороне браузера [9]. Структура экранов формируется средствами HTML [10]. Оформление интерфейса выполняется с применением CSS, что обеспечивает работу приложения в современных браузерах [11].

Выбранный набор технологий согласуется с микросервисной архитектурой: серверные компоненты могут развиваться независимо, базы данных остаются изолированными, а клиентская часть взаимодействует с системой через единый набор HTTP-запросов.

### **2.5.2 Выбор инструментальных средств разработки приложения**

В качестве основной среды разработки используется Visual Studio 2022. Для проектирования и администрирования баз данных применяется pgAdmin 4. Диаграммы, описывающие структуру и поведение системы, разрабатываются с применением нотации UML 2.0. Контейнеризация инфраструктурных сервисов может выполняться с помощью Docker [16]. Для описания и совместного запуска контейнеров применяется Docker Compose [17].

Выбранные инструменты позволяют выполнять разработку, отладку, тестирование и сопровождение приложения в единой технологической среде. Это снижает трудоемкость разработки и обеспечивает согласованность проектных решений с программной реализацией.



Физическая схема сервиса пользователей включает не только основные таблицы предметной области, но и служебные таблицы, необходимые для работы приложения. Таблица истории миграций используется Entity Framework Core для контроля применённых изменений структуры базы данных, а таблицы входящих и исходящих событий обеспечивают устойчивую обработку межсервисного обмена.

Физическая модель базы данных сервиса проектов представлена на рисунке 18.

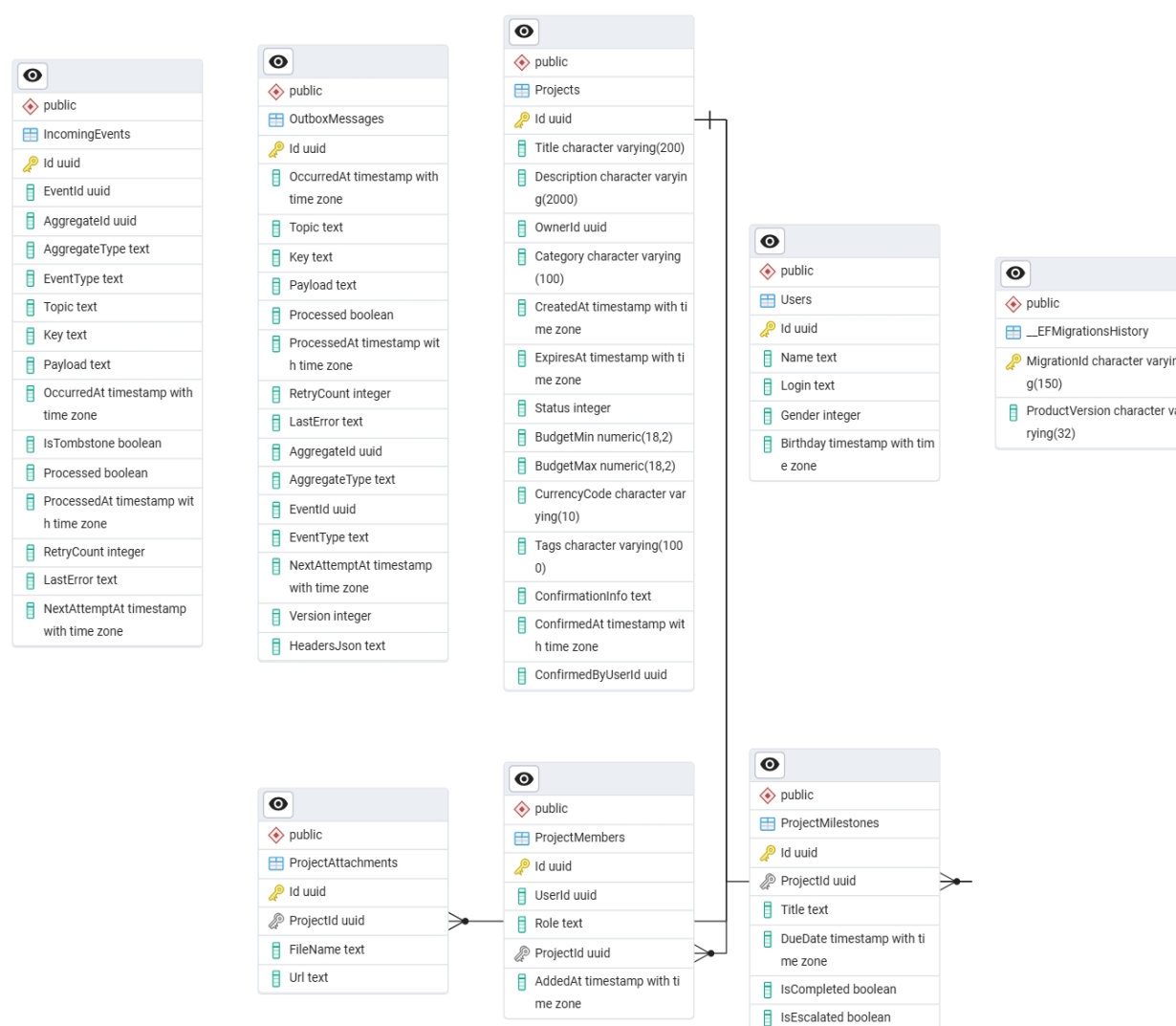


Рисунок 18 – Физическая модель базы данных сервиса проектов

В физической модели сервиса проектов дополнительно присутствуют таблицы outbox и incoming events. Они используются для гарантированной публикации событий о проектах и для идемпотентной обработки входящих

сообщений. За счёт этого повторная доставка события не приводит к дублированию данных и нарушению целостности проекта.

Физическая модель базы данных сервиса задач представлена на рисунке 19.

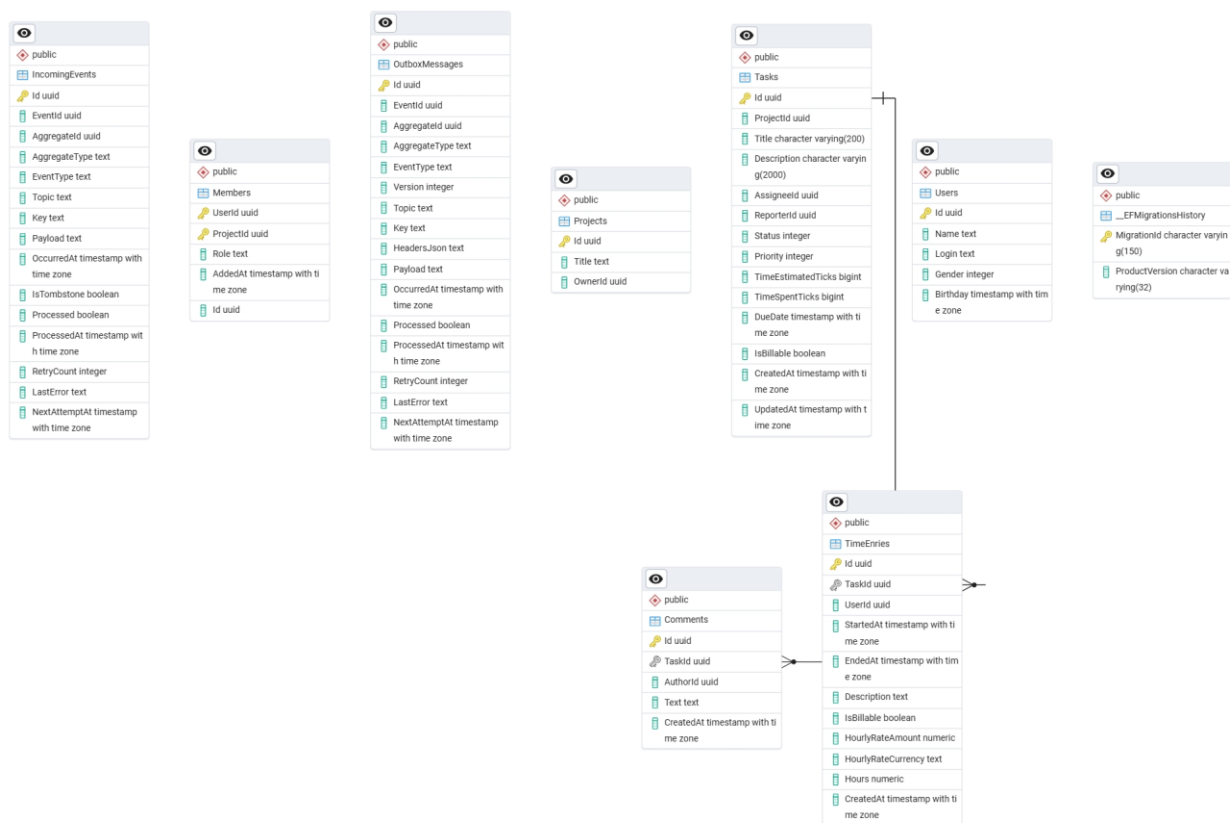


Рисунок 19 – Физическая модель базы данных сервиса задач

В базе данных сервиса задач используются локальные проекции проектов и пользователей. Они не заменяют исходные данные соответствующих сервисов, а хранят минимальный набор сведений, необходимый для быстрого отображения задач, исполнителей и принадлежности задачи к проекту без синхронного обращения к другим базам данных.

В физической реализации баз данных, помимо таблиц предметной области, присутствуют служебные таблицы, которые не выделялись отдельно на этапе логического проектирования. Таблица `__EFMigrationsHistory` создаётся Entity Framework Core и используется для хранения сведений о применённых миграциях. Её наличие необходимо для контролируемого обновления структуры базы данных при развитии приложения.

Таблицы OutboxMessages и IncomingEvents используются для поддержки межсервисного обмена событиями. OutboxMessages хранит события, которые должны быть опубликованы сервисом в брокер сообщений, а IncomingEvents фиксирует полученные события и состояние их обработки. Такой подход позволяет повысить надёжность обмена данными между сервисами и исключить повторную обработку уже принятых сообщений.

Также в отдельных базах данных присутствуют проекционные таблицы, содержащие сокращённые сведения из других сервисов. Например, в базе данных сервиса проектов хранится таблица Users, а в базе данных сервиса задач используются таблицы Users, Projects и Members. Эти таблицы не заменяют исходные данные других микросервисов, а служат локальными проекциями, которые ускоряют выполнение запросов и позволяют сервису работать без постоянного обращения к чужой базе данных. В таблице 23 представлены контексты баз данных и их назначение.

Таблица 23 – Контексты баз данных и их назначение

| Контекст базы данных  | Основные сущности                                                                                               | Назначение                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| UsersDbContext        | User, Role, Permission, UserRole, RolePermission, OutboxMessage                                                 | Хранение учётных записей пользователей, ролей, разрешений и сообщений outbox                                       |
| ProjectsDbContext     | Project, ProjectMember, ProjectMilestone, ProjectAttachment, UserReadModel, OutboxMessage                       | Хранение сведений о проектах, составе участников, этапах, вложениях и локальной read-модели пользователей          |
| ProjectTasksDbContext | ProjectTask, Comment, TimeEntry, IncomingEvent, ProjectReadModel, MemberReadModel, UserReadModel, OutboxMessage | Хранение задач, комментариев, записей учёта времени, входящих событий и read-моделей для автономной работы сервиса |

Развёртывание баз данных выполняется средствами миграций, что обеспечивает согласованное обновление схемы при развитии программного продукта. Таким образом, физическая реализация баз данных соответствует ранее построенным концептуальным и логическим моделям и создаёт основу для устойчивой работы всех сервисов системы.

### 3.2 Реализация серверной части информационной системы

Серверная часть приложения реализована на платформе ASP.NET Core в среде разработки Visual Studio 2022. Доступ к функциям системы организован через набор REST endpoint-ов, реализованных в контроллерах AuthController, UsersController, RolesController, ProjectsController и ProjectTasksController. Операции аутентификации и авторизации реализованы с учётом документации ASP.NET Core [18]. Для валидации входных моделей применяется FluentValidation [19]. Для преобразования DTO используется Mapster [20]. Исходный код системы размещён в репозитории проекта [1].

Взаимодействие клиентской части с сервером осуществляется через API Gateway, реализованный на базе YARP. Шлюз выполняет маршрутизацию HTTP-запросов и не содержит собственной бизнес-логики. Основная прикладная логика сосредоточена в сервисах пользователей, проектов и задач.

Точки доступа сервиса пользователей используют базовые префиксы api/Auth, api/Users и api/Roles. Часть методов доступна анонимно, а операции управления пользователями и ролями защищены средствами авторизации и проверки ролей [18]. Методы API сервиса пользователей представлены в таблицах 24–26.

Методы API аутентификации приведены в таблице 24.

Таблица 24 – Методы API аутентификации и получения данных пользователей

| URL запроса       | HTTP-метод | Описание                        |
|-------------------|------------|---------------------------------|
| api/Auth/register | POST       | Регистрация нового пользователя |
| api/Auth/login    | POST       | Аутентификация пользователя     |

Продолжение таблицы 24

|                            |      |                                                           |
|----------------------------|------|-----------------------------------------------------------|
| api/Users                  | GET  | Получение списка активных пользователей                   |
| api/Users/{id}             | GET  | Получение публичных данных пользователя по идентификатору |
| api/Users/by-login/{login} | GET  | Получение учётной записи по логину                        |
| api/Users/by-email/{email} | GET  | Получение учётной записи по адресу электронной почты      |
| api/Users/get-my-profile   | POST | Получение профиля текущего пользователя                   |

Методы API управления пользователями приведены в таблице 25.

Таблица 25 – Методы API управления пользователями

| URL запроса                    | HTTP-метод | Описание                                      |
|--------------------------------|------------|-----------------------------------------------|
| api/Users                      | POST       | Создание новой учётной записи администратором |
| api/Users/{userId}             | PUT        | Изменение данных пользователя                 |
| api/Users/{userId}/password    | PUT        | Изменение пароля пользователя                 |
| api/Users/{userId}/login       | PUT        | Изменение логина пользователя                 |
| api/Users/{userId}?hard={bool} | DELETE     | Удаление учётной записи пользователя          |
| api/Users/{userId}/restore     | PATCH      | Восстановление удалённой учётной записи       |

Методы API управления ролями приведены в таблице 26.

Таблица 26 – Методы API управления ролями и разрешениями

| URL запроса | HTTP-метод | Описание               |
|-------------|------------|------------------------|
| api/Roles   | GET        | Получение списка ролей |

Продолжение таблицы 26

| URL запроса                               | HTTP-метод | Описание                         |
|-------------------------------------------|------------|----------------------------------|
| api/Roles/{id}                            | GET        | Получение роли по идентификатору |
| api/Roles                                 | POST       | Создание новой роли              |
| api/Roles/{id}                            | DELETE     | Удаление роли                    |
| api/Roles/{id}/permissions/{permissionId} | POST       | Добавление разрешения роли       |
| api/Roles/{id}/permissions/{permissionId} | DELETE     | Удаление разрешения у роли       |

Сервис проектов предоставляет точки доступа для работы с проектами, участниками, ключевыми этапами, вложениями и тегами. Для базового контроллера используется префикс api/Projects, а все операции выполняются только для авторизованных пользователей. Загрузка вложений осуществляется в формате multipart/form-data. Методы API сервиса проектов представлены в таблицах 27–28.

Методы API основных операций с проектами приведены в таблице 27.

Таблица 27 – Методы API основных операций сервиса проектов

| URL запроса                      | HTTP-метод | Описание                                   |
|----------------------------------|------------|--------------------------------------------|
| api/Projects                     | POST       | Создание нового проекта                    |
| api/Projects                     | GET        | Получение списка проектов с фильтрацией    |
| api/Projects/{projectId}         | GET        | Получение данных проекта по идентификатору |
| api/Projects/{projectId}         | PUT        | Изменение данных проекта                   |
| api/Projects/{projectId}         | DELETE     | Удаление проекта                           |
| api/Projects/{projectId}/archive | PATCH      | Архивация проекта                          |
| api/Projects/{projectId}/publish | PATCH      | Публикация проекта                         |

## Продолжение таблицы 27

| URL запроса                       | HTTP-метод | Описание                            |
|-----------------------------------|------------|-------------------------------------|
| api/Projects/{projectId}/complete | PATCH      | Завершение проекта                  |
| api/Projects/{projectId}/members  | GET        | Получение списка участников проекта |

Методы API дополнительных операций с проектами приведены в таблице 28.

Таблица 28 – Методы API дополнительных операций сервиса проектов

| URL запроса                                   | HTTP-метод | Описание                           |
|-----------------------------------------------|------------|------------------------------------|
| api/Projects/{projectId}/add-attachment       | PATCH      | Добавление вложения к проекту      |
| api/Projects/{projectId}/delete-attachment    | PATCH      | Удаление вложения из проекта       |
| api/Projects/{projectId}/add-milestone        | PATCH      | Добавление ключевого этапа проекта |
| api/Projects/{projectId}/delete-milestone     | PATCH      | Удаление ключевого этапа проекта   |
| api/Projects/{projectId}/complete-milestone   | PATCH      | Завершение ключевого этапа проекта |
| api/Projects/{projectId}/reschedule-milestone | PATCH      | Перенос срока ключевого этапа      |
| api/Projects/{projectId}/add-member           | PATCH      | Добавление участника в проект      |
| api/Projects/{projectId}/remove-member        | PATCH      | Удаление участника из проекта      |
| api/Projects/{projectId}/add-tags             | PATCH      | Добавление тегов к проекту         |
| api/Projects/{projectId}/delete-tags          | PATCH      | Удаление тегов из проекта          |

Сервис задач использует базовый префикс api/ProjectTasks и обеспечивает выполнение операций, связанных с задачами проекта, комментариями и учётом времени. Все методы данного сервиса также доступны только авторизованным

пользователям. В таблице 29 представлены методы API основных операций сервиса задач.

Таблица 29 – Методы API основных операций сервиса задач

| URL запроса                        | HTTP-метод | Описание                                          |
|------------------------------------|------------|---------------------------------------------------|
| api/ProjectTasks                   | POST       | Создание новой задачи                             |
| api/ProjectTasks                   | GET        | Получение списка задач с фильтрацией и пагинацией |
| api/ProjectTasks/{taskId}          | GET        | Получение задачи по идентификатору                |
| api/ProjectTasks/{taskId}          | PUT        | Изменение задачи                                  |
| api/ProjectTasks/{taskId}          | DELETE     | Удаление задачи                                   |
| api/ProjectTasks/{taskId}/assign   | PATCH      | Назначение исполнителя на задачу                  |
| api/ProjectTasks/{taskId}/unassign | PATCH      | Снятие исполнителя с задачи                       |
| api/ProjectTasks/{taskId}/start    | PATCH      | Перевод задачи в состояние выполнения             |
| api/ProjectTasks/{taskId}/complete | PATCH      | Завершение задачи                                 |
| api/ProjectTasks/{taskId}/cancel   | PATCH      | Отмена задачи                                     |

Методы API комментариев и учета времени приведены в таблице 30.

Таблица 30 – Методы API комментариев, учёта времени и получения участников проекта

| URL запроса                            | HTTP-метод | Описание                        |
|----------------------------------------|------------|---------------------------------|
| api/ProjectTasks/{taskId}/time-entries | POST       | Добавление записи учёта времени |

Продолжение таблицы 30

| URL запроса                                 | HTTP-метод | Описание                                                |
|---------------------------------------------|------------|---------------------------------------------------------|
| api/ProjectTasks/{taskId}/comments          | POST       | Добавление комментария к задаче                         |
| api/ProjectTasks/{taskId}/comments          | GET        | Получение комментариев задачи                           |
| api/ProjectTasks/{projectId}/projectMembers | GET        | Получение участников проекта для назначения исполнителя |

Асинхронное межсервисное взаимодействие реализовано с использованием Kafka и шаблона outbox [3]. Публикация доменных событий выполняется через OutboxPublisherHostedService, а обработка входящих событий в сервисах задач и уведомлений обеспечивается фоновыми службами-потребителями. Для повышения производительности в сервисах проектов и задач применяется кэширование в Redis [12]. Для выполнения фоновых операций используется Hangfire [13]. Таким образом, серверная часть приложения обеспечивает полный цикл работы с данными и межсервисными событиями.

### 3.3 Реализация клиентской части информационной системы

Клиентская часть приложения представляет собой одностраничное веб-приложение, в котором пользовательская логика реализована с использованием JavaScript [9]. Структура экранов формируется средствами HTML [10]. Внешнее оформление интерфейса выполняется с применением CSS [11]. Загрузка разделов выполняется динамически, а обмен данными с серверной частью осуществляется посредством HTTP-запросов к REST API. Структура клиентской части построена по модульному принципу и включает системные файлы, функциональные JavaScript-модули и набор HTML-шаблонов. Для построения

диаграмм на панели мониторинга применяется Chart.js [21]. В таблице 31 представлены основные системные компоненты клиентской части.

Таблица 31 – Основные системные компоненты клиентской части

| Компонент  | Назначение                                            |
|------------|-------------------------------------------------------|
| index.html | Корневая HTML-страница приложения                     |
| styles.css | Общие стили интерфейса                                |
| api.js     | Отправка HTTP-запросов к серверной части              |
| app.js     | Инициализация и координация работы клиентских модулей |
| main.js    | Точка входа клиентского приложения                    |
| routing.js | Клиентская маршрутизация между разделами приложения   |

Функциональные модули клиентской части приведены в таблице 32.

Таблица 32 – Функциональные модули клиентской части

| Компонент         | Назначение                                                      |
|-------------------|-----------------------------------------------------------------|
| auth.js           | Реализация авторизации и регистрации пользователей              |
| dashboard.js      | Формирование содержимого панели мониторинга                     |
| dashboardStats.js | Отображение статистических показателей                          |
| charts.js         | Построение диаграмм и графиков                                  |
| projects.js       | Работа со списком проектов и карточками проектов                |
| projectForm.js    | Обработка формы создания и редактирования проекта               |
| tasks.js          | Работа со списком задач                                         |
| taskManagement.js | Изменение статусов задач, назначение исполнителей, учёт времени |
| profile.js        | Отображение и редактирование профиля пользователя               |
| modalManager.js   | Управление модальными окнами приложения                         |
| partials/*.html   | HTML-шаблоны отдельных разделов интерфейса                      |

Такое построение клиентской части позволяет изолировать логику отдельных разделов системы, упрощает сопровождение кода и делает возможным дальнейшее расширение функциональности без существенной переработки уже реализованных модулей.

### 3.4 Реализация интерфейса приложения

В соответствии с разработанным пользовательским интерфейсом были реализованы основные экраны системы: панель мониторинга, страница списка проектов, карточка проекта, отдельная страница задач, модальное окно деталей задачи и страница профиля пользователя. Интерфейс ориентирован на повседневную работу заказчиков, менеджеров и исполнителей в рамках фриланс-проектов. Основные формы интерфейса приложения представлены далее.

Панель мониторинга разработанного приложения представлена на рисунке 20.

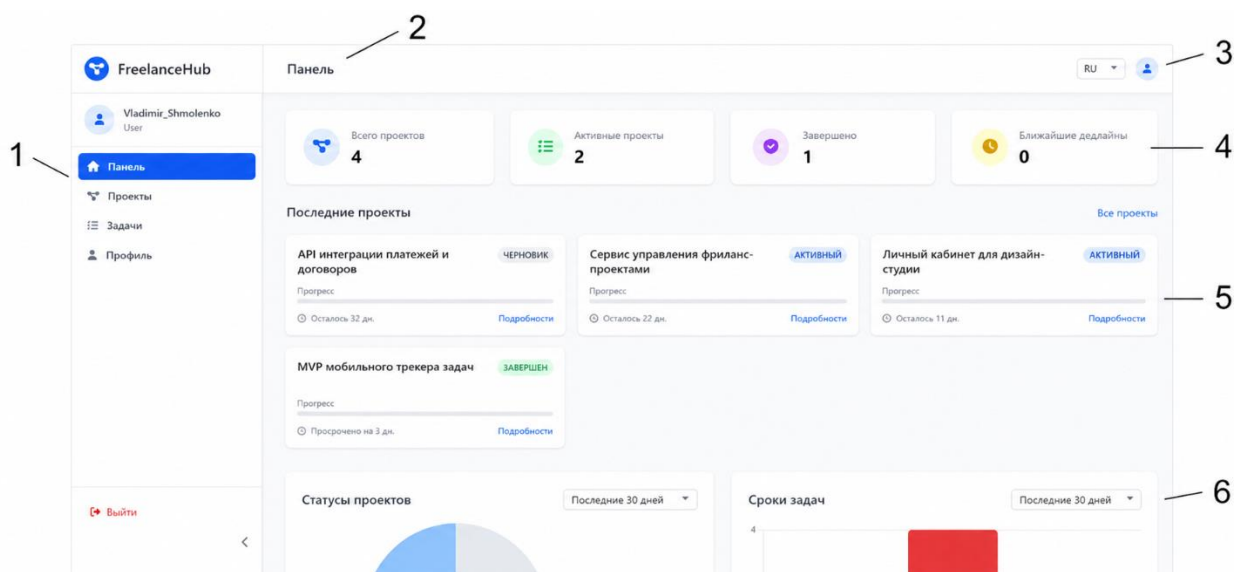


Рисунок 20 – Панель мониторинга пользователя

Рассмотрим основные элементы интерфейса панели мониторинга разработанного приложения, представленные на рисунке 20:

1 – боковое меню;

2 – верхняя панель и заголовок раздела;

- 3 – элементы выбора языка и профиля;
- 4 – сводные показатели;
- 5 – список последних проектов;
- 6 – область графиков.

Панель мониторинга выполняет роль стартового экрана после входа в систему. Она не предназначена для редактирования всех сущностей, а служит для быстрого понимания общей ситуации: сколько проектов активно, какие задачи требуют внимания и какие сроки приближаются. Такой экран сокращает количество действий пользователя при первичном контроле состояния работ.

Панель мониторинга предназначена для быстрого доступа к сводной информации по проектам, задачам и уведомлениям. На ней отображаются ключевые показатели текущей активности пользователя, перечень последних проектов, а также графические элементы, характеризующие состояние выполняемых работ.

Страница списка проектов представлена на рисунке 21.

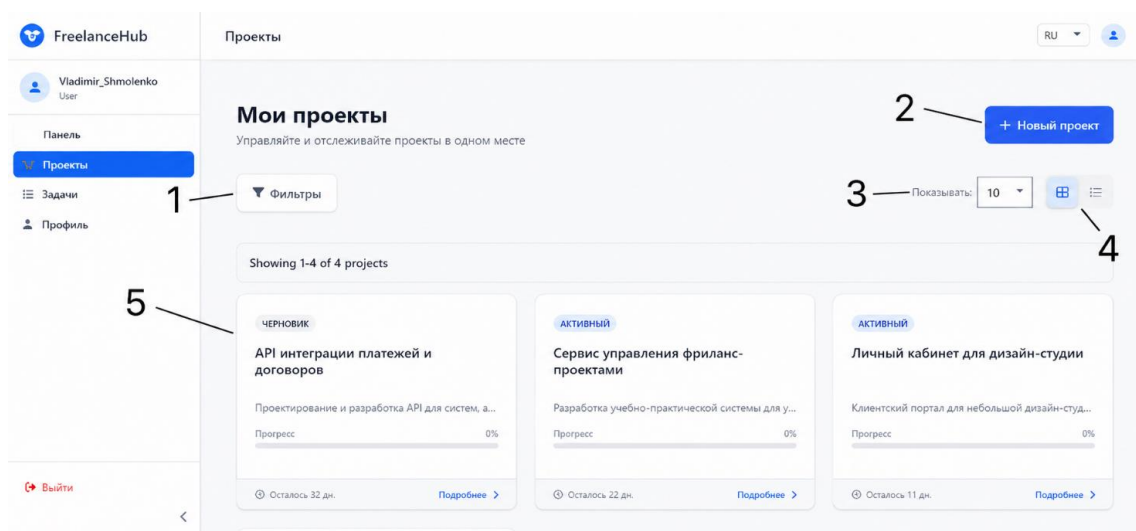


Рисунок 21 – Страница списка проектов

Рассмотрим основные элементы интерфейса страницы списка проектов, представленные на рисунке 21:

- 1 – кнопка фильтрации;
- 2 – кнопка создания проекта;
- 3 – выбор количества отображаемых записей;

4 – переключатель режима отображения;

5 – карточки проектов.

Карточное представление проектов выбрано для того, чтобы основные сведения были доступны без открытия каждой записи. Пользователь сразу видит название проекта, его статус, остаток времени и прогресс, а при необходимости может перейти к подробной карточке и выполнить доступные операции.

На странице списка проектов пользователь получает доступ к перечню доступных проектов, может применять фильтры, переключать режим отображения и переходить к детальному просмотру выбранного проекта. На данном экране также расположена кнопка создания нового проекта. Детальный просмотр выбранного проекта выполняется через карточку проекта, показанную на рисунке 22.

Такой экран используется как основной рабочий список проектов. Он позволяет не только просматривать созданные записи, но и быстро переходить к тем проектам, которые требуют уточнения, изменения статуса или добавления участников.

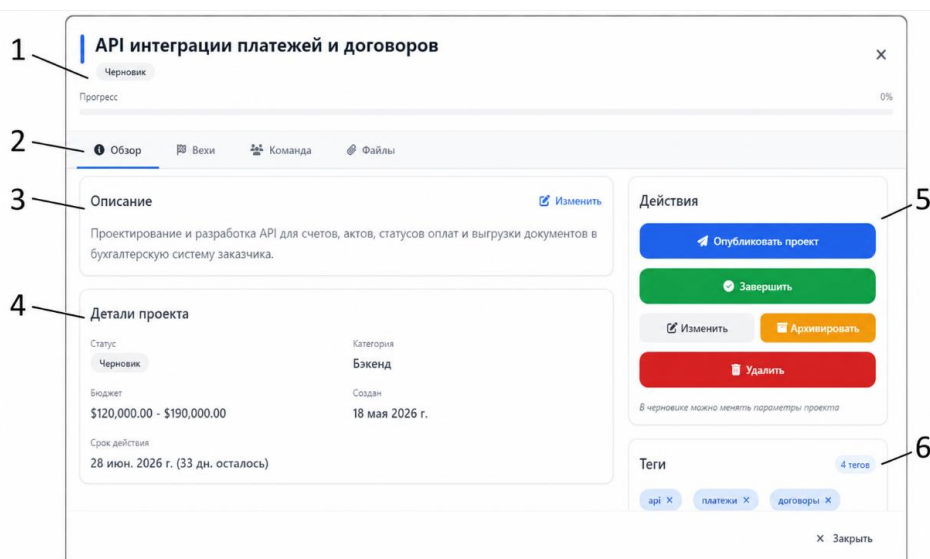


Рисунок 22 – Карточка проекта

Рассмотрим основные элементы интерфейса карточки проекта, представленные на рисунке 22:

1 – заголовок и статус проекта;

- 2 – вкладки карточки;
- 3 – описание проекта;
- 4 – основные сведения о проекте;
- 5 – блок действий;
- 6 – теги проекта.

Карточка проекта реализована в виде модального окна и объединяет сведения о наименовании проекта, его статусе, бюджете, сроках, тегах и доступных действиях. Из данного окна пользователь может перейти к редактированию проекта, публикации, завершению, архивированию или удалению.

Использование модального окна позволяет не разрывать основной сценарий работы со списком проектов. Пользователь получает подробную информацию о выбранном проекте поверх текущей страницы и после закрытия карточки возвращается к прежнему набору фильтров и режиму отображения.

Такое решение особенно удобно при последовательном просмотре нескольких проектов, поскольку пользователь не теряет контекст списка и может быстро сравнивать состояние разных проектных карточек.

Набор действий в карточке зависит от текущего статуса проекта. Благодаря этому пользователь видит только релевантные операции, а риск случайного выполнения недопустимого действия снижается.

Страница задач разработанного приложения представлена на рисунке 23.

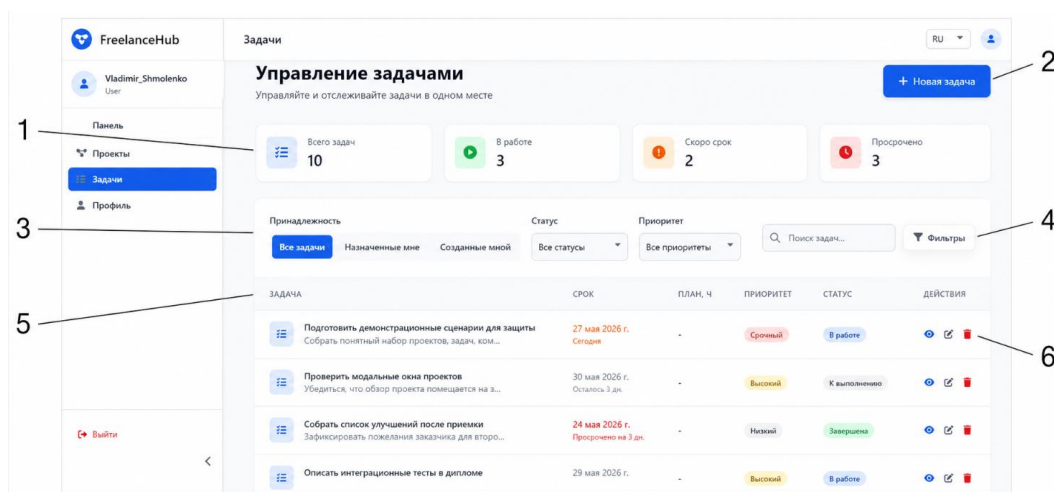


Рисунок 23 – Страница задач

Рассмотрим основные элементы интерфейса страницы задач, представленные на рисунке 23:

- 1 – сводные показатели по задачам;
- 2 – кнопка создания задачи;
- 3 – средства фильтрации;
- 4 – строка поиска;
- 5 – список задач;
- 6 – действия над задачей.

Раздел задач отделён от страницы проектов, поскольку пользователь может работать не только с задачами одного проекта, но и со всеми задачами, назначенными ему или созданными им. Такое решение упрощает ежедневную работу исполнителя и позволяет быстро находить задачи по статусу, приоритету или сроку.

Страница задач является самостоятельным разделом приложения. На ней отображаются сводные показатели по задачам, средства фильтрации по принадлежности, статусу и приоритету, строка поиска, а также табличный список задач с действиями просмотра, редактирования и удаления.

Наличие сводных показателей на странице задач позволяет пользователю быстро оценить текущую нагрузку и выявить проблемные элементы: задачи в работе, задачи с приближающимся сроком и просроченные задачи. Это снижает необходимость вручную просматривать весь список записей.

Фильтрация по принадлежности разделяет задачи, назначенные пользователю, созданные пользователем и доступные ему в рамках проектов. Такой подход делает страницу удобной как для исполнителя, так и для пользователя, контролирующего выполнение работ.

Дополнительно в разделе предусмотрен быстрый переход к карточке задачи, где пользователь может просмотреть подробную информацию и выполнить необходимые действия без перехода в другие разделы приложения.

Модальное окно деталей задачи представлено на рисунке 24.

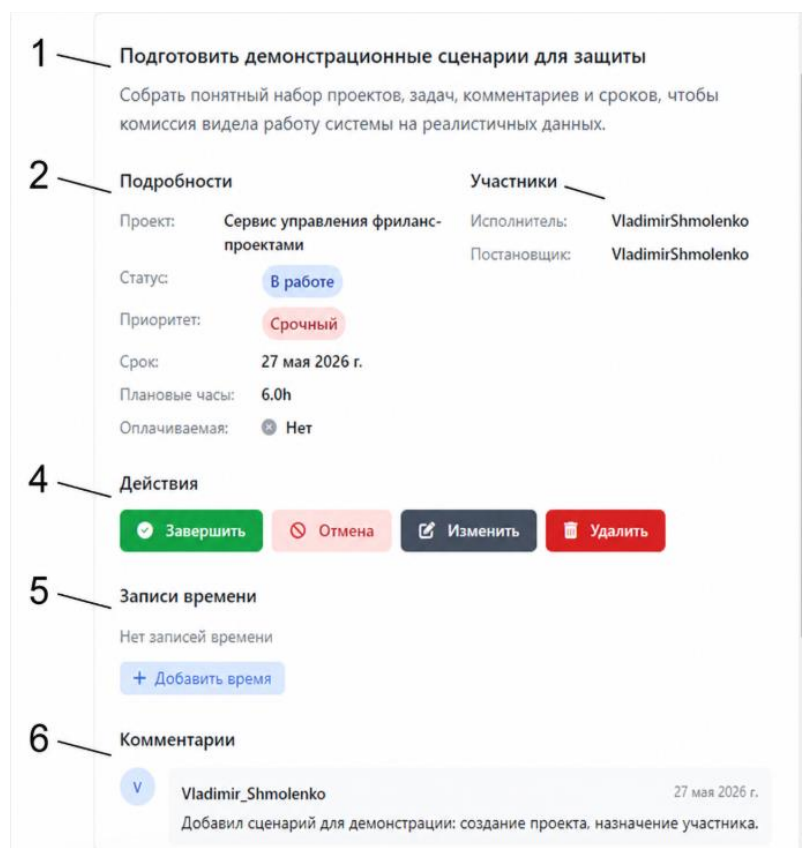


Рисунок 24 – Модальное окно деталей задачи

Рассмотрим основные элементы интерфейса модального окна деталей задачи, представленные на рисунке 24:

- 1 – название и описание задачи;
- 2 – основные параметры задачи;
- 3 – участники задачи;
- 4 – действия над задачей;
- 5 – блок записей времени;
- 6 – комментарии.

Детальная карточка задачи объединяет сведения, которые обычно требуются при выполнении работы: описание, срок, статус, приоритет, участников, комментарии и записи времени. Размещение этих данных в одном окне уменьшает количество переходов и делает сценарий контроля задачи более последовательным.

Модальное окно деталей задачи позволяет просматривать полные сведения о выбранной задаче: статус, приоритет, срок, исполнителя, постановщика,

записи учёта времени и комментарии. Из данного окна также доступны операции назначения исполнителя, запуска, отмены, изменения и удаления задачи.

Таким образом, карточка задачи объединяет информационный и управляющий сценарии: пользователь видит текущее состояние задачи и сразу получает доступ к действиям, которые разрешены для данного состояния.

Размещение комментариев и записей времени в карточке задачи делает историю работы более наглядной. Пользователь может проследить, какие изменения были внесены и сколько времени фактически затрачено на выполнение задачи.

Страница профиля пользователя представлена на рисунке 25.

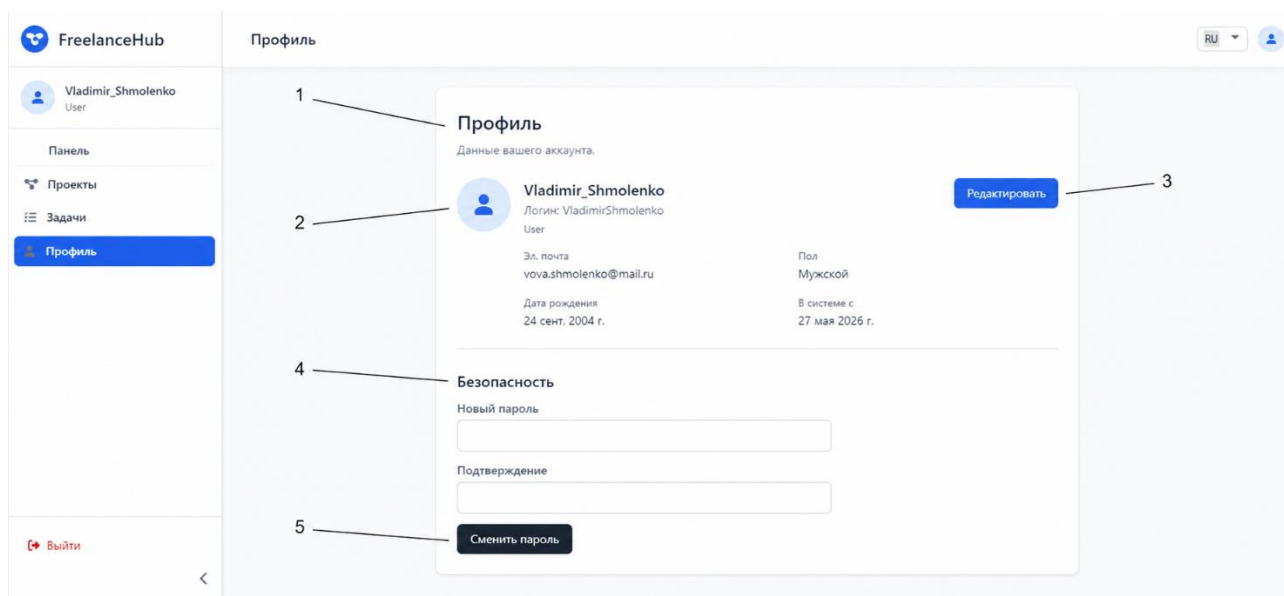


Рисунок 25 – Страница профиля пользователя

Рассмотрим основные элементы интерфейса страницы профиля пользователя, представленные на рисунке 25:

- 1 – карточка профиля;
- 2 – персональные данные пользователя;
- 3 – кнопка редактирования;
- 4 – форма смены пароля;
- 5 – кнопка смены пароля.

Страница профиля предназначена для просмотра и редактирования персональных данных пользователя. На ней отображаются сведения об учётной

записи, адрес электронной почты, пол, дата рождения, дата регистрации в системе, а также блок смены пароля.

Раздел профиля вынесен в отдельный экран, поскольку операции изменения персональных данных и пароля не относятся напрямую к управлению проектами или задачами. Это упрощает навигацию и отделяет пользовательские настройки от рабочих сценариев приложения.

Таким образом, в ходе программной реализации была создана работоспособная система управления фриланс-проектами, включающая реализованные базы данных, серверную и клиентскую части, а также пользовательский интерфейс, обеспечивающий выполнение основных сценариев работы.

## **4 Отладка и тестирование программного обеспечения**

### **4.1 Обоснование выбора режима тестирования**

Тестирование является одним из основных этапов контроля качества программного обеспечения. Оно позволяет выявить ошибки в работе приложения, проверить соответствие реализованных функций заявленным требованиям и убедиться в корректности поведения системы в различных пользовательских сценариях. Модульные проверки выполнялись с использованием xUnit.net [22]. Интеграционные проверки запускались с применением Testcontainers for .NET [23].

Для веб-приложений, построенных на основе клиент-серверного взаимодействия и микросервисной архитектуры, особенно важно проверять не только отдельные функции, но и согласованность работы компонентов между собой. В разрабатываемой системе управление фриланс-проектами связано с обработкой пользователей, проектов, задач, уведомлений и межсервисных событий, поэтому проверка должна охватывать как серверную, так и клиентскую часть приложения.

Существует несколько основных видов тестирования программного обеспечения: функциональное, модульное, интеграционное, системное и тестирование производительности. Каждый из перечисленных видов направлен на проверку отдельного аспекта качества программного продукта и применяется в зависимости от структуры приложения и целей контроля.

Функциональное тестирование позволяет проверить, соответствует ли поведение системы функциональным требованиям. В рамках такого тестирования оцениваются пользовательские сценарии: регистрация и авторизация, создание и редактирование проекта, работа со списком задач, назначение исполнителя, изменение статуса задачи, добавление комментариев и просмотр профиля пользователя.

Модульное тестирование предназначено для проверки отдельных компонентов программного кода без запуска всей системы целиком. Такой

подход позволяет проверить обработчики команд и запросов, сервисы бизнес-логики, механизмы валидации данных и отдельные методы, отвечающие за выполнение прикладных операций.

Интеграционное тестирование направлено на проверку взаимодействия между компонентами системы. Для разрабатываемой системы данный вид тестирования имеет особое значение, так как клиентская часть обращается к серверным сервисам через API Gateway, а микросервисы пользователей, проектов и задач взаимодействуют с собственными базами данных и обмениваются событиями через брокер сообщений.

Системное тестирование применяется для проверки программного продукта как единого целого. Оно позволяет убедиться, что приложение корректно работает в условиях, приближенных к реальной эксплуатации, а основные сценарии пользователя выполняются без нарушения логики переходов и обработки данных.

Для контроля качества системы управления фриланс-проектами был выбран комбинированный подход, включающий функциональное, модульное и интеграционное тестирование. Функциональное тестирование используется для проверки пользовательских сценариев на стороне интерфейса, модульное тестирование – для проверки серверной бизнес-логики, а интеграционное тестирование – для контроля корректности обмена данными между клиентской частью, API и микросервисами.

Выбор указанных видов тестирования объясняется особенностями разработанной системы. Приложение имеет распределённую структуру, использует отдельные базы данных для сервисов и содержит значительное количество пользовательских операций. Поэтому выбранный режим тестирования позволяет проверить как корректность отдельных компонентов, так и работоспособность системы при выполнении сквозных сценариев управления фриланс-проектами.

## **4.2 Тестирование серверной части системы**

Для проверки серверной части системы управления фриланс-проектами были выполнены функциональные, интеграционные и модульные проверки. Такой подход позволяет оценить корректность работы внешних точек доступа, взаимодействие сервисов с базами данных и выполнение отдельных операций бизнес-логики.

Основными объектами проверки являлись сервис пользователей, сервис проектов и сервис задач. Каждый из перечисленных сервисов имеет собственную область ответственности, поэтому результаты серверного тестирования целесообразно рассматривать по видам проверок.

### **4.2.1 Функциональное тестирование серверной части**

Функциональное тестирование проводилось методом «чёрного ящика». При таком подходе проверяется реакция программного средства на входные данные без анализа внутренней реализации. Для разработанной системы проверки выполнялись через пользовательский интерфейс приложения: при регистрации, работе с проектами и задачами клиентская часть обращалась к REST API микросервисов, а результат оценивался по отображаемому состоянию системы.

При выполнении функционального тестирования рассматривались не только корректные сценарии работы, но и ошибочные ситуации: ввод неверных учетных данных, обращение к несуществующим объектам и попытка перехода сущности в недопустимое состояние. Это позволило проверить, что серверная часть возвращает предсказуемый результат и не нарушает целостность данных при некорректных действиях пользователя.

Краткое описание функциональных проверок сервиса пользователей приведено в таблице 33. Результаты выполнения проверок приведены в приложении Е (рисунки Е.1–Е.5).

Таблица 33 – Функциональные проверки сервиса пользователей

| Тест                           | Описание                                                 | Ожидаемый результат                             | Актуальный результат                                  |
|--------------------------------|----------------------------------------------------------|-------------------------------------------------|-------------------------------------------------------|
| Регистрация пользователя       | Отправка корректных регистрационных данных               | Учётная запись пользователя создаётся           | Учётная запись создана, возвращён успешный ответ      |
| Авторизация пользователя       | Проверка входа по логину и паролю                        | Пользователь получает токен доступа             | Токен доступа возвращён пользователю                  |
| Получение профиля пользователя | Запрос данных авторизованного пользователя               | Возвращаются данные текущего пользователя       | Профиль пользователя получен                          |
| Изменение профиля пользователя | Обновление персональных данных пользователя              | Данные пользователя сохраняются                 | Обновлённые данные возвращаются при повторном запросе |
| Авторизация с неверным паролем | Передача неверного пароля для существующего пользователя | Доступ не предоставляется, токен не формируется | Сервер возвращает отказ в авторизации                 |

Краткое описание функциональных проверок сервиса проектов приведено в таблице 34. Результаты выполнения проверок приведены в приложении Е (рисунки Е.6–Е.10).

Таблица 34 – Функциональные проверки сервиса проектов

| Тест              | Описание                                  | Ожидаемый результат                       | Актуальный результат                          |
|-------------------|-------------------------------------------|-------------------------------------------|-----------------------------------------------|
| Создание проекта  | Передача корректных данных нового проекта | Проект создаётся в системе                | Проект создан, возвращён идентификатор записи |
| Получение проекта | Запрос данных существующего проекта       | Возвращаются сведения о выбранном проекте | Данные проекта получены                       |

Продолжение таблицы 34

| Тест                                 | Описание                                                       | Ожидаемый результат                                       | Актуальный результат                                     |
|--------------------------------------|----------------------------------------------------------------|-----------------------------------------------------------|----------------------------------------------------------|
| Изменение проекта                    | Изменение названия, описания и параметров проекта              | Данные проекта обновляются                                | Изменения сохранены и отображаются при повторном запросе |
| Добавление участника проекта         | Добавление пользователя в состав участников проекта            | Пользователь добавляется в проект                         | Список участников проекта обновлён                       |
| Добавление несуществующего участника | Передача идентификатора пользователя, отсутствующего в системе | Участник не добавляется, возвращается сообщение об ошибке | Операция отклонена, состав проекта не изменён            |

Краткое описание функциональных проверок сервиса задач приведено в таблице 35. Результаты выполнения проверок приведены в приложении Е (рисунки Е.11–Е.17).

Таблица 35 – Функциональные проверки сервиса задач

| Тест                     | Описание                                                 | Ожидаемый результат                          | Актуальный результат                           |
|--------------------------|----------------------------------------------------------|----------------------------------------------|------------------------------------------------|
| Создание задачи          | Создание задачи в рамках выбранного проекта              | Задача создаётся и получает начальный статус | Задача создана, возвращён идентификатор записи |
| Назначение исполнителя   | Передача идентификатора исполнителя для выбранной задачи | Исполнитель закрепляется за задачей          | Данные исполнителя сохранены в задаче          |
| Запуск выполнения задачи | Отправка запроса на перевод задачи в работу              | Задача переходит в состояние выполнения      | Статус задачи изменён на состояние выполнения  |

Продолжение таблицы 35

| Тест                                 | Описание                                                  | Ожидаемый результат                           | Актуальный результат                                            |
|--------------------------------------|-----------------------------------------------------------|-----------------------------------------------|-----------------------------------------------------------------|
| Завершение задачи                    | Отправка запроса на завершение задачи                     | Задача переходит в завершённое состояние      | Статус задачи изменён на завершённый                            |
| Добавление комментария               | Добавление комментария к выбранной задаче                 | Комментарий сохраняется в списке комментариев | Комментарий добавлен и доступен при просмотре задачи            |
| Добавление записи времени            | Добавление записи о затраченном времени                   | Запись времени сохраняется                    | Запись времени сохранена и учитывается в данных задачи          |
| Переход задачи в недопустимый статус | Попытка выполнить недопустимое изменение состояния задачи | Статус задачи не изменяется                   | Сервер возвращает ошибку, данные задачи сохранены без изменений |

По результатам функционального тестирования можно сделать вывод, что основные пользовательские сценарии выполняются корректно, а точки доступа серверной части возвращают ожидаемые ответы и обеспечивают изменение состояния системы.

#### 4.2.2 Интеграционное тестирование серверной части

Интеграционное тестирование применялось для проверки взаимодействия серверных компонентов, которые не могут быть полноценно проверены изолированно. В рамках данного вида тестирования контролировалась работа API Gateway, баз данных микросервисов, механизма outbox, хранилища входящих событий, фоновых обработчиков и брокера сообщений Kafka [3]. Для проверки инфраструктурных зависимостей использовались Testcontainers for .NET [23].

Особое внимание уделялось сценариям, в которых результат операции появляется не сразу после HTTP-запроса, а после обработки события. К таким

сценариям относятся публикация сообщения в outbox, передача события через Kafka, сохранение записи во входящих событиях и обновление проекции данных в другом микросервисе.

Краткое описание интеграционных проверок приведено в таблице 36. Результаты запуска интеграционных проверок приведены в приложении Ж (рисунки Ж.1–Ж.3).

Таблица 36 – Интеграционные проверки серверной части

| Тест                                | Описание                                                                              | Ожидаемый результат                                                                           | Актуальный результат                                                              |
|-------------------------------------|---------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Маршрутизация через API Gateway     | Отправка запроса к серверной операции через единый входной адрес приложения           | Запрос передаётся API Gateway в нужный микросервис, клиент получает ответ сервиса             | Запрос обработан соответствующим микросервисом, ответ возвращён через gateway     |
| Сохранение события в outbox         | Выполнение операции, создающей доменное событие, например создание проекта или задачи | Событие сохраняется в таблице исходящих сообщений вместе с изменениями основной сущности      | Запись основной сущности и outbox-сообщение сохранены в одной базе данных сервиса |
| Публикация outbox-сообщения в Kafka | Запуск фонового обработчика публикации исходящих сообщений                            | Неопубликованное сообщение считывается из outbox и отправляется в соответствующий topic Kafka | Сообщение опубликовано, запись outbox помечена как обработанная                   |
| Получение входящего события         | Передача события из Kafka в сервис, подписанный на соответствующий topic              | Consumer получает сообщение и сохраняет его во входящее хранилище событий                     | Входящее событие сохранено и подготовлено к дальнейшей обработке                  |

Продолжение таблицы 36

| Тест                              | Описание                                              | Ожидаемый результат                                                                                  | Актуальный результат                                             |
|-----------------------------------|-------------------------------------------------------|------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| Обработка входящего события       | Запуск фонового обработчика входящих событий          | Событие считывается из хранилища, передаётся нужному обработчику и изменяет связанные данные сервиса | Событие обработано, связанное состояние сервиса обновлено        |
| Идемпотентность обработки события | Повторная передача уже сохранённого входящего события | Повторная обработка не приводит к дублированию связанных данных                                      | Повторное событие распознано, состояние данных не продублировано |

#### 4.2.3 Модульное тестирование серверной части

Модульное тестирование применялось для проверки отдельных обработчиков и сервисов бизнес-логики. Данный вид тестирования позволяет проверить отдельные операции без запуска полного пользовательского сценария и быстрее выявить ошибки в прикладной логике. Проверки выполнялись с использованием xUnit.net [22].

Краткое описание модульных проверок приведено в таблице 37. Результаты запуска модульных проверок приведены в приложении Ж (рисунки Ж.4–Ж.6).

Таблица 37 – Модульные проверки серверной части

| Тест                            | Описание                                                        | Ожидаемый результат                                   | Актуальный результат       |
|---------------------------------|-----------------------------------------------------------------|-------------------------------------------------------|----------------------------|
| Проверка регистрационных данных | Проверка обработки корректных и некорректных данных регистрации | Валидные данные принимаются, некорректные отклоняются | Проверка выполнена успешно |

Продолжение таблицы 37

| Тест                            | Описание                                             | Ожидаемый результат                                | Актуальный результат                           |
|---------------------------------|------------------------------------------------------|----------------------------------------------------|------------------------------------------------|
| Создание проекта                | Проверка выполнения операции создания проекта        | Формируется новая запись проекта                   | Проект создаётся при корректных входных данных |
| Добавление участника проекта    | Проверка добавления пользователя в состав участников | Участник добавляется без нарушения состава проекта | Участник добавлен                              |
| Запуск выполнения задачи        | Проверка перевода задачи в состояние выполнения      | Задача переходит только в допустимое состояние     | Переход состояния выполнен корректно           |
| Добавление комментария к задаче | Проверка добавления комментария к выбранной задаче   | Комментарий создаётся для выбранной задачи         | Комментарий добавлен                           |

Таким образом, серверная часть системы прошла функциональные, интеграционные и модульные проверки. Полученные результаты подтверждают корректность работы основных операций, связанных с пользователями, проектами и задачами.

## **5 Контроль качества программного обеспечения**

### **5.1 Обоснование выбора метрик кода**

Метрики кода используются для оценки качества программного обеспечения. Они позволяют получить количественное представление о сложности программного продукта, объеме реализованной логики и трудоемкости дальнейшего сопровождения.

Для оценки качества системы управления фриланс-проектами выбраны метрики Холстеда и Чепина. Метрика Холстеда применяется для оценки объема и сложности программного кода на основе количества операторов и операндов. Метрика Чепина позволяет оценить сложность модуля через анализ переменных списка ввода-вывода и их функциональной роли в программе.

Расчет выполнен для метода обработки команды изменения статуса задачи `ChangeTaskStatusCommandHandler.Handle`. Данный метод выбран потому, что он относится к ключевой бизнес-логике сервиса задач: проверяет существование задачи, анализирует допустимость перехода между состояниями, изменяет статус и сохраняет результат выполнения операции.

### **5.2 Оценка качества программного кода**

#### **5.2.1 Метрика Холстеда**

Для расчета метрик Холстеда необходимо определить четыре базовые характеристики программного кода: количество уникальных операторов  $n_1$ , количество уникальных операндов  $n_2$ , общее количество операторов  $N_1$  и общее количество операндов  $N_2$ . На основе этих значений вычисляются словарь программы, длина программы, объем, сложность и усилия.

Словарь программы определяется по формуле:

$$n = n_1 + n_2,$$

где  $n_1$  – количество уникальных операторов;  $n_2$  – количество уникальных операндов.

Длина программы определяется по формуле:

$$N = N_1 + N_2,$$

где  $N_1$  – общее количество операторов;  $N_2$  – общее количество операндов.

Объем программы рассчитывается по формуле:

$$V = N \cdot \log_2(n),$$

где  $N$  – длина программы;  $n$  – словарь программы.

Сложность программы определяется по формуле:

$$D = \left(\frac{n_1}{2}\right) \cdot \left(\frac{N_2}{n_2}\right),$$

где  $n_1$  – количество уникальных операторов;  $N_2$  – общее количество операндов;  $n_2$  – количество уникальных операндов.

Усилия рассчитываются по формуле:

$$E = D \cdot V,$$

где  $D$  – сложность программы;  $V$  – объем программы.

Результаты подсчета операторов и операндов для выбранного метода приведены в таблице 38.

Подсчет выполнен по основным операторам и операндам выбранного метода.

Таблица 38 – Расчет операторов и операндов для метрики Холстеда

| №  | Операторы | Число операторов | Операнды            | Число операндов |
|----|-----------|------------------|---------------------|-----------------|
| 1  | public    | 1                | request             | 7               |
| 2  | async     | 1                | task                | 9               |
| 3  | await     | 3                | taskId              | 3               |
| 4  | if        | 4                | newStatus           | 4               |
| 5  | return    | 5                | currentStatus       | 3               |
| 6  | throw     | 2                | cancellationToken   | 4               |
| 7  | =         | 7                | repository          | 2               |
| 8  | ==        | 4                | transitionValidator | 2               |
| 9  | !=        | 2                | domainEvent         | 2               |
| 10 | .         | 21               | TaskStatus          | 5               |
| 11 | ()        | 15               | Result              | 3               |
| 12 | ;         | 12               | OutboxMessage       | 1               |

Продолжение таблицы 38

| №  | Операторы | Число операторов | Операнды        | Число операндов |
|----|-----------|------------------|-----------------|-----------------|
| 13 | =>        | 2                | DateTime.UtcNow | 1               |
| 14 | &&        | 1                | userId          | 2               |
| 15 | new       | 3                | comment         | 1               |
| 16 | switch    | 1                | messageBus      | 1               |
| 17 | case      | 4                | saveChanges     | 1               |
| 18 | break     | 4                | validationError | 1               |

По данным таблицы 38 количество уникальных операторов  $n_1$  равно 18, количество уникальных операндов  $n_2$  равно 18, общее количество операторов  $N_1$  равно 92, общее количество операндов  $N_2$  равно 52. На основании этих значений рассчитаны итоговые показатели метрики Холстеда, приведенные в таблице 39.

Итоговые значения получены на основе базовых показателей  $n_1$ ,  $n_2$ ,  $N_1$  и  $N_2$ .

Таблица 39 – Значения метрики Холстеда

| Характеристика          | Формула                                                         | Значение |
|-------------------------|-----------------------------------------------------------------|----------|
| Словарь программы $n$   | $n_1 + n_2$                                                     | 36       |
| Длина программы $N$     | $N_1 + N_2$                                                     | 144      |
| Объем программы $V$     | $N \cdot \log_2(n)$                                             | 744,5    |
| Сложность программы $D$ | $\left(\frac{n_1}{2}\right) \cdot \left(\frac{N_2}{n_2}\right)$ | 26,0     |
| Усилия $E$              | $D \cdot V$                                                     | 19357,0  |

Полученные значения показывают, что выбранный метод имеет среднюю сложность. Это объясняется наличием нескольких проверок бизнес-правил и вариантов перехода между состояниями задачи. При этом логика метода остается контролируемой, так как операции проверки, сохранения и публикации события разделены между отдельными зависимостями.

### 5.2.2 Метрика Чепина

Метрика Чепина используется для оценки качества модуля на основе анализа переменных списка ввода-вывода. Все переменные распределяются по четырем группам: Р – переменные, используемые для ввода и вывода данных; М – модифицируемые переменные; С – управляющие переменные; Т – неиспользуемые или вспомогательные переменные, не влияющие на работу модуля.

Полная метрика Чепина рассчитывается по формуле:

$$Q = a_1 \cdot P + a_2 \cdot M + a_3 \cdot C + a_4 \cdot T,$$

где  $a_1$  – коэффициент группы Р, равный 1;

$a_2$  – коэффициент группы М, равный 2;

$a_3$  – коэффициент группы С, равный 3;

$a_4$  – коэффициент группы Т, равный 0,5;

Р – вводимые переменные;

М – модифицируемые переменные;

С – управляющие переменные; Т – неиспользуемые переменные.

Распределение переменных выбранного метода по группам представлено в таблице 40.

Переменные распределены по группам в соответствии с их ролью в алгоритме.

Такое распределение позволяет определить, какие данные только поступают в метод, какие изменяются в ходе выполнения, а какие управляют ветвлением логики. За счёт этого метрика отражает не объём текста метода, а влияние переменных на сложность его понимания.

Таблица 40 – Расчет метрики Чепина

| Группа | Переменные                                            | Количество | Коэффициент | Значение |
|--------|-------------------------------------------------------|------------|-------------|----------|
| Р      | request, taskId, newStatus, userId, cancellationToken | 5          | 1           | 5        |

Продолжение таблицы 40

| Группа | Переменные                                                                | Количество | Коэффициент | Значение |
|--------|---------------------------------------------------------------------------|------------|-------------|----------|
| М      | task, currentStatus,<br>domainEvent,<br>outboxMessage                     | 4          | 2           | 8        |
| С      | task, currentStatus,<br>newStatus, validationResult,<br>transitionAllowed | 5          | 3           | 15       |
| Т      | comment,<br>DateTime.UtcNow                                               | 2          | 0,5         | 1        |
| Итого  |                                                                           | 16         |             | 29       |

Значение полной метрики Чепина равно 29. Данный результат показывает, что метод имеет умеренную сложность: в нем присутствуют управляющие переменные, связанные с проверкой допустимости перехода задачи между статусами, однако количество таких переменных не является чрезмерным. Следовательно, метод можно сопровождать без существенного риска при условии сохранения разделения бизнес-правил и инфраструктурной логики.

## 6 Планирование разработки и оценка бюджета

### 6.1 Планирование разработки

Планирование разработки выполнялось с использованием проектного подхода. Для этого был сформирован календарный план работ, включающий анализ требований, проектирование, реализацию серверной и клиентской частей, тестирование, контроль качества и оформление пояснительной записки.

Календарный план подготовлен в ProjectLibre. Основные этапы работ приведены в таблице 41.

Этапы расположены в порядке их выполнения при разработке системы.

Таблица 41 – План выполнения работ

| №  | Этап                                         | Длительность | Предшественник |
|----|----------------------------------------------|--------------|----------------|
| 1  | Анализ предметной области и требований       | 5 дней       | -              |
| 2  | Анализ программ-аналогов                     | 3 дней       | 1              |
| 3  | Формирование функциональных требований       | 3 дней       | 2              |
| 4  | Проектирование архитектуры приложения        | 4 дней       | 3              |
| 5  | Проектирование баз данных                    | 4 дней       | 4              |
| 6  | Проектирование пользовательского интерфейса  | 3 дней       | 4              |
| 7  | Реализация сервиса пользователей             | 6 дней       | 5              |
| 8  | Реализация сервиса проектов                  | 7 дней       | 7              |
| 9  | Реализация сервиса задач                     | 7 дней       | 8              |
| 10 | Настройка API Gateway и межсервисного обмена | 4 дней       | 7; 8; 9        |
| 11 | Реализация клиентской части                  | 8 дней       | 6; 10          |
| 12 | Функциональное тестирование                  | 3 дней       | 11             |
| 13 | Интеграционное и модульное тестирование      | 3 дней       | 12             |
| 14 | Контроль качества программного кода          | 2 дней       | 13             |
| 15 | Оформление пояснительной записки             | 5 дней       | 14             |
| 16 | Завершение проекта                           | 0 дней       | 15             |



|   | ① | Название             | Тип    | Инициалы | Максимальные единицы | Стандартная ставка | Ставка сверхурочных | Затраты на одно испол... | Начислять | Основной календарь |
|---|---|----------------------|--------|----------|----------------------|--------------------|---------------------|--------------------------|-----------|--------------------|
| 1 |   | Аналитик             | Работа | Ан       | 100 %                | 500,00 Р/час       | 750,00 Р/час        | 0,00 Р Пропорционально   |           | Пятидневка         |
| 2 |   | Программист          | Работа | Пр       | 100 %                | 650,00 Р/час       | 950,00 Р/час        | 0,00 Р Пропорционально   |           | Пятидневка         |
| 3 |   | Тестировщик          | Работа | Те       | 100 %                | 450,00 Р/час       | 650,00 Р/час        | 0,00 Р Пропорционально   |           | Пятидневка         |
| 4 |   | Технический писатель | Работа | Те       | 100 %                | 400,00 Р/час       | 600,00 Р/час        | 0,00 Р Пропорционально   |           | Пятидневка         |
| 5 |   | Интернет             | Работа | Ин       | 100 %                | 50,00 Р/час        | 50,00 Р/час         | 0,00 Р Пропорционально   |           | Пятидневка         |
| 6 |   | Компьютеры           | Работа | Ко       | 100 %                | 80,00 Р/час        | 80,00 Р/час         | 0,00 Р Пропорционально   |           | Пятидневка         |

Рисунок 50 – Данные о ресурсах проекта разработки

Всего в календарном плане выделено 16 задач, включая завершающую веху проекта. Перечень задач, сроки их выполнения и предшественники представлены на рисунке 51.

Связи между задачами позволяют определить порядок выполнения работ и избежать ситуации, когда реализация начинается до завершения проектирования. Наличие предшественников также помогает корректно пересчитывать календарный план при изменении длительности отдельных этапов.

|    | ① | Название                                       | Продолжи...                  | Начало            | Окончание | Предшественные | Название ресурса                                              |
|----|---|------------------------------------------------|------------------------------|-------------------|-----------|----------------|---------------------------------------------------------------|
|    |   | Система управления филиалс-проектами           | 51,25 дней 08.04.2025, 09:00 | 18.06.2025, 11:00 |           |                |                                                               |
| 1  |   | Анализ предметной области и требований         | 3,875 дней 08.04.2025, 09:00 | 11.04.2025, 17:00 |           |                | Аналитик[50 %];Интернет;Компьютеры                            |
| 2  |   | Анализ программ-аналогов                       | 3 дней 14.04.2025, 08:00     | 16.04.2025, 17:00 | 1         |                | Аналитик[50 %];Интернет;Компьютеры                            |
| 3  |   | Формирование функциональных требований         | 2,25 дней 17.04.2025, 08:00  | 21.04.2025, 10:00 | 2         |                | Аналитик[50 %];Интернет;Компьютеры                            |
| 4  |   | Проектирование архитектуры приложения          | 4 дней 21.04.2025, 10:00     | 25.04.2025, 10:00 | 3         |                | Аналитик[50 %];Программист[50 %];Интернет;Компьютеры          |
| 5  |   | Проектирование баз данных                      | 4 дней 25.04.2025, 10:00     | 01.05.2025, 10:00 | 4         |                | Программист[50 %];Интернет;Компьютеры                         |
| 6  |   | Проектирование пользовательского интерфейса    | 3 дней 25.04.2025, 10:00     | 30.04.2025, 10:00 | 4         |                | Аналитик[50 %];Программист[50 %];Интернет;Компьютеры          |
| 7  |   | Реализация сервиса пользователей               | 4,75 дней 01.05.2025, 10:00  | 07.05.2025, 17:00 | 5         |                | Программист[50 %];Интернет;Компьютеры                         |
| 8  |   | Реализация сервиса проектов                    | 5 дней 08.05.2025, 08:00     | 14.05.2025, 17:00 | 7         |                | Программист[50 %];Интернет;Компьютеры                         |
| 9  |   | Реализация сервиса задач                       | 4,375 дней 15.05.2025, 08:00 | 21.05.2025, 11:00 | 8         |                | Программист[50 %];Интернет;Компьютеры                         |
| 10 |   | Настройка API Gateway и мексерсингового обмена | 4 дней 21.05.2025, 11:00     | 27.05.2025, 11:00 | 7;8;9     |                | Программист[50 %];Интернет;Компьютеры                         |
| 11 |   | Реализация клиентской части                    | 4 дней 27.05.2025, 11:00     | 02.06.2025, 11:00 | 6;10      |                | Программист[50 %];Интернет;Компьютеры                         |
| 12 |   | Функциональное тестирование                    | 3 дней 02.06.2025, 11:00     | 05.06.2025, 11:00 | 11        |                | Тестировщик[50 %];Интернет;Компьютеры                         |
| 13 |   | Интеграционное и модульное тестирование        | 3 дней 05.06.2025, 11:00     | 10.06.2025, 11:00 | 12        |                | Программист[50 %];Тестировщик[50 %];Интернет;Компьютеры       |
| 14 |   | Контроль качества программного кода            | 2 дней 10.06.2025, 11:00     | 12.06.2025, 11:00 | 13        |                | Аналитик[50 %];Программист[50 %];Тестировщик[50 %];Компьютеры |
| 15 |   | Оформление пояснительной записки               | 4 дней 12.06.2025, 11:00     | 18.06.2025, 11:00 | 14        |                | Технический писатель[50 %];Интернет;Компьютеры                |
| 16 |   | Завершение проекта                             | 0 дней 18.06.2025, 11:00     | 18.06.2025, 11:00 | 15        |                |                                                               |

Рисунок 51 – Планирование разработки системы

## 6.2 Расчет бюджета разработки

Расчет бюджета разработки выполнен на основе трудоемкости задач, стоимости ресурсов и процента загрузки исполнителей. Для трудовых ресурсов указаны стандартная и сверхурочная ставки, а для вспомогательных ресурсов учитывается почасовая стоимость доступа к сети Интернет и использования компьютерного оборудования. Итоговая стоимость рассчитывается как сумма затрат по всем назначенным ресурсам на каждой задаче календарного плана.

Для каждого ресурса указана его роль в проекте и расчетные ставки (таблица 42).

Таблица 42 – Ресурсы проекта

| Ресурс               | Назначение                                              | Стандартная ставка, руб./ч | Сверхурочная ставка, руб./ч |
|----------------------|---------------------------------------------------------|----------------------------|-----------------------------|
| Аналитик             | Анализ требований и проектирование                      | 500                        | 750                         |
| Программист          | Реализация серверной и клиентской частей                | 650                        | 950                         |
| Тестировщик          | Функциональное, интеграционное и модульное тестирование | 450                        | 650                         |
| Технический писатель | Оформление пояснительной записки                        | 400                        | 600                         |
| Интернет             | Доступ к сетевым ресурсам и документации                | 50                         | 50                          |
| Компьютеры           | Использование рабочего оборудования                     | 80                         | 80                          |

Расчет выполнен по основным этапам работ с учетом стоимости задач, отображаемой в ProjectLibre. Доля затрат определена как отношение стоимости этапа к общей стоимости разработки, равной 291520 руб. Итоговая смета затрат на разработку приведена в таблице 43.

Таблица 43 – Смета затрат на разработку

| Этап                                                | Стоимость, руб. | Доля затрат, % |
|-----------------------------------------------------|-----------------|----------------|
| Анализ и постановка требований                      | 27740           | 9,5%           |
| Проектирование архитектуры, баз данных и интерфейса | 64440           | 22,1%          |
| Реализация серверной части                          | 113100          | 38,8%          |
| Реализация клиентской части                         | 24960           | 8,6%           |
| Тестирование                                        | 30240           | 10,4%          |
| Контроль качества                                   | 14080           | 4,8%           |
| Оформление документации                             | 16960           | 5,8%           |
| Итого                                               | 291520          | 100%           |

График затрат и распределение стоимости по задачам можно получить в ProjectLibre после открытия подготовленного файла проекта. На рисунке 52 отображены стоимости отдельных задач: анализ предметной области и требований – 11780 руб., анализ программ-аналогов – 9120 руб., формирование

функциональных требований – 6840 руб., проектирование архитектуры приложения – 22560 руб., проектирование баз данных – 24960 руб., проектирование пользовательского интерфейса – 16920 руб., реализация сервиса пользователей – 29640 руб., реализация сервиса проектов – 31200 руб., реализация сервиса задач – 27300 руб., настройка API Gateway и межсервисного обмена – 24960 руб., реализация клиентской части – 24960 руб., функциональное тестирование – 13920 руб., интеграционное и модульное тестирование – 16320 руб., контроль качества программного кода – 14080 руб., оформление пояснительной записки – 16960 руб. Оценка денежных затрат по задачам разработки представлена на рисунке 52.

Распределение стоимости показывает, что наибольшая доля бюджета приходится на реализацию ключевых модулей системы, интеграцию микросервисов, тестирование и оформление итоговой документации. Это соответствует структуре проекта, поскольку данные этапы требуют наибольшего объёма ручной работы, проверки результата и согласования полученных материалов.

Представленный расчёт используется как ориентировочная оценка затрат на разработку. Он показывает порядок стоимости работ и позволяет сопоставить трудоёмкость этапов с полученным календарным планом.

|    | Название                                     | Оставшаяся стоимость |
|----|----------------------------------------------|----------------------|
|    | <b>Система управления фриланс-проектами</b>  | <b>291520,00 Р</b>   |
| 1  | Анализ предметной области и требований       | 11780,00 Р           |
| 2  | Анализ программ-аналогов                     | 9120,00 Р            |
| 3  | Формирование функциональных требований       | 6840,00 Р            |
| 4  | Проектирование архитектуры приложения        | 22560,00 Р           |
| 5  | Проектирование баз данных                    | 24960,00 Р           |
| 6  | Проектирование пользовательского интерфейса  | 16920,00 Р           |
| 7  | Реализация сервиса пользователей             | 29640,00 Р           |
| 8  | Реализация сервиса проектов                  | 31200,00 Р           |
| 9  | Реализация сервиса задач                     | 27300,00 Р           |
| 10 | Настройка API Gateway и межсервисного обмена | 24960,00 Р           |
| 11 | Реализация клиентской части                  | 24960,00 Р           |
| 12 | Функциональное тестирование                  | 13920,00 Р           |
| 13 | Интеграционное и модульное тестирование      | 16320,00 Р           |
| 14 | Контроль качества программного кода          | 14080,00 Р           |
| 15 | Оформление пояснительной записки             | 16960,00 Р           |
| 16 | Завершение проекта                           | 0,00 Р               |

Рисунок 52 – Оценка бюджета разработки

Итоговая стоимость разработки по расчетной смете составляет 291520 рублей. Расчет складывается из стоимости подготовительных работ, проектирования, реализации серверной и клиентской частей, тестирования, контроля качества и оформления пояснительной записки. Наибольшая доля затрат приходится на реализацию серверной части – 38,8 %, так как она включает разработку микросервисов, межсервисный обмен, работу с базами данных и бизнес-логику системы.

Полученная смета не является бухгалтерским расчетом фактических расходов, а используется как проектная оценка трудоемкости разработки. Она позволяет сопоставить стоимость этапов с календарным планом и определить, какие работы оказывают наибольшее влияние на общий бюджет.

Наименьшая доля затрат приходится на контроль качества программного кода и оформление документации, однако эти этапы не исключаются из плана, поскольку они обеспечивают проверку результата и подготовку материалов, необходимых для защиты выпускной квалификационной работы.

Таким образом, расчет бюджета подтверждает, что основная стоимость проекта связана с реализацией функциональности и интеграцией компонентов системы, а не только с подготовительными или оформительскими работами.

Расчетные данные также показывают, что выбранный календарный план не содержит необоснованно длительных подготовительных этапов. Основная трудоемкость сосредоточена в работах, связанных с разработкой и проверкой программных компонентов, что соответствует практическому характеру выпускной квалификационной работы.

Сформированный план и смета могут использоваться как ориентир при дальнейшем развитии системы: при добавлении новых сервисов или расширении функциональности можно оценивать влияние изменений на сроки, состав ресурсов и итоговую стоимость разработки.

## Заключение

В ходе выполнения выпускной квалификационной работы была разработана система управления фриланс-проектами, предназначенная для организации работы с проектами, задачами, участниками и сроками выполнения работ. Система реализована в виде набора микросервисов Users, Projects и Tasks, взаимодействующих через API Gateway и брокер сообщений.

В процессе проектирования были определены функциональные и нефункциональные требования, разработаны модели предметной области, спроектированы базы данных микросервисов, REST API и механизмы асинхронного обмена событиями. Для разделения операций чтения и изменения данных применены CQRS и MediatR, для надежной публикации событий используется outbox, а для ускорения часто выполняемых операций применено кэширование.

Практическая реализация подтвердила работоспособность основных пользовательских сценариев: регистрация и авторизация пользователей, создание и изменение проектов, добавление участников, создание задач, запуск выполнения задачи и добавление комментариев. Разработанные модульные и интеграционные тесты проверяют ключевые сценарии работы сервисов, маршрутизацию через API Gateway, сохранение и обработку событий.

Таким образом, поставленная цель работы достигнута: разработана масштабируемая и расширяемая система управления фриланс-проектами, соответствующая требованиям микросервисной архитектуры и пригодная для дальнейшего развития.

Основные термины и сокращения, используемые в пояснительной записке, приведены в приложении А.

Фрагменты программного кода серверной и клиентской частей приведены в приложениях Б и В.

Фрагменты модульных и интеграционных проверок серверной части приведены в приложениях Г и Д.

### Список использованных источников

1. Shmolenko V. FreelanceManager: исходный код программной системы [Электронный ресурс]. URL: <https://github.com/Omniphob1a/FreelanceManager> (дата обращения: 30.05.2026).
2. Microsoft. YARP Reverse Proxy documentation [Электронный ресурс]. URL: <https://learn.microsoft.com/aspnet/core/fundamentals/servers/yarp/yarp-overview> (дата обращения: 07.04.2026).
3. Apache Kafka documentation [Электронный ресурс]. URL: <https://kafka.apache.org/documentation/> (дата обращения: 09.04.2026).
4. Fowler M. Microservices [Электронный ресурс]. URL: <https://martinfowler.com/articles/microservices.html> (дата обращения: 25.04.2026).
5. PostgreSQL documentation [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения: 10.04.2026).
6. Microsoft. Entity Framework Core documentation [Электронный ресурс]. URL: <https://learn.microsoft.com/ef/core/> (дата обращения: 06.04.2026).
7. Microsoft. ASP.NET Core documentation [Электронный ресурс]. URL: <https://learn.microsoft.com/aspnet/core/> (дата обращения: 05.04.2026).
8. Swagger. OpenAPI Specification [Электронный ресурс]. URL: <https://swagger.io/specification/> (дата обращения: 14.04.2026).
9. MDN Web Docs. JavaScript [Электронный ресурс]. URL: <https://developer.mozilla.org/docs/Web/JavaScript> (дата обращения: 22.04.2026).
10. MDN Web Docs. HTML [Электронный ресурс]. URL: <https://developer.mozilla.org/docs/Web/HTML> (дата обращения: 23.04.2026).
11. MDN Web Docs. CSS [Электронный ресурс]. URL: <https://developer.mozilla.org/docs/Web/CSS> (дата обращения: 24.04.2026).
12. Redis documentation [Электронный ресурс]. URL: <https://redis.io/docs/> (дата обращения: 11.04.2026).
13. Hangfire documentation [Электронный ресурс]. URL: <https://docs.hangfire.io/> (дата обращения: 18.04.2026).

14. Fowler M. CQRS [Электронный ресурс]. URL: <https://martinfowler.com/bliki/CQRS.html> (дата обращения: 26.04.2026).
15. MediatR project repository [Электронный ресурс]. URL: <https://github.com/jbogard/MediatR> (дата обращения: 15.04.2026).
16. Docker documentation [Электронный ресурс]. URL: <https://docs.docker.com/> (дата обращения: 12.04.2026).
17. Docker Compose documentation [Электронный ресурс]. URL: <https://docs.docker.com/compose/> (дата обращения: 13.04.2026).
18. Microsoft. Authentication and authorization in ASP.NET Core [Электронный ресурс]. URL: <https://learn.microsoft.com/aspnet/core/security/authentication/> (дата обращения: 08.04.2026).
19. FluentValidation documentation [Электронный ресурс]. URL: <https://docs.fluentvalidation.net/> (дата обращения: 16.04.2026).
20. Mapster project repository [Электронный ресурс]. URL: <https://github.com/MapsterMapper/Mapster> (дата обращения: 17.04.2026).
21. Chart.js documentation [Электронный ресурс]. URL: <https://www.chartjs.org/docs/latest/> (дата обращения: 21.04.2026).
22. xUnit.net documentation [Электронный ресурс]. URL: <https://xunit.net/> (дата обращения: 19.04.2026).
23. Testcontainers for .NET documentation [Электронный ресурс]. URL: <https://dotnet.testcontainers.org/> (дата обращения: 20.04.2026).

**Приложение А**  
**(справочное)**  
**Глоссарий**

API – программный интерфейс приложения, используемый клиентской частью для обращения к серверным операциям.

CQRS – подход к разделению операций чтения и изменения данных.

DTO – объект передачи данных между слоями приложения или между клиентом и сервером.

JWT – формат токена доступа, применяемый для авторизации запросов пользователя.

Kafka – брокер сообщений, используемый для асинхронного межсервисного взаимодействия.

Outbox – шаблон сохранения исходящих событий в базе данных сервиса перед публикацией в брокер сообщений.

REST API – стиль организации HTTP-интерфейса для взаимодействия клиентской и серверной частей.

YARP – обратный прокси-сервер, используемый в системе как API Gateway.

**Приложение Б**  
**(обязательное)**  
**Листинг серверной части системы**

### Листинг Б.1 – Обработка запроса создания проекта в контроллере

```
[HttpPost]
[ProducesResponseType(typeof(Guid), StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> CreateProject(
    [FromBody] CreateProjectRequest request,
    CancellationToken cancellationToken)
{
    var command = request.Adapt<CreateProjectCommand>() with
    {
        OwnerId = _currentUserService.UserId
    };

    var result = await _mediator.Send(command, cancellationToken);

    if (result.IsFailed)
    {
        var errors = result.Errors.Select(error => error.Message);
        return BadRequest(new { errors });
    }

    return CreatedAtAction(
        nameof(GetProjectById),
        new { projectId = result.Value },
        result.Value);
}
```

### Листинг Б.2 – Выполнение команды создания проекта

```
public async Task<Result<Guid>> Handle(
    CreateProjectCommand request,
    CancellationToken cancellationToken)
{
    var tagsResult = _tagParserService.ParseTags(request.Tags);

    if (tagsResult.IsFailed)
        return Result.Fail<Guid>(tagsResult.Errors);

    Project project;

    try
    {
        project = Project.CreateDraft(
            request.Title,
            request.Description,
            request.OwnerId,
            new Budget(
                request.BudgetMin,
                request.BudgetMax,
```

```

        CurrencyCode.From(request.CurrencyCode)),
        Category.From(request.Category),
        tagsResult.Value,
        request.ConfirmedByUserId);
    }
    catch (DomainException exception)
    {
        return Result.Fail<Guid>(exception.Message);
    }

    await _repository.AddAsync(project, cancellationToken);
    _unitOfWork.TrackEntity(project);
    await _unitOfWork.SaveChangesAsync(cancellationToken);

    return Result.Ok(project.Id);
}

```

### Листинг Б.3 – Доменная логика изменения проекта

```

public void UpdateDetails(
    string title,
    string description,
    Budget budget,
    Category category,
    IEnumerable<Tag> tags)
{
    if (Status is ProjectStatus.Completed or ProjectStatus.Archived)
        throw new InvalidOperationException(
            "Completed or archived projects cannot be updated.");

    Validate(title, description, OwnerId, budget, category);

    Title = title;
    Description = description;
    Budget = budget;
    Category = category;

    _tags.Clear();
    _tags.AddRange(tags.Distinct());

    AddDomainEvent(new ProjectUpdatedDomainEvent(Id, Title,
        OwnerId));
}

```

### Листинг Б.4 – Публикация сообщений outbox

```

protected override async Task ExecuteAsync(Cancellation_token
    stoppingToken)
{
    while (!stoppingToken.IsCancellationRequested)

```

```

    {
        using var scope = _serviceProvider.CreateScope();
        var db =
scope.ServiceProvider.GetRequiredService<ProjectsDbContext>();

        var batch = await db.OutboxMessages
            .Where(message => !message.Processed)
            .OrderBy(message => message.OccurredAt)
            .Take(_batchSize)
            .ToListAsync(stoppingToken);

        foreach (var message in batch)
        {
            await _producer.ProduceAsync(
                message.Topic,
                message.Key,
                message.Payload,
                stoppingToken);

            message.Processed = true;
            message.ProcessedAt = DateTime.UtcNow;
            message.LastError = null;
        }

        await db.SaveChangesAsync(stoppingToken);
        await Task.Delay(_pollInterval, stoppingToken);
    }
}

```

**Приложение В**  
**(обязательное)**  
**Листинг клиентской части системы**

**Листинг В.1 – Отправка HTTP-запросов клиентского приложения**

```

export async function apiRequest(path, options = {}) {
  const token = localStorage.getItem("accessToken");

  const response = await fetch(`${API_BASE_URL}${path}`, {
    ...options,
    headers: {
      "Content-Type": "application/json",
      ...(token ? { Authorization: `Bearer ${token}` } : {}),
      ...(options.headers || {})
    }
  });

  if (!response.ok) {
    const error = await response.json().catch(() => ({}));
    throw new Error(error.message || "Ошибка выполнения
запроса");
  }

  if (response.status === 204) {
    return null;
  }

  return await response.json();
}

```

**Листинг В.2 – Загрузка и отображение списка проектов**

```

export async function loadProjects(filter = {}) {
  const query = new URLSearchParams(filter).toString();
  const projects = await apiRequest(`/projects?${query}`);

  const container = document.querySelector("[data-project-list]");
  container.innerHTML = "";

  projects.items.forEach(project => {
    const card = document.createElement("article");
    card.className = "project-card";
    card.dataset.projectId = project.id;
    card.innerHTML = `
      <span class="project-
card__status">${project.statusName}</span>
      <h3>${project.title}</h3>
      <p>${project.description}</p>
      <button data-open-project="${project.id}">
        Подробнее
      </button>`;

    container.append(card);
  });
}

```

```
    });
}
```

### Листинг В.3 – Отображение списка задач

```
export async function loadTasks(filter = {}) {
  const query = new URLSearchParams(filter).toString();
  const tasks = await apiRequest(`/projectTasks?${query}`);

  const tableBody = document.querySelector("[data-task-list]");
  tableBody.innerHTML = "";

  tasks.items.forEach(task => {
    const row = document.createElement("tr");
    row.innerHTML = `
      <td>${task.title}</td>
      <td>${formatDate(task.dueDate)}</td>
      <td>${task.priorityName}</td>
      <td>${task.statusName}</td>
      <td>
        <button data-open-task="${task.id}">
          Просмотр
        </button>
      </td>`;

    tableBody.append(row);
  });
}
```

**Приложение Г**  
**(рекомендуемое)**

**Листинг модульных проверок серверной части системы**

### Листинг Г.1 – Проверка создания проекта

```
[Fact]
public async Task Handle_ShouldCreateProject_WhenRequestIsValid()
{
    var command = new CreateProjectCommand(
        Title: "API интеграции платежей и договоров",
        Description: "Проектирование и разработка API",
        BudgetMin: 120000,
        BudgetMax: 190000,
        CurrencyCode: "RUB",
        Category: "Бэкенд",
        Tags: "api, платежи, договоры",
        OwnerId: _ownerId,
        ConfirmedByUserId: _ownerId);

    var result = await _handler.Handle(command,
        CancellationToken.None);

    result.IsSuccess.Should().BeTrue();
    _repository.Received(1).AddAsync(
        Arg.Any<Project>(),
        Arg.Any<CancellationToken>());
}
```

### Листинг Г.2 – Проверка перехода задачи в работу

```
[Fact]
public void
StartProgress_ShouldChangeTaskStatus_WhenTransitionIsAllowed()
{
    var task = ProjectTask.Create(
        projectId: _projectId,
        title: "Подготовить демонстрационные сценарии",
        description: "Собрать набор проектов и задач",
        reporterId: _reporterId,
        dueDate: DateTime.UtcNow.AddDays(2),
        priority: TaskPriority.High);

    task.Assign(_assigneeId);
    task.StartProgress(_assigneeId);

    task.Status.Should().Be(ProjectTaskStatus.InProgress);
    task.DomainEvents.Should()
        .ContainSingle(eventItem => eventItem is
TaskStatusChangedDomainEvent);
}
```

### Листинг Г.3 – Проверка добавления комментария к задаче

```
[Fact]
public void AddComment_ShouldAppendCommentToTask()
{
    var task = CreateTaskWithAssignee();

    task.AddComment(
        authorId: _assigneeId,
        text: "Добавлен сценарий демонстрации создания проекта.");

    task.Comments.Should().HaveCount(1);
    task.Comments.Single().Text.Should()
        .Be("Добавлен сценарий демонстрации создания проекта.");
}
```

**Приложение Д**  
**(рекомендуемое)**

**Листинг интеграционных проверок межсервисного взаимодействия**

### Листинг Д.1 – Проверка маршрутизации запроса через API Gateway

```
[Fact]
public async Task
Gateway_ShouldRouteProjectsRequestToProjectsService()
{
    await using var application = new GatewayTestApplication();
    var client = application.CreateClient();

    var response = await client.GetAsync("/api/projects");

    response.StatusCode.Should().NotBe(HttpStatusCode.NotFound);
    application.ProjectsServiceMock
        .ReceivedRequests
        .Should()
        .Contain(request => request.Path == "/api/projects");
}
```

### Листинг Д.2 – Проверка публикации outbox-сообщения

```
[Fact]
public async Task OutboxPublisher_ShouldPublishUnprocessedMessage()
{
    var message = OutboxMessage.Create(
        topic: "projects.events",
        key: _projectId.ToString(),
        payload: _payload);

    await _db.OutboxMessages.AddAsync(message);
    await _db.SaveChangesAsync();

    await _publisher.PublishBatchAsync(CancellationToken.None);

    var stored = await _db.OutboxMessages.SingleAsync();
    stored.Processed.Should().BeTrue();
    _kafkaProducer.PublishedMessages.Should()
        .Contain(item => item.Topic == "projects.events");
}
```

### Листинг Д.3 – Проверка идемпотентной обработки входящего события

```
[Fact]
public async Task IncomingEventHandler_ShouldNotDuplicateProjection()
{
    var incomingEvent = IncomingEvent.Create(
        eventId: _eventId,
        aggregateId: _projectId,
        aggregateType: "Project",
        eventType: "ProjectCreated",
        payload: _payload);
```

```
        await _handler.HandleAsync(incomingEvent,  
CancellationToken.None);  
        await _handler.HandleAsync(incomingEvent,  
CancellationToken.None);  
  
        var projections = await _db.Projects  
            .Where(project => project.Id == _projectId)  
            .ToListAsync();  
  
        projections.Should().HaveCount(1);  
    }
```

**Приложение Е**  
**(рекомендуемое)**  
**Результаты функционального тестирования**

В приложении приведены результаты функционального тестирования основных пользовательских сценариев: работы с пользователями, проектами и задачами.

The screenshot shows the 'Регистрация' (Registration) page of the FreelanceHub application. The left sidebar contains the FreelanceHub logo, a user status indicator ('Гость'), and navigation links for 'Панель', 'Проекты', 'Задачи', and 'Профиль'. The main content area is titled 'Создание аккаунта' (Account Creation) and features a registration form with the following fields: username (VladimirShmolenko), email (volodya2424@mail.ru), birth date (24.09.2004), gender (Мужской), and password fields. A blue 'Зарегистрироваться' (Register) button is at the bottom of the form, with a link 'Уже есть аккаунт? Войти' (Already have an account? Log in) below it.

Рисунок Е.1 – Результат функционального тестирования регистрации пользователя

The screenshot shows the 'Профиль' (Profile) page of the FreelanceHub application. The left sidebar is identical to the registration page but includes a 'Войти' (Log in) button at the bottom. The main content area is titled 'Профиль' and displays the user's account details: 'ВладимирШмоленько', login 'volodya2424', email 'volodya2424@mail.ru', gender 'Мужской', birth date '24 сент. 2004 г.', and a date range '8 октября с 9 мая 2026 г.'. A 'Редактировать' (Edit) button is located next to the profile information. Below this, there is a 'Безопасность' (Security) section with fields for 'Новый пароль' (New password) and 'Подтверждение' (Confirmation), followed by a 'Сменить пароль' (Change password) button.

Рисунок Е.2 – Результат функционального тестирования авторизации пользователя

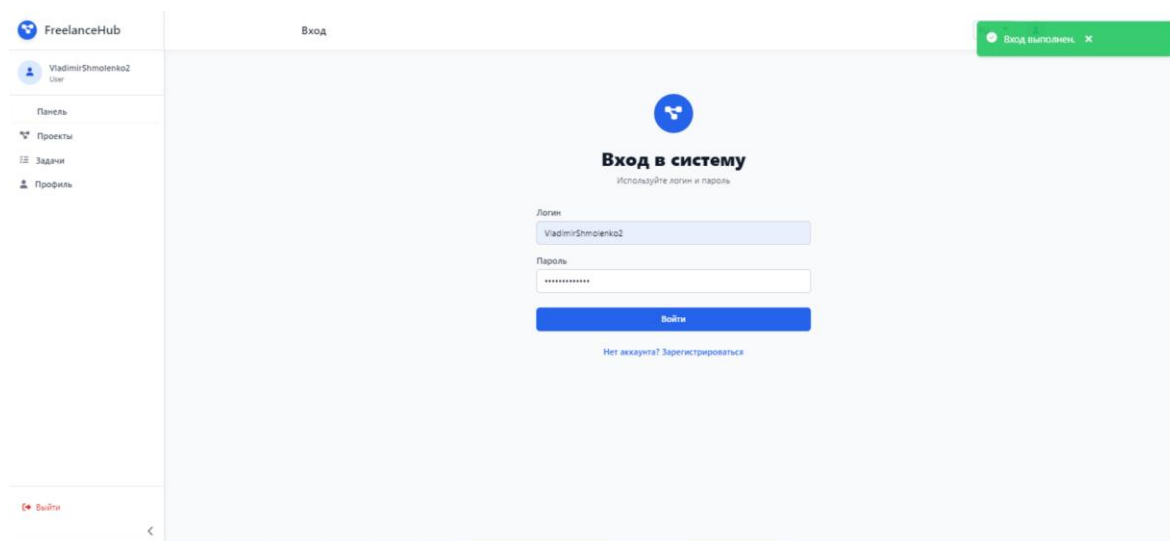


Рисунок Е.3 – Результат функционального тестирования просмотра профиля пользователя

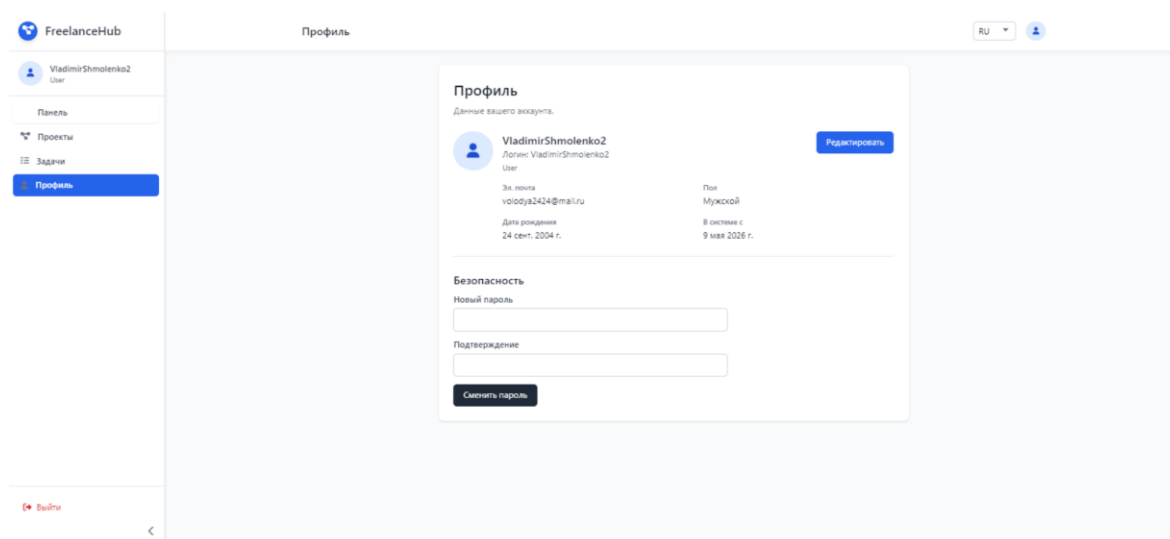


Рисунок Е.4 – Результат функционального тестирования изменения профиля пользователя

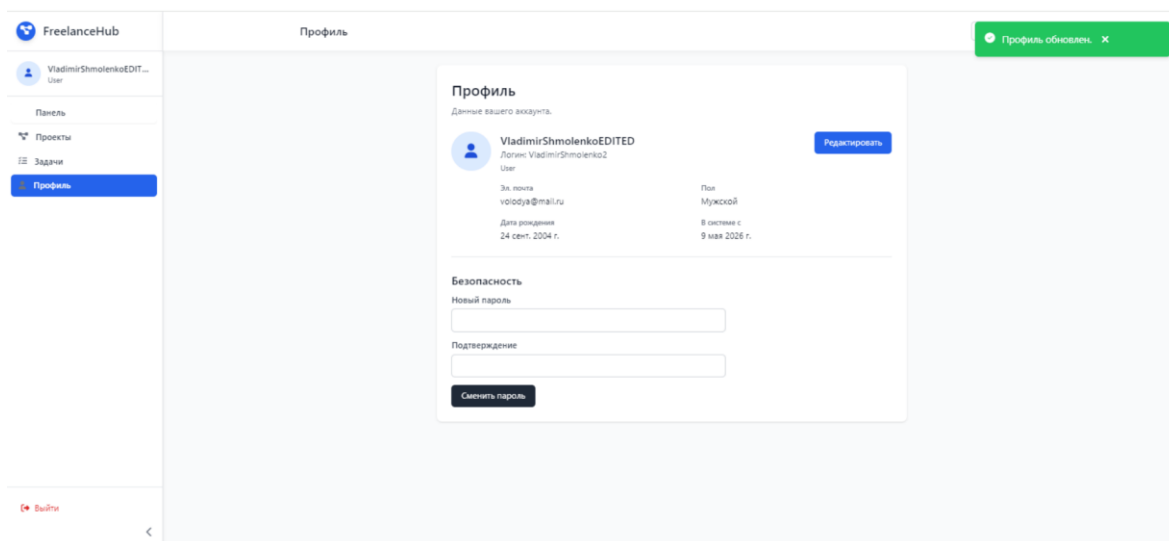


Рисунок Е.5 – Результат функционального тестирования авторизации с неверным паролем

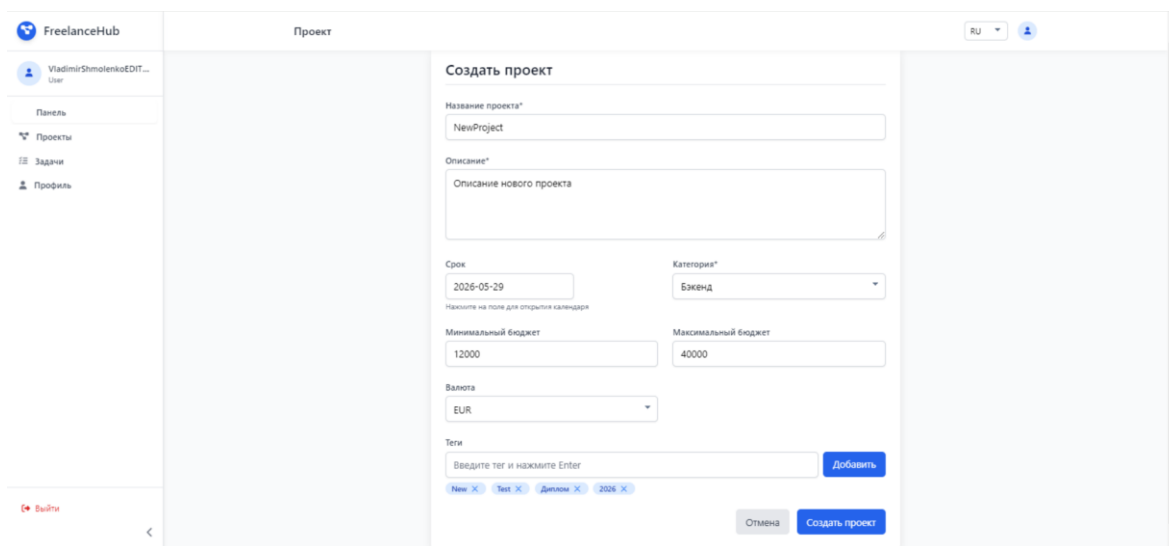


Рисунок Е.6 – Результат функционального тестирования создания проекта

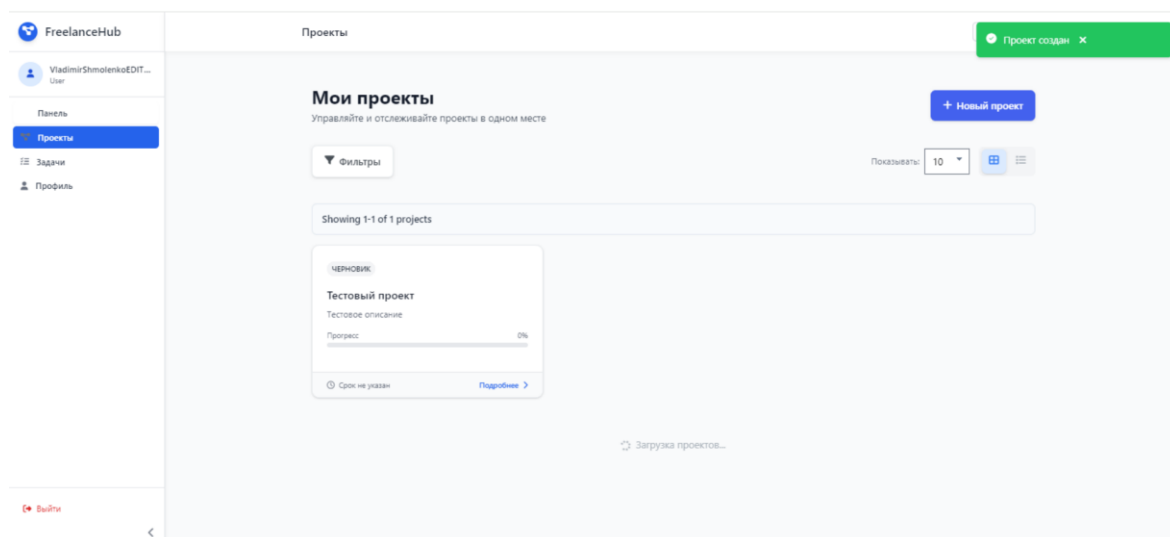


Рисунок Е.7 – Результат функционального тестирования просмотра проекта

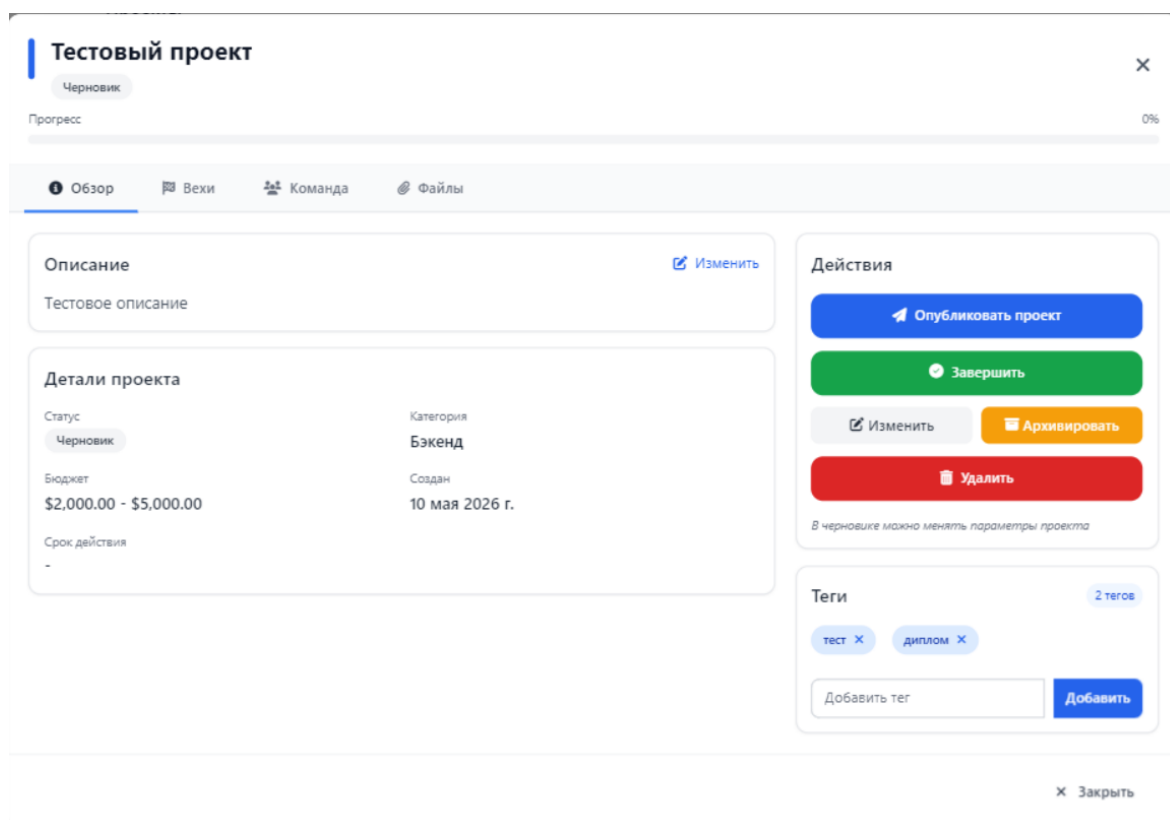


Рисунок Е.8 – Результат функционального тестирования изменения проекта

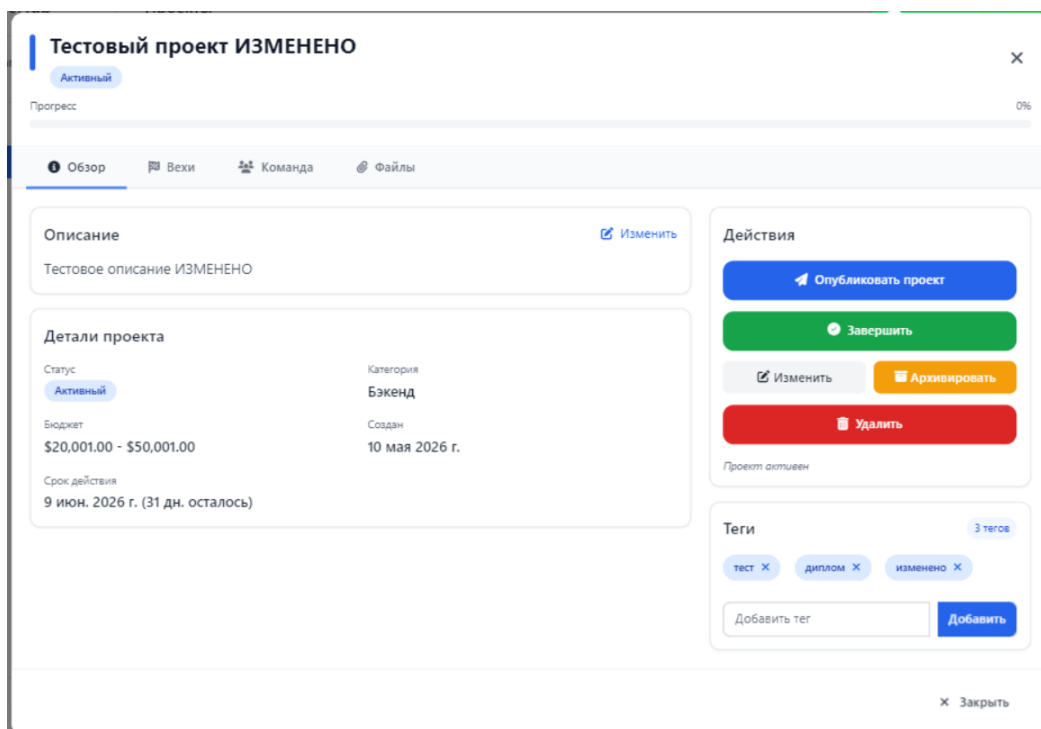


Рисунок Е.9 – Результат функционального тестирования добавления участника проекта

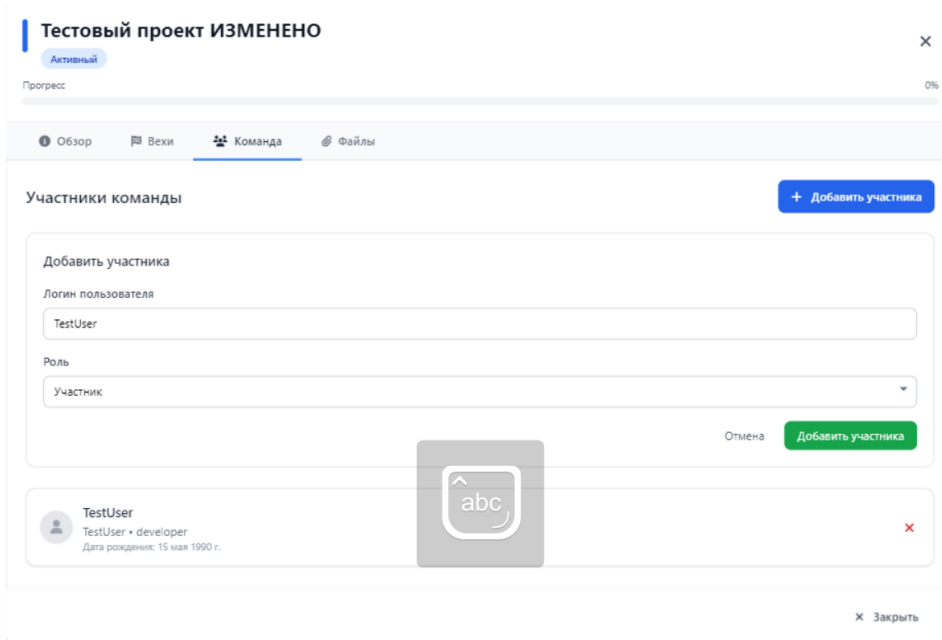


Рисунок Е.10 – Результат функционального тестирования добавления несуществующего участника

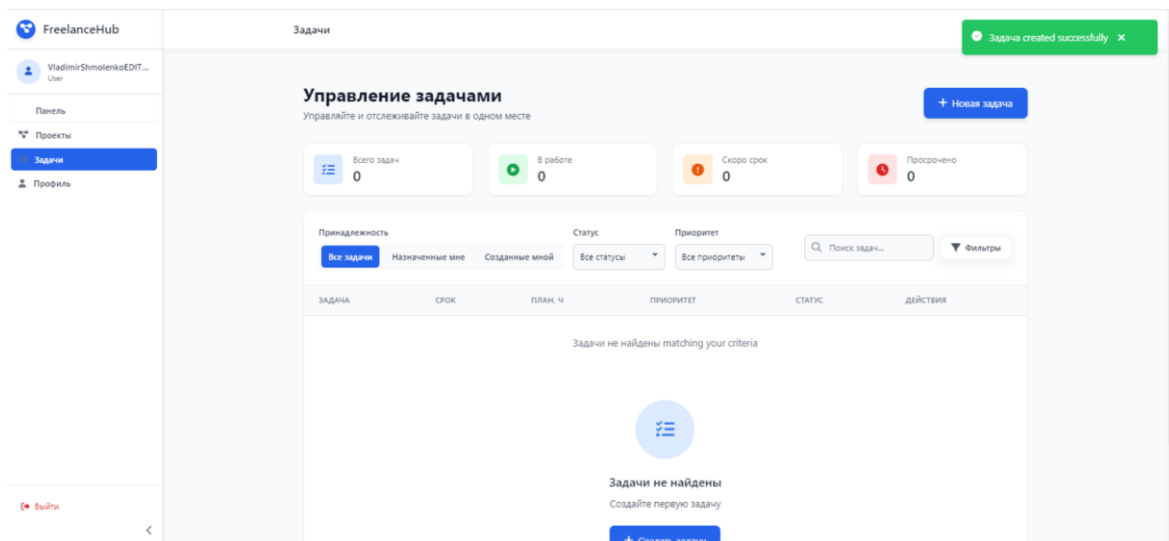


Рисунок Е.11 – Результат функционального тестирования просмотра задачи

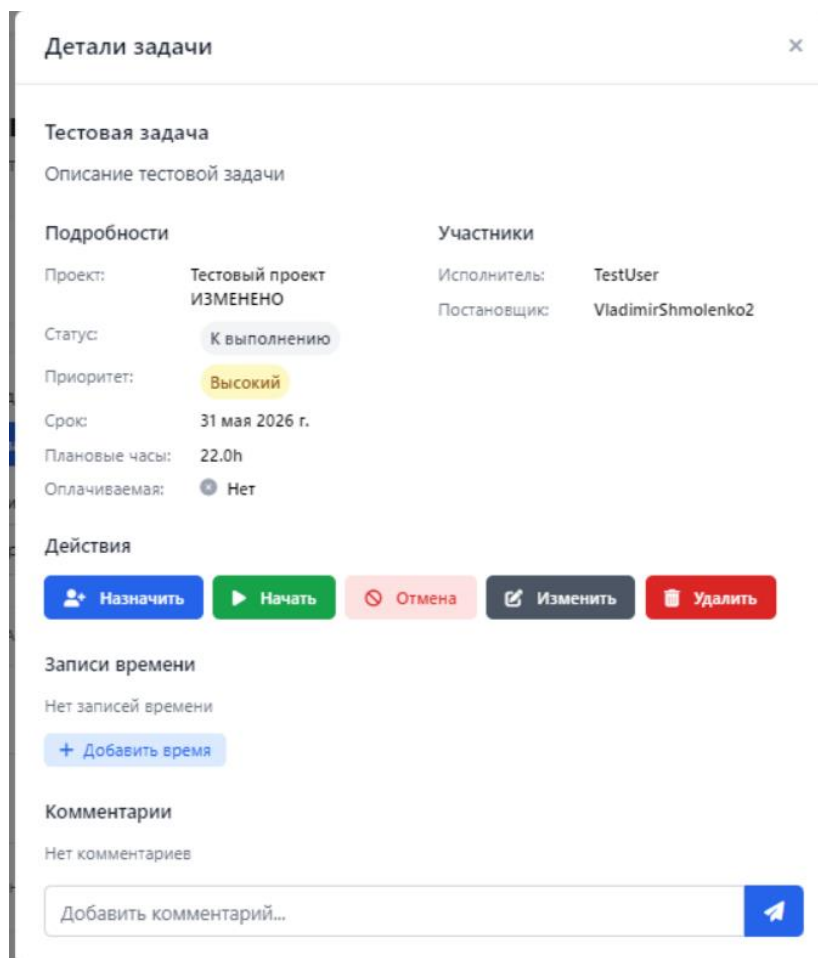


Рисунок Е.12 – Результат функционального тестирования назначения исполнителя задачи

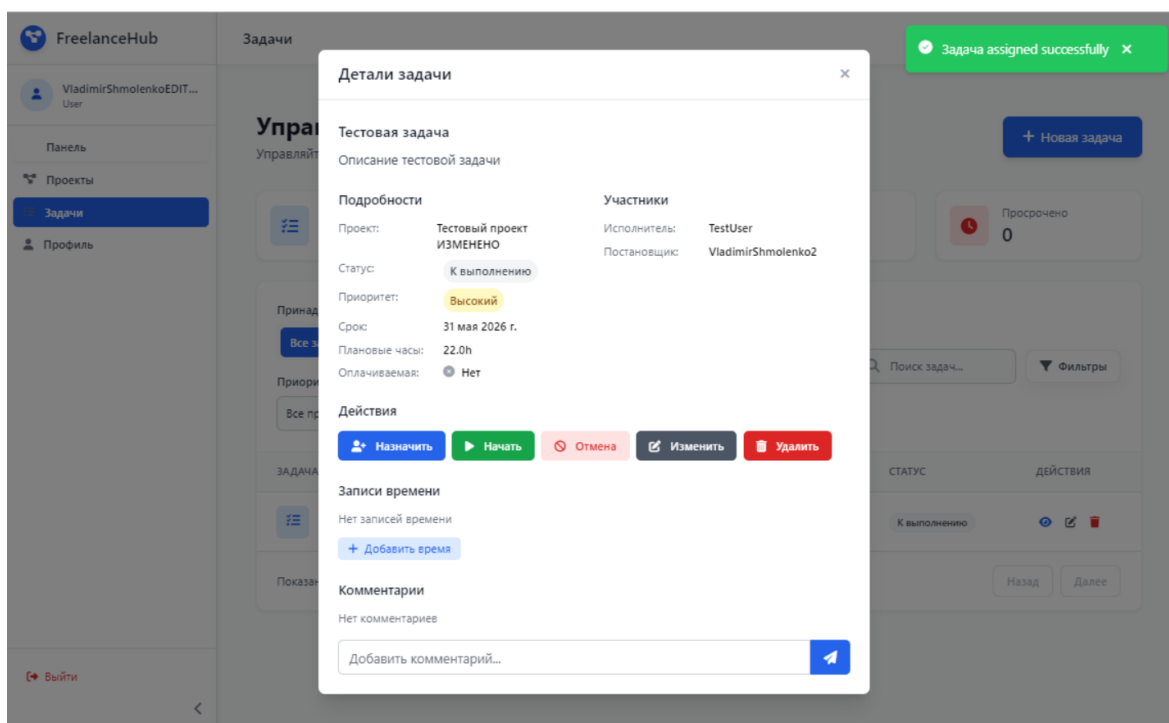


Рисунок Е.13 – Результат функционального тестирования запуска выполнения задачи

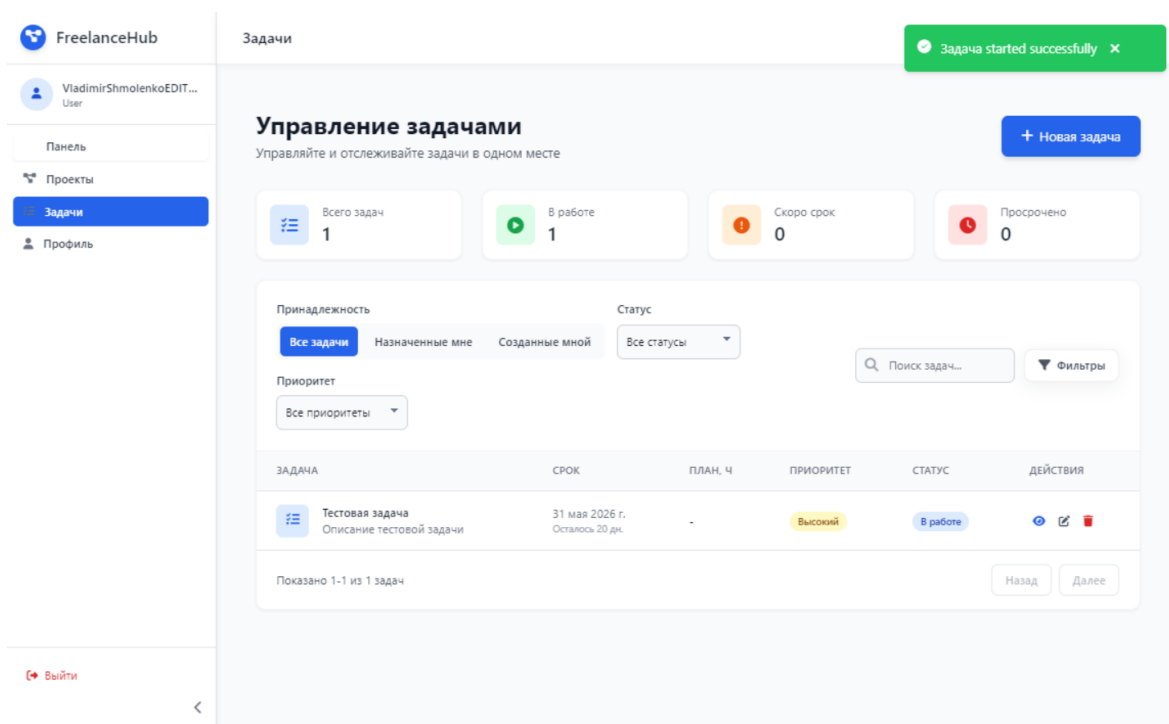


Рисунок Е.14 – Результат функционального тестирования завершения задачи

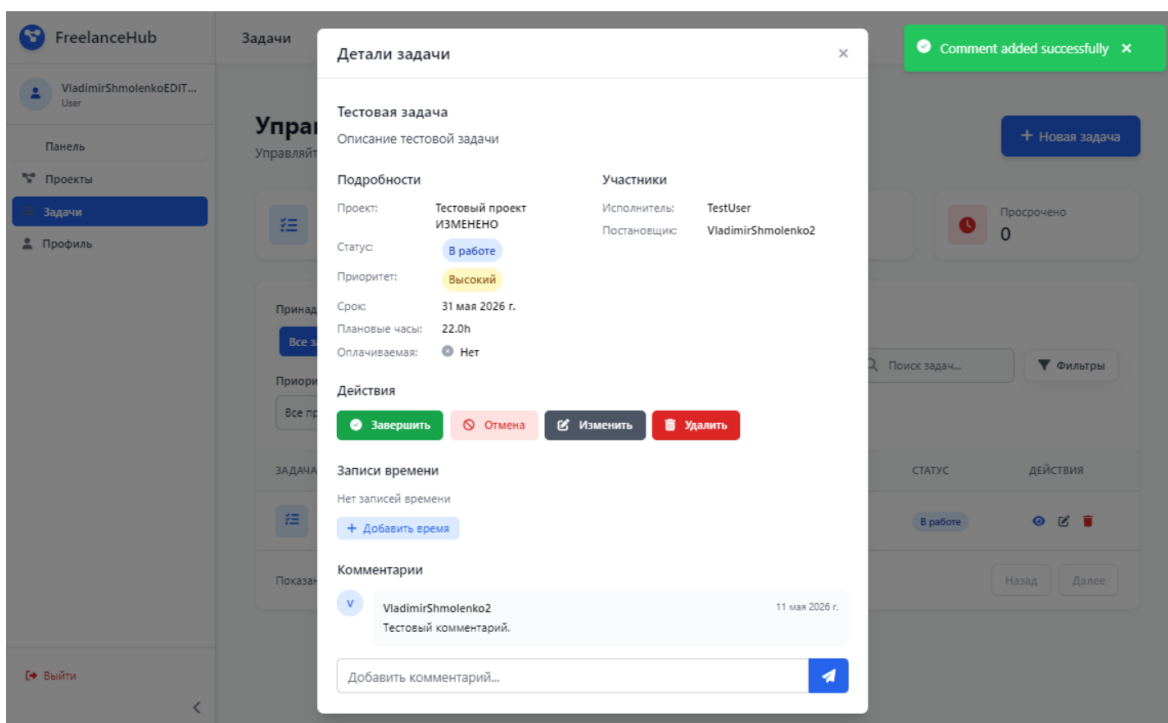


Рисунок Е.15 – Результат функционального тестирования добавления комментария

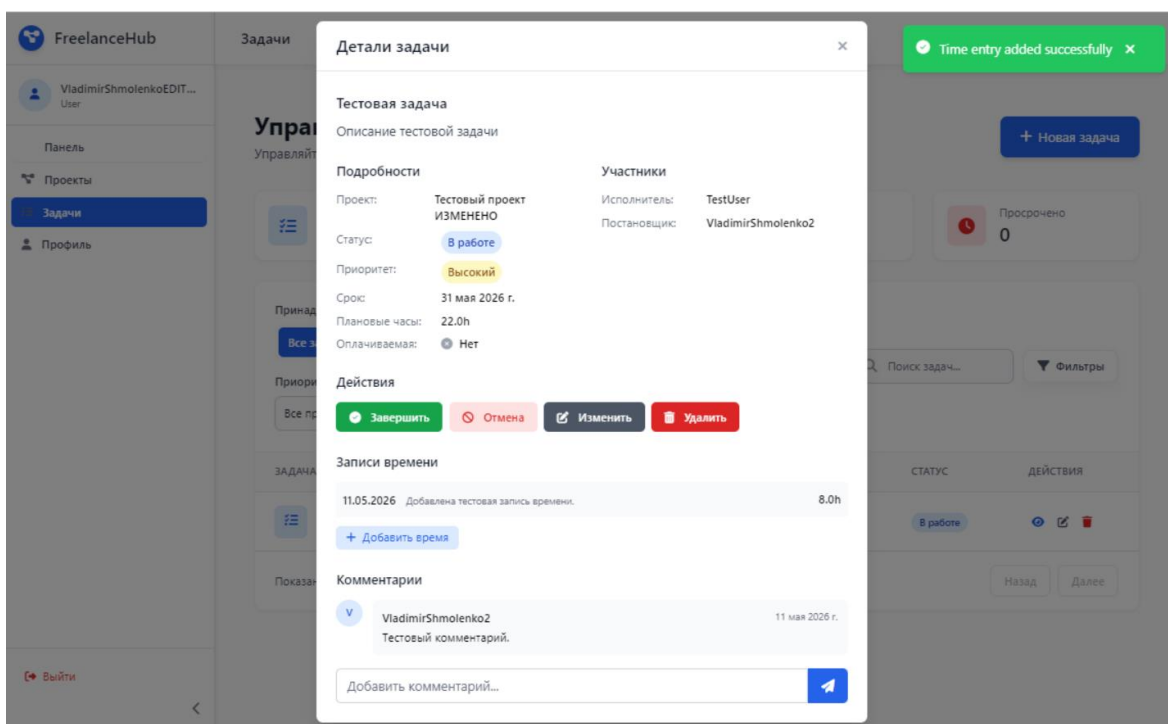


Рисунок Е.16 – Результат функционального тестирования добавления записи времени

| Принадлежность                                                                                                                |                                   | Статус          |           |                |                                                                                                                                                                                                                                                             |
|-------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|-----------------|-----------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Все задачи                                                                                                                    |                                   | Назначенные мне |           | Созданные мной |                                                                                                                                                                                                                                                             |
|                                                                                                                               |                                   | Все статусы     |           |                |                                                                                                                                                                                                                                                             |
| Приоритет                                                                                                                     |                                   | Поиск задач...  |           | Фильтры        |                                                                                                                                                                                                                                                             |
| Все приоритеты                                                                                                                |                                   |                 |           |                |                                                                                                                                                                                                                                                             |
| ЗАДАЧА                                                                                                                        | СРОК                              | ПЛАН, Ч         | ПРИОРИТЕТ | СТАТУС         | ДЕЙСТВИЯ                                                                                                                                                                                                                                                    |
|  Тестовая задача<br>Описание тестовой задачи | 31 мая 2026 г.<br>Осталось 20 дн. | -               | Высокий   | Завершена      |    |

Рисунок Е.17 – Результат функционального тестирования недопустимого перехода статуса задачи

**Приложение Ж**  
**(рекомендуемое)**

**Результаты модульного и интеграционного тестирования**

В приложении приведены результаты запуска модульных и интеграционных проверок серверной части системы в среде Visual Studio 2022.

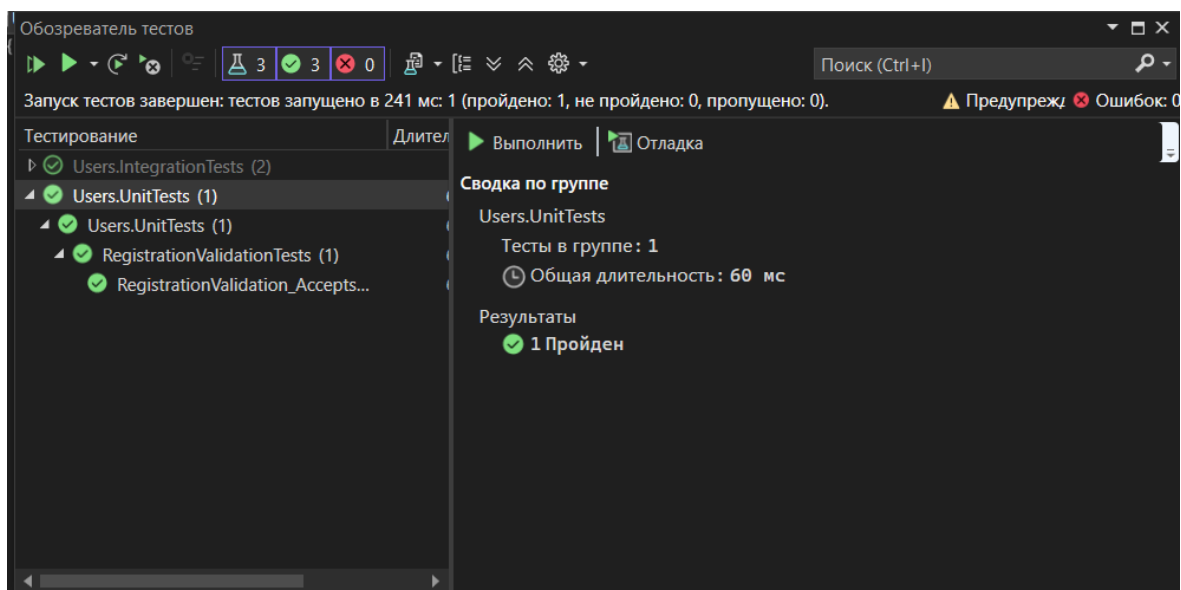


Рисунок Ж.1 – Результаты интеграционного тестирования сервиса пользователей

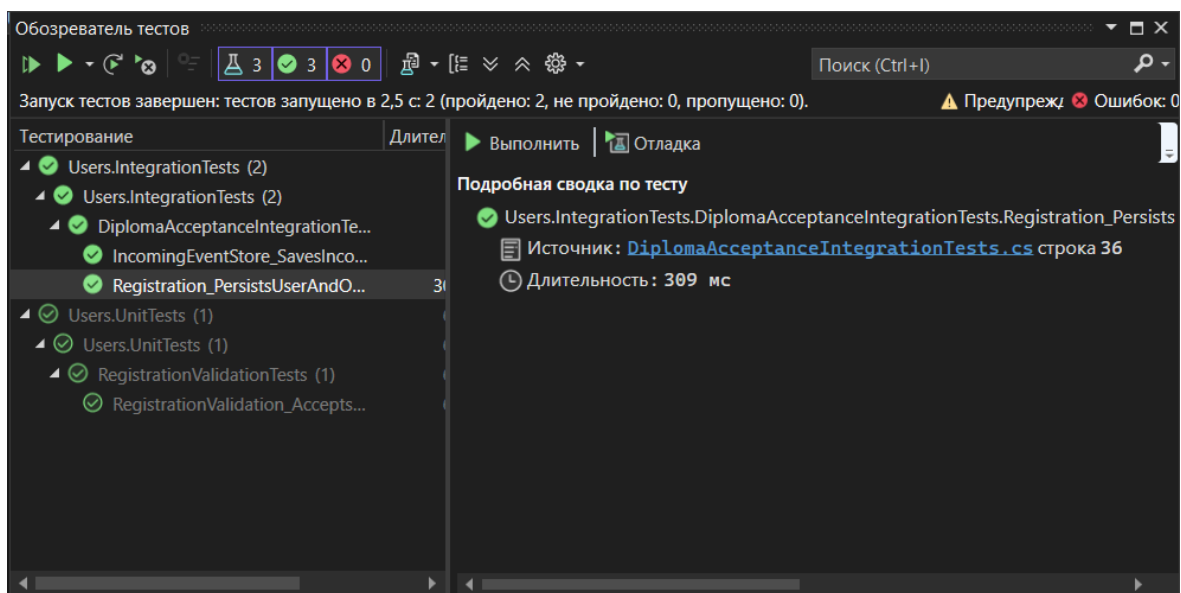


Рисунок Ж.2 – Результаты интеграционного тестирования сервиса проектов

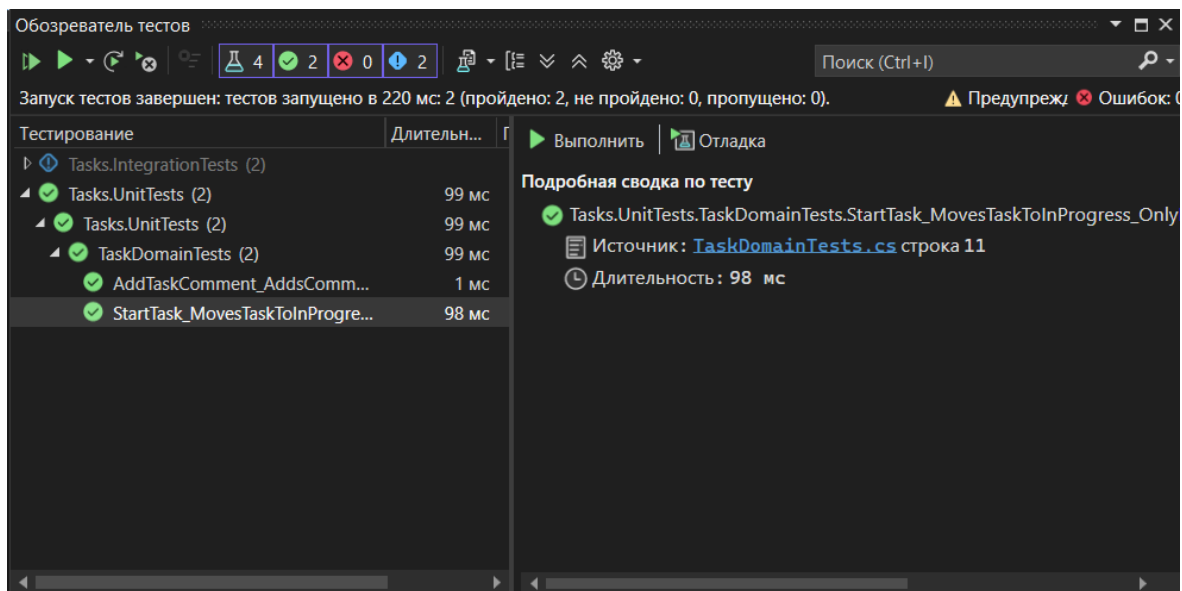


Рисунок Ж.3 – Результаты интеграционного тестирования сервиса задач

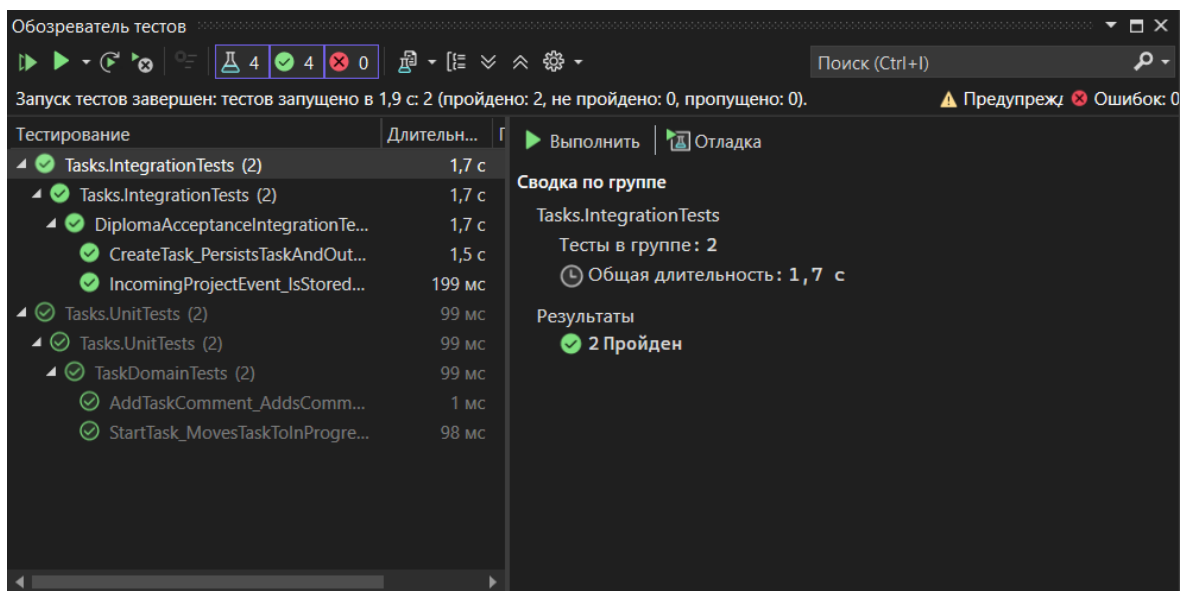


Рисунок Ж.4 – Результаты модульного тестирования сервиса пользователей

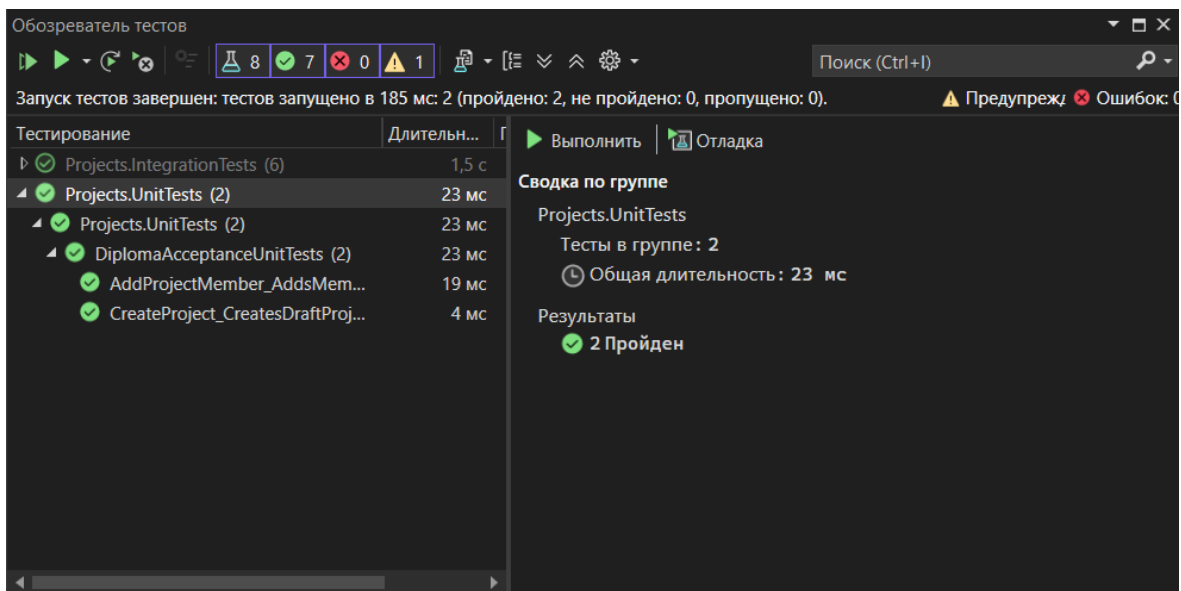


Рисунок Ж.5 – Результаты модульного тестирования сервиса проектов

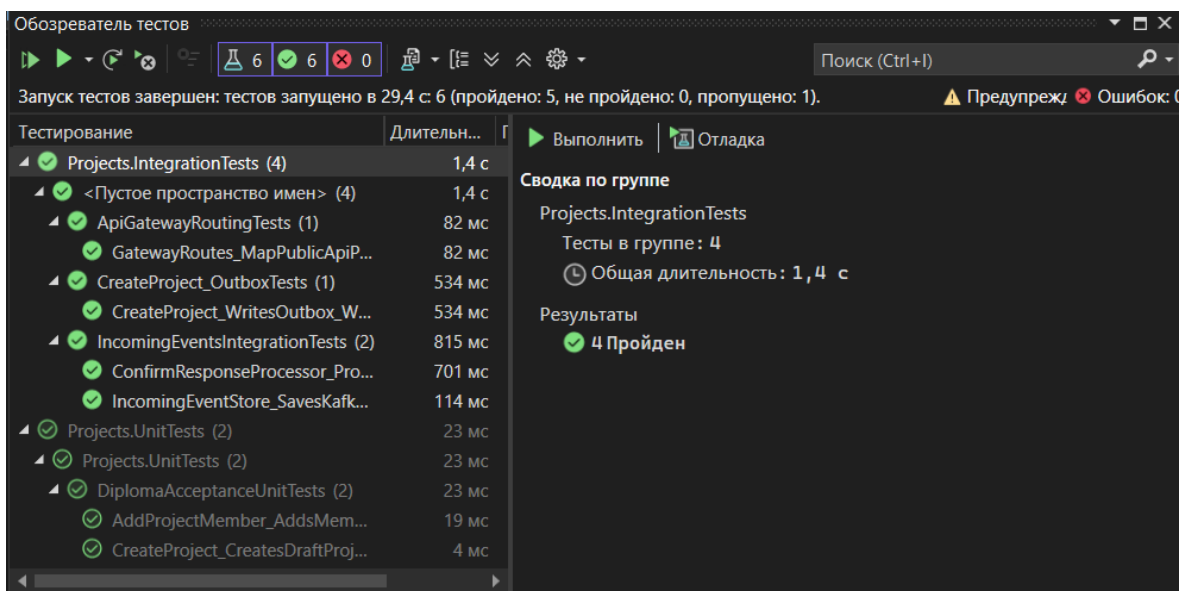


Рисунок Ж.6 – Результаты модульного тестирования сервиса задач