СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc195720146)

[ОСНОВНАЯ ЧАСТЬ 5](#_Toc195720147)

[1. Исследовательский раздел 5](#_Toc195720148)

[1.1 Анализ существующих решений 5](#_Toc195720149)

[1.2 Определение требований к решению 6](#_Toc195720150)

[1.2.1 Формулирование требований к программной части системы 6](#_Toc195720151)

[1.2.2 Формулирование требований к аппаратной части системы 8](#_Toc195720153)

[1.3 Выбор инструментов и методов создания приложения 8](#_Toc195720154)

[1.4 Выбор методологии и технологий концептуального моделирования клиент-серверной системы 10](#_Toc195720155)

[1.5 Постановка задач к проектированию и разработке приложения 11](#_Toc195720156)

[Вывод к разделу 1 12](#_Toc195720157)

[2 Проектный раздел 13](#_Toc195720158)

[2.1 Проектирование адаптированной модели жизненного цикла системы 13](#_Toc195720159)

[2.2 Проектирование архитектуры системы 13](#_Toc195720160)

[2.2. Проектирование основного модуля 16](#_Toc195720161)

[2.3. Проектирование клиентской части системы 18](#_Toc195720162)

[2.4. Проектирование схемы базы данных 19](#_Toc195720163)

[Вывод к разделу 2 21](#_Toc195720164)

[3 Технологический раздел 22](#_Toc195720165)

[3.1 Разработка серверной части системы 22](#_Toc195720166)

[3.2 Разработка пользовательского интерфейса системы 29](#_Toc195720171)

[3.3 Тестирование веб-приложения 31](#_Toc195720172)

[Вывод к разделу 3 32](#_Toc195720173)

[ЗАКЛЮЧЕНИЕ 34](#_Toc195720174)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 35](#_Toc195720175)

[Приложение А 37](#_Toc195720176)

[Приложение Б 40](#_Toc195720178)

# ВВЕДЕНИЕ

Организация групповых мероприятий всегда сопровождается с множеством сложностей. Разделение обязанностей, учет бюджета, согласование задач - все это требует четкой координации, особенно если участников много. В условиях недостатка прозрачности и эффективных инструментов организации возникают финансовые потери, путаница в обязанностях, несогласованность действий. В результате мероприятие может оказаться менее эффективным, чем планировалось, а его участники - недовольными.

Согласно исследовании Mordor Intelligence [1], рынок программного обеспечения для управления мероприятиями стремительно растет: его объем достигнет 13,01 млрд долларов США в 2024 году, а среднегодовой темп роста составит 10,24%. Это подтверждает высокий спрос на цифровые решения в данной сфере. Причина этого проста: традиционные методы организации, основанные на чатах, таблицах и устных договоренностях, не всегда обеспечивают необходимую оперативность и четкость взаимодействия. Чем больше людей вовлечено в процесс, тем сложнее контролировать все аспекты подготовки.

Одной из ключевых проблем является бюджетирование. Без единого инструмента для контроля расходов случаются перерасходы, неравномерное распределение затрат и недоразумения между участниками. Еще одна распространенная трудность - отслеживание задач. Когда нет системы, напоминающей о дедлайнах и статусе выполнения, часть работы может оказаться забытой, что приведет к срывам сроков и ухудшению качества организации.

Современные веб-приложения решают эти проблемы, предлагая централизованные решения для планирования, коммуникации и финансового управления. Они позволяют участникам и организаторам оперативно вносить изменения, отслеживать ход выполнения задач и справедливо распределять бюджет. Это делает процесс не только более прозрачным, но и значительно снижает вероятность ошибок.

Объектом исследования данной работы является процесс управления групповыми мероприятиями, включая координацию участников, управление задачами и финансовыми аспектами.

Предметом исследования выступают методы, инструменты и алгоритмы разработки информационных систем для управления групповыми мероприятиями, обеспечивающие интеграцию функций планирования, распределения бюджета и отслеживания задач.

Целью работы является разработка веб-приложения для управления групповыми мероприятиями с функциями разделения бюджета, отслеживания задач и уведомлений, а также проектирование его архитектуры с применением современных технологий и методологий.

Для выполнения практической работы необходимо решить следующие

задачи:

* провести анализ предметной области, в том числе конкурентных решений;
* определить информационные процессы предметной области и формализовать их;
* спроектировать архитектуру системы;
* разработать веб-приложение для управления мероприятиями с функциями разделения бюджета и отслеживанием задач;
* реализовать модуль с применением технологий виртуализации и механизмов восстановления после сбоев;
* оформить пояснительную записку согласно ГОСТ 7.32-2017 [2];
* подготовить презентацию по результатам практики.

Отчет состоит из введения, 3 разделов, заключения, списка использованных источников.

# ОСНОВНАЯ ЧАСТЬ

1. Исследовательский раздел

1.1 Анализ существующих решений

Существует множество инструментов, которые помогают организовать групповые мероприятия, но большинство из них решают лишь отдельные аспекты этой задачи. Рассмотрим три популярных приложения, которые используются для управления процессами, связанными с координацией участников, распределением задач и учетом финансов.

**Trello** [3] - это визуальный инструмент для управления задачами, основанный на методологии Kanban. Он позволяет создавать доски с колонками и карточками, в которых можно прописывать задачи, добавлять комментарии, прикреплять файлы и назначать ответственных. Trello активно используется командами для планирования мероприятий, однако его основной недостаток - отсутствие встроенного механизма для учета бюджета и финансовых обязательств участников.

**Splitwise** [4] - это приложение, предназначенное для удобного разделения расходов в группах. Оно автоматически рассчитывает, кто кому и сколько должен, что особенно полезно при совместных мероприятиях. Однако, несмотря на удобство работы с финансами, Splitwise не поддерживает функционал для управления задачами и координации участников. Это делает его ограниченным в использовании при организации сложных мероприятий.

**Asana** [5] - это мощный таск-менеджер, предназначенный для управления проектами. Он позволяет создавать задачи, назначать ответственных, устанавливать дедлайны и отслеживать прогресс выполнения. Asana широко используется в корпоративной среде и для сложных мероприятий, но не имеет встроенной системы учета финансов, что делает её менее удобной для случаев, когда требуется управление общим бюджетом.

Заполним таблицу, в которой сравним характеристики рассмотренных сервисов (Таблица 1.1).

Таблица 1.1 – Сравнение аналогов (разработано автором)

|  |  |  |  |
| --- | --- | --- | --- |
| **Функции** | **Trello** | **Splitwise** | **Asana** |
| Управление задачами | + | - | + |
| Разделение бюджета | - | + | - |
| Координация участников | + | - | + |
| Интуитивность интерфейса | + | + | - |
| Гибкость в настройке | + | - | + |
| Интеграции с другими сервисами | + | + | + |
| Поддержка командной работы | + | + | + |
| Простота освоения | + | + | - |

Каждое из рассмотренных приложений хорошо справляется со своей узкой задачей, но не решает проблему комплексного управления групповыми мероприятиями. Trello и Asana удобны для распределения задач, но не имеют встроенного механизма для учета финансов. Splitwise помогает отслеживать расходы, но не поддерживает координацию действий участников. Это подтверждает необходимость создания специализированного решения, объединяющего управление задачами, распределение бюджета и координацию команды в одном интерфейсе.

1.2 Определение требований к решению

1.2.1 Формулирование требований к программной части системы

Функциональные требования

Система должна обеспечивать пользователю возможность организации и управления групповыми мероприятиями, включая координацию участников, управление задачами и учет финансов.

Необходимо реализовать:

* управление мероприятиями: создание, изменение, удаление и просмотр мероприятий;
* управление задачами: назначение задач участникам, установка дедлайнов, отслеживание статуса выполнения;
* разделение бюджета: расчет общей суммы затрат, автоматический расчет долей участников, просмотр истории платежей;
* комментарии к задачам: обмен комментариями между участниками для координации действий;
* напоминания и уведомления: система напоминаний о задачах, платежах и других событиях мероприятия.

Указанные функциональные требования определены на основе анализа предметной области и существующих решений, которые показали недостатки в комплексном управлении мероприятиями.

**Нефункциональные требования**

Основные метрики, которые необходимо обеспечить:

Производительность: время отклика сервера не более 1 секунды при одновременной работе 50 пользователей;

Доступность: система должна быть доступна не менее 99,5% времени в течение месяца;

Безопасность: шифрование данных пользователей;

Кроссплатформенность: система должна запускаться в Docker’е [6] для поддержки различных операционных систем;

Масштабируемость: поддержка увеличения числа пользователей без снижения производительности;

Логирование и мониторинг: сбор логов работы системы и ошибок с возможностью анализа через централизованную систему логирования.

Нефункциональные требования сформулированы с учётом специфики разрабатываемого приложения: веб-приложение для управления мероприятиями предполагает работу в условиях распределённой команды, что требует высокой доступности, производительности и безопасности.

1.2.2 Формулирование требований к аппаратной части системы

Для серверной части системы выдвигаются следующие минимальные требования к аппаратной части:

* процессор разрядности 64 бит;
* 8 ГБ оперативной памяти;
* пропускная способность сетевого соединения: 100 Мбит/c;
* 1 CPU;
* SSD или HDD от 256 GB.

Данные требования определены с учётом особенностей разрабатываемого приложения: микросервисная архитектура с использованием Docker и PostgreSQL требует достаточного объёма оперативной памяти и дискового пространства. Указанные параметры обеспечивают минимально необходимую производительность для тестовой эксплуатации системы с возможностью масштабирования в будущем.

1.3 Выбор инструментов и методов создания приложения

Для разработки веб-приложения с функциями управления групповыми мероприятиями, разделения бюджета и отслеживания задач была выбрана клиент-серверная архитектура. Основная обработка данных в данной архитектуре выполняется на серверной стороне. Это потребовало внимательного подбора технологий для серверной и клиентской частей, а также системы управления базами данных (СУБД). Ниже приведено обоснование выбора конкретного стека технологий.

Для серверной части было решено использовать язык программирования Go (Golang). Выбор обусловлен его высокой производительностью, встроенной поддержкой многозадачности и простотой разработки. Go компилируется в машинный код, что обеспечивает низкие задержки при обработке запросов, а эффективное управление памятью делает его менее требовательным к ресурсам по сравнению с Java. В отличие от Node.js, Go лучше справляется с многопоточностью, что важно для приложений с высокой нагрузкой. Кроме того, строгая типизация снижает вероятность ошибок и упрощает поддержку кода. В таблице 1.2 проведено сравнение языка Go [7], Java [8] и Node.js [9].

Таблица 1.2 – Сравнение языков серверной части (разработано автором)

|  |  |  |  |
| --- | --- | --- | --- |
| **Характеристика** | **Go** | **Java** | **Node.js** |
| Производительность | Хорошо | Хорошо | Средне |
| Масштабируемость | Хорошо | Хорошо | Средне |
| Простота освоения | Средне | Плохо | Хорошо |
| Требовательность к ресурсам | Хорошо | Плохо | Хорошо |
| Безопасность | Хорошо | Хорошо | Средне |
| Асинхронность | Хорошо | Хорошо | Средне |

На клиентской стороне было принято решение использовать библиотеку React вместе с языком JavaScript. React позволяет создавать динамичные интерфейсы благодаря компонентной архитектуре и виртуальному DOM, что ускоряет рендеринг страниц. JavaScript обеспечивает универсальность, так как поддерживается всеми браузерами и легко интегрируется с REST API. По сравнению с Angular, React проще в освоении и более гибок, а по сравнению с Vue.js он выигрывает за счёт более развитой экосистемы и поддержки сообщества. В таблице 1.3 проведено сравнение React [10], Angular [11], Vue.js [12].

Таблица 1.3 – Сравнение фреймоворков для клиентской части (разработано автором)

|  |  |  |  |
| --- | --- | --- | --- |
| **Характеристика** | **React** | **Angular** | **Vue.js** |
| Производительность | Хорошо | Средне | Хорошо |
| Популярность | Хорошо | Средне | Средне |
| Гибкость | Хорошо | Плохо | Хорошо |
| Сложность изучения | Средне | Плохо | Хорошо |

Система управления базами данных (СУБД) играет ключевую роль в обработке и хранении информации, влияя на производительность и безопасность приложения.

Для хранения данных была выбрана реляционная СУБД PostgreSQL [13]. Её использование гарантирует надёжность, поддержку сложных запросов и транзакций, а также гибкость работы с данными благодаря поддержке формата JSON. PostgreSQL легко масштабируется, что важно для дальнейшего развития приложения.

Для изоляции окружения и упрощения процесса развёртывания была применена технология Docker. Контейнеризация помогает стандартизировать разработку и деплой, что особенно удобно для микросервисной архитектуры.

Для серверной части была использована среда Goland, оптимизированная для работы с Go. Для клиентской части было решено использовать Visual Studio Code – лёгкий и гибкий редактор с поддержкой расширений для React и JavaScript.

1.4 Выбор методологии и технологий концептуального моделирования клиент-серверной системы

К числу наиболее распространенных методологий моделирования бизнес-процессов относятся:

* методологии структурного анализа и проектирования (SADT): IDEF0, DFD, IDEF3;
* методология объектно-ориентированного анализа и проектирования, основанного на стандартах языка UML [14];
* методологии, основанной на стандартах нотации BPMN [15], и др.

В рамках данной работы наиболее целесообразно использование методологии структурного анализа, поскольку:

* данная методология позволяет моделировать как верхнеуровневые процессы, так и достичь необходимой степени детализации;
* моделируется иерархия бизнес-процессов;
* позволяет моделировать потоки данных [16].

Программные средства, поддерживающие моделирование в требуемых нотациях: Ramus, Microsoft Visio, ERwin Process Modeler. Основными требованиями к инструменту является: − кроссплатформенность;

* поддержка нотаций IDEF0, DFD, IDEF3;
* автоматизация рутинных процессов при построении (автоматическая декомпозиция процесса и т.д.).

С учётом представленных выше требований был выбран редактор Ramus для построения диаграмм в нотации IDEF0 из-за его основного и главного преимущества в кроссплатформенности. Кроме того, для моделирования взаимодействия объектов системы применена методология UML, а для создания текстовых диаграмм (ER, компонентов, классов) использован Mermaid [17], который обеспечивает простоту и интеграцию с современными инструментами документирования.

1.5 Постановка задач к проектированию и разработке приложения

Исходя из требований, необходимо поставить задачи на проектирование и разработку.

Задачи на проектирование:

* спроектировать архитектуру распределенной системы;
* спроектировать клиентскую часть;
* спроектировать серверную часть;
* спроектировать модуль работы с комментариями;
* спроектировать модуль уведомлений;
* спроектировать схему базы данных.

Задачи на разработку:

* разработать клиентскую часть;
* разработать серверную часть;
* разработать модуль работы с комментариями;
* разработать модуль уведомлений;
* реализовать схему базы данных.

Вывод к разделу 1

В данной главе проанализированы существующие подходы к созданию систем для управления групповыми мероприятиями. Выявлено, что в настоящее время в интернете существует множество аналогов таких систем, однако ни одна из них не сочетает в полной мере функции разделения бюджета и отслеживания задач в рамках единого приложения, ориентированного на групповую работу.

Изучены возможные подходы к разработке приложения для управления групповыми мероприятиями, определены их преимущества и недостатки. В результате анализа выявлена потребность в создании системы, которая интегрирует управление мероприятиями, разделение бюджета и отслеживание задач в одном решении. Были сформированы требования к системе и поставлена задача на её разработку.

2 Проектный раздел

2.1 Проектирование адаптированной модели жизненного цикла системы

Разработка веб-приложения для управления групповыми мероприятиями с функциями разделения бюджета и отслеживания задач требует выбора модели жизненного цикла, обеспечивающей эффективность процесса создания системы. После анализа существующих подходов была выбрана итеративная модель [18], которая позволяет поэтапно реализовывать функциональность, минимизировать риски и адаптироваться к изменениям требований.

Сравнение каскадной модели [18], итеративной модели и гибких методологий показало, что каскадный подход слишком жёсткий и не допускает изменений на поздних стадиях, а гибкие методологии, такие как Agile [19], избыточны по сложности для данного проекта. Итеративная модель оказалась оптимальной благодаря умеренной гибкости, раннему выявлению ошибок через тестирование после каждой итерации и управляемости процесса. Такой выбор соответствует задачам проекта, связанным с постепенным внедрением сложных функций и необходимостью проверки промежуточных результатов.

Адаптированная итеративная модель включает планирование, где определяются требования и выбираются технологии, проектирование с разработкой архитектуры и прототипов, итеративную разработку с поэтапным созданием функциональности, тестирование для проверки работоспособности и развёртывание для финальной подготовки системы. Этот подход обеспечивает структурированность, возможность корректировки и снижение рисков, что делает его подходящим для создания системы с уникальным сочетанием функций управления мероприятиями, бюджетом и задачами.

2.2 Проектирование архитектуры системы

Для описания процессов, которые происходят в системе, созданы функциональные схемы, которые представлены в нотации IDEF0 на рисунках А.1 – А.5. Данные диаграммы бизнес-процессов иллюстрируют систему управления групповыми мероприятиями. Декомпозиция блока A0 отражает общий процесс – от входа пользователя в систему до обеспечения коммуникаций. Она включает ключевые подсистемы: создание мероприятия, управление задачами, бюджетом и коммуникациями.

Декомпозиция блока A2 детализирует этап создания и управления мероприятием. Здесь отображаются процессы ввода данных мероприятия, приглашения участников и редактирования информации, что позволяет гибко настраивать событие под нужды организатора.

Декомпозиция блока A3 описывает управление задачами: создание, назначение, изменение статуса и уведомление участников. Это обеспечивает прозрачность выполнения задач и упрощает координацию командной работы.

Декомпозиция блока A4 описывает управление расходами: создание, выбор метода разделения, расчет и генерация отчета.

В совокупности эти диаграммы демонстрируют структуру взаимодействия пользователей, системных компонентов и данных внутри веб-приложения планирования мероприятий.

Для разрабатываемой системы подойдет именно микросервисная архитектура [20], так как у нас есть четкое разделение модулей по зонам ответственности. Это позволит разрабатывать разные модули отдельно и отдельно их тестировать, а также в дальнейшем легко масштабировать систему путем увеличения экземпляров сервисов.

Можно выделить следующие компоненты системы:

* Брокер сообщений. Получает, хранит и отправляет задачи потребителям, а также получает, хранит и отправляет потребителям результаты выполнения задач и статусы;
* API-шлюз. Обрабатывает запросы от пользователей и маршрутизирует их между сервисами.
* Основной сервис. Занимается обработкой основных данных по мероприятиям и задачам.
* Сервис уведомлений. Получает из брокера задачу на отправку уведомления, создает уведомление и отправляет на почту;
* Сервис комментариев. Получает из брокера задачу на создание комментария и сохраняет его.

Выделение компонентов системы было обусловлено необходимостью обеспечения модульности, масштабируемости и надёжности веб-приложения для управления групповыми мероприятиями. Брокер сообщений был введён для асинхронной обработки задач, что позволяет разгрузить основные сервисы, обеспечивать устойчивость системы при сбоях и гарантировать доставку сообщений между компонентами. Основной сервис был выделен для обработки ключевых данных по мероприятиям и задачам, так как это центральная часть функциональности системы, требующая высокой производительности и целостности данных. Сервис уведомлений был отделён для независимой обработки отправки уведомлений. Сервис комментариев был вынесен отдельно для управления коммуникацией, что обеспечивает изоляцию этой функции и упрощает её масштабирование при росте активности пользователей.

Создадим диаграмму компонентов системы для наглядного отображения проектируемой системы (Рисунок 2.1).

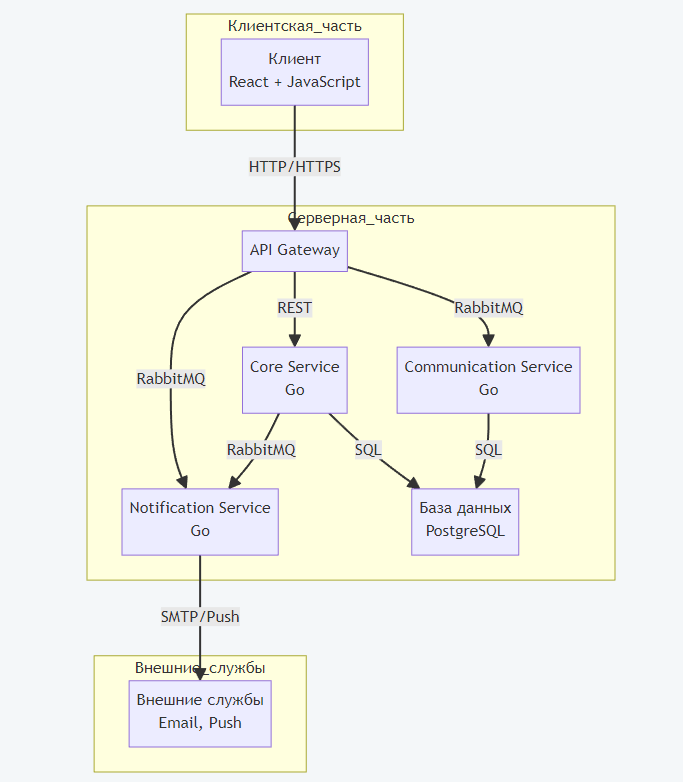


Рисунок 2.1 – Диаграмма компонентов системы (разработано автором)

Диаграмма компонентов, созданная с использованием Mermaid, отражает архитектуру системы управления групповыми мероприятиями. Она включает клиентскую часть на React и JavaScript, взаимодействующую с сервером через API Gateway по HTTP/HTTPS, что обеспечивает стандартное и безопасное соединение. Серверная часть состоит из Core Service, Notification Service и Communication Service, что подчёркивает выбор высокопроизводительной технологии для обработки запросов.

API Gateway служит точкой входа, маршрутизируя запросы к сервисам. Core Service обрабатывает основную логику и связан с базой данных PostgreSQL, гарантируя надёжное хранение данных. Notification Service отправляет уведомления через внешние службы по SMTP или Push, а Communication Service управляет комментариями, также используя базу данных. Особенностью является применение RabbitMQ [21] для асинхронного взаимодействия Notification Service и Communication Service с API Gateway, что повышает устойчивость системы при нагрузках.

Отсутствие связи Notification Service с базой данных указывает на то, что модуль не должен как-либо взаимодействовать с базой данных. Это обусловлено тем, что модуль выполняет работу исключительно с внешними службами. Внешние службы подключаются модульно, что упрощает расширение системы.

2.3. Проектирование основного модуля

При проектировании системы управления групповыми мероприятиями важно было заложить основу, которая обеспечит гибкость, масштабируемость и простоту сопровождения. Диаграмма классов Core Service, представленная на рисунке 2.2, отражает ключевые архитектурные решения, принятые для достижения этих целей. Она визуализирует основные компоненты системы, их атрибуты, методы и взаимосвязи, демонстрируя, как объектно-ориентированный подход помог создать модульную и расширяемую структуру.

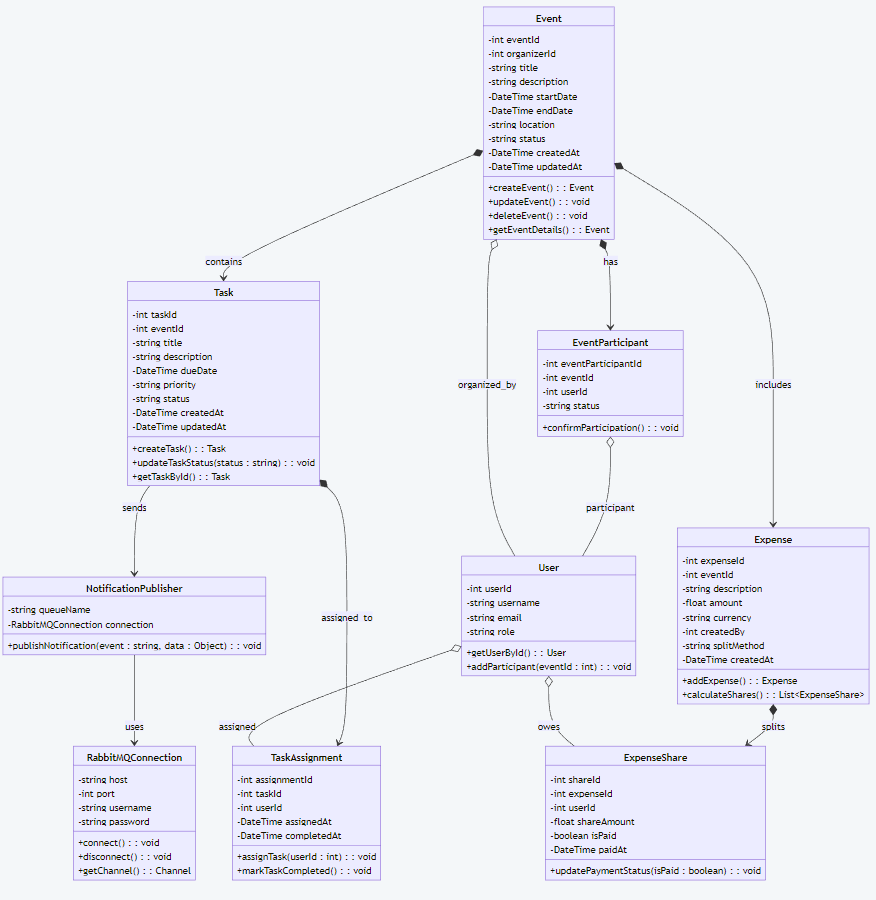
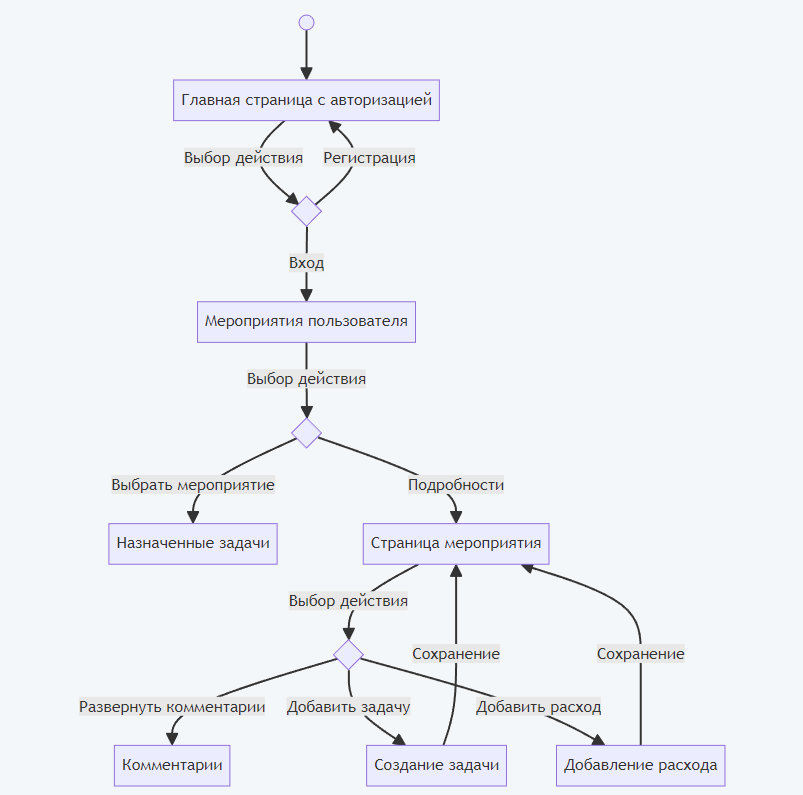


Рисунок 2.2 – Диаграмма классов основного сервиса (разработано автором)

Архитектурно диаграмма демонстрирует чёткое разделение обязанностей: Event выступает центральным узлом, объединяющим задачи (Task), расходы (Expense) и участников (EventParticipant) через отношения композиции и агрегации с User. Задачи и расходы формируют подчинённые структуры, где TaskAssignment и ExpenseShare обеспечивают связь с пользователями. Связь с системой уведомлений интегрирована через NotificationPublisher, который зависит от RabbitMQConnection, реализуя взаимодействие с брокером сообщений. Таким образом, Core Service обрабатывает бизнес-логику мероприятий, координирует действия участников и передаёт события другим системам, сохраняя модульную структуру.

2.4. Проектирование клиентской части системы

Диаграмма навигации клиентской части системы управления групповыми мероприятиями, представленная на рисунке 2.3, показывает структуру пользовательского интерфейса и основные пути взаимодействия пользователя с функциональными модулями приложения. Данная диаграмма отображает иерархическую организацию страниц и переходы между ними, обеспечивая наглядное представление логики навигации.

 Рисунок 2.3 – Навигация по сайту (разработано автором)

Структура и элементы диаграммы:

* страница под названием «Главная страница с авторизацией» служит начальной точкой доступа к системе. С неё пользователь может выполнить вход, что переводит его на страницу мероприятий пользователя, либо зарегистрироваться, возвращаясь на ту же страницу для повторной попытки авторизации;
* «Мероприятия пользователя» представляет собой центральный узел навигации, отображающий список мероприятий, к которым относится пользователь. Отсюда доступны переходы к деталям конкретного мероприятия или списку назначенных задач;
* страница «Назначенные задачи» отображает задачи, назначенные пользователю, с возможностью возврата к списку мероприятий;
* «Страница мероприятия» предоставляет детальную информацию о выбранном мероприятии и служит переходником для дополнительных действий: добавления задач, расходов или просмотра комментариев, а также возврата к списку мероприятий;
* «Создание задачи» позволяет пользователю добавить новую задачу к мероприятию, после чего осуществляется возврат на страницу мероприятия;
* «Добавление расхода» предназначено для регистрации новых расходов, связанных с мероприятием, с последующим возвратом на страницу мероприятия;
* «Комментарии» открываются для просмотра и взаимодействия с комментариями к мероприятию, с возможностью возврата на страницу мероприятия.

2.5. Проектирование схемы базы данных

Для реализации системы управления групповыми мероприятиями была разработана схема базы данных, которая обеспечивает хранение и управление данными о пользователях, мероприятиях, задачах, расходах и сообщениях. Структура базы данных представлена на ER-диаграмме (рисунок 2.4). Диаграмма отображает сущности, их атрибуты и взаимосвязи, обеспечивая целостное представление данных, необходимых для функционирования системы.

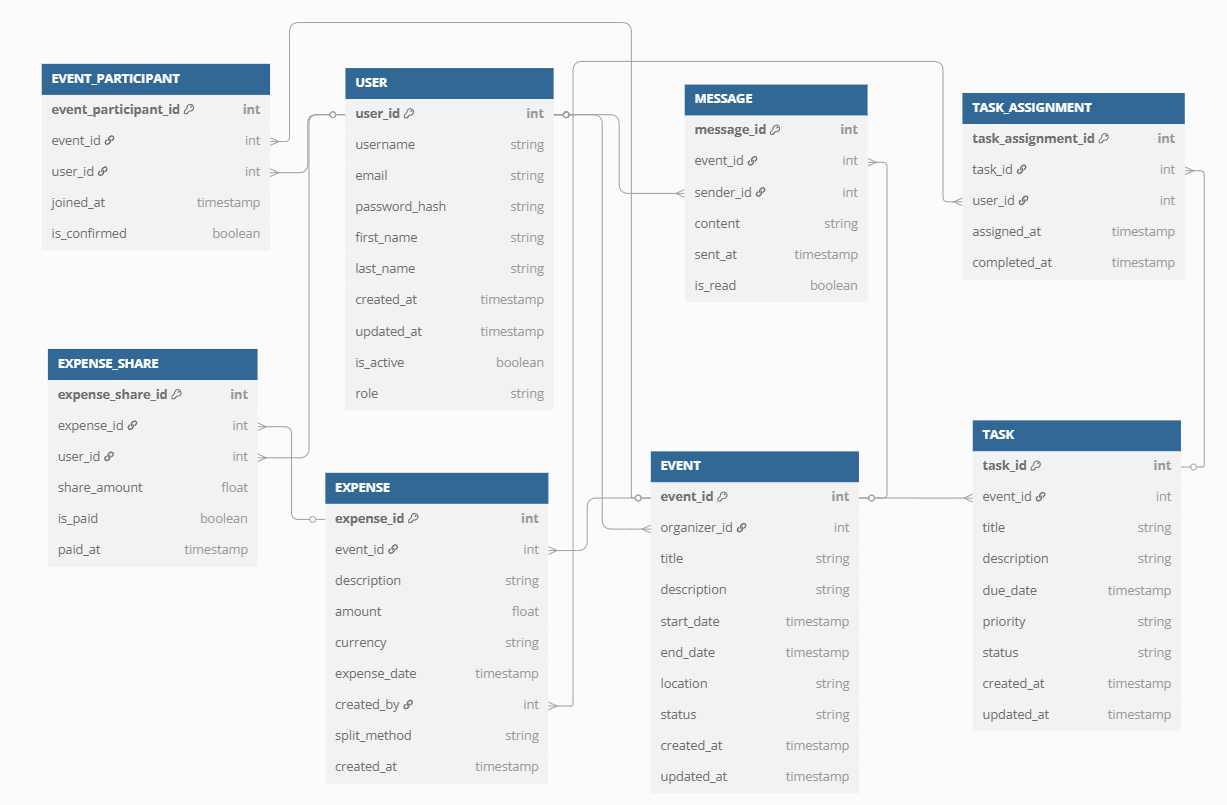


Рисунок 2.4 – ER-диаграмма (разработано автором)

Диаграмма включает основные сущности:

* USER – центральная сущность, представляющая пользователей, которые организуют мероприятия, участвуют в них, выполняют задачи, делят расходы и отправляют сообщения;
* EVENT – основа системы, объединяющая задачи, расходы и сообщения, с привязкой к организатору и участникам;
* TASK и EXPENSE – функциональные модули, связанные с мероприятиями, с дополнительными связями через TASK\_ASSIGNMENT и EXPENSE\_SHARE для распределения обязанностей и финансов;
* MESSAGE – обеспечивает коммуникацию в рамках мероприятий;
* EVENT PARTICIPANT – связывает пользователей с мероприятиями, отражая их статус участия.

Каждая сущность снабжена атрибутами, включая первичные и внешние ключи, что обеспечивает реляционные связи. Взаимодействия между сущностями реализованы через связи типа "один-ко-многим" (один пользователь организует много мероприятий), что отражает иерархическую и модульную архитектуру системы.

Вывод к разделу 2

В данной главе была спроектирована архитектура распределенной системы, описана с помощью UML-диаграммы компонентов. С помощью нотации IDEF0 была создана функциональная схема, отражающая бизнес-процессы системы. Также была детально спроектирована архитектура основного модуля управления мероприятиями и выбран наиболее подходящий жизненный цикл разработки приложения.

3 Технологический раздел

3.1 Разработка серверной части системы

При реализации каждого модуля было принято решение следовать принципам Чистой архитектуры ­­– архитектурного подхода к проектированию программного обеспечения, цель которого – сделать код легко читаемым, тестируемым, расширяемым и независимым от внешних деталей, таких как базы данных, фреймворки или интерфейс пользователя.

Основные идеи чистой архитектуры:

* разделение ответственности – код делится на слои, каждый из которых отвечает за своё;
* зависимости направлены внутрь – модули «внешнего мира» (БД, брокеры сообщений) зависят от бизнес-логики, а не наоборот;
* интерфейсы и инверсии зависимостей – конкретные реализации (например, репозитории, API-клиенты) внедряются через интерфейсы, что облегчает подмену и тестирование.

Следуя этим принципам была реализована структура, представленная на рисунке 3.1.

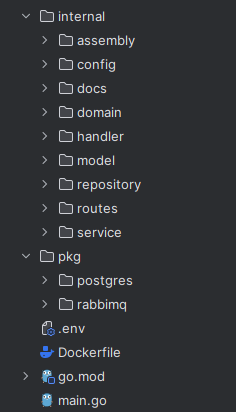
****

Рисунок 3.1 – структура проекта (разработано автором)

Краткое описание для каждого элемента:

* assembly – инициализация и связывание компонентов приложения;
* config – конфигурационные файлы и логика для работы с настройками;
* docs – документация (Swagger и т.п.);
* domain – доменные сущности, которые поступают на вход и выход;
* handler – HTTP-хендлеры (входная точка для запросов);
* model – структуры данных, используемые в приложении;
* repository – реализации для доступа к внешним модулям (БД и др.);
* routes – маршруты и логика роутинга;
* service – бизнес-логика или обработка данных, связующая слой handler и repository;
* pkg – реализация переиспользуемых компонентов (подключение к БД).

Каждый из реализованных модулей реализует все эти элементы.

3.1.1 Разработка модуля уведомлений

Модуль уведомлений (Notification Service) был разработан как часть системы управления групповыми мероприятиями для обеспечения своевременного информирования пользователей о ключевых событиях, таких как создание мероприятия, назначение задачи или добавление расхода.

Основная логика модуля реализована в классе NotificationService, который инкапсулирует функциональность обработки и отправки уведомлений. Структура класса включает два поля: config (конфигурация сервиса) и logger (логгер на основе библиотеки zap для журналирования событий). Конструктор NewNotificationService инициализирует эти поля, принимая конфигурацию и логгер как параметры. Код сервиса представлен на листинге 3.1.

Листинг 3.1 – Код основного класса NotificationService (разработано автором)

|  |
| --- |
| func (s \*NotificationService) ProcessNotification(message \*model.NotificationMessage) error {  emailContent, err := message.GenerateEmailContent()  if err != nil {  return err  }  if err := s.SendEmail(emailContent); err != nil {  return err  }  return nil  }  func (s \*NotificationService) SendEmail(content \*model.EmailContent) error {  smtpConfig := s.config.SMTP  smtpAddress := smtpConfig.GetSMTPAddress()  to := []string{content.To}  message := []byte(fmt.Sprintf(  "From: %s\r\n"+  "To: %s\r\n"+  "Subject: %s\r\n"+  "\r\n"+  "%s\r\n",  smtpConfig.Sender,  content.To,  content.Subject,  content.Body,  ))  err := smtp.SendMail(  smtpAddress, // Должно быть "localhost:1025"  nil, // Без аутентификации для MailHog  smtpConfig.Sender, // Отправитель  to, // Получатель  message, // Тело письма  )  if err != nil {  return fmt.Errorf("failed to send email: %w", err)  }  return nil  } |

Ключевые методы класса:

* ProcessNotification – обрабатывает входящее сообщение из RabbitMQ, генерирует содержимое email на основе типа события (event\_created, task\_assigned, expense\_added) и вызывает метод отправки email. Метод логирует этапы обработки и возвращает ошибку в случае сбоя;
* SendEmail – формирует email-сообщение (с полями From, To, Subject, Body) и отправляет его через SMTP-сервер;
* GetSupportedEvents – возвращает список поддерживаемых типов событий, определённых в модели SupportedEvents;
* GetStats – предоставляет статистику сервиса, включая поддерживаемые события, хост SMTP и адрес отправителя, что полезно для мониторинга.

3.1.2 Разработка модуля комментариев

Работа с комментариями была выделена в отдельный модуль так как предполагается, что нагрузка на систему комментариев будет выше чем на другой функционал приложения. Исходя из этого все операции по комментариям (получение, удаление, создание) реализованы в этом сервисе.

Как было описано выше задача на создание нового комментария кладется в очередь «/comments» и модуль постепенно забирает из неё задачи и обрабатывает. Такой подход подразумевает быструю обработку новых задач, что значительно разгружает API-gateway. Остальные операции проходят по REST.

На листинге Б.1 представлен код консьюмера для сервиса комментариев. Основная суть заключается в том, что мы асинхронно получаем сообщение из очереди, парсим в нужный нам тип данных и дальше отдаем на обработку сервису, если все хорошо и сообщение успешно сохранено – делаем Ack (сообщение для RabbitMQ об успешной обработке), иначе – Nack, что означает что мы отправляем сообщение на повторную обработку или в DLQ (Dead-Letter Queue).

На листинге Б.2 представлен код хэндлеров для операций чтения и удаления. Здесь все происходит похожим образом – валидируем запрос, т.е. проверяем указан ли id для нужного нам действия и отправляем на обработку слою сервиса, а он в свою очередь передает на уровень репозитория где происходит работа с базой банных, и возвращаем соответствующий ответ.

3.1.3 Разработка основного модуля

Главная бизнес-логика или так называемое «ядро приложения» находится в core-service. Здесь происходит основная обработка данных и преобразования в нужные форматы.

Основная логика сервиса:

* работа с пользователями (создание, удаление и др.);
* управление событиями, что включает в себя добавление новых участников события и их удаление;
* управление задачами, в том числе назначение исполнителей задач, представление в удобном формате для сохранения в БД;
* управление расходами – генерация отчета по балансам пользователей в рамках события, кто кому сколько должен вернуть и др.

Рассмотрим две главные особенности веб-приложения – составление отчетов по расходам и управление задачами.

Для того чтобы не сильно усложнять бизнес-логику приложения, и генерация отчетов не занимала много времени был составлен оптимизированный запрос к БД представленный на листинге 3.2.

Листинг 3.2 – Запрос для генерации отчета по расходам (разработано автором)

|  |
| --- |
| SELECT  ep.user\_id,  u.username,  COALESCE(SUM(CASE WHEN e.created\_by = ep.user\_id THEN e.amount ELSE 0 END), 0) -  COALESCE(SUM(COALESCE(es.amount, 0)), 0) AS balance  FROM event\_participant ep  JOIN users u ON u.user\_id = ep.user\_id  LEFT JOIN expense e ON e.event\_id = ep.event\_id  LEFT JOIN expense\_share es ON es.expense\_id = e.expense\_id AND es.user\_id = ep.user\_id  WHERE ep.event\_id = $1  GROUP BY ep.user\_id, u.username  ORDER BY balance DESC |

Этот запрос рассчитывает баланс каждого участника события: сколько он потратил минус сколько должен. Для этого он объединяет таблицы участников, расходов и долей, группирует по пользователю и возвращает их id, имя и итоговый баланс, отсортированный по убыванию. Оптимизирован этот запрос за счет отсутствия вложенных запросов и CTE (Common Table Expression или Обобщенные Табличные Выражения), что существенно ускорило генерацию отчета. На листинге Б.3 представлена обработка запроса на уровне сервиса.

Далее рассмотрим бизнес-логику для управления задачами. На листинге Б.4 показан код сервиса для создания задачи.

Одной из особенностей важно отметить структуру хранения задач. В ходе работы было принято решение добавить в структуру поле parentId, позволяющее создавать подзадачи к любой задаче. Это позволяет организовывать задачи в деревья любой длины и добавляет организации мероприятия определенную гибкость в оформлении структуры задач.

Также как можно увидеть на листинге Б.5 – задачи можно обновлять. В первую очередь это касается статусов задач и их исполнителей. Если пользователь пытается назначить на задачу исполнителя, который уже назначен система не позволит ему это сделать – вернется ошибка.

3.1.4 Разработка API-gateway

Основной задачей API-gateway является проксирование запросов к другим сервисам, в зависимости от переданного URL, но, чтобы упростить взаимодействие между сервисами и не добавлять еще один сервис в архитектуру приложения, было принято решение добавить функционал авторизации.

Авторизация в приложении реализована с помощью JWT, т.е. регистрация пользователей, аутентификация по юзернейму и паролю, генерация и валидация JWT-токенов, а также обновление и отзыв access- и refresh-токенов. При успешной аутентификации генерируются два токена: access-токен с коротким временем жизни и refresh-токен для обновления авторизации без повторного ввода логина и пароля. Все токены подписываются секретным ключом и содержат полезную нагрузку (claims), включая роль пользователя и время действия. На листинге Б.6 представлена реализация генерации токенов. Для хранения access- и refresh-токенов был использован Redis, который выполняет функцию кэша приложения.

Для простоты обработки запросов другими модулями и функциями в контекст запроса включается user\_id и user\_role, что позволяет легко извлекать и использовать их при необходимости (Листинг 3.3).

Листинг 3.3 – Обогащение контекста запроса (разработано автором)

|  |
| --- |
| if claims != nil {  c.Set("user\_id", claims.UserId)  c.Set("user\_role", claims.Role)  c.Next()  } else {  c.JSON(http.StatusUnauthorized, gin.H{"error": "Invalid token claims"})  c.Abort()  } |

На листинге Б.7 представлена полная реализация middleware для защищенных эндпоинтов.

3.2 Разработка пользовательского интерфейса системы

В рамках этого подраздела рассмотрим разработку страниц для основного функционала системы – это управление расходами и задачами.

Для отображения рекомендаций по расходам была написана функция представленная на листинге Б.8.

Функция generateSettlementRecommendations создает оптимальный план расчетов между участниками события с ненулевыми балансами. Алгоритм анализирует, кто кому должен деньги, и формирует минимальное количество транзакций для погашения всех долгов. Функция сортирует пользователей от самых больших должников до самых больших кредиторов, затем предлагает переводы от должников к кредиторам, постепенно нормализуя балансы всех участников. Результатом работы является список рекомендаций по переводам с указанием, кто, кому и сколько должен перевести для справедливого распределения расходов на мероприятии. На рисунке 3.2 можно увидеть отображение в интерфейсе пользователя страницы расходов.

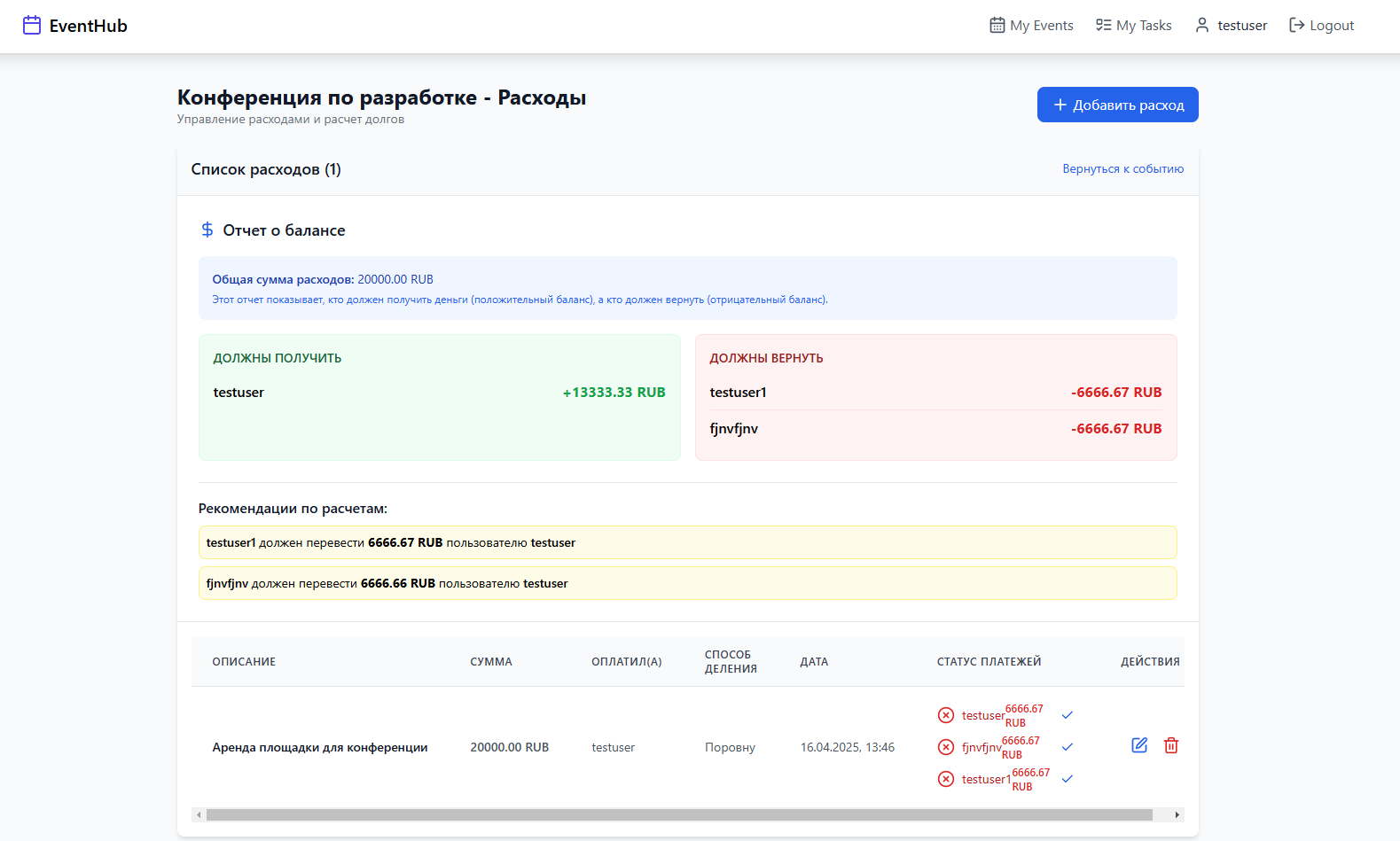


Рисунок 3.2 – Страница управления расходов (разработано автором)

На листинге Б.9 представлен код, который отвечает за обработку иерархической структуры задач в приложении. Функция checkForCycles предотвращает циклические зависимости между задачами, анализируя цепочки родительских связей и разрывая их при обнаружении циклов. Алгоритм преобразует линейный список задач в иерархическую структуру, нормализуя родительские ссылки и создавая карту задач для быстрого доступа. Результат (Рисунок 3.3) обеспечивает корректное древовидное представление задач, что позволяет правильно отображать и управлять вложенными задачами в интерфейсе пользователя.

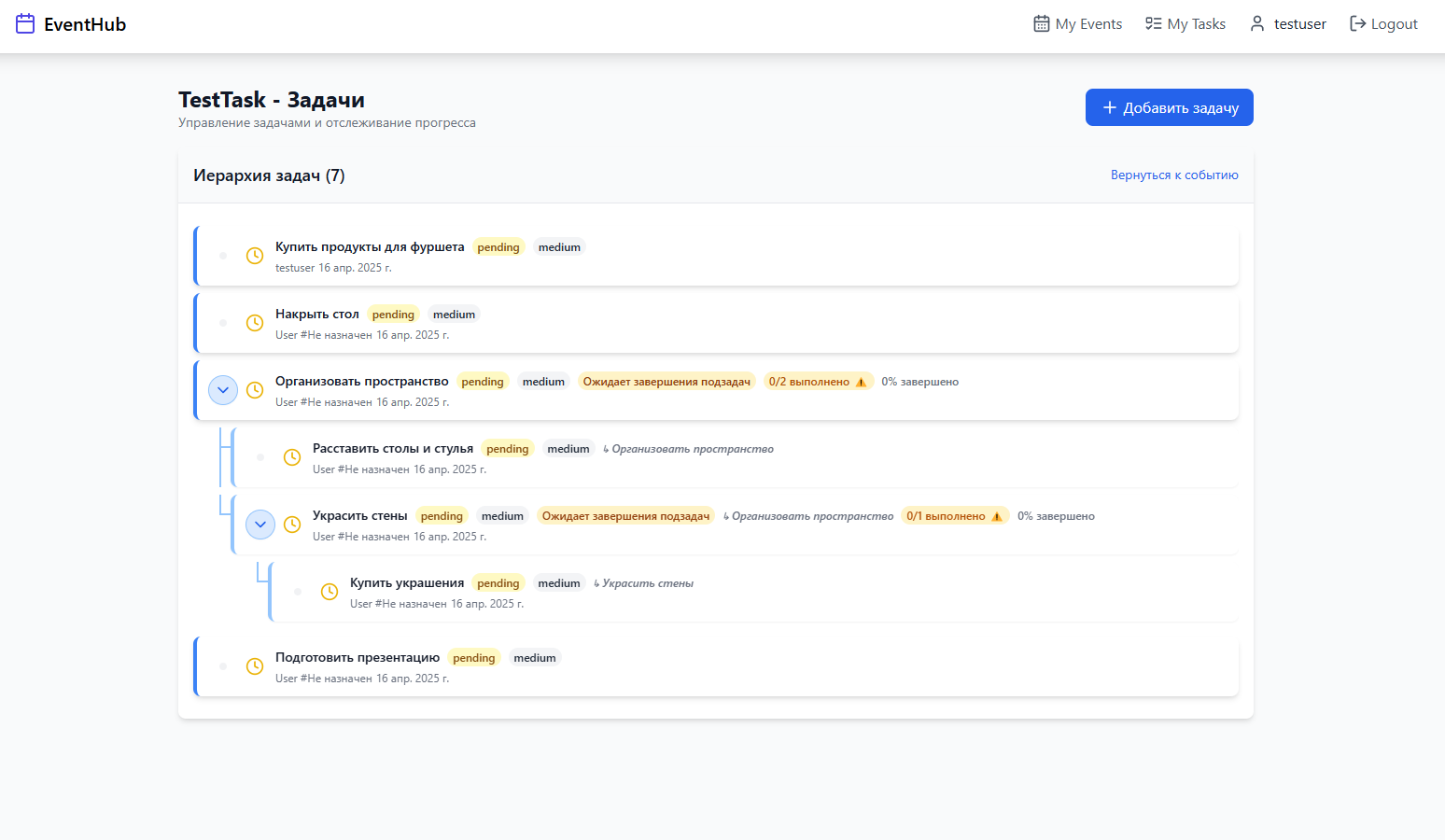


Рисунок 3.3 – Страница управления задачами (разработано автором)

На рисунке 3.4 изображена страница конкретного события. На этой странице отображаются:

* общая информация о событии;
* участники события;
* обсуждение с комментариями;
* краткая информация о последних созданных задачах;
* краткая информация о расходах.

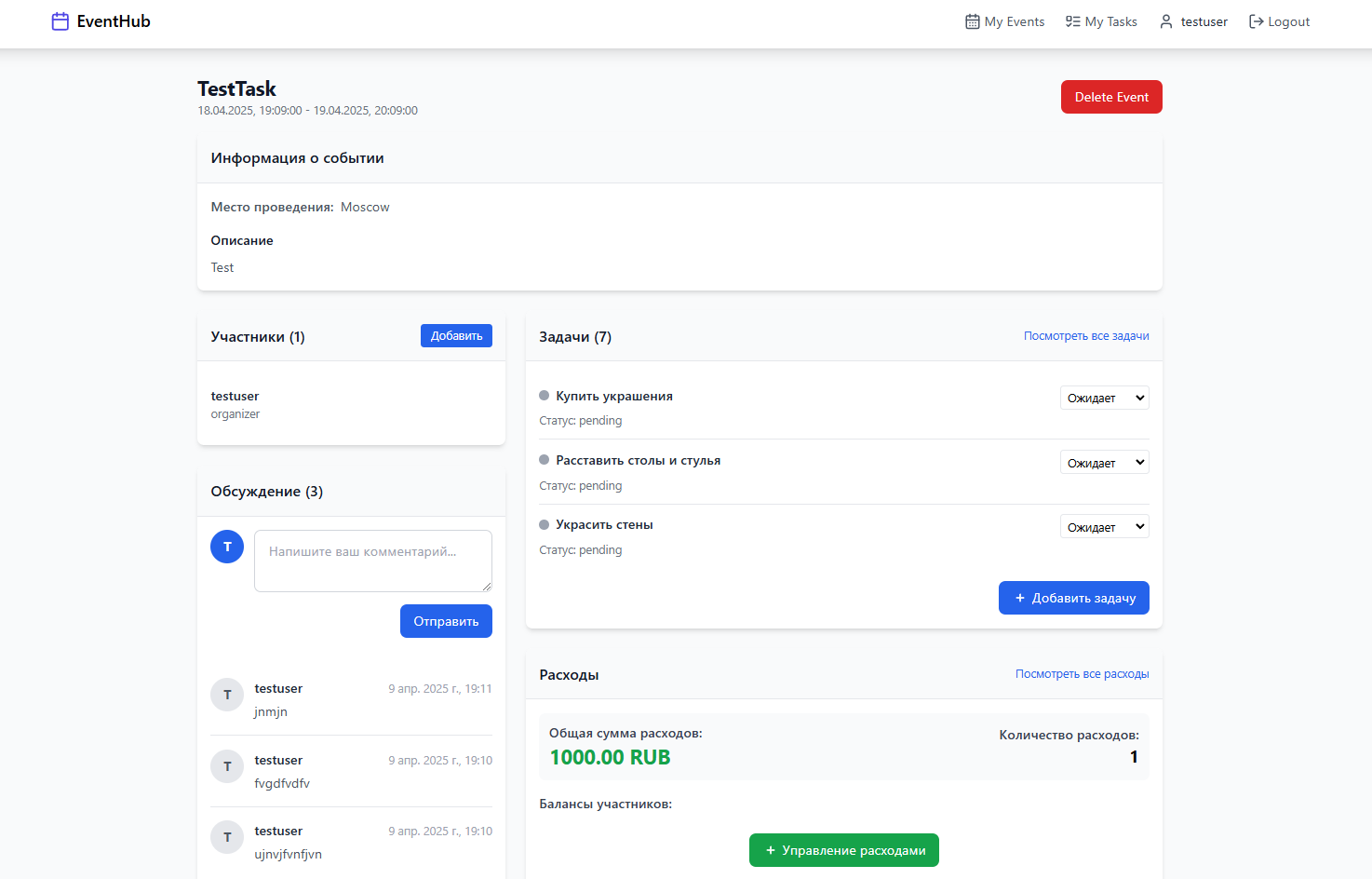


Рисунок 3.4 – Страница конкретного события (разработано автором)

3.3 Тестирование веб-приложения

Помимо тестирования, проводившегося в процессе разработки как подэтап итеративного жизненного цикла, было проведено итоговое тестирование заявленного функционала для демонстрации корректной работы веб-приложения. Функционал был протестирован вручную, тесты и их результаты сведены в таблицу (Таблица 3.1).

Таблица 3.1 – Интеграционное тестирование клиентской и серверной части приложения (разработано автором)

|  |  |  |
| --- | --- | --- |
| **Действие** | **Ожидаемый результат** | **Полученный результат** |
| Вход на сайт | Отобразится главная страница со списком мероприятий | Главная страница со списком мероприятий отображена |
| Добавление участника в мероприятие | Новый участник отобразится в списке участников мероприятия | Участник добавлен и отображен в списке |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Добавление комментария | Комментарий добавится в обсуждение и отобразится сразу | Комментарий добавлен и отображен в списке обсуждений |
| Создание задачи | Задача сохранится на сервере и отобразится в списке задач | Задача создана и отображена в списке |
| Изменение статуса задачи | Статус задачи изменится в системе и отобразится новый статус | Статус задачи обновлен и отображен |
| Добавление расхода | Расход сохранится и обновятся балансы участников | Расход добавлен, балансы участников обновлены |
| Удаление задачи | Задача будет удалена с сервера и исчезнет из списка | Задача удалена и больше не отображается в списке |

Проведем ручное интеграционное тестирование с помощью Postman.

Для начала проверим работоспособность аутентификации. Отправим запрос с телом, содержащим email, username и пароль. В ответе мы должны получить информацию о созданном пользователе, пару access- и refresh-токенов и через сколько истекает refresh-токен. Как мы видим по рисунку 3.5 результат соответствует ожиданиям.

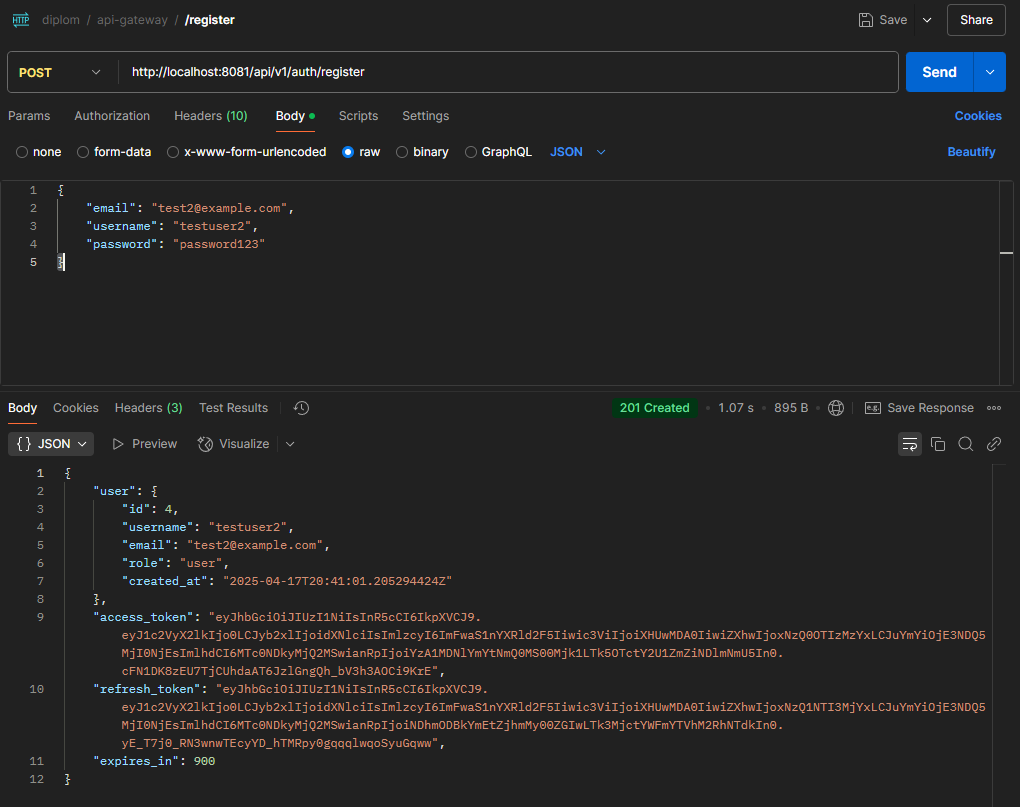


Рисунок 3.5 – Результат выполнения запроса на регистрацию пользователя (разработано автором)

Далее проведем тестирование основного функционала веб-приложения. Начнем с создания расхода. В запросе необходимо указать общую информацию о расходе (сумма расхода, описание, метод разделения и т.д.) и пользователей, которые должны будут разделить этот расход. В ответе должны получить id созданного расхода. На рисунке 3.6 видно, запрос отрабатывает корректно.

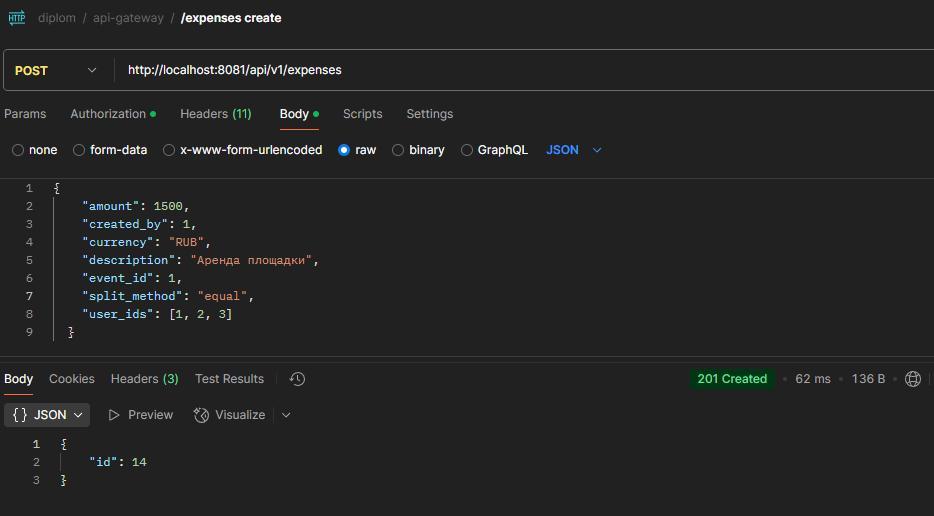


Рисунок 3.6 – Запрос на создание расхода (разработано автором)

Далее проверим получение всей информации по событию. В теле запроса мы ничего не передаём, а id события передаем в URL. В ответе мы получаем довольно большую структуру (Рисунок 3.7), в которой находиться вся информация по событию, включая комментарии, расходы, участников события и задачи.

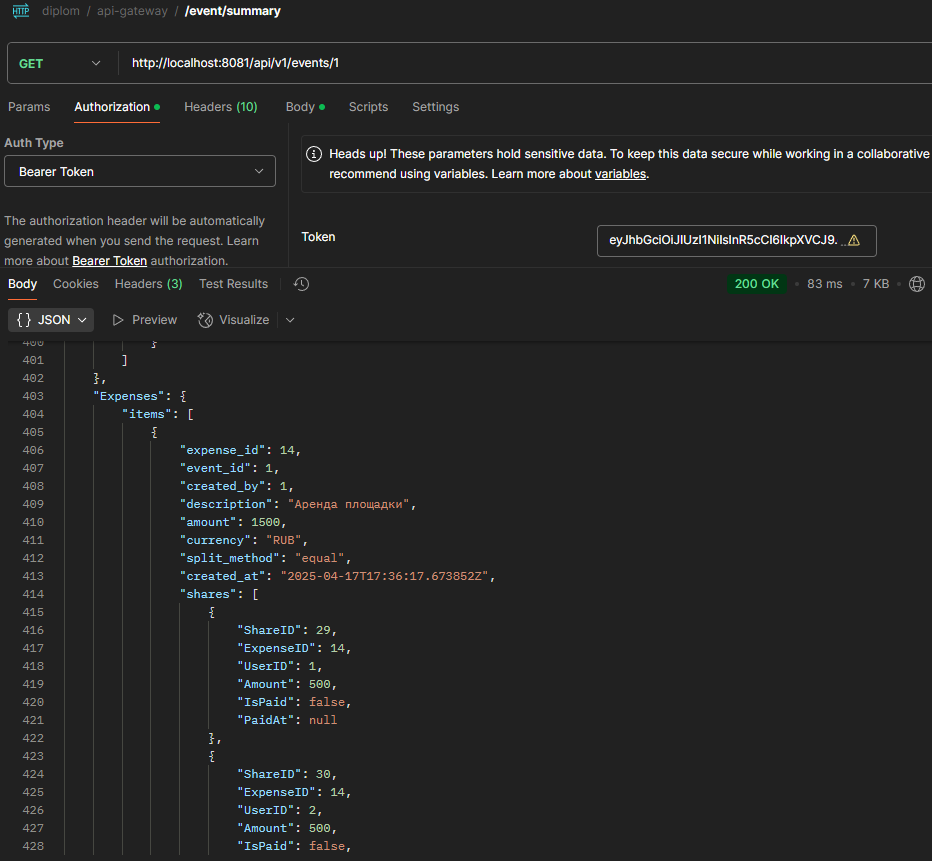


Рисунок 3.7 – Запрос на получение всей информации о событии (разработано автором)

Теперь выполним запрос на получение задач, назначенных конкретному пользователю. Ни в теле запроса, ни в URL мы ничего не передаем, т.к. id пользователя извлекается из его токена и в ответе получаем список задач с их описанием и другими полями. Результат выполнения запроса на рисунке 3.8.

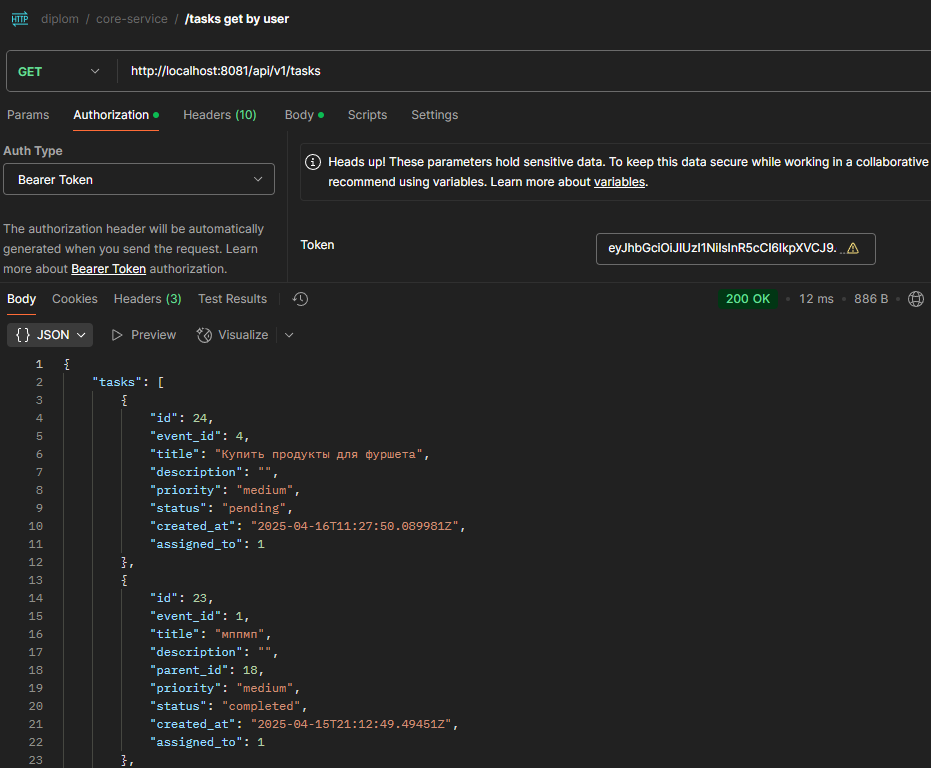


Рисунок 3.8 – Запрос на получение задач, назначенных конкретному пользователю (разработано автором)

Выполним запрос на создание новой задачи. В теле укажем информацию о создаваемой задаче – id события, которому она будет принадлежать, название, описание, id родительской задачи (опционально), приоритет и пользователя, которому хотим назначить (опционально). В ответе получаем созданную задачу с дополненной информацией. Результат на рисунке 3.9.

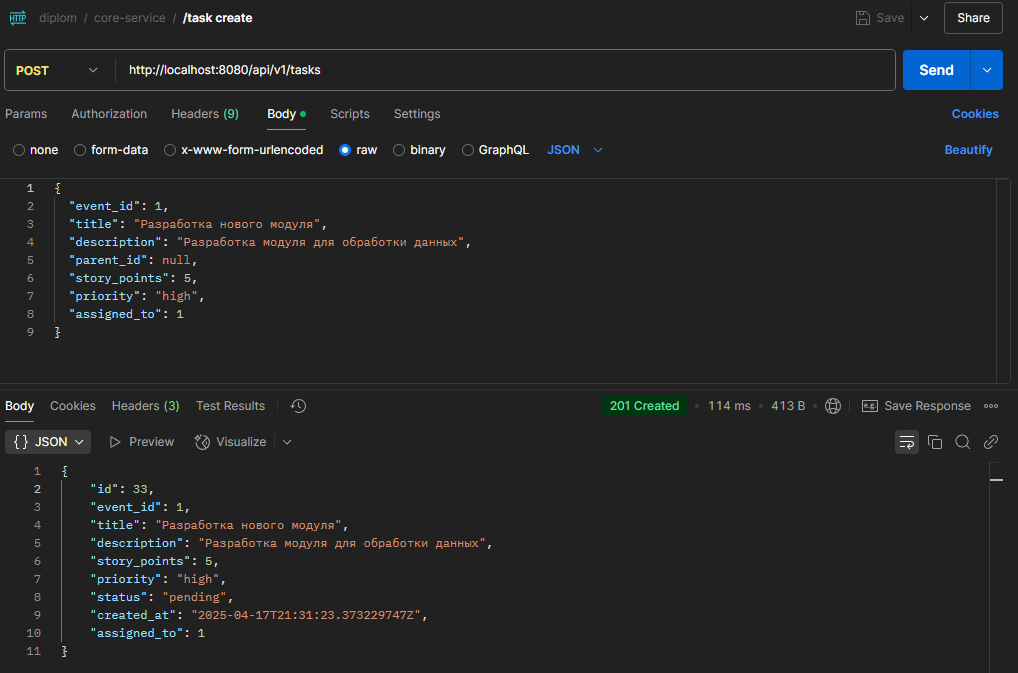


Рисунок 3.9 – Запрос на создание задачи (разработано автором)

Также было проведено модульное тестирование отдельно взятых функций, а в контексте этого веб-приложения – ключевых, а т.е. управления задачами и расходами. Были составлены тест-кейсы с различными вариантами поведения функций как положительными, так и отрицательными.

Для управления расходами были выбраны функции создания расхода и получения отчета по балансам пользователя. Результаты юнит-тестов показаны на рисунке 3.10.

Для управления задачами были выбраны функции создания, обновления и получения задач. Результаты юнит-тестов показаны на рисунке 3.11.

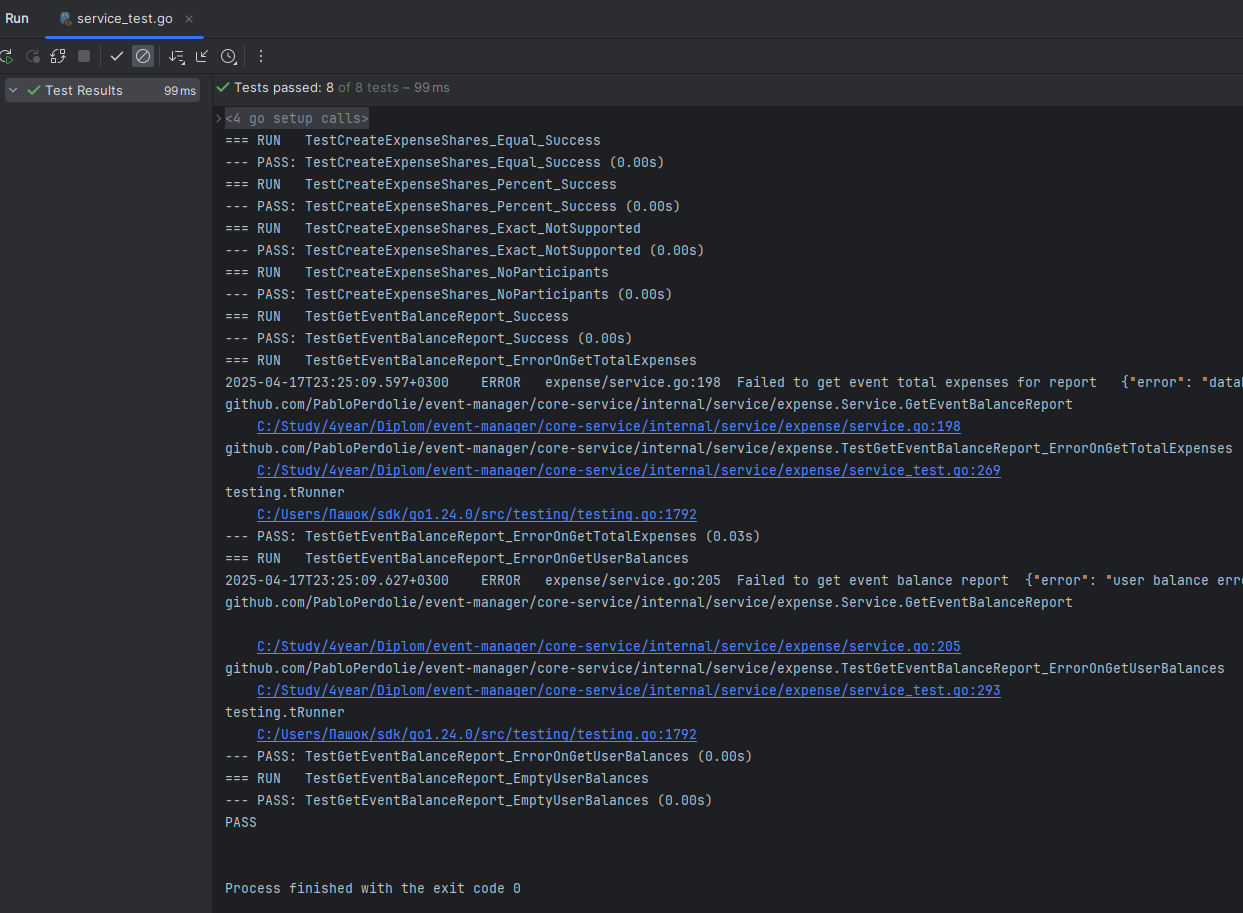


Рисунок 3.10 – Юнит-тесты функционала расходов (разработано автором)

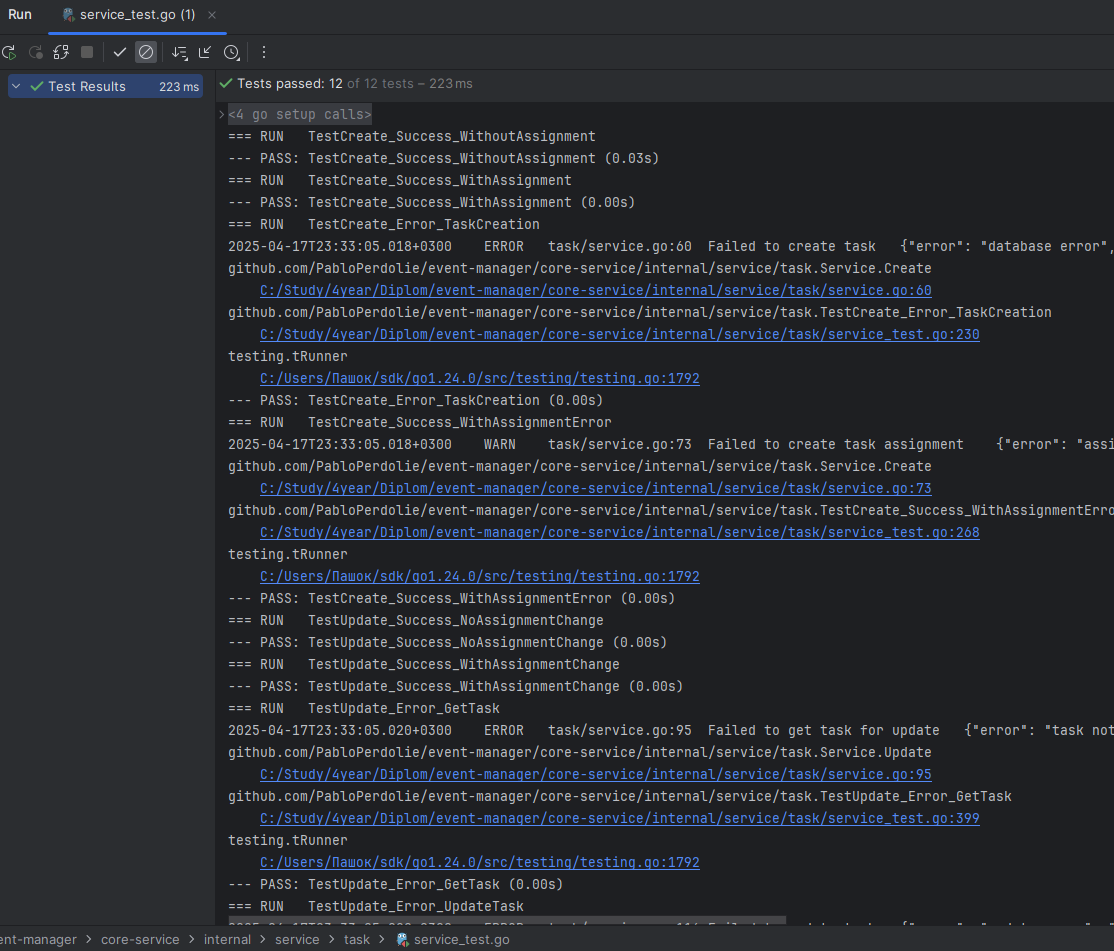


Рисунок 3.11 – Юнит-тесты на функционал управления задачами (разработано автором)

Тестирование показало работоспособность разработанного веб-приложения и ее интерфейса. Все тест-кейсы пройдены успешно.

3.4 Расчет вычислительной и емкостной сложности

Для расчета необходимых ресурсов необходимо рассчитать временную и емкостную сложность алгоритмов в худшем случае. Для расчета будут рассмотрены основные методы серверной части. Количество данных, с которыми работает сервис, будет иметь обозначение n. Результаты расчета сложности алгоритма представлены в таблице 3.1, а сами расчеты в формулах (3.1 - 3.4)

Таблица 3.1 – Сложность основных методов серверной части (разработано автором)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Функция | | Описание | T | V | | |
| ListTaskByEvent | | Метод получения списка задач по событию | O(n) | O(n) | | |
| GetEventBalanceReport | | Метод получения баланса пользователей в событии | O(n+k) | O(k) | | |
|  | | | | (3.1) |
|  | | | | (3.2) |

где:

T – вычислительная сложность;

ListTaskByEvent – метод получения списка задач по идентификатору события;

O – трудоемкость алгоритма;

n – количество задач в событии;

V – емкостная сложность.

|  |  |
| --- | --- |
|  | (3.3) |
|  | (3.4) |

где:

n – количество расходов в событии;

k – количество пользователей в событии;

GetEventBalanceReport – метод получения отчета о балансе пользователей в событии.

В методе ListTaskByEvent временная сложность линейно зависит от количества задач в событии. Метод выполняет один запрос к базе данных для получения всех задач события, что обеспечивает эффективное использование ресурсов даже при большом количестве данных. Метод GetEventBalanceReport также имеет линейную сложность относительно своих входных данных. Таким образом, реализация программной логики и работы с данными в разработанной системе обеспечивает высокую эффективность вычислений для предполагаемой нагрузки.

Вывод к разделу 3

Разработка серверной части системы была выполнена с применением принципов Чистой архитектуры, что обеспечило высокую модульность, удобство сопровождения и расширяемость кода. Каждый модуль имеет чёткое разделение ответственности: бизнес-логика сосредоточена в сервисах, взаимодействие с внешними источниками реализовано через репозитории, а HTTP-интерфейс – через хендлеры. Это позволило упростить тестирование, а также легко адаптировать приложение к изменениям или новым требованиям.

Благодаря использованию асинхронной обработки сообщений через очередь RabbitMQ удалось разгрузить API-gateway и обеспечить быструю реакцию на события. Также реализован продуманный механизм генерации отчетов по расходам с оптимизированными SQL-запросами, что позволило существенно сократить время отклика даже при большом объеме данных.

Проведенное тестирование подтвердило корректность работы всех основных функций системы, включая управление пользователями, задачами и расходами. Интерфейс пользователя был разработан с учётом удобства работы с иерархией задач и отображения финансовых расчетов.

ЗАКЛЮЧЕНИЕ

В результате выполнения проектной практики был проведён анализ предметной области, связанной с управлением групповыми мероприятиями, в результате которого подтверждена актуальность темы и уникальность разрабатываемой системы.

Были определены функциональные и нефункциональные требования к информационной системе, обеспечивающие её соответствие поставленным задачам.

Проведено проектирование системы с использованием нотаций IDEF0 и UML, разработана архитектура распределённой системы, включая схему базы данных и навигацию клиентской части, а также детально спроектирован основной сервис (Core Service). Разработанные схемы и диаграммы полностью описывают процессы и структуру системы.

Выполнена реализация модуля уведомлений в соответствии с спроектированной архитектурой. Модуль был протестирован с использованием MailHog, в ходе чего подтверждено его соответствие заявленным требованиям, а также устранены ошибки, связанные с отправкой уведомлений.

Была создана презентация, отражающая выполненную практику.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Исследование Mordor Intelligence [Электронный ресурс]. – URL: https://www.mordorintelligence.com/industry-reports/event-management-software-market (дата обращения: 20.12.2024). – Текст: электронный.
2. Отчет о научно-исследовательской работе. Структура и правила оформления ГОСТ 7.32-2017 [Электронный ресурс]. – URL: https://cs.msu.ru/sites/cmc/files/docs/2021-11gost\_7.32-2017.pdf (дата обращения: 20.01.2025). – Текст: электронный.
3. Trello [Электронный ресурс]. – URL: <https://trello.com/> (дата обращения: 22.01.2025). – Текст: электронный.
4. Splitwise [Электронный ресурс]. – URL: https://www.splitwise.com/ (дата обращения: 22.01.2025). – Текст: электронный.
5. Asana [Электронный ресурс]. – URL: https://asana.com/ru (дата обращения: 22.01.2025). – Текст: электронный.
6. Docker [Электронный ресурс]. – URL: https://www.docker.com/ (дата обращения: 1.02.2025). – Текст: электронный.
7. Go Programming Language [Электронный ресурс]. – URL: https://go.dev/ (дата обращения: 1.02.2025). – Текст: электронный.
8. Java [Электронный ресурс]. – URL: https://www.java.com/ru/ (дата обращения: 1.02.2025). – Текст: электронный.
9. Node.js [Электронный ресурс]. – URL: https://nodejs.org/en (дата обращения: 1.02.2025). – Текст: электронный.
10. React [Электронный ресурс]. – URL: https://react.dev/ (дата обращения: 1.02.2025). – Текст: электронный.
11. Angular [Электронный ресурс]. – URL: https://angular.dev/ (дата обращения: 1.02.2025). – Текст: электронный.
12. Vue.js [Электронный ресурс]. – URL: https://ru.vuejs.org/ (дата обращения: 1.02.2025). – Текст: электронный.
13. PostgresPro [Электронный ресурс]. – URL: https://postgrespro.ru/ (дата обращения: 1.02.2025). – Текст: электронный.
14. UML [Электронный ресурс]. – URL: https://www.uml.org/ (дата обращения: 15.02.2025). – Текст: электронный.
15. BPMN [Электронный ресурс]. – URL: https://www.bpmn.org/ (дата обращения: 15.02.2025). – Текст: электронный.
16. Нотация IDEF0 [Электронный ресурс]. – URL: https://www.businessstudio.ru/wiki/docs/v4/doku.php/ru/csdesign/b\_remodeling/idef\_0 (дата обращения: 15.02.2025). – Текст: электронный.
17. Mermaid [Электронный ресурс]. – URL: https://mermaid.js.org/ (дата обращения: 15.02.2025). – Текст: электронный.
18. Статья «Микросервисы: что это такое и как их использовать» [Электронный ресурс]. – URL: https://habr.com/ru/articles/804629/ (дата обращения: 21.02.2025). – Текст: электронный.
19. Статья «Модели и методологии разработки ПО: обзор подходов» [Электронный ресурс]. – URL: https://gb.ru/blog/modeli-i-metodologii-razrabotki-po/ (дата обращения: 21.02.2025). – Текст: электронный.
20. Документация Oracle «Microservices Architecture» [Электронный ресурс]. – URL: https://docs.oracle.com/en/database/oracle/oracle-database/23/adfns/microservices-architecture.html#GUID-D8417169-91D7-456B-855A-065AC5068F19 (дата обращения: 21.02.2025). – Текст: электронный.
21. RabbitMQ [Электронный ресурс]. – URL: https://www.rabbitmq.com/ (дата обращения: 4.02.2025). – Текст: электронный.

Приложение А

Проектирование

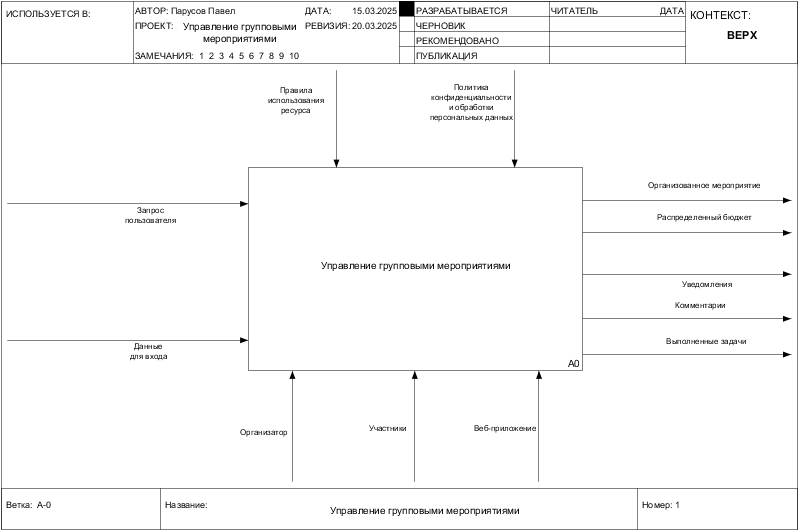


Рисунок А.1 – Диаграмма процессов (разработано автором)

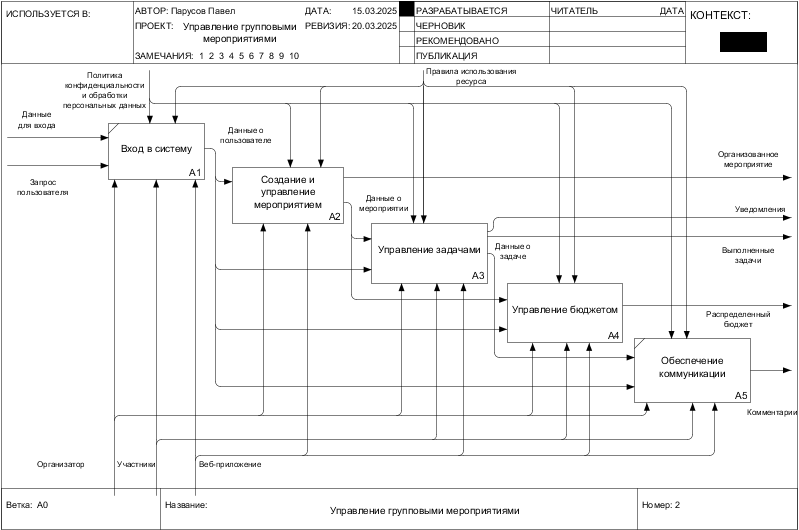


Рисунок А.2 – Декомпозиция блока А0 (разработано автором)

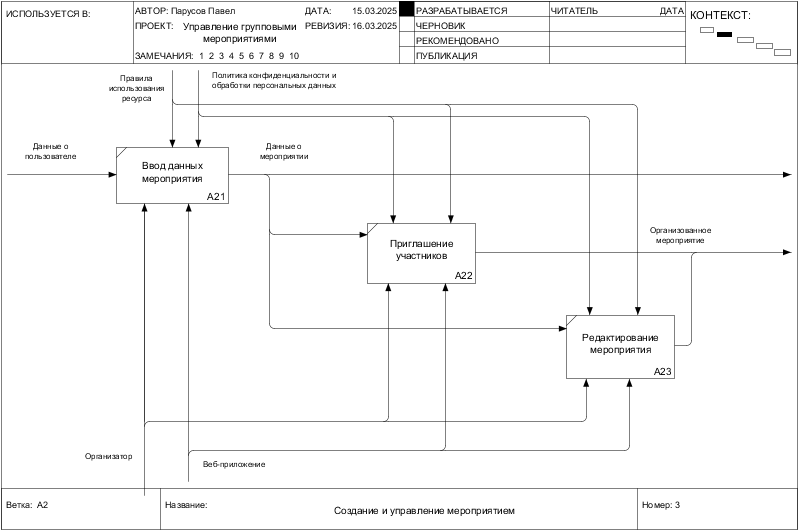


Рисунок А.3 – Декомпозиция блока А2 (разработано автором)

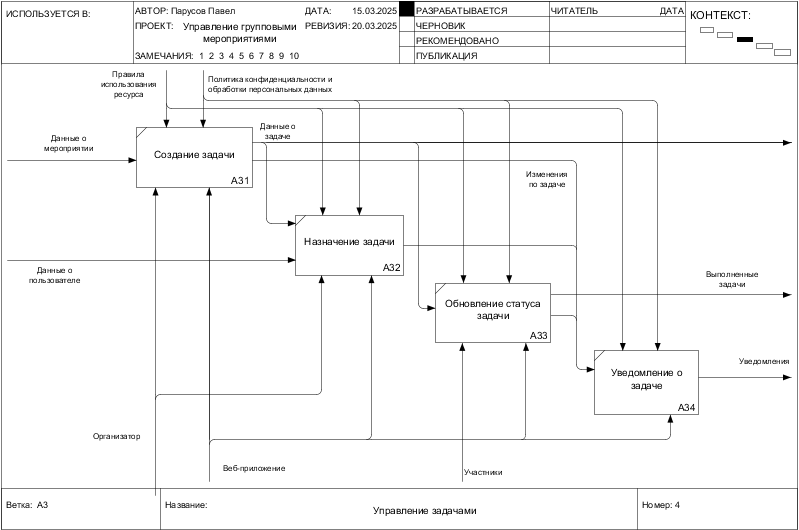


Рисунок А.4 – Декомпозиция блока А3 (разработано автором)

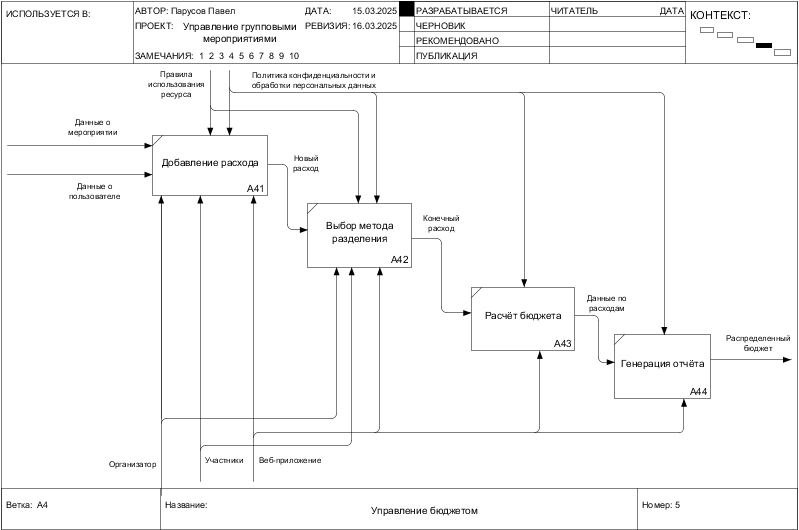


Рисунок А.5 – Декомпозиция блока А4 (разработано автором)

Приложение Б

Разработка

Листинг Б.1 – Код консьюмера комментариев (разработано автором)

|  |
| --- |
| type Comment interface {  CreateComment(ctx context.Context, comment domain.CreateCommentMessage) (int, error)  }  type CommentConsumer struct {  commentService Comment  logger \*zap.SugaredLogger  }  func NewCommentHandler(commentService Comment, logger \*zap.SugaredLogger) CommentConsumer {  return CommentConsumer{  commentService: commentService,  logger: logger,  }  }  func (h CommentConsumer) ProcessMessage(msg amqp.Delivery) {  var message domain.CreateCommentMessage  err := json.Unmarshal(msg.Body, &message)  if err != nil {  h.logger.Errorw("failed to unmarshal message", "error", err)  msg.Reject(false)  return  }  id, err :=  h.commentService.CreateComment(context.Background(), message)  if err != nil {  h.logger.Errorw("failed to create comment", "error", err)  msg.Reject(false)  return  }  h.logger.Infow("comment created", "comment\_id", id)  msg.Ack(false)  } |

Листинг Б.2 – Код хэндлеров модуля комментариев (разработано автором)

|  |
| --- |
| // GetCommentsByEventId godoc  // @Summary Получить комментарии события  // @Description Возвращает все комментарии, связанные с указанным событием  // @Tags comments  // @Produce json  // @Param eventId path int true "ID события"  // @Success 200 {array} model.Comment "Список комментариев"  // @Failure 400 {object} map[string]interface{} "Некорректный ID события"  // @Failure 500 {object} map[string]interface{} "Внутренняя ошибка сервера"  // @Router /comments/{eventId} [get]  func (h \*CommentHandler) GetCommentsByEventId(c \*gin.Context) {  eventIdParam := c.Param("event\_id")  if eventIdParam == "" {  c.JSON(http.StatusBadRequest, gin.H{"error": "missing eventId parameter"})  return  }  eventId, err := strconv.Atoi(eventIdParam)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "invalid eventId parameter"})  return  }  comments, err := h.service.GetCommentsByEventId(c, eventId)  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "internal server error"})  return  }  c.JSON(http.StatusOK, comments)  }  // DeleteComment godoc  // @Summary Удалить комментарий  // @Description Удаляет комментарий по его идентификатору  // @Tags comments  // @Produce json  // @Param id path int true "ID комментария"  // @Success 204 "Комментарий успешно удален"  // @Failure 400 {object} map[string]interface{} "Некорректный ID комментария"  // @Failure 500 {object} map[string]interface{} "Внутренняя ошибка сервера"  // @Router /comments/{id} [delete]  func (h \*CommentHandler) DeleteComment(c \*gin.Context) {  idParam := c.Param("id")  if idParam == "" { |

Продолжение листинга Б.2

|  |
| --- |
| c.JSON(http.StatusBadRequest, gin.H{"error": "missing id parameter"})  return  }  id, err := strconv.Atoi(idParam)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "invalid id parameter"})  return  }  err = h.service.DeleteComment(c, id)  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "internal server error"})  return  }  c.Status(http.StatusNoContent)  } |

Листинг Б.3 – Код сервиса для генерации отчета (разработано автором)

|  |
| --- |
| func (s Service) GetEventBalanceReport(ctx context.Context, eventId int) (domain.BalanceReportResponse, error) {  // Получаем общую сумму расходов для события  totalAmount, err := s.expenseRepo.GetEventTotalExpenses(ctx, eventId)  if err != nil {  s.logger.Errorw("Failed to get event total expenses for report", "error", err, "eventId", eventId)  return domain.BalanceReportResponse{}, errors.WithMessage(err, "failed to get event total expenses")  }  // Получаем балансы участников  balances, err := s.expenseShareRepo.GetEventBalanceReport(ctx, eventId)  if err != nil {  s.logger.Errorw("Failed to get event balance report", "error", err, "eventId", eventId)  return domain.BalanceReportResponse{}, errors.WithMessage(err, "failed to get event balance report")  }  // Преобразуем в доменную модель  userBalances := make([]domain.UserBalance, len(balances))  for i, balance := range balances {  userBalances[i] = domain.UserBalance{  UserID: balance.UserID,  Username: balance.Username, |

Продолжение листинга Б.3

|  |
| --- |
| Balance: balance.Balance,  }  }  return domain.BalanceReportResponse{  EventID: eventId,  TotalAmount: totalAmount,  UserBalances: userBalances,  }, nil  } |

Листинг Б.4 – Код сервиса для создания задачи (разработано автором)

|  |
| --- |
| func (s Service) Create(ctx context.Context, req domain.TaskCreateRequest) (\*domain.TaskResponse, error) {  task := model.Task{  EventId: req.EventId,  ParentId: req.ParentId,  Title: req.Title,  Description: req.Description,  StoryPoints: req.StoryPoints,  Priority: req.Priority,  Status: string(domain.TaskStatusPending),  CreatedAt: time.Now(),  }  id, err := s.taskRepo.Create(ctx, task)  if err != nil {  s.logger.Errorw("Failed to create task", "error", err, "eventId", req.EventId)  return nil, errors.WithMessage(err, "create task")  }  if req.AssignedTo != nil {  assignment := model.TaskAssignment{  TaskId: id,  UserId: \*req.AssignedTo,  AssignedAt: time.Now(),  }  \_, err := s.assignmentRepo.Create(ctx, assignment)  if err != nil {  s.logger.Warnw("Failed to create task assignment", "error", err, "taskId", id, "userId", req.AssignedTo)  // Continue even if one assignment fails  }  } |

Продолжение листинга Б.4

|  |
| --- |
| return &domain.TaskResponse{  Id: id,  EventId: task.EventId,  AssignedTo: req.AssignedTo,  ParentID: task.ParentId,  Title: task.Title,  Description: task.Description,  StoryPoints: task.StoryPoints,  Priority: task.Priority,  Status: domain.TaskStatus(task.Status),  CreatedAt: task.CreatedAt,  }, nil  } |

Листинг Б.5 – Код сервиса для обновления задач (разработано автором)

|  |
| --- |
| func (s Service) Update(ctx context.Context, id int, req domain.TaskUpdateRequest) error {  if req.Title != nil {  task.Title = \*req.Title  }  if req.Description != nil {  task.Description = \*req.Description  }  if req.Priority != nil {  task.Priority = req.Priority  }  if req.Status != nil {  task.Status = string(\*req.Status)  }  if err := s.taskRepo.Update(ctx, task); err != nil {  return errors.WithMessage(err, "update task")  }  assignedTo := req.AssignedTo  if assignedTo != nil {  currentAssignments, \_ := s.assignmentRepo.ListByTask(ctx, id, 100, 0)  currentAssigneeMap := make(map[int]model.TaskAssignment)  for \_, assignment := range currentAssignments {  currentAssigneeMap[assignment.UserId] = assignment  }  if \_, exists := currentAssigneeMap[\*assignedTo]; !exists {  assignment := model.TaskAssignment{  TaskId: id,  UserId: \*assignedTo,  AssignedAt: time.Now(),  } |

Продолжение листинга Б.5

|  |
| --- |
| \_, err := s.assignmentRepo.Create(ctx, assignment)  if err != nil {  s.logger.Warnw("Failed to create task assignment", "error", err, "taskId", id, "userId", assignedTo)  }  }  delete(currentAssigneeMap, \*assignedTo)  for userId, assignment := range currentAssigneeMap {  err := s.assignmentRepo.Delete(ctx, assignment.UserId)  if err != nil {  s.logger.Warnw("Failed to delete task assignment", "error", err, "taskId", id, "userId", userId)  }  }  }  return nil  } |

Листинг Б.6 – Код сервиса для генерации токенов (разработано автором)

|  |
| --- |
| func (s Service) generateTokenPair(userId int, role string) (\*domain.TokenPair, error) {  accessClaims := domain.JWTClaims{  UserId: userId,  Role: role,  RegisteredClaims: jwt.RegisteredClaims{  ExpiresAt: jwt.NewNumericDate(time.Now().Add(s.accessTokenExpiry)),  IssuedAt: jwt.NewNumericDate(time.Now()),  NotBefore: jwt.NewNumericDate(time.Now()),  Issuer: "api-gateway",  Subject: string(rune(userId)),  ID: uuid.New().String(),  },  }  accessToken := jwt.NewWithClaims(jwt.SigningMethodHS256, accessClaims)  accessTokenString, err := accessToken.SignedString([]byte(s.jwtSecret))  if err != nil {  return nil, errors.WithMessage(err, "signed string")  }  refreshClaims := domain.JWTClaims{  UserId: userId,  Role: role,  RegisteredClaims: jwt.RegisteredClaims{ |

Продолжение листинга Б.6

|  |
| --- |
| ExpiresAt: jwt.NewNumericDate(time.Now().Add(s.refreshTokenExpiry)),  IssuedAt: jwt.NewNumericDate(time.Now()),  NotBefore: jwt.NewNumericDate(time.Now()),  Issuer: "api-gateway",  Subject: string(rune(userId)),  ID: uuid.New().String(),  },  }  refreshToken := jwt.NewWithClaims(jwt.SigningMethodHS256, refreshClaims)  refreshTokenString, err := refreshToken.SignedString([]byte(s.jwtSecret))  if err != nil {  return nil, errors.WithMessage(err, "signed string")  }  return &domain.TokenPair{  AccessToken: accessTokenString,  RefreshToken: refreshTokenString,  ExpiresIn: int64(s.accessTokenExpiry.Seconds()),  }, nil  } |

Листинг Б.7 – Middleware для защищенных эндпоинтов (разработано автором)

|  |
| --- |
| func (m \*AuthMiddleware) Authenticate() gin.HandlerFunc {  return func(c \*gin.Context) {  authHeader := c.GetHeader("Authorization")  if authHeader == "" {  c.JSON(http.StatusUnauthorized, gin.H{"error": "Authorization header is required"})  c.Abort()  return  }  parts := strings.Split(authHeader, " ")  if len(parts) != 2 || parts[0] != "Bearer" {  c.JSON(http.StatusUnauthorized, gin.H{"error": "Authorization header format must be Bearer {token}"})  c.Abort()  return  }  refreshToken := jwt.NewWithClaims(jwt.SigningMethodHS256, refreshClaims)  refreshTokenString,err:=refreshToken.SignedString([]byte(s.jwtSecret))  if err != nil {  return nil, errors.WithMessage(err, "signed string")  } |

Продолжение листинга Б.7

|  |
| --- |
| return &domain.TokenPair{  AccessToken: accessTokenString,  RefreshToken: refreshTokenString,  ExpiresIn: int64(s.accessTokenExpiry.Seconds()),  }, nil  } |

Листинг Б.8 – Функция с рекомендациями по расходам (разработано автором)

|  |
| --- |
| const generateSettlementRecommendations = (balances: UserBalance[]) => {  const sortedBalances = [...balances]  .filter(b => Math.abs(b.balance) > 0.01)  .map(b => ({...b}));    type Recommendation = { from: string; to: string; amount: number };  const recommendations: Recommendation[] = [];    while (sortedBalances.length > 1) {  sortedBalances.sort((a, b) => a.balance - b.balance);    const debtor = sortedBalances[0]; // Должник (отрицательный баланс)  const creditor = sortedBalances[sortedBalances.length - 1]; // Кредитор (положительный баланс)    const transferAmount = Math.min(Math.abs(debtor.balance), creditor.balance);    if (transferAmount > 0.01) {  recommendations.push({  from: debtor.username,  to: creditor.username,  amount: transferAmount  });  }    debtor.balance += transferAmount;  creditor.balance -= transferAmount;    // Удаляем пользователей с балансом, близким к нулю  const updatedBalances = sortedBalances.filter(b => Math.abs(b.balance) > 0.01);  sortedBalances.length = 0;  sortedBalances.push(...updatedBalances);  }    return recommendations;  }; |

Листинг Б.9 – Функция для построения иерархической структуры задач (разработано автором)

|  |
| --- |
| // Предотвращение циклических зависимостей  const checkForCycles = (tasks: TaskResponse[]): TaskResponse[] => {  const result: TaskResponse[] = [];  const visited = new Set<number>();  tasks.forEach(task => {  visited.clear();  let currentTask = task;  let currentParentId = normalizeParentId(currentTask);  let hasCycle = false;  visited.add(currentTask.id);  while (currentParentId !== null && !hasCycle) {  if (visited.has(currentParentId)) {  hasCycle = true;  break;  }  const parentTask = tasks.find(t => t.id === currentParentId);  if (!parentTask) break;  visited.add(currentParentId);  currentTask = parentTask;  currentParentId = normalizeParentId(currentTask);  }  if (hasCycle) {  result.push({ ...task, parent\_id: null });  } else {  result.push({ ...task, parent\_id: normalizeParentId(task) });  }  });  return result;  };  // Проверяем циклические зависимости и нормализуем parent\_id  const normalizedTasks = checkForCycles(tasksList).map(task => ({  ...task,  parent\_id: normalizeParentId(task)  }));    // Создаем карту задач для быстрого доступа  const taskMap = new Map<number, HierarchicalTask>();  // Инициализируем иерархическую структуру для всех задач  normalizedTasks.forEach(task => {  taskMap.set(task.id, {  ...task,  children: [],  isExpanded: false  });  }); |