

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc193244193)

[ОСНОВНАЯ ЧАСТЬ 8](#_Toc193244194)

[1. Исследовательский раздел 8](#_Toc193244195)

[1.1 Анализ существующих решений 8](#_Toc193244196)

[1.2 Определение требований к решению 9](#_Toc193244197)

[1.3 Выбор инструментов и методов создания приложения 11](#_Toc193244201)

[1.4 Выбор методологии и технологий концептуального моделирования клиент-серверной системы 13](#_Toc193244202)

[1.5 Постановка задач к проектированию и разработке приложения 14](#_Toc193244203)

[Вывод к разделу 1 15](#_Toc193244204)

[2 Проектный раздел 16](#_Toc193244205)

[2.1 Проектирование адаптированной модели жизненного цикла системы 16](#_Toc193244206)

[2.2 Проектирование архитектуры системы 16](#_Toc193244207)

[2.2. Проектирование основного модуля 19](#_Toc193244208)

[2.3. Проектирование клиентской части системы 21](#_Toc193244209)

[2.4. Проектирование схемы базы данных 22](#_Toc193244210)

[Вывод к разделу 2 24](#_Toc193244211)

[3 Технологический раздел 25](#_Toc193244212)

[3.1 Разработка модуля уведомлений 25](#_Toc193244213)

[3.2 Тестирование модуля уведомлений 25](#_Toc193244214)

[Вывод к разделу 3 28](#_Toc193244215)

[ЗАКЛЮЧЕНИЕ 29](#_Toc193244216)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 30](#_Toc193244217)

[Приложение А 32](#_Toc193244218)

# 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 исследованию 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 и инструменты 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.

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)

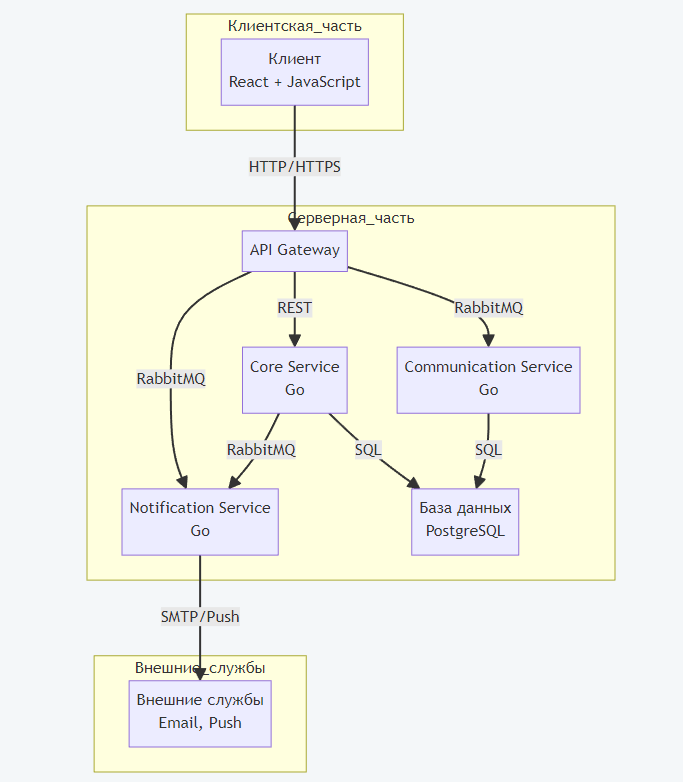


Рисунок 2.1 – Диаграмма компонентов системы (разработано автором)

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.

2.2. 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.

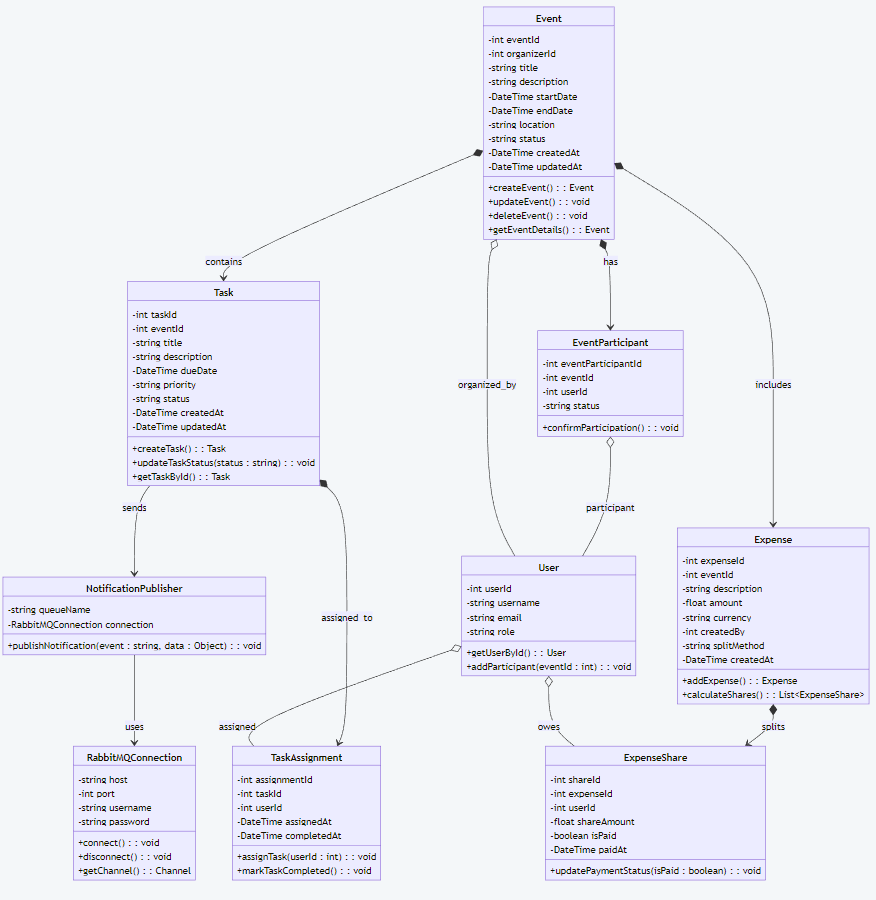


Рисунок 2.2 – Диаграмма классов основного сервиса (разработано автором)

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.

2.3. 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.

2.4. 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 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"+ |

Продолжение листинга 3.1

|  |
| --- |
| "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.

3.2 Тестирование модуля уведомлений

Создадим задачу на создания нового уведомления. В теле задачи укажем тип произошедшего события, на которое планируется нотификация, идентификатор события, название и почта для рассылки. Задача помешается в очередь «/notifications» брокера сообщений RabbitMQ (Рисунок 3.1).

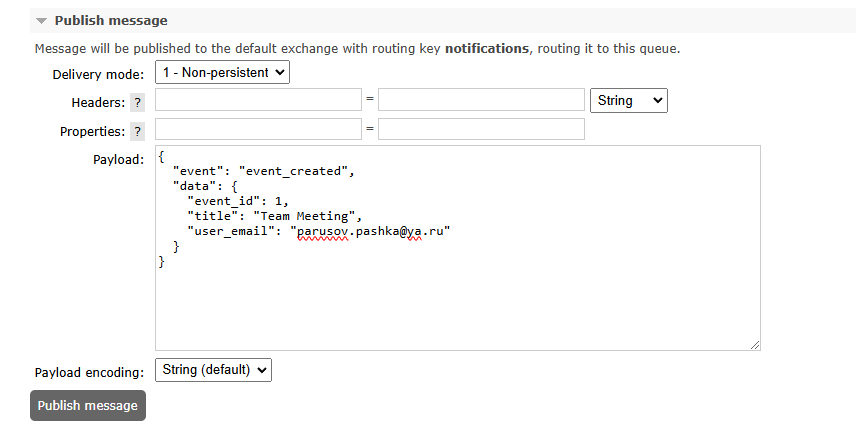


Рисунок 3.1 – Публикация сообщения в очередь

Теперь посмотрим на содержимое тестового SMTP-сервера и увидим, что в него пришло письмо с содержимым задачи. Письмо указано на рисунке 3.2.

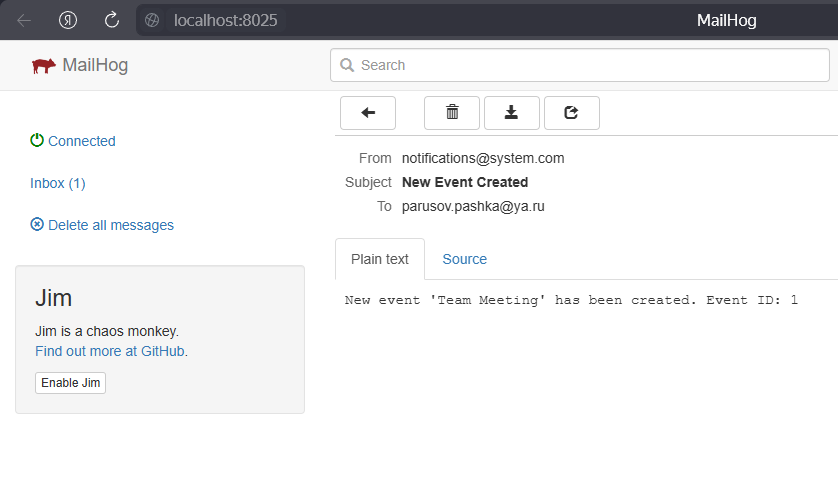


Рисунок 3.2 – Письмо, пришедшее в Mailhog

Результат полностью соответствует данным задачи отправленной в брокер.

Вывод к разделу 3

В рамках данного раздела была выполнена разработка модуля уведомлений (Notification Service) для системы управления групповыми мероприятиями. Модуль реализован на языке Go с использованием RabbitMQ для получения сообщений и SMTP для отправки email-уведомлений, что позволило обеспечить оперативное информирование пользователей о ключевых событиях.

Тестирование модуля подтвердило его работоспособность: устранены ошибки, связанные с отправкой email, а также проверена корректность интеграции с другими компонентами системы. Модуль полностью соответствует поставленным требованиям, обеспечивая надёжное и своевременное уведомление пользователей.

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)

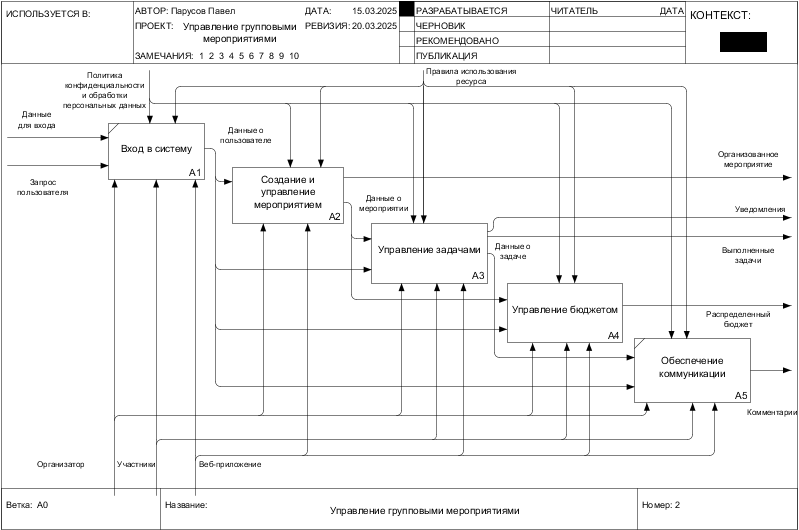


Рисунок А.2 – Декомпозиция блока А0 (разработано автором)

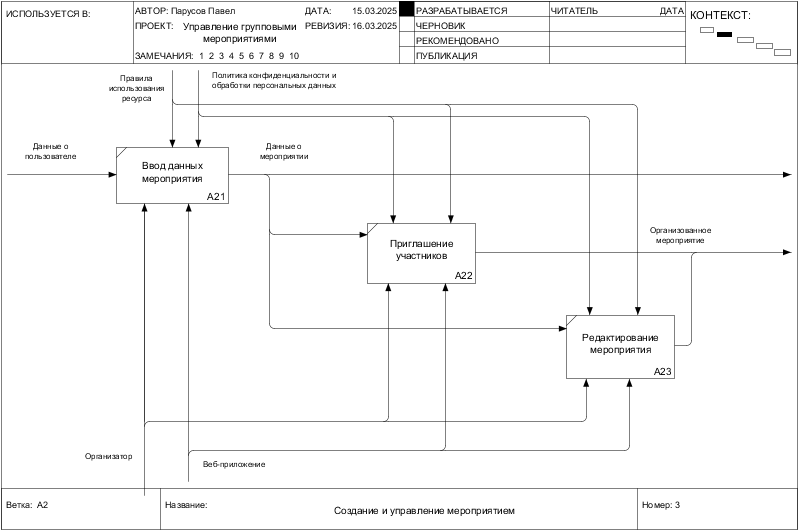


Рисунок А.3 – Декомпозиция блока А2 (разработано автором)

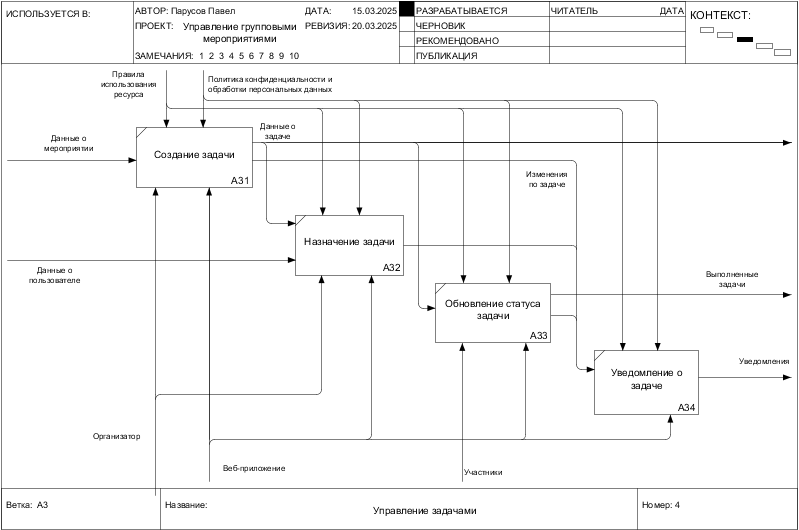


Рисунок А.4 – Декомпозиция блока А3 (разработано автором)

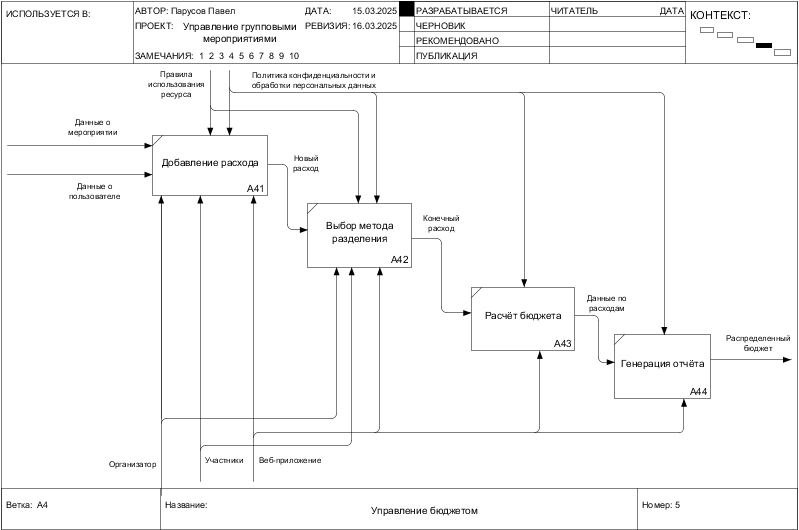


Рисунок А.5 – Декомпозиция блока А4 (разработано автором)