МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

Шал Гүлминұр Нұрмұқанқызы

# Разработка вэб–приложения для создания личных мероприятий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**
к дипломному проекту

Специальность 6B06102 – Computer Science

Алматы 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

**ДОПУЩЕН К ЗАЩИТЕ**
Заведующий кафедрой ПИ
канд. тех. наук, ассоц.профессор
_____Ф.Н.Абдолдина
«10» 06 2025 г.

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

На тему: «Разработка вэб – приложения для создания личных мероприятий»

по специальность 6B06102 – Computer Science

Выполнил Шал Гүлминұр Нұрмұқанқызы

Рецензент
PhD, ассоц. проф-исследователь
АУЭС им. Г. Дукеева
_____ Тойбаева Ш. Д.
"04" 06 2025 г.

Научный руководитель
Кандидат технических наук
Профессор
_____ Кубеков Б. С.
"04" июня 2025 г.

Алматы 2025

Қолтаңбаны растаймын
Подпись заверяю
Батырбекова А.Д.
қызметі аты-жөні
«04» 06 2025 ж.

Персоналды басқару жөніндегі департамент
Департамент по управлению персоналом

РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

**УТВЕРЖДАЮ**
Заведующий кафедрой ПИ
канд. тех. наук, ассоц.профессор
_____ Ф.Н.Абдолдина
«04» 06 2025 г.

**ЗАДАНИЕ**
**на выполнение дипломного проекта**

Обучающемуся: *Шал Гүлминұр Нұрмұқанқызы*
Тема: Разработка вэб – приложения для создания личных мероприятий

Утверждена приказом проректора по академической работе:

№ 1804-до

| | |
|---|---|
| | *от «25» ноября 2024 г.* |
| Срок сдачи законченного проекта | «04» июня 2025 г. |

Исходные данные к дипломному проекту:
*А) Анализ предметной области;*
*Б) Разработка технического задания;*
*В) Проектирование вэб–приложения;*
*Г) Выбор технологий для разработки;*
*Д) Разработка функционала серверной и клиентской частей;*
*Е) Тестирование вэб–приложения.*
Перечень подлежащих разработке в дипломном проекте вопросов: (с точным указанием обязательных чертежей): *представлены 13 слайдов презентации.*
Рекомендуемая основная литература: *из 8 наименований.*

**ГРАФИК**

подготовки дипломного проекта

| Наименование разделов, перечень разрабатываемых вопросов | Сроки представления научному руководителю и консультантам | Примечание |
|---|---|---|
| 1. Исследование предметной области и анализ существующих решений | 06.12.24 – 12.12.24 | Выполнено |
| 2. Постановка задачи, моделирование разрабатываемой системы | 13.12.24 – 19.12.24 | Выполнено |
| 3. Проектирование системы и выбор технологий разработки | 20.12.24 – 10.01.25 | Выполнено |
| 4. Разработка функционала серверной и клиентских частей | 14.01.25 – 25.03.25 | Выполнено |
| 5. Тестирование вэб-приложения | 10.04.25 – 23.05.25 | Выполнено |

**Подписи**

консультантов и нормоконтролера на законченную дипломный проект
с указанием относящихся к ним разделов проекта

| Наименования разделов | Консультанты, Ф.И.О. (уч. степень, звание) | Дата подписания | Подпись |
|---|---|---|---|
| Программное обеспечение | Шаханов Ә. Р. преподаватель, магистр | 30.05.25 | |
| Нормоконтролер | Имаматдинова К. Ф. преподаватель, магистр | 04.06.2025 | |

Научный руководитель _____ Кубеков Б. С.

Задание принял к исполнению обучающийся _____ Шал Г. Н.

Дата «04» июня 2025 г.

# АҢДАТПА

Бұл дипломдық жобаның мақсаты жеке іс-шараларды ұйымдастыруға арналған веб-қосымшаны әзірлеу. Қосымша пайдаланушыларға тапсырмаларды басқару, бюджетті жоспарлау және іс-шараға қатысушылар арасындағы өзара әрекеттесуді қамтамасыз ету бойынша кешенді мүмкіндіктер береді. Әзірленген жүйе пайдаланушылардың кең ауқымына: жеке тұлғаларға, шағын және орта бизнес өкілдеріне, сондай-ақ оқиға индустриясы саласында жұмыс істейтін жеке кәсіпкерлерге бағытталған.

Осы мақсатқа жету үшін жоба аясында пәндік салаға егжей-тегжейлі талдау жасалды және осыған ұқсас шешімдер зерттелді. Осы талдау негізінде клиент-сервер моделіне негізделген жүйенің архитектурасы жасалды. MERN (MongoDB, Express.js, React, Node.js) технологиялық стек ретінде таңдалды. React, Node.js). Әзірлеу кезінде объектіге бағытталған бағдарламалау принциптері және Model-View-Controller (MVC) архитектуралық үлгісі қолданылды.

Нәтижесінде функционалды пайдаланушы интерфейсі құрылды және жүйенің негізгі модульдері енгізілді, соның ішінде пайдаланушыларды тіркеу және авторизациялау, іс-шаралар құру, тапсырмаларды басқару, мердігерлермен өзара әрекеттесу, чат жүйесі және кездесулерді жоспарлау.

# АННОТАЦИЯ

Целью данного дипломного проекта стала разработка веб-приложения, предназначенного для организации личных мероприятий. Приложение призвано предоставить пользователям комплексные возможности по управлению задачами, планированию бюджета и обеспечению взаимодействия между участниками события. Разрабатываемая система ориентирована на широкий круг пользователей: частных лиц, представителей малого и среднего бизнеса, а также индивидуальных предпринимателей, действующих в сфере событийной индустрии.

Для достижения поставленной цели в рамках проекта был выполнен детальный анализ предметной области и изучены существующие аналогичные решения. На основе этого анализа спроектирована архитектура системы, основанная на клиент-серверной модели. В качестве технологического стека выбран MERN (MongoDB, Express.js, React, Node.js). При разработке применялись принципы объектно-ориентированного программирования и архитектурный паттерн Model-View-Controller (MVC).

В результате был создан функциональный пользовательский интерфейс и реализованы ключевые модули системы, включая регистрацию и авторизацию пользователей, создание мероприятий, управление задачами, взаимодействие с подрядчиками, систему чатов и планирование встреч.

# ABSTRACT

The purpose of this graduation project was to develop a web application designed for organizing personal events. The application is designed to provide users with comprehensive task management, budget planning, and interaction between event participants. The system being developed is aimed at a wide range of users: individuals, representatives of small and medium-sized businesses, as well as individual entrepreneurs operating in the event industry.

To achieve this goal, the project carried out a detailed analysis of the subject area and examined existing similar solutions. Based on this analysis, the system architecture based on the client-server model is designed. The technology stack is MERN (MongoDB, Express.js, React, Node.js). The principles of object-oriented programming and the Model-View-Controller (MVC) architectural pattern were applied during the development.

As a result, a functional user interface was created and key system modules were implemented, including user registration and authorization, event creation, task management, interaction with contractors, a chat system, and meeting scheduling.

**СОДЕРЖАНИЕ**

# Введение

В истории человечества организация мероприятий играла важнейшую роль. Входя в исторический экскурс, в разные эпохи мероприятия были инструментами легитимации власти, подчинение сознания иерархии, а с наступлением нового времени главной идеей организации мероприятий стали демонстрация прогресса и просвещения, а наступлением эпохи индустриализации мероприятия стали инструментом групповой мобилизации людей.

В наше же время главным запросом в организации мероприятий стало создание опыта. Так как в современном обществе люди хотят быть сопричастными к событиям, в которых убеждения формируются через вовлечения. Для организации таких событий нужны современные инструменты закрывающие потребности целевой аудитории, т.е. человека и компаний, предоставляющих свои услуги в сегменте культурно–событийного бизнеса. События от личных семейных торжеств до корпоративных мероприятий требуют планирования, и координации действ всех участников в организации какого–либо вида мероприятия. Необходимость в таких инструментах в свете нынешнего темпа жизни становятся все более актуальным.

Целью дипломного проекта является разработка вэб–приложения для организации мероприятий с функциями управления задачами и бюджетом, которое повысит эффективность планирования событий и улучшит коммуникацию между участниками.

В соответствии с целью определены следующие задачи:

– проанализировать существующие решения в области организации мероприятий и выявить их достоинства и недостатки;

– определить требования к разрабатываемой системе на основе анализа потребностей потенциальных пользователей;

– спроектировать архитектуру приложения и структуру базы данных;

– разработать пользовательский интерфейс;

– реализовать ключевые функциональные модули системы.

# 1 Исследовательско –технологический раздел

## 1.1 Анализ предметной области

В наше время организация различных событий является важной частью как профессиональной деятельности, так и личной жизни людей. От семейных вечеров до корпоративных мероприятий – все эти социально–культурные события требуют взаимодействия и сотрудничества между участниками. Поэтому необходимо использовать эффективные методы коммуникации и делегирования задач для успешной координации всех процессов.

Событийный менеджмент как дисциплина интегрирован в более обширную индустрию культурно–событийной деятельности. Она включает в себя 6 базовых и 15 сопутствующих категорий сервисных предложений (см. Рисунок 1.1). Следует отметить, что процесс формирования и реализации подобных мероприятий потенциально инкорпорирует каждый из обозначенных типов услуг.

К примеру, по данным Бюро национальной статистики по итогам анализа за 2024 год количество заключённых браков в Казахстане составило123,6 тыс. [1]. Можно предположить, что каждый 3–й брак сопровождался последующим мероприятием. На подобные торжественные мероприятия как правило требуется персонал сферы обслуживания, место проведения, музыканты, ведущие и т.д.

В итоге, в случае самостоятельной организации мероприятий остро встают вопросы поисков, сравнений, переговоров, а в случае форс–мажорных ситуаций и вовсе может потребоваться замена подрядчика, оказывающего определенный вид услуг.

| | | ЮЛ | ИП | Итого | ДЕЙСТВУЮЩИХ |
|---|---|---|---|---|---|
| **Основные** | Организация конференций и торговых выставок<br>Театральная деятельность<br>Концертная деятельность<br>Деятельность, способствующая проведению культурно-зрелищных мероприятий<br>Прочие виды деятельности по организации отдыха и развлечений<br>Иная профессиональная, научная и техническая деятельность, не включенная в другие группировки | **7 260** | **7 752** | **15 012** | **11 610** |
| **Смежные** | Предоставление гостиничных услуг с ресторанами для официальных мероприятий<br>Доставка готовой пищи на заказ<br>Деятельность в сфере звукозаписи и издания музыкальных произведений<br>Деятельность рекламных агентств<br>Специализированная дизайнерская деятельность<br>Деятельность в области фотографии<br>Деятельность туристских операторов<br>Прочие услуги по бронированию и сопутствующая деятельность<br>Деятельность цирков<br>Художественное и литературное творчество<br>Деятельность концертных и театральных залов<br>Деятельность музеев<br>Деятельность ботанических садов и зоопарков<br>Деятельность развлекательных и тематических парков<br>Деятельность кукольных театров | 7 032 | 14 199 | 21 231 | 16 679 |
| **ВСЕГО** | | **14 292** | **21 951** | **36 243** | **28 289** |

Рисунок 1.1 – Виды услуг индустрии культурно – событийной деятельности

Что касательно сферы предоставления своих услуг агентств по организации мероприятий, в ходе проведенного интервью международным ивент–агентством «EZ Solutions», директор Казахстанской Ассоциации Маркетинга Анелия Мухамедкаримова акцентировала внимание на потребности многих ивент–агентств в доступе к дифференцированной базе потенциальных клиентов [2].

Исходя из вышеизложенного мы можем определить, что целевой аудиторией разрабатываемой системы являются рядовые частные лица, малый и средний бизнес и индивидуальные предприниматели чья работа напрямую связана с культурно–событийной индустрией.

## 1.2 Анализ существующих решений

Поиск аналогов разрабатываемого, в данном дипломном проекте, вэб – приложения проводился по трем основным параметрам определённых посредством анализа предметной области:

Готовые платформы для организации мероприятий, такие как Eventbrite и MeetUp. Данные аналоги имею схожий друг с другом функционал возможностей:

– создание страницы мероприятия, где можно выбрать даты проведения, место и описание мероприятий;

– регистрировать участников;

– рассылка приглашений для участников мероприятия.

Касательно ограничений, стоит отметить, что на данных платформах отсутствует возможность участников быть вовлеченными в процесс организации планируемых событий, т.е. они могут быть только в роли гостей. Так же отсутствует торговая площадка для поиска подрядчиков по предоставлению услуг, инструменты финансового распределения, а также ограниченная коммуникация между участниками планируемых мероприятий. Закрытие потребностей пользователей по вышеописанным ограничениям могут быть осуществлены только посредством сторонних сервисов.

Второй параметр, определенный для аналогов это управление проектами – сервисы Trello и Asana. Данные платформы обладают так же схожей функциональностью, но с некоторыми ограничениями:

– возможность добавления коллабораторов;

– назначение задач для коллабораторов;

– комментирование задач.

Данные сервисы не заточены под специфику проведения мероприятий, отсутствует возможность назначить место проведения, нет возможности

распределения бюджета по задачам для выполнения, возможность коммуникации с участниками ограничены только комментариями для задач.

Третий параметр определения аналогов – это рабочие пространства такие как Microsoft Teams и Slack. Данные рабочие пространства подходят больше всего под определения аналогов исходя из требуемых функций. В них можно

– создавать как личные чаты, так и групповые,

–интегрированы календари в которых отмечаются запланированные события

– можно закреплять сообщения для постоянного отображения в чатах.

Но и в этих сервисах есть свои недостатки, для отслеживания задач нужно интегрировать среды управления проектами вручную, отсутствует торговая площадка для размещения услуг для подрядчиков, детали планируемых событий не связаны в единую сущность, т.е. дата, место и процесс организации содержатся либо в чатах, либо в объявлениях, документах, и.т.д.

Выше представленные сервисы ориентированы на отдельные аспекты создания и организации и не предоставляют комплексного решения поставленных задач. В некоторых отсутствуют инструменты для финансового контроля, и торговая площадка, а в некоторых не интегрированы с системы управления задачами и планированием.

Существующие решения не обеспечивают полноценное взаимодействие между участниками процесса организации мероприятия. К тому же, в некоторых из представленных решений отсутствует кроссплатформенность, они требую либо установки дополнительного программного обеспечения, либо имеют десктопную версию, в которую нельзя использовать на смартфонах.

## 1.3 Постановка задач

На основании вышеизложенного в разделе анализа существующих решений можно сделать вывод, что разработка вэб–приложения для создания мероприятий нуждается в возможностях для координации и инструментах менеджмента, что поспособствовало бы оптимизации процессов планирования, повышению их операционной эффективности. А для подрядчиков, оказывающих услуги связанных с событийной–развлекательной сферой, обеспечило бы доступ к базе потенциальных клиентов.

Учитывая выявленные недостатки проанализированных существующих решений, разработка собственного вэб – приложения требует реализации следующих функций:

Исходя из списка требований к базовому функционалу разрабатываемого, в данном дипломном проекте, вэб-приложения были поставлены следующие функции, в которых требуется создать:

– разработать единый механизм аутентификации и управления аккаунтом

– объект мероприятие с полями заполнения информации по событиям: название, описание, теги, дата начала и окончании события, место проведения, статус, бюджет и добавления участников;

– систему ролей таких как пользователь и подрядчик;

– возможность просмотра всех нужных деталей таких как информация о мероприятиях, чат;

– при добавлении участников связать их с конкретным мероприятием и созданными задачами;

– торговую площадку с функциями каталога и фильтром по категориям

– функционал для подрядчиков для создания карточки услуг

– реализацию процесса публикации на торговой площадке и выбор пользователем услуги подрядчика;

– функцию добавления подрядчика в качестве участника мероприятия;

– резервирование бюджета при назначении задачи, списание средств после подтверждения выполнения;

– систему групповых чатов;

– обновление статуса мероприятия на основе прогресса задач;

## 1.4 Выбор технологии проектирования

Метод объектно–ориентированного проектирования является оптимальным подходом к реализации проектируемого вэб-приложения, поскольку он позволяет структурировать систему, разделяя её на автономные, но взаимодействующие компоненты.

Одним из ключевых преимуществ ООП является инкапсуляция, позволяющая представить каждую сущность системы, такую как пользователь, подрядчик, услуга, событие или задача, в виде объекта с определёнными свойствами и методами. Такой подход изолирует логику работы отдельных элементов, повышая управляемость системы. Композиция обеспечивает объединение различных объектов в сложные структуры. Например, событие может включать в себя задачи, подрядчиков, бюджет и индикатор выполнения, что создаёт целостную модель взаимодействия.

Принцип повторного использования позволяет задействовать одни и те же объекты в различных частях системы. Подрядчик, назначенный на одно событие,

может использоваться и в других, что минимизирует дублирование кода и упрощает поддержку. Модульность системы способствует автономной разработке её функциональных блоков, таких как авторизация, управление событиями, система чатов и поиск. Благодаря этому отдельные компоненты могут изменяться и тестироваться независимо.

Полиморфизм обеспечивает удобную работу с разными категориями подрядчиков, такими как фотографы или организаторы мероприятий, позволяя им наследовать общие свойства и методы с возможностью добавления специфической логики. Это делает обработку различных типов объектов более универсальной. Эффективное управление состоянием системы становится возможным за счёт обновления параметров событий, включая бюджет и статус выполнения задач, что позволяет оперативно отслеживать изменения.

Расширяемость системы достигается благодаря возможности добавления новых объектов или методов без необходимости внесения значительных изменений в существующую архитектуру. Это позволяет адаптировать программное обеспечение к изменяющимся требованиям, добавляя новые категории подрядчиков или типы задач. Таким образом, объектно – ориентированное проектирование способствует созданию гибкой и масштабируемой системы, полностью соответствующей требованиям рассматриваемого сценария.

## 1.5 Клиент–серверная архитектура

Архитектура клиент–сервер – это вычислительная модель, в которой сервер размещает, доставляет и управляет большинством ресурсов и услуг, которые будут потребляться клиентом. Этот тип архитектуры имеет один или несколько клиентских компьютеров, подключенных к центральному серверу через сеть или интернет–соединение.

В сравнении с одноранговой архитектурой (P2P) у трехуровневой архитектуры серверная часть может масштабироваться независимо от клиентов, что позволяет эффективно распределять ресурсы при росте нагрузки. Например, добавление серверов или использование кластеризации улучшает производительность без изменения клиентского кода. Разделение ответственности между клиентом и сервером упрощает горизонтальное масштабирование.

Клиент–серверная архитектура является оптимальным выбором для реализации моделирования архитектуры данного проекта (Рисунок 1.5.1), так как она обеспечивает чёткое разделение функциональности между клиентской и серверной частями системы. Такое разграничение способствует удобству

разработки, тестирования и поддержки, позволяя каждой стороне выполнять строго определённые задачи.

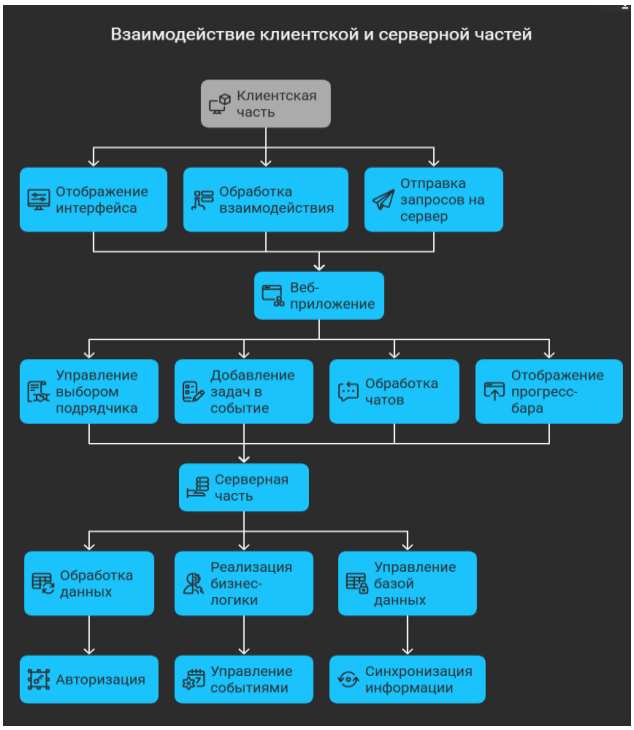



Рисунок 1.5.1 – Клиент–серверное взаимодействие проектируемой системы

Клиентская часть отвечает за отображение интерфейса, обработку взаимодействия с пользователем и отправку запросов на сервер. Например, веб–приложение, управляет выбором подрядчика, добавлением задач в событие, обработкой чатов и отображением прогресс–бара.

Серверная часть, в свою очередь, выполняет обработку данных, реализацию бизнес–логики и управление базой данных, обеспечивая авторизацию, управление событиями и синхронизацию информации о подрядчиках и категориях.

## 1.6 Стек технологий MERN

Для разработки веб–приложения был выбран стек MERN, включающий MongoDB, Express.js, React и Node.js.

Язык JavaScript был выбран за его универсальность – он мультипарадигменный, поддерживает объектно – ориентированное программирование, а также сочетает в себе императивные и декларативные

подходы. Благодаря этим качествам JavaScript отлично подходит для реализации гибких и масштабируемых приложений.

Система управления базами данных MongoDB это документно-ориентированная NoSQL- СУБД, которая хранит данные в гибких бианарных JSON- документах (BSON), она позволяет делать произвольные структуры событий без необходимости изменения схемы базы данных.

Express.js минималистичный и гибкий фреймворк, который позволяет создать надежный API для коммуникации между клиентской частью и базой данных. Его высокая производительность и поддержка асинхронной обработки – делают его подходящим выбором для ключевого фактора в разработке многозадачных приложений.

Виртуальный DOM в React обеспечивает быструю реакцию интерфейса на действия пользователя. Он показывает на 28% более быструю отрисовку интерфейса по сравнению с традиционным jQuery при работе со сложными формами делегирования задач [4]. React позволяет создавать адаптивные интерфейсы, а технологии PWA обеспечат работу приложения как на десктопе, так и на мобильных устройствах. Обеспечивает на 33% более высокую конверсию по сравнению с другими [4].

Неблокирующая модель ввода–вывода Node.js обеспечивает эффективную обработку множества одновременных операций.

В результате анализа эталонных тестов–сравнений между Node.js, PHP, Python наилучшую производительность при увеличении числа пользователей показал Node.js, до 100, достигая 3703,5 (Рисунок 1.6.1) запросов в секунду. Среднее время выполнения запроса – 0,27 мс, в то время как Python показал 500 запросов в секунду, с максимальным значением 559,42 и 1,788 мс, а PHP достиг 2977,54 запросов в секунду при 100 пользователях, а среднее время выполнения запроса составило 0,336 мс, что доказывает то, что Node.js способен обслуживать значительно больше пользователей по сравнению с PHP и Python [5].

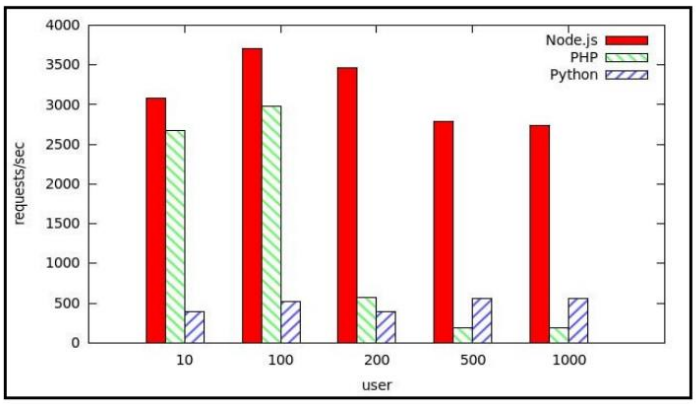



Рисунок 1.6.1 – Модуль расчета среднего количества запросов в секунду

Так как огромную роль играет и приспособляемость к работе с приложениями, интенсивно использующими Ввод/Вывод (Input/Output), Node.js, PHP и Python были протестированы в модуле "Выбор операции БД" (Рисунок 1.6.2).

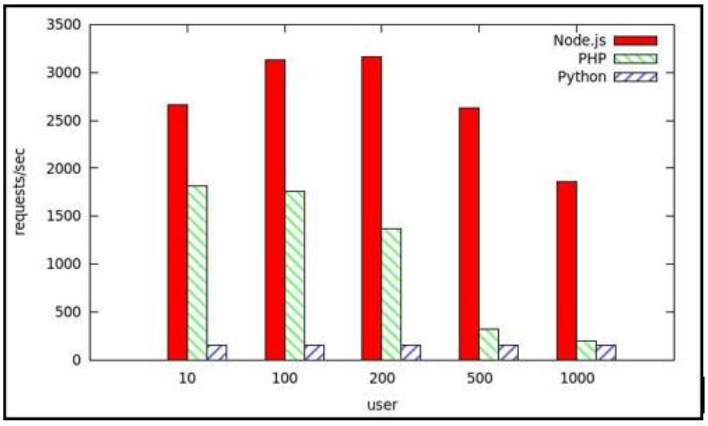



Рисунок 1.6.2 – Модуль "Выбор операции БД "

Максимальное значение "среднего количества запросов в секунду" у Node.js составляет 3164,46, что в 20 раз больше, чем у Python–Web, и в 2 раза больше, чем у PHP. Кроме того, пиковое значение модуля "Выбор операции БД " у Node.js не сильно отличается от модуля "Среднего количества запросов в секунду ". Среднее значение запросов в секунду у Python держится около 150 и остается стабильным по мере роста пользователей. А вот у PHP "среднее количество запросов в секунду" замедляется, а "среднее время выполнения одного запроса" увеличивается. Это доказывает, что Node.js больше подходит для приложений с большим объемом операций ввода–вывода, в то время как PHP применим для небольших веб–сайтов.

Основываясь на представленных результатах тестов, можно сделать четкий вывод в пользу Node.js как наиболее эффективного решения для веб–приложений, особенно тех, которые интенсивно используют операции ввода–вывода [6].

# 2 Проектно – экспериментальный раздел

## 2.1 Унифицированный язык моделирования

Унифицированный язык моделирования (UML) – это семейство графических нотаций, в основе которого лежит единая метамодель. Он помогает в описании и проектировании программных систем, в особенности систем, построенных с использованием объектно–ориентированных технологий [8].

Ниже приведены функциональные требования к разрабатываемому вэб-приложению:

– пользователь регистрируется на платформе, переходя с главной страницы по кнопке «Вход». На экране регистрации он вводит имя, адрес электронной почты, пароль, предполагаемый бюджет и выбирает свою роль – обычный пользователь или подрядчик. Система требует от пользователя подтверждение согласия на обработку персональных данных, прежде чем завершить регистрацию. Без этого шага регистрация невозможна. После создания аккаунта пользователь входит в систему через форму авторизации. Он указывает ранее введённые адрес электронной почты и пароль, после чего система перенаправляет его в личный кабинет;

– система отображает в личном кабинете основные элементы навигации: кнопку создания нового мероприятия, календарь с запланированными событиями, ссылку на каталог подрядчиков, вкладки для переключения между активными и прошедшими мероприятиями, а также кнопки для создания встреч и чатов. Пользователь просматривает свои мероприятия в календаре и отслеживает статус каждого события в зависимости от времени проведения. Он может оперативно переходить к конкретному мероприятию или использовать календарь для планирования. Пользователь также использует ссылку в личном кабинете для перехода в каталог подрядчиков, где он может выбирать и заказывать услуги;

– пользователь создаёт мероприятие, заполняя форму с обязательными полями: название, описание, дата, место, предполагаемый бюджет, метки и список участников. Он может включать в участников других пользователей и подрядчиков. После создания мероприятия пользователь получает возможность редактировать его. Он изменяет параметры события, добавляет или исключает участников, а также назначает задачи, указывая описание и сумму, которая будет вычтена из бюджета при выполнении. Система визуализирует выполнение задач с помощью индикатора прогресса;

– когда участник отмечает задачу как выполненную, система обновляет прогресс-бар и автоматически вычитает указанную сумму из общего бюджета мероприятия. Создатель мероприятия управляет его завершением: он может

вручную завершить мероприятие, удалить его полностью или выйти из него, передав контроль другим участникам;

– подрядчик создаёт собственные услуги, которые система размещает в общем каталоге для просмотра другими пользователями. Каждая услуга содержит название и описание, а также может быть использована в мероприятиях. Обычный пользователь фильтрует доступные услуги подрядчиков по названию, чтобы быстро найти нужное предложение. После выбора нужной услуги он нажимает кнопку «Заказать», инициируя заказ. Система автоматически создаёт чат между пользователем и подрядчиком после нажатия кнопки заказа. Это упрощает коммуникацию и позволяет оперативно обсудить детали сотрудничества. Пользователь также имеет возможность добавить подрядчика в список участников любого из своих мероприятий, чтобы назначить ему задачи или включить в обсуждение;

– пользователь организует встречи в рамках мероприятия, указывая конкретную дату и время. Система сохраняет встречи и отображает их в календаре соответствующего события. Пользователь создаёт чаты для общения. Он может инициировать личную переписку с другим пользователем или создать групповой чат, включив в него всех участников конкретного мероприятия;

– система обеспечивает синхронизацию сообщений и отображает историю переписки в реальном времени, что делает взаимодействие между пользователями более прозрачным и эффективным;

– подрядчик принимает участие в мероприятиях как полноценный участник. Он просматривает назначенные ему задачи и взаимодействует с другими участниками. Когда подрядчик выполняет задачу, он отмечает её как завершённую. Система сразу реагирует на это действие, обновляя общий прогресс мероприятия и корректируя бюджет с учётом стоимости выполненной задачи. Система чётко разделяет функциональные возможности между обычными пользователями и подрядчиками, предоставляя каждому только релевантный набор действий. Это помогает избежать путаницы и обеспечивает надёжный контроль доступа;

– завершённые мероприятия система перемещает в отдельный раздел «Прошедшие», где пользователь может просматривать историю событий. Также система синхронизирует изменения в задачах с текущим бюджетом мероприятия и индикатором прогресса, обеспечивая точное и своевременное отображение состояния проекта. Таблица представленная ниже описывает поток событий и определяет, то, что должна делать система когда актор запускает вариант использования «Создание мероприятия» (Таблица 1).

Таблица 1 – Описательная спецификация варианта использования "Создание мероприятия"

| Вариант использования | Создание мероприятия |
|---|---|
| Краткое описание | Вариант использования даёт возможность пользователю создать новое мероприятие в системе. Включает ввод основной информации о мероприятии, даты, места проведения, бюджета и других параметров. |
| Акторы | Пользователь |
| Предусловие | Пользователь авторизован в системе и находится на странице управления мероприятиями. |
| Основной поток событий | Начало варианта использования совпадает с намерением пользователя создать новое мероприятие. Пользователь выбирает функцию "Создать мероприятие". Система отображает форму для ввода информации о мероприятии (название, описание, дата, место проведения, бюджет). Пользователь заполняет обязательные поля и подтверждает создание. Система проверяет корректность введённых данных. Если все данные корректны, система сохраняет новое мероприятие в базе данных, присваивает ему уникальный идентификатор и отображает сообщение об успешном создании. Мероприятие появляется в списке мероприятий пользователя. |
| Альтернативные потоки событий | Если пользователь не заполнил обязательные поля, система отображает сообщение об ошибке и просит заполнить недостающие данные. Пользователь может вернуться к форме и внести исправления. Если при сохранении произошла ошибка (например, сбой соединения с базой данных), система уведомляет пользователя о неудачной попытке создания мероприятия. |
| Постусловие | Если основной вариант использования был успешно завершён, новое мероприятие появляется в системе и доступно для дальнейшего управления. В противном случае состояние системы остаётся неизменным. |

### 2.1.1 Диаграмма прецедентов

Диаграмма прецедентов была создана на основе описательной спецификации прецедента использования, показанной на таблице 2.2. О показывает взаимодействие пользователя и подрядчика с системой при выборе услуг и их дальнейшем назначении на задачи.

Пользователь начинает с открытия вкладки «Подрядчики», где просматривает список доступных исполнителей. Затем он выбирает услугу, что даёт ему возможность начать чат с подрядчиком. В чате стороны обсуждают детали и договариваются о цене.

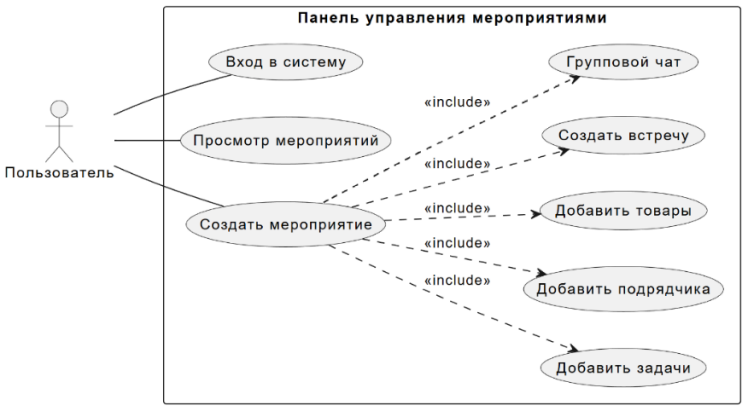



Рисунок 2.1.1 – Диаграмма прецедента создания мероприятия

После согласования условий пользователь добавляет подрядчика в коллабораторы и назначает его исполнителем мероприятия. Затем подрядчик получает конкретную задачу в рамках мероприятия. После её выполнения пользователь подтверждает выполнение и переводит оплату подрядчику.

Связи между прецедентами отражают естественный порядок действий: чат доступен только после выбора услуги, а назначение подрядчика возможно только после согласования условий. Оплата происходит только после завершения работы. Подрядчик в этом процессе участвует как исполнитель, но не может инициировать выбор услуги или назначать себя на задачи.

### 2.1.2 Диаграмма последовательности

Диаграмма последовательности на основе спецификации прецедента «Создание мероприятия», показанным в предыдущем разделе, отображает процесс выбора услуг, взаимодействия с подрядчиком и назначения его на задачи.

Пользователь начинает с открытия панели управления (дашборд) и переходит во вкладку «Подрядчики», где просматривает список доступных исполнителей. Выбрав нужную услугу, он инициирует чат с подрядчиком для обсуждения условий сотрудничества. В чате стороны договариваются о цене и деталях выполнения задачи.

После согласования условий пользователь добавляет подрядчика в коллабораторы мероприятия и назначает его на конкретную задачу. Подрядчик принимает назначение и выполняет задачу, после чего отправляет уведомление о завершении работы.

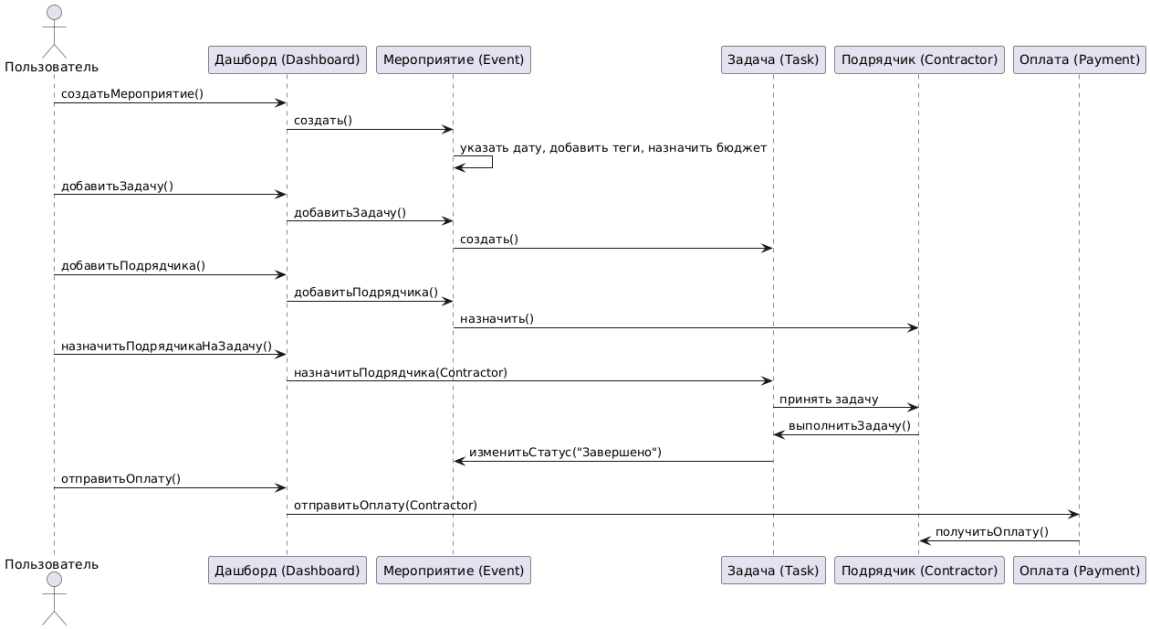



Рисунок 2.1.2.1 – Диаграмма последовательности на основе прецедента создания мероприятия

Пользователь проверяет выполнение задачи и, если всё в порядке, подтверждает оплату. Система обрабатывает перевод средств, и подрядчик получает оплату за выполненную работу.

Вся последовательность строго соблюдает порядок действий, обеспечивая контроль на каждом этапе – от выбора исполнителя до завершения задачи и оплаты.

### 2.1.3 Диаграмма классов

В рамках методологии объектно–ориентированного проектирования следует различать статическую и динамическую модели системы. Диаграмма

классов представляет собой статическую структуру системы, отражая организацию программных сущностей и их взаимосвязи, тогда как диаграммы вариантов использования описывают динамические аспекты функционирования системы в процессе выполнения различных сценариев взаимодействия. В процессе проектирования на основе спецификации варианта использования был с разработки базовой диаграммы классов, которая служит концептуальной моделью предметной области. На последующих этапах, при анализе отдельных вариантов использования, осуществляется последовательное уточнение и дополнение данной диаграммы. Такой итеративный подход позволяет постепенно выявлять необходимые элементы системы и их взаимосвязи. [9]

После полного рассмотрения всех функциональных сценариев, тщательного анализа выявленных требований и соответствующей корректировки модели, формируется итоговая диаграмма классов, полностью соответствующая предъявляемым к системе требованиям. Только по завершении данного этапа проектирования рекомендуется переходить к непосредственной реализации системы в виде программного кода.

В данной системе классы (Рисунок 2.1.3.1) взаимодействуют через четко определенные отношения, образуя логичную структуру управления мероприятиями. Центральным элементом выступает МероприятиеModel, которое содержит всю основную информацию о событии и связано с другими ключевыми сущностями.

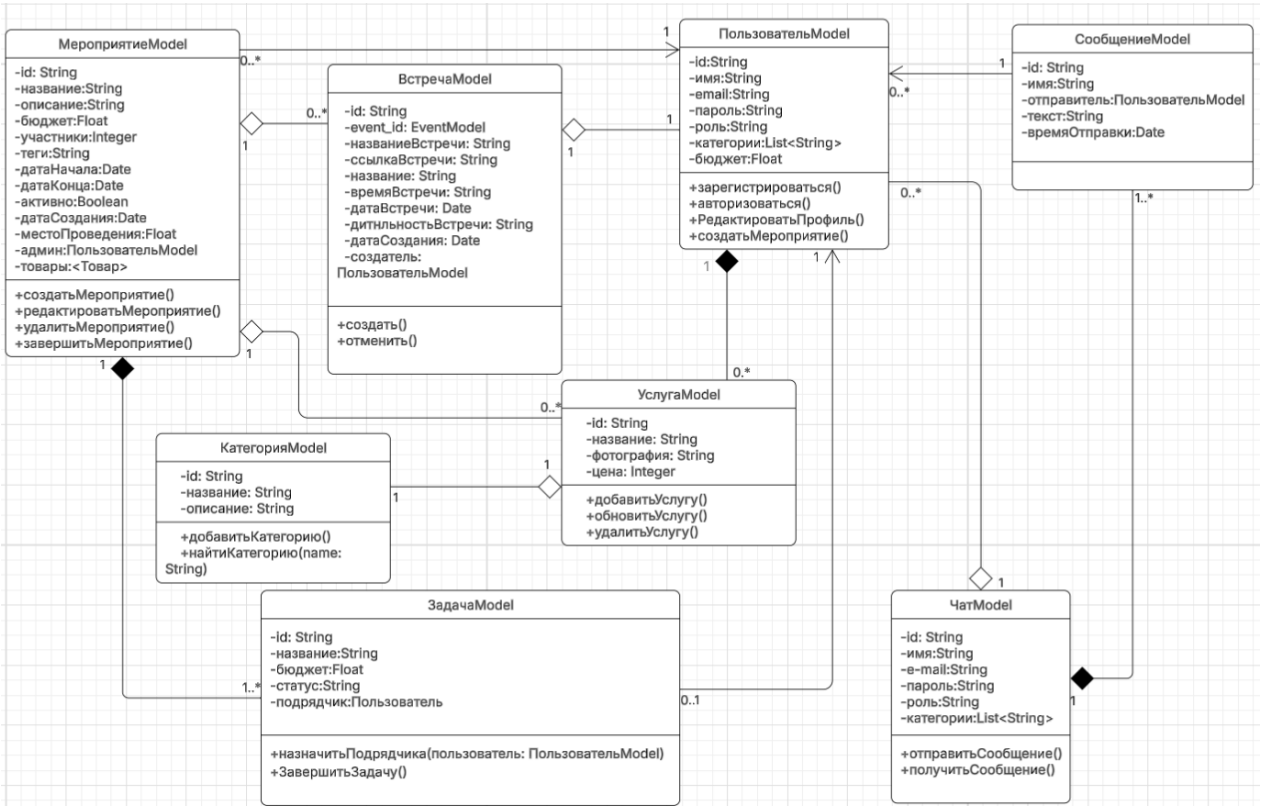



Рисунок 2.1.3.1 – Диаграмма классов (пакет "Model")

Каждое мероприятие создается конкретным пользователем через метод создать Мероприятие(), что устанавливает связь между ПользовательModel и МероприятиеModel. При этом пользователь, указанный в поле "администрация", получает особые права управления этим мероприятием.

Для организации конкретных онлайн встреч в рамках мероприятия используется ВстречаModel. Встречи жестко привязаны к родительскому мероприятию через поле event_id, что создает отношение композиции – встречи не могут существовать независимо от основного мероприятия. Каждая встреча в свою очередь имеет своего создателя из числа пользователей, что добавляет дополнительный уровень связей в системе.

Особую роль играет взаимодействие с задачами через ЗадачаModel. Задачи создаются в контексте конкретного мероприятия и могут быть назначены подрядчикам – специальной категории пользователей. Назначение осуществляется через метод назначитьПодрядчика(), который устанавливает связь между задачей и конкретным пользователем. При этом бюджет задачи связан с общим бюджетом мероприятия, создавая финансовую зависимость.

Для коммуникации между участниками предусмотрен ЧатModel, который аккумулирует сообщения от различных пользователей. Каждое СообщениеModel обязательно содержит ссылку на отправителя, обеспечивая прозрачность переписки.

Отдельный интерес представляет работа с услугами через УслугаModel. Услуги, предоставляемые подрядчиками, могут быть привязаны к мероприятиям через систему категорий. КатегорияModel служит для организации как услуг, так и самих мероприятий, позволяя осуществлять их классификацию и поиск по заданным параметрам.

Часть системы ответственная за финансовый учет реализована через распределение бюджета между мероприятиями и отдельными задачами и при этом каждый пользователь также имеет персональный бюджет.

Между моделями установлены отношения: пользователь владеет множеством мероприятий, каждое мероприятие содержит задачи услуги, задачи могут быть назначены пользователям, чат агрегирует участников и содержит сообщения, а каждое сообщение связано с отправителем.

На рисунке 2.1.3.2 показан пример паттерна MVC между МероприятиеModel, МероприятиеView и МероприятиеController. В данном случае контроллер МероприятиеController управляет всеми действиями, связанными с мероприятиями.

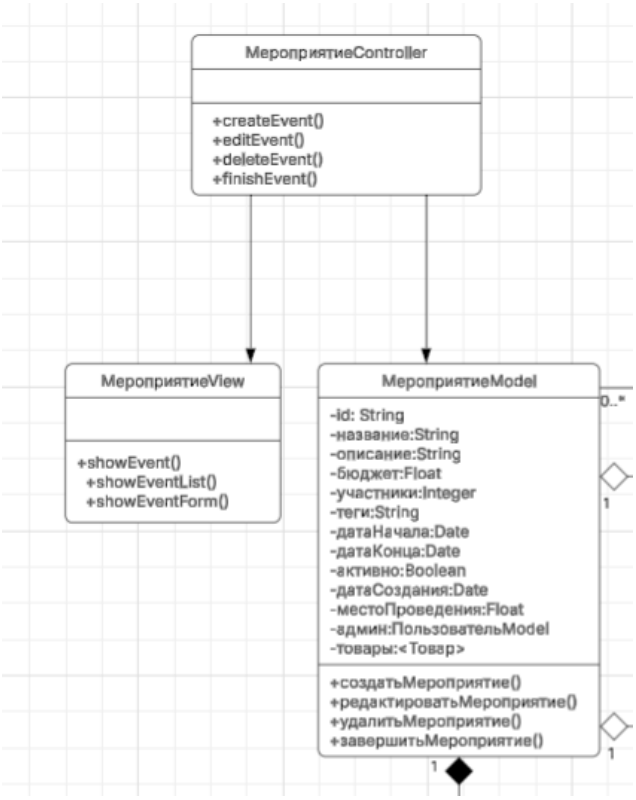



Рисунок 2.1.3.2 – Диаграмма классов на примере класса Мероприятия (паттерн Model-View-Controller)

Когда пользователь хочет создать, изменить или удалить мероприятие, именно МероприятиеController принимает этот запрос, обрабатывает его и взаимодействует с моделью мероприятия для выполнения нужной операции. После этого он передаёт результат во View, чтобы пользователь увидел обновлённую информацию. МероприятиеView отвечает за то, как пользователь видит мероприятия. Она получает данные от контроллера и отображает их в виде списка, формы или подробной карточки события, чтобы пользователь мог легко работать с интерфейсом. МероприятиеModel хранит все данные о мероприятии и реализует основную бизнес-логику. Она отвечает за создание, изменение, удаление и завершение мероприятия, а также за работу с базой данных, чтобы информация всегда оставалась актуальной и целостной.

### 2.1.4 Диаграмма кооперации объектов

На кооперативной диаграмме показано, как разные объекты системы взаимодействуют между собой при создании мероприятия. Пользователь, выступающий в роли автора, инициирует процесс через интерфейс, который играет роль агента и принимает данные от пользователя. Интерфейс передаёт эти

данные контроллеру мероприятий, также выполняющему функцию агента, который координирует дальнейшие действия. Контроллер обращается к сервису мероприятий, который уже является сервером и отвечает за бизнес–логику и работу с данными.

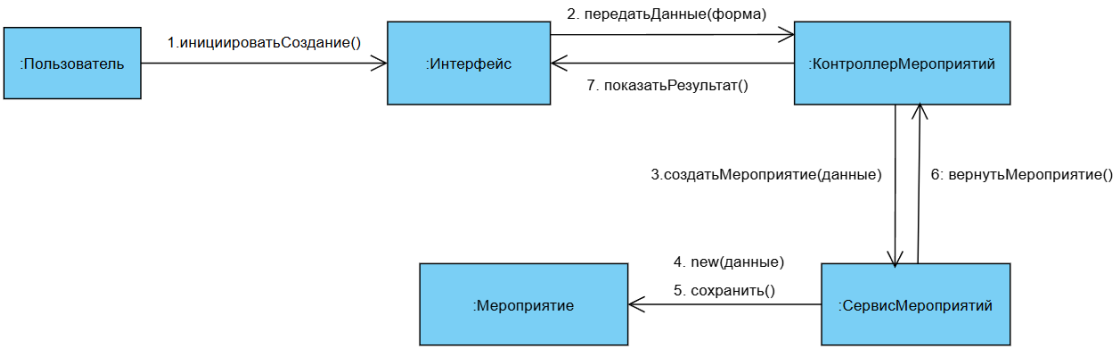



Рисунок 2.1.4.1 – Диаграмма кооперации объектов прецедента создания мероприятия

Сервис мероприятий создаёт новый объект мероприятия, используя конструктор, и затем сохраняет его, что является операцией–модификатором, изменяющей состояние объекта. После этого сервис возвращает контроллеру информацию о созданном мероприятии, используя селектор, то есть операцию, не изменяющую состояние объекта. Контроллер, в свою очередь, сообщает интерфейсу о результате, также с помощью селектора. Таким образом, диаграмма иллюстрирует не только последовательность взаимодействий, но и типы операций, которые выполняются на каждом этапе, а также роли объектов в этом процессе [8].

### 2.1.5 Диаграмма активности

Диаграммы деятельности позволяет описывать логику процедур, бизнес–процессы и потоки работ. Они показывают, какие шаги выполняются параллельно.

В этой диаграмме последовательности (Рисунок 2.1.5.1) отражён типовой сценарий работы пользователя в системе управления мероприятиями, где параллельно взаимодействуют несколько ключевых объектов: Пользователь, Мероприятие, Услуга и Задача.

Сначала пользователь открывает главную страницу приложения и выбирает, войти в существующий аккаунт или зарегистрироваться. Если у пользователя уже есть аккаунт, он вводит e–mail и пароль. Если аккаунта нет, он

проходит регистрацию, указывая имя, e–mail, пароль, бюджет, выбирает свою роль и соглашается на обработку персональных данных.

После успешной авторизации пользователь попадает в панель управления, где может инициировать создание нового мероприятия. На этапе создания мероприятия пользователь заполняет основные параметры: название, описание, дату, место проведения и бюджет. Далее система предлагает назначить участников – пользователь может добавить их сразу или продолжить без них.

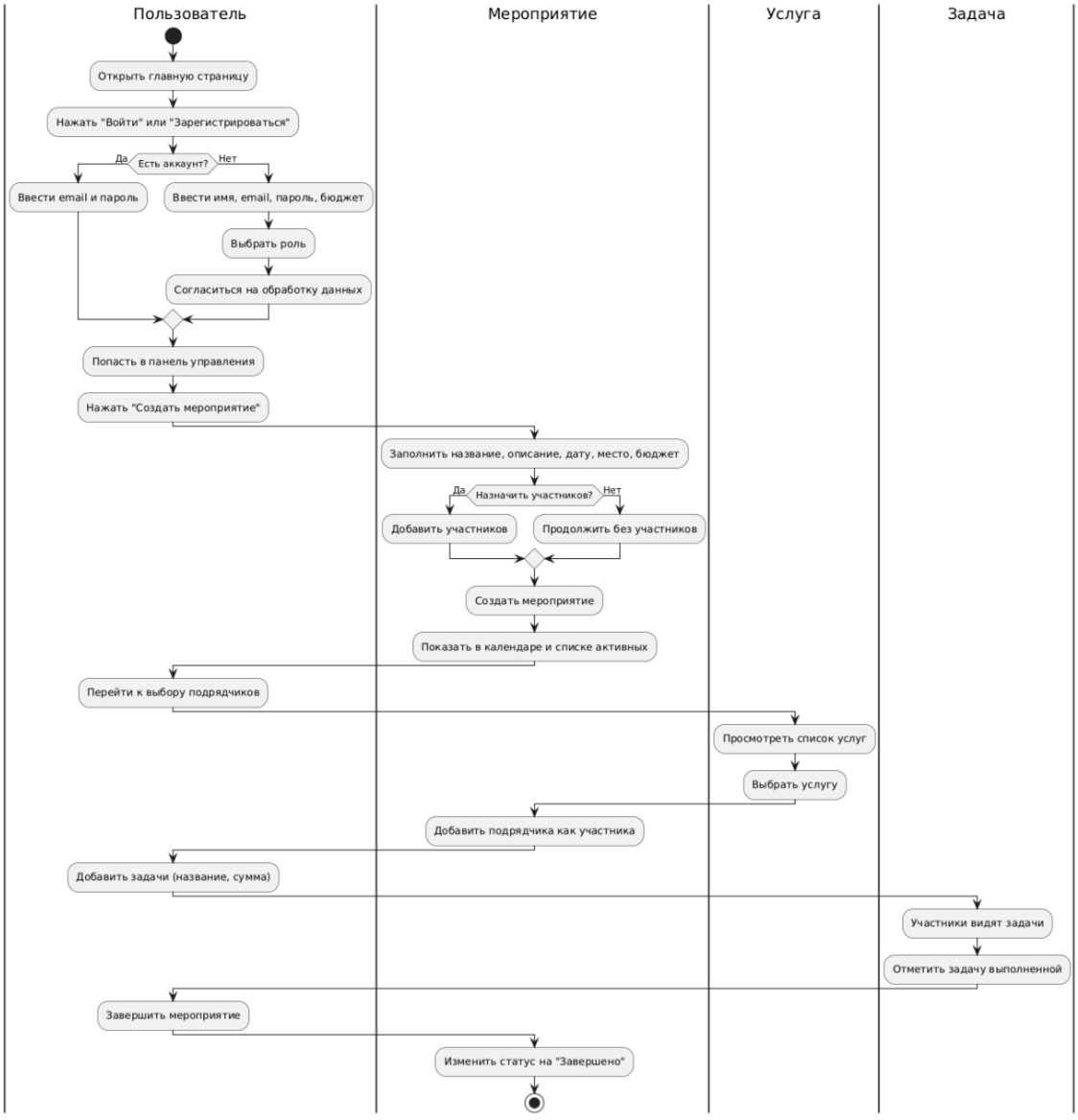



Рисунок 2.1.5.1 – Диаграмма активности прецедента создания мероприятия

Когда мероприятие создано, оно появляется в календаре и в списке активных мероприятий пользователя. Затем пользователь переходит к выбору подрядчиков: просматривает список доступных услуг, выбирает подходящую услугу, и выбранный подрядчик добавляется как участник мероприятия.

Следующий этап – добавление задач. Пользователь формулирует задачи, указывает их название и сумму, выделяемую из бюджета. После этого участники мероприятия видят назначенные им задачи и могут отмечать их как выполненные.

В завершение пользователь инициирует завершение мероприятия. Система меняет статус мероприятия на "Завершено", что фиксирует успешное выполнение всех этапов и задач, связанных с этим событием.

Таким образом, диаграмма иллюстрирует последовательное и параллельное взаимодействие между пользователем, мероприятиями, услугами и задачами, показывая, как шаг за шагом формируется и реализуется процесс организации события в системе.

### 2.1.6 Диаграмма состояний

Данная диаграмма состояний (Рисунок 2.1.6.1) описывает жизненный цикл экземпляра мероприятия в системе. Всё начинается с этапа создания, когда инициируется новый объект и пользователь начинает заполнять его параметры. На этом этапе мероприятие находится в состоянии черновика: пользователь может редактировать данные, а система ожидает заполнения всех обязательных полей. Как только все необходимые данные введены и подтверждены, мероприятие переходит в активное состояние. Теперь оно добавляется в календарь, становится доступным для участников, и пользователь может продолжать его редактировать, а также добавлять задачи, встречи и новых участников.

Когда все задачи по мероприятию завершены, и пользователь отмечает его как завершённое, объект переходит в состояние завершения. В этот момент система уведомляет участников о завершении, а само мероприятие становится доступным только для просмотра. Если пользователь решает удалить мероприятие, оно переходит в состояние удаления, где происходит очистка всех связанных данных. После этого жизненный цикл мероприятия считается завершённым.

Также предусмотрена возможность отмены создания мероприятия ещё на этапе черновика. В этом случае черновик удаляется, связанные данные очищаются, и объект также переходит в конечное состояние. Таким образом, диаграмма отражает все основные этапы и переходы, через которые проходит мероприятие от момента создания до полного удаления из системы.

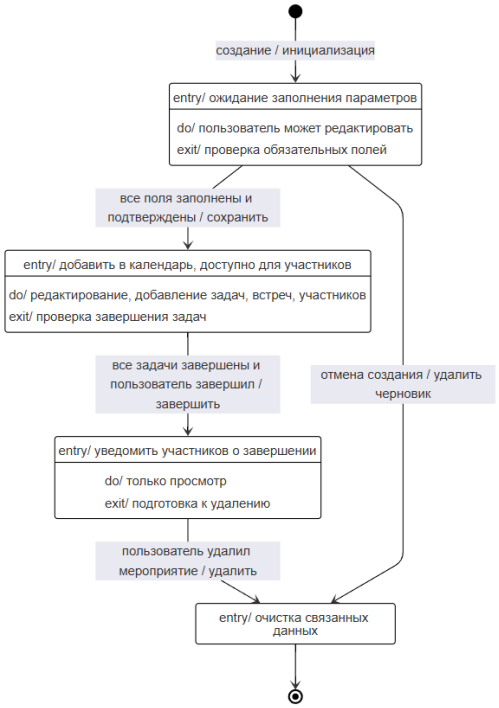



Рисунок 2.1.6.1 – Диаграмма состояний объекта «Мероприятие»

### 2.1.7 Диаграмма компонентов

Физические части системы (компоненты) и их зависимости показаны на рисунке 2.1.7.1, где,

ReactJS – фронтенд–компонент, реализующий пользовательский интерфейс. Клиентская часть представляет собой фронтенд–приложение, предоставляющее интерфейс для взаимодействия пользователя с системой. Она обменивается данными с серверной частью посредством API–запросов. В рамках клиентского интерфейса реализованы функции авторизации, панели управления, чата, управления мероприятиями, работы с поставщиками и профиля пользователя.

Mongoose – компонент, реализующий работу с базой данных на стороне сервера (ORM для MongoDB).

MongoDB – база данных.

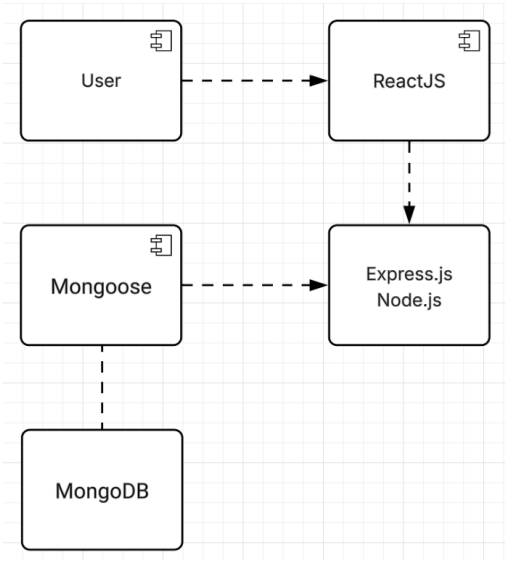



Рисунок 2.1.7.1 – Диаграмма компонентов

Express.js и Node.js – интерфейс взаимодействия сервера с базой данных. В серверной логике реализованы сервисы авторизации, управления пользователями, мероприятиями, задачами, подрядчиками и чатами.

## 2.2 База данных

На ER (сущность-связь) диаграмме (Рисунок 2.1.4.1) показано как хранится и организована база данных в системе.

В центре структуры находится таблица events – каждое мероприятие содержит основную информацию: название, описание, бюджет, даты, место проведения, администратора и другие параметры. С мероприятием связаны другие сущности: например, через таблицу collaborators к мероприятию могут быть привязаны участники (users), а через tasks – задачи, которые нужно выполнить в рамках этого события. Каждая задача содержит описание, сумму, статус выполнения и ссылку на исполнителя.

Для организации встреч используется таблица meetings, где хранятся данные о встречах, связанных с определённым мероприятием. Участники встреч фиксируются в отдельной таблице meeting_participants. Это позволяет одной встрече иметь несколько участников и одному пользователю участвовать в разных встречах.

В системе реализована возможность общения через чаты (chats), которые также могут быть связаны с определённым мероприятием. В чате могут участвовать разные пользователи, что отражено в таблице chat_participants.

Сообщения, отправленные в чате, хранятся в таблице chat_messages вместе с информацией об авторе, времени отправки и содержимом сообщения.

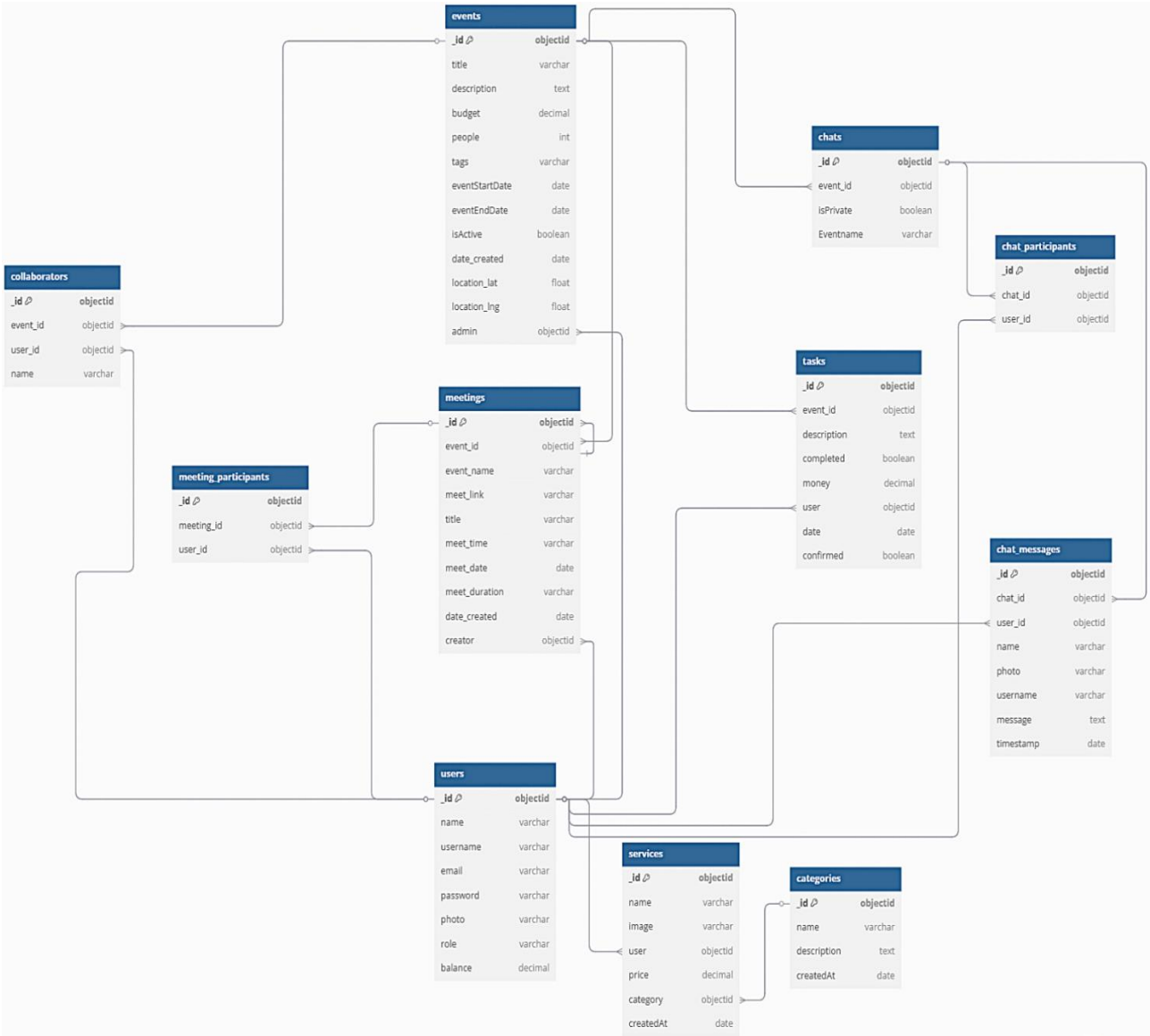



Рисунок 2.2.1 – ER диаграмма базы данных

Пользователи (users) могут создавать и предлагать услуги (services), которые относятся к определённым категориям (categories). Это позволяет реализовать поиск и фильтрацию услуг по категориям, а также связывать услуги с их владельцами.

## 2.3 Реализация вэб–приложения

### 2.3.1 Слой данных

В отличие от традиционных реляционных баз, в MongoDB данные хранятся в более гибком формате коллекций (Рисунок 2.2.1.1).

Когда пользователь регистрируется в системе, вся его информация – имя, email, зашифрованный пароль и права доступа – попадает в коллекцию Users. Это основа всего, ведь именно пользователи создают контент и взаимодействуют между собой.

Все мероприятия, которые организуют пользователи, сохраняются в коллекции Events. Здесь есть всё: от названия и описания события до точного времени проведения и списка участников. Интересно, что мы специально сделали структуру максимально гибкой, чтобы можно было легко добавлять новые поля, если понадобится расширить функционал.

Для общения между участниками у нас реализована система чатов. Все сообщения аккуратно складываются в коллекцию Chats. При этом мы храним не только текст сообщений, но и всю сопутствующую информацию – кто отправил, кому, когда, что очень помогает при анализе активности пользователей.

Отдельная коллекция Meets отвечает за организацию встреч. Здесь мы фиксируем все детали: кто встречается, где и когда. Причём встречи могут быть как самостоятельными событиями, так и частью более крупных мероприятий из коллекции Events.


```
_id: ObjectId('6831c8a96ba4d33bbc170c8c')
name : "admin"
username : "admin"
email : "admin@admin.co"
password : "$2a$10$/lyQK2kiRA12l2CNNGPGl.MM5UWPtQq1CqftjmZds4HcDsyswpJ0a"
photo : "defaultUpload.jpg"
role : "Пользователь"
balance : 232872
__v : 0
```


Рисунок 2.3.1.1– Пример структуры коллекции Users

Этот API служит связующим звеном между клиентскими приложениями и базой данных, предоставляя стандартизированные интерфейсы для выполнения базовых операций с данными: создания, чтения, обновления и удаления записей.

Основные функциональные маршруты организованы по логическим группам (Рисунок 2.2.1.3). Путь /api/auth отвечает за всё, что связано с учётными записями пользователей: регистрацию новых аккаунтов, процедуру входа в

систему и управление сессиями. Для работы с мероприятиями существует отдельный маршрут /api/event, через который можно создавать события, просматривать их детали, вносить изменения и управлять участием пользователей.

```
router.post("/login", async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });
  if (!user || !(await bcrypt.compare(password, user.password))) {
    return res.status(401).json({ message: "Неверный email или пароль" });
  }
  const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, { expiresIn: "1h" });
  res.json({ token });
});
```

Рисунок 2.3.1.2 – Пример маршрута для авторизации

Обмен сообщениями между участниками системы осуществляется через эндпоинты /api/chat. Этот интерфейс позволяет отправлять личные и групповые сообщения, получать историю переписки и управлять чатами. Организация встреч пользователей реализована через маршрут /api/meet, который поддерживает создание встреч, изменение их параметров и управление списками участников.

Дополнительно реализован специализированный маршрут /api/sendmail, отвечающий за отправку e–mail–уведомлений. Этот функционал используется для рассылки подтверждений регистрации, напоминаний о предстоящих мероприятиях и других важных событиях.

Все маршруты защищены механизмом аутентификации на основе JWT–токенов, что гарантирует безопасный доступ к данным. Валидация входящих запросов выполняется перед обработкой, обеспечивая целостность информации в системе. Для работы с MongoDB используется библиотека Mongoose, которая упрощает взаимодействие с базой данных через систему схем и моделей.

Ниже представлены основные функции слоя представления:

– формы для создания и редактирования мероприятий;

– отображение списка мероприятий, чатов и встреч;

– уведомления для пользователей.

```jsx
import React, { useEffect, useState } from "react";
import axios from "axios";

const EventList = () => {
  const [events, setEvents] = useState([]);

  useEffect(() => {
    const fetchEvents = async () => {
      const response = await axios.get("/api/event");
      setEvents(response.data);
    };
    fetchEvents();
  }, []);

  return (
    <div>
      <h1>Список мероприятий</h1>
      <ul>
        {events.map((event) => (
          <li key={event._id}>
            {event.title} - {new Date(event.date).toLocaleDateString()}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default EventList;
```

Рисунок 2.3.1.3 – Пример компонента React для отображения списка мероприятий

Пользователи могут зарегистрироваться и войти в систему. Роли пользователей такие как подрядчик и пользователь были добавлены (Рисунок 2.2.1.4) для ограничения доступа.

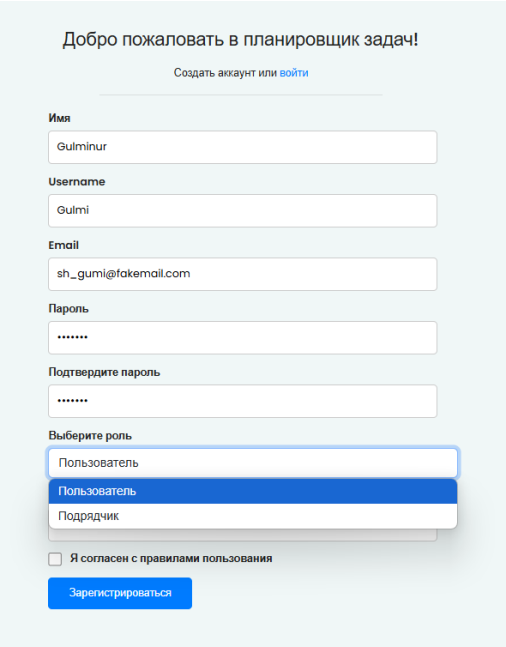



Рисунок 2.3.1.4 – Форма авторизации новых пользователей

Благодаря данной форме (Рисунок 2.2.1.4) пользователь сможет зарегистрироваться в приложении, выбрав себе роль.

```
router.post("/register", async (req, res) => {
  const { name, email, password } = req.body;
  const hashedPassword = await bcrypt.hash(password, 10);
  const newUser = new User({ name, email, password: hashedPassword });
  await newUser.save();
  res.status(201).json({ message: "Пользователь зарегистрирован" });
});
```

Рисунок 2.3.1.5 – Пример регистрации пользователя

Пользователи могут создавать, редактировать, удалять и просматривать мероприятия. Каждое мероприятие может содержать участников, дату, место и описание. (Рисунок 2.2.1.5).

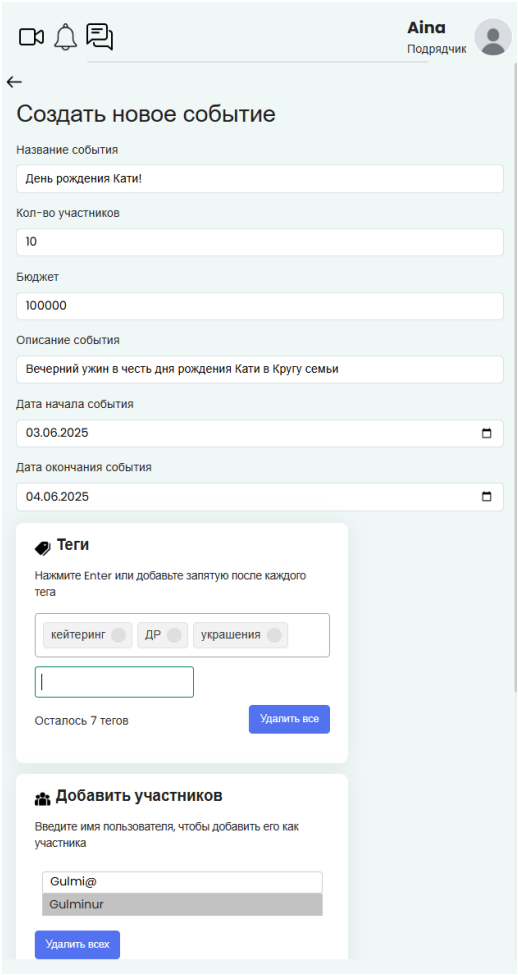



Рисунок 2.3.1.6 – Форма создания мероприятия

Благодаря данной форме (Рисунок 2.2.1.6) пользователь сможет создать собственное мероприятие, добавив предварительно теги, коллабораторов мероприятия, указав бюджет и количество участников.

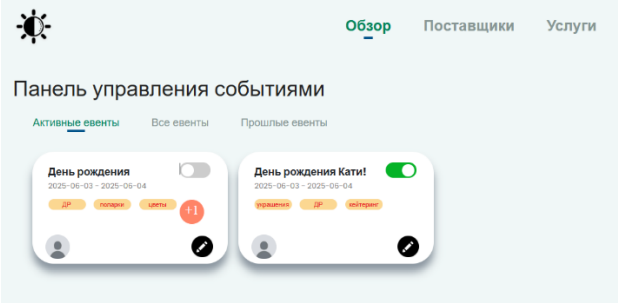



Рисунок 2.3.1.7 – Отображение мероприятий на панели управления

На данной панели (Рисунок 2.2.1.7) пользователь сможет увидеть мероприятия, в которых он состоит и которые он создал. Также он сможет отключить мероприятие, чтобы скрыть его из вида.

```
router.post("/", async (req, res) => {
  const { title, description, date, location } = req.body;
  const newEvent = new Event({ title, description, date, location });
  await newEvent.save();
  res.status(201).json(newEvent);
});
```

Рисунок 2.3.1.8 – Пример создания мероприятия

На скриншоте (Рисунок 2.2.1.8) показан запрос в API для создания нового мероприятия. Пользователи могут отправлять сообщения друг другу. Сообщения сохраняются в базе данных с указанием отправителя, получателя и времени отправки.

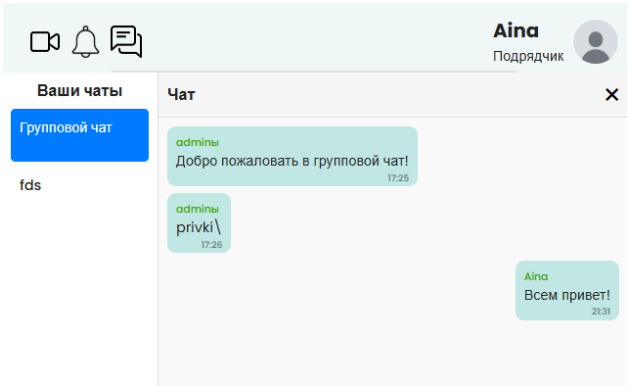



Рисунок 2.3.1.9 – Список чатов\чат

Данный компонент (Рисунок 2.2.1.9) реализует чат. Показан групповой чат, созданный от мероприятия и личный чат с пользователем.

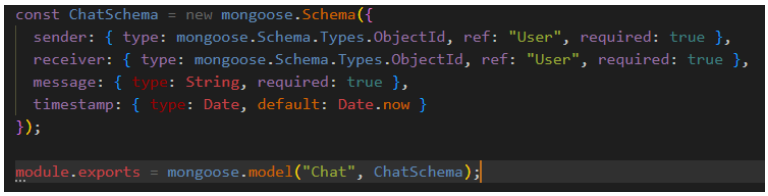


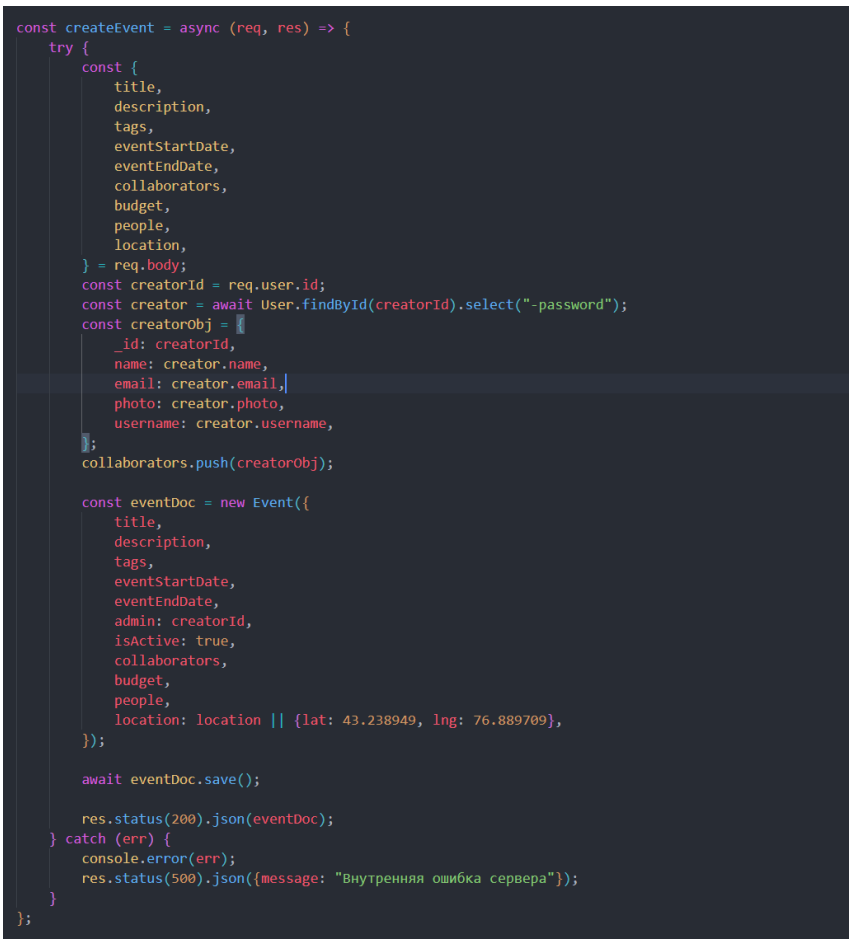

Рисунок 2.3.1.10 – Пример модели Chat

На данной картинке (Рисунок 2.2.1.10) показана реализация модели Chat. Класс, который проецирует таблицу chats, со всеми полями и свойствами. Пользователи могут планировать встречи, добавлять участников и указывать место проведения.

### 2.3.2 Слой контроллеров

Данный слой является ключевым в приложении, ведь в нем реализуется бизнес логика приложения. В нем соединяются слой моделей и слой представлений. Основные операции проводятся в этом слое, так как методы контроллеров реализуют композицию работы с данными, функционалом фреймворка и передачу данных в слой представлений.

Например, для создания нового события контроллер принимает POST-запрос с JSON данными, затем валидирует их, вызывает метод mongoose для сохранения события в базе данных и возвращает результат клиенту (Рисунок 2.3.2.1). Каждый контроллер отвечает за отдельную сущность. EventController отвечает за операции с мероприятиями, ChatController – за операции с чатами и т. д. Такой подход удовлетворяет несколько принципов SOLID. Принцип разделения ответственности и расширяемости приложения.



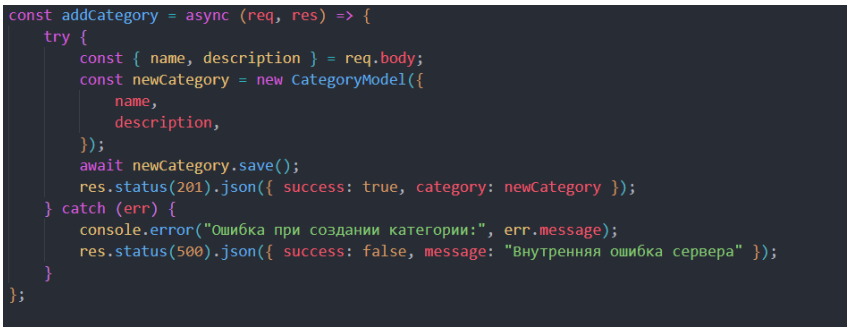
```
const createEvent = async (req, res) => {
    try {
        const {
            title,
            description,
            tags,
            eventStartDate,
            eventEndDate,
            collaborators,
            budget,
            people,
            location,
        } = req.body;
        const creatorId = req.user.id;
        const creator = await User.findById(creatorId).select("-password");
        const creatorObj = {
            _id: creatorId,
            name: creator.name,
            email: creator.email,
            photo: creator.photo,
            username: creator.username,
        };
        collaborators.push(creatorObj);

        const eventDoc = new Event({
            title,
            description,
            tags,
            eventStartDate,
            eventEndDate,
            admin: creatorId,
            isActive: true,
            collaborators,
            budget,
            people,
            location: location || {lat: 43.238949, lng: 76.889709},
        });

        await eventDoc.save();

        res.status(200).json(eventDoc);
    } catch (err) {
        console.error(err);
        res.status(500).json({message: "Внутренняя ошибка сервера"});
    }
};
```

Рисунок 2.3.2.1 – Реализация создания события



```
const addCategory = async (req, res) => {
    try {
        const { name, description } = req.body;
        const newCategory = new CategoryModel({
            name,
            description,
        });
        await newCategory.save();
        res.status(201).json({ success: true, category: newCategory });
    } catch (err) {
        console.error("Ошибка при создании категории:", err.message);
        res.status(500).json({ success: false, message: "Внутренняя ошибка сервера" });
    }
};
```

Рисунок 2.3.2.2 – Реализация метода addCategory

Контроллер не содержит бизнес логику напрямую, а делегирует её сервисам фреймворка. Это не только повышает читаемость кода и делает его лаконичнее и понятнее, но и повышает тестируемость функционала (Рисунок 2.3.2.2). Слой контроллеров способствует четкой организации приложения и последующую его расширяемость.

### 2.3.3 Слой представления

Этот слой отвечает за представление данных перед пользователем в формате UI. Благодаря этому слою, пользователь сможет взаимодействовать со всем функционалом приложения. Основная задача представления – превратить данные, полученные из контроллера, в понятный для пользователя вид и отобразить в браузере.

Также слой представлений обязан обеспечить удобство использование приложения для пользователя. Это включает в себя адаптивность, простоту, ёмкость информации.

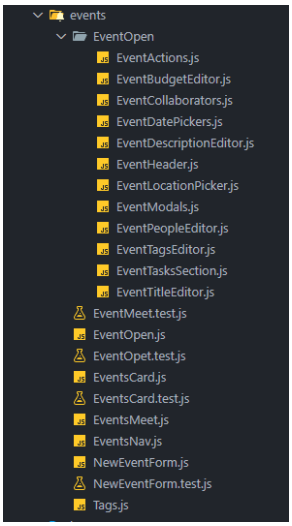



Рисунок 2.3.3.1 – структура папок компонентов, разбитая на сущности

Структура слоя представлений включает в себя компоненты, веб-перехватчики, которые используются для удобного переиспользования функций и состояний в разных компонентах. Всё это делится на сущности. К примеру, компоненты EventOpen и EventCard лежат в одной директории «events» (Рисунок 2.3.3.1).


Рисунок 2.3.3.2 – Реализация компонента EventCard

Слой представлений содержит только HTML верстку, получение данных от контроллера и функционал для удобства использования UI. В нём не содержится бизнес логика и взаимодействие с моделями, так как это противоречит принципу разделения ответственности. (Рисунок 2.3.3.2)

Этот слой, обеспечивая удобное отображение информации и удобство использования, играет ключевую роль во взаимодействии клиента с приложением.

### 2.3.4 Применение паттерна MVC на практике

Ранее мы определили, что основные сущности — это пользователь и мероприятие. Начнем разработку с них. Напишем модель пользователя с полем баланса и ролью: (Рисунок 2.3.4.1)


Рисунок 2.3.4.1 – Модель User

Далее создадим модель мероприятий со связью модели User и подсущность Task. (Рисунок 2.3.4.2)


Рисунок 2.3.4.2 – Модель Event

От них создадим контроллер авторизации и контроллер управления мероприятием:

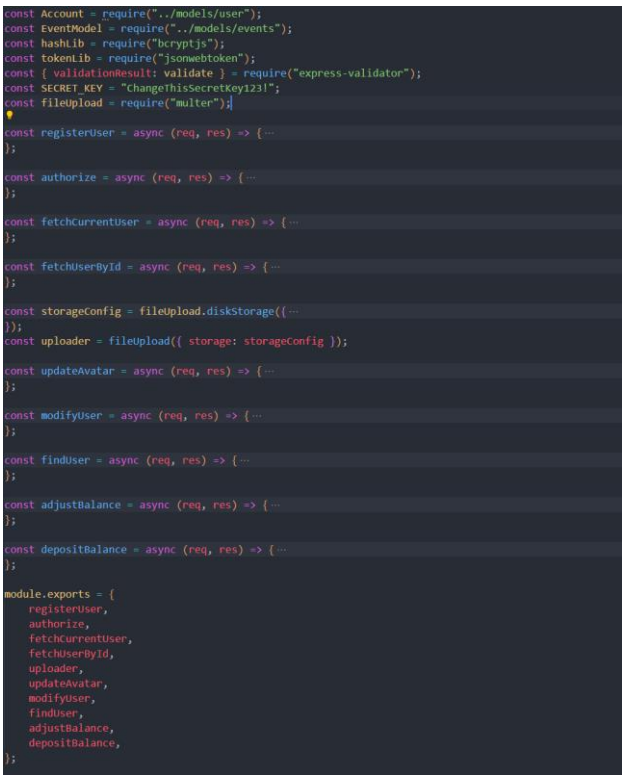



Рисунок 2.3.4.3 – Реализация AuthController

В данном контроллере мы реализовали методы регистрации, авторизации, поиск пользователя и другие важные бизнес функции (Рисунок 2.3.4.3).


Рисунок 2.3.4.4 – Реализация EventController

Также реализуем все необходимые методы для мероприятий, такие как удаление, создание, редактирование, создание задач, добавление подрядчиков и т. д. (Рисунок 2.3.4.4)

```
const express = require("express");
const router = express.Router();
const fetchuser = require("../middlewares/fetchuser");
const checkRole = require("../middlewares/checkRole");
const EventsController = require("../controllers/EventsController");

router.post("/add", fetchuser, EventsController.createEvent);

router.get("/list", fetchuser, EventsController.getEvents);

router.put("/status/:id", fetchuser, EventsController.updateEventStatus);

router.put("/update/:id", fetchuser, EventsController.updateEvent);

router.post("/task/add/:id", EventsController.createTask);

router.put("/task/edit/:eventId/:taskId", EventsController.editTask);

router.post("/task/confirm", fetchuser, EventsController.confirmTask);

router.delete("/task/delete/:eventId/:taskId", EventsController.deleteTask);

router.delete("/remove/:id", fetchuser, EventsController.deleteEvent);

router.get("/search/:search", fetchuser, EventsController.searchEvent);

router.put("/collaborators/:id", fetchuser, EventsController.updateCollaborators);

router.get("/byid/:id", fetchuser, EventsController.getEventById);

router.put("/task/update/:eventId/:taskId", EventsController.updateTask);

module.exports = router;
```

Рисунок 2.3.4.5 – Маршрутизация мероприятий

Теперь добавим маршрутизацию по нашим методам, чтобы можно было получить доступ к методам контроллера извне (Рисунок 2.3.4.5).

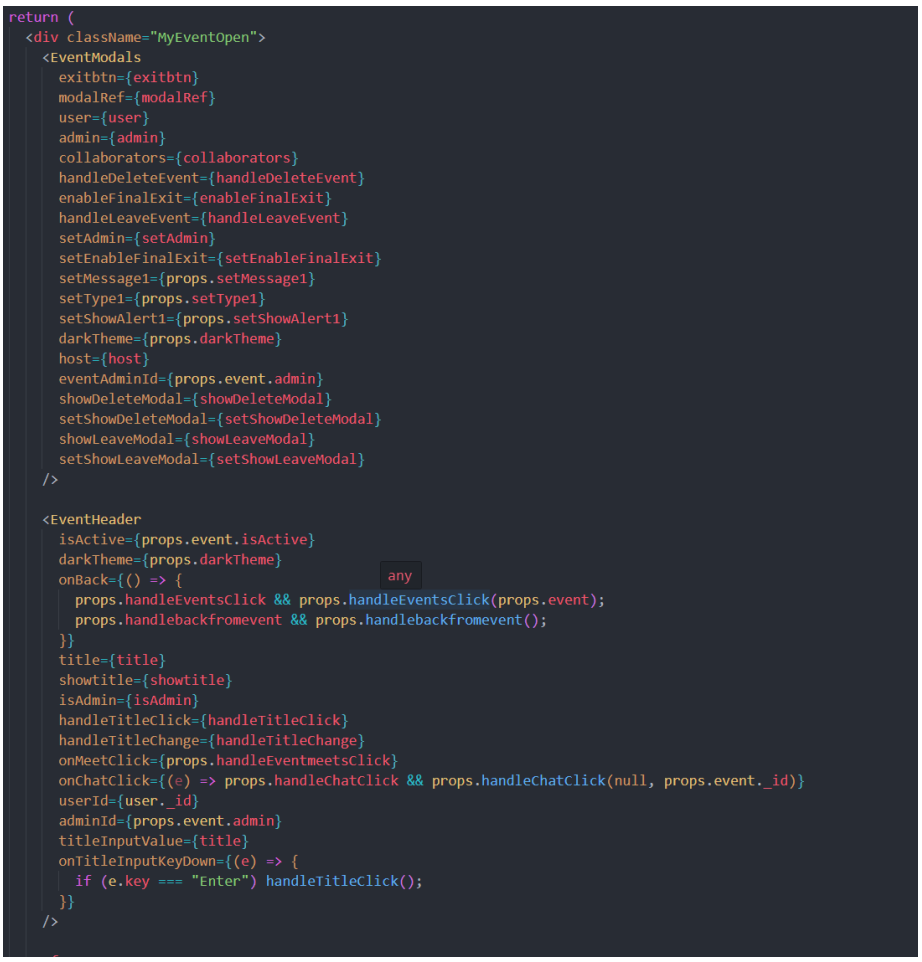



Рисунок 2.3.4.6 – Компонент EventOpen

Теперь, когда мы имеем доступ к маршрутам, можем создать компоненты для мероприятия. Чтобы повысить читаемость кода, мы делаем декомпозицию компонента на подкомпоненты, а свойства и функции передаем через props.

```
const handleTitleClick = () => {
  if (user._id === event.admin) {
    setShowtitle(!showtitle);
  } else {
    showAlert("Only Admin of the event can change the Title", "danger");
  }
};

const handleDescClick = () => setShowdesc(!showdesc);
const handleTagClick = () => setShowtags(!showtags);
const handleTasksClick = () => setShowtasks(!showtasks);
const handleCollabsClick = () => setShowcollabs(!showcollabs);
const handleBudgetClick = () => setShowbudget(!showbudget);

const handlePeopleClick = () => {
  if (user._id === event.admin) {
    setShowpeople(!showpeople);
  } else {
    showAlert("Only Admin of the event can change the number of people", "danger");
  }
};

const handlePeopleChange = (e) => {
  setPeople(e.target.value);
  setEnableSave(true);
};

const handleTitleChange = (e) => {
  setTitle(e.target.value);
  setEnableSave(true);
};

const handleDescChange = (e) => {
  setDescription(e.target.value);
  setEnableSave(true);
};

const handleBudgetChange = (e) => {
  setBudget(e.target.value);
  setEnableSave(true);
};

const handleRemoveTag = (tag) => {
  removeTag(tag);
  setEnableSave(true);
};
```

Рисунок 2.3.4.7 – Веб-перехватчик useEventOpen

Чтобы не «засорять» файлы одними и теми же методами, мы создаем веб-перехватчик useEventOpen. Он предназначен для инкапсуляции бизнес логики связанной с отображением, редактированием информации о мероприятии. В нём присутствует такой функционал как управление EventState – состояние мероприятия полученное один раз от сервера и переиспользуемое в дальнейшем в разных компонентах. Он также содержит и взаимодействие с сервером. Благодаря нему мы упростили компоненты представления редактирования мероприятия и централизовали всю логику в одном месте (Рисунок 2.3.4.7).

```
const { host } = require("../env.js");

export async function searchUserByName(name, token) {
  const response = await fetch(
    `${host}/api/auth/finduser/${name}`,
    {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        "auth-token": token,
      },
    }
  );
  const json = await response.json();
  if (json.success) {
    return json.users;
  }
  return null;
}

export async function fetchEvents(token) {
  const response = await fetch(`${host}/api/events/list`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      "auth-token": token,
    },
  });
  return response.json();
}

export async function fetchEventById(eventId, token) {
  const response = await fetch(`${host}/api/events/byid/${eventId}`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      "auth-token": token,
    },
  });
  return response.json();
}
```

Рисунок 2.3.4.8 – Пример eventApi.js

Так как в нашем приложении мы неоднократно используем методы обращения к API сервера, было решено вынести эндпоинты в отдельный файл eventApi.js. Так мы сможем переиспользовать запросы к серверу, избегая повторяющегося кода. (Рисунок 2.3.4.8)

```
const server = http.createServer(app);
const io = new Server(server, {cors: {origin: "*"}});

io.on('connection', (socket) => {
    socket.on('join', (roomId) => {
        socket.join(roomId);
    });

    socket.on('leave', (roomId) => {
        socket.leave(roomId);
    });

    socket.on('message', async (msgObj) => {
        const roomId = msgObj.chatId;
        if (!roomId) return;

        const chatMessage = {
            user_id: msgObj.sender._id,
            name: msgObj.sender.name,
            photo: msgObj.sender.photo,
            username: msgObj.sender.username,
            message: msgObj.content,
            timestamp: msgObj.timestamp || new Date(),
        };

        try {
            await ChatModel.findByIdAndUpdate(
                roomId,
                {$push: {chats: chatMessage}},
                {new: true}
            );
            io.to(roomId).emit('message', chatMessage);
        } catch (err) {
            console.error("Ошибка при сохранении сообщения:", err);
        }
    });

    socket.on('deleteMessage', async ({chatId, messageId}) => {
        try {
            const chat = await ChatModel.findById(chatId);
            if (!chat) return;
            const msgIndex = chat.chats.findIndex(m => m._id.toString() === messageId);
```

Рисунок 2.3.4.9 – Пример обработчика событий веб-сокета

Так же в приложении присутствует более интересный функционал –веб-сокет. Веб-сокет – технология, позволяющая получать сообщения от сервера мгновенно, без взаимодействия с интерфейсом. Например, когда мы открываем чат, нам не нужно перезагружать страницу для получения сообщения. Это заслуга веб-сокета. Так же и с отправлением сообщения. Инициализационный контроллер в серверной части создает подключение к веб-сокету, который прослушивает свои события в другом контроллере. В слое представлений мы так же создаем веб-перехватчик, который слушает события веб-сокета. И когда события срабатывают при каком-то действии, слушатели выполняют свой код, который, например, отображает сообщение в чате. (Рисунок 2.3.4.9)

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была спроектирована и реализована информационная система для управления мероприятиями, отвечающая современным требованиям автоматизации процессов в данной предметной области.

В процессе работы были проведены анализ и формализация требований, построены основные UML– диаграммы (прецедента, последовательности, классов, кооперации объектов, активности, состояний, компонентов), ER-диаграмма базы данных, что позволило структурировать архитектуру системы и обеспечить её масштабируемость и поддерживаемость.

В результате разработки реализовано вэб-приложение, включающее пользовательский интерфейс, серверную часть и работу с базой данных. Система поддерживает регистрацию и авторизацию пользователей, создание и управление мероприятиями, назначение задач и подрядчиков, организацию чатов и встреч, а также автоматизированный контроль бюджета и статусов выполнения задач. Особое внимание уделено безопасности, удобству взаимодействия и расширяемости функционала.

Разработанная система может быть использована как самостоятельное решение для автоматизации управления мероприятиями, а также служить основой для дальнейшего развития и интеграции с внешними сервисами.

Таблица 2 – Сокращения, термины и их определения

| Сокращения и термины | Определение |
|---|---|
| UML | Unified Modeling Language – язык графического моделирования для проектирования систем |
| ER-диаграмма | Entity-Relationship диаграмма – схема, отображающая связи между сущностями в базе данных |
| MVC | Model-View-Controller – архитектурный шаблон (модель, представление и контроллер) |
| JavaScript | Язык программирования, используемый для создания интерактивных веб-интерфейсов |
| Mongoose | Библиотека для работы с MongoDB в среде Node.js |
| ORM | Object-Relational Mapping – подход, при котором объекты кода отображаются на таблицы базы данных |
| JWT | JSON Web Token – способ безопасной передачи информации для авторизации |
| API | Application Programming Interface – интерфейс взаимодействия между программными компонентами |
| PWA | Progressive Web App – технология, позволяющая веб-приложению работать как мобильное |
| CRUD | Create, Read, Update, Delete – базовые операции с данными |
| JSON | JavaScript Object Notation – формат обмена данными между клиентом и сервером |
| REST API | Стиль архитектуры API, использующий стандартные HTTP-запросы для доступа к данным |
| HTML | HyperText Markup Language – язык разметки веб-страниц. |
| MongoDB | NoSQL база данных, хранящая данные в формате BSON-документов |
| Express.js | Минималистичный веб-фреймворк для Node.js, применяемый для создания API |
| React | JavaScript-библиотека для создания пользовательских интерфейсов |
| Node.js | Среда выполнения JavaScript на сервере |
| P2P | Peer-to-Peer – одноранговая сеть |
| Веб-сокеты | Технология, позволяющая устанавливать постоянное соединение между клиентом и сервером для мгновенного обмена данными |

## Список использованной литературы

Сайты

1 Бюро национальной статистики агентства по стратегическому планированию и реформам Республики Казахстан: [сайт]. – Астана, 2024 https://stat.gov.kz/ru/news/braki–i–razvody–v–2024–godu/ (дата обращения: 20.12.2024). – Текст.

2 EZ solutions международное ивент–агентство [сайт]. – Алматы, 2024 https://ezs.kz/ru/news/innovacii-i-tradicii-v-ivent-industrii-interviu-s-ispolnitelnym-direktorom-kazaxstanskoi-associacii-marketinga-aneliei-muxamedkarimovoi (дата обращения: 03.01.2025). – Текст.

3 MongoDB [сайт]. https://www.mongodb.com/docs/manual/introduction/ (дата обращения: 02.03.2025). – Текст.

4 React [сайт]. https://ru.legacy.reactjs.org/docs/getting-started.html (дата обращения: 04.03.2025). – Текст.

Электронные ресурсы

5 Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js, Conference: 2014 IEEE 17th International Conference on Computational Science and Engineering [Электронный ресурс]. Режим доступа: URL: https://www.researchgate.net/publication/286594024_Performance_Comparison_and_Evaluation_of_Web_Development_Technologies_in_PHP_Python_and_Nodejs (дата обращения 10.03.2025).

Книги

6 Дронов В. А. Node.js, Express, MongoDB React. 23 урока для начинающих — СПб: БХВ - Петербург, 2024. — 608 с

7 Фаулер М. UML. Основы, 3–е издание – Пер. с англ. – СПб: Символ Плюс, 2004. – 27 с.

8 Кубеков Б. С. Технологии разработки программного обеспечения: Учебник. — Алматы: 2025.–322 с.

## ПРИЛОЖЕНИЕ А

**Техническое задание**

### А.1.1 Техническое задание на разработку вэб–приложения для создания личных мероприятий

#### А.1.1.1 Основание для разработки

Система разрабатывается на основании приказа проректора по академической работе № 26–П/Ө от «29» января 2025 г.

#### А.1.1.2 Назначение

Данная система предназначена для создания, управления и координации различных типов мероприятий, как для частных пользователей, так и для бизнес–организаций. Система должна предоставить пользователям возможность создания мероприятий, делегирования задач, выбора подрядчиков, контроля за выполнением задач и бюджета мероприятия, а также взаимодействия с другими участниками через встроенные коммуникационные каналы.

#### А.1.1.3 Требования к функциональным характеристикам

Система должна обеспечивать возможность выполнения следующих функций:

– регистрация пользователя: пользователи должны иметь возможность зарегистрироваться и создать личный кабинет для доступа к функционалу системы.

– создание мероприятий: пользователи могут создавать мероприятия, указывая название, бюджет, теги, задачи, товары, даты начала и окончания.

– добавление подрядчиков: Пользователи могут добавить подрядчиков в своё мероприятие, зарегистрировав их как поставщиков услуг.

– чат с подрядчиком: для обсуждения деталей и условий сотрудничества предусмотрен чат с подрядчиками.

– делегирование задач: задачи можно делегировать подрядчикам с указанием бюджета, необходимого для выполнения каждой задачи.

– контроль прогресса: отслеживание хода выполнения задач и общего прогресса подготовки мероприятия с помощью прогресс–бара.

– управление бюджетом: бюджет мероприятия автоматически уменьшается при добавлении задач, а также начисляются средства подрядчикам по мере выполнения задач.

– удаление или выход из мероприятия: пользователи могут выйти из мероприятия или удалить его, если оно неактуально.

– поиск мероприятий и подрядчиков: система должна поддерживать поиск по мероприятиям и подрядчикам, с возможностью фильтрации по категориям.

– категории подрядчиков: подрядчики могут быть классифицированы по типам услуг (например, свадебные организаторы, фотографы и т. д.).

### А.1.1.4 Требования к надежности

Система должна обеспечивать бесперебойную работу в режиме онлайн с высокой степенью доступности. Должна быть реализована автоматическая система резервного копирования данных и восстановление после сбоев. Платформа должна поддерживать многозадачность и позволять пользователю безопасно взаимодействовать с системой одновременно с другими пользователями.

### А.1.1.5 Требования к составу и параметрам технических средств

Серверное оборудование: сервер с 4 ГБ RAM, 1 CPU;

Пакеты программ: Node.js, Express.js, MongoDB 5.0, Axios, JWT для аутентификации, Express–validator для валидации данных;

Сетевые ресурсы: Система должна работать в интернет–среде и обеспечивать возможность доступности через веб–клиент без необходимости установки дополнительного ПО.

**ПРИЛОЖЕНИЕ Б**

Листинг (код) программы

```
//Листинг Index.js
const express = require("express");
const connectDB = require("./db");
const cors = require("cors");
require('dotenv').config();
const app = express();
const port = process.env.PORT || 5000;
const path = require('path');
const servicesRoute = require("./routes/services");
const http = require('http');
const {Server} = require('socket.io');
const ChatModel = require('./models/chats');

connectDB();
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({extended: true}));
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
app.use("/api/events", require("./routes/events"));
app.use("/api/auth", require("./routes/auth"));
app.use("/api/meet", require("./routes/meets"));
app.use("/api/chat", require("./routes/chats"));
app.use("/api/categories", require("./routes/categories"));
app.use("/api/services", servicesRoute);

const server = http.createServer(app);
const io = new Server(server, {cors: {origin: "*"}});

io.on('connection', (socket) => {
   socket.on('join', (roomId) => {
      socket.join(roomId);
   });

   socket.on('leave', (roomId) => {
      socket.leave(roomId);

 });

   socket.on('message', async (msgObj) => {
      const roomId = msgObj.chatId;
      if (!roomId) return;

      const chatMessage = {
         user_id: msgObj.sender._id,
         name: msgObj.sender.name,
         photo: msgObj.sender.photo,
         username: msgObj.sender.username,
```

**ПРИЛОЖЕНИЕ Б**

```
  message: msgObj.content,
          timestamp: msgObj.timestamp || new Date(),
        };
        try {
          await ChatModel.findByIdAndUpdate(
            roomId,
            {$push: {chats: chatMessage}},
            {new: true}
          );
          io.to(roomId).emit('message', chatMessage);
        } catch (err) {
          console.error("Ошибка при сохранении сообщения:", err);
        }
    });
    socket.on('deleteMessage', async ({chatId, messageId}) => {
        try {
          const chat = await ChatModel.findById(chatId);
          if (!chat) return;
          const msgIndex = chat.chats.findIndex(m => m._id.toString() === messageId);
          if (msgIndex !== –1) {
            chat.chats.splice(msgIndex, 1);
            await chat.save();
            io.to(chatId).emit('deleteMessage', {messageId});
          }
        } catch (err) {
          console.error("Ошибка при удалении сообщения:", err);
        }
    });
});
server.listen(port, () => {
    console.log(Server listening on port ${port});
    console.log('WebSocket server running');
});
```

## ПРИЛОЖЕНИЕ В

### Тестирование веб-приложения

### 1. Цели и задачи тестирования

Главной целью тестирования является проверка методов и компонентов приложения на работоспособность, чтобы избежать ошибок при внедрении конечного продукта и использовании его настоящим пользователем. В такое тестирование входит: проверка функциональности (работоспособность всех модулей), проверка безопасности (защита от взлома, приватизация данных), тестирование производительности.

В нашем приложении мы используем автоматические Unit тесты и QA тестирование (вручную). Unit тесты позволяют нам протестировать то, до чего мы можем не дойти обычным QA тестированием, а также повышает скорость разработки и тестирования, ведь, чтобы тестировать приложение каждый раз при каком-либо изменении понадобится много времени и ресурсов. Unit тесты же позволяют нам одной лишь командой проверить приложение на работоспособность.

Использованная технология написания тестов JEST – это javascript фреймворк для тестирования кода, который разработан для обеспечения уверенности в корректной работе любого куска кода JS. Позволяет нам разрабатывать тесты с приемлемым и функциональным API, и быстро достигать желаемых результатов. В нашем приложении мы будем использовать jest как на фронтенде (React), так и на бэкенде (Node js, Express js), так как все эти фреймворки совместимы с данным инструментом тестирования.

### 2 Подготовка тестовой среды

Сначала нужно развернуть docker-container с с разработанным вэб-приложением, который содержит backend, frontend и контейнер с mongodb.

Для этого напишем команду «docker-compose up -d –build b» после чего выводится результат успешно «поднятого» Контейнера (Рисунок П.2.1.2).

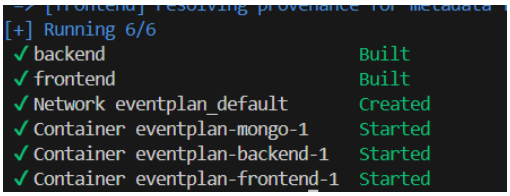



Рисунок П.2.1.2– скриншот успешного поднятия контейнера

### 2.2 Backend тестирование

Чтобы протестировать backend часть клиент серверного приложения, нужно определить, что мы будем тестировать. В паттерне MVC основной функционал содержится в

контроллерах, так как модели содержат лишь информацию о полях и свойствах сущности. Контроллер же выполняет композиционные действия с данными приложения и функционалом фреймворка, чтобы достичь желаемого результата для клиента. В контроллерах содержится бизнес логика. Чтобы написать Unit тесты для контроллеров приложения, сначала установим зависимости для тестирования. В нашем случае jest и supertest.

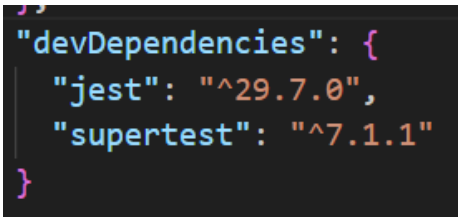



Рисунок П.2.2.1 – Зависимости jest и supertest

После успешной установки создадим папку __tests__ в корневой директории Бэкенда. А в ней создадим файлы тестирования для каждого контроллера с расширением test.js

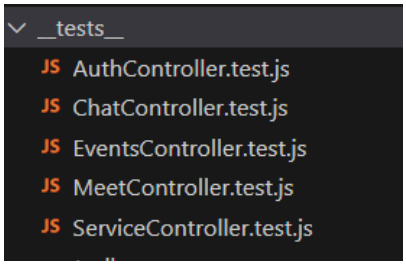



Рисунок П.2.2.2 – Директория тестов контроллеров

В контроллере аутентификации содержатся методы для работы с авторизацией, аутентификацией, регистрацией пользователя. Это основной контроллер приложения, и он должен быть протестирован. На каждый метод напишем тест в **AuthController.test.js**:

```
test('registerUser - успешная регистрация', async () => {
  const res = await request(app)
    .post('/api/auth/signup')
    .send({
      name: 'Test User',
      username: 'testuser',
      email: 'testuser@example.com',
      password: 'Password123',
      photo: "uploads/defaultUpload.jpg",
      role: 'Пользователь',
      balance: 0,
    });
  expect(res.statusCode).toBe(200);
  expect(res.body.success).toBe(true);
  expect(res.body.authtoken).toBeDefined();

  token = res.body.authtoken;
  const decoded = jwt.verify(token, SECRET_KEY);
  userId = decoded.user.id;
});
```

Рисунок П.2.2.3 – Пример кейса успешной регистрации

## ПРИЛОЖЕНИЕ В

Данный тест кейс позволит (Рисунок П.2.2.3) нам авторизовать пользователя для тестирования, так как некоторый функционал приватен и работает только для авторизованного пользователя.

```
beforeAll(async () => {
  const userRes = await request(app)
    .post('/api/auth/signup')
    .send({
      name: 'Event User',
      username: 'eventuser',
      email: 'eventuser@example.com',
      password: 'Password123',
      role: 'Пользователь',
      balance: 1000,
      photo: "uploads/defaultUpload.jpg"
    });
  token = userRes.body.authtoken;
  const decoded = jwt.decode(token);
  userId = decoded.user.id;
});
```

Рисунок П.2.2.4 – Заглушка тестового пользователя (EventsController.test.js)

EventController работает с данными мероприятий (Рисунок П.2.2.4, Рисунок П.2.2.5). Это основной контроллер, так как мероприятия – основная сущность проекта. Напишем тесты для каждого метода в как показано в примере EventsController.test.js

```
test('getEvents - получить список событий пользователя', async () => {
  const res = await request(app)
    .get('/api/events/list')
    .set('auth-token', token);
  expect(res.statusCode).toBe(200);
  expect(Array.isArray(res.body)).toBe(true);
  expect(res.body.length).toBeGreaterThan(0);
});
```

Рисунок П.2.2.5 – получение списка мероприятий пользователя

Контроллер чатов (**ChatController.js**) содержит в себе методы взаимодействия с моделью чатов. Чат является основным компонентом приложения, так как в нём происходит коммуникация между клиентом и подрядчиком. Это основа бизнес-логики приложения. Покроем тестами каждый метод этого контроллера: такие как создание чата для событий, открытие чата по мероприятию, оправка сообщения в чат, удаление сообщения из чата и удаление самого чата. Но перед этим создадим заглушку пользователя и мероприятия (Рисунок П.2.2.6).

```
    }],
    date_created: new Date(),
    location: {
      lat: 43.238949,
      lng: 76.889709
    }
  });
  eventId = eventRes._id.toString();
});
```

Рисунок П.2.2.6 – Заглушка пользователя и мероприятия

## ПРИЛОЖЕНИЕ В

```
test('appendMessage - добавить сообщение в чат', async () => {
  const res = await request(app)
    .post(`/api/chat/chat-message/${chatId}`)
    .set('auth-token', token)
    .send({
      content: 'Второе сообщение',
      sender: {
          _id: userId,
          name: 'Chat User',
          photo: 'uploads/defaultUpload.jpg',
          username: 'chatuser',
          email: "chatuser@example.com",
          role: "Пользователь",
          balance: 0
      },
      chatId,
      timestamp: new Date().toISOString()
  });
  expect(res.statusCode).toBe(201);
  expect(res.body.success).toBe(true);
  expect(res.body.message.message).toBe('Второе сообщение');
});
```

Рисунок П.2.2.7 – Добавление сообщения в чат

На картинке выше (Рисунок П.2.2.7) показан пример теста для отправки сообщения в чат. Такими тестами были покрыты кейсы событий с чатом, ошибки которых могли бы нарушить функциональность всего приложения.

### 2.3 Frontend тестирование

Frontend тестирование отличается от backend тестирования. Потому что тестируются не просто какие-то функции и куски кода, но и представление того, что увидит пользователь. Тестируется удобство использования и целостность компонентов представления. Но не всегда удается проверить компонент функционально, ведь User Interface имеет свойство меняться. Для этого будет использовать такая технология, как Screen тесты.

Screen тесты – это метод автоматизированного тестирования, который проверяет корректность пользовательского интерфейса путём создания и сравнения скриншотов.

Суть метода в том, что на этапе разработки сохраняются эталонные скриншоты интерфейса, при запуске тестов, создаются новые скриншоты приложения и производится сравнение с эталонными. Если есть различия, тест отмечается как «непройденный».

Данным методом проверяется кроссбраузерность, адаптивность, верстка и эффекты анимаций. В данном динамическом приложении они необходимы. К примеру, в Message.test.js (Рисунок П.2.3.1) проверяется компонент сообщения.

## ПРИЛОЖЕНИЕ В

```
describe('Message component', () => {
  const baseProps = {
    chat: {
      _id: 'msg1',
      user_id: '1',
      name: 'Test User',
      message: 'Привет, это тест!',
      timestamp: new Date().toISOString(),
    },
    user: { _id: '1', name: 'Test User', username: 'testuser' },
    darkTheme: '',
    onImageClick: jest.fn(),
    onDelete: jest.fn(),
  };

  test('рендерит текстовое сообщение', () => {
    render(<Message {...baseProps} />);
    expect(screen.getByText('Test User')).toBeInTheDocument();
    expect(screen.getByText(/Привет, это тест!/)).toBeInTheDocument();
  });
```

Рисунок П.2.3.1– Пример проверки компонента Message

```
test('вызывает onDelete при клике на корзину', () => {
  render(<Message {...baseProps} />);
  fireEvent.mouseEnter(screen.getByText('Test User'));
  const deleteBtn = screen.getByTitle('Удалить сообщение');
  fireEvent.click(deleteBtn);
  expect(baseProps.onDelete).toHaveBeenCalled();
});
```

Рисунок П.2.3.2 – Пример кейса вызова onDelete при клике на корзину

Screen тестами проверяется содержит ли элемент сообщения текст, который должен отобразится . Если совпадения на скриншоте найдено не будет – тест упадет, т.е. он не пройден.

Также мы используем mock данные на примере Chat.test.js (Рисунок П.2.3.4). Mock данные - это данные которые могли бы использоваться в боевой версии приложения, но не затрагивают базу данных, фиктивную реализацию интерфейса. Такие данные создаются по типам полей в модели, но не записываются в настоящую базу данных, что упрощает тестирование, и не нарушает целостность рабочей версии приложения.

```
jest.mock('../../api/chatApi', () => ({
  getAllChats: jest.fn(() => Promise.resolve({ success: true, chats: [] })),
  fetchChatByUser: jest.fn(() => Promise.resolve({ success: false })),
  fetchChatByEvent: jest.fn(() => Promise.resolve({ success: false })),
  createPrivateChat: jest.fn(() => Promise.resolve({ success: false })),
  createGroupChat: jest.fn(() => Promise.resolve({ success: false })),
  fetchChat: jest.fn(() => Promise.resolve({ success: false })),
  sendMessage: jest.fn(),
  uploadFile: jest.fn(() => Promise.resolve({ success: false })),
  deleteChat: jest.fn(() => Promise.resolve({ success: true })),
  deleteMessage: jest.fn(() => Promise.resolve({ success: true })),
}));
jest.mock('../../api/eventsApi', () => ({
  fetchEventById: jest.fn(() => Promise.resolve({ success: false })),
  updateCollaborators: jest.fn(() => Promise.resolve({ success: true })),
}));
```

Рисунок 2.3.3 – Пример «Mock» данных из методов chatApi и eventsApi

На данной картинке (Рисунок 2.3.3) создаются фиктивные методы Api, которые в кейсах тестирования должны были вернуть определенные данные. Таким образом компонент чат был покрыт тестами Chat, его внутренние компоненты и не затронули рабочую версию данных.

На примере картинки ниже (Рисунок П.2.3.4) были созданы фиктивные вложенные компоненты в EventOpen, переданы данные в props, переданы данные в веб-перехватчик и

созданы screen тесты на каждый вложенный компонент, что упростило задачу и код, так как создали тестовый массив, объект или функцию для каждого пропса.

```
// Мокаем все вложенные компоненты
jest.mock('./EventOpen/EventActions', () => () => <div>EventActions</div>);
jest.mock('./EventOpen/EventBudgetEditor', () => () => <div>EventBudgetEditor</div>);
jest.mock('./EventOpen/EventCollaborators', () => () => <div>EventCollaborators</div>);
jest.mock('./EventOpen/EventDatePickers', () => () => <div>EventDatePickers</div>);
jest.mock('./EventOpen/EventDescriptionEditor', () => () => <div>EventDescriptionEditor</div>);
jest.mock('./EventOpen/EventHeader', () => () => <div>EventHeader</div>);
jest.mock('./EventOpen/EventLocationPicker', () => () => <div>EventLocationPicker</div>);
jest.mock('./EventOpen/EventModals', () => () => <div>EventModals</div>);
jest.mock('./EventOpen/EventPeopleEditor', () => () => <div>EventPeopleEditor</div>);
jest.mock('./EventOpen/EventTagsEditor', () => () => <div>EventTagsEditor</div>);
jest.mock('./EventOpen/EventTasksSection', () => () => <div>EventTasksSection</div>);
jest.mock('../../hooks/useEventOpen', () => ({
  useEventOpen: () => ({
    user: { _id: '1', name: 'Test User' },
    tasks: [],
    setTasks: jest.fn(),
    spentBudget: 0,
    enableSave: true,
    setEnableSave: jest.fn(),
    enableDelete: true,
    setEnableDelete: jest.fn(),
    enableFinalExit: true,
    setEnableFinalExit: jest.fn(),
    admin: { _id: '1', name: 'Test User' },
    setAdmin: jest.fn(),
```

Рисунок П.2.3.4 – Тест компонента EventOpen

## 2.4 Результаты тестирования

Теперь нужно проверить тесты. Для этого нужно пройти в контейнер Бэкенда и выполнить команду «npm test»( Рисунок П.2.4.1).

```
# npm test

> backend@1.0.0 test
> jest --runInBand

 PASS  __tests__/EventsController.test.js
  ● Console

    console.log
      Connected to MongoDB

      at log (db.js:17:17)

 PASS  __tests__/AuthController.test.js
  ● Console

    console.log
      Connected to MongoDB

      at log (db.js:17:17)

 PASS  __tests__/ServiceController.test.js
  ● Console

    console.log
      Connected to MongoDB

      at log (db.js:17:17)

 PASS  __tests__/ChatController.test.js
  ● Console

    console.log
      Connected to MongoDB

      at log (db.js:17:17)
```

Рисунок П.2.4.1 – результат тестов backend

## ПРИЛОЖЕНИЕ В

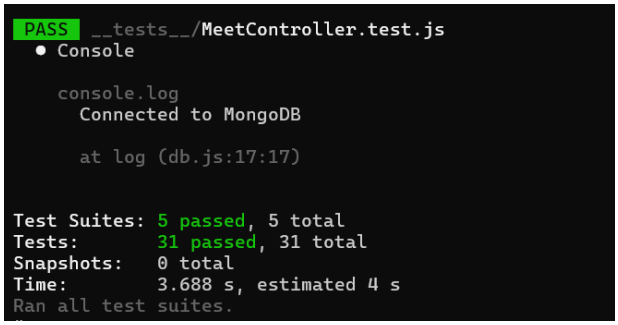



Рисунок П.2.4.2– Суммарный результат тестов backend

Тестирование прошло успешно. 5 тест кейсов и 31 тест пройдены без остановки (Рисунок 2.4.2). Что говорит о целостности и устойчивости серверной части нашего приложения. Тест был выполнен за 3.688 секунды.

Далее показаны тесты для frontend части:


Рисунок П.2.4.3 – результат тестов frontend(1)


Рисунок П.2.4.4 – результат тестов frontend(2)

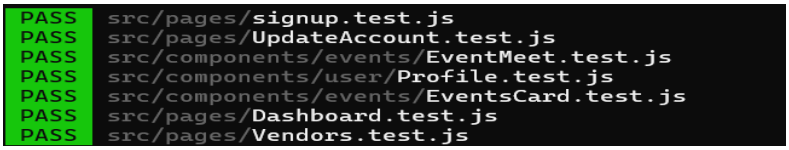



Рисунок П.2.4.5 – результат тестов frontend(3)

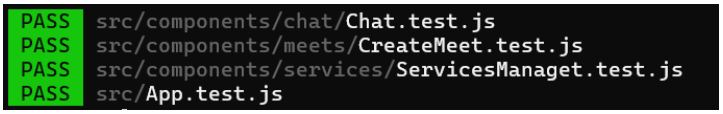



Рисунок П.2.4.6 – результат тестов frontend(4)

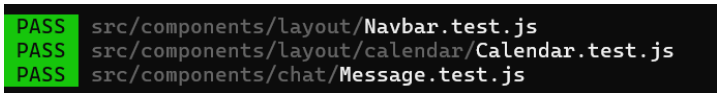



Рисунок П.2.4.7 – результат тестов frontend(5)

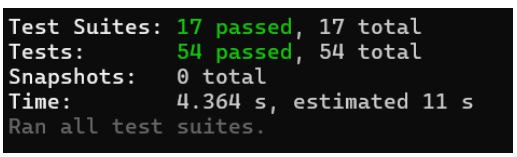



Рисунок П.2.4.8 – Суммарный результат тестов frontend

## ПРИЛОЖЕНИЕ В

Все 17 тест кейсов пройдены, все 54 теста пройдены успешно. Время выполнения: 4.364 s. Это говорит о том, что frontend часть вэб-приложения устойчива к разного вида ошибкам.

### 2.5 Выводы по тестированию

Протестировав автоматическими тестами наше приложения, мы убедились, что функционал приложения полностью работоспособен и готов к боевому использованию. Тестами были покрыты все основные элементы и ядро приложения.

Тесты показали не только надежность, но и высокую производительность приложения. (не более 5 секунд на все тесты frontend части и не более 4 секунд на backend части). Screen тесты показали соответствие задумке дизайна и отсутствие дефектов. При проверке на уязвимости угроз не выявлено. В Backend части мы имеем 31 тест, из которых 31 прошел успешно. Во Frontend части 54 теста, из которых 54 прошли успешно. Итого 85 тестов во всем приложении. Поставленные перед тестированием задачи были выполнены, веб приложение демонстрирует высокую надежность и удобство использования.

Таблица 1 – Результаты тестирования по тестовым наборам (Test Suites)

| Test Suite | Статус | Комментарий |
|---|---|---|
| EventsController.test.js | Пройден | Нет замечаний |
| ChatController.test.js | Пройден | Нет замечаний |
| ServiceController.test.js | Пройден | Нет замечаний |
| MeetController.test.js | Пройден | Нет замечаний |
| AuthController.test.js | Пройден | Нет замечаний |