Министерство науки и высшего образования РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
___________О.В. Непомнящий
"___"__________2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Приложение для учета взносов и расходов на проведение коллективного мероприятия

Руководитель	____________	_______	ст. преподаватель	О.В. Шмелёв
	подпись	*дата*	*должность, ученая степень*	
Выпускник	____________	_______		Л.Л. Скорик
	подпись	*дата*		
Нормоконтролёр	____________	_______	ст. преподаватель	О.В. Шмелёв
	подпись	*дата*	*должность, ученая степень*	

Красноярск 2023

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
__________О.В. Непомнящий
«___»__________2023 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Студенту Скорик Леониду Леонидовичу
фамилия, имя, отчество

Группа ЗКИ18-07Б Направление (специальность) 090301
номер код

Информатика и вычислительная техника
наименование

Тема выпускной квалификационной работы: Приложение для учета взносов и расходов на провидение коллективного мероприятия.

Утверждена приказом по университету № 7667/с от 17.05.2023

Руководитель ВКР: О.В. Шмелёв, ст. преподаватель кафедры ВТ ИКИТ СФУ
инициалы, фамилия, учёная степень, должность, место работы

Исходные данные для ВКР:

1) Анализ предметной области
2) Выбор средств разработки
3) Разработка базы данных
4) Разработка серверной части приложения
5) Разработка клиентской части приложения
6) Тестирование приложения

Перечень разделов ВКР:

1) Введение
2) Анализ предметной области
3) Реализация

Перечень графического материала:

1) Презентация в формате PowerPoint

Руководитель ВКР ____________ О.В. Шмелёв
подпись инициалы, фамилия

Задание принял к исполнению ____________ Л.Л. Скорик
подпись инициалы, фамилия

« » ______ ______ г.
дата

РЕФЕРАТ

Выпускная квалификационная работа по теме «Приложение для учета взносов и расходов на проведение коллективного мероприятия» содержит 57 страницу текстового документа, 31 иллюстрации, 2 таблицы, 1 приложение, 13 использованных источников.

ПРОГРАММА, ПРОЕКТИРОВАНИЕ, REACT, NODE.JS, POSTGRESQL, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА ПРИЛОЖЕНИЯ.

Цель работы – разработка приложения для учета взносов и доходов на проведение коллективного мероприятия.

Для достижения цели поставлены следующие задачи:

– Обзор предметной области;

– Выбор средств разработки;

– Разработка базы данных;

– Разработка серверной части системы;

– Разработка клиентской части системы;

– Тестирование автоматизированного рабочего места.

Содержание

Введение 6
1 Анализ предметной области 7
1.1 Анализ задания и обоснование выбранного решения 7
1.2 Постановка требования при выполнении задачи 7
1.3 Цели создания системы 8
1.4 Функциональные возможности 8
1.5 Анализ существующих решений 9
1.5.1 Splitwise 9
1.5.2 Toshl Finance 10
1.5.3 Tricount 11
2. Проектирование 13
2.1 Функциональная модель IDEF0 13
2.2 Диаграмма вариантов использования 14
2.3 Карта интерфейса 15
2.5 Архитектура приложения 16
2.5 Выбор средств разработки 17
2.6 Проектирование базы данных 27
3. Реализация 30
3.1 Серверная часть 30
3.2 Клиентская часть 40
Заключение 48
Список использованных источников 49
ПРИЛОЖЕНИЕ А 51

ВВЕДЕНИЕ

В наше время большинство мероприятий проводятся совместно, в коллективе. Это могут быть корпоративные мероприятия, дни рождения, свадьбы и другие торжества. Организация такого мероприятия требует значительных финансовых затрат.

В таких случаях, особенно если количество участников мероприятия большое, возникает необходимость ведения учета взносов и расходов. Организаторы мероприятия должны знать, сколько денег уже собрано, сколько еще необходимо собрать, какие расходы были совершены и сколько денег осталось на счету.

Для упрощения этой задачи предлагается создание приложения для учета взносов и расходов на проведении коллективного мероприятия. Такое приложение поможет организаторам быстро и удобно отслеживать финансовый баланс мероприятия. Оно позволит автоматически рассчитывать, сколько денег уже собрано и сколько еще необходимо собрать, а также облегчит процесс контроля за расходами.

Таким образом, создание приложения для учета взносов и расходов на проведении коллективного мероприятия является актуальной задачей и поможет сделать организацию совместного мероприятия более простой и прозрачной.

Цель работы — разработка приложения для учета взносов и доходов на проведение коллективного мероприятия

Задачи работы:

1. Анализ предметной области
2. Выбор средств разработки
3. Разработка базы данных
4. Разработка серверной части приложения
5. Разработка клиентской части приложения

1 Анализ предметной области

1.1 Анализ задания и обоснование выбранного решения

Организация коллективных мероприятий требует значительных финансовых затрат. В таких случаях возникает необходимость ведения учета взносов и расходов. Организаторы и каждый член мероприятия должны знать, сколько денег необходимо, сколько еще необходимо собрать, сколько внес каждый участник, какие расходы были совершены и сколько денег осталось на счету. Однако вести такой учет вручную может быть довольно сложно и трудоемко. Для упрощения этого процесса предлагается создание приложения для учета взносов и расходов на проведении коллективного мероприятия.

1.2 Постановка требования при выполнении задачи

Целью текущей бакалаврской работы является разработка приложения для учета финансов коллективного мероприятия. Разработанная система должна упрощать отслеживание и учет затрат и взносов на мероприятии.

Система должна удовлетворять следующим требованиям:

– корректный учет финансов;

– возможность создавать мероприятия;

– возможность добавлять сотрудников к мероприятиям;

– возможность создавать отчеты по сотрудникам и по мероприятиям;

– создание общего склада, где будут храниться вещи.

Система должна выполнять следующие задачи:

– обеспечить учет взносов и затрат для представления их в удобном для пользователей виде;

Рассмотрим пример работы приложения. Допустим, что сотрудники кафедры решили устроить пикник в честь окончания учебного года. Они выбрали место и назначили ответственного. Ответственный в разрабатываемом

приложении создал мероприятие и добавил в него сотрудников. После этого он внес список необходимых вещей, примерные затраты на все. Приложение вычислило по сколько рублей необходимо скинуться на мероприятие. У каждого участника мероприятия эта сумма отображается на его балансе. Если какой-то участник купит что-то еще для пикника, то он сможет добавить это к общему списку. Также в приложении существует склад. Там хранятся вещи, которые остались с прошедших мероприятий или вещи, которые не заканчиваются. Например, палатки, складные стулья и стол. Это помогает видеть, что уже есть в организации.

1.3 Цели создания системы

Целями создания данной системы являются:

– автоматизация процесса учета финансов при проведении коллективного мероприятия;

– создание гибких финансовых отчетов о коллективных мероприятиях.

1.4 Функциональные возможности

Функциональные возможности системы:

– отслеживание взносов каждого участника мероприятия: чтобы обеспечить надлежащее финансовое управление мероприятием, важно отслеживать взносы, внесенные каждым участником. Этого можно достичь путем разработки четкой и организованной системы отслеживания платежей и обеспечения того, чтобы все участники были осведомлены о статусе своих взносов;

– отслеживание расходов на проведении мероприятия: отслеживание расходов на проведении мероприятия является важным аспектом организации любого мероприятия. Необходимо учитывать расходы на аренду помещения, закупку необходимого оборудования и материалов, закупку продуктов и

напитков для фуршета или ужина, а также оплату услуг профессиональных подрядчиков при необходимости. Важно заранее спланировать бюджет мероприятия, чтобы избежать непредвиденных расходов и обеспечить успешное проведение.;

– создание отчета о балансе каждого сотрудника: каждый сотрудник должен знать сколько и куда он должен заплатить, он должен знать о расходах как по каждому мероприятию, так и общую сумму за все прошедшие и запланированные мероприятия. Если сотрудник останется должным за мероприятие или если он заплатил больше, чем нужно было, то он также должен видеть это;

– создание отчета о расходах на мероприятие: отчет о расходах на мероприятие поможет увидеть в целом сколько необходимо было денег на мероприятие, какие категории трат были. Также должен быть общий отчет для всех мероприятий;

– создание склада: в этой таблице хранятся вещи, которые уже есть в коллективе и которыми можно пользоваться при организации мероприятий.

1.5 Анализ существующих решений

Существует множество приложений для учета доходов и расходов, которые можно использовать для организации коллективного учета. Ниже рассмотрены три примера.

1.5.1 Splitwise

Splitwise — это отличное приложение для тех, кто хочет держать свои расходы под контролем. Оно позволяет пользователям создавать группы и делиться расходами с другими участниками группы, что в свою очередь может помочь снизить финансовые затраты каждого участника.

Кроме того, каждый участник может добавлять свои расходы и указывать, кому они принадлежат. Вы можете добавлять все расходы на еду, аренду, проезд и т.д., и Splitwise автоматически рассчитывает, кто сколько должен или кому сколько должны.

Также, Splitwise имеет множество дополнительных функций, таких как уведомления об оплатах, возможность добавлять несколько валют, а также функции для управления несколькими счетами.

Используя Splitwise, вы можете быть уверены, что ваши финансы находятся под контролем, вы сможете избежать недоразумений с друзьями или соседями по квартире и даже сократить свои расходы.

1.5.2 Toshl Finance

Toshl Finance — это приложение для учета финансов, которое позволяет пользователям удобно контролировать свои расходы и доходы. Но это далеко не все, что может предложить приложение.

Дополнительной функцией является возможность создания групп и распределения расходов между участниками. Это особенно полезно, если вы планируете коллективное мероприятие или совместную покупку.

Однако, возможности приложения не ограничиваются только этими функциями. Функция "чета" поможет пользователям определять, кто должен кому сколько денег. Это очень удобно, если вы общаетесь с друзьями или коллегами и делитесь расходами.

Кроме того, приложение позволяет создавать множество категорий расходов, чтобы вы могли точно отслеживать свои траты.

Таким образом, использование Toshl Finance поможет вам не только контролировать свои расходы, но и упростит финансовое планирование в вашей жизни.

1.5.3 Tricount

Tricount — это удобное приложение для учета расходов, которое помогает не только следить за своими затратами, но и делиться ими в группах. Это особенно удобно в поездках или при организации совместного мероприятия.

Помимо функции учета расходов, приложение также обладает функцией "баланс", которая показывает, кто должен сколько или кому сколько должны. Это позволяет избежать недопониманий и конфликтов в группе.

Более того, приложение позволяет создавать список покупок, что упрощает планирование и организацию покупок в группе.

В общем, Tricount — это отличный помощник в организации совместных мероприятий и поездок, который позволяет с легкостью управлять расходами и избежать недопониманий в группе.

Каждое из этих приложений может быть использовано для организации коллективного учета доходов и расходов.

Необходимо отметить, что каждое из этих приложений имеет свои недостатки:

– Splitwise: приложение не предлагает возможности вести учет расходов и доходов в разных валютах, что может быть проблематично для пользователей, живущих в разных странах. Кроме того, приложение не позволяет отслеживать долги между пользователями в режиме реального времени, что может привести к недопониманиям.

– Toshl Finance: приложение не всегда точно определяет категории расходов, что может привести к неточностям в учете. Также, некоторые пользователи могут считать его интерфейс слишком сложным и запутанным.

– Tricount: приложение не предлагает возможности создавать группы с более чем тремя участниками, что может быть неудобно для больших групп. Кроме того, некоторые пользователи могут считать его дизайн устаревшим и неудобным.

В таблице 1 приведена сравнительная характеристика рассмотренных аналогов.

Таблица 1 – Сравнительная характеристика аналогов

Критерий	Splitwise	Toshl Finance	Tricount
Возможность создавать группы	+	+	+
Возможность делить расход между участниками группы	+	+	+
Уведомления об оплатах	+	-	-
Каждый участник может добавить расходы	+	+	+
Наличие категорий расходов	-	+	-
Просмотр отчётов о задолженностях	+	+	+

2. Проектирование

Проектирование приложения — это процесс создания архитектуры и дизайна приложения, который обеспечивает его функциональность, надежность и удобство использования. Он включает в себя разработку бизнес-логики, выбор технологий, проектирование пользовательского интерфейса и многое другое.

2.1 Функциональная модель IDEF0

IDEF0 — методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является ее акцент на соподчиненность объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность.

Функциональная модель IDEF0 представляет собой набор блоков, каждый из которых представляет собой «черный ящик» со входами и выходами, управлением и механизмами, которые детализируются до необходимого уровня. Соединяются функции между собой при помощи стрелок и описаний функциональных блоков. При этом каждый вид стрелки или активности имеет собственное значение. Данная модель позволяет описать все основные виды процессов, как административные, так и организационные. Стрелки могут быть:

Входящие – вводящие, которые ставят определенную задачу.

Исходящие – выводящие результат деятельности.

Контроль (сверху вниз) – механизмы управления (положения, инструкции и пр).

Механизмы (снизу вверх) – что используется для того, чтобы произвести необходимую работу.

Функциональная модель для разрабатываемого приложения для учета взносов и расходов коллективного мероприятия представлена на рисунке 1.

Рисунок 1 – Функциональная модель IDEF0

На рисунке видно, что в бизнес-процессе участвуют сотрудники, контролируемые Законодательством Российской Федерации. На вход поступают следующие данные – данные о сотрудниках, данные о мероприятиях, данные о взносах, данные о расходах. На выходе получается отчет о взносах сотрудников, отчет о тратах мероприятия, отчет об остатках на складе. Каждый сотрудник сможет видеть на какое мероприятие и сколько денег он сдал или ему необходимо сдать. Отчет о мероприятиях будет включать в себя данные о том какие категория трат необходимы и сумму по каждому, а также количество человек, которые будут на мероприятие. В отчете об остатках на складе будет информация о том, какие товары остались с прошедших мероприятий.

2.2 Диаграмма вариантов использования

Диаграмма вариантов использования – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

В разрабатываемом приложении одна группа пользователей – сотрудники. Функции системы, созданные для этой группы представлены на рисунке 2.

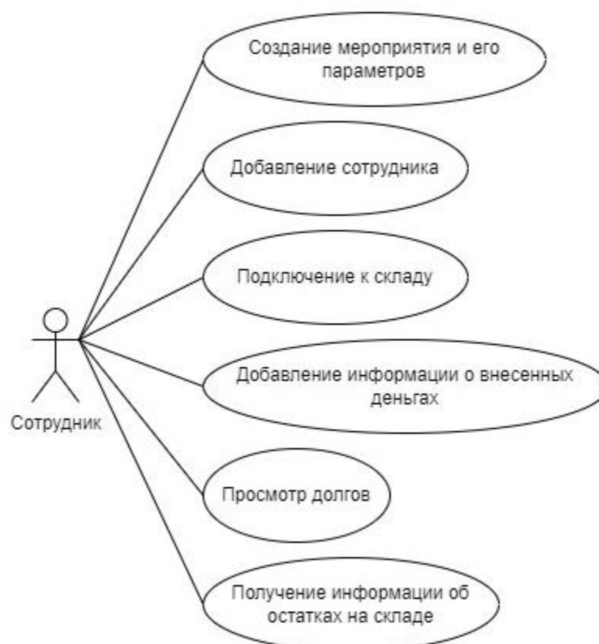


Рисунок 2 – Диаграмма вариантов использования

Сотрудники могут пользоваться всеми возможностями приложения: добавлять сотрудников, создавать мероприятия, подключать сотрудников к мероприятиям, подключаться к складу, просматривать остатки на складе, добавлять информацию о внесенных деньгах и потраченных на мероприятие деньгах, просматривать долги.

2.3 Карта интерфейса

Структура приложения показана на рисунке 3.

Рисунок 3 – Структура приложения

Прямоугольниками обозначены страницы интерфейса, стрелками – переходы из одной страницы в другую.

Через главный экран появляется возможность открыть три других экрана: «Сотрудники», «Мероприятия», «Склад».

На экране «Сотрудники» пользователь может просмотреть список сотрудников и добавить нового сотрудника в приложение.

На экране «Мероприятия» пользователь может просмотреть уже созданные мероприятия, названные в приложение, как текущие, просмотреть историю прошедших мероприятий или создать новое мероприятие, заполнив соответствующую форму.

Через экран «Склад» пользователь может получить информацию об остатках на складе. Это может быть как общий отчет, так и информация по конкретному мероприятию.

2.5 Архитектура приложения

В разрабатываемом приложение было решено использовать клиент-серверную архитектуру.

Клиент-серверная архитектура приложения — это подход к разработке приложений, при котором приложение разделено на две основные части - клиентскую и серверную. Клиентская часть является пользовательским интерфейсом, который взаимодействует с пользователем, а серверная часть обрабатывает запросы клиента и возвращает результаты.

Клиентская часть приложения — это пользовательский интерфейс, который взаимодействует с пользователем и отображает данные, полученные от серверной части. Клиентская часть может быть написана на разных языках программирования и может работать на разных платформах и устройствах, таких как веб-браузеры, мобильные устройства и настольные приложения. Клиентская

часть может использовать различные технологии, такие как HTML, CSS, JavaScript, React, Angular и другие.

Серверная часть приложения — это программа, которая обрабатывает запросы от клиентской части, выполняет необходимые операции и возвращает результаты. Серверная часть может быть написана на разных языках программирования, таких как Node.js, Python, Ruby и другие. Серверная часть может использовать различные технологии, такие как базы данных, API, веб-серверы и другие инструменты для обработки запросов и возврата данных клиентской части.

Клиент-серверная архитектура приложения позволяет разработчикам легко масштабировать и расширять приложение, так как клиентская часть и серверная часть могут быть разработаны и развернуты независимо друг от друга. Клиентская часть может быть развернута на разных платформах и устройствах, а серверная часть может быть развернута на разных серверах в зависимости от потребностей приложения.

2.5 Выбор средств разработки

Для разработки приложения необходимо выбрать средства для написания клиентской части приложения, серверной части приложения, базы данных.

Для начала рассмотрены средства разработки базы данных.

MongoDB — это документоориентированная база данных, которая позволяет хранить данные в виде документов BSON (Binary JSON). Она отличается от реляционных баз данных тем, что не требует жесткой схемы данных и может хранить данные разных типов и структур. MongoDB также обладает высокой масштабируемостью и производительностью благодаря возможности горизонтального масштабирования и индексации данных. Однако, MongoDB имеет ограниченную поддержку транзакций и требует определенных знаний и навыков для ее управления и настройки.

Плюсы использования MongoDB:

– Гибкость: MongoDB позволяет хранить данные в различных форматах и структурах, что делает ее очень гибкой и адаптивной к различным задачам.

– Масштабируемость: MongoDB легко масштабируется горизонтально, что позволяет ей обрабатывать большие объемы данных и высокие нагрузки.

– Высокая производительность: MongoDB обеспечивает быстрый доступ к данным и быстрые операции записи и чтения.

– Удобный язык запросов: MongoDB использует простой и интуитивно понятный язык запросов, что упрощает работу с данными.

Минусы использования MongoDB:

– Ограниченная поддержка транзакций: MongoDB поддерживает только ограниченный набор операций транзакций, что может быть проблемой для приложений, где требуется высокая консистентность данных.

– Сложность администрирования: MongoDB требует определенных знаний и навыков для управления и настройки базы данных.

– Ограниченный набор инструментов для анализа данных: MongoDB не предоставляет такой широкий набор инструментов для анализа данных, как реляционные базы данных.

MySQL — это реляционная база данных, которая поддерживает множество функций и возможностей для хранения и управления данными. Она используется для многих типов приложений, включая веб-приложения, системы управления содержимым и т.д. MySQL имеет открытый исходный код и является одной из самых популярных баз данных в мире.

Плюсы использования MySQL:

– Простота использования: MySQL предоставляет простой и интуитивно понятный язык запросов, что упрощает работу с данными.

– Высокая производительность: MySQL обеспечивает быстрый доступ к данным и быстрые операции записи и чтения.

– Широкий набор инструментов для анализа данных: MySQL предоставляет множество инструментов для анализа и обработки данных.

– Высокая поддержка транзакций: MySQL обеспечивает высокую консистентность данных с помощью транзакций.

Минусы использования MySQL:

– Лимитированная масштабируемость: MySQL может столкнуться с проблемами масштабирования при обработке больших объемов данных.

– Жесткая схема данных: MySQL требует строгой схемы данных, что может усложнить работу с данными.

– Ограниченные возможности хранения данных: MySQL имеет ограничения на количество данных, которые могут храниться в одной таблице.

PostgreSQL — это объектно-реляционная система управления базами данных (ОРСУБД), которая предоставляет широкий набор функций и возможностей для хранения и управления данными. Она используется для многих типов приложений, включая веб-приложения, системы управления содержимым и т.д. PostgreSQL обладает открытым исходным кодом и является одной из самых популярных баз данных в мире.

Плюсы использования PostgreSQL:

– Гибкость: PostgreSQL позволяет хранить данные в различных форматах и структурах, что делает ее очень гибкой и адаптивной к различным задачам.

– Масштабируемость: PostgreSQL легко масштабируется горизонтально и вертикально, что позволяет ей обрабатывать большие объемы данных и высокие нагрузки.

– Высокая производительность: PostgreSQL обеспечивает быстрый доступ к данным и быстрые операции записи и чтения.

– Высокая поддержка транзакций: PostgreSQL обеспечивает высокую консистентность данных с помощью транзакций и поддерживает множество уровней изоляции транзакций.

– Широкий набор инструментов для анализа данных: PostgreSQL предоставляет множество инструментов для анализа и обработки данных.

– Большое сообщество разработчиков: PostgreSQL имеет крупное сообщество разработчиков и пользователей, что обеспечивает ее постоянное развитие и поддержку.

Минусы использования PostgreSQL:

– Сложность администрирования: PostgreSQL требует определенных знаний и навыков для управления и настройки базы данных.

– Ограниченные возможности хранения данных: PostgreSQL имеет ограничения на количество данных, которые могут храниться в одной таблице.

– Ограниченные возможности масштабирования: PostgreSQL может столкнуться с проблемами масштабирования при обработке больших объемов данных.

В таблице 2 показаны сравнительные характеристики и концепции PostgreSQL и MongoDB.

Таблица 2 – Сравнение SQL и NoSQL базы данных

Характеристика	**SQL базы данных**	**NoSQL базы данных**
Типы и структуры данных	Информация хранится в таблицах, связанных между собой. В таблице имеется набор столбцов, каждый из которых соответствует определённому типу данных.	Данные хранятся в документах, коллекциях или графах. Документ — это структурированный контейнер для хранения данных в формате пар ключ-значение, где пары могут иметь разные типы данных. Коллекция — это группа документов, связанных между собой. Граф — это набор вершин и связей между ними.
Способы хранения данных и производительность	Данные в таблицах структурированы по строгим правилам и могут быть связаны с помощью внешних ключей.	Нет строгих правил для организации данных, что делает их более гибкими для хранения различных типов данных и структур

Надёжность и устойчивость	Используется транзакционная модель, которая позволяет сохранять целостность данных и обеспечивать ACID свойства для отказоустойчивости и надёжности	Применяется распределённая архитектура, чтобы повысить надёжность и отказоустойчивость
Гибкость и масштабируемость	Более ограничены в гибкости структуры данных	Легко масштабируются горизонтально, добавляя новые серверы и распределяя нагрузку между ними

После анализа средств разработки базы данных было решено использовать в данной работе PostgreSQL.

Современная frontend разработка — это комплексный процесс, который включает в себя множество технологий и инструментов. Она охватывает не только создание дизайна и верстку сайта, но также и его функциональное наполнение, и оптимизацию.

Основными технологиями, которые используются в разработке frontend, являются HTML, CSS и JavaScript. Они позволяют создавать различные элементы интерфейса, определять их стили и обеспечивать интерактивность на странице.

Однако, использование этих технологий само по себе недостаточно для создания современного приложения. Здесь на помощь приходят фреймворки и библиотеки, такие как React, Vue.js и Angular. Они позволяют ускорить процесс разработки, упростить создание сложных интерфейсов и обеспечить более высокую производительность.

Далее рассмотрены средства разработки клиентской части приложения.

Angular — это фреймворк для разработки веб-приложений, созданный компанией Google и используемый многими разработчиками по всему миру. Он позволяет быстро и эффективно создавать сложные и масштабируемые

приложения, благодаря своей компонентной архитектуре и декларативному подходу к созданию пользовательского интерфейса.

Angular также предоставляет множество инструментов и библиотек для упрощения процесса разработки, таких как маршрутизация, формы, валидация и тестирование. Эти инструменты позволяют создавать приложения более быстро и безопаснее.

Кроме того, Angular имеет широкую поддержку сообщества, что облегчает поиск решений и помощи при возникновении проблем. Это позволяет разработчикам быстрее решать проблемы и ускорять процесс разработки.

Плюсы использования Angular:

– Мощный функционал: Angular предоставляет широкий набор инструментов и функций для создания сложных веб-приложений.

– Высокая производительность: Angular обеспечивает быстрый доступ к данным и быстрые операции записи и чтения.

– Широкий набор инструментов для анализа данных: Angular предоставляет множество инструментов для анализа и обработки данных.

– Поддержка многих платформ: Angular может использоваться для создания веб-приложений на разных платформах, включая мобильные устройства и настольные компьютеры.

– Высокая степень модульности: Angular позволяет создавать и использовать модули, что делает его очень гибким и адаптивным к различным задачам.

Минусы использования Angular:

– Высокий порог входа: Angular требует определенных знаний и навыков для его использования, что может усложнить работу с ним.

– Ограниченные возможности масштабирования: Angular может столкнуться с проблемами масштабирования при обработке больших объемов данных.

Vue.js – это прогрессивный фреймворк для создания пользовательских интерфейсов, который позволяет создавать динамические одностраничные

приложения (SPA). Этот фреймворк обеспечивает возможность переиспользования компонентов интерфейса, что упрощает разработку и позволяет экономить время. Многие разработчики выбирают Vue.js из-за его легковесности и простоты в использовании. Кроме того, Vue.js обладает широкими возможностями для масштабирования проектов и удобными инструментами для тестирования кода. В целом, использование Vue.js позволяет создавать качественные и профессиональные интерфейсы для пользователей.

Плюсы использования Vue.js:

– Легковесность: Vue.js является легковесным и быстрым фреймворком, что позволяет создавать быстрые и отзывчивые веб-приложения.

– Простота использования: Vue.js обладает простым и интуитивно понятным синтаксисом, что делает его доступным для начинающих разработчиков.

– Гибкость: Vue.js позволяет гибко настраивать и расширять его функциональность с помощью плагинов и дополнительных библиотек.

– Высокая скорость разработки: Vue.js облегчает процесс разработки и позволяет быстро создавать компоненты и интерфейсы.

– Хорошая поддержка документации: Vue.js имеет качественную документацию и большое сообщество разработчиков, что обеспечивает его постоянное развитие и поддержку.

Минусы использования Vue.js:

– Ограниченные возможности масштабирования: Vue.js может столкнуться с проблемами масштабирования при обработке больших объемов данных.

– Ограниченная функциональность: Vue.js может не подходить для создания очень сложных приложений, так как не имеет такой широкой функциональности, как Angular.

React – это библиотека для создания пользовательских интерфейсов, которая позволяет создавать динамические одностраничные приложения (SPA) и многокомпонентные интерфейсы. React обеспечивает быстрый доступ к

данным и быстрые операции записи и чтения. React также обладает высокой степенью модульности, что позволяет создавать и использовать компоненты, что делает его очень гибким и адаптивным к различным задачам.

Плюсы использования React:

– Высокая производительность: React обеспечивает быстрый доступ к данным и быстрые операции записи и чтения.

– Гибкость: React позволяет гибко настраивать и расширять его функциональность с помощью плагинов и дополнительных библиотек.

– Высокая степень модульности: React позволяет создавать и использовать компоненты, что делает его очень гибким и адаптивным к различным задачам.

– Широкий набор инструментов для анализа данных: React предоставляет множество инструментов для анализа и обработки данных.

– Поддержка многих платформ: React может использоваться для создания веб-приложений на разных платформах, включая мобильные устройства и настольные компьютеры.

Минусы использования React:

– Необходимость использования JSX: React использует синтаксис JSX, который может быть непривычным для некоторых разработчиков.

– Необходимость использования инструментов сборки: React требует использования инструментов сборки, таких как Webpack или Gulp.

– Необходимость использования Redux: для управления состоянием приложения в React часто используется библиотека Redux, что может привести к усложнению кода приложения.

Рассмотрев вышеописанные варианты разработки клиентской части приложения было решено использовать React.

Современная backend-разработка включает в себя множество технологий и инструментов, которые позволяют создавать высокопроизводительные и масштабируемые приложения. Некоторые из наиболее популярных технологий включают в себя: Node.js, Express.js, Nest.js. Однако, одних технологий недостаточно для создания качественного backend-приложения. Важно также

учитывать требования к безопасности, масштабируемости и производительности.

Node.js — это платформа для создания высокопроизводительных приложений на языке JavaScript, которая работает на серверной стороне. Node.js использует единую потоковую модель для обработки запросов, что делает его очень масштабируемым и быстрым. Node.js также предоставляет широкий набор инструментов и библиотек для создания веб-приложений и API.

Плюсы использования Node.js:

– Высокая производительность: благодаря своей асинхронной потоковой модели, Node.js обеспечивает высокую производительность и быстрый доступ к данным.

– Масштабируемость: Node.js позволяет масштабировать приложения при росте нагрузки.

– Единая потоковая модель: Node.js использует единую потоковую модель для обработки запросов, что делает его очень масштабируемым и быстрым.

– Широкий набор инструментов и библиотек: Node.js предоставляет широкий набор инструментов и библиотек для создания веб-приложений и API.

– Низкие затраты на разработку: Node.js облегчает процесс разработки и позволяет быстро создавать веб-приложения и API.

Минусы использования Node.js:

– Ограниченная поддержка старых браузеров: Node.js не поддерживает старые версии браузеров, что может быть проблемой для некоторых пользователей.

– Необходимость использования модулей для некоторых функций: Node.js не содержит некоторых функций, которые могут быть необходимы для создания приложения, что требует использования модулей.

Express.js — это фреймворк для Node.js, который облегчает создание веб-приложений и API. Express.js предоставляет множество инструментов и функций для создания быстрых и масштабируемых приложений.

Плюсы использования Express.js:

– Быстрая разработка приложений: Express.js облегчает процесс разработки и позволяет быстро создавать веб-приложения и API.

– Гибкость: Express.js позволяет гибко настраивать и расширять его функциональность с помощью плагинов и дополнительных библиотек.

– Низкие затраты на разработку: Express.js облегчает процесс разработки и позволяет быстро создавать веб-приложения и API.

– Широкий набор инструментов и библиотек: Express.js предоставляет широкий набор инструментов и библиотек для создания веб-приложений и API.

Минусы использования Express.js:

– Необходимость использования инструментов сборки: Express.js требует использования инструментов сборки, таких как Webpack или Gulp.

– Необходимость использования middleware: для некоторых функций, таких как обработка данных формы или аутентификация, требуется использование middleware, что может усложнить код приложения.

– Необходимость использования сторонних библиотек: для некоторых функций, таких как работа с базами данных, требуется использование сторонних библиотек, что может усложнить код приложения.

Nest.js — это фреймворк для Node.js, который облегчает создание масштабируемых и легко тестируемых веб-приложений. Nest.js использует TypeScript и предоставляет множество инструментов и функций для создания быстрых и масштабируемых приложений.

Плюсы использования Nest.js:

– TypeScript: Nest.js использует TypeScript, что облегчает разработку и улучшает качество кода.

– Масштабируемость: Nest.js позволяет создавать масштабируемые приложения.

– Легко тестируемый: Nest.js облегчает тестирование приложения благодаря использованию TypeScript и Dependency Injection.

– Гибкость: Nest.js позволяет гибко настраивать и расширять его функциональность с помощью плагинов и дополнительных библиотек.

– Низкие затраты на разработку: Nest.js облегчает процесс разработки и позволяет быстро создавать веб-приложения и API.

Минусы использования Nest.js:

– Необходимость использования TypeScript: для использования Nest.js необходимо знание TypeScript, что может быть проблемой для некоторых разработчиков.

– Сложность: Nest.js может быть сложным для новых пользователей из-за его широких возможностей и инструментов.

После анализа вышеописанных средств было принято решение использовать в работе Nest.js.

2.6 Проектирование базы данных

Для разработки приложения для учета взносов и расходов на проведение коллективного мероприятия была создана база данных, показанная на рисунке 4.

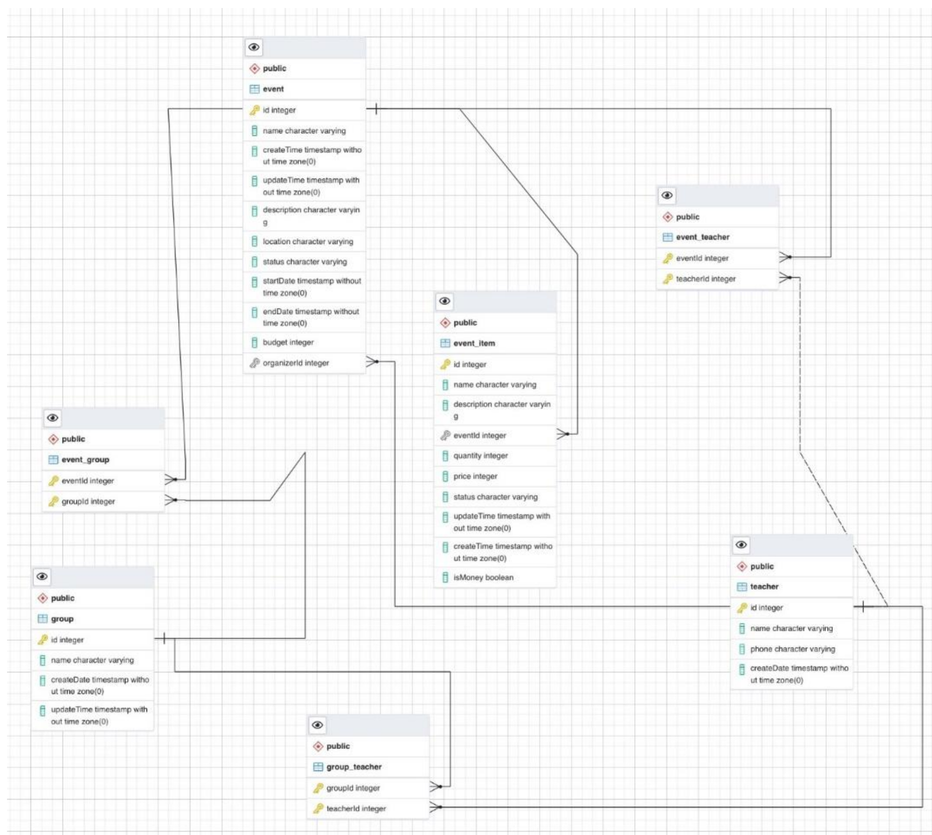


Рисунок 4 – База данных разрабатываемого приложения

База данных содержит в себе 4 основные таблицы: «Преподаватель», «Группа», «Мероприятие», «Склад (вещь для мероприятия)». Кроме этого, в базе данных есть ещё 3 дополнительные таблицы, которые связывают основные.

Таблица «Мероприятие» содержит в себе такие поля: название мероприятия (текстовый тип данных), описание (текстовый тип данных), место проведения мероприятия (текстовый тип данных), статус (текстовый тип данных), время и дата начала и окончания мероприятия, бюджет (числовой тип данных). Поле «Бюджет» хранит в себе сумму, необходимую, для проведения мероприятия. Эта сумма распределяется на количество участник, таким образом выясняется, сколько каждый сотрудник должен оплатит за это мероприятие.

Таблица «Вещи для мероприятия» - это склад со всеми вещами, которые необходимы для текущих мероприятий, а также предметов, которые уже есть в коллективе. Эта таблица содержит в себе следующие поля: название (текстовый тип данных), описание (текстовый тип данных), цена (числовой тип данных), количество (числовой тип данных), статус (текстовый тип данных). Также, есть поле «Деньги». Через это поле можно отметить денежный ресурс.

В таблице «Преподаватель» созданы такие поля: имя (текстовый тип данных), номер телефона (текстовый тип данных).

Таблица «Группа» имеет только одно поле – название (текстовый тип данных).

Также, есть таблицы для связи основный таблиц – группа и преподаватель, преподаватель и мероприятие, мероприятие и группа. Таблицы связаны с помощью связи многие ко многим.

Связь многие ко многим (Many-to-Many) — это тип связи, когда каждая запись из одной таблицы может быть связана с несколькими записями из другой таблицы, и наоборот. Это означает, что одной записи может соответствовать несколько записей из другой таблицы, а наоборот - одной записи из другой таблицы может соответствовать несколько записей из первой таблицы.

Для реализации связи многие ко многим в базе данных часто используют промежуточную таблицу, которая содержит ключи обеих таблиц, связываемых

друг с другом. Эта таблица позволяет связать множество записей из одной таблицы с множеством записей из другой таблицы.

Таблицы «Мероприятия» и «Вещи для мероприятия» связаны с помощью связи один ко многим.

Связь один ко многим (One-to-Many) — это тип связи, когда одна запись в одной таблице может быть связана с несколькими записями в другой таблице, но каждая запись во второй таблице может быть связана только с одной записью в первой таблице.

3. Реализация

3.1 Серверная часть

Для разработки серверной части использовалась чистая архитектура.

Чистая архитектура (Clean Architecture) — это концепция проектирования программного обеспечения, которая предлагает подход к организации кода и архитектуры, чтобы сделать приложение более понятным, гибким и легко изменяемым.

Основная идея чистой архитектуры заключается в том, что различные компоненты приложения должны быть разделены и изолированы друг от друга. Это позволяет легко изменять один компонент без влияния на другие. При этом, чистая архитектура определяет правила, которые должны быть соблюдены для достижения такой изоляции. В частности, это касается того, как компоненты зависят друг от друга, какие данные между ними передаются и как они обрабатываются.

В разрабатываемом приложении создано два слоя: домен и инфраструктура.

Слой домена — это центральный слой приложения, который содержит бизнес-логику и бизнес-правила. Этот слой не зависит от других слоев приложения и не зависит от фреймворков или библиотек. Слой домена должен быть написан на языке, который наиболее точно отображает бизнес-модель и бизнес-правила, которые реализуются приложением. Этот слой должен быть легко тестируемым, так как здесь находится основная логика приложения.

Слой инфраструктуры — это слой, который обеспечивает взаимодействие приложения с внешним миром. Сюда входят базы данных, сетевые сервисы, файловые системы и другие внешние компоненты. Слой инфраструктуры содержит адаптеры, которые связывают слой домена с внешним миром. Адаптеры — это классы, которые позволяют взаимодействовать с внешними компонентами, но скрывают детали реализации от слоя домена.

Были созданы три компонента: мероприятие, преподаватель, группа. У каждого компонента есть модель и контроллер. Они разделены друг от друга, что позволяет легко изменять один компонент без влияния на другой.

Модель (Model) — это компонент, который содержит бизнес-логику и бизнес-правила приложения. Это может быть набор классов, структур данных, сервисов или других компонентов, которые отвечают за обработку данных и выполнение бизнес-операций. Модель не зависит от пользовательского интерфейса или других компонентов приложения, что делает ее легко переносимой и тестируемой.

Контроллер (Controller) — это компонент, который обрабатывает пользовательский ввод и управляет взаимодействием между пользовательским интерфейсом и моделью. Контроллер может быть реализован в виде класса, который принимает запросы от пользовательского интерфейса, вызывает соответствующие методы модели и возвращает результаты обратно в пользовательский интерфейс. Контроллер может также отвечать за обработку ошибок и управление состоянием приложения.

На рисунках 5-7 показана модель сущности мероприятие.

```
15  export class EventModel {
16      @PrimaryKeyField()
17      id: number;
18
19      @StringField()
20      name: string;
21
22      @StringField()
23      description: string;
24
25      @StringField()
26      location: string;
27
28      @EnumField({
29          enum: EventStatusEnum,
30          defaultValue: EventStatusEnum.PLANNED,
31      })
32      status: string;
33
34      @RelationIdField({
35          relationName: 'teachers',
36          isArray: true,
37      })
38      teachersIds: number[];
39
40      @RelationField({
41          type: 'ManyToMany',
42          isOwningSide: true,
43          relationClass: () => TeacherModel,
```

Рисунок 5 – Модель сущности мероприятие

```
48      @RelationIdField({
49          relationName: 'groups',
50          isArray: true,
51      })
52      groupsIds: number[];
53
54      @RelationField({
55          type: 'ManyToMany',
56          isOwningSide: true,
57          relationClass: () => GroupModel,
58          isArray: true,
59      })
60      groups: GroupModel[];
61
62      @RelationIdField({
63          relationName: 'items',
64          isArray: true,
65      })
66      itemsIds: number[];
67
68      @RelationField({
69          type: 'OneToMany',
70          relationClass: () => EventItemModel,
71          isArray: true,
72          inverseSide: (item: EventItemModel) => item.event,
73      })
74      items: EventItemModel[];
75
76      @DateTimeField({
```

Рисунок 6 – Модель сущности мероприятие (продолжение)

```
78      })
79      startDate: string;
80
81      @DateTimeField({
82          skipSeconds: true,
83      })
84      endDate: string;
85
86      @IntegerField()
87      budget: number;
88
89      @RelationIdField({
90          relationName: 'organizer',
91      })
92      organizerId: number;
93
94      @RelationField({
95          type: 'ManyToOne',
96          relationClass: () => TeacherModel,
97      })
98      organizer: TeacherModel;
99
100     @CreateTimeField()
101     createTime: string;
102
103     @UpdateTimeField()
104     updateTime: string;
105 }
```

Рисунок 7 – Модель сущности мероприятие (продолжение)

На рисунках 8 и 9 показан контроллер сущности мероприятие.

```
import {Body, Controller, Delete, Get, Param, Post, Query} from '@nestjs/common';
import {ApiBody, ApiOkResponse, ApiQuery, ApiTags} from '@nestjs/swagger';
import {EventService} from '../../domain/services/EventService';
import {EventSearchDto} from '../../domain/dtos/EventSearchDto';
import {EventSchema} from '../schemas/EventSchema';
import {EventSaveDto} from '../../domain/dtos/EventSaveDto';

@Controller('/event')
@ApiTags()
export class EventController {
    constructor(
        private service: EventService,
    ) {
    }

    @Get()
    @ApiQuery({type: EventSearchDto})
    @ApiOkResponse({type: EventSchema, isArray: true})
    async get(@Query() dto: EventSearchDto) {
        return this.service.search(dto);
    }

    @Get('/:id')
    @ApiOkResponse({type: EventSchema, isArray: false})
    async getOne(@Param('id') id: number) {
        return this.service.findById(id);
    }

```

Рисунок 8 – Контроллер сущности мероприятие

```
    @Post()
    @ApiBody({type: EventSaveDto})
    @ApiOkResponse({type: EventSchema, isArray: false})
    async create(
        @Body() dto: EventSaveDto,
    ) {
        return this.service.create(dto);
    }

    @Post('/:id')
    @ApiBody({type: EventSaveDto})
    @ApiOkResponse({type: EventSchema, isArray: false})
    async update(
        @Body() dto: EventSaveDto,
        @Param('id') id: number,
    ) {
        return this.service.update(id, dto);
    }

    @Delete('/:id')
    async delete(@Param('id') id: number) {
        return this.service.remove(id);
    }
}
```

Рисунок 9 – Контроллер сущности мероприятие (продолжение)

В контроллере использовались запросы для обработки пользовательского ввода и выполнения бизнес-логики. HTTP-запросы веб-приложения обычно делятся на два типа: GET и POST.

GET-запросы — это запросы на получение данных с сервера. Контроллер может использовать GET-запросы для отображения данных на странице или для получения данных из базы данных или других источников. GET-запросы могут содержать параметры, которые передаются контроллеру для выполнения определенных операций.

POST-запросы — это запросы на отправку данных на сервер. Контроллер может использовать POST-запросы для создания, обновления или удаления данных на сервере. POST-запросы могут содержать данные формы или другие данные, которые передаются контроллеру для обработки.

В контроллерах веб-приложения GET-запросы обычно обрабатываются методами, которые возвращают представления (Views) для отображения данных на странице. Например, метод Index() контроллера может использоваться для отображения списка элементов в базе данных.

POST-запросы в контроллерах обычно обрабатываются методами, которые изменяют данные на сервере. Например, метод Create() контроллера может использоваться для создания нового элемента в базе данных на основе данных, отправленных пользователем из формы.

На рисунках 10 и 11 показана модель сущности преподаватель.

```
11  export class TeacherModel {
12      @PrimaryKeyField()
13      id: number;
14
15      @StringField()
16      name: string;
17
18      @StringField()
19      phone: string;
20
21      @RelationIdField({
22          relationName: 'groups',
23          isArray: true,
24      })
25      groupsIds: number[];
26
27      @RelationField({
28          type: 'ManyToMany',
29          relationClass: () => GroupModel,
30          isOwningSide: false,
31          isArray: true,
32      })
33      groups: GroupModel[];
34
35      @RelationIdField({
36          relationName: 'events',
37          isArray: true,
38      })
39      eventsIds: number[];
```

Рисунок 10 – Модель сущности преподаватель

```
41      @RelationField({
42          type: 'ManyToMany',
43          relationClass: () => EventModel,
44          isOwningSide: false,
45          isArray: true,
46      })
47      events: EventModel[];
48
49      @RelationIdField({
50          relationName: 'organizationEvents',
51          isArray: true,
52      })
53      organizationEventsIds: number[];
54
55      @RelationField({
56          type: 'OneToMany',
57          relationClass: () => EventModel,
58          inverseSide: (event: EventModel) => event.organizer,
59          isArray: true,
60      })
61      organizationEvents: EventModel[];
62
63      @CreateTimeField()
64      createDate: string;
65  }
```

Рисунок 11 – Модель сущности преподаватель (продолжение)

На рисунках 12 и 13 показан контроллер сущности преподаватель.

```
1   import {Body, Controller, Delete, Get, Param, Post, Query} from '@nestjs/common';
2   import {ApiBody, ApiOkResponse, ApiQuery} from '@nestjs/swagger';
3   import {TeacherService} from '../../domain/services/TeacherService';
4   import {TeacherSearchDto} from '../../domain/dtos/TeacherSearchDto';
5   import {TeacherSaveDto} from '../../domain/dtos/TeacherSaveDto';
6   import {TeacherSchema} from '../schemas/TeacherSchema';
7
8   @Controller('/teacher')
9   export class TeacherController {
10      constructor(
11          private service: TeacherService,
12      ) {
13      }
14
15      @Get()
16      @ApiQuery({type: TeacherSearchDto})
17      @ApiOkResponse({type: TeacherSchema, isArray: true})
18      async search(
19          @Query() dto: TeacherSearchDto,
20      ) {
21          return this.service.search(dto);
22      }
23
24      @Get('/:id')
25      @ApiOkResponse({type: TeacherSchema, isArray: false})
26      async getOne(@Param('id') id: number) {
27          return this.service.findById(id);
28      }
```

Рисунок 12 – Контроллер сущности преподаватель

```
30      @Post()
31      @ApiBody({type: TeacherSaveDto})
32      @ApiOkResponse({type: TeacherSchema, isArray: false})
33      async create(
34          @Body() dto: TeacherSaveDto,
35      ) {
36          return this.service.create(dto);
37      }
38
39      @Post('/:id')
40      @ApiBody({type: TeacherSaveDto})
41      @ApiOkResponse({type: TeacherSchema, isArray: false})
42      async update(
43          @Body() dto: TeacherSaveDto,
44          @Param('id') id: number,
45      ) {
46          return this.service.update(id, dto);
47      }
48
49      @Delete('/:id')
50      async delete(@Param('id') id: number) {
51          return this.service.remove(id);
52      }
53  }
```

Рисунок 13 – Контроллер сущности преподаватель (продолжение)

На рисунках 14 и 15 показана модель сущности группа.

```
12  export class GroupModel {
13      @PrimaryKeyField()
14      id: number;
15
16      @StringField()
17      name: string;
18
19      @RelationIdField({
20          relationName: 'teachers',
21          isArray: true,
22      })
23      teachersIds: number[];
24
25      @RelationField({
26          type: 'ManyToMany',
27          relationClass: () => TeacherModel,
28          isOwningSide: true,
29      })
30      teachers: TeacherModel[];
31
32      @RelationIdField({
33          relationName: 'events',
34          isArray: true,
35      })
36      eventsIds: number[];
37
38      @RelationField({
39          type: 'ManyToMany',
40          relationClass: () => EventModel,
```

Рисунок 14 – Модель сущности группа

```
40          relationClass: () => EventModel,
41          isOwningSide: false,
42          isArray: true,
43      })
44      events: EventModel[];
45
46      @RelationIdField({
47          relationName: 'items',
48          isArray: true,
49      })
50      itemsIds: number[];
51
52      @RelationField({
53          type: 'OneToMany',
54          relationClass: () => EventItemModel,
55          isArray: true,
56          inverseSide: (item: EventItemModel) => item.event,
57      })
58      items: EventItemModel[];
59
60      @CreateTimeField()
61      createDate: string;
62
63      @UpdateTimeField()
64      updateTime: string;
65  }
```

Рисунок 15 – Модель сущности группа (продолжение)

На рисунках 16 и 17 показан контроллер сущности группа.

```
1   import {Body, Controller, Delete, Get, Param, Post, Query} from '@nestjs/common';
2   import {ApiBody, ApiOkResponse, ApiQuery, ApiTags} from '@nestjs/swagger';
3   import {GroupService} from '../../domain/services/GroupService';
4   import {GroupSearchDto} from '../../domain/dtos/GroupSearchDto';
5   import {GroupSchema} from '../schemas/GroupSchema';
6   import {GroupSaveDto} from '../../domain/dtos/GroupSaveDto';
7
8   @Controller('/group')
9   @ApiTags('Группы')
10  export class GroupController {
11      constructor(
12          private service: GroupService,
13      ) {
14      }
15
16      @Get()
17      @ApiQuery({type: GroupSearchDto})
18      @ApiOkResponse({type: GroupSchema, isArray: true})
19      async search(
20          @Query() dto: GroupSearchDto,
21      ) {
22          return this.service.search(dto);
23      }
24
25      @Get('/:id')
26      @ApiOkResponse({type: GroupSchema, isArray: false})
27      async getOne(@Param('id') id: number) {
28          return this.service.findById(id);
29      }
```

Рисунок 16 – Контроллер сущности группа

```
    @Post()
    @ApiBody({type: GroupSaveDto})
    @ApiOkResponse({type: GroupSchema, isArray: false})
    async create(
        @Body() dto: GroupSaveDto,
    ) {
        return this.service.create(dto);
    }

    @Post('/:id')
    @ApiBody({type: GroupSaveDto})
    @ApiOkResponse({type: GroupSchema, isArray: false})
    async update(
        @Body() dto: GroupSaveDto,
        @Param('id') id: number,
    ) {
        return this.service.update(id, dto);
    }

    @Delete('/:id')
    async delete(@Param('id') id: number) {
        return this.service.remove(id);
    }
}
```

Рисунок 17 – Контроллер сущности группа (продолжение)

Сущности материал, преподаватель, группа запрограммированы с помощью методологии Dependency Injection.

Dependency Injection (DI) — это методология программирования, которая позволяет управлять зависимостями между компонентами приложения. Она предоставляет способ создания объектов, которые зависят от других объектов, без явного создания этих объектов внутри класса.

Суть DI заключается в том, что зависимости компонента передаются ему извне, например, через конструктор или свойства. Это позволяет создавать компоненты, которые могут быть использованы в различных ситуациях и на различных платформах.

В Nest.js, Dependency Injection предоставляет механизм для создания и вставки зависимостей в компоненты. Компоненты могут быть классами, сервисами, провайдерами или контроллерами.

Зависимости в Nest.js определяются с помощью провайдеров (providers). Провайдеры — это классы, которые предоставляют объекты или функции, которые могут быть вставлены в другие компоненты. Например, провайдер может предоставлять объект базы данных или функцию для обработки файлов.

Чтобы вставить зависимость в компонент, необходимо определить провайдер для этой зависимости и указать ее в конструкторе или свойстве компонента.

На рисунках 18 и 19 показан главный модуль Dependency Injection приложения.

```
25  @Module({
26      imports: [
27          ConfigModule.forRoot({
28              load: config,
29          }),
30          TypeOrmModule.forRootAsync({
31              imports: [ConfigModule],
32              inject: [ConfigService],
33              useFactory: (configService: ConfigService) => (configService.get('database') as PostgresConnectionOptions),
34          } as TypeOrmModuleAsyncOptions),
35          process.env.APP_ENVIRONMENT === 'dev' && ServeStaticModule.forRoot({
36              rootPath: process.env.APP_ROOT_FILES_DIR,
37              serveRoot: process.env.APP_STATIC_URL_PREFIX,
38              exclude: ['/api*'],
39          }),
40          process.env.APP_SENTRY_DSN && SentryModule.forRoot({
41              dsn: process.env.APP_SENTRY_DSN,
42              environment: process.env.APP_ENVIRONMENT,
43          }),
44          ScheduleModule.forRoot(),
45          GlobalModule,
46          InitModule,
47          CommandModule,
48          FileModule,
49          AuthModule,
50          UserModule,
51          NotifierModule,
52          TeacherModule,
53          GroupModule,
```

Рисунок 18 – Главный DI модуль приложения

```
53          GroupModule,
54          EventModule,
55      ].filter(Boolean),
56      providers: [
57          MigrateCommand,
58          {
59              provide: APP_FILTER,
60              useClass: ValidationExceptionFilterCustom,
61          },
62          {
63              provide: APP_FILTER,
64              useClass: RequestExecutionExceptionFilter,
65          },
66      ],
67  })
68  export class AppModule {
69  }
```

Рисунок 19 – Главный DI модуль приложения (продолжение)

3.2 Клиентская часть

Клиентская часть написана с помощью библиотеки React. При разработке приложений на React используются компоненты, которые могут быть повторно использованы и собраны в иерархическую структуру.

Архитектура React приложения обычно строится на основе Flux или Redux. Эти архитектурные паттерны предоставляют механизмы для управления состоянием приложения и обмена данными между компонентами.

Кроме того, в архитектуре React могут использоваться компоненты высшего порядка (HOC), контекст (context) и хуки (hooks). HOC — это функции, которые принимают компонент и возвращают новый компонент с дополнительными свойствами или функциональностью. Контекст позволяет передавать данные между компонентами без необходимости передавать их через все промежуточные компоненты. Хуки предоставляют механизмы для добавления состояния и функциональности в функциональные компоненты.

При разработке клиентской части использовалась библиотека Tanstack Query. Tanstack Query — это библиотека для работы с запросами данных на клиенте и сервере. Она использует концепцию декларативных запросов, что означает, что разработчики могут описывать, какие данные им нужны, а не как их получить.

Tanstack Query работает следующим образом:

– Описывается запрос на данные с помощью хука useQuery или других хуков, которые предоставляет библиотека.

– Когда компонент, содержащий хук useQuery, рендерится, Tanstack Query автоматически отправляет запрос на сервер, используя указанный URL и параметры.

– Пока запрос выполняется, компонент может показать загрузочный индикатор или другой элемент интерфейса, который сообщает пользователю, что данные загружаются.

– Когда запрос завершается, Tanstack Query сохраняет полученные данные в кеше и вызывает функцию обратного вызова, которую разработчик предоставил при определении запроса.

– Если данные изменились, Tanstack Query автоматически повторно выполнит запрос и обновит кеш.

Tanstack Query также поддерживает множество других функций, таких как оптимистические обновления, отмена запросов, кеширование и многое другое.

На рисунке 20 показаны маршруты приложения.

```
client > src > lib > TS routes.ts > ...
export const routes = [
  {
    label: 'Мероприятия',
    href: '/',
  },
  {
    label: 'Преподаватели',
    href: '/teachers',
  },
  {
    label: 'Группы',
    href: '/groups',
  },
]
```

Рисунок 20 – Маршруты приложения

На рисунке 21 показаны обращения к серверу по API.

```
client > src > lib > TS server.ts > ...
import axios from 'axios'

export const server = axios.create({
  baseURL: 'http://localhost:8080/api/v1',
})

export const getRecords = async <T>({
  pathname,
  params,
}: {
  pathname: '/groups' | '/teachers' | '/ivents'
  params?: Record<string, string | number>
}) => {
  return server
    .get<{
      total: number
      items: Array<T & { id: number }>
    }>(pathname, {
      params,
    })
    .then((res) => res.data)
}

```

Рисунок 21 – Обращения к серверу по API

Реализована возможность создавать мероприятия, группы, добавлять преподавателей.

Клиентская часть позволяет добавлять и хранить группы преподавателей и каждому участнику группы делать взносы. Все взносы рассчитываются общей суммой на группу. Эти группы можно прикреплять к мероприятиям, кроме этого, к мероприятиям можно добавлять отдельных пользователей. В результате работы программа просуммирует сколько поступило средств от групп и от отдельных пользователей и посчитает итоговую сумму мероприятия. Также, можно получить отчет о том, как использовались деньги и предметы. Деньги представлены в качестве предмета.

На главной странице размещен дашборд с текущими и ближайшими мероприятиями. Текущие мероприятия определяются следующим способом. Каждое мероприятие имеет дату начала и дату окончания. Если текущая дата входит в этот промежуток времени, то мероприятие считается текущим.

На рисунке 22 показан начальный экран приложения.

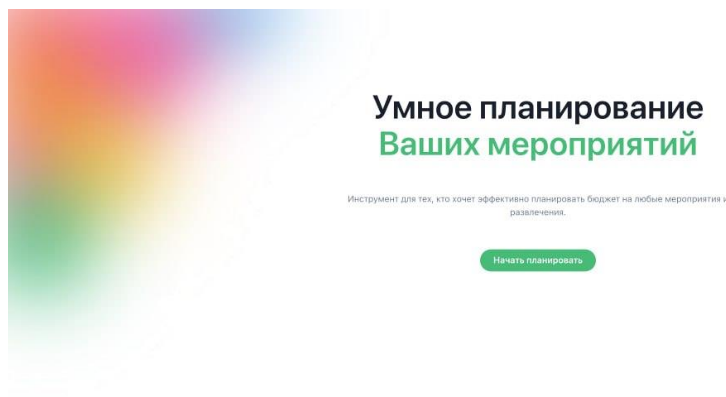


Рисунок 22 – Начальный экран приложения

На рисунке 23 показан экран регистрации.

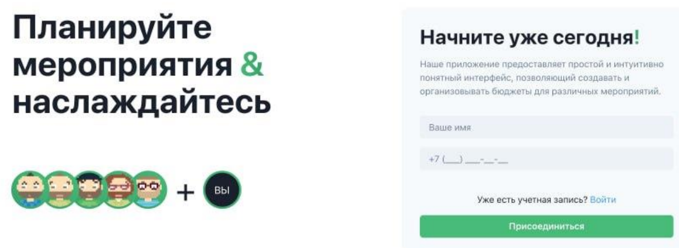


Рисунок 23 – Экран регистрации

На рисунке 24 представлен экран входа в приложение.

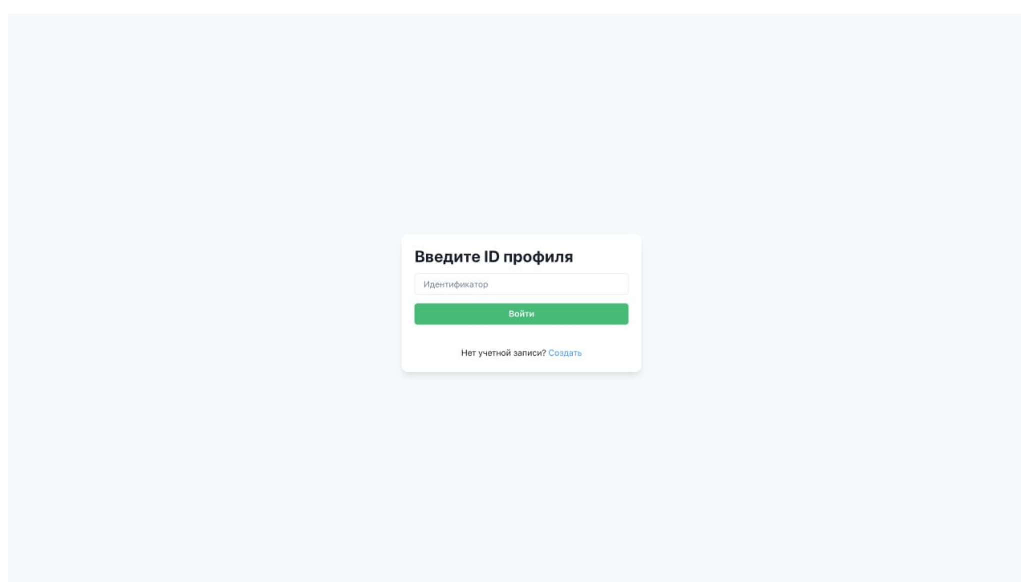


Рисунок 24 – Экран входа в приложение

На рисунке 25 показаны запланированные мероприятия пользователя.

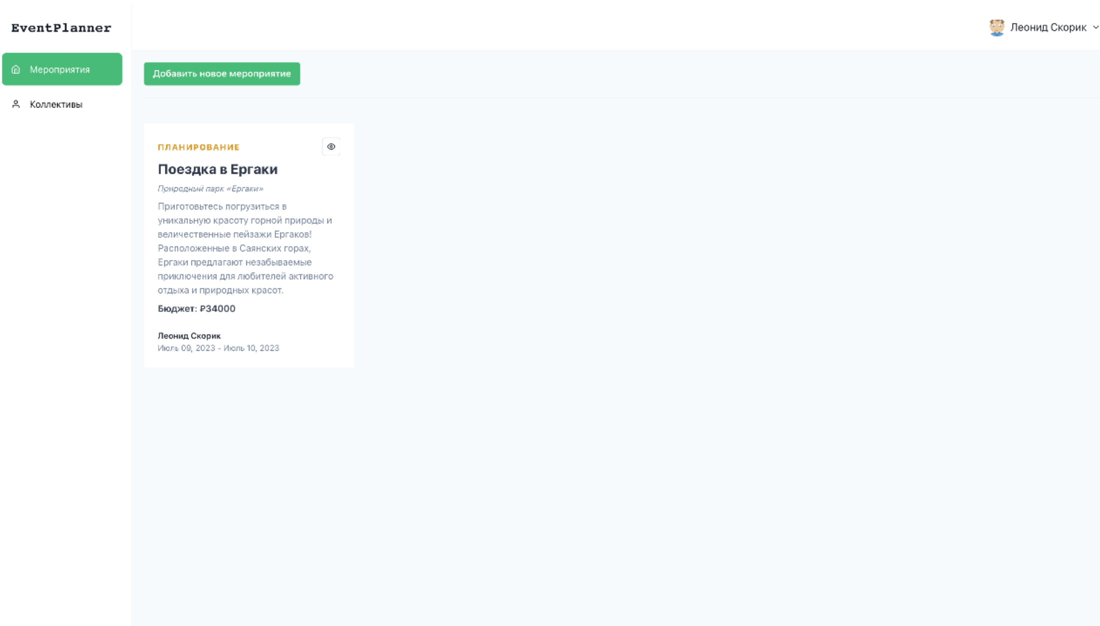


Рисунок 25 – Экран «Запланированные мероприятия»

На рисунке 26 показана форма для добавления нового мероприятия.

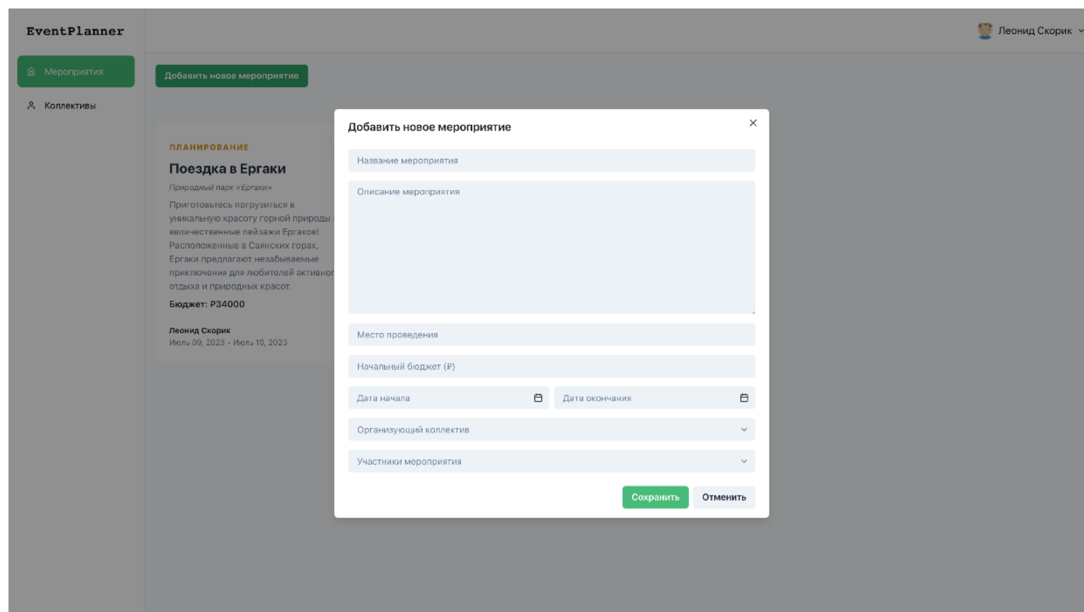


Рисунок 26 – Форма «Добавление мероприятия»

На рисунке 27 показан экран с полным описанием мероприятия.

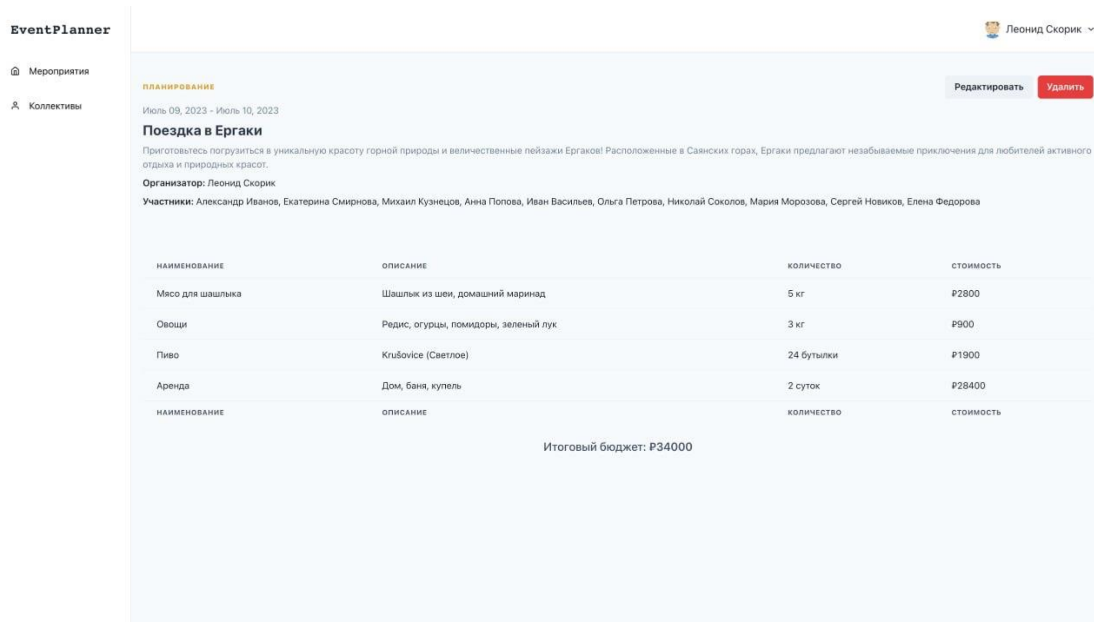


Рисунок 27 – Экран «Мероприятие»

На рисунке 28 показана форма для добавления предметов к мероприятию.

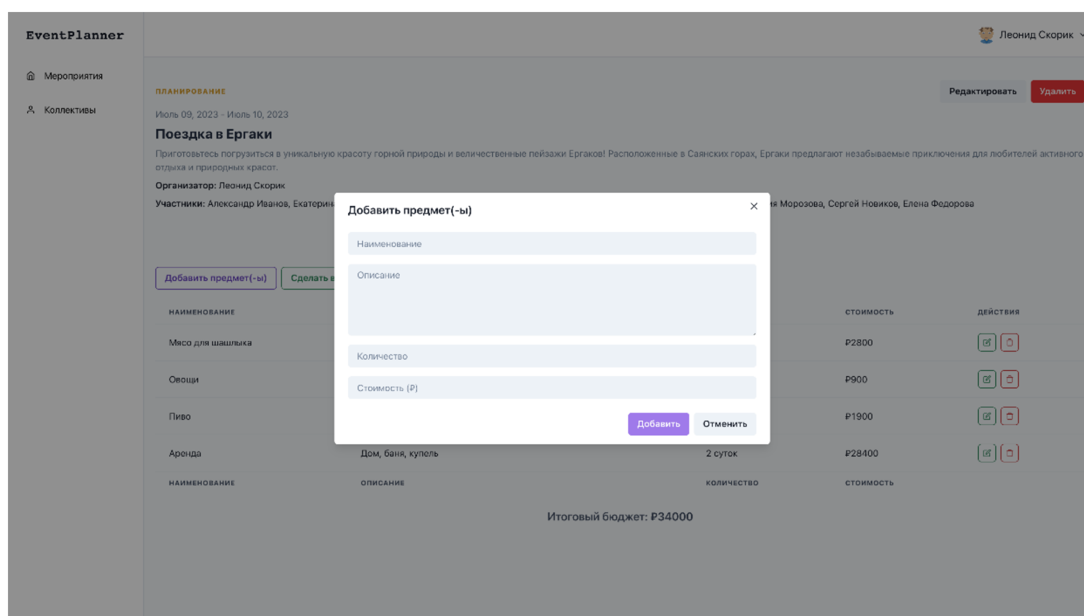


Рисунок 28 – Форма добавления предметов

На рисунке 29 показана форма, для того чтобы внести деньги в бюджет мероприятия.

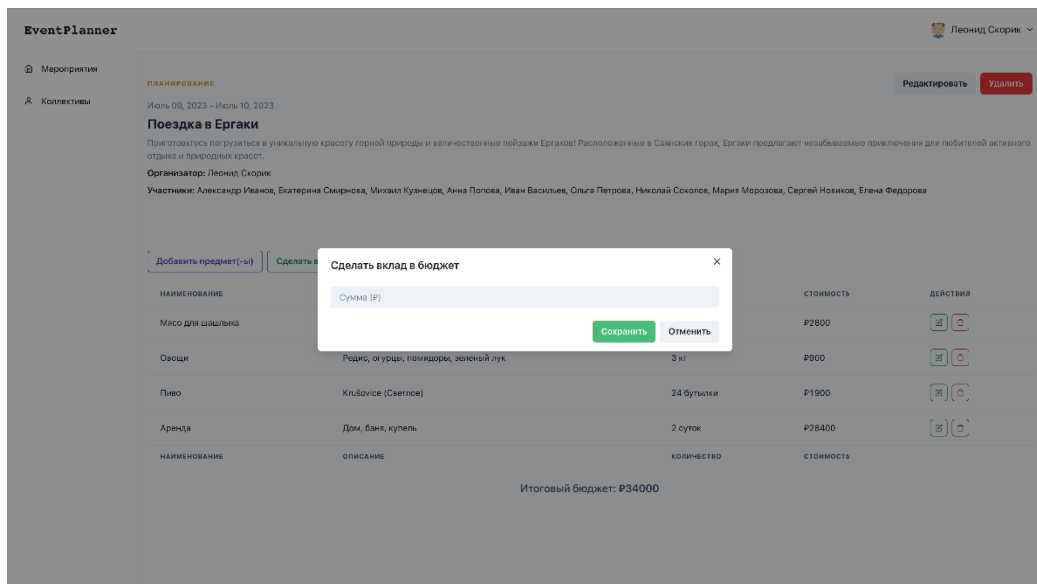


Рисунок 29 – Форма «Вклад в бюджет»

На рисунке 30 показан экран с коллективами, к которым пользователь может прикрепиться, кликнув на соответствующую кнопку.

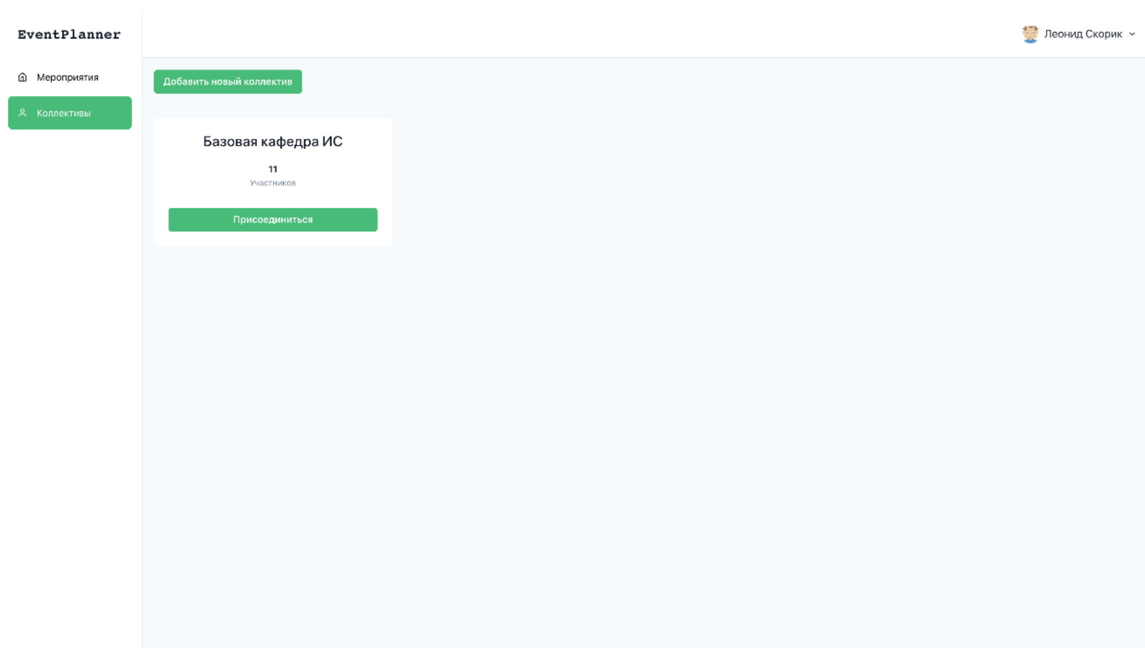


Рисунок 30 – Экран «Коллективы»

На рисунке 31 показана форма для добавления нового коллектива.

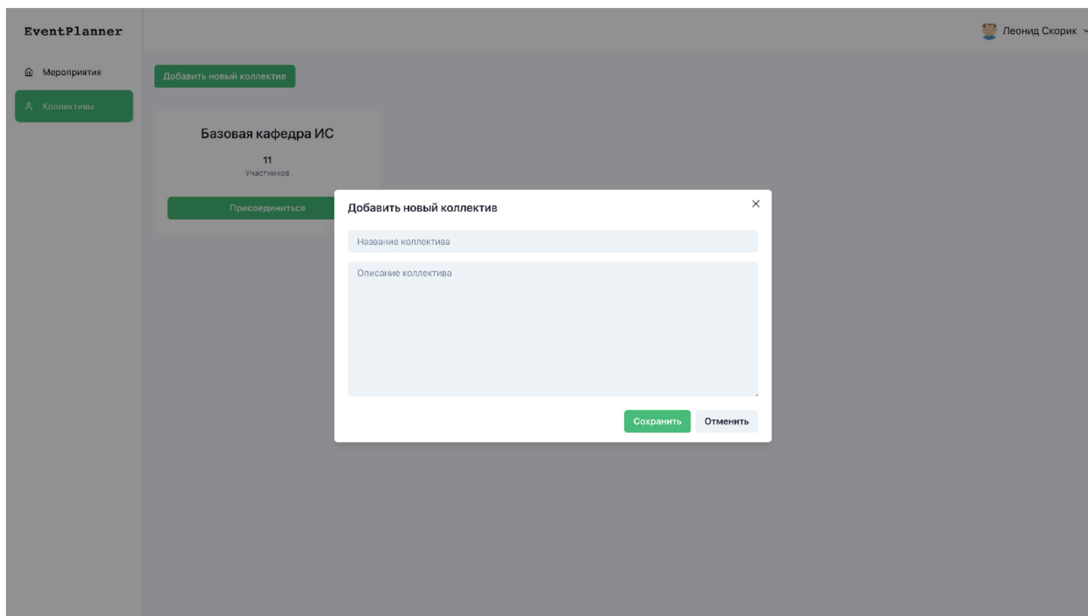


Рисунок 31 – Форма «Добавление коллектива»

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы по теме «Приложение для учета взносов и расходов на проведение коллективного мероприятия» была достигнута цель и решены поставленные задачи.

Проведен анализ предметной области, в котором рассмотрены аналогичные приложения. Выявлены их достоинства и недостатки. Также выявлены требования к системе и сформулированы функциональные возможности.

Следующим этапом проведено проектирование. Создана функциональная модель IDEF0, диаграмма вариантов использования и карта интерфейса.

Также был выполнен анализ различных средств разработки баз данных, frontend, backend. Рассмотрены достоинства и недостатки, на основе которых определены наиболее подходящие средства разработки.

После этого была создана база данных, написана клиентская и серверная часть приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Splitwise / WECHOOSE : сайт. – URL: https://wechoose.pro/service/splitwise (дата обращения: 20.03.2023)

2 Toshl Finance / TOSHL : сайт. – URL: / https://toshl.com/ru/ (дата обращения: 20.03.2023)

3 Tricount / TRICOUNT : сайт. – URL: / https://www.tricount.com/ru/ (дата обращения: 20.03.2023)

4 IDEF0. Знакомство с нотацией и пример использования / TRINION : сайт. – URL: https://trinion.org/blog/idef0-znakomstvo-s-notaciey-i-primer-ispolzovaniya (дата обращения: 05.05.2023)

5 Использование диаграммы вариантов использования UML при проектировании программного обеспечения / HABR : сайт. – URL: https://habr.com/ru/articles/566218/ (дата обращения: 05.05.2023)

6 Архитектура «Клиент-Сервер» / ITELON : сайт. – URL: https://itelon.ru/blog/arkhitektura-klient-server/ (дата обращения: 05.05.2023)

7 В чем особенности MongoDB и когда эта база данных вам подходит / MAIL : сайт. – URL: https://mcs.mail.ru/blog/osobennosti-mongodb-kogda-baza-dannyh-vam-podhodit (дата обращения: 05.05.2023)

8 MySQL: что это за сервер базы данных, пример / SKILLFACTORY : сайт. – URL: https://blog.skillfactory.ru/glossary/mysql/ (дата обращения: 05.05.2023)

9 Руководство по Node.js для новичков / SKILLBOX : сайт. – URL: https://skillbox.ru/media/code/chem_khorosh_node_js/ (дата обращения: 05.05.2023)

10 NestJS против Express.js / LOGROCKET : сайт. – URL: https://blog.logrocket.com/nestjs-vs-express-js/ (дата обращения: 05.05.2023)

11 Чистая архитектура / HABR : сайт. – URL: https://habr.com/ru/articles/269589/ (дата обращения: 05.05.2023)

12 Методы GET и POST. Использование и отличия / GURUWEBA : сайт. – URL: https://guruweba.com/html/metody-get-i-post-ispolzovanie-i-otlichiya/ (дата обращения: 05.05.2023)

13 Dependency injection / HABR GURUWEBA : сайт. – URL: https://habr.com/ru/articles/350068/ (дата обращения: 05.05.2023)

ПРИЛОЖЕНИЕ А

Код моделей и контроллеров сущностей

Код модели сущности мероприятие:

```
import {
    CreateTimeField,
    DateTimeField, EnumField, IntegerField,
    PrimaryKeyField,
    RelationField,
    RelationIdField,
    StringField,
    UpdateTimeField,
} from '@steroidsjs/nest/src/infrastructure/decorators/fields';
import {TeacherModel} from '../../../teacher/domain/models/TeacherModel';
import {GroupModel} from '../../../group/domain/models/GroupModel';
import {EventItemModel} from './EventItemModel';
import {EventStatusEnum} from '../enums/EventStatusEnum';

export class EventModel {
    @PrimaryKeyField()
    id: number;

    @StringField()
    name: string;

    @StringField()
    description: string;

    @StringField()
    location: string;

    @EnumField({
        enum: EventStatusEnum,
        defaultValue: EventStatusEnum.PLANNED,
    })
    status: string;

    @RelationIdField({
        relationName: 'teachers',
        isArray: true,
    })
    teachersIds: number[];

    @RelationField({
        type: 'ManyToMany',
        isOwningSide: true,
        relationClass: () => TeacherModel,
        isArray: true,
    })
```

```
    teachers: TeacherModel[];

    @RelationIdField({
        relationName: 'groups',
        isArray: true,
    })
    groupsIds: number[];

    @RelationField({
        type: 'ManyToMany',
        isOwningSide: true,
        relationClass: () => GroupModel,
        isArray: true,
    })
    groups: GroupModel[];

    @RelationIdField({
        relationName: 'items',
        isArray: true,
    })
    itemsIds: number[];

    @RelationField({
        type: 'OneToMany',
        relationClass: () => EventItemModel,
        isArray: true,
        inverseSide: (item: EventItemModel) => item.event,
    })
    items: EventItemModel[];

    @DateTimeField({
        skipSeconds: true,
    })
    startDate: string;

    @DateTimeField({
        skipSeconds: true,
    })
    endDate: string;

    @IntegerField()
    budget: number;

    @RelationIdField({
        relationName: 'organizer',
    })
    organizerId: number;

    @RelationField({
        type: 'ManyToOne',
```

```
        relationClass: () => TeacherModel,
    })
    organizer: TeacherModel;

    @CreateTimeField()
    createTime: string;

    @UpdateTimeField()
    updateTime: string;
}
```

Код модели сущности преподаватель:

```
import {
    CreateTimeField,
    PrimaryKeyField,
    RelationField,
    RelationIdField,
    StringField,
} from '@steroidsjs/nest/src/infrastructure/decorators/fields';
import {GroupModel} from '../../../group/domain/models/GroupModel';
import {EventModel} from '../../../event/domain/models/EventModel';

export class TeacherModel {
    @PrimaryKeyField()
    id: number;

    @StringField()
    name: string;

    @StringField()
    phone: string;

    @RelationIdField({
        relationName: 'groups',
        isArray: true,
    })
    groupsIds: number[];

    @RelationField({
        type: 'ManyToMany',
        relationClass: () => GroupModel,
        isOwningSide: false,
        isArray: true,
    })
    groups: GroupModel[];

    @RelationIdField({
        relationName: 'events',
        isArray: true,
```

```
    })
    eventsIds: number[];

    @RelationField({
        type: 'ManyToMany',
        relationClass: () => EventModel,
        isOwningSide: false,
        isArray: true,
    })
    events: EventModel[];

    @RelationIdField({
        relationName: 'organizationEvents',
        isArray: true,
    })
    organizationEventsIds: number[];

    @RelationField({
        type: 'OneToMany',
        relationClass: () => EventModel,
        inverseSide: (event: EventModel) => event.organizer,
        isArray: true,
    })
    organizationEvents: EventModel[];

    @CreateTimeField()
    createDate: string;
}
```

Код модели сущности группа:

```
import {
    CreateTimeField,
    PrimaryKeyField,
    RelationField,
    RelationIdField,
    StringField, UpdateTimeField,
} from '@steroidsjs/nest/src/infrastructure/decorators/fields';
import {TeacherModel} from '../../../teacher/domain/models/TeacherModel';
import {EventModel} from '../../../event/domain/models/EventModel';
import {EventItemModel} from '../../../event/domain/models/EventItemModel';

export class GroupModel {
    @PrimaryKeyField()
    id: number;

    @StringField()
    name: string;

    @RelationIdField({
```

```
        relationName: 'teachers',
        isArray: true,
    })
    teachersIds: number[];

    @RelationField({
        type: 'ManyToMany',
        relationClass: () => TeacherModel,
        isOwningSide: true,
    })
    teachers: TeacherModel[];

    @RelationIdField({
        relationName: 'events',
        isArray: true,
    })
    eventsIds: number[];

    @RelationField({
        type: 'ManyToMany',
        relationClass: () => EventModel,
        isOwningSide: false,
        isArray: true,
    })
    events: EventModel[];

    @RelationIdField({
        relationName: 'items',
        isArray: true,
    })
    itemsIds: number[];

    @RelationField({
        type: 'OneToMany',
        relationClass: () => EventItemModel,
        isArray: true,
        inverseSide: (item: EventItemModel) => item.event,
    })
    items: EventItemModel[];

    @CreateTimeField()
    createDate: string;

    @UpdateTimeField()
    updateTime: string;
}
```

Код контроллера сущности мероприятие:

```
import {Body, Controller, Delete, Get, Param, Post, Query} from '@nestjs/common';
```

```
import {ApiBody, ApiOkResponse, ApiQuery, ApiTags} from '@nestjs/swagger';
import {EventService} from '../../domain/services/EventService';
import {EventSearchDto} from '../../domain/dtos/EventSearchDto';
import {EventSchema} from '../schemas/EventSchema';
import {EventSaveDto} from '../../domain/dtos/EventSaveDto';

@Controller('/event')
@ApiTags()
export class EventController {
    constructor(
        private service: EventService,
    ) {
    }

    @Get()
    @ApiQuery({type: EventSearchDto})
    @ApiOkResponse({type: EventSchema, isArray: true})
    async get(@Query() dto: EventSearchDto) {
        return this.service.search(dto);
    }

    @Get('/:id')
    @ApiOkResponse({type: EventSchema, isArray: false})
    async getOne(@Param('id') id: number) {
        return this.service.findById(id);
    }

    @Post()
    @ApiBody({type: EventSaveDto})
    @ApiOkResponse({type: EventSchema, isArray: false})
    async create(
        @Body() dto: EventSaveDto,
    ) {
        return this.service.create(dto);
    }

    @Post('/:id')
    @ApiBody({type: EventSaveDto})
    @ApiOkResponse({type: EventSchema, isArray: false})
    async update(
        @Body() dto: EventSaveDto,
        @Param('id') id: number,
    ) {
        return this.service.update(id, dto);
    }

    @Delete('/:id')
    async delete(@Param('id') id: number) {
        return this.service.remove(id);
    }
```

```
}
```

Код контроллера сущности преподаватель:

```
import {Body, Controller, Delete, Get, Param, Post, Query} from '@nestjs/common';
import {ApiBody, ApiOkResponse, ApiQuery} from '@nestjs/swagger';
import {TeacherService} from '../../domain/services/TeacherService';
import {TeacherSearchDto} from '../../domain/dtos/TeacherSearchDto';
import {TeacherSaveDto} from '../../domain/dtos/TeacherSaveDto';
import {TeacherSchema} from '../schemas/TeacherSchema';

@Controller('/teacher')
export class TeacherController {
    constructor(
        private service: TeacherService,
    ) {
    }

    @Get()
    @ApiQuery({type: TeacherSearchDto})
    @ApiOkResponse({type: TeacherSchema, isArray: true})
    async search(
        @Query() dto: TeacherSearchDto,
    ) {
        return this.service.search(dto);
    }

    @Get('/:id')
    @ApiOkResponse({type: TeacherSchema, isArray: false})
    async getOne(@Param('id') id: number) {
        return this.service.findById(id);
    }

    @Post()
    @ApiBody({type: TeacherSaveDto})
    @ApiOkResponse({type: TeacherSchema, isArray: false})
    async create(
        @Body() dto: TeacherSaveDto,
    ) {
        return this.service.create(dto);
    }

    @Post('/:id')
    @ApiBody({type: TeacherSaveDto})
    @ApiOkResponse({type: TeacherSchema, isArray: false})
    async update(
        @Body() dto: TeacherSaveDto,
        @Param('id') id: number,
    ) {
        return this.service.update(id, dto);
```

```
    }

    @Delete('/:id')
    async delete(@Param('id') id: number) {
        return this.service.remove(id);
    }
}
```

Код контроллера сущности группа:

```
import {Body, Controller, Delete, Get, Param, Post, Query} from '@nestjs/common';
import {ApiBody, ApiOkResponse, ApiQuery, ApiTags} from '@nestjs/swagger';
import {GroupService} from '../../domain/services/GroupService';
import {GroupSearchDto} from '../../domain/dtos/GroupSearchDto';
import {GroupSchema} from '../schemas/GroupSchema';
import {GroupSaveDto} from '../../domain/dtos/GroupSaveDto';

@Controller('/group')
@ApiTags('Группы')
export class GroupController {
    constructor(
        private service: GroupService,
    ) {
    }

    @Get()
    @ApiQuery({type: GroupSearchDto})
    @ApiOkResponse({type: GroupSchema, isArray: true})
    async search(
        @Query() dto: GroupSearchDto,
    ) {
        return this.service.search(dto);
    }

    @Get('/:id')
    @ApiOkResponse({type: GroupSchema, isArray: false})
    async getOne(@Param('id') id: number) {
        return this.service.findById(id);
    }

    @Post()
    @ApiBody({type: GroupSaveDto})
    @ApiOkResponse({type: GroupSchema, isArray: false})
    async create(
        @Body() dto: GroupSaveDto,
    ) {
        return this.service.create(dto);
    }

    @Post('/:id')
```

```
    @ApiBody({type: GroupSaveDto})
    @ApiOkResponse({type: GroupSchema, isArray: false})
    async update(
        @Body() dto: GroupSaveDto,
        @Param('id') id: number,
    ) {
        return this.service.update(id, dto);
    }

    @Delete('/:id')
    async delete(@Param('id') id: number) {
        return this.service.remove(id);
    }
}
```

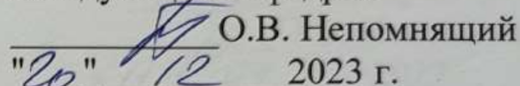
Министерство науки и высшего образования РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
________ О.В. Непомнящий
"20" ___12___ 2023 г.

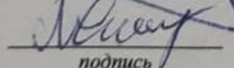


БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Приложение для учета взносов и расходов на проведение коллективного мероприятия

Руководитель ____________ 10.06.2023 ст. преподаватель О.В. Шмелёв
подпись *дата* *должность, ученая степень*



Выпускник ____________ 20.06.2023 Л.Л. Скорик
подпись *дата*

Нормоконтролёр ____________ 20.06.2023 ст. преподаватель О.В. Шмелёв
подпись *дата* *должность, ученая степень*

Красноярск 2023