

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

Факультет
«Вычислительная техника»

(наименование)

Кафедра
«Системы автоматизированного
проектирования»

(наименование)

правление подготовки

02.03.03 Математическое обеспечение и АИС

(код и наименование)

офиль

Администрирование информационных систем

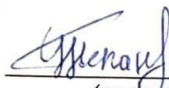
(наименование)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему:

Разработка серверной части приложения для проведения оффлайн
мероприятий

удент



(подпись, дата)

Талабаев С.В.

(фамилия, инициалы)

ководитель



(подпись, дата)

Финогеев А.А.

(фамилия, инициалы)

рмоконтролёр

проф. каф. САПР

(должность, место работы)



(подпись, дата)

Бождай А.С.

(фамилия, инициалы)

бота допущена к защите

ведующий
федрой



(подпись)

Бурукина И.П.

(фамилия, инициалы)

бота защищена с отметкой отн (протокол заседания ГЭК от 18.06.24 № 9)

кретарь ГЭК



(подпись)

Финогеев А.А.

(фамилия, инициалы)

“Утверждаю”

/И.П.Бурукина/

Заведующий кафедрой САПР

“ 10 ” “ 06 ” 2024 г.

ЗАДАНИЕ

на выполнение бакалаврской работы

Студент гр. 20BA1 факультета ВТ специальности Математическое обеспечение и администрирование информационных систем

Талабаев Степан Васильевич

(фамилия, имя, отчество)

Руководитель проекта к.т.н., доцент кафедры САПР Финогеев Антон Алексеевич

Время дипломного проектирования с _____ 2024 г. по _____ 2024 г.

Место преддипломной практики кафедра САПР ФВТ ПГУ

Тема бакалаврской работы: «Разработка серверной части приложения для оффлайн мероприятий» Тема утверждена приказом ПГУ № _____ от “ _____ ” _____ г.

Техническое задание на бакалаврскую работу (назначение устройства, условия применения, внешние взаимодействия, специальные требования т.п.)

6.1. Назначение разработки: Разрабатываемый программный комплекс предназначен для проведения оффлайн мероприятий.

6.2. Требования к функциональности системы: Компоненты системы позволяют: Зарегистрироваться и авторизоваться в системе как организатор или участник. Работу в нескольких режимах – организатор и участник. Участник должен иметь возможность отмечаться в лекциях, получать, тратить макскоины, участвовать в событиях, проводимых на конференции, смотреть расписание лекций и их местоположение, оставлять заявку на консультацию, видеть список партнеров, видеть галерею мероприятия прошлого и текущего года, видеть события, которые присылают партнеры или организаторы, видеть текущий счет прошедших событий, участия в розыгрышах и счет личных макскоинов. Организатор должен иметь такие же возможности что и участник, но при этом еще: начислять макскоины, создавать и отправлять события участникам.

6.3. Требования к ПО: Язык TypeScript, платформа Node js, фреймворк Nest js, , база данных PostgreSQL, Dbeaver 22.4 и выше.

6.4. Требования к аппаратному обеспечению: Персональный компьютер с операционной системой Ubuntu 20.04

Объем и содержание основной части проекта

7.1. Содержание пояснительной записки бакалаврской работы: перечень подлежащих разработке, расчетов, обоснований, описаний:

7.1.1. Анализ предметной области.

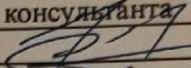
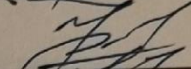
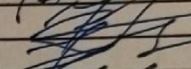
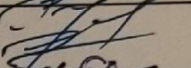
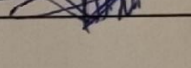
7.1.2. Проектирование структуры системы.

7.1.3. Разработка программных модулей системы.

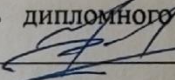
7.1.3. Разработка программных модулей системы.

7.1.4. Тестирование информационной систем

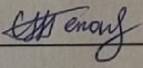
Календарный график работ по выполнению проекта.

Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя, консультанта
1. Анализ ТЗ	5 %	7.2-21.2	
2. Проектирование структуры системы	10 %	25.3-10.4	
3. Разработка ИО	15 %	10.4-20.4	
4. Разработка ПО	30%	20.4-10.5	
5. Тестирование и опытная эксплуатация программы	30%	10.5-15.5	
6. Оформление ПЗ	10 %	15.5-30.5	
7. Нормоконтроль		31.05	

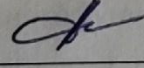
Дата выдачи задания “ ” 2024 г.

Руководитель дипломного проекта  к.т.н., доцент кафедры САПР Финогеев Антон Алексеевич / А.А. Финогеев

Задание к исполнению принял “ ” 2024 г.

Студент Талабаев Степан Васильевич  /

Бакалаврскую работу к защите допустить

Декан ФВТ  / Л.Р. Фионова

10.06.2024

Реферат

Пояснительная записка содержит 62 страницы, 42 рисунка, 2 таблицы, 10 использованных источников и 1 приложение.

ВЕБ-СЕРВЕР, ОРГАНИЗАЦИЯ МЕРОПРИЯТИЙ, КОНФЕРЕНЦИИ, ОБЪЕКТНО-РЕЛЯЦИОННАЯ БАЗА ДАННЫХ, FIREBASE, NESTJS, NODE.JS, POSTGRESQL.

Объект разработки – серверная часть для проведения оффлайн мероприятий.

Цель разработки: Повысить эффективность проведения оффлайн мероприятий
за счет разработки многофункционального сервера.

Для достижения заданной цели были решены следующие задачи:

1. Анализ предметной области.
2. Выбор средств разработки.
3. Формализация предметной области.
4. Проектирование системы.
5. Проектирование базы данных.
6. Проектирование программных модулей.
7. Тестирование.

Результат разработки: многофункциональный и производительный веб-сервер.

Средства разработки: Webstorm 2019, Dbeaver 22.3.4, Ubuntu 22.04, Node js, escript.

Требования к ПО: ОС - Ubuntu 22.04, CPU - 2 * 3.3 ГГц, RAM - 2 Гб, NVMe 1 Гб,

				ВКР.020303.12 24 84 01			
Лист	№ докум.	Подпись	Дата	Разработка серверной части приложения для проведения оффлайн мероприятий Пояснительная записка	Лит.	Лист	Листов
	Талабаев С.В.						
	Финогеев А.А.					4	
	Бождай А.С.		20.06.24		гр. 20BA1		
	Бурукина И.П.		12.06.24				

Содержание

Обозначения и сокращения	6
Введение.....	7
1. Анализ предметной области	8
1.1 Обзор аналогов разрабатываемой системы	8
1.2 Методы и средства разработки.....	17
2. Проектирование структуры системы	19
2.1 Разработка структуры системы	19
2.2 Алгоритм работы системы.....	25
3. Разработка программного обеспечения	28
3.1 Инфологическое проектирование	28
3.2 Связи между таблицами.....	31
3.3 Перечень SQL-запросов	32
4. Тестирование отдельных сценариев работы приложения	38
4.1 Тестирование с помощью Swagger	38
Заключение	52
Список литературы:	53
Приложение А. Фрагмента исходного кода	54

Обозначения и сокращения

БД — база данных.

Api — программный интерфейс.

Firebase – внешний сервис позволяющий расширять функционал приложения.

Firebase Cloud Messaging (FCM) – механизм отправки push уведомлений.

СУБД — система управления базами данных.

UML (англ. Unified Modeling Language) – унифицированный язык моделирования.

JavaScript – мультипарадигмальный динамический интерпретируемый язык программирования.

Typescript – более расширенная версия языка JavaScript.

Node.js – программная платформа для разработки серверных приложений на основе языка JavaScript.

SQL (англ. Structured Query Language) – это декларативный язык программирования.

CRUD – базовые операция в СУБД (Создание, Чтение, Обновление, Удаление)

Frontend – пользовательский интерфейс и его функции, направленные на взаимодействие с пользователем.

Коин-модель – модель пополнения либо списывание макскоинов.

Backend – серверное приложение, предназначенное для хранения и выполнения доменной логики приложения.

DTO (data transport object) – объект который переходит от одного функционального сегмента к другому.

Many to many – связь между таблицами многим ко многим.

Макскоины – условная цифровая валюта, выдаваемая за активность на мероприятии.

Введение

В современном мире, где технологии пронизывают все аспекты жизни, неудивительно, что и сфера организации мероприятий не осталась в стороне, ведь эта задача становится все сложнее. Нужно учитывать множество факторов: от поиска подходящего места проведения, до регистрации участников и рассылки напоминаний. Приложение для проведения оффлайн-мероприятий может значительно упростить этот процесс, автоматизируя многие задачи и предоставляя организаторам и участникам удобные инструменты для взаимодействия.

При организации мероприятий одним из важных моментов является хранение данных, а это в свою очередь непосредственно связано с их поиском, подсчетом и оформлением. Информация хранится в бумажном виде из-за чего процесс работы с ней бывает неудобен, а порой даже затруднителен. Веб-приложение решает данный вопрос.

В данной выпускной квалификационной работе представлена разработка серверной части приложения для проведения оффлайн-мероприятий. Рассмотрены функциональные возможности приложения, архитектура серверной части, выбор технологий и реализация основных модулей.

Веб-приложение ориентировано на помощь организаторам в создании и проведении мероприятий. Кроме того, приложение позволяет любому пользователю получать уведомления об изменении или каком-либо происшествии на мероприятии. Кроме того, все данные после проведения удобно выгружать для дальнейшего анализа.

1. Анализ предметной области

1.1 Обзор аналогов разрабатываемой системы

В текущее время в интернете можно найти различные аналоги, которые предоставляют мобильное приложение для организации мероприятий. Исследовав открытые источники найдены похожие решения разрабатываемой системы. Часть из платных сервисов имеет демонстрационную версию на своём сайте. Некоторые сервисы предлагают реализовать нужный пользователю функционал с самого начала, а не которые иначе предоставляют набросок приложения, под который нужно адаптировать своё мероприятие. Подробно рассмотрим примеры таких систем:

1.Eventrocks[1] – это конструктор приложений и web-платформа. Опрашивая клиента о контексте и сценария проведения мероприятия, система создает приложение отвечающие на все необходимые запросы. На рисунке 1 представлен первый экран сайта, данного продукта.

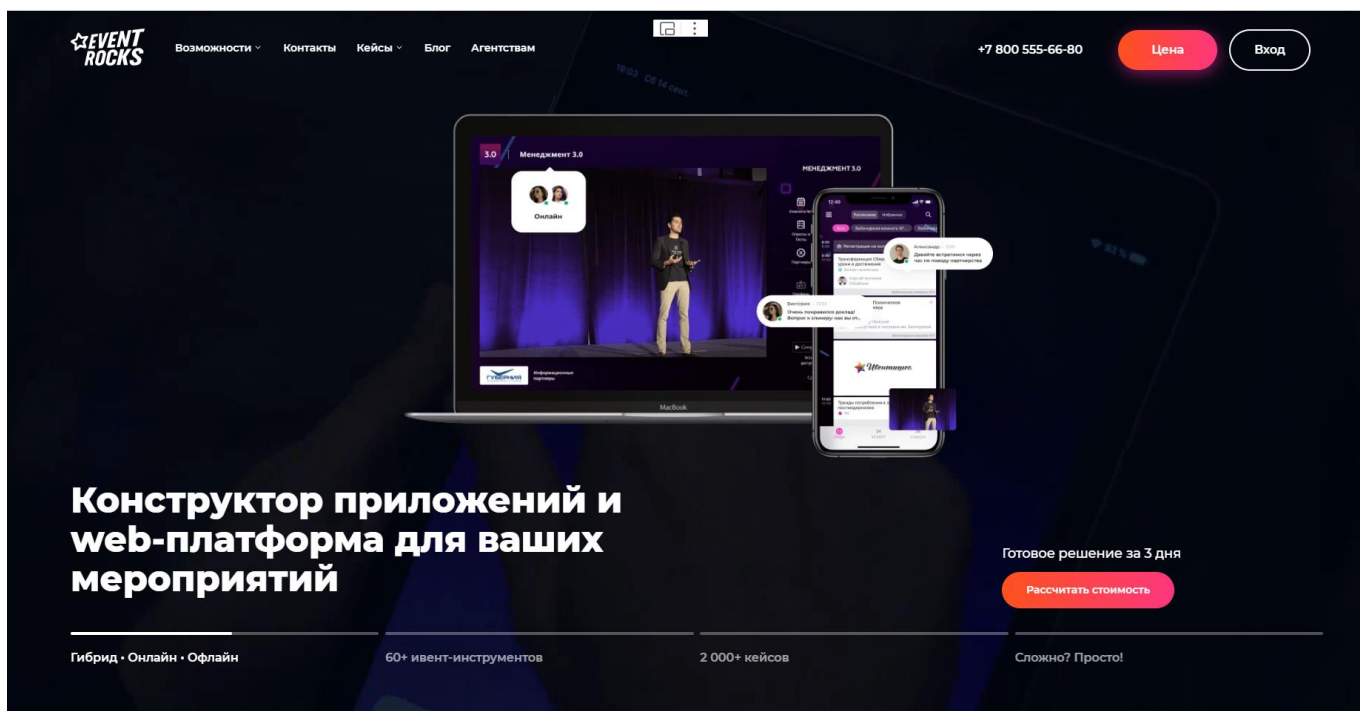


Рисунок 1 – Первый экран сайта eventrocks.ru.

На сайте можно ознакомиться с отзывами предыдущих клиентов и другой полезной информацией.

Изучав аналог, выявил факт, что создание приложение происходит конструктором и в краткие сроки, в связи с этим можно говорить нескольких минусах:

Конструкторы подходят для приложений с простым набором функций, если в процессе разработки заказчик все больше включает необходимых запросов это наверняка скажется в его гибкости, а также производительности приложения. Ведь вся логика приложения будет лежать не в руках разработчика, а уже в заготовленном функциональном блоке конструктора. Говоря в этом случае, возникает вопрос о масштабируемости подобного решения, и вообще возможно ли оно.

Также есть подозрение о факте данного сервиса, что приложение будет готово через три дня. Если представить, что заказчику необходима интеграция с уже существующим сервисом, только получение ключей доступа для пользования чужим API может уйти больше заявленного срока.

2. Ticketscloud[2] – это персональный цифровой помощник для организации мероприятий. Данный сервис берет на себя полную ответственность за создание, проведение и анализ результатов, проведенного события. На сайте можно ознакомиться с новостями и рекомендациями их прошлых клиентов. А также, что это не просто цифровой помощник, а площадка для коммуникации и успешного взаимодействия. На рисунках 2-3 представлен сайт первый экран сайта ticketscloud.com.

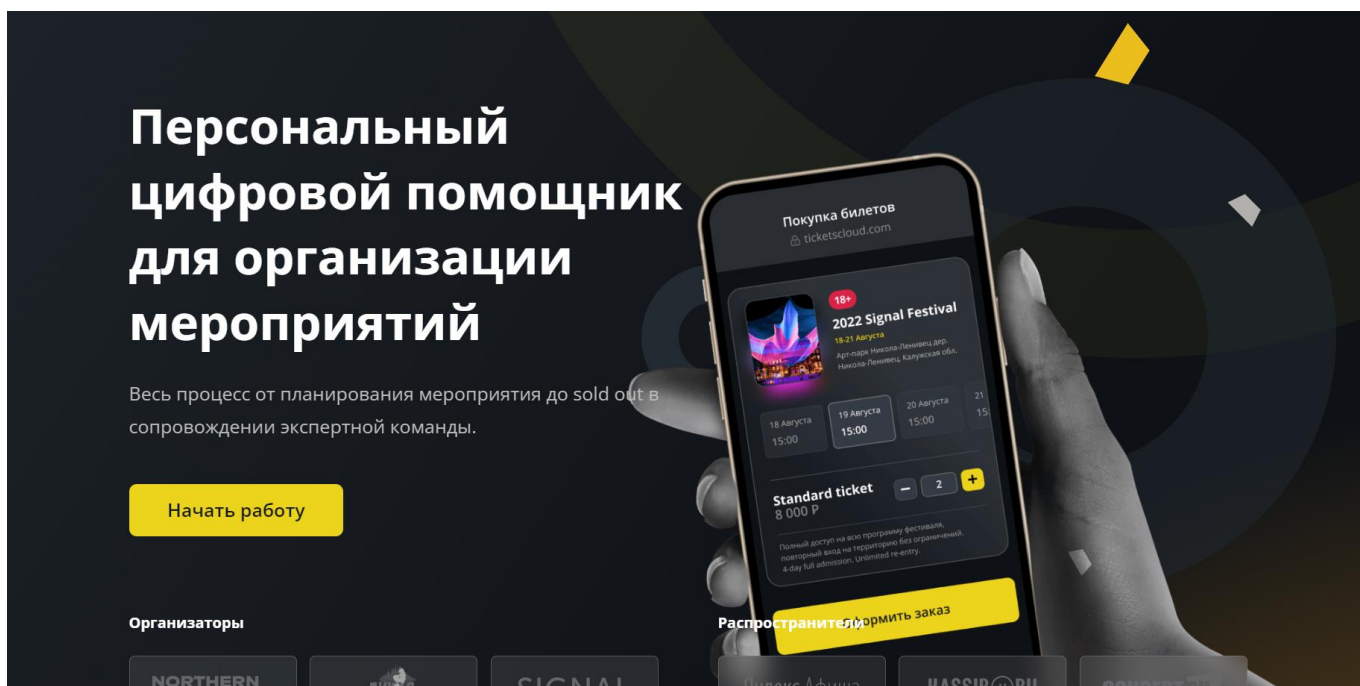


Рисунок 2 – Главная страница сайта ticketscloud.com.

Функционал схож с предыдущим примером, аналогично предоставляет помощь в организации мероприятий. Позиционирует себя как цифрового помощника как для организаторов, так и для участников. На рисунке 3 регистрация на сайте продукта (рисунок 3):



Рисунок 3 - Форма регистрации ticketscloud.com.

Основными плюсами данного аналога является сотрудничество с множеством внешних сервисов, что действительно покрывает большой список запросов от заказчика. Однозначным аргументом, почему выбирают именно этот сервис – это мощные инструменты для аналитики на базе искусственного интеллекта.

Однако, есть и минусы у данного продукта. В связи с большим функционалом еще не доработано оформление документации, она изображена на рисунке 4:

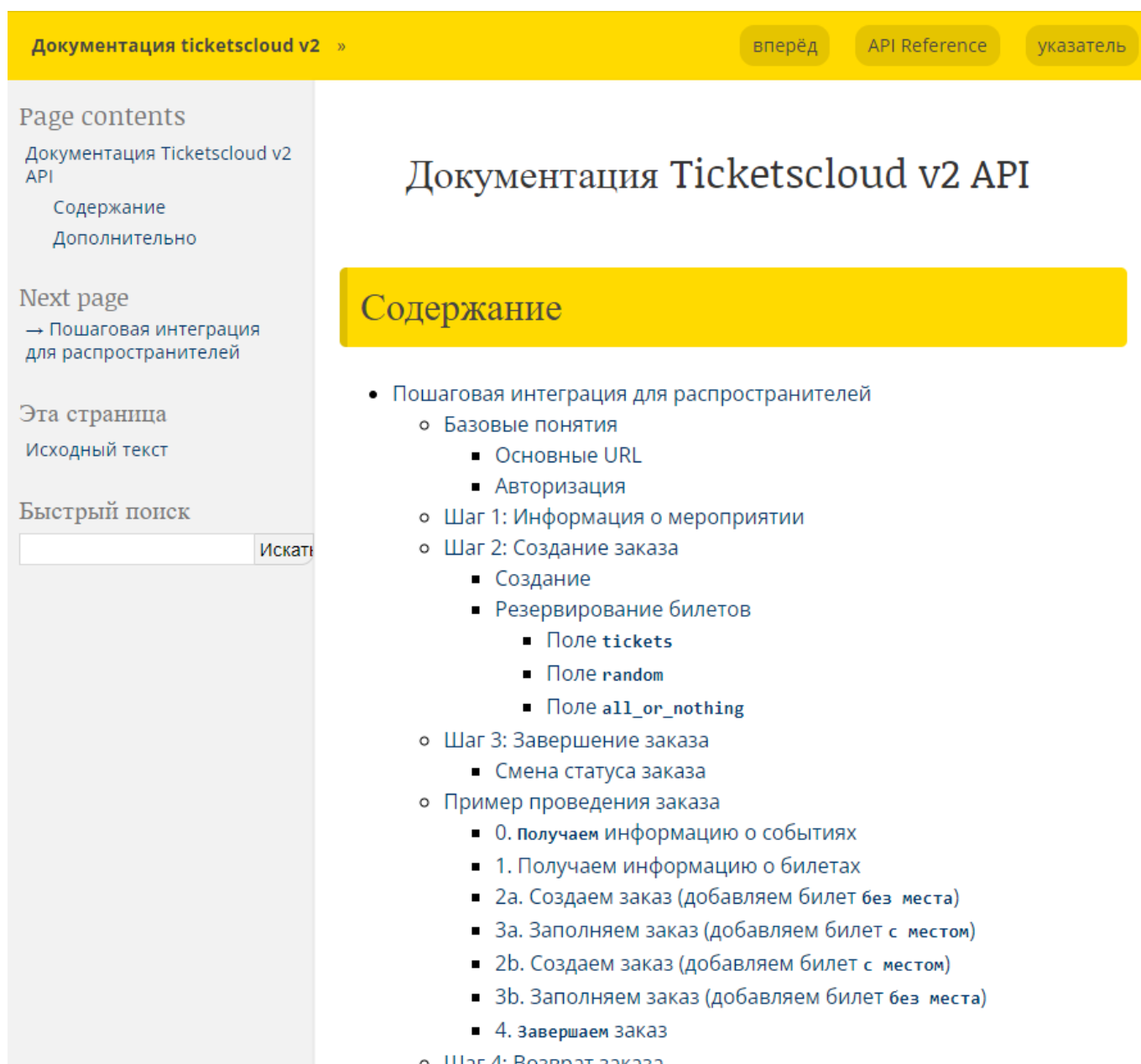


Рисунок 4 - Документация ticketscloud api.

Еще один минус - это отсутствие навигации. При проведении мероприятия, участник может потеряться в разных локациях.

3. uForum.pro[3] – мобильное приложение для мероприятий. Данный сервис предоставляет универсальное приложение с возможностью адаптировать под запросы заказчика, а также предоставляет административную панель для редактирования информации, чтобы участники были всегда в курсе изменений. Однако, каким именно способом осуществляется уведомление участников об обновлении информации - неизвестно. Хочется отметить, что это единственный

аналог, где на сайте указана точная стоимость приложения. На рисунке 5 представлен сайт uForum.pro.

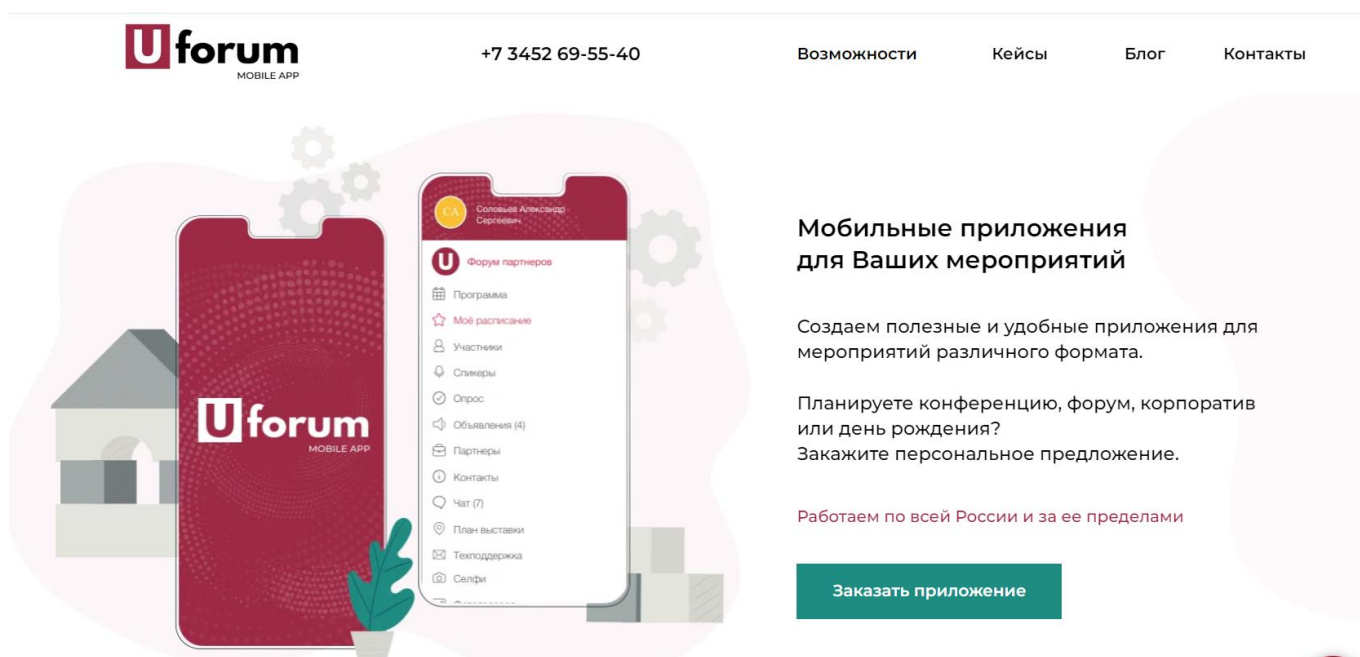


Рисунок 5 – Сайт uForum.pro.

Проанализировав продукт удалось выявить недостатки:

1. Основным минусом, выделяющим данную систему на фоне предыдущих, является минимальная адаптация под запросы заказчика.
2. Относительно конкурентов, пустой сайт данного приложения и не современный дизайн, что вызовет сомнения стоит ли использовать именно данный сервис.
3. Минус, который часто можно заметить и у других аналогов – это отсутствие навигации для участников мероприятий и личного кабинета.

В каждой вышеперечисленной системе есть свои преимущества и недостатки по сравнению с разработанной системой, для удобства восприятия и понимания это было представлено в таблице 1.

Таблица 1 – аналоги систем для проведения оффлайн мероприятий

Сравниваемые характеристики	Аналоги			Разработанное веб-приложение
	Eventrocks	Ticketscloud	uForum	
Возможность и анализа после завершения мероприятия	- Отсутствует функционал	+ Есть целые инструменты для анализа с искусственным интеллектом	+/- После проведения мероприятия составляется отчет	+ Есть выгрузка всех пользователей присутствовавших на конференции с полной информацией об их активности
Стоимость	+/- Стоимость продукта зависит от требований заказчика на момент формирования требований к приложению.	+/- Стоимость продукта зависит от требований заказчика на момент формирования требований к приложению.	+ Из-за того, что платформа позиционирует себя как универсальное приложение, точная сумма указана на сайте, она ниже конкурентов.	+ Стоимость приложения ниже рыночной.

Продолжение таблицы 1

Сравниваемые характеристики	Аналоги			Разработанное веб-приложение
	Eventrocks	Ticketscloud	uForum	
Удобство использования	+/- Интерфейс приложения современен и удобен, однако он содержит большое количество информации, что может затруднить восприятие	+/- Интерфейс приложения прост и удобен, однако он содержит большое количество информации, что может затруднить восприятие	+ Интерфейс крайне прост и удобен	+ Интерфейс приложения удобный, не содержит большой информации затрудняющий пользоваться приложением
Предлагаемые возможности.	+ Предлагается большой функционал, который может покрыть большую часть запросов	+/- Большой функционал приложения, но больше применим к формату онлайн	+ На главной странице представлен о крайне малый список возможностей приложения	+ Предлагается огромный функционал для проведения мероприятий.

Продолжение таблицы 1

Сравниваемые характеристики	Аналоги			Разработанное веб-приложение
	Eventrocks	Ticketscloud	uForum	
Навигация пользовател ей на мероприяти и	+/- Карта отсутствует, но для каждой локации указана аудитория ее проведения	- Отсутствует функционал	+/- Карта отсутствует, но для каждой локации указана аудитория ее проведения	+ Есть карта для локаций, а также расписание с номерами аудиторий
Личный кабинет	+ Есть личный кабинет, с котором можно отследить активность на мероприятии	+ Есть личный кабинет, но не возможности редактирования профиля	- Личный кабинет отсутствует	+ Есть личный кабинет, в котором можно посмотреть историю активности на мероприятии и возможность редактировать профиль

Продолжение таблицы 1

Сравниваемые характеристики	Аналоги			Разработанное веб-приложение
	Eventrocks	Ticketscloud	uForum	
Масштабируемость	<p>+</p> <p>Приложение создается за 3 дня в конструкторе, поэтому масштабируемость зависит от потребностей заказчика.</p>	<p>+/-</p> <p>Для каждого мероприятия почти заново создается приложение и в зависимости от потребностей оно будет масштабируемо или нет.</p>	<p>+</p> <p>Простой интерфейс и малый список возможностей не только удовлетворяет базовые потребности организаторов, но и позволяет легко масштабировать приложение.</p>	<p>+</p> <p>Большой спектр функций нескольких тем приложения позволяет не создавать приложение заново и только дополнять его нужно функционалом.</p>

Как видно из таблицы, рассмотренные аналоги приложений для проведения мероприятий как свои плюсы, так и недостатки. Обобщив недостатки получен следующий список:

- Отсутствие навигации пользователей.

- Сложность в расширении функционала, для добавление новых сервисов.
- Неудобный интерфейс (шрифт, затрудняющий чтение, переизбыток информации на одной странице, из-за чего сложно найти нужную информацию).
- Отсутствие возможности анализа активностей пользователей.
- Отсутствие личного кабинета или отсутствие возможности редактирования персональных данных.

1.2 Методы и средства разработки

Для разработки веб-приложения был выбран продукт JetBrains Webstorm [4]. В настоящий момент является довольно популярной средой разработки для языка Typescript. Данный продукт предоставляет широкий и полезный функционал для пользователей.

В качестве хранилища данных использовалась СУБД Postgresql [5]. Очень популярное среди разработчиков, в следствие нескольких причин:

1. Открытый исходный код: PostgreSQL является бесплатной и open-source СУБД, поэтому можно использовать, модифицировать и распространять без каких-либо ограничений.
2. СУБД известна своей надежностью и стабильностью. Она используется в многих отраслях, включая банковское дело, здравоохранение и государственное управление.
3. Масштабируемость: PostgreSQL способен масштабироваться, именно поэтому его используют и крупнейшие компании.
4. PostgreSQL является крайне производительной системой управления баз данных, которая позволяет обрабатывать сложные запросы с высокой скоростью, относительно ее аналогов.
5. Сообщество: не мало важным фактором является большое сообщество пользователей и разработчиков, поэтому в случае затруднения при использовании легко найти решение или получить быстрый ответ на форуме.

В качестве case средства для системы управления базой данных Postgresql использовался Dbeaver[6]. Популярный и простой инструмент, который давно уже

заслужил своё доверие среди пользователей, к тому же является мультиплатформенным.

Для разработки серверной части системы проведения оффлайн мероприятий использовалась в качестве ORM использовался Sequelize[7]. Несколько ключевых причин его использования:

1. Простота использования: управление базой данных с помощью объектно-ориентированного подхода, используя Typescript[8].
2. Продуктивность: автоматизация многих задач в частности, выполнения CRUD операций.
3. Типизация: ORM поддерживает TypeScript, что обеспечивает строгую типизацию кода, делая его более надежным.
4. Поддержка нескольких баз данных: Sequelize поддерживает не только PostgreSQL, а еще MySQL, SQLite, MariaDB, Microsoft SQL Server и Oracle.
5. Активное сообщество: ORM имеет активное и большое сообщество разработчиков и пользователей, благодаря которым можно отыскать ответ на тот или иной вопрос, возникший при разработке.

Сам веб-сервер был написан на Node js[9]. Технология зарекомендовала себя на рынке специалистов как мощная универсальная платформа для разработки. Скорость, масштабируемость, асинхронная и неблокирующая архитектура, универсальность, широкий спектр применений, большой набор модулей, простота изучения и использования – это причины, по которым многие разработчики выбирают Node js.

При разработке своей выпускной квалификационной работы использовался Firebase[10], как средство отправки push уведомлений. Выбор конкретно этой технологии был очевиден из-за простоты использования. У данного сервиса простой интерфейс и понятная документация. В качестве отправки используется внутренний механизм FCM, который не только позволит отправить уведомления на устройство в автономном режиме, но крайне легко масштабируется при большом трафике.

2. Проектирование структуры системы

2.1 Разработка структуры системы

На рисунке 6 представлена диаграмма прецедентов, которая отображает разнообразные варианты использования системы для проведения оффлайн мероприятий.

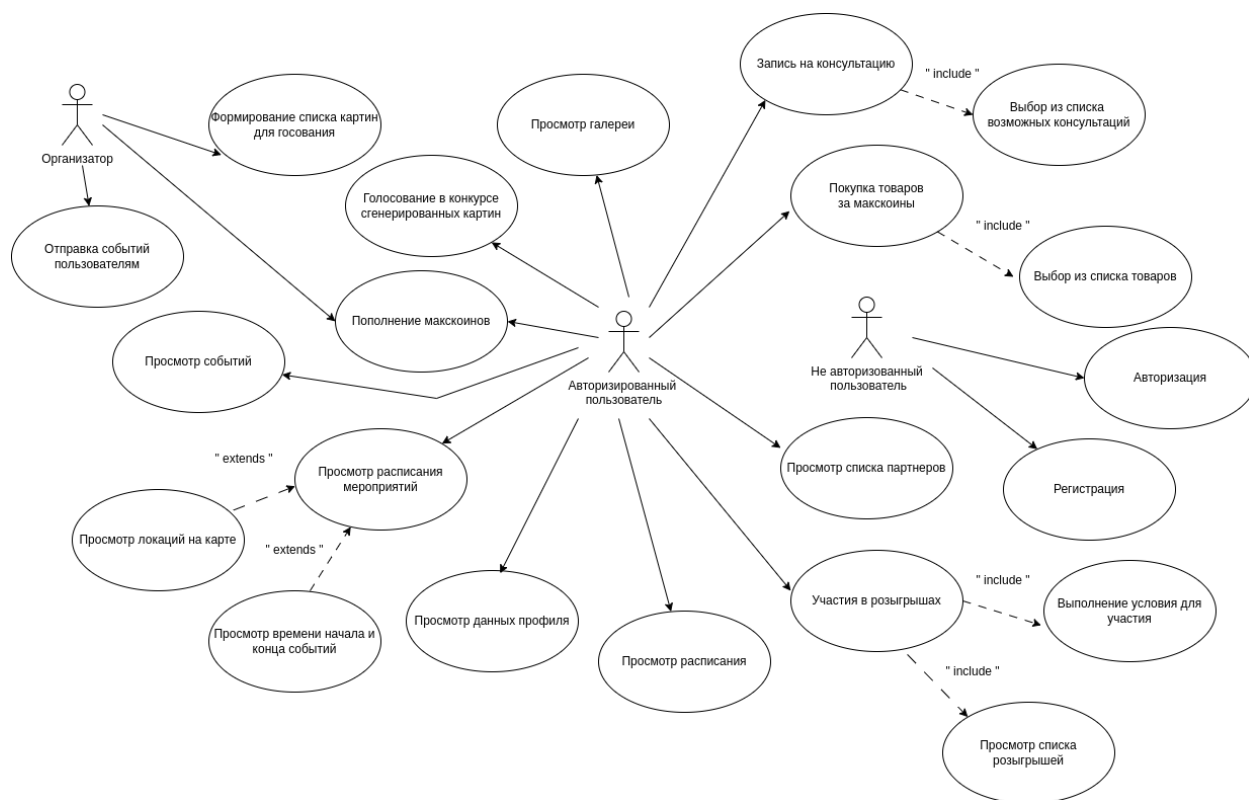


Рисунок 6 – Варианты использования системы.

На предоставленной диаграмме прецедентов есть три актора:

1. Неавторизованный пользователь.
2. Авторизованный пользователь.
3. Организатор.

С помощью диаграммы прецедентов была описана работа с точки зрения пользователя, что позволяет разработчикам ранее продумать систему изнутри.

Неавторизованному пользователю доступны следующие возможности:

1. Авторизоваться.
2. Зарегистрироваться.

Авторизованному пользователю доступны следующие возможности:

1. Просмотр галереи.

2. Покупка товаров за макскоины.
3. Просмотр списка партнеров.
4. Участие в розыгрышах.
5. Просмотр расписания.
6. Просмотр данных профиля.
7. Пополнение макскоинов.
8. Голосование в конкурсе сгенерированных картин.
9. Запись на консультацию.

Организатору доступны следующие возможности:

1. Отправка уведомлений пользователям.
2. Начисление макскоинов.
3. Формирование списка сгенерированных картин.

Пользователи данного ресурса:

1. Посетители мероприятия, заинтересованные в конференции или ее спонсорах.
2. Администраторы, заинтересованные анализе активностей посетителей, сборе статистики, заявок на консультации.

Модули:

1. Модуль аутентификации и авторизации.
 - 1.1. Регистрация (ввод таких данных как: имя, фамилия, отчество, номер телефона, почта, пароль).
 - 1.2. Авторизация (ввод данных: номер телефона и пароль).
2. Модуль посетителя конференции:
 - 2.1. Просмотр основной информации пользователя его персональной информации, количество заработанных макскоинов, участие в розыгрышах, просмотра расписания с локациями, список спонсоров, просмотр галереи.
3. Модуль организатора.
 - 3.1. Просмотр профиля пользователя.
 - 3.2. Возможность начислять макскоины и отправлять уведомления.

Для получения доступа ко всей функциональности приложение в первую очередь пользователь должен авторизоваться. Последовательность данного процесса проста:

1. Первый этап: неавторизованный пользователь заходит в приложение, где осуществляется проверка на наличие токенов доступа, которые как раз и проверяются на веб-сервере в качестве критерия, заходил ли участник или организатор раньше в систему. Если не найдено этих токенов, то пользователь должен получить их.

2. Второй этап: запрос на вход, который приходит на сам веб-сервер. Это может быть запрос на вход или регистрацию, для проверки логина и пароля или для регистрации нового пользователя так или иначе необходим запрос в базу данных, для создание новой записи или уже существующих.

3. Третий этап: запрос на данные входа в базу данных осуществляют с помощью ORM Sequalize, который формирует запрос.

4. Четвертый этап: возвращение данных из СУБД.

5. Пятый этап: Проверка данных из СУБД и из DTO с клиента, после чего формируется объект для ответа на запрос с клиента.

6. Шестой этап: На основе DTO из веб-сервера формируется ответ для пользователя в приложении, где он либо успешный, когда участник или организатор получает токены для дальнейшего пользование приложением или наоборот получает статус ошибки, где он скорее всего неправильно ввел данные, тогда приложение должно описать, где конкретно ошибся пользователь.

На рисунке 7 рассматривается диаграмма последовательности действий во время авторизации пользователя:

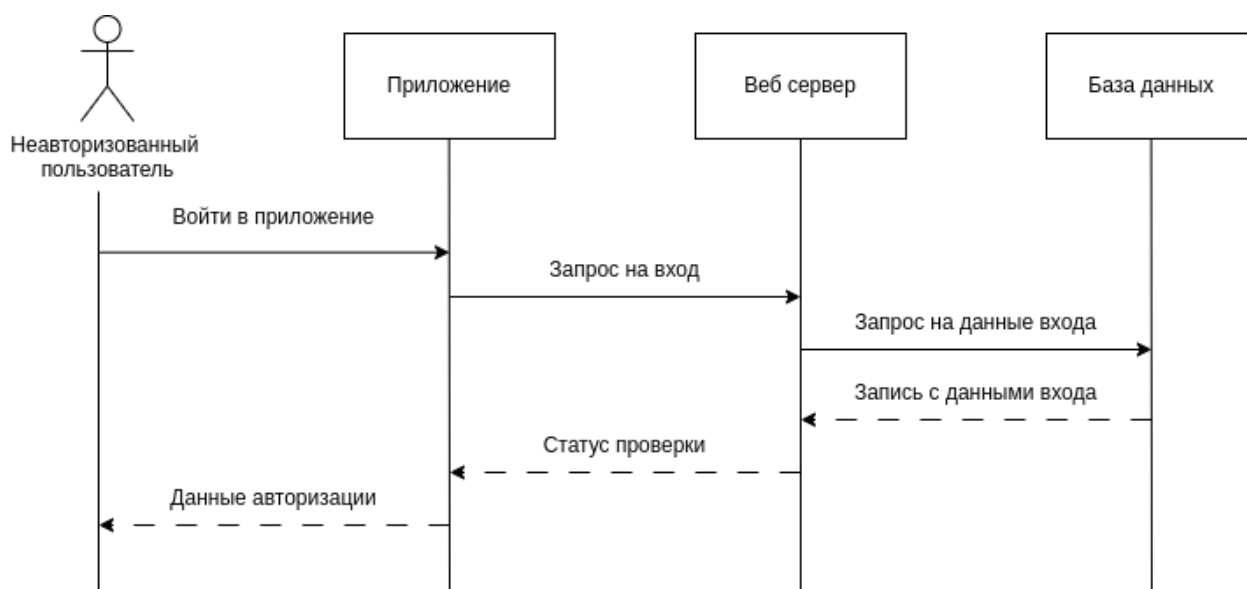


Рисунок 7 – Аутентификация пользователя.

Получение расписания - это немаловажный этап, т.к. посетитель не знает о месте и времени проведения различных мероприятий на конференции. Описание данной последовательности состоит из нескольких шагов:

1. Первый шаг: запрос от пользователя для получения расписания.
2. Второй шаг: запрос от приложения к веб серверу для получения объекта расписания, но для его формирования необходимы данные, поэтому осуществляется, следующий шаг.
3. Третий шаг: запрос в базу данных.
4. Четвертый шаг: получение из базы необходимых записей для формирования объекта.
5. Пятый шаг: формирование объекта расписания и отправка его приложению.
6. Шестой шаг: отрисовка клиентов полученного объекта расписания из веб-сервера.

Наглядно эти шаги можно увидеть на рисунке 8, на котором представлена uml диаграмма последовательности:

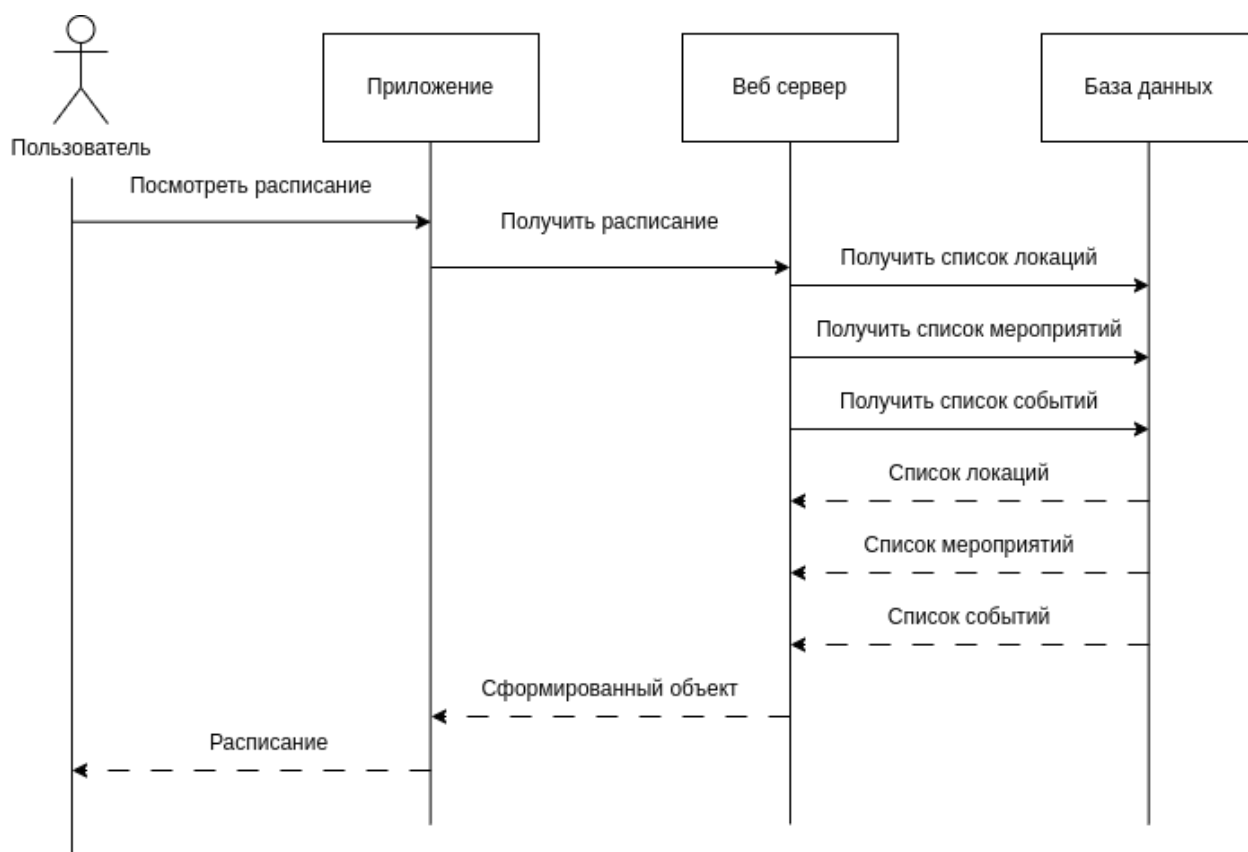


Рисунок 8 – Получение расписания.

В случае непредвиденных изменений в расписании и не только, организатор может разослать уведомление для оповещения пользователей. Данная последовательность аналогично с предыдущими процессами можно описать следующими шагами:

1. Первый шаг: запрос организатора отправить уведомления, где уже сформирован DTO для отправки на веб-сервер.
2. Второй шаг: запрос на сам веб сервер, который формирует собственный объект, который направляет на Firebase.
3. Третий шаг: Firebase принимает объект, сформированный на веб сервере, проверяет его валидность, далее по полученному списку идентификационных токенов высылает push уведомления на устройства.
4. Четвертый шаг: Это сохранение записи отправленного или не отправленного уведомления в базу данных.
5. Пятый шаг: это возвращение данных и статуса операции из базы данных и Firebase.

6. Шестой шаг: это формирование объекта для приложения о статусе операции.

7. Седьмой шаг: это отрисовка на основе пришедшего объекта с веб-сервера;

Наглядно данные шаги на рисунке 9:

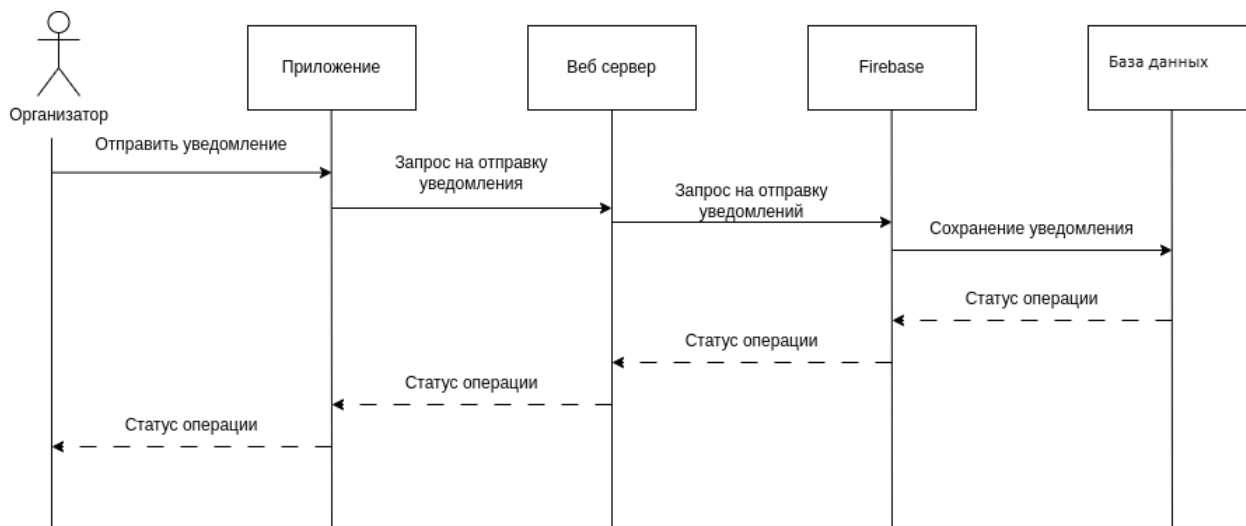


Рисунок 9 –Отправка уведомлений.

Не мало важная информация — это спонсоры конференции, представители которых будут присутствовать. Данная последовательность шагов достаточно проста:

1. Первый шаг: запрос на веб-сервер для получения списка партнеров.
2. Второй шаг: запрос на получение записей из базы данных.
3. Третий шаг: получение записей и формирование объекта для отрисовки на клиенте.
4. Четвертый шаг: отрисовка в приложении списка партнеров мероприятия.

Наглядно данные шаги на рисунке 10:



Рисунок 10 – Получение списка спонсоров.

На основе вышесказанного можно сделать следующий вывод: серверная часть для проведения оффлайн мероприятий состоит из различных модулей, которые обеспечивают пользователям возможность комфортного времяпровождения и общения посетителей и администрирования конференции и возможность предупреждения пользователей, в случае изменения расписания или другой ситуации.

2.2 Алгоритм работы системы

Рассматривается алгоритм работы приложения по отношению к посетителю конференции.

1. Вход в приложение.
2. Проверка аутентификации.
 - 2.1. Осуществляется проверка токенов доступа, есть ли они.
 - 2.2. При условии «Да» пользователь входит в приложение.
 - 2.3. При условии «Нет» пользователя отправляют на экран входа/регистрации.
3. Выбор раздела.
 - 3.1. Профиль.
 - 3.1.1. Редактирование профиля.
 - 3.1.2. Запись на консультацию.

3.1.3. Просмотр списка спонсоров.

3.1.4. Просмотр галереи.

3.1.5. Просмотр программы мероприятий.

3.2. Уведомления.

3.2.1. Посмотреть список уведомлений.

3.2.2. Открыть подробности уведомления.

3.3. QR-code.

3.3.1. Сканирование qr-code.

3.3.2. Показ qr-code пользователя.

3.4. Розыгрыши.

3.4.1. Посмотреть список розыгрышей.

3.4.2. Поучаствовать в розыгрыше.

3.5. Максикоины.

3.5.1. Получить список возможных товаров.

3.5.2. Купить товар.

4. Выход из приложения.

Система предоставляет большой функционал, как для посетителей конференции, так и для организаторов. Позволяет легко отслеживать активность пользователей, также в системе есть возможность покупать товары за активность на конференции.

Блок-схема на рисунке 11 наглядно представляет алгоритм работы системы.

Целевая аудитория: организаторы и посетители конференций;

Условные обозначения:

- Начало/конец (овал): начало и конец алгоритма;
- Процесс (прямоугольник): действия пользователя;
- Предопределенный процесс (прямоугольник в прямоугольнике):

взаимодействие пользователя с системой;

- Решение (ромб): принятые решения;
- Канал связи (стрелка): переход между действиями или решениями.

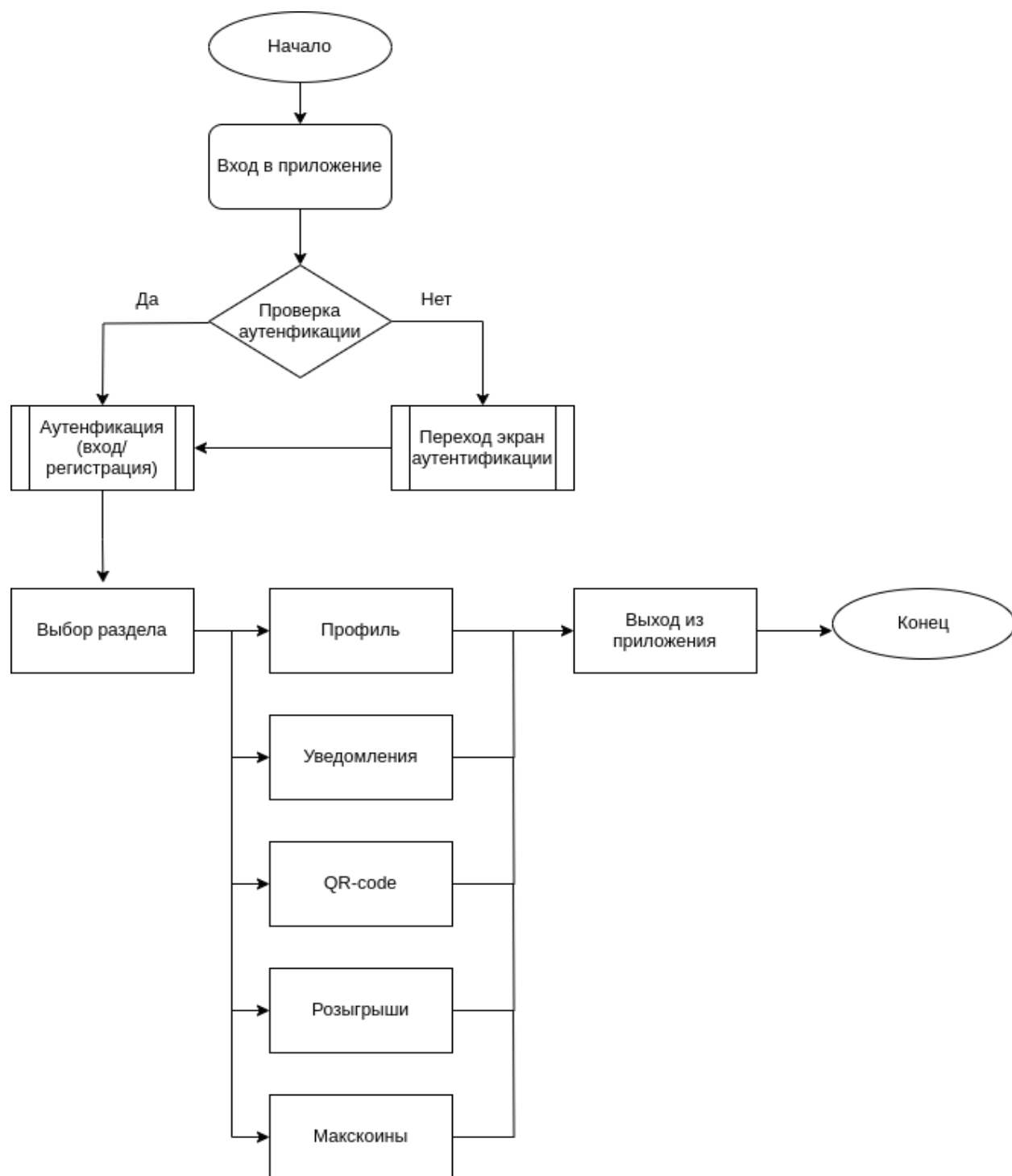


Рисунок 11 – Схема работы приложения - Блок схема.

Таким образом, блок-схема предоставляет алгоритм работы системы для проведения оффлайн мероприятий.

3.Разработка программного обеспечения

В связи с большим объемом данных, появилась необходимость структурировать информацию, для этого использовалась СУБД PostgreSQL.

3.1 Инфологическое проектирование

Предметная область: Система для проведения оффлайн мероприятий

Назначение системы: Система, предназначенная для хранения информации о пользователе, партнерах конференции, организаторах, событиях, расписания, новостях, локациях, лекциях, услугах для зарабатывания макскоинов, списка товаров для списания макскоинов, розыгрыши, ролях пользователей.

База данных должна содержать сведения о следующих объектах:

Пользователи, новости, розыгрыши, партнеры, лекции, локации, события, ролях пользователей, модели за списания и начисления макскоинов.

На основе анализа предметной области, назначение системы и диаграммы прецедентов, с помощью CASE-средства Dbeaver, была разработана схема базы данных, которая приведена на рисунке 12 и описана в таблице 2.

Таблица 2 - Таблицы в базе данных

Таблицы	Атрибуты	Описание
lectures	id, topic, speaker	Таблица для хранения лекций.
locations	id, name, logo	Таблица для хранения локаций.
vendors	id, name, logo	Таблица для хранения спонсоров.
simple_objects	id, title, description, eventId	Таблица для хранения объектов событием

Продолжение таблицы 2

Таблицы	Атрибуты	Описание
roles	id, value, description	Таблица для хранения ролей
users	id, firstName, lastName, patronymic, phone, company, position, email, logo, code, status	Таблица для хранения пользователей
profiles	id, userId, coinEarned, applications, eventCounter, raffleConter, statusPush, token, lecture, eventStation, raffles	Таблица для хранения профилей
news	id, title, shortDescription, fullDescription, type, photo, statusRead, profileId	Таблица для хранения новостей
raffles	id, title, type, idCompleted, isActive, timeStart, participants	Таблица для хранения розыгрышей
Coin_model	id, title, type, body, value	Таблица для хранения модели для траты и пополнения макскоинов
complex_object	id, locacationId, lectureId, vendorId, eventId	Таблица для хранения объектов с локацией, лекцией и событием.
User_roles	id, roleId, userId	Таблица для связи many to many

Продолжение таблицы 2

Таблицы	Атрибуты	Описание
Image_migorni	id, numberVotes, image, profileId	Таблица для хранения сгенерированных картинок
events	Id, timeStart, timeFinish	Таблица для хранения событий

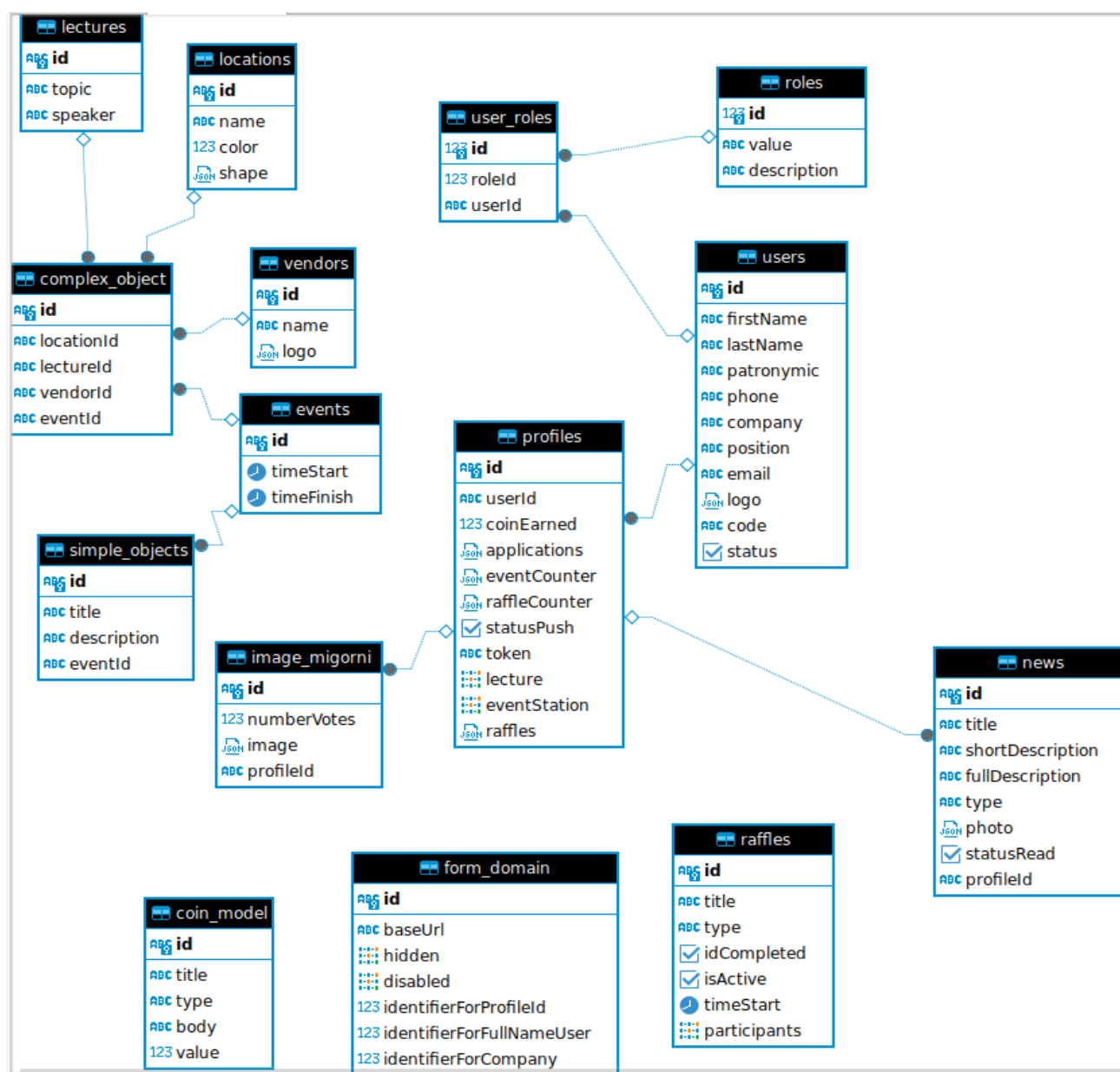


Рисунок 12 – Схема база данных – Case средство dbeaver.

Полученный список таблиц в dbeaver на рисунке 13:

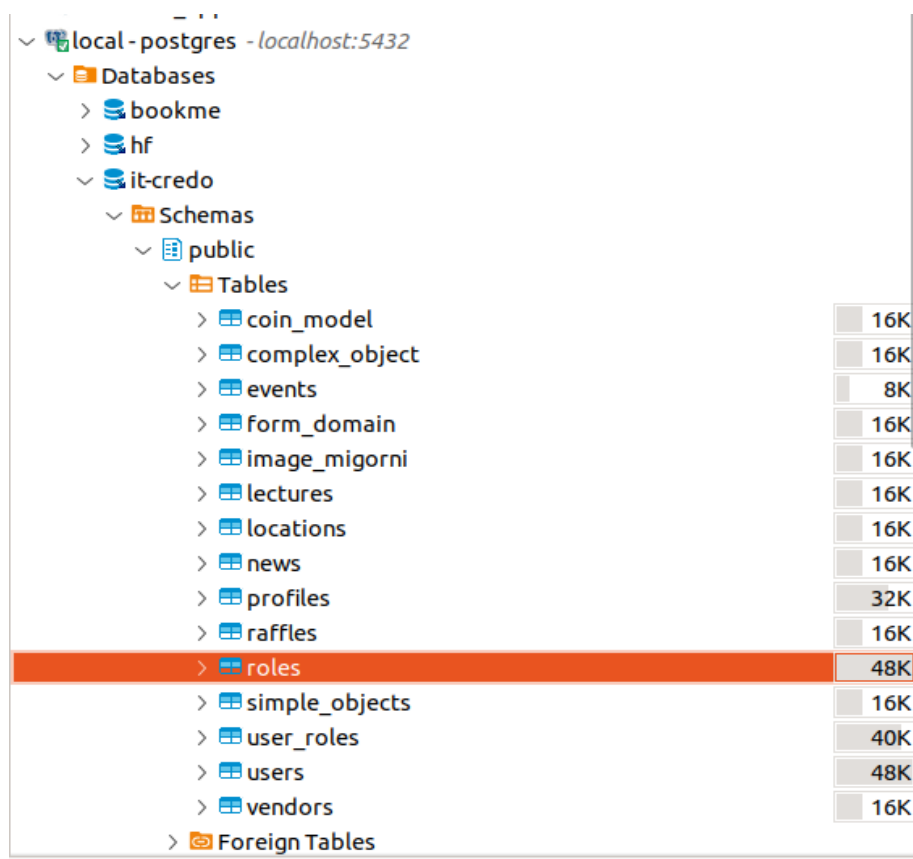


Рисунок 13 - Список таблиц в case средство dbeaver.

Таким образом была сформирована схема база данных для серверной части приложения, состоящая из 15 таблиц.

3.2 Связи между таблицами

На рисунке 12 была представлена список таблиц в базе данных, между ними были созданы связи такие как:

1. Users (Пользователи) имеет связь многим ко многим с таблицей roles и связь один ко многим к Profiles (Профиль).
2. Profiles(Профили) имеет связь один ко многим image_migorni (сгенерированные картинки) и news (новости) и связь один ко одному с таблицей users(пользователи).
3. Lectures(Лекции) имеет связь один ко многим к complex_object (Объект с данными о локациях событиях, спонсоре).

4. Locations (Локации) имеет связь один ко многим к complex_object (Объект с данными о локациях событий, спонсоре).

5. Vendors (Спонсоры) имеет связь один ко многим к complex_object (Объект с данными о локациях событий, спонсоре).

6. Events (События) имеет связь один ко многим к complex_object (Объект с данными о локациях событий, спонсоре) и таблице simple_object (Объект с данными о событии).

7. Complex_object (Объект с данными о локациях событий, спонсоре) имеет связь один ко одному с таблицами lectures(лекции) locations (локации) vendors (спонсоры) events (события).

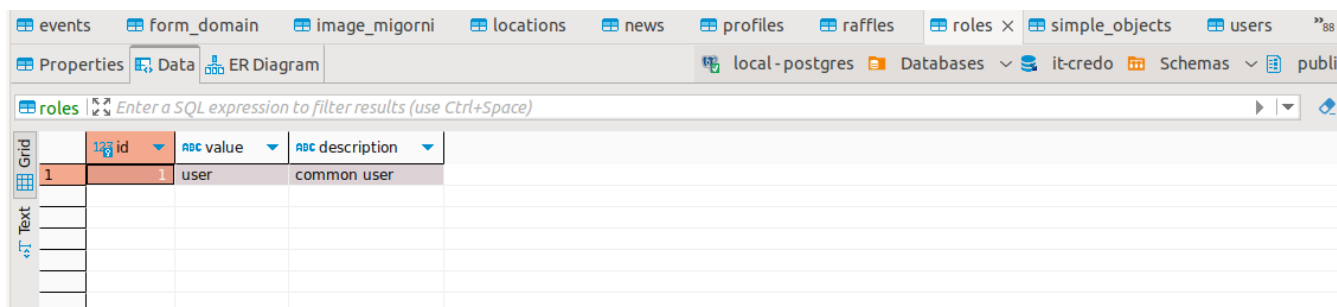
8. Simple_objects (Объект с данным о событии) имеет связь с таблицей events (события).

3.3 Перечень SQL-запросов

После разработанной схемы базы данных, было разработаны sql запросы дабы проверить работоспособность. Один из ключевых запросов, без которого невозможно создать ни одного пользователя это запрос, на создание роли и возвращение созданной записи. Выглядит он таким образом.

```
INSERT INTO "roles" ("id","value","description") VALUES (DEFAULT,$1,$2)
RETURNING "id","value","description";
```

Результат выполнение запроса на рисунке 14:



	id	value	description
1	1	user	common user

Рисунок 14 - Результат выполнения запроса на создание роли – Case средство dbeaver.

Данный запрос просто создает новую запись в таблице roles.

После созданной роли user, следует регистрация пользователя, и возвращение объекта с информацией о профиле, пользовательских данных и его роли на уровне базы данных, данный sql скрипт выглядит следующим образом:

```
Executing (default): INSERT INTO "roles" ("id","value","description") VALUES (DEFAULT,$1,$2) RETURNING "id","value","description";
```

```
Executing (default): SELECT "User"."id", "User"."firstName", "User"."lastName", "User"."patronymic", "User"."phone", "User"."company", "User"."position", "User"."email", "User"."logo", "User"."code", "User"."status", "roles"."id" AS "roles.id", "roles"."value" AS "roles.value", "roles"."description" AS "roles.description", "roles->UserRoles"."id" AS "roles.UserRoles.id", "roles->UserRoles"."roleId" AS "roles.UserRoles.roleId", "roles->UserRoles"."userId" AS "roles.UserRoles.userId" FROM "users" AS "User" LEFT OUTER JOIN ( "user_roles" AS "roles->UserRoles" INNER JOIN "roles" AS "roles" ON "roles"."id" = "roles->UserRoles"."roleId") ON "User"."id" = "roles->UserRoles"."userId" WHERE "User"."phone" = '+77777777777';
```

```
Executing (default): SELECT "id", "value", "description" FROM "roles" AS "Role" WHERE "Role"."value" = 'user';
```

```
Executing (default): INSERT INTO "users" ("id","firstName","lastName","patronymic","phone","company","position","email","code","status") VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10) RETURNING "id","firstName","lastName","patronymic","phone","company","position","email","logo","code","status";
```

```
Executing (default): SELECT "id", "roleId", "userId" FROM "user_roles" AS "UserRoles" WHERE "UserRoles"."userId" = 'a6b89394-9b82-407e-834c-b34d75338da7';
```

```
Executing (default): INSERT INTO "user_roles" ("id","roleId","userId") VALUES (DEFAULT,1,'a6b89394-9b82-407e-834c-b34d75338da7') RETURNING "id","roleId","userId";
```

Результат работы скрипта на рисунках 15 и 16:

	id	firstName	lastName	patronymic	phone	company	position	email
1	a6b89394-9b82-407e-834c-b34d75338da7	Ivan	Ivanov	Ivanovich	+7777777777	Maksoft	developer	test@ma

Рисунок 15 - Записи в таблице users - Case средство dbeaver.

	id	userId	coinEarned	applications	eventCounter	raffleCounter	statusPush	token	lecture
1	b5b53c89-	a6b89394-9b	0	{ "id": 1, "name": "IT	{ "totalCount": 8, "curr	{ "totalCount": 4, "curr	[v]	token	{ }

Рисунок 16 - Записи в таблице profiles - Case средство dbeaver.

Данный запрос создает две записи. В таблице user, информацию о конкретном пользователе, а в таблице profiles, храниться активность пользователя на мероприятии.

Не мало важный запрос на подтверждение аккаунта после входа. В моей выпускной работе предусмотрено подтверждение аккаунта по номеру телефона, только после этого запроса можно получить токены для доступа к основному функционалу приложения. Данный запрос выглядит таким образом:

Executing (default): SELECT "id", "userId", "coinEarned", "applications", "eventCounter", "raffleCounter", "statusPush", "token", "lecture", "eventStation", "raffles" FROM "profiles" AS "Profile" WHERE "Profile"."userId" = 'a6b89394-9b82-407e-834c-b34d75338da7' LIMIT 1;

Executing (default): SELECT "User"."id", "User"."firstName", "User"."lastName", "User"."patronymic", "User"."phone", "User"."company", "User"."position", "User"."email", "User"."logo", "User"."code", "User"."status", "roles"."id" AS "roles.id", "roles"."value" AS "roles.value", "roles"."description" AS "roles.description", "roles->UserRoles"."id" AS "roles.UserRoles.id", "roles->UserRoles"."roleId" AS

```
"roles.UserRoles.roleId", "roles->UserRoles"."userId" AS "roles.UserRoles.userId" FROM
"users" AS "User" LEFT OUTER JOIN ( "user_roles" AS "roles->UserRoles" INNER JOIN
"roles" AS "roles" ON "roles"."id" = "roles->UserRoles"."roleId") ON "User"."id" = "roles-
>UserRoles"."userId" WHERE "User"."phone" = '+77777777777';
```

```
Executing (default): INSERT INTO "profiles"
("id","userId","coinEarned","applications","eventCounter","raffleCounter","statusPush","t
oken","lecture","eventStation","raffles") VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11)
RETURNING
"id","userId","coinEarned","applications","eventCounter","raffleCounter","statusPush","to
ken","lecture","eventStation","raffles";
```

Данный запрос получает запись с кодом, который был отправлен пользователю на телефон и сравнивает его с введённым, после чего возвращает ему объект с персональными данными о пользователе и его активности на мероприятии с токенами доступа к приложению

Вот пример данного объекта:

```
{
  "id": "uuid",
  "userData": {
    "id": "uuid",
    "firstName": "Ivan",
    ".....
  },
  "eventCounter": {
    "totalCount": 8,
    "currentCount": 0
  },
  "coinEarned": 0,
  "raffleCounter": {
    "totalCount": 4,
```

```

"currentCount": 0
},
"applications": [{}, {}, {}],
"statusPush": true,
"lecture": [‘, ‘, ‘],
"raffles": [{}, {}, {}],
"role": "user",
"eventStations": [],
"token": "string"
}

```

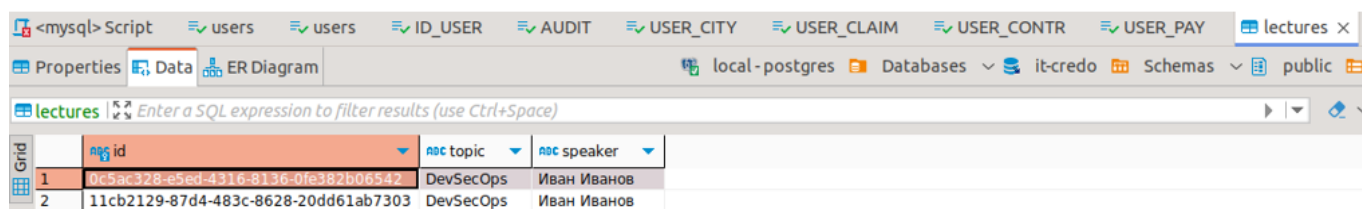
Создание новой записи в таблице лекции, запрос выглядит следующим образом:

```

INSERT INTO "lectures" ("id","topic","speaker") VALUES ($1,$2,$3)
RETURNING "id","topic","speaker";

```

Результатом, данного запроса является новая запись в таблице lectures, на рисунке 17 новая строка в case средстве dbeaver.



The screenshot shows the DBeaver interface with the 'lectures' table selected. The table has three columns: 'id', 'topic', and 'speaker'. The first row has an ID of '0c5ac328-e5ed-4316-8136-0fe382b06542', topic 'DevSecOps', and speaker 'Иван Иванов'. The second row has an ID of '11cb2129-87d4-483c-8628-20dd61ab7303', topic 'DevSecOps', and speaker 'Иван Иванов'.

	id	topic	speaker
1	0c5ac328-e5ed-4316-8136-0fe382b06542	DevSecOps	Иван Иванов
2	11cb2129-87d4-483c-8628-20dd61ab7303	DevSecOps	Иван Иванов

Рисунок 17 – Записи в таблице lectures в case средстве dbeaver.

Одним из необходимых условий участия в розыгрыше - это присутствие на лекциях. Запрос для добавления новой лекции в список, посещенных пользователем, выглядит следующим образом:

```

UPDATE "profiles" SET "lecture"=$1 WHERE "id" = $2

```

Для оповещения участников конференции используются push уведомления, но также в приложение есть запрос, отменяющий эту возможность, который выглядит так:

```

UPDATE "profiles" SET "statusPush"=$1 WHERE "id" = $2

```

Созданная база данных отвечает все поставленные задачи. Для хранения, создание, обновления и удаление записей. Разработанные запросы, выполняют поставленную задачу и результат выполнения отображается в case средстве dbeaver.

4. Тестирование отдельных сценариев работы приложения

После разработки всех запросов необходимо проверить их работоспособность и в случае нахождении ошибок исправить их. Для тестирования была обычно используются средства такие как Postman, но при разработке endpoint-ов они были задокументированы с помощью swagger

4.1 Тестирование с помощью Swagger

На рисунке 18 представлен задокументированный фрагмент:

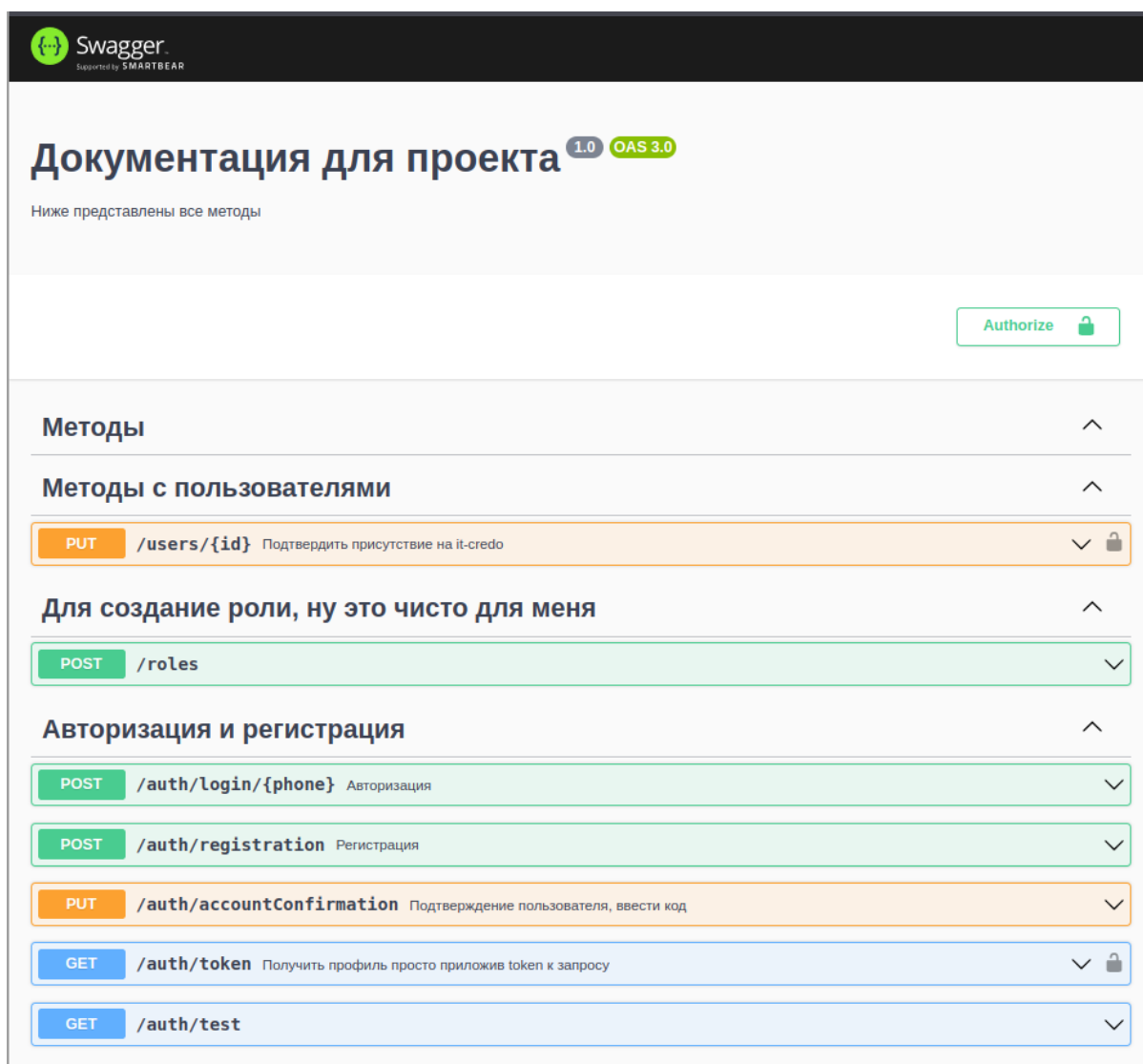


Рисунок 18 – Фрагмент документации в Swagger.

Также с помощью этого инструмента можно протестировать серверную часть.

Большая часть функций сервера доступна только для авторизованного пользователя, поэтому необходимо входим в систему. В рамках swagger, нужно получить токен доступа:

Первый шаг: войти в систему это запрос на авторизацию(auth/login/{phone}), где вводится номер телефона и на него приходит смс код;

Второй шаг: ввести номер телефона и код, который пришел, после чего в результате получили токен;

Третий шаг: вставляем токен в swagger, это на позволит прикладывать к запросу использовать весь функционал приложения;

Данную последовательность представлена на рисунках 19-23:

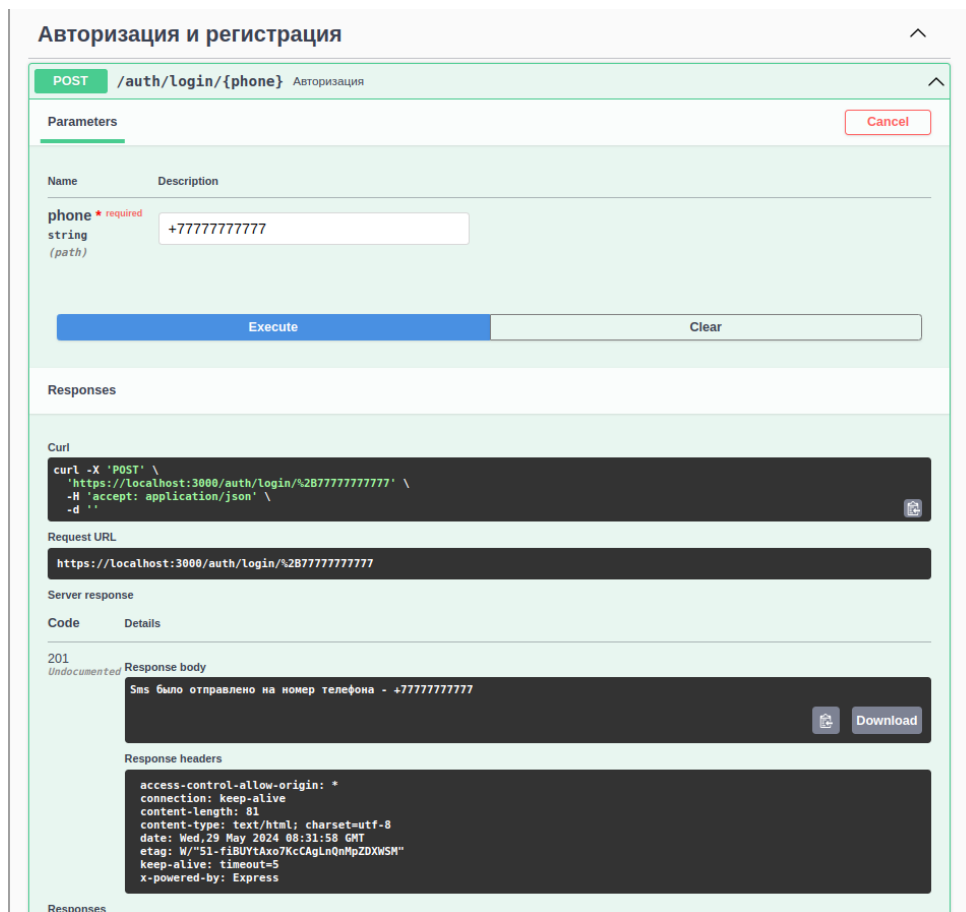


Рисунок 19 - Запрос на авторизацию в swagger.


```

25   async login(phone: string) : Promise<string> { Show usages  s.talabael *
26       const user : User = await this.userService.getUserByPhone(phone);
27       if (!user) {
28           throw new HttpException(
29               response: "Пользователь с таким номером телефона не существует",
30               HttpStatus.BAD_REQUEST
31           );
32       }
33       > if (phone === "+7777777777") {...}
38       const code : number = this.generateCode();
39       > if (phone !== "+7777777777") {...}
43       user.code = await bcrypt.hash(code.toString(), salt: 5);
44       await this.userService.saveUser(user);
45       console.log(code);
46
47       return `Sms было отправлено на номер телефона - ${phone}`;
48   }

```

Рисунок 22 – Фрагмент программы входа.

```

86   async accountConfirmation(accountConfirmation: AccountConfirmationDto) : Promise<
87       const user : User = await this.userService.getUserByPhone(
88           accountConfirmation.phone
89       );
90       if (!user) {
91           throw new HttpException(
92               response: "Пользователя с таким номером телефона нет",
93               HttpStatus.NOT_FOUND
94           );
95       }
96
97       if (
98           accountConfirmation.phone == "+7777777777" &&
99           accountConfirmation.code == "77777"
100      ) {...}
101
102       console.log(`CODE IN DB::::::::::${user.code}`);
103       console.log(`CODE IN DTO::::::::::${accountConfirmation.code}`);
104
105       if (await bcrypt.compare(accountConfirmation.code, user.code)) {
106           const userNew : User = await this.userService.updateStatusUserAndCleanCode(
107               accountConfirmation.phone
108           );
109           const token : string = await this.generateToken(user);
110
111           const statusHaveProfile : boolean = await this.profileService.statusHaveProfile(
112               userNew.id
113           );
114           if (statusHaveProfile) {
115               const profile : Profile = await this.profileService.findProfileToUserId(
116                   userNew.id
117               );
118               const profileToken : string = await this.profileService.updateToken(
119                   profile.id,
120                   accountConfirmation.token
121               );
122               return {

```

Рисунок 23 – Фрагмент программы подтверждения кода.

Помимо функционала описанных в главах ранее, наш бекенд способен раздавать статику(фотографии):

На рисунках 26-27 фрагменты кода, которые работают при запросе получения фотографий.

```
8 @Module({ metadata: { Show usages  s.stalabael
9   providers: [FilesService],
10  exports: [FilesService],
11  controllers: [FilesController],
12  imports: [
13    forwardRef( fn: ()=>AuthModule),
14    MulterModule.register( options: {
15      storage: diskStorage( options: {
16        destination: './photos', // Папка для сохранения загруженных файлов
17        filename: (req: Request, file: File, cb) :void => {
18          const uniqueSuffix: string = `${Date.now()}-${Math.round( Math.random() * 1e9)}`;
19          const extension: string = file.originalname.split( separator: '.' ).pop(); // Получение расширения загружаемого файла
20          cb( error: null, filename: `${file.fieldname}-${uniqueSuffix}.${extension}` );
21        },
22      },
23    ),
24  ],
25 })
26 export class FilesModule {}
```

Рисунок 26 – Программный код для хранения файлов.

```
215 async getPhotosYear(year: string) : Promise<...> { Show usages  s.stalabael
216   try {
217     const filePath: string = join(__dirname, '..', '..', 'photos', year);
218     const directories: string[] = await fsPromises.readdir(filePath);
219
220     let photos: any[] = []
221
222     for (let directoria: string of directories) {
223       const filePathCurrentDir: string = join(filePath, directoria)
224       const photosInDir: string[] = await fsPromises.readdir(filePathCurrentDir)
225       photos.push({
226         main: `https://${process.env.HOST}:${process.env.PORT}/${year}/${directoria}/${photosInDir[0]}`,
227         preview: `https://${process.env.HOST}:${process.env.PORT}/${year}/${directoria}/${photosInDir[1]}`,
228       })
229
230     }
231     return photos
232   } catch (e) {
233     throw new HttpException( response: 'Произошла ошибка при переименовании фото', HttpStatus.INTERNAL_SERVER_ERROR)
234   }
235 }
236 }
```

Рисунок 27 – Программный код для получения массива фотографий.

Также был подготовлен запрос, способный по наличию токена определять, конкретно пользователя. Работа такого запроса представлена на рисунке 28:

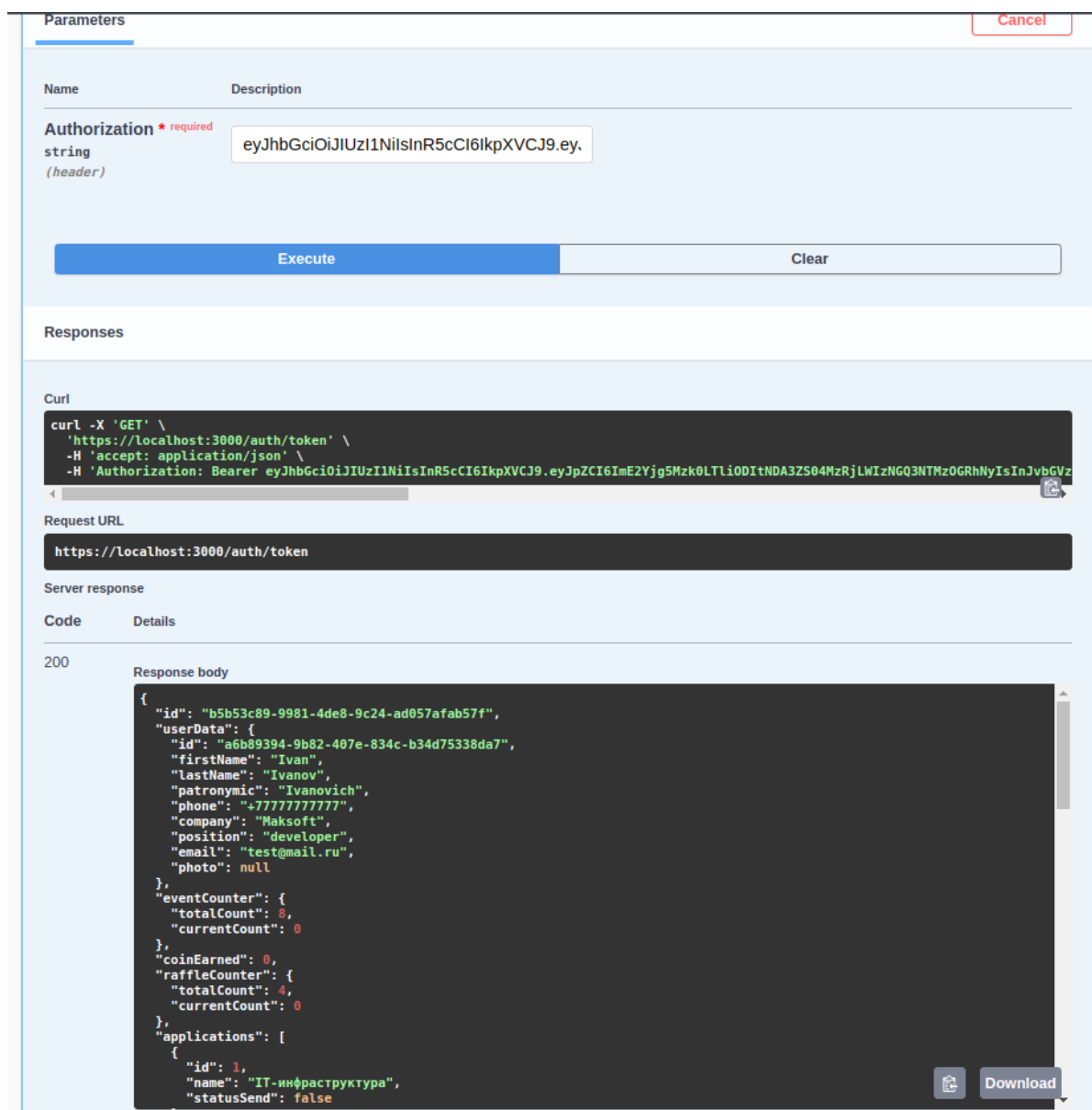


Рисунок 28 – Запрос на получение профиля по токenu доступа в swagger.

Участники, которые заинтересовались тематикой конференции, могут оставить заявку на консультацию, по предложенным темам или на иную. Работа такого запроса описана на рисунках 29-31:

На рисунке 31 представлен программный код, который осуществляет создание заявки.

```
281 async signConsultations(signConsultationsDto: SignConsultationsDto) : Promise<Profile> { Show usages ne
282   const profile : Profile = await this.profileRepository.findByPk(
283     signConsultationsDto.idUser
284   );
285   let applications : {conditionsOrPath: ..., value?: ...(), conditionsOrPath: ..., value?: ...()} = profile.applications;
286   applications[signConsultationsDto.idConsultations - 1].statusSend = true;
287
288   await this.profileRepository.update(
289     values: { applications: applications },
290     options: { where: { id: profile.id } }
291   );
292   return profile;
293 }
```

Рисунок 31 – Программный код для создания заявки на консультацию.

На сервере реализовано создание модели для траты или пополнения макскоинов. На рисунках 32 – 35 представлен и программный код, которые реализует данный запрос и тест в swagger:

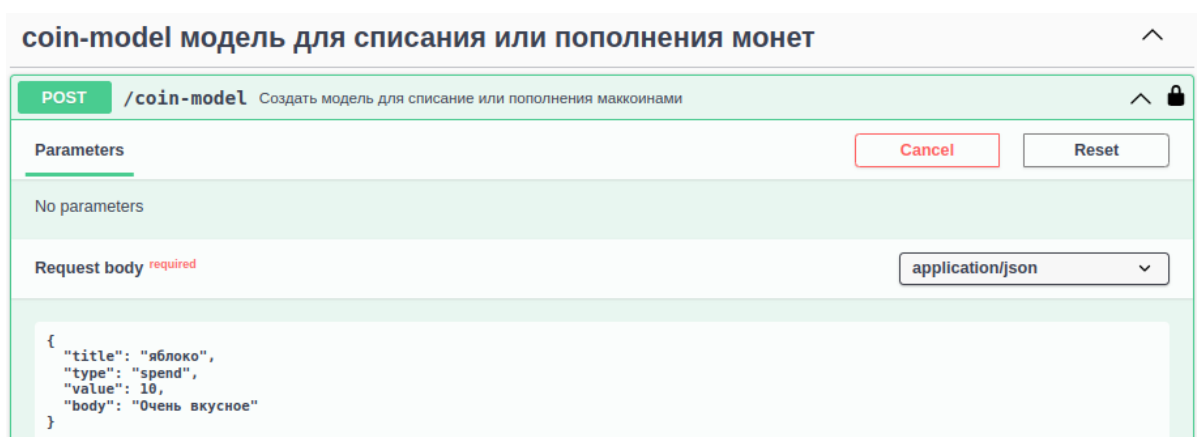


Рисунок 32 – Запрос на создание коин-модели в swagger.

```
16 @ApiOperation( options: {summary: 'Создать модель для списание или пополнения маккоинами'})
17 @ApiResponse( options: {status: 200, type: CoinModel})
18 @ApiBearerAuth()
19 @Roles( roles: 'user', 'admin')
20 @UseGuards(RolesGuard)
21 @Post()
22 createEvent(@Body() createCoinModelDto: CreateCoinModelDto) {
23   return this.coinModelService.create(createCoinModelDto);
24 }
```

Рисунок 33 – Программный код на создание контроллера коин-модели.

```

11   async create(createCoinModelDto: CreateCoinModelDto) : Promise<CoinModel> { Show usages s.s.talabael
12     const res : CoinModel = await this.coinModelRepository.create( values: {id: uuidv4(), ...createCoinModelDto})
13     return res
14   }

```

Рисунок 34 – Программный код сервиса на создание коин-модели.

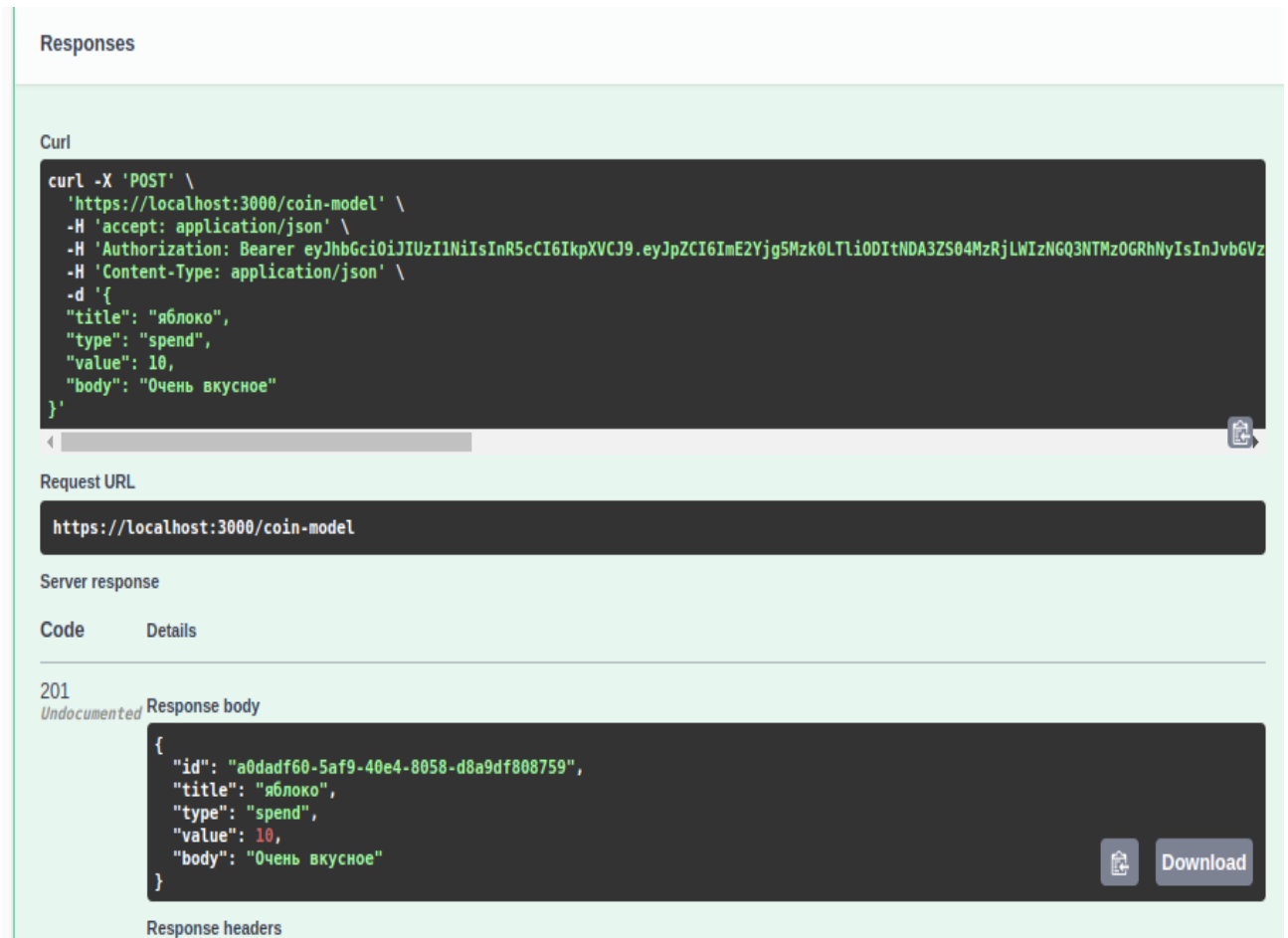


Рисунок 35 – Результат на создание коин-модели в swagger.

В предыдущей главе разработки sql запросов, были подготовлен сценарий для создания лекции. На рисунках 36-37 представлено тестирование данного запроса в swagger.

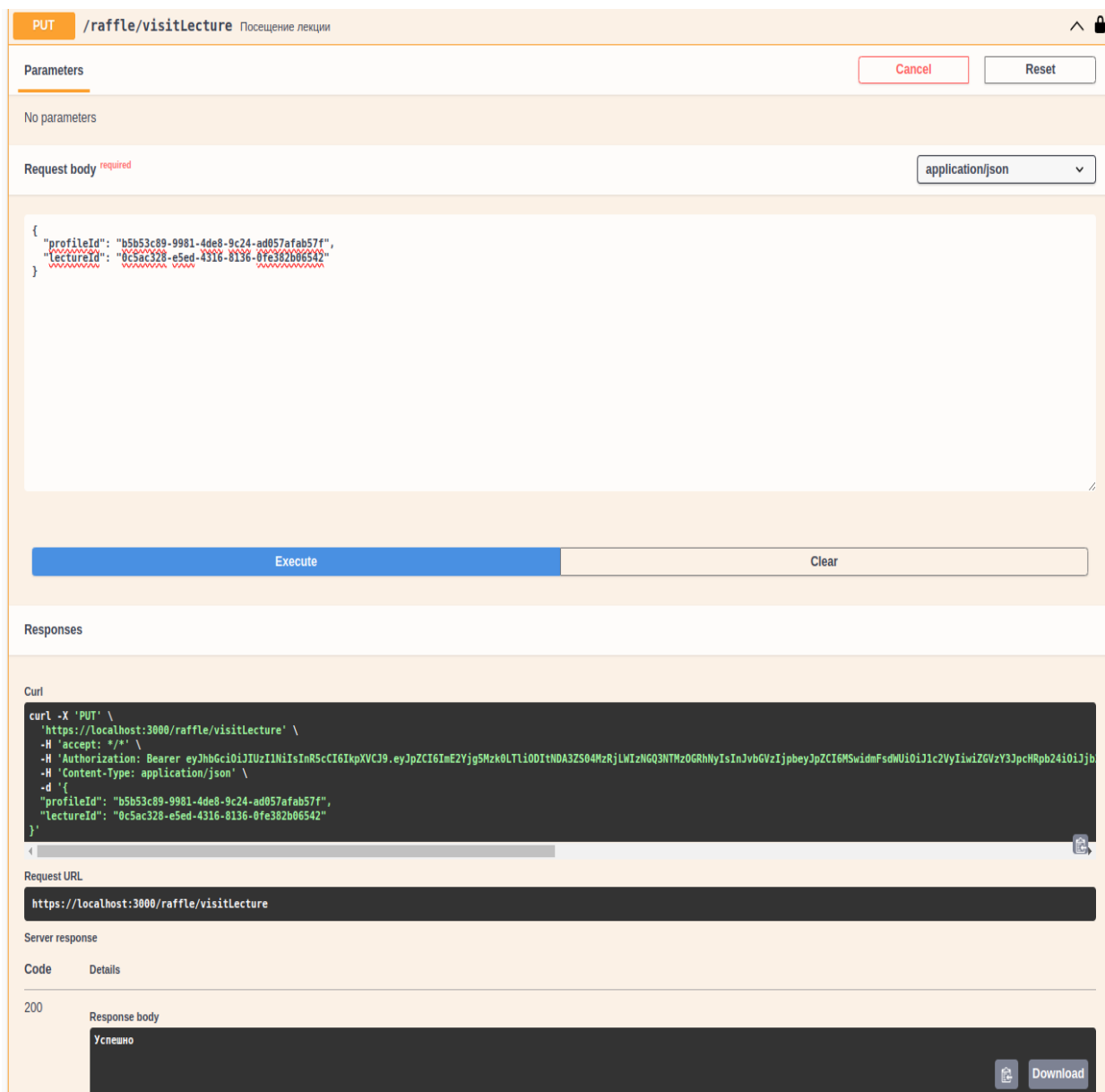


Рисунок 38 – Результат запроса на создание лекции в swagger.

В рамках данного запроса, происходит проверка на наличие данного пользователя и лекции. В случае отсутствия записи введенного идентификатора, запрос не выполниться успешно. На рисунке 39 представлен фрагмент кода, который осуществляет проверку.

```

399  async visitLecture(visitLectureDto: VisitLectureDto) : Promise<Profile> { Show usages new *
400      let profile : Profile = await this.profileRepository.findByPk(
401          visitLectureDto.profileId
402      );
403      if (!profile) {
404          throw new HttpException( response: "Профиль не найден", HttpStatus.NOT_FOUND);
405      }
406      let lectureTest : Lecture = await this.lecturesService.findById(
407          visitLectureDto.lectureId
408      );
409      if (!lectureTest) {
410          throw new HttpException( response: "На найдена такая лекция", HttpStatus.NOT_FOUND);
411      }
412      let lecture : [string] = profile.lecture;
413      const statusHave : string = lecture.find(
414          (item : string) : boolean => item === visitLectureDto.lectureId
415      );
416      if (statusHave) {
417          throw new HttpException(
418              response: "Вы уже посетили эту лекцию",
419              HttpStatus.INTERNAL_SERVER_ERROR
420          );
421      }

```

Рисунок 39 – Фрагмент кода осуществляющий проверку идентификаторов.

Не однократно упоминалось об оповещение участников посредством push уведомлений, но данную функцию можно отключить. На рисунке 40 фрагмент кода, который срабатывает при выполнении данного запроса. На рисунке 41 результат, работы запроса в dbeaver. На рисунке 42 представлен запрос с результатом выполнения.

```

73  async changeStatusPush(id: string) : Promise< { Show usages new *
74      const profile : Profile = await this.profileRepository.findByPk(id);
75      if (!profile) {
76          throw new NotFoundException( objectOrError: "Профиль не найден");
77      }
78      const statusPush : boolean = profile.statusPush;
79      await this.profileRepository.update(
80          values: { statusPush: !statusPush },
81          options: { where: { id: profile.id } }
82      );

```

Рисунок 40 –Фрагмент кода для смены статуса уведомлений.

The screenshot shows the DBeaver interface with a SQL query result. The query is filtering results based on a SQL expression. The result is displayed in a table with columns: eventCounter, raffleCounter, statusPush, token, and lecture.

eventCounter	raffleCounter	statusPush	token	lecture
{ "totalCount": 8, "currentCount": 0 }	{ "totalCount": 4, "currentCount": 0 }	[]	token	{0c5ac328-e5ed-4311-...

Рисунок 41 –Результат работы запроса для смены статуса уведомлений в dbeaver.

The screenshot shows the Swagger UI for a PUT request to the endpoint `/profile/changeStatusPush/{id}`. The request body is a JSON object with the following structure:

```
{
  "id": 8,
  "name": "Айрис и сепарс",
  "statusSend": false
}, {
  "id": 9,
  "name": "Дырое",
  "statusSend": false
}

```

The response body is a JSON object with the following structure:

```
{
  "id": 8,
  "name": "Айрис и сепарс",
  "statusSend": false
}, {
  "id": 9,
  "name": "Дырое",
  "statusSend": false
}

```

Рисунок 42 –Результат работы запроса для смены статуса уведомлений в swagger.

Таким образом система показала устойчивую работоспособность на примере отдельных сценариев работы.

Заключение

В процессе выполнения работы были исследованы аналоги разработанной системы. Была разработана структура схемы базы данных, sql запросы, диаграммы последовательностей, описаны алгоритмы работы приложения.

Таким образом, разработанная серверная часть предоставляет возможность зарегистрироваться и авторизоваться в системе как организатор или участник. Работу в нескольких режимах – организатор и участник. Участник способен:

1. Отмечаться на лекциях;
2. Получать и тратить макскоины;
3. Участвовать в событиях, проводимых на конференции;
4. Смотреть расписание лекций и их местоположение;
5. Оставлять заявку на консультацию;
6. Видеть список партнеров;
7. Видеть галерею мероприятия прошлого и текущего года;
8. Видеть события, которые присылают партнеры или организаторы;
9. Видеть счет прошедших событий на мероприятии;
10. Видеть счетчик участия в розыгрышах и счет личных макскоинов;

Организатор должен иметь такие же возможности что и участник, но при этом еще:

1. Начислять макскоины;
2. Создавать и отправлять события участникам;

Таким образом, выпускная квалификационная работа выполнена в полном соответствии с требованиями технического задания и с соблюдением сроков, уставленных календарном планом.

Список литературы:

1. Eventrocks - платформа для проведения мероприятий [Электронный ресурс]. – Режим доступа: <https://eventrocks.ru/> (дата обращения 15.05.24).
2. ticketscloud.com – платформа для проведения мероприятий [Электронный ресурс]. – Режим доступа: [https:// ticketscloud.com /](https://ticketscloud.com/) (дата обращения 15.05.24).
3. uForum – платформа для проведения мероприятий [Электронный ресурс]. – Режим доступа <https://uforum.pro/> (дата обращения 15.05.24).
4. JetBrains WebStorm [Электронный ресурс]. – Режим доступа: <https://www.jetbrains.com/webstorm/> (дата обращения 17.05.24).
5. Документация Postgres [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/docs/> (дата обращения 15.05.24).
6. Dbeaver.io сайт case средства для администрирования СУБД [Электронный ресурс] – Режим доступа: <https://dbeaver.io/> (дата обращения 17.05.24)
7. Документация Sequelize [Электронный ресурс]. – Режим доступа: <https://sequelize.org/docs/v6/> (дата обращения 15.05.24).
8. Документация Typescript [Электронный ресурс] – Режим доступа: <https://www.typescriptlang.org/docs/> (дата обращения 17.05.24).
9. Документация Node js [Электронный ресурс]. – Режим доступа: <https://nodejs-org-git-meta-performance-bundle-improvements-openjs.vercel.app/it/docs> (дата обращения 15.05.24).
10. Документация Firebase [Электронный ресурс] – Режим доступа: <https://firebase.google.com/docs?hl=en> (дата обращения 17.05.24)

Приложение А. Фрагмента исходного кода

Main.ts // Точка входа

```
async function main() {
  const port = process.env.PORT || 8000;
  //const port = 3000;
  //const host = '198.16.74.204';
  const host = process.env.HOST;
  const app = await NestFactory.create(AppModule, {
    httpsOptions: {
      key: fs.readFileSync('private.key'),
      cert: fs.readFileSync('trusted-solutions_ru.crt'),
      ca: [fs.readFileSync('AlphaSSL CA - SHA256 - G4.crt'), fs.readFileSync('GlobalSign Root
CA.crt')]
    }
  });
  //const app = await NestFactory.create(AppModule)

  const corsOptions = {
    allowedHeaders: 'X-Requested-With, X-HTTP-Method-Override, Content-Type, Accept,
Observe, Authorization',
    origin: '*', // Замените это на ваш фронтенд-сервер
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    credentials: false,
  };
  app.enableCors(corsOptions);

  const config = new DocumentBuilder()
    .setTitle('Документация для проекта')
    .setDescription('Ниже представлены все методы')
    .setVersion('1.0')
    .addTag('Методы')
    .addBearerAuth()
    .build();

  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('api/docs', app, document);

  await app.listen(port, host);
  console.log(`PORT = ${host}:${port}`)
}
main();
```

Profiles.service.ts: // сервисы для профиля

```
@Injectable()
export class ProfilesService {
  constructor(
    @InjectModel(Profile) private profileRepository: typeof Profile,
    private usersService: UsersService,
    private coinModelService: CoinModelService,
    private jwtService: JwtService,
    private lecturesService: LecturesService
  ) {}

  let profilesRes = [];
  for (let profile of profiles) {
    const profRes = await this.getById(profile.id);
    profilesRes.push(profRes);
  }
  return profilesRes;
}

async findProfilesByRaffle(profilesId: string[]) {
  let profilesRes = [];
  let counter = 0;
  for (let profileId of profilesId) {
    let profile = await this.profileRepository.findByPk(profileId);
    if (profile) {
      let user = await this.getByIdMinInfo(profileId);
      if (user) {
        counter += 1;
        const profRes = `${counter}, ${user.firstName}, ${user.lastName}, ${user.patronymic},
${user.company}, ${user.position}, ${user.email}`;
        profilesRes.push(profRes);
      }
    }
  }
  return profilesRes;
}

async changeStatusPush(id: string) {
  const profile = await this.profileRepository.findByPk(id);
  if (!profile) {
    throw new NotFoundException("Профиль не найден");
  }
  const statusPush = profile.statusPush;
  await this.profileRepository.update(
    { statusPush: !statusPush },
    { where: { id: profile.id } }
  );
  const newProfile = await this.profileRepository.findByPk(id);
  const user = await this.usersService.getById(newProfile.userId);
  return {
```

```

id: newProfile.id,
userData: {
  id: user.id,
  firstName: user.firstName,
  lastName: user.lastName,
  patronymic: user.patronymic,
  phone: user.phone,
  company: user.company,
  position: user.position,
  email: user.email,
  photo: user.logo,
},
eventCounter: newProfile.eventCounter,
coinEarned: newProfile.coinEarned,
raffleCounter: newProfile.raffleCounter,
applications: newProfile.applications,
statusPush: newProfile.statusPush,
lecture: newProfile.lecture,
raffles: newProfile.raffles,
role: user.roles[0].value,
eventStations: newProfile.eventStation,
};
}

async findProfilesByApplicationId(applicationId: number) {
  const profiles = await this.profileRepository.findAll();
  let res = [];
  for (let profile of profiles) {
    if (profile.applications[applicationId - 1].statusSend === true) {
      const user = await this.getByIdMinInfo(profile.id);
      res.push(
        `${user.firstName}, ${user.lastName}, ${user.patronymic}, ${user.email}, ${user.phone}`
      );
    }
  }
  return res;
}

getIdFromToken(authHeader: string): string {
  if (!authHeader || !authHeader.startsWith("Bearer ")) {
    return null;
  }
  return authHeader.split(" ")[1];
}

async updatePhotoProfile(id: string, photo) {
  let profile = await this.profileRepository.findById(id);
  if (!profile) {
    throw new NotFoundException("Профиль не найден");
  }

  const user = await this.userService.updatePhotoUser(profile.userId, photo);
  return {
    id: profile.id,
    userData: {
      id: user.id,

```

```

    firstName: user.firstName,
    lastName: user.lastName,
    patronymic: user.patronymic,
    phone: user.phone,
    company: user.company,
    position: user.position,
    email: user.email,
    photo: user.logo,
  },
  eventCounter: profile.eventCounter,
  coinEarned: profile.coinEarned,
  raffleCounter: profile.affleCounter,
  applications: profile.applications,
  statusPush: profile.statusPush,
  lecture: profile.lecture,
  raffles: profile.raffles,
  role: user.roles[0].value,
  eventStations: profile.eventStation,
};
}

async getProfilesWithMigorni() {
  return await this.profileRepository.findAll({ include: [ImageMigorni] });
}
async findProfileToUserId(userId: string) {
  return await this.profileRepository.findOne({ where: { userId } });
}
async getAll() {
  const profiles = await this.profileRepository.findAll({
    include: { all: true },
  });
  return profiles;
}

async statusProfile(id: string) {
  return await this.profileRepository.findByPk(id);
}

async getById(id: string) {
  const profile = await this.profileRepository.findOne({ where: { id } });

  async getByIdMinInfo(id: string) {
    const profile = await this.profileRepository.findOne({ where: { id } });
    const user = await this.usersService.getById(profile.userId);
    return {
      firstName: user.firstName,
      lastName: user.lastName,
      patronymic: user.patronymic,
      phone: user.phone,
      company: user.company,
      position: user.position,
    };
  }
}

```

```

    email: user.email,
    coinEarned: profile.coinEarned,
    raffleCounter: profile.affleCounter,
    lecture: profile.lecture,
    raffles: profile.raffles,
  };
}

async findProfilesByLecture(lectureId: string) {
  const profiles = await this.profileRepository.findAll();
  const lecture = await this.lecturesService.findById(lectureId);
  let res = [];
  for (let profile of profiles) {
    const statusVisitLecture = profile.lecture.find(
      (lect) => lect === lecture.id
    );
    if (statusVisitLecture) {
      let user = await this.getByIdMinInfo(profile.id);
      res.push( `${user.firstName} ${user.lastName} ${user.patronymic}` );
    }
  }
  return res;
}

let newRaffles = profile.raffles.find(
  (item) => item.name === confirmationParticipation.raffles
);
console.log(`TEST NEWRAFFLES:.....${newRaffles}`);
let raffles = profile.raffles;
if (newRaffles.statusParticipant === true) {
  throw new HttpException(
    "Вы уже участвуете в этом розыгрыше",
    HttpStatus.OK
  );
}
newRaffles.statusParticipant = true;
raffles[newRaffles.id - 1] = newRaffles;
let newRaffleCounter = profile.affleCounter;
newRaffleCounter.currentCount += 1;
if (confirmationParticipation.raffles === "contest") {
  let coin = profile.coinEarned + 10;
  await this.profileRepository.update(
    { raffles: raffles, raffleCounter: newRaffleCounter, coinEarned: coin },
    { where: { id: confirmationParticipation.profileId } }
  );
} else {
  await this.profileRepository.update(
    { raffles: raffles, raffleCounter: newRaffleCounter },
    { where: { id: confirmationParticipation.profileId } }
  );
}

```

```

    }
    return await this.profileRepository.findByPk(
      confirmationParticipation.profileId
    );
  }

  try {
    const response = await this.httpService.post(
      'https://fcm.googleapis.com/fcm/send',
      message,
      { headers },
    ).toPromise();

    return response.data;
  } catch (error) {
    throw error;
  }
}*/

/* async sendPushNotification(sendNewsDto: SendNewsDto, image) {
  const serverKey = // Замените на ваш ключ сервера FCM

  const headers = {
    Authorization: `key=${serverKey}`,
    'Content-Type': 'application/json',
  };
  const title = sendNewsDto.title
  const body = sendNewsDto.shortDescription
  const registrationTokens = await this.getAllTokens()
  const message = {
    notification: {
      title,
      body,
    },
    registration_ids: registrationTokens,
  };

  try {
    const response = await this.httpService.post(
      'https://fcm.googleapis.com/fcm/send',
      message,
      { headers },
    ).toPromise();
    const notification = await this.newsService.createNews( {title: title, shortDescription: body,
type: sendNewsDto.type}, image)
    return notification;

  } catch (error) {
    throw error;
  }
}

```

```

}*/

async visitLecture(visitLectureDto: VisitLectureDto) {
  let profile = await this.profileRepository.findByPk(
    visitLectureDto.profileId
  );
  if (!profile) {
    throw new HttpException("Профиль не найден", HttpStatus.NOT_FOUND);
  }
  let lectureTest = await this.lecturesService.findById(
    visitLectureDto.lectureId
  );
  if (!lectureTest) {
    throw new HttpException("На найдена такая лекция", HttpStatus.NOT_FOUND);
  }
  let lecture = profile.lecture;
  const statusHave = lecture.find(
    (item) => item === visitLectureDto.lectureId
  );
  if (statusHave) {
    throw new HttpException(
      "Вы уже посетили эту лекцию",
      HttpStatus.INTERNAL_SERVER_ERROR
    );
  }
}

async updateToken(profileId: string, token: string) {
  await this.profileRepository.update(
    { token: token },
    { where: { id: profileId } }
  );
  return token;
}

/* async paticipaceContest(profileId: string) {
  3s4
}*/

```

Auth.service.ts // сервисы для авторизации и регистрации

```

@Injectable()
export class AuthService {
  constructor(
    private userService: UsersService,
    private jwtService: JwtService,
    private profileService: ProfilesService
  ) {}

  async login(phone: string) {

```

```

const user = await this.userService.getUserByPhone(phone);
if (!user) {
  throw new HttpException(
    "Пользователь с таким номером телефона не существует",
    HttpStatus.BAD_REQUEST
  );
}
if (phone === "+77777777777") {
  user.code = "77777";
  await this.userService.saveUser(user);
  return `Sms было отправлено на номер телефона - ${phone}`;
}
const code = this.generateCode();
if (phone !== "+77777777777") {
  await this.registrationPhone(phone, code.toString());
  console.log("SEND PHONE CODE");
}
user.code = await bcrypt.hash(code.toString(), 5);
await this.userService.saveUser(user);
console.log(code);

return `Sms было отправлено на номер телефона - ${phone}`;
}

async registration(userDto: CreateUserDto, photo) {
  let code;

  /* if (userDto.phone === "+77777777777") {
    userDto.code = "77777";
  } else {
    code = this.generateCode();
    userDto.code = await bcrypt.hash(code.toString(), 5);
  } */
  let test = userDto.phone[0];
  const candidate = await this.userService.getUserByPhone(userDto.phone);
  if (candidate) {
    throw new HttpException(
      "Пользователь с таким номером телефона существует",
      HttpStatus.BAD_REQUEST
    );
  }
  userDto.status = false;
  console.log(`CODE ::::::::::: ${code}`);
  const user = await this.userService.createUser(
    { id: uuidv4(), ...userDto },
    "user",
    photo
  );
  console.log(`CODE ::::::::::: ${code}`);
  if (userDto.phone !== "+77777777777") {
    await this.registrationPhone(userDto.phone, code)
  }
}

```

```

    console.log("SEND PHONE CODE");
  }
  return `Sms было отправлено на номер телефона - ${user.phone}`;
}

private generateCode = () => {
  return Math.floor(Math.random() * (99999 - 10000 + 1)) + 10000;
};
async registrationPhone(phone: string, code: string) {
  let SMSru = require("sms_ru");
  let sms = new SMSru("40EA2BCA-F49C-086F-6352-E36B7A49AD70");
  sms.sms_send(
    {
      to: phone,
      text: `Код для it-credo: ${code}`,
    },
    function (e) {
      console.log(e.description);
    }
  );
}

private async generateToken(user: any) {
  const payload = { id: user.id, roles: user.roles };
  return this.jwtService.sign(payload);
}

// Создание токена доступа
async generateAccessToken(userId: string): Promise<string> {
  const payload = { userId };
  const accessTokenSecret = process.env.ACCESS_TOKEN_SECRET;
  const accessToken = await this.jwtService.signAsync(payload, {
    secret: accessTokenSecret,
    expiresIn: "15m",
  });
  return accessToken;
}

// Создание токена обновления
async generateRefreshToken(userId: string): Promise<string> {
  const payload = { userId };
  const refreshTokenSecret = process.env.REFRESH_TOKEN_SECRET;
  const refreshToken = await this.jwtService.signAsync(payload, {
    secret: refreshTokenSecret,
    expiresIn: "7d",
  });
  return refreshToken;
}

async getUserIdFromToken(authHeader: string) {
  if (!authHeader || !authHeader.startsWith("Bearer ")) {

```

```

    throw new UnauthorizedException("Invalid authorization header format");
}

const token = authHeader.split(" ")[1]; // Получаем токен из заголовка

try {
    const decodedToken: any = this.jwtService.decode(token); // Декодируем токен
    if (!decodedToken || !decodedToken.id) {
        throw new UnauthorizedException("Invalid beaver token");
    }

    const profile = await this.profileService.findProfileToUserId(
        decodedToken.id
    );
    const user = await this.userService.getById(profile.userId);
    return {
        id: profile.id,
        userData: {
            id: user.id,
            firstName: user.firstName,
            lastName: user.lastName,
            patronymic: user.patronymic,
            phone: user.phone,
            company: user.company,
            position: user.position,
            email: user.email,
            photo: user.logo,
        },
        eventCounter: profile.eventCounter,
        coinEarned: profile.coinEarned,
        raffleCounter: profile.affleCounter,
        applications: profile.applications,
        statusPush: profile.statusPush,
        lecture: profile.lecture,
        raffles: profile.raffles,
        role: user.roles[0].value,
        eventStations: profile.eventStation,
        token,
    };
}

```