Техническое задание на разработку систему по бронированию билетов на мероприятия EventHub

Тема: Система бронирования билетов на мероприятия с расширенным функционалом

Исполнители проекта:
Айдар - Fullstack разработчик
Элдос - Fullstack разработчик
Нуркыз - Frontend разработчик, UX/UI-дизайнер

Краткое описание:

Проект представляет собой платформу для бронирования билетов на мероприятия. Система разработана для удовлетворения потребностей широкой аудитории: от обычных пользователей, желающих быстро и удобно приобретать билеты на мероприятия, до организаторов, стремящихся эффективно управлять продажей билетов и контролировать посещаемость своих мероприятий.

Цель:

Создать удобную и многофункциональную платформу для бронирования билетов на мероприятия, а также предоставляет организаторам инструменты для эффективного управления продажами и контролем посещаемости.

Целевая аудитория:

- **Обычные пользователи:**
  Люди, желающие бронировать и покупать билеты, просматривать информацию о мероприятиях, оставлять отзывы и искать компанию для совместного посещения.
- **Организаторы мероприятий:**
  Лица и организации, проводящие мероприятия, использующие

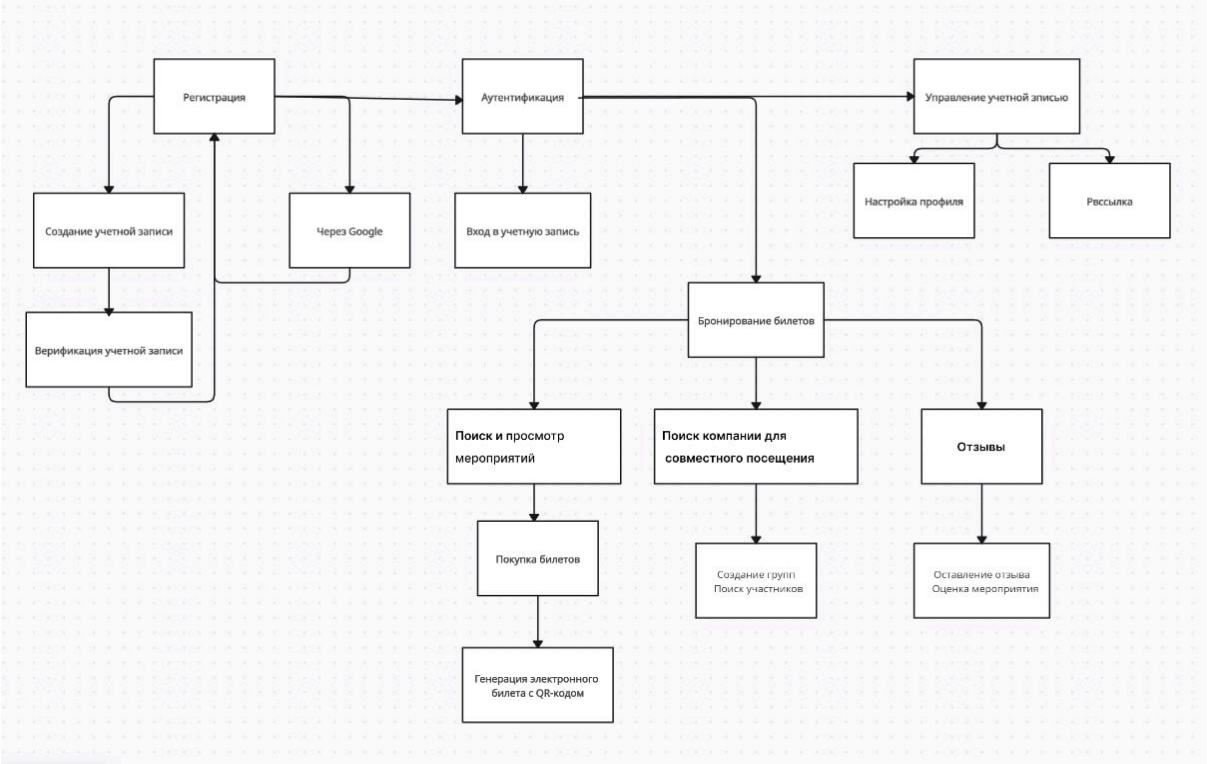
платформу для продажи билетов и управления бронированиями.

Уникальные особенности проекта:

- **Генерация электронных билетов с QR-кодом:**
  Автоматическая генерация билетов с зашифрованной информацией, позволяющая ускорить процесс входа на мероприятие.
- **Расширенный социальный функционал:**
  Возможность оставлять комментарии, оценки, создавать и присоединяться к группам для совместного посещения мероприятий.
- **Персонализированные рассылки:**
  Подписка на избранные категории мероприятий с автоматической отправкой подборок на email
- **Auth через через Google account (OAuth 2.0)**

  Интеграция с google api

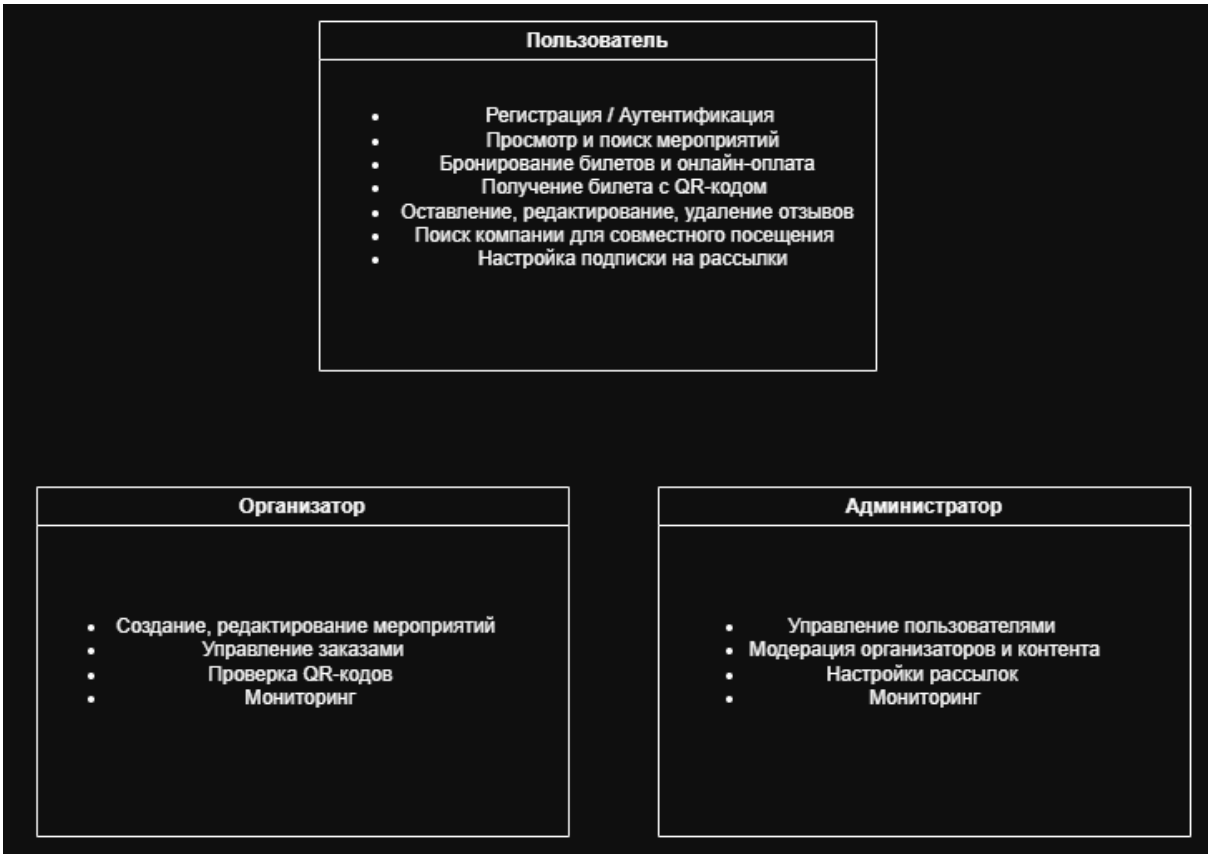
Функционал (Functional diagram)



Основные модули системы:

1. **Пользовательский модуль:**
    - **Регистрация и аутентификация:** Вход/регистрация через email, Google Auth.
    - **Просмотр и поиск мероприятий:** Фильтрация по дате, категории, местоположению.
    - **Бронирование и покупка билетов:** Добавление билетов в корзину, оформление заказа, онлайн-оплата.
    - **Генерация билетов:** Автоматическое создание электронного билета с уникальным QR-кодом.
    - **Отзывы и рейтинг:** Оставление комментариев, оценок (звёзды, лайки/дизлайки) и возможность редактирования/удаления собственных отзывов.
    - **Поиск компании:** Создание заявок для совместного посещения мероприятий, просмотр существующих групп, общение через чат.
    - **Настройка рассылок:** Выбор избранных категорий, настройка периодичности рассылок.
2. **Модуль организаторов:**
    - **Управление мероприятиями:** Создание, редактирование и удаление мероприятий.
    - **Мониторинг продаж и бронирований:** Просмотр статистики, управление заказами.
    - **Генерация и проверка билетов:** Контроль использования QR-кодов на входе.
3. **Административный модуль:**
    - **Модерация контента:** Управление отзывами, комментариями и пользовательскими заявками.
    - **Управление пользователями:** Блокировка/разблокировка, верификация организаторов.
    - **Управление рассылками и уведомлениями:** Настройка отправки уведомлений через email.
    - **Аналитика и отчёты:** Статистика посещаемости, продаж и активности пользователей.

Use case diagram:



Used technologies stack:

Backend:

- Язык и платформа: TypeScript, Node.js.
- Фреймворк: NestJS.
- ORM: TypeORM или Prisma.
- База данных: PostgreSQL (или MySQL).
- Аутентификация: JWT, OAuth2.

Frontend:

- Язык: TypeScript, JavaScript.
- Фреймворк: React.

Интеграционные модули:

- Генерация QR-кодов: библиотека qrcode .
- Auth 2.0: Google Api
- Платежные модули: Stripe

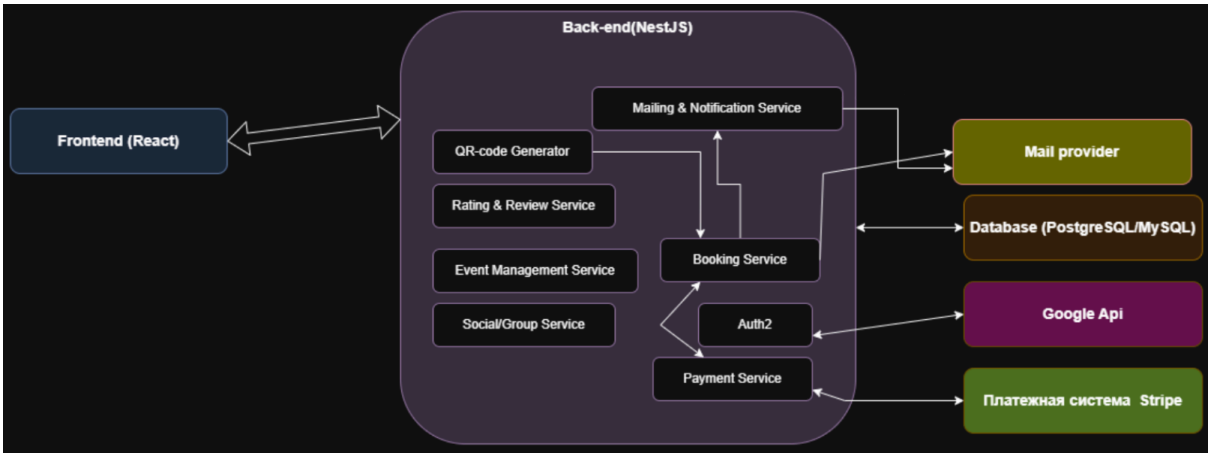
DevOps и инфраструктура:

- Контейнеризация: Docker.
- Хостинг: DigitalOcean или другой облачный провайдер.

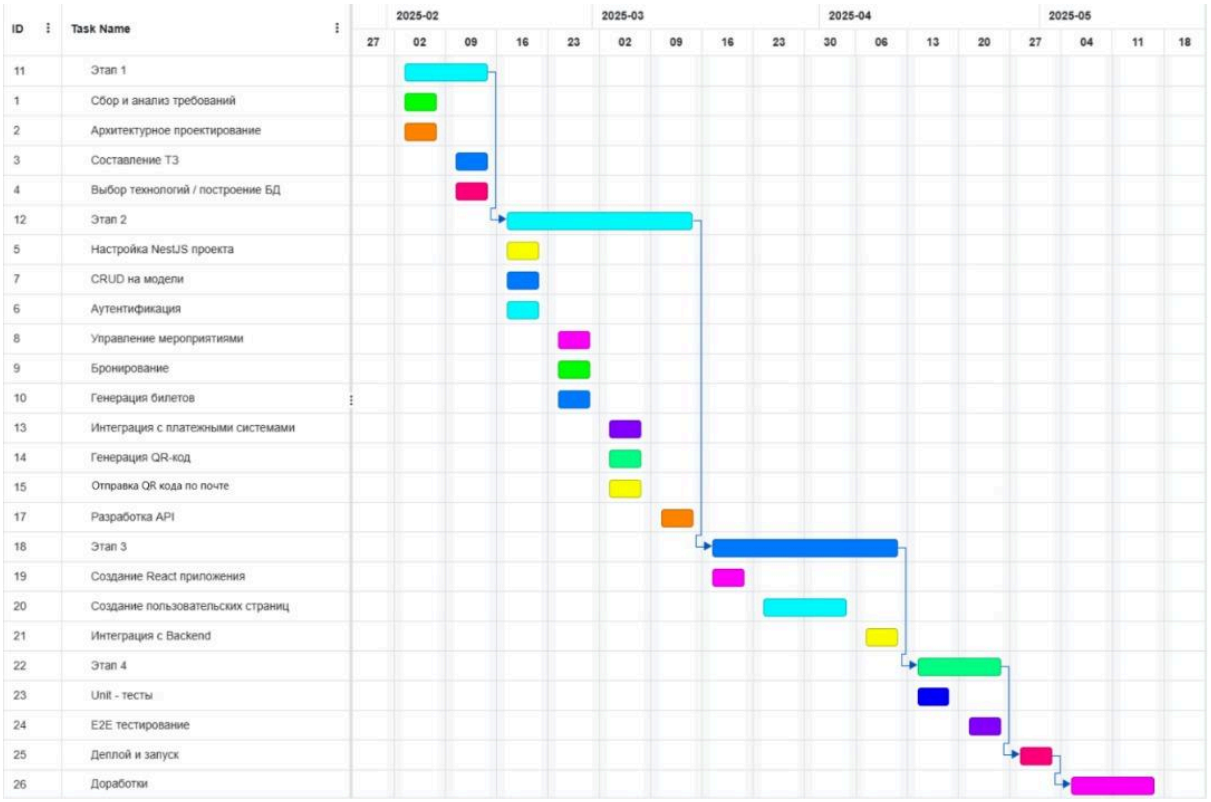
Тестирование и документация:

- Unit и интеграционные тесты: Jest, Mocha.
- Документация API: Swagger/OpenAPI.

Общая архитектура системы

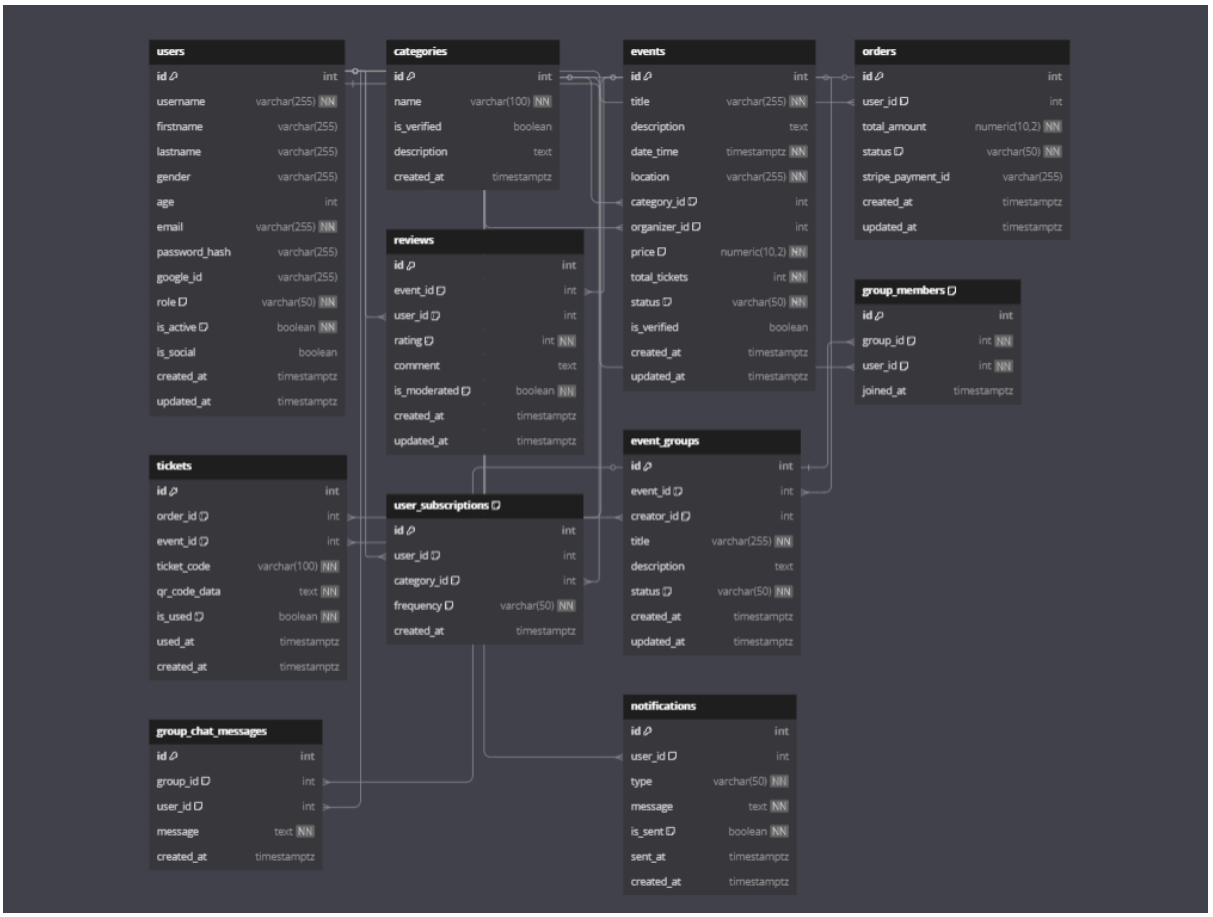


Gantt Diagram



ID	Task Name
11	Этап 1
1	Сбор и анализ требований
2	Архитектурное проектирование
3	Составление ТЗ
4	Выбор технологий / построение БД
12	Этап 2
5	Настройка NestJS проекта
7	CRUD на модели
6	Аутентификация
8	Управление мероприятиями
9	Бронирование
10	Генерация билетов
13	Интеграция с платежными системами
14	Генерация QR-код
15	Отправка QR кода по почте
17	Разработка API
18	Этап 3
19	Создание React приложения
20	Создание пользовательских страниц
21	Интеграция с Backend
22	Этап 4
23	Unit - тесты
24	E2E тестирование
25	Деплой и запуск
26	Доработки

ER модель



Сценарии

1. Сценарий: Регистрация пользователя

Описание:

Новый пользователь регистрируется через форму, вводит данные (логин, имя, фамилию, email, пароль и пр.).

Шаги:

1. Валидация входных данных:

— Проверяется корректность email, уникальность username/email.

2. Хеширование пароля:

— Пароль преобразуется в безопасный хэш.

3. INSERT в таблицу `users`:

— Вставляется новая запись со значениями:

- username, firstname, lastname, gender, age, email

- password_hash (хэшированное значение)

- role по умолчанию: "user"

- is_active = TRUE, is_social = FALSE (если регистрация не через соцсеть)

- created_at и updated_at = NOW()

4. Возврат результата:

— Клиенту возвращается подтверждение регистрации (без передачи хэшированного пароля).

---

2. Сценарий: Авторизация (логин) пользователя

Описание:

Пользователь вводит свои учетные данные для входа в систему.

Шаги:

1. Поиск пользователя:

— Выполняется SELECT из таблицы users по username или email.

2. Проверка пароля:

— Сравнивается введенный пароль с сохранённым password_hash.

3. (Опционально) Обновление метаданных:

— Если необходимо, обновляется поле updated_at или записывается факт входа в отдельный лог.

4. Возврат результата:

— При успешной авторизации возвращается токен (например, JWT) и данные пользователя.

---

3. Сценарий: Создание мероприятия

Описание:

Организатор (пользователь с ролью organizer или admin) создаёт новое мероприятие.

Шаги:

1. Валидация прав пользователя:

— Проверяется, имеет ли пользователь права для создания мероприятия.

2. Валидация входных данных:

— Проверяются корректность даты, наличие всех обязательных полей (название, описание, место проведения, цена и т.п.).

3. INSERT в таблицу `events`:

— Создаётся запись с полями:

- title, description, date_time, location, category_id

- organizer_id – идентификатор текущего пользователя

- price, total_tickets

- status по умолчанию: "scheduled"

- is_verified – если предусмотрена модерация, выставляется FALSE или NULL

- created_at и updated_at = NOW()

4. Возврат результата:

— Возвращаются данные созданного мероприятия.

---

4. Сценарий: Обновление мероприятия

Описание:

Организатор или администратор вносит изменения в данные уже созданного мероприятия.

Шаги:

1. Проверка существования мероприятия:

— Выполняется SELECT по id мероприятия.

2. Валидация прав пользователя:

— Проверяется, имеет ли пользователь право обновлять данное мероприятие.

3. UPDATE в таблице `events`:

— Обновляются только переданные поля (например, title, price, date_time и т.д.).

— Поле updated_at обновляется на текущее время.

4. Возврат результата:

— Возвращается обновлённая информация о мероприятии.

---

5. Сценарий: Удаление мероприятия

Описание:

Организатор или администратор удаляет мероприятие, что может привести к каскадному удалению связанных записей.

Шаги:

1. Проверка существования мероприятия:

— Находим запись в events по id.

2. Проверка прав:

— Убедиться, что пользователь имеет право удалить мероприятие.

3. DELETE из таблицы `events`:

— Удаляется запись по id.

— Благодаря каскадным связям (например, в tickets, event_groups) связанные записи также удаляются автоматически.

4. (Опционально) Запись в `audit_logs`:

— Добавляется запись об удалении для аудита (указывается admin_id, действие, целевая таблица и target_id).

5. Возврат результата:

— Клиент получает подтверждение удаления.

---

6. Сценарий: Создание заказа

Описание:

Пользователь оформляет заказ на покупку билетов.

Шаги:

1. Валидация входных данных:

— Проверяется, что запрошенное количество билетов доступно (сравнение с total_tickets мероприятия).

2. Расчёт итоговой суммы:

— Итоговая сумма = сумма по каждому мероприятию: price * quantity.

3. INSERT в таблицу `orders`:

— Создаётся заказ с полями:

- user_id, total_amount,

- status = "pending",

- stripe_payment_id = NULL (до оплаты)

- created_at и updated_at = NOW()

4. Возврат результата:

— Возвращается идентификатор заказа, который далее используется для генерации билетов.

---

7. Сценарий: Генерация билетов

Описание:

После успешной оплаты заказа для каждой позиции создаются билеты.

Шаги:

1. Проверка заказа:

— Выполняется SELECT по orders с указанным order_id.

2. Генерация уникального кода билета:

— Для каждого билета генерируется уникальное значение (ticket_code), которое может быть UUID или иным.

3. INSERT в таблицу `tickets`:

— Создаются записи для каждого билета с полями:

- order_id, event_id

- ticket_code, qr_code_data (содержит зашифрованную информацию)

- is_used = FALSE, used_at = NULL

- created_at = NOW()

4. Возврат результата:

— Клиент получает информацию о созданных билетах.

8. Сценарий: Использование (сканирование) билета

Описание:

При входе на мероприятие сканируется QR-код билета.

Шаги:

1. Поиск билета:

— Выполняется поиск в таблице tickets по ticket_code.

2. Валидация использования:

— Проверяется, что билет существует и is_used = FALSE.

3. UPDATE в таблице `tickets`:

— Если билет действителен, обновляется запись:

- is_used устанавливается в TRUE

- used_at записывается текущее время (NOW())

4. Возврат результата:

— Возвращается сообщение об успешном использовании билета. Если билет уже использован — возвращается ошибка.

9. Сценарий: Оставление отзыва на мероприятие

Описание:

Пользователь оставляет отзыв о посещённом мероприятии.

Шаги:

1. Проверка участия:

— (Опционально) проверяется, что пользователь действительно посещал мероприятие.

2. INSERT в таблицу `reviews`:

— Создаётся запись с полями:

- event_id, user_id, rating, comment

- is_moderated = FALSE (если требуется модерация)

- created_at и updated_at = NOW()

3. Возврат результата:

— Возвращается информация об оставленном отзыве.

10. Сценарий: Создание группы для совместного посещения мероприятия

Описание:

Пользователь создаёт группу для поиска компании на мероприятие.

Шаги:

1. Валидация входных данных:

— Проверяется наличие мероприятия и корректность информации о группе.

2. INSERT в таблицу `event_groups`:

— Создаётся запись с полями:

- event_id, creator_id, title, description,

- status = "active",

- created_at и updated_at = NOW()

3. Автоматическое добавление участника:

— (Опционально) автоматически добавляется создатель группы в таблицу group_members с joined_at = NOW().

4. Возврат результата:

— Возвращаются данные созданной группы.

11. Сценарий: Присоединение к группе

Описание:

Пользователь присоединяется к уже существующей группе для совместного посещения мероприятия.

Шаги:

1. Проверка существования группы:

— Выполняется поиск группы по id.

2. Проверка уникальности записи:

— Проверяется, что для комбинации group_id и user_id запись отсутствует (уникальный индекс).

3. INSERT в таблицу `group_members`:

— Создаётся запись с полями:

- group_id, user_id,

- joined_at = NOW()

4. Возврат результата:

— Клиент получает подтверждение успешного присоединения.

12. Сценарий: Отправка сообщения в чате группы

Описание:

Участник группы отправляет сообщение в чат для обсуждения деталей мероприятия.

Шаги:

1. Проверка членства в группе:

— Проверяется наличие записи в таблице group_members для данного user_id и group_id.

2. INSERT в таблицу `group_chat_messages`:

— Добавляется новое сообщение с полями:

- group_id, user_id, message

- created_at = NOW()

3. Возврат результата:

— Клиенту возвращается подтверждение отправки сообщения.

13. Сценарий: Оформление подписки на категорию мероприятий

Описание:

Пользователь подписывается на уведомления по выбранной категории мероприятий.

Шаги:

1. Проверка существования подписки:

— Выполняется проверка, существует ли уже запись с данной комбинацией user_id и category_id.

2. INSERT в таблицу `user_subscriptions`:

— Если подписка отсутствует, создаётся новая запись с полями:

- user_id, category_id, frequency (например, "weekly")

- created_at = NOW()

3. Возврат результата:

— Клиент получает подтверждение оформления подписки.

---

14. Сценарий: Отправка уведомления пользователю

Описание:

Система отправляет уведомление (например, email или через Telegram) пользователю.

Шаги:

1. INSERT в таблицу `notifications`:

— Создаётся запись с полями:

- user_id, type (например, "email"), message

- is_sent = FALSE, sent_at = NULL

- created_at = NOW()

2. Фоновая отправка уведомления:

— Сервис уведомлений обрабатывает запись, отправляет сообщение.

3. UPDATE записи уведомления:

— После успешной отправки обновляется запись:

- is_sent = TRUE, sent_at = NOW()

4. Возврат результата:

— Клиент получает информацию о статусе уведомления (если предусмотрено).

15. Сценарий: Запись действий в журнал аудита

Описание:

При выполнении административных действий (например, удаление мероприятия) создаётся запись в аудите.

Шаги:

1. Определение действия:

— При выполнении критического действия (удаление, блокировка и т.п.) собираются детали:

- ID администратора, описание действия, целевая таблица, ID затронутой записи, дополнительные детали.

2. INSERT в таблицу `audit_logs`:

— Создаётся запись с полями:

- admin_id, action, target_table, target_id, details

- created_at = NOW()

3. Возврат результата:

— Запись в журнал аудита используется для дальнейшего анализа и контроля.