БД и эндпоинты

```
CREATE TABLE users (
    id             SERIAL PRIMARY KEY,
    username       VARCHAR(50)  NOT NULL UNIQUE,
    email          VARCHAR(100) NOT NULL UNIQUE,
    password_hash  VARCHAR(255) NOT NULL,
    confirmed      BOOLEAN      NOT NULL DEFAULT FALSE,
    created_at     TIMESTAMP    NOT NULL DEFAULT NOW(),
    updated_at     TIMESTAMP    NOT NULL DEFAULT NOW()
);

CREATE TABLE email_confirmations (
    id                SERIAL PRIMARY KEY,
    user_id           INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    confirmation_code VARCHAR(6) NOT NULL,
    expires_at        TIMESTAMP  NOT NULL,
    created_at        TIMESTAMP  NOT NULL DEFAULT NOW()
);

CREATE INDEX idx_email_confirmations_user_code
  ON email_confirmations (user_id, confirmation_code);
```

Метод	Путь	Bearer	Входные параметры	Возможные ответы	Описание	Логика работы
POST	`/api/v1/auth/register`	Нет	Body (JSON): `{"username": string, "email": string, "password": string}`	• **200 OK** → `{"message":"Confirmation code sent"}` • **400 Bad Request** → ошибки валидации • **409 Conflict** → пользователь/email уже существует	Регистрирует пользователя и отправляет код подтверждения на e-mail	• Принимает `username`, `email`, `password`. • Создаёт запись в **users** (`confirmed = false` и хэширует пароль конечно). • Генерирует 6-тизначный (в целом количество знаков не особо важно) `confirmation_code`, вставляет строку в **email_confirmations** (код со сроком `expires_at`, нужно продумать храним ли мы коды временно или чистим). • Отправляем письмо с кодом. • Ответ 200 OK `{"message":"Confirmation code sent"}` или ошибки, указанные в `Возможные ответы`
POST	`/api/v1/auth/register/confirm`	Нет	Body (JSON): `{"email": string,`	• **200 OK** → `{"id":"...","username":"...","email":"..."}`	Подтверждает регистрацию по коду,	• Принимает `email`, `confirmationCode`.

Метод	Путь	Bearer	Входные параметры	Возможные ответы	Описание	Логика работы
			`"confirmationCode": string}`	• **400 Bad Request** → неверный или просроченный код • **404 Not Found** → email не найден	активирует аккаунт. После этого фронтенд перенаправляет на страницу логина	• По `email` находим `id` в таблице **users**. Если не найдено, то 404 Not Found. Если найдено, то идем дальше. • Ищем в **email_confirmations** строку, где `user_id == id(полученным, на предыдущем шаге)`, `confirmation_code == confiramtionCode(переданный пользователем)` и `expires_at > NOW()` • При успехе — удаляет эту строку из **email_confirmations** и ставит `users.confirmed = true`. • Возвращает данные пользователя `{"id","username","email"}`. • Фронт перенаправляет на страницу логина.
POST	`/api/v1/auth/login`	Нет	Body (JSON): `{"username": string, "password": string}`	• **200 OK** → `{"access_token":"...","refresh_token":"...","user": {"id":"...","username":"...","email":"..."}}` • **400 Bad Request** → некорректный формат запроса • **401 Unauthorized** → неверные учётные данные	Аутентифицирует пользователя, возвращает токены и данные пользователя	• Принимает `username`, `password`. • Проверяет хэш пароля, а также `users.confirmed = true`. • Выдаёт `access_token`, `refresh_token` и вложенный `user`-объект.
GET	`/api/v1/auth/me`	Да	Header: `Authorization: Bearer <access_token>`	• **200 OK** → `{"id":"...","username":"...","email":"..."}` • **401 Unauthorized** → отсутствует или недействителен токен	Возвращает данные текущего аутентифицированного пользователя	• Требует заголовок `Authorization: Bearer <access_token>`. • Декодирует JWT, достаёт `user_id`, возвращает `{"id","username","email"}` (или 401 при недействительном токене).