1. **Код API**

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

from flask\_marshmallow import Marshmallow

from flask\_jwt\_extended import JWTManager, create\_access\_token, jwt\_required, get\_jwt\_identity, get\_jwt, verify\_jwt\_in\_request

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from datetime import datetime

# Инициализация приложения

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

    return "Welcome to the Messenger API!"

# Настройки базы данных

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'postgresql://user:password@localhost:5432/messenger'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

# Настройки JWT

app.config['JWT\_SECRET\_KEY'] = 'your\_secret\_key\_here'  # Укажи свой секретный ключ

app.config['JWT\_ACCESS\_TOKEN\_EXPIRES'] = 360000  # Время жизни токена в секундах

jwt = JWTManager(app)

# Инициализация базы данных и Marshmallow и JWT

db = SQLAlchemy(app)

ma = Marshmallow(app)

jwt = JWTManager(app)

# Черный список для токенов

blacklist = set()

@app.before\_request

def check\_token\_in\_blacklist():

    token = request.headers.get('Authorization', None)

    if token:

        token = token.split(" ")[1]  # Извлекаем токен из заголовка Authorization

        try:

            verify\_jwt\_in\_request()

            # Извлекаем данные токена

            decoded\_token = get\_jwt()

            jti = decoded\_token.get('jti')

            # Проверяем, есть ли токен в черном списке

            if jti in blacklist:

                return jsonify({"description": "The token has been revoked.", "error": "token\_revoked"}), 401

        except Exception as e:

            return jsonify({"description": str(e), "error": "invalid\_token"}), 401

@jwt.revoked\_token\_loader

def revoked\_token\_callback(jwt\_header, jwt\_payload):

    return jsonify(

        {"description": "The token has been revoked.", "error": "token\_revoked"}

    ), 401

# Модели

class User(db.Model):

    \_\_tablename\_\_ = 'users'

    id = db.Column(db.Integer, primary\_key=True)

    username = db.Column(db.String(80), unique=True, nullable=False)

    email = db.Column(db.String(120), unique=True, nullable=False)

    password\_hash = db.Column(db.String(128))

    def \_\_repr\_\_(self):

        return f'<User {self.username}>'

class Chat(db.Model):

    \_\_tablename\_\_ = 'chats'

    id = db.Column(db.Integer, primary\_key=True)

    name = db.Column(db.String, nullable=False)

    def \_\_repr\_\_(self):

        return f'<Chat {self.name}>'

class Message(db.Model):

    \_\_tablename\_\_ = 'messages'

    id = db.Column(db.Integer, primary\_key=True)

    chat\_id = db.Column(db.Integer, db.ForeignKey('chats.id'))

    sender\_id = db.Column(db.Integer, db.ForeignKey('users.id'))

    content = db.Column(db.String, nullable=False)

    timestamp = db.Column(db.DateTime, default=datetime.utcnow)

    def \_\_repr\_\_(self):

        return f'<Message {self.content}>'

# Схемы для сериализации

class UserSchema(ma.SQLAlchemyAutoSchema):

    class Meta:

        model = User

class ChatSchema(ma.SQLAlchemyAutoSchema):

    class Meta:

        model = Chat

class MessageSchema(ma.SQLAlchemyAutoSchema):

    class Meta:

        model = Message

# Регистрация пользователя

@app.route('/register', methods=['POST'])

def register():

    username = request.json.get('username')

    email = request.json.get('email')

    password = request.json.get('password')

    if not username or not email or not password:

        return jsonify({"error": "Missing data"}), 400

    if User.query.filter((User.username == username) | (User.email == email)).first():

        return jsonify({"error": "User already exists"}), 400

    hashed\_password = generate\_password\_hash(password)

    new\_user = User(username=username, email=email, password\_hash=hashed\_password)

    db.session.add(new\_user)

    db.session.commit()

    return jsonify({"message": "User created successfully"}), 201

# Вход пользователя

@app.route('/login', methods=['POST'])

def login():

    username = request.json.get('username')

    password = request.json.get('password')

    user = User.query.filter\_by(username=username).first()

    if not user or not check\_password\_hash(user.password\_hash, password):

        return jsonify({"error": "Invalid username or password"}), 401

    # Создание токена с 'jti'

    access\_token = create\_access\_token(identity=user.id)

    print(f"Generated JWT: {access\_token}")  # Log token for debugging

    return jsonify(access\_token=access\_token)

# Разлогинивание (удаление токена из черного списка)

@app.route('/logout', methods=['POST'])

@jwt\_required()

def logout():

    # Получаем текущий токен

    token = get\_jwt()

    # Извлекаем 'jti' из токена

    jti = token['jti']

    # Добавляем 'jti' в черный список

    blacklist.add(jti)

    return jsonify({"message": "Successfully logged out"}), 200

# Смена пароля

@app.route('/change-password', methods=['POST'])

@jwt\_required()

def change\_password():

    current\_user\_id = get\_jwt\_identity()

    user = User.query.get(current\_user\_id)

    old\_password = request.json.get('old\_password')

    new\_password = request.json.get('new\_password')

    if not check\_password\_hash(user.password\_hash, old\_password):

        return jsonify({"error": "Old password is incorrect"}), 400

    user.password\_hash = generate\_password\_hash(new\_password)

    db.session.commit()

    return jsonify({"message": "Password changed successfully"})

# Пример защищённого эндпоинта

@app.route('/protected', methods=['GET'])

@jwt\_required()

def protected():

    current\_user\_id = get\_jwt\_identity()

    user = User.query.get(current\_user\_id)

    return jsonify(logged\_in\_as=user.username)

# Получение всех пользователей

@app.route('/users', methods=['GET'])

@jwt\_required()

def get\_users():

    users = User.query.all()

    user\_schema = UserSchema(many=True)

    return jsonify(user\_schema.dump(users))

# Получение одного пользователя по ID

@app.route('/users/<int:id>', methods=['GET'])

@jwt\_required()

def get\_user(id):

    user = User.query.get(id)

    if user is None:

        return jsonify({"error": "User not found"}), 404

    user\_schema = UserSchema()

    return jsonify(user\_schema.dump(user))

# Создание нового чата

@app.route('/chats', methods=['POST'])

@jwt\_required()

def add\_chat():

    name = request.json.get('name')

    new\_chat = Chat(name=name)

    db.session.add(new\_chat)

    db.session.commit()

    chat\_schema = ChatSchema()

    return jsonify(chat\_schema.dump(new\_chat)), 201

# Получение всех сообщений в чате

@app.route('/chats/<int:chat\_id>/messages', methods=['GET'])

@jwt\_required()

def get\_messages(chat\_id):

    messages = Message.query.filter\_by(chat\_id=chat\_id).all()

    message\_schema = MessageSchema(many=True)

    return jsonify(message\_schema.dump(messages))

# Отправка нового сообщения

@app.route('/messages', methods=['POST'])

@jwt\_required()

def add\_message():

    chat\_id = request.json.get('chat\_id')

    sender\_id = get\_jwt\_identity()

    content = request.json.get('content')

    new\_message = Message(chat\_id=chat\_id, sender\_id=sender\_id, content=content)

    db.session.add(new\_message)

    db.session.commit()

    message\_schema = MessageSchema()

    return jsonify(message\_schema.dump(new\_message)), 201

# Удаление чата

@app.route('/chats/<int:chat\_id>', methods=['DELETE'])

@jwt\_required()

def delete\_chat(chat\_id):

    chat = Chat.query.get(chat\_id)

    if not chat:

        return jsonify({"error": "Chat not found"}), 404

    Message.query.filter\_by(chat\_id=chat\_id).delete()

    db.session.delete(chat)

    db.session.commit()

    return jsonify({"message": f"Chat with ID {chat\_id} and all its messages deleted successfully."}), 200

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

1. **Модель пользователя**

 id: Уникальный идентификатор пользователя (первичный ключ).

 username: Имя пользователя (уникальное и обязательное).

 email: Электронная почта пользователя (уникальная и обязательная).

 password\_hash: Хеш пароля пользователя, который хранится в базе данных вместо самого пароля.

1. **JWT Аутентификация**

**JWT Настройки:**

* JWT\_SECRET\_KEY: Секретный ключ для создания и проверки токенов. Этот ключ должен быть уникальным и секретным.
* JWT\_ACCESS\_TOKEN\_EXPIRES: Время жизни токена (в данном случае 100 часов).
* JWTManager: Инициализация менеджера JWT, который будет использоваться для создания и проверки токенов.

**Черный список для токенов (для выхода):**

Токены, которые были отозваны (например, при выходе из системы), сохраняются в blacklist. Функция проверяет, находится ли токен в черном списке.

**Роуты для аутентификации:**

* Регистрация:

При регистрации создается новый пользователь с хешированным паролем.

* Вход (Login):

Если имя пользователя и пароль правильные, создается JWT токен, который отправляется клиенту.

* Выход (Logout):

При выходе текущий JWT токен добавляется в черный список, что делает его недействительным.

**Защищенные эндпоинты:**

Для того, чтобы доступ к определенным эндпоинтам был только для авторизованных пользователей, используется декоратор @jwt\_required(). Например:

@app.route('/protected', methods=['GET'])

@jwt\_required()

def protected():

current\_user\_id = get\_jwt\_identity() # Получаем ID текущего пользователя из токена

user = User.query.get(current\_user\_id)

return jsonify(logged\_in\_as=user.username)

1. **Примеры**

