1. **Код программы**

app.py

from flask import Flask, request, render\_template, redirect, url\_for, session, flash

from flask\_sqlalchemy import SQLAlchemy

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

from datetime import datetime

import os

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

app.secret\_key = 'your\_super\_secret\_key'

app.config['SQLALCHEMY\_DATABASE\_URI'] = os.getenv(

    'DATABASE\_URL',

    'postgresql://user:password@localhost:5432/messenger'  # fallback для локального запуска

)

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

db = SQLAlchemy(app)

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))

class Chat(db.Model):

    \_\_tablename\_\_ = 'chats'

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

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

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)

class ChatMember(db.Model):

    \_\_tablename\_\_ = 'chat\_members'

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

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

    user\_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

@app.route('/')

def home():

    return redirect(url\_for('login\_page'))

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

def register\_page():

    if request.method == 'POST':

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

        email = request.form.get('email')

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

        if not username or not email or not password:

            flash("Все поля обязательны для заполнения.")

            return render\_template('register.html')

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

            flash("Пользователь с таким логином или email уже существует.")

            return render\_template('register.html')

        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()

        flash("Успешная регистрация. Войдите в систему.")

        return redirect(url\_for('login\_page'))

    return render\_template('register.html')

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

def login\_page():

    if request.method == 'POST':

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

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

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

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

            flash("Неправильный логин или пароль.")

            return render\_template('login.html')

        session['user\_id'] = user.id

        return redirect(url\_for('chats\_page'))

    return render\_template('login.html')

@app.route('/logout-page')

def logout\_page():

    session.clear()

    return redirect(url\_for('login\_page'))

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

def chats\_page():

    user\_id = session.get('user\_id')

    if not user\_id:

        return redirect(url\_for('login\_page'))

    if request.method == 'POST':

        chat\_name = request.form.get('name')

        if chat\_name:

            new\_chat = Chat(name=chat\_name)

            db.session.add(new\_chat)

            db.session.commit()

            chat\_member = ChatMember(chat\_id=new\_chat.id, user\_id=user\_id)

            db.session.add(chat\_member)

            db.session.commit()

            initial\_message = Message(chat\_id=new\_chat.id, sender\_id=user\_id, content="Чат создан")

            db.session.add(initial\_message)

            db.session.commit()

            flash("Чат создан.")

        else:

            flash("Имя чата не задано.")

        return redirect(url\_for('chats\_page'))

    chats = (

        db.session.query(Chat)

        .join(ChatMember, Chat.id == ChatMember.chat\_id)

        .filter(ChatMember.user\_id == user\_id)

        .all()

    )

    return render\_template('chats.html', chats=chats)

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

def leave\_chat(chat\_id):

    user\_id = session.get('user\_id')

    if not user\_id:

        return redirect(url\_for('login\_page'))

    # Найдём пользователя

    user = User.query.get(user\_id)

    if not user:

        flash("Пользователь не найден.")

        return redirect(url\_for('chats\_page'))

    # Удалим участника из чата

    ChatMember.query.filter\_by(chat\_id=chat\_id, user\_id=user\_id).delete()

    db.session.commit()

    # Добавим сообщение о выходе пользователя

    leave\_message = Message(chat\_id=chat\_id, sender\_id=user\_id, content=f"{user.username} покинул чат")

    db.session.add(leave\_message)

    db.session.commit()

    # Проверим, остались ли участники

    remaining\_members = ChatMember.query.filter\_by(chat\_id=chat\_id).count()

    if remaining\_members == 0:

        # Удалим все сообщения

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

        # Удалим сам чат

        Chat.query.filter\_by(id=chat\_id).delete()

        db.session.commit()

        flash("Чат удалён, так как в нём не осталось участников.")

    else:

        flash("Вы покинули чат.")

    return redirect(url\_for('chats\_page'))

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

def view\_chat(chat\_id):

    user\_id = session.get('user\_id')

    if not user\_id:

        return redirect(url\_for('login\_page'))

    if request.method == 'POST':

        action = request.form.get('action')

        if action == 'send\_as\_self':

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

            if content:

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

                db.session.add(new\_message)

                db.session.commit()

        elif action == 'add\_user':

            new\_user\_id\_str = request.form.get('new\_user\_id')

            try:

                new\_user\_id = int(new\_user\_id\_str)

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

                if user:

                    exists = ChatMember.query.filter\_by(chat\_id=chat\_id, user\_id=new\_user\_id).first()

                    if not exists:

                        db.session.add(ChatMember(chat\_id=chat\_id, user\_id=new\_user\_id))

                        db.session.commit()

                        db.session.add(Message(chat\_id=chat\_id, sender\_id=user\_id, content=f"{user.username} добавлен"))

                        db.session.commit()

                        flash("Пользователь добавлен.")

                else:

                    flash("Пользователь не найден.")

            except:

                flash("Неверный ID пользователя.")

        return redirect(url\_for('view\_chat', chat\_id=chat\_id))

    messages = (

        db.session.query(Message, User)

        .join(User, Message.sender\_id == User.id)

        .filter(Message.chat\_id == chat\_id)

        .order\_by(Message.timestamp.asc())

        .all()

    )

    formatted = [

        {'timestamp': m.timestamp.strftime('%Y-%m-%d %H:%M:%S'), 'sender': u.username, 'content': m.content}

        for m, u in messages

    ]

    return render\_template('messages.html', messages=formatted, chat\_id=chat\_id)

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

    app.run(host="0.0.0.0", port=5000)

docker-compose.yml

services:

  db:

    image: postgres:latest

    container\_name: messenger\_db

    restart: always

    environment:

      POSTGRES\_DB: messenger

      POSTGRES\_USER: user

      POSTGRES\_PASSWORD: password

    ports:

      - "5432:5432"

    volumes:

      - pg\_data:/var/lib/postgresql/data

  web:

    build: .

    container\_name: messenger\_web

    restart: always

    depends\_on:

      - db

    ports:

      - "5431:5000"

    environment:

      - FLASK\_APP=app.py

      - FLASK\_RUN\_HOST=0.0.0.0

      - DATABASE\_URL=postgresql://user:password@db:5432/messenger

    volumes:

      - .:/app

volumes:

  pg\_data:

dockerfile

# Базовый образ Python

FROM python:3.10-slim

# Устанавливаем рабочую директорию внутри контейнера

WORKDIR /app

# Копируем зависимости

COPY requirements.txt .

# Устанавливаем зависимости

RUN pip install --no-cache-dir -r requirements.txt

# Копируем всё приложение внутрь контейнера

COPY . .

# Открываем порт (если нужно)

EXPOSE 5000

# Запуск приложения

CMD ["python", "app.py"]

1. **Структура контейнеризации и настройка окружения**

Приложение развернуто с использованием Docker Compose и состоит из двух основных сервисов:

|  |  |  |
| --- | --- | --- |
| **Сервис** | **Образ** | **Назначение** |
| db | postgres:latest | Хранение данных пользователей, чатов, сообщений |
| web | Кастомный (FROM python:3.10-slim) | Flask-приложение (веб-интерфейс) |

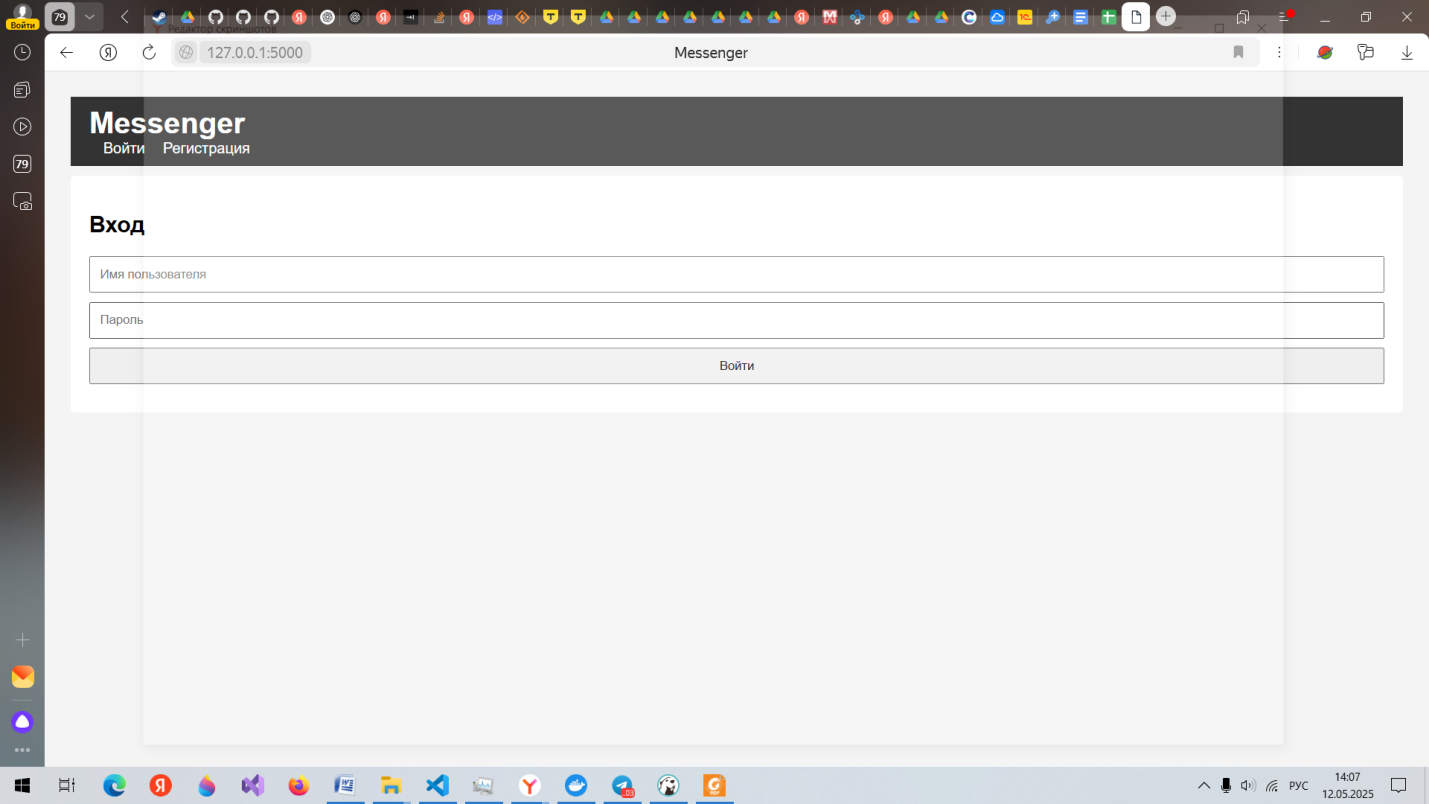
**Файлы:**

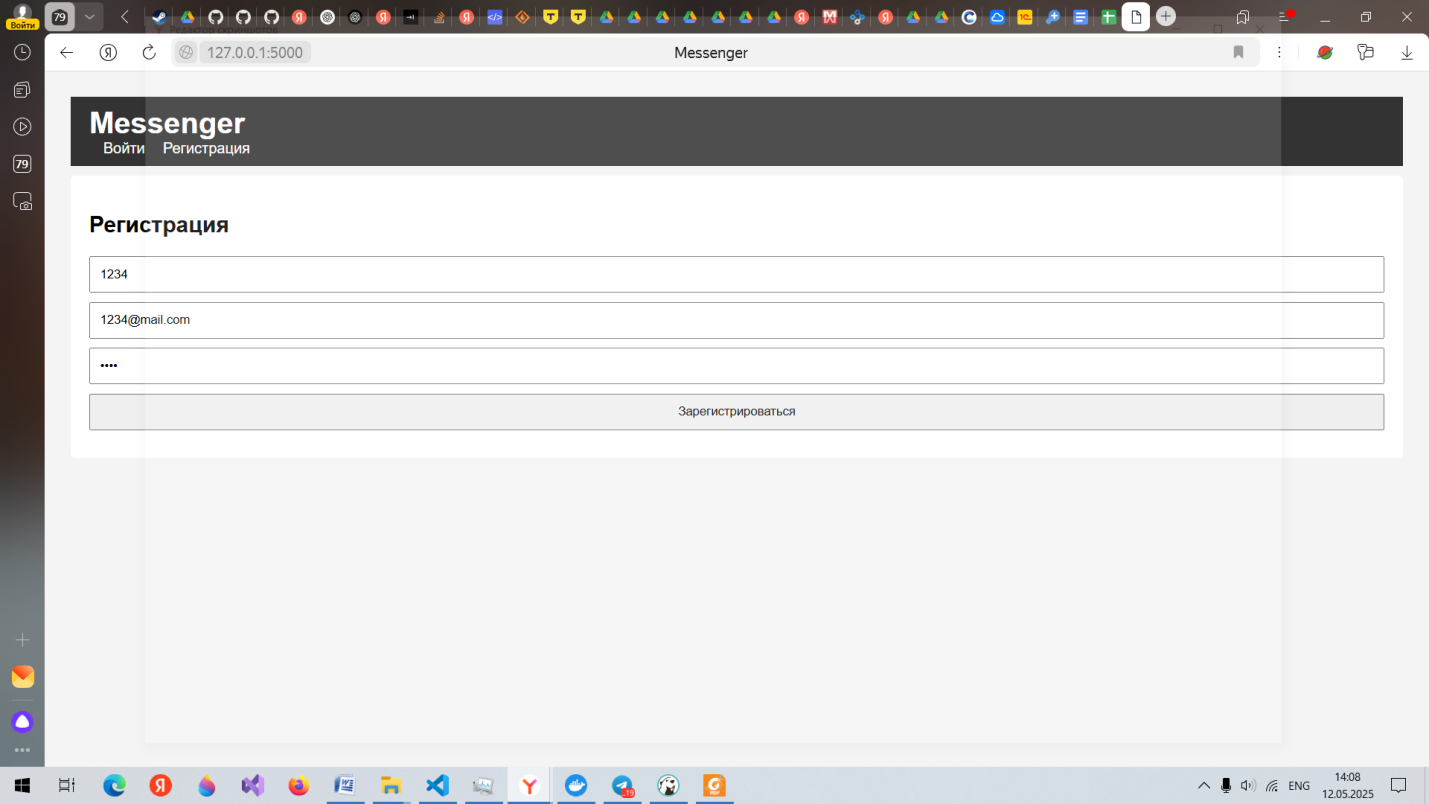
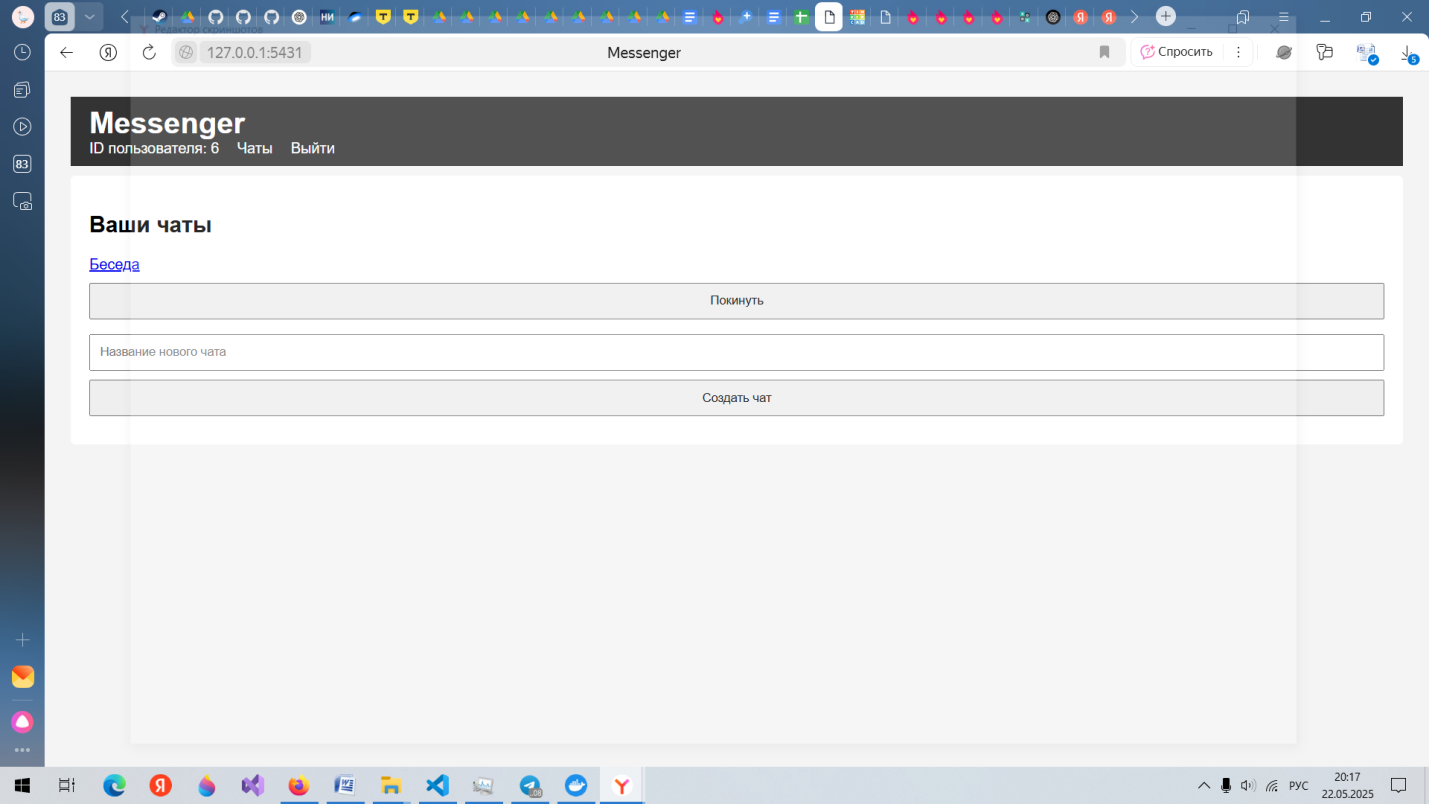
* docker-compose.yml: описывает оба сервиса, монтирует том для PostgreSQL.
* Dockerfile: создает образ Flask-приложения.
* requirements.txt: содержит зависимости (Flask, SQLAlchemy, psycopg2 и др.).

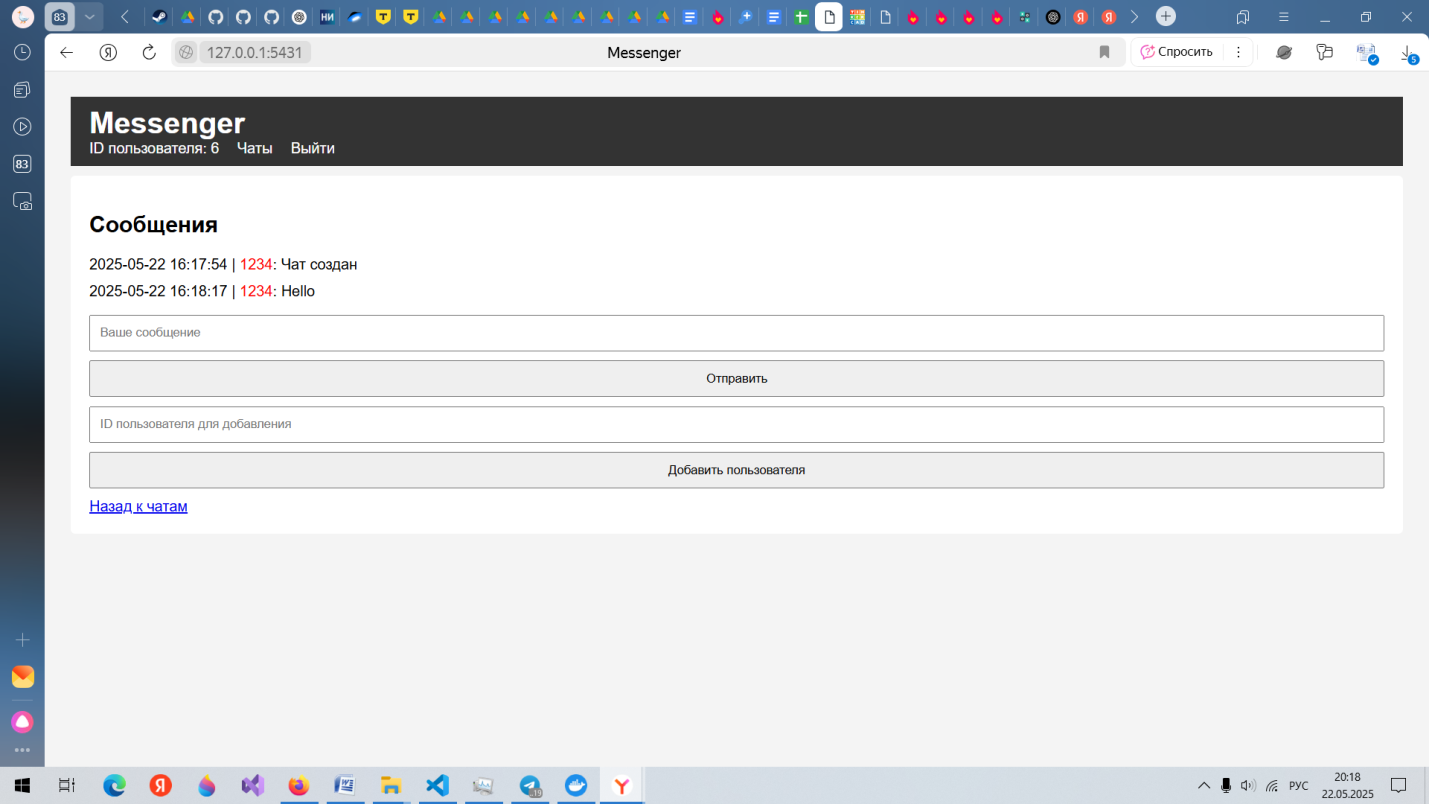
**Пример запуска:**

docker-compose up --build

1. **Скриншоты работающего приложения в контейнерах**





1. **Описание API (в формате таблицы)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Маршрут** | **Метод** | **Назначение** | **Аутентификация** |
| / | GET | Редирект на /login-page | нет |
| /register-page | GET/POST | Регистрация нового пользователя | нет |
| /login-page | GET/POST | Вход в систему | нет |
| /logout-page | GET | Выход из системы | нет |
| /chats-page | GET/POST | Просмотр чатов / создание нового | да |
| /chats/<chat\_id>/leave | POST | Покинуть чат | да |
| /chats/<chat\_id>/messages-page | GET/POST | Просмотр сообщений и взаимодействие | да |

### Архитектура разработанного приложения

**Основные компоненты:**

* Flask (Backend) — веб-приложение, обрабатывающее маршруты, формы и шаблоны.
* Jinja2 (Frontend) — серверная генерация HTML.
* PostgreSQL — СУБД для хранения данных.
* SQLAlchemy — ORM для работы с БД.

**Основные сущности:**

* User: хранит имя пользователя, email и хеш пароля.
* Chat: групповые чаты.
* Message: сообщения, привязанные к чатам.
* ChatMember: связующая таблица many-to-many между пользователями и чатами.

**Поток взаимодействия:**

1. Регистрация / логин.
2. Создание чатов, автоматическое добавление автора.
3. Отправка сообщений.
4. Добавление участников по ID.
5. Удаление чата при выходе последнего участника.