

Manual Completo: API Flask com Login, Cadastro e Reconhecimento Facial (Nível Iniciante)

Este manual foi criado para pessoas **sem experiência prévia em Python**. Cada passo explica o que está acontecendo e mostra os comandos separados para **Windows, macOS e Linux**.

1. Visão Geral

- **Objetivo:** construir uma API (programa que responde a pedidos pela internet) capaz de:
 - Criar usuários (cadastro)
 - Validar login com senha
 - Guardar informações pessoais
 - Cadastrar e reconhecer rostos em imagens
- **Tecnologias principais:** Python 3, Flask, SQLite3, biblioteca `face_recognition`
- **Arquitetura:** MVC (Model-View-Controller)

O que é MVC?

- **Model (Modelo):** conversa com o banco de dados.
 - **View (Visão):** decide como responder (JSON com mensagens e dados).
 - **Controller (Controlador):** recebe a requisição, usa o Model e devolve a View.
-

2. Pré-requisitos por Sistema Operacional

Sistema	Programas necessários	Como verificar
Windows	Python 3, Git (opcional), Visual Studio Build Tools (para dlib), CMake	Abrir PowerShell e rodar <code>py --version</code>
macOS	Python 3, Homebrew, CMake	Abrir Terminal e rodar <code>python3 --version</code>
Linux	Python 3, build-essential, CMake	Abrir Terminal e rodar <code>python3 --version</code>

Instalando Python

- **Windows:** baixar em [python.org/downloads](https://www.python.org/downloads/) (<https://www.python.org/downloads/>). Na instalação marque “Add Python to PATH”.
- **macOS:** o Python 3 já vem instalado (use `python3`). Se precisar, instale via [python.org](https://www.python.org/) (<https://www.python.org/>) ou `brew install python`.
- **Linux:** normalmente já instalado. Se não estiver, use o gerenciador da sua distro, por exemplo: `sudo apt install python3 python3-pip`.

Ferramentas extras para `face_recognition`

Sistema	Passos
Windows	1. Instale o CMake (https://cmake.org/download/) (adicione ao PATH). 2. Instale o Visual Studio Build Tools (componentes C++).

Sistema

Passos

macOS Rode `brew install cmake` (instale o Homebrew antes em <https://brew.sh> (<https://brew.sh>)).

Linux Rode `sudo apt install cmake build-essential libssl-dev` (Debian/Ubuntu).

3. Preparar a Pasta do Projeto

Escolha uma pasta para trabalhar (por exemplo, `Documentos`). Abra o terminal correto para seu sistema:

Sistema

Como abrir o terminal

Windows Abra o **PowerShell** e use `cd` até a pasta desejada.

macOS Abra o **Terminal** pelo Spotlight e use `cd`.

Linux Abra o terminal padrão e use `cd`.

Criar a pasta principal

Sistema

Comando

Windows `mkdir projeto_api` e depois `cd projeto_api`

macOS/Linux `mkdir -p projeto_api && cd projeto_api`

Para confirmar onde você está: `pwd` (macOS/Linux) ou `Get-Location` (Windows).

4. Criar Ambiente Virtual (recomendado)

Ambiente virtual evita conflitos de versões.

Sistema

Criar ambiente

Ativar ambiente

Desativar

Windows `py -m venv venv` `venv\Scripts\activate` `deactivate`

macOS/Linux `python3 -m venv venv` `source venv/bin/activate` `deactivate`

Quando o ambiente está ativo, o terminal mostra algo como `(venv)` no começo da linha.

5. Dependências (`requirements.txt`)

Crie um arquivo chamado `requirements.txt` com o conteúdo:

```
Flask==2.3.3
Flask-CORS==4.0.0
Werkzeug==2.3.7
face-recognition==1.3.0
numpy==1.24.3
Pillow==10.0.1
bcrypt==4.0.1
PyJWT==2.8.0
```

Instalar dependências

Sistema

Comando

Windows `py -m pip install --upgrade pip`

`py -m pip install -r requirements.txt`

macOS/Linux `python3 -m pip install --upgrade pip`

Sistema

Comando

```
python3 -m pip install -r requirements.txt
```

Dica: Se aparecer erro dizendo que `dlib` não foi compilado, confirme se o CMake está instalado e reinicie o terminal.

6. Estrutura do Projeto

Organize as pastas dessa forma:

```
projeto_api/
├─ app/
│   ├─ __init__.py
│   ├─ config.py
│   ├─ models/
│   │   ├─ __init__.py
│   │   ├─ database.py
│   │   └─ user.py
│   ├─ views/
│   │   ├─ __init__.py
│   │   └─ responses.py
│   ├─ controllers/
│   │   ├─ __init__.py
│   │   ├─ auth_controller.py
│   │   └─ user_controller.py
│   ├─ services/
│   │   ├─ __init__.py
│   │   ├─ auth_service.py
│   │   └─ face_recognition_service.py
│   └─ utils/
│       ├─ __init__.py
│       └─ validators.py (opcional)
├─ uploads/
├─ database/
│   └─ app.db (criado automaticamente)
├─ requirements.txt
└─ run.py
```

Para criar as pastas:

- **Windows:** `mkdir app app\models app\views app\controllers app\services app\utils uploads database`
- **macOS/Linux:** `mkdir -p app/models app/views app/controllers app/services app/utils uploads database`

Crie os arquivos vazios com New-Item (Windows) ou touch (macOS/Linux).

7. Arquivos do Projeto (copie e cole)

7.1 app/config.py

```
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'troque-esta-chave-em-producao'

    BASE_DIR = os.path.dirname(os.path.dirname(__file__))
    DATABASE_PATH = os.path.join(BASE_DIR, 'database', 'app.db')
    UPLOAD_FOLDER = os.path.join(BASE_DIR, 'uploads')

    MAX_CONTENT_LENGTH = 16 * 1024 * 1024
    ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
    FACE_ENCODING_TOLERANCE = 0.6
```

7.2 app/models/database.py

```

import sqlite3
import os
from app.config import Config

class Database:
    def __init__(self):
        self.db_path = Config.DATABASE_PATH
        os.makedirs(os.path.dirname(self.db_path), exist_ok=True)

    def get_connection(self):
        conn = sqlite3.connect(self.db_path)
        conn.row_factory = sqlite3.Row
        return conn

    def init_db(self):
        conn = self.get_connection()
        cursor = conn.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                email TEXT UNIQUE NOT NULL,
                password_hash TEXT NOT NULL,
                full_name TEXT,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS user_info (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                phone TEXT,
                address TEXT,
                birth_date DATE,
                bio TEXT,
                FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS face_encodings (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                encoding BLOB NOT NULL,
                image_path TEXT,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
            )
        ''')

```

```
conn.commit()
conn.close()
print('Banco de dados inicializado com sucesso!')

db = Database()
```

7.3 app/models/user.py

```

import sqlite3
import bcrypt
from datetime import datetime
from app.models.database import db

class User:

    def __init__(self, username, email, password_hash, full_name=None, user_id=None):
        self.id = user_id
        self.username = username
        self.email = email
        self.password_hash = password_hash
        self.full_name = full_name

    @staticmethod
    def hash_password(password):
        return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

    def verify_password(self, password):
        return bcrypt.checkpw(password.encode('utf-8'), self.password_hash.encode('utf-8'))

    def save(self):
        conn = db.get_connection()
        cursor = conn.cursor()
        try:
            cursor.execute('''
                INSERT INTO users (username, email, password_hash, full_name)
                VALUES (?, ?, ?, ?)
            ''', (self.username, self.email, self.password_hash, self.full_name))
            conn.commit()
            self.id = cursor.lastrowid
            return self
        except sqlite3.IntegrityError:
            conn.rollback()
            raise ValueError('Usuário ou email já existe')
        finally:
            conn.close()

    @staticmethod
    def find_by_username(username):
        conn = db.get_connection()
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM users WHERE username = ?', (username,))
        row = cursor.fetchone()
        conn.close()
        if row:
            return User(row['username'], row['email'], row['password_hash'], row['full_name'], row['id'])
        return None

    @staticmethod
    def find_by_email(email):
        conn = db.get_connection()
        cursor = conn.cursor()

```

```

        cursor.execute('SELECT * FROM users WHERE email = ?', (email,))
        row = cursor.fetchone()
        conn.close()
        if row:
            return User(row['username'], row['email'], row['password_hash'], row['full_name'], row['id'])
        return None

    @staticmethod
    def find_by_id(user_id):
        conn = db.get_connection()
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM users WHERE id = ?', (user_id,))
        row = cursor.fetchone()
        conn.close()
        if row:
            return User(row['username'], row['email'], row['password_hash'], row['full_name'], row['id'])
        return None

    def to_dict(self):
        return {
            'id': self.id,
            'username': self.username,
            'email': self.email,
            'full_name': self.full_name
        }

    def save_user_info(self, phone=None, address=None, birth_date=None, bio=None):
        conn = db.get_connection()
        cursor = conn.cursor()
        cursor.execute('SELECT id FROM user_info WHERE user_id = ?', (self.id,))
        existing = cursor.fetchone()

        if existing:
            cursor.execute('''
                UPDATE user_info
                SET phone = ?, address = ?, birth_date = ?, bio = ?
                WHERE user_id = ?
            ''', (phone, address, birth_date, bio, self.id))
        else:
            cursor.execute('''
                INSERT INTO user_info (user_id, phone, address, birth_date, bio)
                VALUES (?, ?, ?, ?, ?)
            ''', (self.id, phone, address, birth_date, bio))

        conn.commit()
        conn.close()

    def get_user_info(self):
        conn = db.get_connection()
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM user_info WHERE user_id = ?', (self.id,))
        row = cursor.fetchone()

```

```
conn.close()
if row:
    return {
        'phone': row['phone'],
        'address': row['address'],
        'birth_date': row['birth_date'],
        'bio': row['bio']
    }
return None
```

7.4 app/views/responses.py

```
from flask import jsonify

class ResponseView:
    @staticmethod
    def success(data=None, message='Operação realizada com sucesso', status_code=200):
        return jsonify({'success': True, 'message': message, 'data': data}), status_code

    @staticmethod
    def error(message='Erro na operação', status_code=400, errors=None):
        return jsonify({'success': False, 'message': message, 'errors': errors}), status_code

    @staticmethod
    def unauthorized(message='Não autorizado'):
        return ResponseView.error(message, 401)

    @staticmethod
    def not_found(message='Recurso não encontrado'):
        return ResponseView.error(message, 404)
```

7.5 app/services/auth_service.py

```
import jwt
import datetime
from app.config import Config

class AuthService:
    @staticmethod
    def generate_token(user_id):
        payload = {
            'user_id': user_id,
            'exp': datetime.datetime.utcnow() + datetime.timedelta(days=7),
            'iat': datetime.datetime.utcnow()
        }
        return jwt.encode(payload, Config.SECRET_KEY, algorithm='HS256')

    @staticmethod
    def verify_token(token):
        try:
            payload = jwt.decode(token, Config.SECRET_KEY, algorithms=['HS256'])
            return payload.get('user_id')
        except jwt.ExpiredSignatureError:
            return None
        except jwt.InvalidTokenError:
            return None
```

7.6 app/services/face_recognition_service.py

```

import face_recognition
import numpy as np
import os
from PIL import Image
from app.config import Config
from app.models.database import db

class FaceRecognitionService:
    @staticmethod
    def allowed_file(filename):
        return '.' in filename and filename.rsplit('.', 1)[1].lower() in Config.ALLOWED_EXTENSIONS

    @staticmethod
    def save_uploaded_file(file, user_id):
        if not FaceRecognitionService.allowed_file(file.filename):
            raise ValueError('Tipo de arquivo não permitido')

        os.makedirs(Config.UPLOAD_FOLDER, exist_ok=True)
        filename = f'user_{user_id}_{os.urandom(8).hex()}.jpg'
        filepath = os.path.join(Config.UPLOAD_FOLDER, filename)

        image = Image.open(file)
        image = image.convert('RGB')
        image.save(filepath, 'JPEG')
        return filepath

    @staticmethod
    def get_face_encoding(image_path):
        image = face_recognition.load_image_file(image_path)
        face_encodings = face_recognition.face_encodings(image)
        if len(face_encodings) == 0:
            return None
        return face_encodings[0]

    @staticmethod
    def register_face(user_id, file):
        try:
            image_path = FaceRecognitionService.save_uploaded_file(file, user_id)
            encoding = FaceRecognitionService.get_face_encoding(image_path)
            if encoding is None:
                os.remove(image_path)
                return {'success': False, 'message': 'Nenhuma face detectada na imagem'}

            conn = db.get_connection()
            cursor = conn.cursor()
            encoding_bytes = encoding.tobytes()
            cursor.execute('''
                INSERT INTO face_encodings (user_id, encoding, image_path)
                VALUES (?, ?, ?)
            ''', (user_id, encoding_bytes, image_path))
            conn.commit()
            face_id = cursor.lastrowid

```

```

        conn.close()

        return {'success': True, 'data': {'face_id': face_id, 'image_path': image_path}}
    except Exception as e:
        return {'success': False, 'message': f'Erro ao cadastrar face: {str(e)}'}

    @staticmethod
    def recognize_face(file):
        temp_name = f"temp_{os.urandom(8).hex()}.jpg"
        temp_path = os.path.join(Config.UPLOAD_FOLDER, temp_name)
        os.makedirs(Config.UPLOAD_FOLDER, exist_ok=True)

        image = Image.open(file)
        image = image.convert('RGB')
        image.save(temp_path, 'JPEG')

        unknown_encoding = FaceRecognitionService.get_face_encoding(temp_path)
        os.remove(temp_path)

        if unknown_encoding is None:
            return {'success': False, 'message': 'Nenhuma face detectada na imagem'}

        conn = db.get_connection()
        cursor = conn.cursor()
        cursor.execute('''
            SELECT fe.id, fe.user_id, fe.encoding, u.username, u.full_name
            FROM face_encodings fe
            JOIN users u ON fe.user_id = u.id
        ''')
        faces = cursor.fetchall()
        conn.close()

        if len(faces) == 0:
            return {'success': False, 'message': 'Nenhuma face cadastrada no sistema'}

        for face in faces:
            known_encoding = np.frombuffer(face['encoding'], dtype=np.float64)
            distance = face_recognition.face_distance([known_encoding], unknown_encoding)[0]
            if distance < Config.FACE_ENCODING_TOLERANCE:
                return {
                    'success': True,
                    'data': {
                        'user_id': face['user_id'],
                        'username': face['username'],
                        'full_name': face['full_name'],
                        'confidence': float(1 - distance),
                        'face_id': face['id']
                    }
                }

        return {'success': False, 'message': 'Face não reconhecida'}

```

7.7 app/controllers/auth_controller.py

```

from flask import Blueprint, request
from functools import wraps
from app.models.user import User
from app.views.responses import ResponseView
from app.services.auth_service import AuthService

auth_bp = Blueprint('auth', __name__)

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')
        if not token:
            return ResponseView.unauthorized('Token de autenticação não fornecido')

        if token.startswith('Bearer '):
            token = token[7:]

        user_id = AuthService.verify_token(token)
        if not user_id:
            return ResponseView.unauthorized('Token inválido ou expirado')

        request.current_user_id = user_id
        return f(*args, **kwargs)

    return decorated

@auth_bp.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    if not data:
        return ResponseView.error('Dados não fornecidos')

    username = data.get('username')
    email = data.get('email')
    password = data.get('password')
    full_name = data.get('full_name')

    if not username or not email or not password:
        return ResponseView.error('Username, email e senha são obrigatórios')

    if User.find_by_username(username):
        return ResponseView.error('Username já está em uso', 409)
    if User.find_by_email(email):
        return ResponseView.error('Email já está em uso', 409)

    password_hash = User.hash_password(password)
    user = User(username=username, email=email, password_hash=password_hash, full_name=full_name)
    user.save()

    token = AuthService.generate_token(user.id)
    return ResponseView.success(

```

```

        data={'user': user.to_dict(), 'token': token},
        message='Usuário cadastrado com sucesso',
        status_code=201
    )

@auth_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    if not data:
        return ResponseView.error('Dados não fornecidos')

    username = data.get('username')
    password = data.get('password')

    if not username or not password:
        return ResponseView.error('Username e senha são obrigatórios')

    user = User.find_by_username(username)
    if not user or not user.verify_password(password):
        return ResponseView.unauthorized('Credenciais inválidas')

    token = AuthService.generate_token(user.id)
    return ResponseView.success(
        data={'user': user.to_dict(), 'token': token},
        message='Login realizado com sucesso'
    )

@auth_bp.route('/logout', methods=['POST'])
@token_required
def logout():
    # Em uma implementação mais completa, você poderia invalidar o token
    # Por enquanto, apenas confirmamos que o usuário está autenticado
    return ResponseView.success(
        message='Logout realizado com sucesso'
    )

```

7.8 app/controllers/user_controller.py

```

from flask import Blueprint, request
from app.models.user import User
from app.views.responses import ResponseView
from app.controllers.auth_controller import token_required
from app.services.face_recognition_service import FaceRecognitionService

user_bp = Blueprint('user', __name__)

@user_bp.route('/profile', methods=['GET'])
@token_required
def get_profile():
    user_id = request.current_user_id
    user = User.find_by_id(user_id)
    if not user:
        return ResponseView.not_found('Usuário não encontrado')

    data = user.to_dict()
    extra = user.get_user_info()
    if extra:
        data.update(extra)

    return ResponseView.success(data=data, message='Perfil recuperado com sucesso')

@user_bp.route('/info', methods=['POST', 'PUT'])
@token_required
def save_user_info():
    user_id = request.current_user_id
    user = User.find_by_id(user_id)
    if not user:
        return ResponseView.not_found('Usuário não encontrado')

    data = request.get_json()
    if not data:
        return ResponseView.error('Dados não fornecidos')

    user.save_user_info(
        phone=data.get('phone'),
        address=data.get('address'),
        birth_date=data.get('birth_date'),
        bio=data.get('bio')
    )

    return ResponseView.success(
        message='Informações salvas com sucesso',
        status_code=201 if request.method == 'POST' else 200
    )

@user_bp.route('/face/register', methods=['POST'])
@token_required
def register_face():
    user_id = request.current_user_id
    user = User.find_by_id(user_id)

```

```

if not user:
    return ResponseView.not_found('Usuário não encontrado')

if 'image' not in request.files:
    return ResponseView.error('Imagem não fornecida')

file = request.files['image']
if file.filename == '':
    return ResponseView.error('Nenhum arquivo selecionado')

result = FaceRecognitionService.register_face(user_id, file)
if result['success']:
    return ResponseView.success(result['data'], 'Face cadastrada com sucesso', 201)

return ResponseView.error(result['message'])

@user_bp.route('/face/recognize', methods=['POST'])
def recognize_face():
    if 'image' not in request.files:
        return ResponseView.error('Imagem não fornecida')

    file = request.files['image']
    if file.filename == '':
        return ResponseView.error('Nenhum arquivo selecionado')

    result = FaceRecognitionService.recognize_face(file)
    if result['success']:
        return ResponseView.success(result['data'], 'Face reconhecida com sucesso')

    return ResponseView.error(result['message'], 404)

```

7.9 app/__init__.py

```

from flask import Flask
from flask_cors import CORS
from app.config import Config
from app.models.database import db
from app.controllers.auth_controller import auth_bp
from app.controllers.user_controller import user_bp

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)
    CORS(app)

    db.init_db()

    app.register_blueprint(auth_bp, url_prefix='/api/auth')
    app.register_blueprint(user_bp, url_prefix='/api/user')

    @app.route('/')
    def index():
        return {
            'message': 'API Flask - Login, Cadastro e Reconhecimento Facial',
            'version': '1.0.0',
            'endpoints': {
                'auth': {
                    'register': 'POST /api/auth/register',
                    'login': 'POST /api/auth/login',
                    'logout': 'POST /api/auth/logout'
                },
                'face': {
                    'register': 'POST /api/user/face/register',
                    'recognize': 'POST /api/user/face/recognize'
                }
            }
        }

    return app

```

7.10 run.py

```

from app import create_app

app = create_app()

if __name__ == '__main__':
    print('=' * 50)
    print('API Flask iniciando...')
    print('=' * 50)
    app.run(debug=True, host='0.0.0.0', port=5000)

```

8. Iniciando o Servidor

Ative o ambiente virtual (passo 4) e rode:

Sistema	Comando
Windows	<code>py run.py</code>
macOS/Linux	<code>python3 run.py</code>

Se tudo der certo, verá a mensagem indicando que a API está rodando em `http://localhost:5000`.

Para parar o servidor: `Ctrl + C` (Windows/Linux) ou `Ctrl + C / ⌘ + .` (macOS, dependendo do terminal).

8.5 Documentação Completa dos Endpoints

Esta seção documenta todos os endpoints da API com exemplos de requisição e resposta.

Endpoints de Autenticação

1. Cadastro de Usuário

POST `/api/auth/register`

Descrição: Cria um novo usuário no sistema.

Body (JSON):

```
{
  "username": "joao_silva",
  "email": "joao@example.com",
  "password": "senha123",
  "full_name": "João Silva"
}
```

Campos:

- `username` (obrigatório): Nome de usuário único
- `email` (obrigatório): Email válido e único
- `password` (obrigatório): Senha do usuário
- `full_name` (opcional): Nome completo

Resposta de Sucesso (201):

```
{
  "success": true,
  "message": "Usuário cadastrado com sucesso",
  "data": {
    "user": {
      "id": 1,
      "username": "joao_silva",
      "email": "joao@example.com",
      "full_name": "João Silva"
    },
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2Vybm91dCI6ImR1b3VudCIsImV4cCI6MTY0MjUwMDAwfQ.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2Vybm91dCI6ImR1b3VudCIsImV4cCI6MTY0MjUwMDAwfQ."
  }
}
```

Resposta de Erro (409):

```
{
  "success": false,
  "message": "Username já está em uso"
}
```

2. Login

POST /api/auth/login

Descrição: Autentica um usuário e retorna um token de acesso.

Body (JSON):

```
{
  "username": "joao_silva",
  "password": "senha123"
}
```

Campos:

- `username` (obrigatório): Nome de usuário
- `password` (obrigatório): Senha do usuário

Resposta de Sucesso (200):

```
{  
  "success": true,  
  "message": "Login realizado com sucesso",  
  "data": {  
    "user": {  
      "id": 1,  
      "username": "joao_silva",  
      "email": "joao@example.com",  
      "full_name": "João Silva"  
    },  
    "token": "eyJ0eXAiOiJKVlQiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmFtZSI6ImNyeSIsImVtYWkiOiJkbWlnaGVkZW8yMDYwLTAxLTA1IiwiaWF0IjoxNDQ1MDE0ODAuOTAwfQ==<div></div>"
```

Resposta de Erro (401):

```
{
  "success": false,
  "message": "Credenciais inválidas"
}
```

3 Logout

```
POST /api/auth/logout
```

Descrição: Encerra a sessão do usuário (requer autenticação).

Headers:

Authorization: Bearer SEU_TOKEN_AQUI

Body: Não requer body.

Resposta de Sucesso (200):

```
{
  "success": true,
  "message": "Logout realizado com sucesso",
  "data": null
}
```

Resposta de Erro (401):

```
{
  "success": false,
  "message": "Token de autenticação não fornecido"
}
```

Endpoints de Reconhecimento Facial

4. Cadastrar Face

POST /api/user/face/register

Descrição: Cadastra uma imagem facial do usuário autenticado para reconhecimento futuro.

Headers:

```
Authorization: Bearer SEU_TOKEN_AQUI
```

Body (form-data):

- **image** (obrigatório): Arquivo de imagem (JPG, PNG, JPEG, GIF)

Exemplo de requisição:

- **Campo:** image
- **Tipo:** File
- **Valor:** [Selecione uma imagem com rosto visível]

Resposta de Sucesso (201):

```
{
  "success": true,
  "message": "Face cadastrada com sucesso",
  "data": {
    "face_id": 1,
    "image_path": "uploads/user_1_a3f5b2c1.jpg"
  }
}
```

Resposta de Erro (400):

```
{
  "success": false,
  "message": "Nenhuma face detectada na imagem"
}
```

5. Reconhecer Face

POST /api/user/face/recognize

Descrição: Reconhece uma face em uma imagem comparando com faces cadastradas (não requer autenticação).

Body (form-data):

- **image** (obrigatório): Arquivo de imagem (JPG, PNG, JPEG, GIF)

Exemplo de requisição:

- **Campo:** image
- **Tipo:** File
- **Valor:** [Selecione uma imagem com rosto visível]

Resposta de Sucesso (200):

```
{
  "success": true,
  "message": "Face reconhecida com sucesso",
  "data": {
    "user_id": 1,
    "username": "joao_silva",
    "full_name": "João Silva",
    "confidence": 0.95,
    "face_id": 1
  }
}
```

Resposta de Erro (404):

```
{
  "success": false,
  "message": "Face não reconhecida"
}
```

Resumo dos Endpoints

Método	Endpoint	Descrição	Autenticação
POST	/api/auth/register	Cadastrar novo usuário	<input type="checkbox"/> Não
POST	/api/auth/login	Fazer login	<input type="checkbox"/> Não
POST	/api/auth/logout	Fazer logout	<input type="checkbox"/> Sim
POST	/api/user/face/register	Cadastrar face	<input type="checkbox"/> Sim
POST	/api/user/face/recognize	Reconhecer face	<input type="checkbox"/> Não

Legenda:

- ☐ Sim = Requer token no header `Authorization: Bearer TOKEN`
- ☐ Não = Não requer autenticação

9. Testando os Endpoints

Você pode usar Postman/Insomnia ou o `curl`. Lembre-se de substituir `TOKEN_AQUI` pelo token retornado no login.

9.1 Cadastro

Windows (PowerShell)

```
curl -X POST http://localhost:5000/api/auth/register ^
-H "Content-Type: application/json" ^
-d "{\"username\":\"joao\",\"email\":\"joao@example.com\",\"password\":\"senha123\"}"
```

macOS/Linux

```
curl -X POST http://localhost:5000/api/auth/register \  
-H "Content-Type: application/json" \  
-d '{"username":"joao","email":"joao@example.com","password":"senha123}"'
```

9.2 Login

```
curl -X POST http://localhost:5000/api/auth/login \  
-H "Content-Type: application/json" \  
-d '{"username":"joao","password":"senha123}"'
```

9.3 Perfil (com token)

```
curl -X GET http://localhost:5000/api/user/profile \  
-H "Authorization: Bearer TOKEN_AQUI"
```

9.4 Salvar informações pessoais

```
curl -X POST http://localhost:5000/api/user/info \  
-H "Authorization: Bearer TOKEN_AQUI" \  
-H "Content-Type: application/json" \  
-d '{"phone":"(11) 99999-9999","address":"Rua Exemplo","bio":"Dev}"'
```

9.5 Cadastrar face

```
curl -X POST http://localhost:5000/api/user/face/register \  
-H "Authorization: Bearer TOKEN_AQUI" \  
-F "image=@/caminho/para/imagem.jpg"
```

9.6 Reconhecer face

```
curl -X POST http://localhost:5000/api/user/face/recognize \  
-F "image=@/caminho/para/imagem.jpg"
```

9.7 Logout

Windows (PowerShell)

```
curl -X POST http://localhost:5000/api/auth/logout ^  
-H "Authorization: Bearer TOKEN_AQUI"
```

macOS/Linux

```
curl -X POST http://localhost:5000/api/auth/logout \  
-H "Authorization: Bearer TOKEN_AQUI"
```

10. Fluxo de Autenticação (explicado)

1. **Cadastro:** cria um usuário e já devolve um token JWT (chave digital).
2. **Login:** verifica usuário e senha; devolve token novo.
3. **Token:** deve ser enviado no cabeçalho `Authorization` em todas as rotas protegidas (`Bearer SEU_TOKEN`).
4. **Logout:** confirma o encerramento da sessão (em produção, você pode invalidar o token).
5. **Validade:** 7 dias. Após isso, faça login novamente.

Se o token estiver errado ou vencido, a API responde com 401 (não autorizado).

Fluxo Completo de Uso

1. Cadastrar usuário → Recebe token
2. Fazer login → Recebe novo token
3. Usar token para:
 - Ver perfil
 - Cadastrar face
 - Salvar informações pessoais
4. Fazer logout quando terminar

11. Dicas Específicas por Sistema

Windows

- Execute o PowerShell como administrador quando instalar ferramentas do sistema.
- Se `py run.py` não funcionar, tente `python run.py`.
- Para facilitar instalações, use o [Chocolatey](https://chocolatey.org/install) (<https://chocolatey.org/install>) e depois rode `choco install cmake`.

macOS

- Instale o Xcode Command Line Tools com `xcode-select --install`.
- Use `brew install cmake` antes de instalar `face-recognition`.

Linux

- Use o gerenciador da sua distro (`apt`, `dnf`, `pacman`).
- Em WSL (Windows Subsystem for Linux) instale tudo dentro do WSL (Python, `pip`, `cmake`).

12. Solução de Problemas

Erro	Causa comum	Como resolver
<code>ModuleNotFoundError: No module named 'flask'</code>	Ambiente virtual não ativo ao instalar dependências	Ative o ambiente e rode o comando de instalação novamente.
<code>Failed building wheel for dlib</code>	CMake ou compilador ausentes	Instale CMake + ferramentas conforme seu sistema e tente <code>pip install dlib</code> de novo.
<code>sqlite3.OperationalError: database is locked</code>	Fechou o servidor sem encerrar conexões	Pare o servidor, apague <code>database/app.db</code> (apenas em desenvolvimento) e rode <code>run.py</code> novamente.
Token inválido	Header ausente ou token expirado	Refaça o login e envie <code>Authorization: Bearer TOKEN</code> .

Erro	Causa comum	Como resolver
Nenhuma face detectada	Imagem ruim	Use foto nítida, rosto centralizado, boa iluminação.

13. Próximos Passos

- Criar testes automáticos com `pytest`.
 - Substituir SQLite por um banco mais robusto (PostgreSQL) em produção.
 - Criar painel web ou aplicativo mobile para consumir a API.
 - Adicionar upload múltiplo de faces ou histórico de reconhecimentos.
-

14. Checklist Final

1. Python instalado e comando `python/python3/py` funcionando.
2. Ambiente virtual criado e ativado.
3. Dependências instaladas sem erros.
4. Pastas e arquivos criados conforme o diagrama.
5. Servidor iniciado com `run.py` sem mensagens de erro.
6. Testes básicos (cadastro, login, perfil) funcionando.
7. Upload e reconhecimento de rosto testados com imagens reais.

Se todos os itens estiverem ok, você concluiu a API! Continue praticando e adaptando o projeto para novas ideias.