FastAPI Auth API

Guia pessoal (com explicações) para relembrar como o projeto foi montado e porquê.

Objetivo: construir uma API de autenticação com FastAPI com boas práticas reais: registo seguro, login, JWT (stateless) e endpoints protegidos.

1. Visão geral do que foi construído

- /health - endpoint simples para verificar se a API está viva.
- /register - cria utilizadores (email + password), guarda hash na BD e nunca devolve a password.
- /login - valida credenciais e devolve JWT (access token) com expiração.
- /me - endpoint protegido: exige token no header e devolve dados do utilizador autenticado.

Ficheiros principais (estrutura mínima recomendada):

- main.py - FastAPI app + endpoints.
- database.py - funções SQLite (criar tabela, inserir e procurar utilizadores).
- security.py - hashing de password + criação/validação de JWT + constantes de autenticação.

2. Base de dados (SQLite) - porquê e como

Usamos SQLite porque é simples para protótipo e portfólio: não exige servidor e mostra persistência real.

Tabela users (campos):

Campo	Tipo	Porquê existe
id	INTEGER PK	Identificador interno.
email	TEXT UNIQUE	Cada utilizador tem email único.
password_hash	TEXT	Guardamos hash (nunca texto simples).
created_at	TEXT (ISO)	Registo de criação em formato texto padronizado.
is_active	INTEGER (0/1)	Permite desativar utilizadores sem apagar dados.

3. Hashing de passwords (bcrypt via passlib)

A password do utilizador nunca deve ser guardada em texto. Guardamos um hash gerado com bcrypt, que inclui salt. Isso permite validar a password sem nunca a revelar.

Ideia essencial:

- hash_password(password) - recebe texto e devolve um hash seguro para guardar na BD.
- verificar_password(password, password_hash) - compara a password enviada com o hash guardado e devolve True/False.

Nota importante: bcrypt tem limite de ~72 bytes. Passwords muito grandes podem gerar erro. Em projetos reais, a recomendação é impor um limite razoável no input (ex: 8-64 caracteres) e informar o utilizador.

4. JWT e o que significa 'stateless'

Neste projeto o login devolve um JWT (JSON Web Token). A API é stateless: o servidor não guarda sessão em memória. Em vez disso, o cliente envia o token em cada pedido protegido.

- O JWT tem 3 partes: HEADER.PAYLOAD.SIGNATURE.
- O payload é o dicionário com os dados (ex: sub=email e exp=expiração).
- O token é assinado (não é encriptado por padrão). O conteúdo pode ser lido, mas não pode ser alterado sem invalidar a assinatura.
- A segurança depende da SECRET_KEY. Se vazar, alguém pode criar tokens válidos.

5. Constantes em MAIÚSCULAS (porquê)

Em Python, usar MAIÚSCULAS é uma convenção para indicar constantes (configurações fixas). Não é obrigatório pela linguagem, mas é padrão profissional e melhora a leitura.

```
# CONFIGURAÇÕES DE AUTENTICAÇÃO (constantes)
# SECRET_KEY: usada para assinar e validar JWT (em produção, variável de ambiente)
# ALGORITHM: algoritmo de assinatura (HS256 é o mais comum e simples em APIs pequenas)
# ACCESS_TOKEN_EXPIRE_MINUTES: validade do access token em minutos
SECRET_KEY = "..."
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30
```

6. Fluxos principais do sistema

6.1 Registo (/register)

- Recebe JSON com email e password (validação automática via Pydantic).
- Gera password_hash com bcrypt.
- Insere na tabela users.
- Se email já existir, a BD lança IntegrityError e devolvemos 409 Conflict (conflito de recurso).

```
# Exemplo de body (Swagger / Postman)
{
  "email": "marco@email.com",
  "password": "UmaPasswordSegura123"
}
```

6.2 Login (/login)

- Recebe email e password.
- Vai buscar o utilizador na BD (email + password_hash).
- Compara com bcrypt (verificar_password).
- Se estiver correto, cria um JWT com expiração e devolve ao cliente.

```
# Resposta típica do /login
{
  "access_token": "eyJhbGciOi...",
  "token_type": "bearer"
}
```

6.3 Endpoint protegido (/me)

O /me exige token no header HTTP Authorization. O cliente envia no formato Bearer.

```
# Exemplo de header enviado pelo cliente
Authorization: Bearer eyJhbGciOi...
```

- Validamos o formato do header (tem de começar com 'Bearer ').
- Extraímos o token (split por espaço, índice 1).
- Chamamos verificar_token(token): internamente faz jwt.decode e valida assinatura + expiração.
- Se válido, usamos payload['sub'] para obter o email do utilizador autenticado.

7. Porquê 409 no registo?

Quando tentas criar um utilizador com email que já existe, há um conflito com o estado atual do recurso (o recurso 'user' com esse email já existe). Por isso 409 Conflict é apropriado.

Alternativa aceitável: 400 Bad Request. Mas 409 comunica melhor a causa do erro.

8. Testes rápidos (Swagger e cURL)

Swagger UI (docs automáticos do FastAPI):

```
Normalmente:
- http://127.0.0.1:8000/docs  (Swagger UI)
- http://127.0.0.1:8000/redoc (ReDoc)
```

cURL (exemplos):

```
# Register
curl -X POST "http://127.0.0.1:8000/register"   -H "Content-Type: application/json"   -d '{"emai

# Login
curl -X POST "http://127.0.0.1:8000/login"   -H "Content-Type: application/json"   -d '{"email":

# Me (substituir TOKEN_AQUI)
curl -X GET "http://127.0.0.1:8000/me"   -H "Authorization: Bearer TOKEN_AQUI"
```

9. Problemas reais encontrados e como resolveste

- Erro bcrypt/passlib (ex: 'trapped error reading bcrypt version' ou limite de 72 bytes) - resolvido recriando a venv e reinstalando dependências corretamente.
- sqlite3.OperationalError: no such table: users - aconteceu quando a tabela ainda não tinha sido criada; solução: garantir que criar_tabela_utilizadores_db() é chamada no arranque (ou num script de inicialização).

Estas notas são úteis para ti, mas para portfólio público normalmente só se coloca no README o resumo do problema e a solução de forma curta.

10. Checklist de 'projeto sólido' para portfólio

- Separação de ficheiros (main/database/security) e nomes consistentes.
- Erros HTTP corretos (401 para autenticação, 409 para email duplicado).
- Não expor dados desnecessários (no /me devolver só o necessário).
- Comentários curtos mas claros: explicar intenção (stateless, Bearer, decode valida exp).
- README curto com: objetivos, endpoints, como correr, e exemplos de requests.