

Escola Secundária Miguel Torga
Curso Técnico de Gestão de Equipamentos Informáticos
Projeto Stable Pack API



Estudante: Gabriel Alexandre Rocha

Data: 1 Março de 2026

Ano letivo: 2025/2026

Agradecimentos

Agradeço a todos os professores que me acompanharam ao longo destes anos, que auxiliaram e contribuíram para a minha evolução durante o curso, em particular ao Diretor do Curso Profissional Curso Técnico de Gestão de Equipamentos Informáticos e orientador deste projeto, professor Ricardo Pires e ao professor João Monteiro, que acompanhou o desenvolvimento desta Prova de Aptidão Profissional.

Resumo

Este projeto foi desenvolvido no âmbito da Prova de Aptidão Profissional do Curso Profissional de Técnico de Gestão de Equipamentos Informáticos e consiste na implementação de uma API destinada a suportar uma aplicação móvel inspirada em plataformas de entregas. O sistema permite a gestão integrada de contas de clientes, disponibilizando funcionalidades tais como registo, autenticação, recuperação de password de forma totalmente segura. Para garantir uma maior segurança e fiabilidade dos utilizadores deste sistema foram utilizadas tecnologias como FastAPI para construção da API, bcrypt para encriptação de passwords e JWT para gestão de sessões e autenticação. O relatório detalha o processo de desenvolvimento, a arquitetura da API, os modelos de dados utilizados e os principais mecanismos implementados para assegurar a integridade e correto funcionamento da aplicação.

Índice

Agradecimentos.....	2
Resumo.....	3
Índice.....	4
Introdução.....	6
Cronograma.....	7
Enquadramento Teórico.....	7
1. Conceito de API.....	7
2. Arquitetura REST.....	8
3. FastAPI.....	8
4. Base de Dados (MySQL).....	8
5. Encriptação de Passwords com bcrypt.....	8
6. Autenticação com JWT.....	9
7. Envio de Emails com SMTP.....	9
8. Cloudflare Tunnel e CGNAT.....	9
Desenvolvimento do projeto.....	9
1. Início do projeto.....	9
2. Planeamento e Organização.....	10
3. Escolha das tecnologias.....	10
4. Desenvolvimento da Estrutura da API.....	11
5. Desenvolvimento dos ficheiros.....	12
6. Estruturação da Base de Dados.....	12
7. Acesso Externo com Cloudflare Tunnel.....	13
8. Testes e Validação.....	13
Implementação Técnica da API.....	13
1. Importação das Bibliotecas.....	13
2. Criação do Router.....	14
3. Modelos de Dados.....	14
4. Endpoint de Registo.....	14
5. Endpoint de Recuperação de Password.....	15
6. Endpoint de Redefinição de Password.....	15
7. Endpoint de Login.....	16
Conclusão.....	16
Bibliografia.....	17
Anexos.....	17

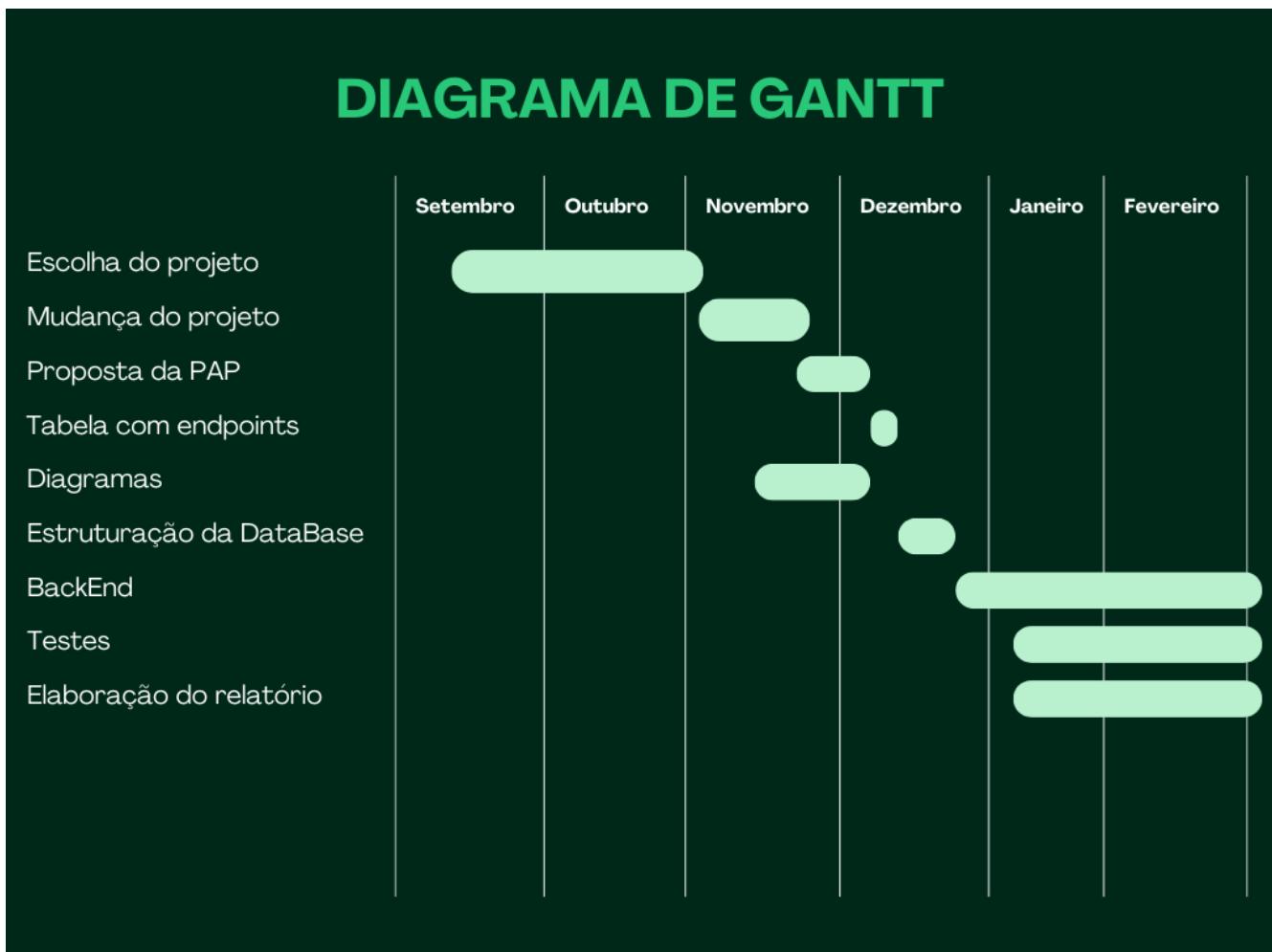
Introdução

O presente projeto foi desenvolvido no âmbito da prova de aptidão profissional do Curso Profissional Técnico de Gestão de Equipamentos Informáticos e integra o trabalho realizado para a criação de uma startup em desenvolvimento, cujo objetivo é criar uma solução tecnológica para o setor de entregas de alimentos combinando com uma aplicação móvel.

A componente desenvolvida neste projeto corresponde à API principal da plataforma, responsável pela autenticação, segurança e comunicação. A API foi construída com recurso ao FastAPI, utilizando MySQL como base de dados, bcrypt para encriptação de passwords e JWT para gestão de sessões e autenticação seguras assim como smtplib para envio dos emails de recuperação. Para permitir a utilização de usuários fora da rede local, foi utilizado também o sistema de túneis da cloudflare para permitir que dispositivos fora da rede local acedam à API. Entre as funcionalidades desenvolvidas destacam-se o registo de utilizadores, login, recuperação de password através de tokens temporários, redefinição de credenciais e emissão de tokens de acesso.

Esta API constitui o núcleo lógico da aplicação, servindo de ponte entre o frontend, Base de dados, desenvolvido pela Prova de Aptidão Profissional de um elemento pertencente à startup.

Cronograma



Anexo 1 - Diagrama de Gantt

Enquadramento Teórico

1. Conceito de API

A palavra API designa-se “Application Programming Interface” e esta consiste num conjunto de regras e endpoints que permite uma conexão entre diferentes partes de um sistemas como por exemplo: frontend e backend. Neste contexto, a API funciona como um agente intermediário entre a aplicação móvel e a base de dados, recebendo pedidos, processando-os e devolvendo respostas. Por norma as APIs modernas seguem o estilo REST, que utiliza métodos tais como: HTTP (GET, POST, PUT, DELETE) para realizar as operações.

2. Arquitetura REST

REST é um estilo arquitetural em inglês para “Representational State Transfer” possui:

Stateless - Cada pedido é independente e contém informações

Uso de Métodos HTTP - Get para obter dados, POST para criar, PUT para atualizar, DELETE para remover

Respostas em JSON - Formato das respostas dos pedidos

Separação entre cliente e servidor - o frontend e o backend funcionam de forma independente sem necessitar do outro para funcionar.

3. FastAPI

O FastAPI é uma recente framework moderna em Python para criação de APIs rápidas e seguras e de forma assíncrona com mais facilidade, devido a esses fatores muitas das pessoas atualmente estão a utilizar FastAPI algumas das suas principais características como:

- Elevado desempenho
- Suporte nativo a validação de dados Pydantic
- Geração automática da rota /docs utilizando redoc e swagger
- Integração simples com JWT, bases de dados ...

4. Base de Dados (MySQL)

O Sistema de Gestão Base de Dados MySQL é um sistema de gestão de bases de dados relacional amplamente utilizado devido à sua estabilidade, desempenho e compatibilidade. Neste projeto MySQL têm um papel fundamental em armazenar os dados de utilizadores, passwords encriptadas, tokens temporários e perfis entre outros dados.

5. Encriptação de Passwords com bcrypt

A segurança das passwords é uma das partes mais importantes de qualquer sistema, devido a isso neste projeto há a utilização do bcrypt em que bcrypt é um algoritmo de hashing que encripta as senhas de todos os utilizadores de forma irreversível por

alguém evitando que mesmo que haja um ataque informático à SGBD as senhas não sejam descobertas.

6. Autenticação com JWT

O JWT ou JSON Web Token é um método seguro e moderno para autenticação de sessões dos quais praticamente todas as organizações no mundo utilizam. Um Token JWT contém Identificação do utilizador, email, cargo, data de expiração. O JWT não necessita de manter sessões no servidor da API, é fácil de validar e seguro quando assinado com chave secreta.

7. Envio de Emails com SMTP

SMTP é uma biblioteca padrão do python que permite enviar emails através de servidores SMTP, neste projeto aplica-se no envio de emails para os utilizadores receberem o token e conseguirem alterar a password em caso de esquecimento.

8. Cloudflare Tunnel e CGNAT

Durante a fase de testes, surgiu um desafio no qual apercebi-me de que a API só funcionava em modo local sem qualquer problema devido a ser para uma demonstração da PAP porém necessita de um link fixo para poder se comunicar com o backend então após pesquisa sobre como poderei ter uma solução de possuir um IP fixo sem necessitar de grandes custos encontrei a Cloudflare Tunnel para a resolução deste problema.

Cloudflare Tunnel cria um túnel seguro entre o servidor local e a Cloudflare permitindo acesso externo sem abrir portas de router. O servidor da API fica mais seguro devido a não ter que se conectar diretamente à máquina e sim ao servidor da cloudflare e o servidor da cloudflare se comunica na máquina, evitando ataques.

Desenvolvimento do projeto

1. Início do projeto

A ideia do desenvolvimento deste projeto começou devido à criação da startup solvex, que consiste na criação de uma API para uma aplicação móvel de entregas de alimentos. O objetivo principal é utilizar uma API segura, escalável e eficiente, que

permitisse fazer autenticação de forma segura, conectar-se com o frontend, recuperação de credenciais.

Inicialmente existia uma ideia diferente para a PAP, porém após análise concluiu-se que faria sentido alinhar o projeto com o trabalho desenvolvido na startup na PAP devido a irmos desenvolver a nossa startup no estágio.

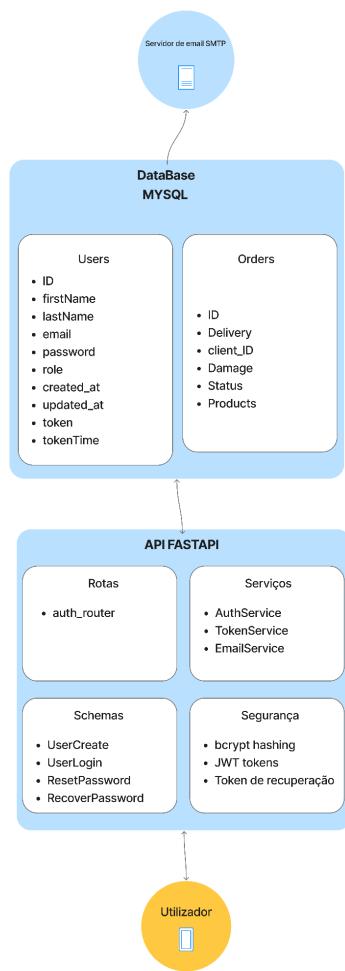
2. Planeamento e Organização

Para garantir a realização deste projeto as partes do desenvolvimento foram distribuídas em um diagrama de Gantt estas partes como:

- Escolha e reformulação do projeto
- Elaboração da proposta da PAP
- Criação da tabela de endpoints
- Desenvolvimento dos diagramas
- Estruturação da base de dados
- Desenvolvimento do backend
- Testes e validação
- Elaboração do relatório

3. Escolha das tecnologias

A seleção do FastAPI veio do facto devido à sua rapidez, simplicidade e documentação automática, MySQL pela estabilidade e facilidade e conhecimento, bcrypt para garantir hashing seguro das passwords, JWT para autenticação segura e moderna, SMTP para envio dos emails, Cloudflare Tunnel para permitir acesso externo sem abrir portas do router.



Anexo 2 - Estrutura da API com as tecnologias

4. Desenvolvimento da Estrutura da API

A API foi organizada em ficheiros independentes, facilitando a leitura, manutenção e escalabilidade do código.

Nome	Estado	Data de modificação	Tipo	Tamanho
.venv	Cloud	06/02/2026 17:22	Pasta de ficheiros	
__pycache__	Cloud	27/02/2026 18:01	Pasta de ficheiros	
Documentos	Cloud	01/03/2026 14:42	Pasta de ficheiros	
Relatório	Cloud	06/02/2026 17:21	Pasta de ficheiros	
.env	✓	26/02/2026 23:01	Ficheiro ENV	1 KB
.gitignore	✓	09/02/2026 08:34	Arquivo Fonte Git ...	1 KB
auth_routes.py	✓	26/02/2026 22:35	Arquivo Fonte Pyt...	5 KB
database.py	✓	22/02/2026 15:32	Arquivo Fonte Pyt...	1 KB
email_recover.py	✓	22/02/2026 15:58	Arquivo Fonte Pyt...	2 KB
jwt_handler.py	✓	26/02/2026 22:06	Arquivo Fonte Pyt...	1 KB
main.py	✓	26/02/2026 10:04	Arquivo Fonte Pyt...	1 KB
requirements.txt	✓	01/03/2026 20:11	Arquivo Fonte Text	3 KB

Anexo 3 - Estrutura dos ficheiros

5. Desenvolvimento dos ficheiros

.venv - É um Diretório que contém um ambiente virtual do python, incluindo todas as bibliotecas utilizadas neste projeto.

__pycache__ - Pasta criada automaticamente pelo Python para armazenar ficheiros compilados permitindo uma execução mais rápida do código.

.env - Ficheiro onde são guardadas variáveis sensíveis, como palavras-passe, credenciais e chaves secretas.

requirements.txt - Lista de todas as bibliotecas necessárias para executar a API, permite instalar rapidamente todas as dependências.

main.py - É responsável pela inicialização da API, nele são configuradas rotas.

database.py - Este ficheiro contém a função get_database(), responsável por estabelecer a ligação com a base de dados.

jwt_handler.py - Implementa a criação de tokens JWT utilizados no processo de autenticação, gera tokens assinados com chave secreta e tempo de expiração.

email_recover.py - Responsável por enviar os emails de recuperação e utiliza o protocolo SMTP. É executado em segundo plano para não atrasar

auth_routes.py - Ficheiro principal da API no que diz respeito à autenticação.

6. Estruturação da Base de Dados

A base de dados foi criada em MySQL, possuindo a base de dados pap e a tabela users e orders. A tabela *users* contém todos os dados necessários para a autenticação e gestão de utilizadores. A estrutura foi pensada para uma expansão futura para permitir integração de restaurantes, entregadores e pedidos.

7. Acesso Externo com Cloudflare Tunnel

Durante o desenvolvimento deste projeto surgiu o desafio de como utilizar a minha API em um dispositivo fora da minha rede local. Para ultrapassar esse problema, foi implementado o Cloudflare Tunnel que cria um túnel seguro entre o servidor local e a cloudflare, permitindo acesso externo sem expor qualquer informação do servidor da API.

8. Testes e Validação

Os testes foram realizados com a ferramenta Postman e com a própria ferramenta do FastAPI /docs que possui um método de try em que podemos testar. Também foram feitos testes com o frontend.

Implementação Técnica da API

1. Importação das Bibliotecas

Estas foram as bibliotecas utilizadas em auth_routes.py para garantir a autenticação

- fastapi - É a biblioteca da nossa framework, e dentro dela existem as bibliotecas APIRouter que permite organizar os nossos endpoints, HTTPException para criar exceções, status contêm códigos de HTTP, e BackgroundTasks que permite executar tarefas em segundo plano neste caso está a ser utilizado para enviar emails
- pydantic - usado para validar dados
- database - Vai buscar a conexão da função get_database em database.py e retorna a conexão da base de dados
- send_recovery_email - Função que permite enviar os emails de recuperação
- create_access_token - Gera os TokensJWT para permitir que os utilizadores iniciem sessão.
- bcrypt - Encriptação de passwords.
- secrets - Geração de tokens seguros-
- datetime - Manipulação de datas
- pymysql - Comunicação com MySQL

```
from fastapi import APIRouter, HTTPException, status, BackgroundTasks
from pydantic import BaseModel
from database import get_database
from email_recover import send_recovery_email
from jwt_handler import create_access_token
import bcrypt
import secrets
import datetime
import pymysql
```

Anexo 4 - Bibliotecas Utilizadas

2. Criação do Router

auth_route define o prefixo e organiza os endpoints relacionados com autenticação contidos no ficheiro. Exemplo: /auth/...

```
auth_route = APIRouter(prefix="/auth", tags=["Auth"])
```

Anexo 5 - auth_route

3. Modelos de Dados

Este modelo de dados define o formato obrigatório dos pedidos, por exemplo para o endpoint /auth/Register vai precisar obrigatoriamente de firstName, lastName, email, password, role.

```
class RegisterRequest(BaseModel):
    firstName: str
    lastName: str
    email: str
    password: str
    role: str

class LoginRequest(BaseModel):
    email: str
    password: str

class RecoverPassword(BaseModel):
    email: str
class ResetPassword(BaseModel):
    email: str
    password: str
    token: str
```

Anexo 6 - Pydantic

4. Endpoint de Registo

Este Endpoint vai receber os dados obrigatórios do pedido e vai verificar se o email já existe na base de dados caso exista dá erro caso não exista irá encriptar a password com bcrypt e insere um novo utilizador na base de dados e devolve uma mensagem de sucesso.

```
@auth_route.post("/register")
async def register(request: RegisterRequest):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Email já foi registrado")

    hashed_password = bcrypt.hashpw(request.password.encode("utf-8"), bcrypt.gensalt())
    hashed_password = hashed_password.decode("utf-8")

    try:
        cursor.execute("INSERT INTO users (firstName, lastName, email, password, role) VALUES (%s, %s, %s, %s, %s)", (request.firstName, request.lastName, request.email, hashed_password, request.role))
        print(f"Utilizador com o nome: {request.firstName} foi criado")
    except pymysql.MySQLError:
        raise HTTPException(status_code=500, detail="Erro ao criar utilizador")

    return {"message": "Utilizador registado com sucesso"}
```

Anexo 7 - Endpoint de Registo

5. Endpoint de Recuperação de Password

Recebe o email, caso não exista irá dar erro Não foi encontrado nenhum email correspondente, caso exista irá criar um token com a biblioteca secrets e um tempo de expiração do token de 10 minutos utilizando o datetime, em seguida a função é chamada para enviar o email e enviar o email para o utilizador caso não haja nenhum erro na criação dos tokens e retornará sucesso

```
@auth_route.post("/recoverPassword")
async def recoverPassword(request: RecoverPassword, background: BackgroundTasks):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user is None:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Não foi encontrado nenhum email correspondente")

    token = secrets.token_urlsafe(32)
    expire = datetime.datetime.now() + datetime.timedelta(minutes=10)

    try:
        cursor.execute("UPDATE users SET token = %s, tokenTime = %s WHERE email = %s", (token, expire, request.email))
        background.add_task(send_recovery_email, request.email, token, user["firstName"])
    except pymysql.MySQLError:
        raise HTTPException(status_code=500, detail="Erro ao criar os tokens de verificação")

    return {"message": "Email enviado com sucesso"}
```

Anexo 8 - Endpoint de Recuperação de password

6. Endpoint de Redefinição de Password

Este endpoint acontecerá ao resetar a password após clicar no link recebido, irá ser verificado o email e se o token é válido e se o tempo também é válido caso tudo seja válido a password será encriptada com bcrypt e alterada na base de dados e o token será apagado.

```
@auth_route.post("/resetPassword")
async def resetPassword(request:ResetPassword):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user == None:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Não foi possível encontrar o email na base de dados")

    if user["token"] != request.token:
        raise HTTPException(status_code=400, detail="Token inválido")

    token_time = datetime.datetime.fromisoformat(str(user["tokenTime"]))

    if token_time < datetime.datetime.now():
        raise HTTPException(status_code=400, detail="Token expirado")

    password = bcrypt.hashpw( request.password.encode("utf-8"), bcrypt.gensalt() ).decode("utf-8")
    cursor.execute( "UPDATE users SET password = %s, token = NULL, tokenTime = NULL WHERE ID = %s", (password, user["ID"]) )

    return{"message": "Password alterada com sucesso"}
```

Anexo 9 - Endpoint de Redefinição de Password

7. Endpoint de Login

Este endpoint procura o email do utilizador na base de dados caso não exista dá erro, caso exista compara com a password enviada com a password da Base de dados encriptada caso esteja correta gera um JWT com: ID, email, role e devolve o token ao frontend

```
@auth_route.post("/login")
async def login(request: LoginRequest):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user is None:
        raise HTTPException(status_code=400, detail="Credenciais Inválidas")

    if not bcrypt.checkpw(request.password.encode("utf-8"), user["password"].encode("utf-8")):
        raise HTTPException(status_code=400, detail="Credenciais Inválidas")

    access_token = create_access_token({"sub": str(user["ID"]), "email": user["email"], "role": user["role"]})

    return {
        "access_token": access_token,
        "token_type": "bearer"
    }
```

Anexo 10 - Endpoint de Login

Conclusão

O desenvolvimento desta PAP permitiu consolidar conhecimentos técnicos, nomeadamente na criação de APIs, estruturas de dados e como proteger autenticações. ao longo do projeto foram abordados aspectos científicos e tecnológicos, como a utilização do FastAPI, a implementação com tokens pelo JWT, a encriptação de passwords com bcrypt, gestão de Base de dados MySQL e a criação de mecanismos de recuperação de credenciais através de tokens temporários. A integração com o Cloudflare Tunnel foi igualmente relevante, permitindo disponibilizar a API externamente sem necessidade de se encontrar na mesma rede local, e possibilitou também utilizar esta API com HTTPS tornando ainda mais segura sem necessidade de abrir qualquer porta de algum router e mesmo em ambientes com restrições de rede.

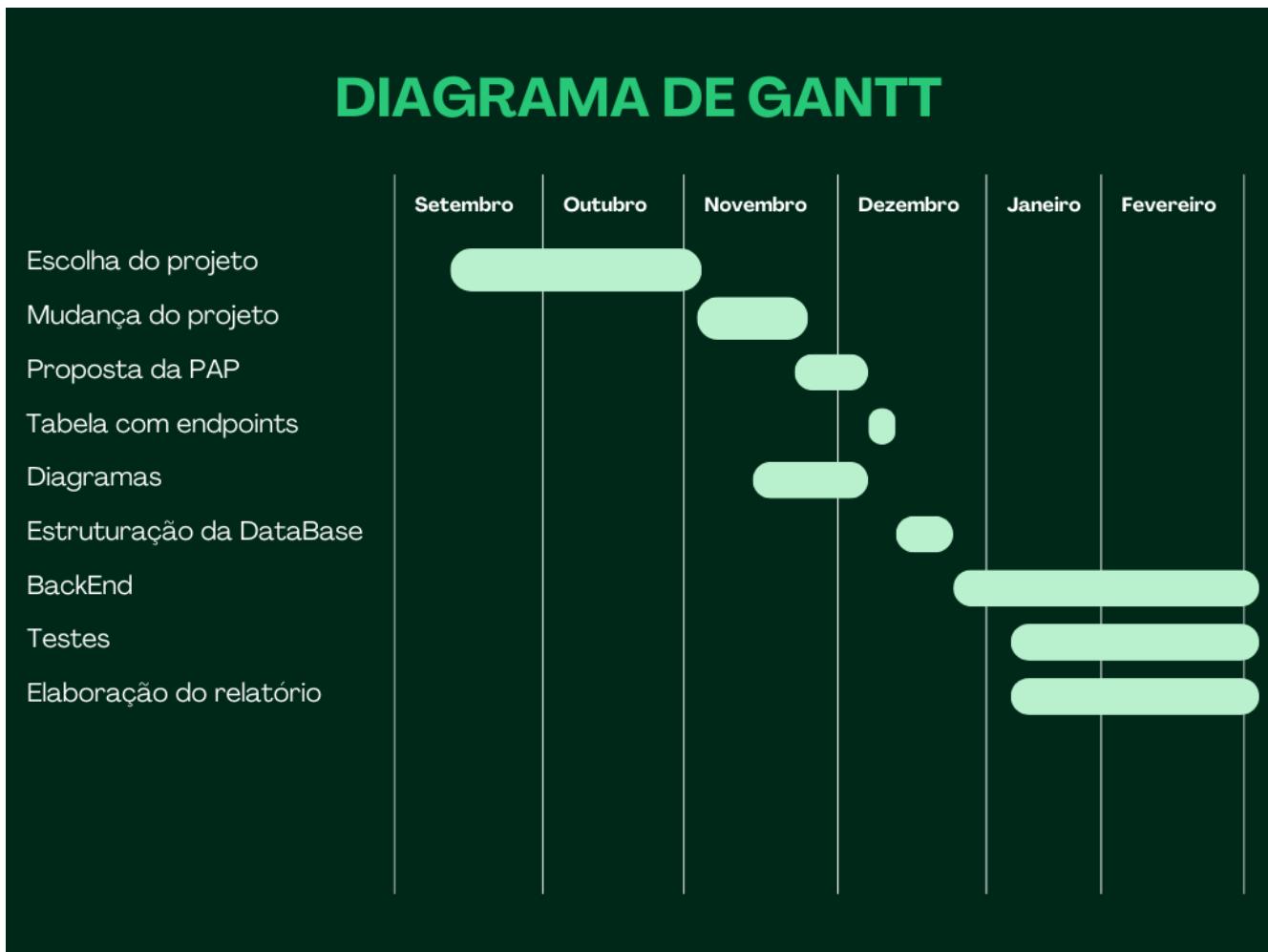
Para além da componente técnica este trabalho contribuiu significativamente para o desenvolvimento de competências profissionais. Devido à necessidade de organização de tarefas, documentar os processos feitos, e até mesmo em avançar com a startup porque este projeto integra-se diretamente no trabalho desenvolvido na startup o que permite um aumento significativo.

Esta PAP representou um desafio que exigiu autonomia, responsabilidade, capacidade de resolução de problemas, persistência e automotivação. A aprendizagem destas ferramentas, até então desconhecidas, representou um desafio que contribuiu significativamente para o meu crescimento, uma vez que a superação de obstáculos promove evolução pessoal e profissional, e este foi um crescimento tanto académico como profissional como pessoal.

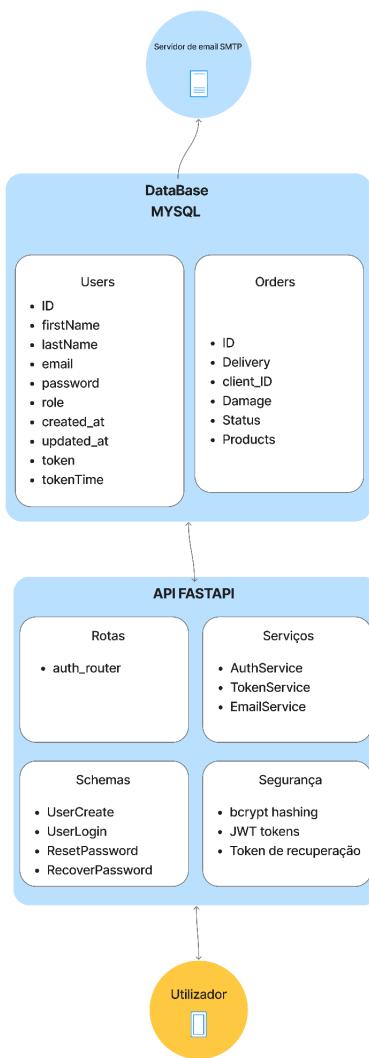
Bibliografia

- <https://fastapi.tiangolo.com/tutorial/>
- <https://www.python.org/doc>
- <https://dev.mysql.com/doc>
- <https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/>
- <https://jwt.io>
- <https://pypi.org/project/bcrypt>
- <https://pymysql.readthedocs.io>
- <https://www.w3schools.com>
- <https://stackoverflow.com>
- <https://github.com>

Anexos



Anexo 1



Anexo 2

Nome	Estado	Data de modificação	Tipo	Tamanho
.venv	cloud	06/02/2026 17:22	Pasta de ficheiros	
__pycache__	cloud	27/02/2026 18:01	Pasta de ficheiros	
Documentos	cloud	01/03/2026 14:42	Pasta de ficheiros	
Relatório	cloud	06/02/2026 17:21	Pasta de ficheiros	
.env	checkmark	26/02/2026 23:01	Ficheiro ENV	1 KB
.gitignore	checkmark	09/02/2026 08:34	Arquivo Fonte Git ...	1 KB
auth_routes.py	checkmark	26/02/2026 22:35	Arquivo Fonte Pyt...	5 KB
database.py	checkmark	22/02/2026 15:32	Arquivo Fonte Pyt...	1 KB
email_recover.py	checkmark	22/02/2026 15:58	Arquivo Fonte Pyt...	2 KB
jwt_handler.py	checkmark	26/02/2026 22:06	Arquivo Fonte Pyt...	1 KB
main.py	checkmark	26/02/2026 10:04	Arquivo Fonte Pyt...	1 KB
requirements.txt	checkmark	01/03/2026 20:11	Arquivo Fonte Text	3 KB

Anexo 3

```
from fastapi import APIRouter, HTTPException, status, BackgroundTasks
from pydantic import BaseModel
from database import get_database
from email_recover import send_recovery_email
from jwt_handler import create_access_token
import bcrypt
import secrets
import datetime
import pymysql
```

Anexo 4

```
auth_route = APIRouter(prefix="/auth", tags=["Auth"])
```

Anexo 5

```
class RegisterRequest(BaseModel):
    firstName: str
    lastName: str
    email: str
    password: str
    role: str

class LoginRequest(BaseModel):
    email: str
    password: str

class RecoverPassword(BaseModel):
    email: str
class ResetPassword(BaseModel):
    email: str
    password: str
    token: str
```

Anexo 6

```

@auth_route.post("/register")
async def register(request: RegisterRequest):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Email já foi registrado")

    hashed_password = bcrypt.hashpw(request.password.encode("utf-8"), bcrypt.gensalt())
    hashed_password = hashed_password.decode("utf-8")
    try:
        cursor.execute("INSERT INTO users (firstName, lastName, email, password, role) VALUES (%s, %s, %s, %s, %s)", (request.firstName, request.lastName, request.email, hashed_password, request.role))
        print(f"Utilizador com o nome: {request.firstName} foi criado")
    except pymysql.MySQLError:
        raise HTTPException(status_code=500, detail="Erro ao criar utilizador")
    return {"message": "Utilizador registado com sucesso"}

```

Anexo 7

```

@auth_route.post("/recoverPassword")
async def recoverPassword(request: RecoverPassword, background: BackgroundTasks):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user is None:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Não foi encontrado nenhum email correspondente")

    token = secrets.token_urlsafe(32)
    expire = datetime.datetime.now() + datetime.timedelta(minutes=10)

    try:
        cursor.execute("UPDATE users SET token = %s, tokenTime = %s WHERE email = %s", (token, expire, request.email))
        background.add_task(send_recovery_email, request.email, token, user["firstName"])
    except pymysql.MySQLError:
        raise HTTPException(status_code=500, detail="Erro ao criar os tokens de verificação")
    return {"message": "Email enviado com sucesso"}

```

Anexo 8

```

@auth_route.post("/resetPassword")
async def resetPassword(request: ResetPassword):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user == None:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Não foi possível encontrar o email na base de dados")

    if user["token"] != request.token:
        raise HTTPException(status_code=400, detail="Token inválido")

    token_time = datetime.datetime.fromisoformat(str(user["tokenTime"]))

    if token_time < datetime.datetime.now():
        raise HTTPException(status_code=400, detail="Token expirado")

    password = bcrypt.hashpw(request.password.encode("utf-8"), bcrypt.gensalt()).decode("utf-8")
    cursor.execute("UPDATE users SET password = %s, token = NULL, tokenTime = NULL WHERE ID = %s", (password, user["ID"]))

    return {"message": "Password alterada com sucesso"}

```

Anexo 9

```

@auth_route.post("/login")
async def login(request: LoginRequest):
    db = get_database()
    cursor = db.cursor()

    cursor.execute("SELECT * FROM users WHERE email = %s", (request.email,))
    user = cursor.fetchone()

    if user is None:
        raise HTTPException(status_code=400, detail="Credenciais Inválidas")

    if not bcrypt.checkpw(request.password.encode("utf-8"), user["password"].encode("utf-8")):
        raise HTTPException(status_code=400, detail="Credenciais Inválidas")

    access_token = create_access_token({"sub": str(user["ID"]), "email": user["email"], "role": user["role"]})

    return {
        "access_token": access_token,
        "token_type": "bearer"
    }

```

Anexo 10

```

main.py > ...
1
2     from dotenv import load_dotenv
3     from fastapi import FastAPI
4     from auth_routes import auth_route
5
6
7
8
9     app = FastAPI()
10
11    load_dotenv()
12
13
14
15    app.include_router(auth_route)
16
17
18    @app.get("/")
19    async def teste():
20        return {"API": "A minha API está a funcionar!"}
21
22
23
24

```

Anexo 11

```
❶ jwt_handler.py > ...
Q 1  from dotenv import load_dotenv
  2  import os
  3  import jwt
  4  from datetime import datetime, timedelta
  5
  6  load_dotenv()
  7
  8  SECRET_KEY = os.getenv("SECRET_KEY")
  9  ALGORITHM = os.getenv("ALGORITHM")
10
11 def create_access_token(data: dict, expires_delta: timedelta = timedelta(hours=1)):
12     to_encode = data.copy()
13     expire = datetime.utcnow() + expires_delta
14     to_encode.update({"exp": expire})
15
16     encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
17     return encoded_jwt
18
```

Anexo 12

```
❶ database.py > ⏺ get_database
 1  import pymysql
 2
 3
 4  print("Script iniciado")
 5
 6  def get_database():
 7      return pymysql.connect(
 8          host="127.0.0.1",
 9          user="root",
10          password="123456789",
11          database="pap",
12          autocommit=True,
13          cursorclass=pymysql.cursors.DictCursor
14      )
 15
 16
 17
 18  db = get_database()
 19  cursor = db.cursor()
 20
 21
 22  cursor.execute("SELECT * FROM users")
 23  print(cursor.fetchall())
```

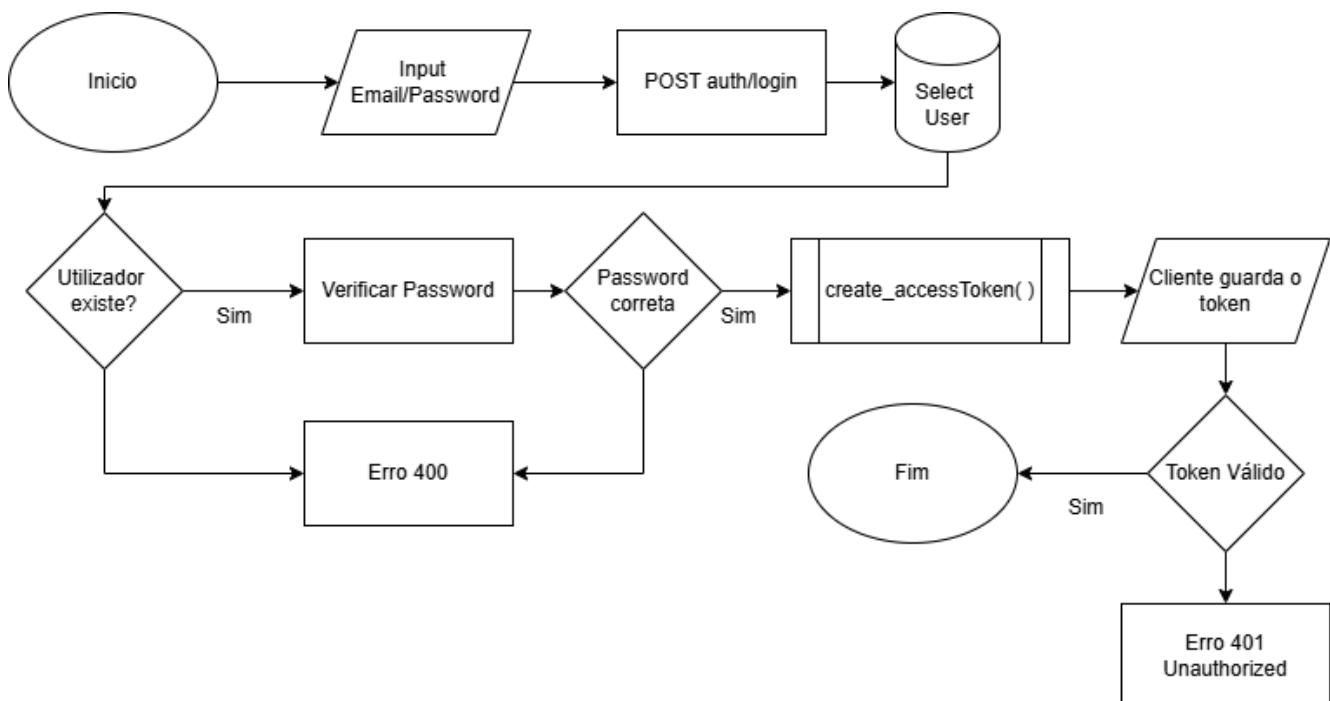
Anexo 13

```

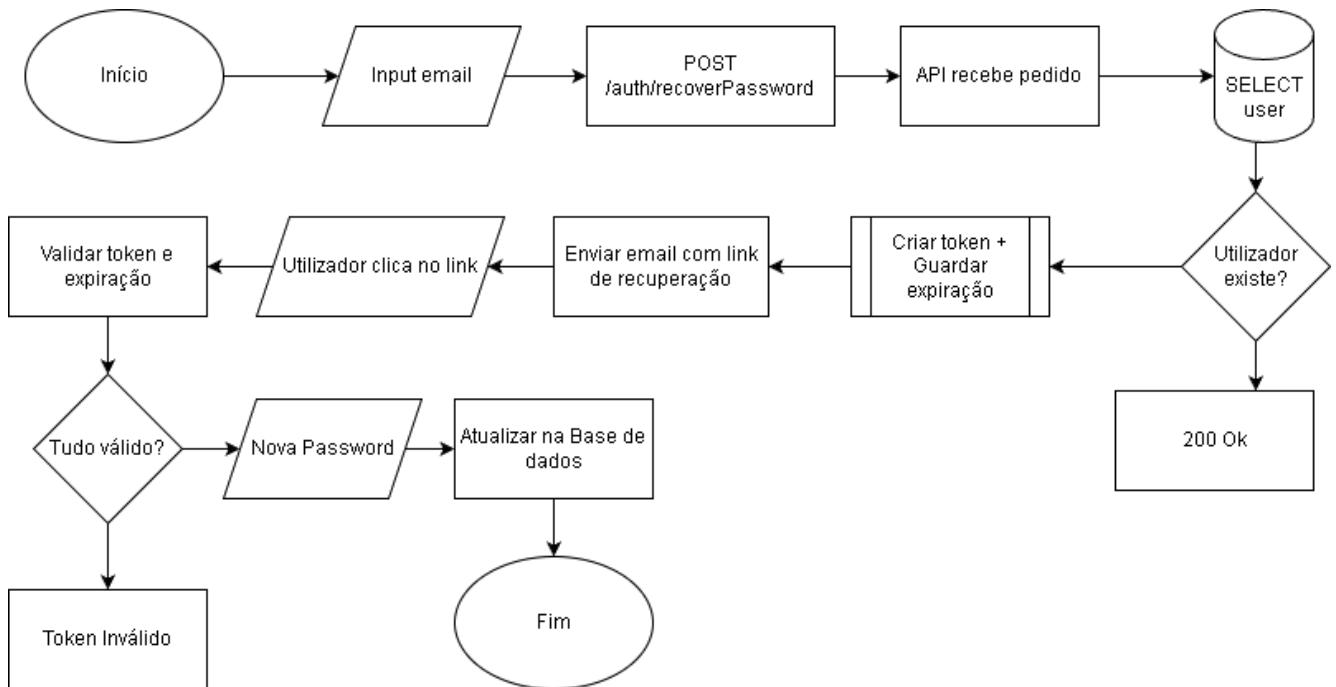
● email_recover.py > send_recovery_email
1  import os
2  import smtplib
3  from email.mime.text import MIMEText
4  from email.mime.multipart import MIMEMultipart
5
6  def send_recovery_email(to_email: str, token: str, first_name: str):
7      sender_email = os.getenv('EMAIL_USER')
8      sender_password = os.getenv('EMAIL_PASS')
9
10     print(f'Email: {sender_email}')
11     reset_link = f"http://localhost:3000/reset-password?token={token}"
12
13     subject = "Recuperação de Password"
14     body = f'''
15     Olá {first_name},
16
17     Este é o suporte da plataforma Solvex, enviste um pedido para recuperar a tua conta caso não tenhas sido tu a efetuar este pedido, certifica-te que tens acesso à tua conta e altera a palavra-passe caso necessário
18
19     Clica no link abaixo para redefinir a tua password:
20
21     {reset_link}
22
23     ...
24
25
26     msg = MIMEMultipart()
27     msg['from'] = sender_email
28     msg['to'] = to_email
29     msg['subject'] = subject
30     msg.attach(MIMEText(body, "plain"))
31
32     with smtplib.SMTP("smtp.gmail.com", 587) as server:
33         server.starttls()
34         server.login(sender_email, sender_password)
35         server.sendmail(sender_email, to_email, msg.as_string())
36

```

Anexo 14



Anexo 15



Anexo 16

```

POST https://lecture-correlation-informal-classics.trycloudflare.com/auth/login
Content-Type: application/json
{
  "email": "a.rochagabri@gmail.com",
  "password": "teste"
}

{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyIiwihfaWwiOiJhLnJvY2hhZ2FicmlAZ21haWwuY29tIiwiZm9sZSI6ImNsawWudCisImV4cCI6MTc3MjQwNjMwNX0.-Ytu2kIoyerJd3VIgmsJytOmAj2DodiGWJ-.80g",
  "token_type": "bearer"
}
  
```

TERMINAL

```

INFO: Started server process [2528]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:55756 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:55756 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 148.71.17.34:0 - "POST /auth/login HTTP/1.1" 200 OK
  
```

Anexo 17

POST https://lecture-correlation-informal-classics.trycloudflare.com/auth/recoverPassword

Params Authorization Headers Body Scripts Settings

Body (JSON)

```

1 {
2   "email": "a.rochagabri@gmail.com"
3 }

```

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON

```

1 {
2   "message": "Email enviado com sucesso"
3 }

```

Status: 200 OK Time: 250 ms Size:

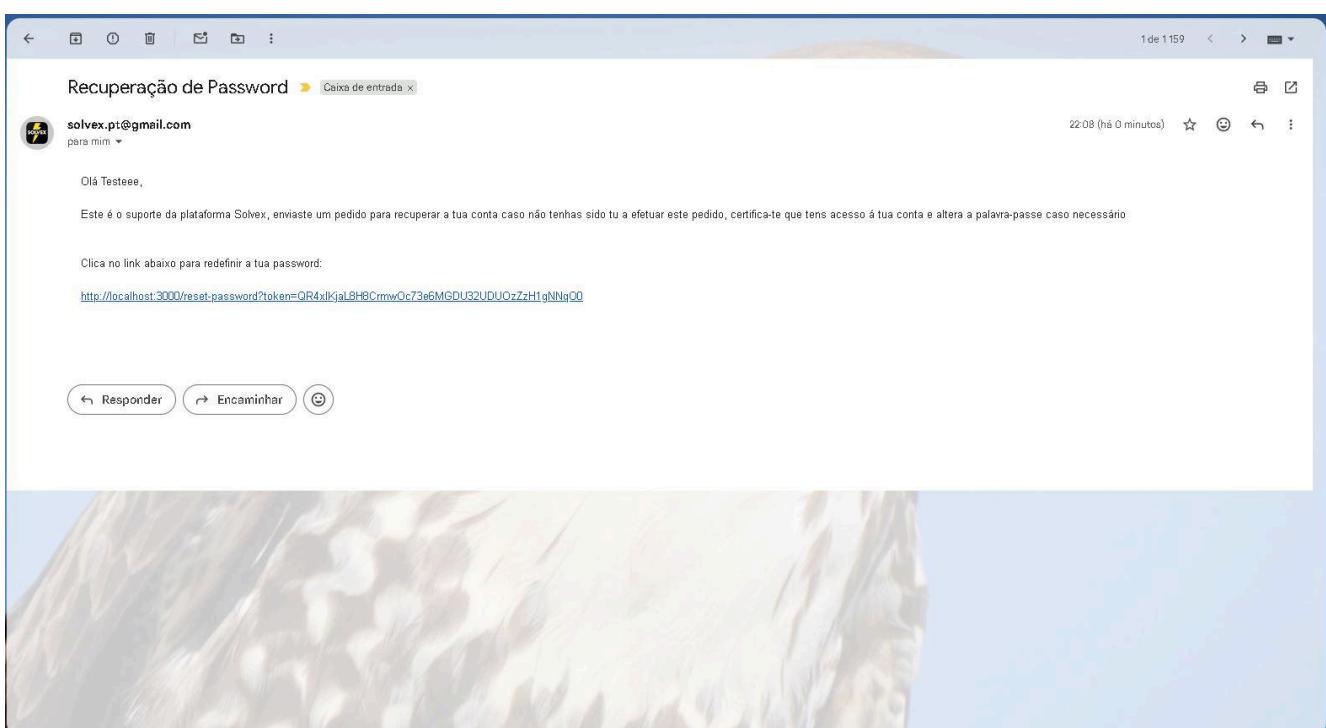
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```

INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:55756 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:55756 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 148.71.17.34:0 - "POST /auth/login HTTP/1.1" 200 OK
Utilizador com o nome: Solvex foi criado
INFO: 148.71.17.34:0 - "POST /auth/register HTTP/1.1" 200 OK
INFO: 148.71.17.34:0 - "POST /auth/recoverPassword HTTP/1.1" 200 OK

```

Anexo 18



Anexo 19

The screenshot shows the Postman interface with the following details:

- HTTP Method:** POST
- URL:** https://lecture-correlation-informal-classics.trycloudflare.com/auth/resetPassword
- Body:** JSON (selected)


```

1 []
2 ... "email": "a.rochagabri@gmail.com",
3 ... "token": "Q4x1KjaLBHBCrmwOc79e6MGDU32UDUzZzH1gNNq00",
4 ... "password": "teste"
5 []

```
- Test Results:** Status: 200 OK Time: 449 ms Size: 0


```

1 [{"message": "Senha alterada com sucesso"}]

```
- Terminal Output:**

```

INFO: 148.71.17.34:0 - "POST /auth/login HTTP/1.1" 200 OK
Utilizador com o nome: Solvex foi criado
INFO: 148.71.17.34:0 - "POST /auth/register HTTP/1.1" 200 OK
INFO: 148.71.17.34:0 - "POST /auth/recoverPassword HTTP/1.1" 200 OK
Email: solvex.pt@gmail.com
INFO: 148.71.17.34:0 - "POST /auth/recoverPassword HTTP/1.1" 200 OK
Email: solvex.pt@gmail.com
INFO: 148.71.17.34:0 - "POST /auth/resetPassword HTTP/1.1" 422 Unprocessable Entity
INFO: 148.71.17.34:0 - "POST /auth/resetPassword HTTP/1.1" 200 OK

```

Anexo 20

The screenshot shows the Postman interface with the following details:

- HTTP Method:** POST
- URL:** https://lecture-correlation-informal-classics.trycloudflare.com/auth/register
- Body:** JSON (selected)


```

1 []
2 ... "firstName": "Solvex",
3 ... "lastName": "Solvex",
4 ... "email": "solvex.pt@gmail.com",
5 ... "password": "teste",
6 ... "role": "client"
7 []

```
- Test Results:** Status: 200 OK Time: 489 ms Size: 0


```

1 [{"message": "Utilizador registado com sucesso"}]

```
- Terminal Output:**

```

datetime.datetime(2026, 2, 6, 16, 41, 11), 'update_at': datetime.datetime(2026, 2, 26, 21, 51, 1), 'token': None, 'tokenTime': None})
INFO: Started server process [25128]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:55756 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:55756 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 148.71.17.34:0 - "POST /auth/login HTTP/1.1" 200 OK
Utilizador com o nome: Solvex foi criado
INFO: 148.71.17.34:0 - "POST /auth/register HTTP/1.1" 200 OK

```

Anexo 21