O backend foi desenvolvido para suportar autenticação de usuários, validação de credenciais da AWS e interação com o MongoDB. Abaixo está o fluxo geral:

1. \*\*Conexão com o MongoDB\*\*:

- O backend conecta-se ao MongoDB usando a URI fornecida no arquivo `.env`.

- A coleção `users` é usada para armazenar informações dos usuários.

2. \*\*Registro de Usuários\*\*:

- Os usuários podem se registrar fornecendo nome, email e senha.

- A senha é criptografada antes de ser salva no banco de dados.

3. \*\*Login de Usuários\*\*:

- Os usuários podem fazer login fornecendo email e senha.

- Após a autenticação, um token JWT é gerado e retornado ao cliente.

4. \*\*Validação de Credenciais da AWS\*\*:

- Após o login, os usuários podem fornecer suas credenciais da AWS (Access Key e Secret Key).

- As credenciais são validadas e, se corretas, permitem acesso a funcionalidades relacionadas à AWS.

5. \*\*Geração e Validação de Tokens JWT\*\*:

- Tokens JWT são usados para autenticar requisições subsequentes.

- Os tokens têm uma validade limitada (1 hora) e são validados pelo backend em cada requisição protegida.

6. \*\*Rotas Protegidas\*\*:

- Algumas rotas exigem autenticação via token JWT.

- Um middleware verifica o token antes de permitir o acesso às rotas protegidas.

---

### \*\*Funções Criadas no Backend\*\*

Aqui estão as principais funções implementadas até agora:

#### \*\*1. Configuração do MongoDB\*\*

```python

from pymongo import MongoClient

from dotenv import load\_dotenv

import os

# Carregar variáveis de ambiente

load\_dotenv()

# Conectar ao MongoDB

MONGODB\_URI = os.getenv("MONGODB\_URI")

if not MONGODB\_URI:

raise ValueError("URI do MongoDB não encontrada nas variáveis de ambiente")

client = MongoClient(MONGODB\_URI)

db = client.get\_database() # Obtém o banco de dados da URI

users\_collection = db["users"] # Coleção de usuários

```

---

#### \*\*2. Registro de Usuários\*\*

```python

from werkzeug.security import generate\_password\_hash

def create\_user(name, email, password):

# Verificar se o email já existe

existing\_user = users\_collection.find\_one({"email": email})

if existing\_user:

return {"error": "Email já cadastrado"}

# Criptografar a senha

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

# Salvar o usuário no banco de dados

user\_id = users\_collection.insert\_one({

"name": name,

"email": email,

"password": hashed\_password

}).inserted\_id

return str(user\_id)

```

---

#### \*\*3. Login de Usuários\*\*

```python

from werkzeug.security import check\_password\_hash

import jwt

from datetime import datetime, timedelta

def verify\_password(email, password):

user = users\_collection.find\_one({"email": email})

if not user or not check\_password\_hash(user["password"], password):

return False

return True

def generate\_token(user\_id):

secret\_key = os.getenv("JWT\_SECRET")

if not secret\_key:

raise ValueError("Chave secreta JWT não encontrada nas variáveis de ambiente")

payload = {

"user\_id": user\_id,

"exp": datetime.utcnow() + timedelta(hours=1) # Token expira em 1 hora

}

return jwt.encode(payload, secret\_key, algorithm="HS256")

def validate\_token(token):

secret\_key = os.getenv("JWT\_SECRET")

if not secret\_key:

raise ValueError("Chave secreta JWT não encontrada nas variáveis de ambiente")

try:

payload = jwt.decode(token, secret\_key, algorithms=["HS256"])

return payload["user\_id"]

except jwt.ExpiredSignatureError:

return {"error": "Token expirado"}

except jwt.InvalidTokenError:

return {"error": "Token inválido"}

```

---

#### \*\*4. Middleware de Autenticação JWT\*\*

```python

from functools import wraps

from flask import request, jsonify

def token\_required(f):

@wraps(f)

def decorated(\*args, \*\*kwargs):

token = None

# Verificar se o token está presente no cabeçalho da requisição

if "Authorization" in request.headers:

token = request.headers["Authorization"].split(" ")[1]

if not token:

return jsonify({"error": "Token ausente"}), 401

# Validar o token

user\_id = validate\_token(token)

if isinstance(user\_id, dict) and "error" in user\_id:

return jsonify(user\_id), 401

kwargs["current\_user\_id"] = user\_id

return f(\*args, \*\*kwargs)

return decorated

```

---

#### \*\*5. Rota de Registro (`POST /api/register`)\*\*

```python

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

def register():

data = request.json

name = data.get("name")

email = data.get("email")

password = data.get("password")

result = create\_user(name, email, password)

if isinstance(result, dict) and "error" in result:

return jsonify(result), 400

return jsonify({"message": "Usuário criado com sucesso!", "user\_id": result}), 201

```

---

#### \*\*6. Rota de Login (`POST /api/login`)\*\*

```python

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

def login():

data = request.json

email = data.get("email")

password = data.get("password")

if not email or not password:

return jsonify({"error": "Email e senha são obrigatórios"}), 400

if not verify\_password(email, password):

return jsonify({"error": "Credenciais inválidas"}), 401

user = users\_collection.find\_one({"email": email})

token = generate\_token(str(user["\_id"]))

return jsonify({"message": "Login bem-sucedido!", "token": token}), 200

```

---

#### \*\*7. Rota de Validação de Credenciais da AWS (`POST /api/aws/credentials`)\*\*

```python

@app.route('/api/aws/credentials', methods=['POST'])

def validate\_credentials():

data = request.json

access\_key = data.get("accessKey")

secret\_key = data.get("secretKey")

# Simulação de validação de credenciais

if not access\_key or not secret\_key:

return jsonify({"error": "Credenciais incompletas"}), 400

# Aqui você pode adicionar lógica real para validar as credenciais na AWS

is\_valid = True # Simulação de validação

if not is\_valid:

return jsonify({"error": "Credenciais inválidas"}), 401

return jsonify({"message": "Credenciais válidas"}), 200

```

---

### \*\*Resumo das Rotas Implementadas\*\*

| Método | Rota | Descrição |

|--------|--------------------------|--------------------------------------------|

| POST | `/api/register` | Registrar um novo usuário |

| POST | `/api/login` | Autenticar um usuário |

| POST | `/api/aws/credentials` | Validar credenciais da AWS |

---

### \*\*Próximos Passos\*\*

1. Integrar o frontend com as rotas do backend.

2. Implementar funcionalidades adicionais, como recuperação de senha ou integração com serviços AWS (S3, Lambda, etc.).

3. Adicionar testes automatizados para garantir a qualidade do código.