**Implementação da API com Flask, MongoDB e JWT**

A solução apresentada é uma API desenvolvida com Flask, um framework leve para Python, e MongoDB, um banco de dados NoSQL. A API oferece funcionalidades de registro e login de usuários com autenticação via JWT (JSON Web Token), além de permitir a criação, atualização e exclusão de itens. A seguir, explicamos como a solução foi construída, as tecnologias utilizadas, e discutimos suas vantagens e desvantagens.

**Tecnologias Usadas**

* Flask: Framework minimalista para criar APIs rápidas e simples.
* MongoDB: Banco de dados NoSQL, ideal para armazenar dados flexíveis e escaláveis.
* Flask-JWT-Extended: Extensão para gerenciar autenticação com JWT, garantindo que apenas usuários autenticados possam acessar recursos protegidos.
* Werkzeug: Biblioteca para criar e verificar senhas de forma segura.

**Como Funciona:**

**A API tem as seguintes funcionalidades principais:**

1. Cadastro e Login: Usuários podem se registrar e fazer login, recebendo um token JWT para autenticação.
2. Gestão de Itens: Após logar, os usuários podem criar, atualizar e excluir itens. Apenas quem criou um item pode editá-lo ou removê-lo.

**As rotas principais são:**

* POST /register: Para registrar novos usuários.
* POST /login: Para gerar um token JWT e fazer login.
* GET /profile: Para acessar o perfil do usuário.
* POST /items: Para criar itens.
* PUT /items/<id>: Para atualizar itens (incluindo nome e preço).
* DELETE /items/<id>: Para excluir itens.

**Pontos Fortes**

* Segurança: A autenticação com JWT garante que apenas usuários logados possam acessar as rotas protegidas. Além disso, as senhas dos usuários são armazenadas de forma segura.
* Escalabilidade: MongoDB permite adicionar dados de forma flexível e escalar a solução conforme a necessidade.
* Simplicidade: O Flask oferece uma base simples e fácil de usar, ideal para quem precisa de uma solução rápida.

**Desvantagens**

1. Gerenciamento de Conexões: Flask, por ser minimalista, pode ter dificuldades em gerenciar muitas conexões simultâneas, o que exige otimizações quando a aplicação cresce.
2. Segurança do Token: Como os tokens JWT são armazenados no lado do cliente, é necessário cuidado com vulnerabilidades como XSS.
3. Consistência de Dados: MongoDB, sendo NoSQL, não garante a mesma consistência de bancos SQL, o que pode ser um problema em sistemas que exigem transações fortes.

**Conclusão**

A API é simples, eficiente e escalável, usando Flask e MongoDB para fornecer funcionalidades básicas de gerenciamento de usuários e itens. Embora tenha algumas limitações relacionadas à consistência e segurança, ela é uma boa escolha para aplicações web que exigem uma solução rápida e flexível.

LINK POSTMAN: <https://app.getpostman.com/jointeam?invite_code=e87e2dfa5e7743d5560bac78e0ce61cb>

**Resultado Final – Código para teste:**

from flask import Flask, request, jsonify

from flask\_jwt\_extended import JWTManager, create\_access\_token, jwt\_required, get\_jwt\_identity

from flask\_pymongo import PyMongo

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

from bson import ObjectId

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

# Configuração do JWT

app.config['JWT\_SECRET\_KEY'] = 'secreta' # Defina a chave secreta para o JWT

jwt = JWTManager(app)

# Configuração do MongoDB

app.config["MONGO\_URI"] = "mongodb://localhost:27017/usuarios\_db" # Altere para o seu banco de dados

mongo = PyMongo(app)

# Rota de registro

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

def register():

data = request.get\_json()

if not data:

return jsonify({"msg": "No data provided"}), 400

email = data.get('email')

password = data.get('password')

if not email or not password:

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

# Verificando se o usuário já existe

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

if user:

return jsonify({"msg": "Usuário já existe"}), 400

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

mongo.db.users.insert\_one({"email": email, "password": hashed\_password})

return jsonify({"msg": "Usuário criado com sucesso"}), 201

# Função para realizar login e gerar token

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

def login():

data = request.get\_json()

if not data.get('email') or not data.get('password'):

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

email = data.get('email')

password = data.get('password')

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

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

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

access\_token = create\_access\_token(identity=email)

return jsonify({"access\_token": access\_token}), 200

# Função para acessar o perfil do usuário

@app.route('/profile', methods=['GET'])

@jwt\_required()

def profile():

current\_user = get\_jwt\_identity()

return jsonify({

'msg': 'Perfil acessado com sucesso',

'email': current\_user

})

# Função para criar um novo item (apenas para usuários autenticados)

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

@jwt\_required()

def create\_item():

data = request.get\_json()

item\_name = data.get('name')

if not item\_name:

return jsonify({"msg": "Nome do item é obrigatório"}), 400

item = {

"name": item\_name,

"created\_by": get\_jwt\_identity()

}

item\_id = mongo.db.items.insert\_one(item).inserted\_id

# Retornando o ObjectId do item recém-criado

return jsonify({

"msg": "Item criado com sucesso",

"item\_id": str(item\_id) # Convertendo o ObjectId para string para retorno

}), 201

# Função para atualizar um item (apenas para usuários autenticados)

@app.route('/items/<id>', methods=['PUT'])

@jwt\_required()

def update\_item(id):

try:

item\_id = ObjectId(id)

except:

return jsonify({"msg": "ID do item inválido"}), 400

data = request.get\_json()

item\_name = data.get('name')

item\_price = data.get('price') # Novo campo para o preço

if not item\_name and item\_price is None: # Verificando se pelo menos um campo foi passado

return jsonify({"msg": "Nome do item ou preço são obrigatórios"}), 400

update\_fields = {}

if item\_name:

update\_fields["name"] = item\_name

if item\_price is not None:

update\_fields["price"] = item\_price # Atualizando o preço

# Atualiza o item no banco de dados

result = mongo.db.items.update\_one(

{"\_id": item\_id, "created\_by": get\_jwt\_identity()},

{"$set": update\_fields} # Atualiza apenas os campos que foram passados

)

if result.matched\_count == 0:

return jsonify({"msg": "Item não encontrado ou você não tem permissão para editá-lo"}), 404

return jsonify({"msg": "Item atualizado com sucesso"}), 200

# Função para excluir um item (apenas para usuários autenticados)

@app.route('/items/<id>', methods=['DELETE'])

@jwt\_required()

def delete\_item(id):

try:

item\_id = ObjectId(id)

except:

return jsonify({"msg": "ID do item inválido"}), 400

result = mongo.db.items.delete\_one(

{"\_id": item\_id, "created\_by": get\_jwt\_identity()}

)

if result.deleted\_count == 0:

return jsonify({"msg": "Item não encontrado ou você não tem permissão para excluí-lo"}), 404

return jsonify({"msg": "Item excluído com sucesso"}), 200

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

app.run(debug=True)