at_S9_A3_SL06_backend

Roteiro de Atividade Prática

Nome: _____ Turma: _____

Título da atividade: Segurança em APIs

Objetivo:
Implementar práticas de segurança em APIs, protegendo-as contra-ataques comuns e garantindo que apenas usuários autenticados e autorizados possam acessar ou modificar dados sensíveis.

Conceito técnico: segurança em APIs

As APIs são pontos de entrada para sistemas e, portanto, alvos frequentes de ataques. Implementar práticas de segurança robustas é crucial para proteger os dados e os recursos do servidor. Alguns dos principais conceitos de segurança em APIs incluem:

- Autenticação – Verificar a identidade de um usuário ou sistema que está tentando acessar a API. Os métodos mais comuns incluem JWT (JSON Web Tokens) e OAuth;
- Autorização – Controlar o que um usuário autenticado pode fazer, garantindo que eles possam acessar ou modificar apenas os recursos aos quais têm permissão;
- Rate limiting – Limitar o número de requisições que um cliente pode fazer dentro de um determinado período, protegendo a API contra-ataques de negação de serviço (DoS);
- Criptografia – Garantir que os dados enviados e recebidos pela API estejam criptografados, geralmente utilizando HTTPS;

- Validação de dados – Conferir se os dados enviados pelo cliente são válidos e dentro das expectativas, prevenindo injeções de código ou outros ataques maliciosos.

Objetivo da atividade:

Implementar autenticação e autorização utilizando JWT em uma API Python. A API deve garantir que apenas usuários autenticados possam acessar os dados de produtos e que ações de criação, atualização e remoção sejam restritas a usuários autorizados.

Enunciado:

Você foi encarregado de implementar uma API que proteja o acesso aos dados de produtos de um e-commerce utilizando JWT para autenticação e autorização. A API deve garantir que apenas usuários autenticados possam visualizar os produtos e que apenas administradores possam adicionar, atualizar ou remover produtos. Além disso, é necessário implementar práticas de segurança, como criptografia e validação de dados.

Código de programação:
A seguir está um exemplo de como implementar a autenticação e a autorização utilizando JWT em Python com Flask e PyJWT.

Instalação das bibliotecas:

Antes de rodar o código, instale as seguintes bibliotecas:

pip install Flask PyJWT

Exemplo em Python:

```
from flask import Flask, jsonify, request, make_response
import jwt
import datetime
from functools import wraps

app = Flask(__name__)

# Chave secreta para assinar os tokens JWT
app.config['SECRET_KEY'] = 'minha_chave_secreta'

# Função para verificar o token JWT
def token_requerido(f):
    @wraps(f)
    def decorator(*args, **kwargs):
        token = request.headers.get('x-access-token')
        if not token:
            return jsonify({'mensagem': 'Token de acesso é necessário!'}), 401
        try:
            dados = jwt.decode(token, app.config['SECRET_KEY'], algorithms=["HS256"])
            request.user_data = dados
        except:
            return jsonify({'mensagem': 'Token inválido!'}), 401
        return f(*args, **kwargs)
    return decorator

# Rota para gerar o token JWT (Autenticação)
```

```
@app.route('/login', methods=['POST'])
def login():
    dados = request.get_json()
    if dados['usuario'] == 'admin' and dados['senha'] == '1234':
        token = jwt.encode({
            'usuario': dados['usuario'],
            'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=30)
        }, app.config['SECRET_KEY'])
        return jsonify({'token': token})
    return make_response('Usuário ou senha incorretos!', 401)

# Rota protegida (apenas para usuários autenticados)
@app.route('/produtos', methods=['GET'])
@token_requerido
def listar_produtos():
    produtos = [
        {"id": 1, "nome": "Camiseta", "preco": 50.00},
        {"id": 2, "nome": "Tênis", "preco": 120.00}
    ]
    return jsonify(produtos)

# Rota para adicionar produto (restrito para administradores)
@app.route('/produtos', methods=['POST'])
@token_requerido
def adicionar_produto():
    if request.user_data['usuario'] != 'admin':
        return jsonify({'mensagem': 'Ação não permitida!'}), 403
    novo_produto = request.get_json()
    # Lógica para adicionar o produto
    return jsonify({'mensagem': 'Produto adicionado com sucesso!'}), 201
```

```
# Rota para remover produto (restrito para administradores)
@app.route('/produtos/<int:id>', methods=['DELETE'])
@token_requerido
def remover_produto(id):
    if request.user_data['usuario'] != 'admin':
        return jsonify({'mensagem': 'Ação não permitida!'}), 403
    # Lógica para remover o produto
    return jsonify({'mensagem': 'Produto removido com sucesso!'}), 200

if __name__ == '__main__':
    app.run(debug=True)
```

Explicação técnica:

- **Autenticação com JWT** – A rota/o login garante que o usuário se autentique enviando um nome de usuário e uma senha. Se as credenciais forem válidas, um **JWT** (token) é gerado e enviado para o cliente, que deve incluir esse token em todas as requisições subsequentes;
- **Autorização com JWT** – As rotas protegidas (/produtos, /produtos/<id>) utilizam o decorador @token_requerido para garantir que apenas usuários autenticados possam acessar os dados. Apenas o usuário "admin" pode adicionar ou remover produtos;
- **Validação de tokens** – O token é validado em cada requisição. Se for inválido ou expirado, o acesso é negado.

Perguntas para conclusão da atividade:

- Explique o papel do JWT em APIs e como ele é utilizado para autenticação e autorização.
- Por que é importante validar os dados do token em cada requisição, e o que acontece se o token estiver expirado ou inválido?
- Descreva como a função @wraps e o decorador @token_requerido ajudam a proteger as rotas da API.