Com certeza! Como cientista de dados, compreendo perfeitamente a necessidade de disponibilizar modelos de forma segura e escalável em uma arquitetura orientada a serviços (SOA). A autenticação via token é um pilar fundamental para garantir que apenas aplicações autorizadas consumam os recursos de IA.

A seguir, apresento o código completo de uma aplicação em Python, utilizando o framework **Flask**, que simula a disponibilização de quatro modelos de IA como serviços web, todos protegidos por um token estático.

Estrutura do Projeto

Para organizar o projeto, você precisará de apenas um arquivo.

1. **`app.py`**: Este será o arquivo principal contendo todo o código da nossa aplicação Flask, incluindo a lógica de autenticação e os endpoints dos serviços.

Código Completo da Aplicação (`app.py`)

Crie um arquivo chamado `app.py` e cole o seguinte código dentro dele.

Python

```python
import os
import random
from functools import wraps
from flask import Flask, request, jsonify

# ==============================================================================
# INICIALIZAÇÃO E CONFIGURAÇÃO DA APLICAÇÃO
# ==============================================================================

app = Flask(__name__)

# Token de autenticação estático. Em um ambiente de produção, isso viria
# de uma variável de ambiente ou de um serviço de gerenciamento de segredos.
SECRET_TOKEN = "MEU_TOKEN_SECRETO_12345"

# ==============================================================================
# MECANISMO DE AUTENTICAÇÃO VIA TOKEN (DECORATOR)
# ==============================================================================

def token_required(f):
    """
    Decorator que implementa a validação do token de autenticação.
    Ele verifica se o token está presente no cabeçalho 'Authorization'
    e se corresponde ao token secreto da aplicação.
```

```
    """
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        # O token deve ser passado no formato "Bearer <token>"
        if 'Authorization' in request.headers:
            auth_header = request.headers['Authorization']
            try:
                # Extrai o token da string "Bearer <token>"
                token = auth_header.split(" ")[1]
            except IndexError:
                return jsonify({'message': 'Formato do token inválido! Use
"Bearer <token>".'}), 401

        if not token:
            return jsonify({'message': 'Token de autenticação não
encontrado!'}), 401

        if token != SECRET_TOKEN:
            return jsonify({'message': 'Token inválido ou expirado!'}), 403 #
403 Forbidden

        # Se o token for válido, executa a rota original
        return f(*args, **kwargs)

    return decorated

# ==============================================================================
# SIMULAÇÃO DOS MODELOS DE INTELIGÊNCIA ARTIFICIAL
# ==============================================================================

def simular_predicao_venda(mes, ano):
    """Simula uma predição de vendas baseada no mês e ano."""
    # Lógica de simulação: um valor base mais uma variação aleatória
    base_venda = 100000 + (ano - 2023) * 20000 + mes * 1500
    variacao = random.uniform(0.95, 1.05)
    predicao = base_venda * variacao
    return round(predicao, 2)

def simular_classificacao_cliente(cpf):
    """Simula a classificação de crédito de um cliente a partir do CPF."""
    # Lógica de simulação: a classificação depende do último dígito do CPF
    # para ser determinística para o mesmo CPF.
    if not cpf or not cpf.isdigit() or len(cpf) != 11:
        return "CPF inválido"

    ultimo_digito = int(cpf[-1])
```

```
    if ultimo_digito in [0, 1]:
        return "Risco Alto (D)"
    elif ultimo_digito in [2, 3, 4]:
        return "Risco Moderado (C)"
    elif ultimo_digito in [5, 6, 7]:
        return "Risco Baixo (B)"
    else: # 8, 9
        return "Risco Muito Baixo (A)"

def simular_predicao_demanda(produto_id, periodo):
    """Simula a predição da demanda de um produto."""
    # Lógica de simulação: valor baseado no ID do produto e com variação
    base_demanda = 50 + (produto_id % 100) * 5
    variacao_periodo = len(periodo) * 0.1 # Períodos mais longos tem mais
demanda
    predicao = base_demanda * (1 + variacao_periodo) * random.uniform(0.9,
1.1)
    return int(round(predicao))

def simular_classificacao_sentimento(texto):
    """Simula uma análise de sentimento a partir de um texto."""
    # Lógica de simulação: busca por palavras-chave
    texto = texto.lower()
    palavras_positivas = ["bom", "ótimo", "excelente", "gostei", "incrível",
"maravilhoso", "recomendo"]
    palavras_negativas = ["ruim", "péssimo", "terrível", "odiei",
"decepcionado", "problema"]

    score = 0
    for palavra in palavras_positivas:
        if palavra in texto:
            score += 1
    for palavra in palavras_negativas:
        if palavra in texto:
            score -= 1

    if score > 0:
        return "Positivo"
    elif score < 0:
        return "Negativo"
    else:
        return "Neutro"

# ==============================================================================
# ENDPOINTS DA API (SERVIÇOS WEB)
# ==============================================================================
```

```python
@app.route('/')
def index():
    return "<h1>API de Modelos de IA no ar!</h1><p>Acesse os endpoints via POST e com token de autenticação.</p>"

# --- SERVIÇO 1: PREDIÇÃO DE VENDAS ---
@app.route('/api/v1/predicaoVenda', methods=['POST'])
@token_required
def predicao_venda_service():
    data = request.get_json()
    if not data or 'mes' not in data or 'ano' not in data:
        return jsonify({'message': 'Erro: JSON body deve conter "mes" e "ano".'}), 400

    try:
        mes = int(data['mes'])
        ano = int(data['ano'])
        if not (1 <= mes <= 12 and ano > 2000):
            raise ValueError()
    except (ValueError, TypeError):
        return jsonify({'message': 'Erro: "mes" e "ano" devem ser números válidos.'}), 400

    predicao = simular_predicao_venda(mes, ano)
    return jsonify({
        'servico': 'predicao_venda',
        'input': {'mes': mes, 'ano': ano},
        'predicao_vendas_R$': predicao
    })

# --- SERVIÇO 2: CLASSIFICAÇÃO DE CLIENTE ---
@app.route('/api/v1/classificacaoCliente', methods=['POST'])
@token_required
def classificacao_cliente_service():
    data = request.get_json()
    if not data or 'cpf' not in data:
        return jsonify({'message': 'Erro: JSON body deve conter "cpf".'}), 400

    cpf = data['cpf'].replace('.', '').replace('-', '') # Limpa formatação do CPF
    classificacao = simular_classificacao_cliente(cpf)

    if classificacao == "CPF inválido":
        return jsonify({'message': 'Erro: CPF inválido.'}), 400

    return jsonify({
        'servico': 'classificacao_cliente',
```

```
        'input': {'cpf': data['cpf']},
        'classificacao_credito': classificacao
    })

# --- SERVIÇO 3: PREDIÇÃO DE DEMANDA ---
@app.route('/api/v1/predicaoDemanda', methods=['POST'])
@token_required
def predicao_demanda_service():
    data = request.get_json()
    if not data or 'produto_id' not in data or 'periodo' not in data:
        return jsonify({'message': 'Erro: JSON body deve conter "produto_id" e
"periodo".'}), 400

    try:
        produto_id = int(data['produto_id'])
    except (ValueError, TypeError):
        return jsonify({'message': 'Erro: "produto_id" deve ser um número
inteiro.'}), 400

    periodo = data['periodo']
    predicao = simular_predicao_demanda(produto_id, periodo)
    return jsonify({
        'servico': 'predicao_demanda',
        'input': {'produto_id': produto_id, 'periodo': periodo},
        'predicao_demanda_unidades': predicao
    })

# --- SERVIÇO 4: CLASSIFICAÇÃO DE SENTIMENTO ---
@app.route('/api/v1/classificacaoSentimento', methods=['POST'])
@token_required
def classificacao_sentimento_service():
    data = request.get_json()
    if not data or 'texto' not in data:
        return jsonify({'message': 'Erro: JSON body deve conter "texto".'}),
400

    texto = data['texto']
    if not isinstance(texto, str) or len(texto.strip()) == 0:
        return jsonify({'message': 'Erro: O campo "texto" não pode ser
vazio.'}), 400

    sentimento = simular_classificacao_sentimento(texto)
    return jsonify({
        'servico': 'classificacao_sentimento',
        'input': {'texto': texto},
        'sentimento_classificado': sentimento
    })
```

```
# ============================================================================
# EXECUÇÃO DA APLICAÇÃO
# ============================================================================

if __name__ == '__main__':
    # Usar a porta 8080 para evitar conflitos com outras aplicações
    port = int(os.environ.get('PORT', 8080))
    # '0.0.0.0' torna a aplicação acessível na rede local
    app.run(host='0.0.0.0', port=port, debug=True)
```

Instruções para Execução no Windows

Siga estes passos para configurar e executar a aplicação em seu ambiente Windows.

1. Pré-requisitos

- **Python 3 instalado**: Se você não tiver o Python instalado, baixe-o em python.org. Durante a instalação, **certifique-se de marcar a caixa "Add Python to PATH"**.
- **Terminal**: Você pode usar o **Prompt de Comando (cmd)** ou o **PowerShell**, que já vêm com o Windows.

2. Configuração do Ambiente

Abra o Prompt de Comando (pressione `Win + R`, digite `cmd` e pressione Enter).

a. Crie uma pasta para o projeto:

```
Bash
mkdir projeto_ia_servicos
cd projeto_ia_servicos
```

b. Crie e ative um ambiente virtual (Virtual Environment):

Isso isola as dependências do seu projeto e é uma excelente prática.

```
Bash
python -m venv venv
venv\Scripts\activate
```

Após executar o segundo comando, você verá `(venv)` no início da linha do seu terminal, indicando que o ambiente virtual está ativo.

c. Instale a única dependência necessária (Flask):

```
Bash
pip install Flask
```

3. Salve o Código

Dentro da pasta `projeto_ia_servicos`, crie o arquivo `app.py` com o código fornecido acima. Você pode usar um editor de texto como o Bloco de Notas, VS Code ou qualquer outro de sua preferência.

4. Execute a Aplicação

Com o ambiente virtual ainda ativo (você deve ver `(venv)` no terminal), execute o seguinte comando:

```
Bash
python app.py
```

Você deverá ver uma saída parecida com esta, indicando que o servidor está rodando:

```
* Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://<seu-ip-local>:8080
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: ...
```

A sua API de serviços de IA está no ar e pronta para receber requisições!

5. Testando os Serviços (Endpoints)

Para testar, você precisará de uma ferramenta para fazer requisições HTTP, como o **cURL** (que pode ser usado no PowerShell/cmd) ou o **Postman**. Abaixo estão os exemplos usando cURL.

O token de autenticação que definimos no código é: `MEU_TOKEN_SECRETO_12345`

Exemplo 1: Testando a autenticação (Token Inválido)

```
Bash
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer
TOKEN_ERRADO" -d "{\"mes\": 11, \"ano\": 2025}"
http://127.0.0.1:8080/api/v1/predicaoVenda
```

Resposta esperada (ERRO):

```JSON
{
  "message": "Token inválido ou expirado!"
}
```

Exemplo 2: Serviço de Predição de Venda (Token Correto)

```Bash
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer MEU_TOKEN_SECRETO_12345" -d "{\"mes\": 11, \"ano\": 2025}" http://127.0.0.1:8080/api/v1/predicaoVenda
```

Resposta esperada (SUCESSO):

```JSON
{
  "input": {
    "ano": 2025,
    "mes": 11
  },
  "predicao_vendas_R$": 160431.11,
  "servico": "predicao_venda"
}
```

Exemplo 3: Serviço de Classificação de Cliente

```Bash
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer MEU_TOKEN_SECRETO_12345" -d "{\"cpf\": \"123.456.789-00\"}" http://127.0.0.1:8080/api/v1/classificacaoCliente
```

Resposta esperada (SUCESSO):

```JSON
{
  "classificacao_credito": "Risco Alto (D)",
  "input": {
    "cpf": "123.456.789-00"
  },
  "servico": "classificacao_cliente"
}
```

Exemplo 4: Serviço de Predição de Demanda

```Bash
```

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer MEU_TOKEN_SECRETO_12345" -d "{\"produto_id\": 753, \"periodo\": \"Proxima Quinzena\"}" http://127.0.0.1:8080/api/v1/predicaoDemanda
```

Resposta esperada (SUCESSO):

```
JSON
{
  "input": {
    "periodo": "Proxima Quinzena",
    "produto_id": 753
  },
  "predicao_demanda_unidades": 444,
  "servico": "predicao_demanda"
}
```

Exemplo 5: Serviço de Classificação de Sentimento

```
Bash
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer MEU_TOKEN_SECRETO_12345" -d "{\"texto\": \"Achei o atendimento muito bom e o produto é excelente!\"}" http://127.0.0.1:8080/api/v1/classificacaoSentimento
```

Resposta esperada (SUCESSO):

```
JSON
{
  "input": {
    "texto": "Achei o atendimento muito bom e o produto é excelente!"
  },
  "sentimento_classificado": "Positivo",
  "servico": "classificacao_sentimento"
}
```

Para parar a aplicação, volte ao terminal onde ela está rodando e pressione `Ctrl + C`.