

Machine Learning Engineer

Python para ML e IA

Modelos em API

Leonardo Pena

/ Seja muito bem vindo



OBJETIVO

Construindo a API de
predição com Flask e
JWT



CONEXÃO

Conexão com
modelo_iris.pkl para
inferir classe do Iris



SEGURANÇA

Segurança via token,
além de
armazenamento de
previsões no SQLite

Objetivo dessa parte



Passo 1

Explicar a estrutura geral do script (JWT config, DB config)



Passo 2

Mostrar como carregar modelo_iris.pkl



Passo 3

Implementar endpoints: /login (gera token), /predict (usa token), /predictions (lista histórico)



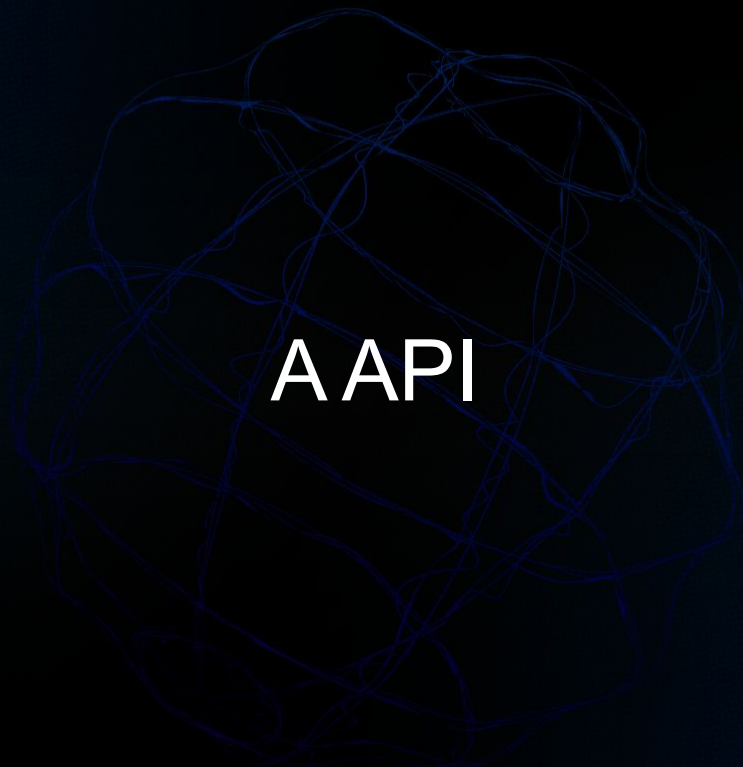
Passo 4

Gerar logs, lidar com cache e tratar erros



Passo 5

Conclusão: API local pronta, rodando python api_modelo.py



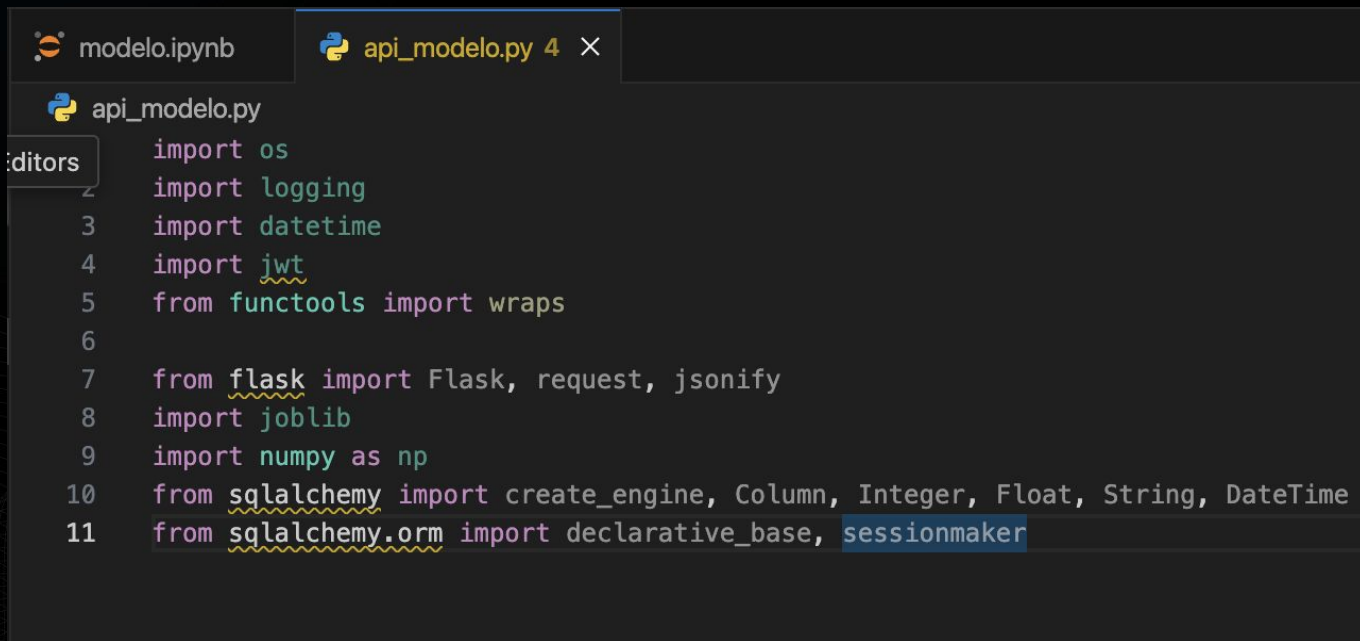


/ O arquivo api_modelo.py

- Vamos montar o arquivo da API, que será alimentado pelo modelo. Com:
 - Configurações do JWT (chave secreta, algoritmo, expiração)
 - SQLAlchemy + declarative_base() para armazenar previsões em "predictions.db"
 - Carregamento do modelo com `joblib.load("modelo_iris.pkl")`
 - Roteamento Flask via `@app.route(...)`
 - Principais rotas: /login, /predict, /predictions, /health

/

Trazendo as bibliotecas necessárias



The image shows a Jupyter Notebook interface with two tabs at the top: 'modelo.ipynb' and 'api_modelo.py 4 x'. The 'api_modelo.py' tab is active, displaying a Python file with the following code:

```
1 import os
2 import logging
3 import datetime
4 import jwt
5 from functools import wraps
6
7 from flask import Flask, request, jsonify
8 import joblib
9 import numpy as np
10 from sqlalchemy import create_engine, Column, Integer, Float, String, DateTime
11 from sqlalchemy.orm import declarative_base, sessionmaker
```

The code imports various libraries: `os`, `logging`, `datetime`, `jwt`, `functools` (for `wraps`), `flask` (for `Flask`, `request`, and `jsonify`), `joblib`, `numpy` (as `np`), and `sqlalchemy` (for `create_engine`, `Column`, `Integer`, `Float`, `String`, `DateTime`, `declarative_base`, and `sessionmaker`). The `sessionmaker` import on line 11 is highlighted with a blue selection box.



/ Preparando ambiente

- Para funcionar as importações anteriores, crie um arquivo requirements.txt com as bibliotecas abaixo. Que depois será instalado com *pip install - requirements.txt*
 - Flask
 - PyJWT
 - joblib
 - numpy
 - SQLAlchemy



/ Utilização de cada biblioteca

- **Flask**
 - Framework web utilizado para criar a API, gerenciar rotas e lidar com requisições e respostas HTTP.
- **PyJWT**
 - Biblioteca usada para criar e verificar tokens JWT, implementando autenticação e autorização na API.
- **joblib**
 - Utilizada para carregar o modelo de machine learning pré-treinado (modelo_iris.pkl) de forma eficiente.
- **NumPy**
 - Biblioteca empregada para manipular e preparar os dados de entrada como arrays numéricos para as previsões do modelo.



/ Utilização de cada biblioteca

- **SQLAlchemy**
 - ORM utilizado para definir o modelo de dados Prediction, interagir com o banco de dados SQLite e realizar operações CRUD.
- **os**
 - Biblioteca padrão do Python usada para interagir com o sistema operacional, como gerenciar variáveis de ambiente.
- **logging**
 - Utilizada para configurar e registrar logs da aplicação, facilitando o monitoramento e depuração.
- **datetime**
 - Biblioteca padrão do Python usada para manipular datas e horários, como definir timestamps para predições.
- **functools (wraps)**
 - Utilizado para criar decorators, como token_required, mantendo as informações da função original.

/

Configurações de JWT e Logs

```
14     JWT_SECRET = "MEUSEGREDOAQUI"
15     JWT_ALGORITHM = "HS256"
16     JWT_EXP_DELTA_SECONDS = 3600
17
18     logging.basicConfig(level=logging.INFO)
19     logger = logging.getLogger("api_modelo")
20
```

/

Configuração de SQLAlchemy

```
20  
21 DB_URL = "sqlite:///predictions.db"  
22 engine = create_engine(DB_URL, echo=False)  
23 Base = declarative_base()  
24 SessionLocal = sessionmaker(bind=engine)  
25
```

Classe Prediction

```
27 class Prediction(Base):
28     __tablename__ = "predictions"
29     id = Column(Integer, primary_key=True, autoincrement=True)
30     sepal_length = Column(Float, nullable=False)
31     sepal_width = Column(Float, nullable=False)
32     petal_length = Column(Float, nullable=False)
33     petal_width = Column(Float, nullable=False)
34     predicted_class = Column(Integer, nullable=False)
35     created_at = Column(DateTime, default=datetime.datetime.utcnow)
36
```

Carregando o Modelo

```
36
37     # Cria as tabelas no banco (em produção utilizar Alembic)
38     Base.metadata.create_all(engine)
39
40     model = joblib.load("modelo_iris.pkl")
41     logger.info("Modelo carregado com sucesso.")
42
```


/ Aplicação Flask

```
43     app = Flask(__name__)  
44     predictions_cache = {}  
45     |
```

/ Autenticação Simples (JWT)

46

47 TEST_USERNAME = "admin"

48 TEST_PASSWORD = "secret"

49

create_token e token_required

```
51 ∨ def create_token(username):
52 ∨     payload = {
53         "username": username,
54         "exp": datetime.datetime.utcnow() + datetime.timedelta(seconds=JWT_EXP_DELTA_SECONDS)
55     }
56     token = jwt.encode(payload, JWT_SECRET, algorithm=JWT_ALGORITHM)
57     return token
58
59 ∨ def token_required(f):
60     @wraps(f)
61     ∨ def decorated(*args, **kwargs):
62         # pegar token do header Authorization: Bearer <token>
63         # decodificar e checar expiração
64         return f(*args, **kwargs)
65     return decorated
66
```

/

Endpoint /login

```
68 @app.route("/login", methods=["POST"])
69 √ def login():
70     data = request.get_json(force=True)
71     username = data.get("username")
72     password = data.get("password")
73 √     if username == TEST_USERNAME and password == TEST_PASSWORD:
74         token = create_token(username)
75         return jsonify({"token": token})
76 √     else:
77         return jsonify({"error": "Credenciais inválidas"}), 401
78
```

Endpoint /predict

```
80
81 @app.route("/predict", methods=["POST"])
82 @token_required
83 def predict():
84     """
85     Endpoint protegido por token para obter predição.
86     Corpo (JSON):
87     {
88         "sepal_length": 5.1,
89         "sepal_width": 3.5,
90         "petal_length": 1.4,
91         "petal_width": 0.2
92     }
93     """
94     data = request.get_json(force=True)
95     try:
96         sepal_length = float(data["sepal_length"])
97         sepal_width = float(data["sepal_width"])
98         petal_length = float(data["petal_length"])
99         petal_width = float(data["petal_width"])
100     except (ValueError, KeyError) as e:
101         logger.error("Dados de entrada inválidos: %s", e)
102         return jsonify({"error": "Dados inválidos, verifique parâmetros"}), 400
103
104     # Verificar se já está no cache
105     features = (sepal_length, sepal_width, petal_length, petal_width)
106     if features in predictions_cache:
107         logger.info("Cache hit para %s", features)
108         predicted_class = predictions_cache[features]
109     else:
110         # Rodar o modelo
111         input_data = np.array([features])
112         prediction = model.predict(input_data)
113         predicted_class = int(prediction[0])
114         # Armazenar no cache
115         predictions_cache[features] = predicted_class
116         logger.info("Cache updated para %s", features)
```


Endpoint /predict

```
116         logger.info('Cache updated para %s', features)
117
118     # Armazenar no banco de dados a predição
119     db = SessionLocal()
120     new_pred = Prediction(
121         sepal_length=sepal_length,
122         sepal_width=sepal_width,
123         petal_length=petal_length,
124         petal_width=petal_width,
125         predicted_class=predicted_class
126     )
127     db.add(new_pred)
128     db.commit()
129     db.close()
130
131     return jsonify({"prediction": predicted_class})
132
```

/

Lógica /predict

```
features = (sepal_length, sepal_width, petal_length, petal_width)
if features in predictions_cache:
    predicted_class = predictions_cache[features]
else:
    input_data = np.array([features])
    prediction = model.predict(input_data)
    predicted_class = int(prediction[0])
    predictions_cache[features] = predicted_class

# Armazenar em DB
db = SessionLocal()
new_pred = Prediction(...)
db.add(new_pred)
db.commit()
db.close()
```

Endpoint /predictions

```
134 @app.route("/predictions", methods=["GET"])
135 @token_required
136 def list_predictions():
137     """
138     Lista as predições armazenadas no banco.
139     Parâmetros opcionais (via query string):
140     - limit (int): quantos registros retornar, padrão 10
141     - offset (int): a partir de qual registro começar, padrão 0
142     Exemplo:
143     /predictions?limit=5&offset=10
144     """
145     limit = int(request.args.get("limit", 10))
146     offset = int(request.args.get("offset", 0))
147     db = SessionLocal()
148     preds = db.query(Prediction).order_by(Prediction.id.desc()).limit(limit).offset(offset).all()
149     db.close()
150     results = []
151     for p in preds:
152         results.append({
153             "id": p.id,
154             "sepal_length": p.sepal_length,
155             "sepal_width": p.sepal_width,
156             "petal_length": p.petal_length,
157             "petal_width": p.petal_width,
158             "predicted_class": p.predicted_class,
159             "created_at": p.created_at.isoformat()
160         })
161     return jsonify(results)
```

/

E pra fechar

```
164  ∨ if __name__ == "__main__":  
165      |     app.run(debug=True)
```

Testando após rodar: login

The screenshot displays the HTTP Live API interface for a 'New Request'. The request is a POST to 'http://127.0.0.1:5000/login'. The body is in JSON format, containing 'username': 'admin' and 'password': 'secret'. The response is a 200 OK status, received in 23 ms with a body size of 316 B. The response body is shown in 'Pretty' JSON format, containing a 'token' field with a long alphanumeric string.

Live - API / New Request

POST http://127.0.0.1:5000/login

Params Auth Headers (9) Body Scripts Settings

raw JSON

```
1 {
2   "username": "admin",
3   "password": "secret"
4 }
5
```

Body 200 OK • 23 ms • 316 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaXNjaXhwIjozNzYyMzA3Mzk0fQ.TFLsIYCBWrh840jGA1wdiQNSZ6GZC FdR01C2z4 0Ic"
```


/

Testando após rodar: predict

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:5000/predict`
- Method:** `POST`
- Body (JSON):**

```
{  "sepal_length": 5.1,  "sepal_width": 3.5,  "petal_length": 1.4,  "petal_width": 0.2}
```
- Response:** `200 OK` (27 ms, 188 B)
- Response Body (Pretty):**

```
{  "prediction": 0}
```

/

Testando após rodar: banco criado



o.py × predictions.db × requirements.txt

~/.Documents/FIAP/MLET/material_plataforma/aula6-new/api_modelo.py

Filter 1 rows... Upgrade to PRO

	id	sepal_le...	sepal_wi...	petal_le...	petal_wi...	predice...	created_at
1	1	5.1	3.5	1.4	0.2	0	2025-01-12 17:44:57.853696
+	2						

POSTECH

FIAP + alura