Perfeito — vou fornecer uma aplicação completa em Python (FastAPI) que expõe os quatro serviços solicitados, protegidos por autenticação via token. Os modelos são *simulados* (determinísticos: mesma entrada → mesma saída). Código, dependências e instruções de execução no Windows estão abaixo.

Estrutura proposta (arquivos)

- `requirements.txt`
- `main.py` — API + rotas
- `auth.py` — validação de token
- `models_sim.py` — simulações dos modelos (predição, classificação, sentimento)
- `README.md` (instruções rápidas — incluídas abaixo)

`requirements.txt`

```
fastapi==0.95.2
uvicorn[standard]==0.22.0
pydantic==1.10.9
```

`auth.py`

```
# auth.py
from fastapi import HTTPException, Header, status, Depends
from typing import Optional

# Tokens válidos (em produção trocar por store seguro)
VALID_TOKENS = {
    "secrettoken123",     # token de exemplo
    "reportingtoken456"
}

def get_current_token(authorization: Optional[str] = Header(None)):
    """
    Espera header: Authorization: Bearer <token>
    """
```

```
    if not authorization:
        raise
HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Missing Authorization header")
    parts = authorization.split()
    if len(parts) != 2 or parts[0].lower() != "bearer":
        raise
HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid Authorization header format. Use: Bearer <token>")
    token = parts[1]
    if token not in VALID_TOKENS:
        raise
HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid or expired token")
    # Retornamos o token como "current user" simples
    return token

# Dependency para usar nas rotas:
def require_token(token: str = Depends(get_current_token)):
    return token
```

models_sim.py

```
# models_sim.py
from typing import Tuple, Dict, Any
import hashlib
import math
from datetime import datetime
import re

def _seed_from_args(*args) -> int:    joined = "|".join(str(a) for a
in args)
    h = hashlib.sha256(joined.encode("utf-8")).hexdigest()
    # converte parte do hash em int
    return int(h[:16], 16)

def _confidence_from_seed(seed: int, low=0.6, high=0.98) -> float:
    # normaliza determinístico entre low e high
    r = (seed % 10000) / 10000.0
```

```
    return round(low + (high - low) * r, 3)

# Predição de vendas: mês (1-12) e ano (YYYY)
def predicao_venda(mes: int, ano: int) -> Dict[str, Any]:
    seed = _seed_from_args("predicao_venda", mes, ano)
    # base mensal aleatória determinística
    base = ((ano % 100) * 1000) + (mes * 200) + (seed % 500)
    # adiciona sazonalidade simples (dezembro +20%, jan -10%, jul
+10%)
    saz = 1.0
    if mes == 12:
        saz = 1.20
    elif mes == 1:
        saz = 0.90
    elif mes == 7:
        saz = 1.10
    predicted = int(base * saz)
    conf = _confidence_from_seed(seed)
    return {
        "model": "predicaoVenda_sim",
        "mes": mes,
        "ano": ano,
        "predicted_sales": predicted,
        "confidence": conf,
        "generated_at": datetime.utcnow().isoformat() + "Z"
    }

# Validação simples de CPF (algoritmo oficial)
def _clean_digits(s: str) -> str:
    return "".join(ch for ch in s if ch.isdigit())

def validate_cpf(cpf: str) -> bool:
    cpf = _clean_digits(cpf)
    if len(cpf) != 11 or cpf == cpf[0] * 11:
        return False
    def calc(digs):
        s = sum(int(a) * b for a, b in zip(digs, range(len(digs)+1,
1, -1)))
        r = (s * 10) % 11
        return r if r < 10 else 0
    first = calc(cpf[:9])
    second = calc(cpf[:9] + str(first))
```

```
    return cpf[-2:] == f"{first}{second}"

# Classificação de crédito por CPF (simulada determinística)
def classificacao_cliente(cpf: str) -> Dict[str, Any]:
    clean = _clean_digits(cpf)
    valid = validate_cpf(clean)
    seed = _seed_from_args("classificacao_cliente", clean)
    score = seed % 1000  # 0..999
    # mapear para categorias simples
    if score >= 800:
        cat = "A"
        risk = "Baixo"
    elif score >= 600:
        cat = "B"
        risk = "Moderado"
    elif score >= 400:
        cat = "C"
        risk = "Alto"
    else:
        cat = "D"
        risk = "Muito Alto"
    conf = _confidence_from_seed(seed)
    return {
        "model": "classificacaoCliente_sim",
        "cpf": cpf,
        "valid_cpf": valid,
        "score": int(score),
        "category": cat,
        "risk_level": risk,
        "confidence": conf,
        "generated_at": datetime.utcnow().isoformat() + "Z"
    }

# Predição de demanda por produto e periodo
# period: "YYYY-MM" or "YYYY-MM:YYYY-MM"
def _parse_period(period: str):
    if ":" in period:
        parts = period.split(":")
        if len(parts) != 2:
            raise ValueError("period must be YYYY-MM or YYYY-
MM:YYYY-MM")
        start, end = parts
```

```
        return start, end
    else:
        return period, period

def _months_between(start: str, end: str):
    # start/end as YYYY-MM
    y1, m1 = map(int, start.split("-"))
    y2, m2 = map(int, end.split("-"))
    months = (y2 - y1) * 12 + (m2 - m1) + 1
    if months < 1:
        raise ValueError("end must be after or equal to start")
    return months

def predicao_demanda(product_id: str, period: str) -> Dict[str,
Any]:
    start, end = _parse_period(period)
    months = _months_between(start, end)
    seed = _seed_from_args("predicao_demanda", product_id, start,
end)
    # base mensal dependente do product_id hash
    base_unit = 50 + (seed % 200)  # 50..249
    # add small variation across months deterministically
    total = 0
    monthly = []
    for i in range(months):
        s = (seed + i * 97) % 1000
        qty = int(base_unit * (0.8 + (s % 41) / 100.0))  # 0.8..1.2
        monthly.append(qty)
        total += qty
    conf = _confidence_from_seed(seed)
    return {
        "model": "predicaoDemanda_sim",
        "product_id": product_id,
        "period": period,
        "months": months,
        "monthly_estimate": monthly,
        "total_estimate": total,
        "confidence": conf,
        "generated_at": datetime.utcnow().isoformat() + "Z"
    }

# Classificação de sentimento (simples lexicon)
```

```
_POS = {"bom", "ótimo", "otimo", "excelente", "gostei", "adorei",
"satisfeito", "fantástico", "positivo", "feliz", "maravilhoso"}
_NEG = {"ruim", "péssimo", "pessimo", "detestei", "ódio", "odio",
"insatisfeito", "horrível", "horrivel", "negativo", "triste"}

def classificacao_sentimento(text: str) -> Dict[str, Any]:
    txt = text.lower()
    # token simples
    words = re.findall(r"\w+", txt, flags=re.UNICODE)
    pos = sum(1 for w in words if w in _POS)
    neg = sum(1 for w in words if w in _NEG)
    raw_score = pos - neg  # integer
    # normaliza entre -1 e 1
    if pos + neg == 0:
        score = 0.0
    else:
        score = (raw_score) / (pos + neg)
    # mapa em etiqueta
    if score > 0.3:
        label = "positive"
    elif score < -0.3:
        label = "negative"
    else:
        label = "neutral"
    # confidence depends on number of sentiment words
    conf = round(min(0.99, 0.5 + 0.1 * (pos + neg)), 3)
    return {
        "model": "classificacaoSentimento_sim",
        "text": text,
        "pos_count": pos,
        "neg_count": neg,
        "score": round(score, 3),
        "label": label,
        "confidence": conf,
        "generated_at": datetime.utcnow().isoformat() + "Z"
    }
```

main.py

```
# main.py
from fastapi import FastAPI, Depends, HTTPException, status
from pydantic import BaseModel, Field
from typing import Optional
from auth import require_token
import models_sim as sim

app = FastAPI(title="IA-as-a-Service (simulado)", version="1.0")

# --- Request / Response models ---
class PredicaoVendaRequest(BaseModel):
    mes: int = Field(..., ge=1, le=12, example=9)
    ano: int = Field(..., ge=1900, le=3000, example=2025)

class ClassificacaoClienteRequest(BaseModel):
    cpf: str = Field(..., example="123.456.789-09")

class PredicaoDemandaRequest(BaseModel):
    product_id: str = Field(..., example="SKU-9876")
    period: str = Field(..., example="2025-09" ) # or "2025-06:2025-
09"

class ClassificacaoSentimentoRequest(BaseModel):
    text: str = Field(..., example="O produto foi ótimo, adorei!")

# --- Endpoints (protegidos) ---
@app.post("/predicaoVenda")
def predicao_venda(req: PredicaoVendaRequest, token: str =
Depends(require_token)):
    try:
        result = sim.predicao_venda(req.mes, req.ano)
        return result
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/classificacaoCliente")
def classificacao_cliente(req: ClassificacaoClienteRequest, token:
str = Depends(require_token)):
    try:
        result = sim.classificacao_cliente(req.cpf)
```

```
        return result
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/predicaoDemanda")
def predicao_demanda(req: PredicaoDemandaRequest, token: str =
Depends(require_token)):
    try:
        result = sim.predicao_demanda(req.product_id, req.period)
        return result
    except ValueError as ve:
        raise HTTPException(status_code=400, detail=str(ve))
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/classificacaoSentimento")
def classificacao_sentimento(req: ClassificacaoSentimentoRequest,
token: str = Depends(require_token)):
    try:
        result = sim.classificacao_sentimento(req.text)
        return result
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

# rota simples para checar status (também protegida)
@app.get("/health")
def health(token: str = Depends(require_token)):
    return {"status": "ok", "time":
__import__("datetime").datetime.utcnow().isoformat() + "Z"}
```

Instruções para executar no Windows

1. **Instalar Python 3.10+**

Verifique: `python --version` (no PowerShell ou CMD). Se não tiver, instale de https://www.python.org/downloads/.

2. **Criar pasta do projeto** (ex.: `c:\projects\ia_service`) e dentro dela crie os arquivos acima (`main.py`, `auth.py`, `models_sim.py`, `requirements.txt`).
3. **Criar e ativar ambiente virtual** (PowerShell):

```
cd C:\projects\ia_service
python -m venv .venv
.\.venv\Scripts\Activate.ps1
```

(Se `Activate.ps1` não executar por política do PowerShell, rode `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser` como admin ou use `.\.venv\Scripts\activate.bat` no CMD.)

4. **Instalar dependências**

```
pip install --upgrade pip
pip install -r requirements.txt
```

5. **Executar a API**

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

   a. A API ficará disponível em http://127.0.0.1:8000.
   b. Documentação automática: http://127.0.0.1:8000/docs (Swagger UI).

Atenção: você ainda precisará colocar o header `Authorization: Bearer <token>` nas requisições.

6. **Tokens de exemplo** (definidos em `auth.py`):
   a. `secrettoken123`
   b. `reportingtoken456`

Em produção substitua por um sistema seguro (OAuth, JWT com expiração, storage em DB, etc).

Exemplos de requisições (PowerShell e curl)

PowerShell (Invoke-RestMethod) — exemplo `predicaoVenda`:

```
$headers = @{ Authorization = "Bearer secrettoken123" }
$body = @{ mes = 12; ano = 2025 } | ConvertTo-Json
Invoke-RestMethod -Uri "http://127.0.0.1:8000/predicaoVenda" -Method
```

```
POST -Headers $headers -Body $body -ContentType "application/json"
```

curl:

```
curl -X POST "http://127.0.0.1:8000/predicaoDemanda" \
 -H "Authorization: Bearer secrettoken123" \
 -H "Content-Type: application/json" \
 -d '{"product_id":"SKU-001","period":"2025-09:2025-11"}'
```

Sentimento:

```
curl -X POST "http://127.0.0.1:8000/classificacaoSentimento" \
 -H "Authorization: Bearer secrettoken123" \
 -H "Content-Type: application/json" \
 -d '{"text":"O produto foi ótimo, adorei! Muito satisfeito."}'
```

Classificação cliente (CPF):

```
curl -X POST "http://127.0.0.1:8000/classificacaoCliente" \
 -H "Authorization: Bearer secrettoken123" \
 -H "Content-Type: application/json" \
 -d '{"cpf":"123.456.789-09"}'
```

Observações e próximos passos (sugestões rápidas)

- Atualmente a autenticação é uma verificação simples de token em memória. Em produção:
  - use JWT com expiração + revogação ou OAuth2.
  - registre tokens no DB ou serviço de identidade.
- Para conectar modelos reais:
  - implemente camada `ModelAdapter` que chame modelos locais (PyTorch/TensorFlow) ou serviços (Sagemaker, endpoints HTTP).
- Logs, métricas e rate-limiting são recomendados (Prometheus, Grafana, FastAPI middleware).
- Documentação já disponível via `/docs`. Você pode testar a API usando o Swagger UI e inserindo o header `Authorization` (clicar no botão Authorize no Swagger UI).

Se quiser, eu:

- adapto para usar JWT com refresh tokens e rota de login;
- adiciono um exemplo de Dockerfile + docker-compose;
- escrevo testes unitários (pytest) para cada endpoint;
- integro um armazenamento simples (SQLite) para tokens e auditoria.

Qual desses prefere que eu adicione agora? (Se quiser já faço sem perguntar muitas opções, eu implemento o mais comum: JWT + rota /login).