## 01_BALANCEADOR.py

```
# FortesIA - Balanceador Inteligente de Tarefas

import random
import time
import threading
from queue import Queue

# Endpoints simulados por categoria
WORKERS = {
    "codigo": ["http://localhost:8000"],
    "imagem": ["http://localhost:8001"],
    "texto":  ["http://localhost:8002"],
    "video":  ["http://localhost:8003"]
}

fila_tarefas = Queue()

def classificar_tarefa(prompt):
    prompt = prompt.lower()
    if any(x in prompt for x in ["python", "algoritmo", "cdigo"]):
        return "codigo"
    elif any(x in prompt for x in ["imagem", "foto", "desenho"]):
        return "imagem"
    elif any(x in prompt for x in ["vdeo", "trailer", "cena"]):
        return "video"
    return "texto"

def executor():
    while True:
        prompt = fila_tarefas.get()
        categoria = classificar_tarefa(prompt)
        destino = random.choice(WORKERS.get(categoria, WORKERS["texto"]))
        print(f"[Balancer] Roteando: {prompt[:60]}  {destino}")
        time.sleep(1)  # Simula chamada
        fila_tarefas.task_done()

for _ in range(4):
    threading.Thread(target=executor, daemon=True).start()

if __name__ == "__main__":
    prompts = [
        "Cdigo em Python para somar dois nmeros",
        "Imagem de um drago azul",
        "Vdeo explicando IA",
        "O que  inteligncia artificial?"
    ]
    for p in prompts:
        fila_tarefas.put(p)
    fila_tarefas.join()
```

## 02_INFERENCIA_LOCAL.py

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
from peft import PeftModel

MODEL_NAME = "deepseek-ai/deepseek-llm-1.3b-base"
CHECKPOINT_PATH = "fortesia_lora_checkpoints"
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
base_model = AutoModelForCausalLM.from_pretrained(MODEL_NAME).to(DEVICE)
model = PeftModel.from_pretrained(base_model, CHECKPOINT_PATH)
model = model.merge_and_unload()
model.eval()

def gerar_resposta(prompt, contexto=""):
    entrada = f"<|user|> {prompt}\n<|assistant|>"
    if contexto:
        entrada = contexto + "\n\n" + entrada
    inputs = tokenizer(entrada, return_tensors="pt").to(DEVICE)
    with torch.no_grad():
        saida = model.generate(**inputs, max_new_tokens=200)
    return tokenizer.decode(saida[0], skip_special_tokens=True).split("<|assistant|>")[-1].strip()

if __name__ == "__main__":
    print(gerar_resposta("Explique recurso em Python"))
```

## 03_FRONTEND.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8" />
  <title>FortesIA Interface Web</title>
</head>
<body>
  <h1>FortesIA</h1>
  <textarea id="prompt"></textarea><br/>
  <button onclick="consultarIA()">Enviar</button>
  <div id="output"></div>
  <div id="feedback" style="display:none;">
    <p>Feedback:</p>
    <button onclick="feedback(1.0)"></button>
    <button onclick="feedback(0.5)"></button>
    <button onclick="feedback(0.0)"></button>
  </div>

<script>
let respostaId = null;
async function consultarIA() {
  const prompt = document.getElementById("prompt").value;
  document.getElementById("output").innerText = "Processando...";
  const res = await fetch("http://localhost:8000/consultar", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify({prompt})
  });
  const data = await res.json();
  respostaId = data.id;
  document.getElementById("output").innerText = data.resposta;
  document.getElementById("feedback").style.display = "block";
}
async function feedback(score) {
  await fetch("http://localhost:8000/feedback", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify({id: respostaId, score})
  });
  alert("Obrigado pelo feedback!");
}
</script>
</body>
</html>
```

## 04_BACKEND.py

```
# FortesIA - Backend Principal com FastAPI
# Inclui: CORS, banco SQLite, LoRA local, fallback GPT-4, memria vetorial, logging e feedback

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
import httpx
import sqlite3
import uuid
import time
import asyncio
import json
import faiss
import numpy as np
from sentence_transformers import SentenceTransformer
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
from peft import PeftModel

# Inicializa app FastAPI
app = FastAPI(title="FortesIA - Backend Orquestrador")

# CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# SQLite
conn = sqlite3.connect("fortesia_logs.db", check_same_thread=False)
cursor = conn.cursor()
cursor.execute('''
    CREATE TABLE IF NOT EXISTS logs (
        id TEXT PRIMARY KEY,
        prompt TEXT,
        ia_usada TEXT,
        resposta TEXT,
        timestamp INTEGER,
        score REAL DEFAULT 0.0
    )
''')
conn.commit()

# Estruturas
class PromptInput(BaseModel):
    prompt: str
class FeedbackInput(BaseModel):
    id: str
    score: float
```

```
# API Keys
API_KEYS = {
    "gpt4": "sk-COLOQUE_SUA_CHAVE_OPENAI",
    "deepseek": "ds-simulada",
    "gemini": "gm-simulada"
}

# Vetorial
index = faiss.read_index("fortesia_data/fortesia_faiss.index")
with open("fortesia_data/fortesia_metadata.json", "r", encoding="utf-8") as f:
    metadados = json.load(f)
model_embed = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")

# LoRA
try:
    MODEL_NAME = "deepseek-ai/deepseek-llm-1.3b-base"
    CHECKPOINT_PATH = "fortesia_lora_checkpoints"
    tokenizer_local = AutoTokenizer.from_pretrained(MODEL_NAME)
        base_model  =  AutoModelForCausalLM.from_pretrained(MODEL_NAME,  torch_dtype=torch.float16,
device_map="auto")
    model = PeftModel.from_pretrained(base_model, CHECKPOINT_PATH)
    model = model.merge_and_unload()
    model.eval()
    DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
    usar_modelo_local = True
except Exception as e:
    print(f"[Erro ao carregar modelo local: {e}]")
    usar_modelo_local = False

def recuperar_contexto_vetorial(prompt: str, top_k: int = 3):
    emb = model_embed.encode(prompt, normalize_embeddings=True)
    emb = np.array([emb]).astype("float32")
    D, I = index.search(emb, top_k)
    return [metadados[i] for i in I[0] if i < len(metadados)]

def escolher_ia(prompt):
    if usar_modelo_local:
        return "fortesia_local"
    elif any(x in prompt.lower() for x in ["cdigo", "python", "script"]):
        return "deepseek"
    elif any(x in prompt.lower() for x in ["imagem", "vdeo", "desenho"]):
        return "gemini"
    else:
        return "gpt4"

async def consultar_fortesia_local(prompt: str, contexto: list) -> str:
     exemplos = "\n\n".join([f"<|user|> {ex['prompt']}\n<|assistant|> {ex['response']}" for ex in
contexto])
    entrada = exemplos + f"\n\n<|user|> {prompt}\n<|assistant|>"
    inputs = tokenizer_local(entrada, return_tensors="pt").to(DEVICE)
    with torch.no_grad():
                saida  =  model.generate(**inputs,  max_new_tokens=200,  temperature=0.7,  top_p=0.9,
do_sample=True, repetition_penalty=1.2)
                                                    return            tokenizer_local.decode(saida[0],
skip_special_tokens=True).split("<|assistant|>")[-1].strip()
```

```
async def consultar_gpt4(prompt: str, contexto: list) -> str:
      exemplos = "\n\n".join([f"Usurio: {ex['prompt']}\nAssistente: {ex['response']}" for ex in
contexto])
    prompt_final = f"Contexto:\n{exemplos}\n\nAgora, responda:\n{prompt}"
    headers = {"Authorization": f"Bearer {API_KEYS['gpt4']}", "Content-Type": "application/json"}
    body = {"model": "gpt-4", "messages": [{"role": "user", "content": prompt_final}]}
    async with httpx.AsyncClient() as client:
            r = await client.post("https://api.openai.com/v1/chat/completions", headers=headers,
json=body)
        if r.status_code == 200:
            return r.json()["choices"][0]["message"]["content"]
        return f"[Erro na chamada GPT-4: {r.status_code}]"

async def consultar_mock(ia: str, prompt: str) -> str:
    await asyncio.sleep(1)
    return f"[Resposta simulada de {ia.upper()} para: {prompt}]"

async def consultar_ia(ia: str, prompt: str) -> str:
    contexto = recuperar_contexto_vetorial(prompt)
    if ia == "fortesia_local":
        return await consultar_fortesia_local(prompt, contexto)
    elif ia == "gpt4":
        return await consultar_gpt4(prompt, contexto)
    else:
        return await consultar_mock(ia, prompt)

@app.post("/consultar")
async def consultar(prompt_input: PromptInput):
    prompt = prompt_input.prompt
    ia_escolhida = escolher_ia(prompt)
    resposta = await consultar_ia(ia_escolhida, prompt)
    log_id = str(uuid.uuid4())
    timestamp = int(time.time())
     cursor.execute("INSERT INTO logs VALUES (?, ?, ?, ?, ?, ?)", (log_id, prompt, ia_escolhida,
resposta, timestamp, 0.0))
    conn.commit()
    return {"ia_usada": ia_escolhida, "resposta": resposta, "id": log_id}

@app.post("/feedback")
def registrar_feedback(feedback: FeedbackInput):
    cursor.execute("UPDATE logs SET score = ? WHERE id = ?", (feedback.score, feedback.id))
    conn.commit()
    return {"status": "Feedback registrado com sucesso"}

@app.get("/logs")
def listar_logs():
    cursor.execute("SELECT * FROM logs ORDER BY timestamp DESC LIMIT 10")
    return cursor.fetchall()
```