Jour 2 TP3 : API TTS (Coqui)
en Python/Flask → consommée par .NET

Parcours : Assistant vocal multimodal (.NET + Python + React)
Durée estimée : 1h – 1h30

Objectif pédagogique

- Déployer une **API Flask** qui synthétise du texte en **audio WAV** avec **Coqui TTS**.
- Exposer 2 endpoints :
  - POST /tts/wav → renvoie un **flux WAV** (audio/wav)
  - GET /health → état du service
- Écrire un **proxy .NET (Minimal API)** qui appelle l'API Flask et renvoie soit l'audio brut, soit du **base64**.

Pré-requis

- Python 3.10+
- .NET 8/9
- **Modèle Coqui TTS** : tts_models/fr/css10/vits (FR)
- GPU optionnel (CUDA) — sinon CPU 📦 Installation (Python)

💡 Sur certaines plateformes, **torch** doit être installé séparément (CPU/GPU).
Si besoin, voir la page PyTorch pour la bonne roue (CPU/GPU).
Ensuite :

```
python -m venv tts_env
tts_env\Scripts\activate   # (Linux/Mac: source tts_env/bin/activate)

pip install flask flask-cors pydantic "pydantic[email]" soundfile numpy
pip install TTS            # Coqui TTS (installera aussi de nombreuses dépendances)
# (Option) GPU: installe torch/cuXX adapté à ton GPU avant TTS si nécessaire
```

Flask API — Coqui TTS

Crée un fichier **app.py** :

```
# app.py
import os, io, tempfile
from flask import Flask, request, Response, jsonify
from flask_cors import CORS
from pydantic import BaseModel, ValidationError
import torch
import numpy as np
import soundfile as sf

# Modèle français VITS (open-source)
MODEL_NAME = "tts_models/fr/css10/vits"
# (Payant) Multilingue XTTS v2 :
# MODEL_NAME = "tts_models/multilingual/multi-dataset/xtts_v2"

# Lazy import de Coqui TTS
TTS = None
tts = None
device = "cuda" if torch.cuda.is_available() else "cpu"

app = Flask(__name__)
CORS(app)  # autorise les appels cross-origin si front local

class TTSRequest(BaseModel):
    text: str
    speaker_wav: str | None = None
    speaker: str | None = None
    speed: float | None = None     # 1.0 = normal

def ensure_model_loaded():
    global TTS, tts
    if tts is not None:
        return
    print(f"Loading Coqui TTS model '{MODEL_NAME}' on {device} ...")
    from TTS.api import TTS as _TTS
    TTS = _TTS
    tts = TTS(MODEL_NAME).to(device)
    print("Model ready.")

@app.route("/health", methods=["GET"])
def health():
    return jsonify({"ready": tts is not None, "model": MODEL_NAME, "device": device})

@app.route("/tts/wav", methods=["POST"])
def tts_wav():
    try:
        ensure_model_loaded()
        # JSON strict
        data = request.get_json(silent=True)
        if not data:
            return jsonify({"error": "JSON body required"}), 400
```

```
        try:
            req = TTSRequest(**data)
        except ValidationError as ve:
            return jsonify({"error": "Invalid payload", "details": ve.errors()}), 400

        txt = (req.text or "").strip()
        if not txt:
            return jsonify({"error": "`text` is required"}), 400

        with tempfile.TemporaryDirectory() as td:
            out_path = os.path.join(td, "out.wav")

            if req.speaker_wav:
                # Voix clonée à partir d'un WAV de référence (si supporté par le modèle)
                tts.tts_to_file(
                    text=txt,
                    speaker_wav=req.speaker_wav,
                    file_path=out_path,
                    speed=req.speed
                )
            else:
                # Voix par défaut du modèle
                tts.tts_to_file(
                    text=txt,
                    speaker=req.speaker,  # souvent None pour css10/vits
                    file_path=out_path,
                    speed=req.speed
                )

            with open(out_path, "rb") as f:
                wav = f.read()

        # Retourne l'audio brut (audio/wav)
        return Response(wav, mimetype="audio/wav", headers={
            "X-Engine": "coqui-tts",
            "X-Model": MODEL_NAME,
            "X-Device": device
        })
    except Exception as e:
        print("TTS error:", e)
        return jsonify({"error": str(e)}), 500

if __name__ == "__main__":
    # Pour dev local (ne pas utiliser en prod)
    app.run(host="0.0.0.0", port=5055, debug=False)
```

Lancer l’API

```
python app.py
# écoute sur http://localhost:5055
```

Test rapide (cURL)

```
curl -X POST http://localhost:5055/tts/wav \
  -H "Content-Type: application/json" \
  -d "{\"text\":\"Bonjour, je suis votre assistant vocal.\", \"speed\":1.0}" \
  --output tts_out.wav
```

.NET - Consommer l’API Flask

Dans un projet Minimal API (.NET 8/9), ajoute un **endpoint proxy**.
Exemple d’un **DTO** et de deux routes : **WAV pass-through** et **JSON base64**.

```
// Program.cs (extraits)
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

const string TTS_ENDPOINT = "http://localhost:5055";

// DTOs
public record TtsProxyRequest(
    [property: JsonPropertyName("text")] string Text,
    [property: JsonPropertyName("speaker_wav")] string? SpeakerWav = null,
    [property: JsonPropertyName("speaker")] string? Speaker = null,
    [property: JsonPropertyName("speed")] float? Speed = null
);

public record TtsProxyResponse(string ContentType, string Base64Audio);

// ========== WAV PASS-THROUGH ==========
app.MapPost("/python-tts/wav", async (TtsProxyRequest dto) =>
{
    if (string.IsNullOrWhiteSpace(dto.Text))
        return Results.BadRequest("`text` is required.");

    using var http = new HttpClient { BaseAddress = new Uri(TTS_ENDPOINT) };

    var reqBody = new
    {
        text = dto.Text,
        speaker_wav = dto.SpeakerWav,
        speaker = dto.Speaker,
        speed = dto.Speed
    };

    var resp = await http.PostAsJsonAsync("/tts/wav", reqBody);
```

```
    if (!resp.IsSuccessStatusCode)
        return Results.Problem($"Python TTS error: {(int)resp.StatusCode} {resp.ReasonPhrase}");

    var wav = await resp.Content.ReadAsByteArrayAsync();
    // Retourne un flux audio directement (à jouer côté front)
    return Results.File(wav, "audio/wav");
})
.WithName("PythonTtsWav");

// ========== JSON (Base64) ==========
app.MapPost("/python-tts/base64", async (TtsProxyRequest dto) =>
{
    if (string.IsNullOrWhiteSpace(dto.Text))
        return Results.BadRequest("`text` is required.");

    using var http = new HttpClient { BaseAddress = new Uri(TTS_ENDPOINT) };

    var reqBody = new
    {
        text = dto.Text,
        speaker_wav = dto.SpeakerWav,
        speaker = dto.Speaker,
        speed = dto.Speed
    };

    var resp = await http.PostAsJsonAsync("/tts/wav", reqBody);
    if (!resp.IsSuccessStatusCode)
        return Results.Problem($"Python TTS error: {(int)resp.StatusCode} {resp.ReasonPhrase}");

    var wav = await resp.Content.ReadAsByteArrayAsync();
    var b64 = Convert.ToBase64String(wav);
    return Results.Ok(new TtsProxyResponse("audio/wav", b64));
})
.WithName("PythonTtsBase64");

app.Run();
```