Contexte

Une PME souhaite automatiser la création d'études de marché pour ses produits en fonction des tendances et des avis clients disponibles en ligne. L'objectif de ce TP est de construire une mini application web avec **Flask** et un **LLM** (Ollama, GPT ou Gemini) qui génère automatiquement un rapport PDF synthétique.

Objectifs pédagogiques

— Comprendre comment intégrer un LLM via API ou local.

— Construire un backend Flask pour orchestrer l'analyse.

— Structurer automatiquement le texte généré par le LLM.

— Générer un PDF avec un rapport d'étude de marché exploitable.

Introduction

Ce guide explique comment transformer un projet Flask en FastAPI en conservant l'utilisation des **templates HTML**, de la **génération PDF** et de l'intégration d'un **LLM**.

1. Installation de FastAPI

```
# Cr er l'environnement virtuel
python -m venv venv
# Linux / Mac
source venv/bin/activate
# Windows
venv\Scripts\activate

# Installer FastAPI, Uvicorn et Jinja2 pour templates
pip install fastapi uvicorn jinja2 requests python-multipart reportlab
```

2. Structure du projet

```
my_fastapi_app/

 main.py          # fichier principal
 templates/       # HTML avec Jinja2
    index.html
    result.html
 static/          # CSS / JS / images
 venv/
```


3. Endpoints et templates HTML

```python
from fastapi import FastAPI, Form, Request
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates
import requests

app = FastAPI()
templates = Jinja2Templates(directory="templates")

def call_llm(prompt: str) -> str:
    response = requests.post(
        "http://localhost:11434/api/generate",
        json={"model": "llama3", "prompt": prompt}
    )
    return response.json()["response"]

@app.get("/", response_class=HTMLResponse)
def home(request: Request):
    return templates.TemplateResponse("index.html", {"request": request
        })

@app.post("/analyse_market", response_class=HTMLResponse)
def analyse_market(request: Request, produit: str = Form(...), secteur:
    str = Form(...)):
    prompt = f"Fais une  tude  de march  synth tique pour {produit}
        dans le secteur {secteur}, inclut concurrents, tendances et
        points forts/faibles."
    texte = call_llm(prompt)
    return templates.TemplateResponse("result.html", {"request": request
        , "contenu": texte})
```

4. Templates HTML

index.html

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Analyse de march </title>
</head>
<body>
    <h1>Analyse de march  automatique</h1>
    <form action="/analyse_market" method="post">
        Produit: <input type="text" name="produit"><br>
        Secteur: <input type="text" name="secteur"><br>
        <input type="submit" value="Analyser">
```

```
    </form>
</body>
</html>
```

result.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>R sultat de l'analyse</title>
</head>
<body>
    <h1>R sultat de l'analyse de march </h1>
    <pre>{{ contenu }}</pre>
    <a href="/">Retour</a>
</body>
</html>
```

5. Génération de PDF

```
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from fastapi.responses import FileResponse

def generate_pdf(content, filename="rapport.pdf"):
    c = canvas.Canvas(filename, pagesize=A4)
    c.setFont("Helvetica-Bold", 16)
    c.drawString(50, 800, " tude  de march  automatis e")
    c.setFont("Helvetica", 12)
    y = 780
    for line in content.split('\n'):
        c.drawString(50, y, line)
        y -= 15
    c.save()
    return filename

@app.get("/download_pdf")
def download_pdf():
    filename = generate_pdf("Exemple de contenu")
    return FileResponse(filename, media_type='application/pdf', filename
       ="rapport.pdf")
```

6. Lancer le serveur


```
uvicorn main:app --reload
# Acc s    l'application: http://127.0.0.1:8000
```

Challenge avancé : Étude de marché complète en autonomie

Dans cette partie, vous devez aller plus loin et construire **vous-même** des fonctionnalités supplémentaires pour rendre l'application plus proche d'une vraie solution business. Aucune solution complète n'est fournie : vous devez concevoir la logique, structurer vos prompts, et générer les livrables.

Objectifs du challenge

— Comparer plusieurs produits dans un même secteur et générer un tableau comparatif dans le PDF.

— Ajouter des graphiques simples (histogrammes, tendances, parts de marché simulées) à partir des données générées par le LLM.

— Permettre à l'utilisateur de sélectionner plusieurs produits à analyser en une seule requête.

— Ajouter la possibilité d'envoyer le rapport PDF par email à un destinataire choisi.

— Structurer le PDF avec des sections et sous-sections pour rendre le rapport professionnel.

Suggestions pour démarrer

— Réfléchissez à la façon de **structurer les prompts** pour que le LLM renvoie des informations exploitables (ex. JSON ou texte bien découpé).

— Explorez des librairies Python pour générer des tableaux et graphiques dans vos PDF (`reportlab`, `matplotlib`, etc.).

— Testez votre application étape par étape : d'abord le prompt et le LLM, ensuite la génération PDF, puis l'envoi par email.

— Documentez votre travail et soyez capable d'expliquer vos choix de conception.

Livrables attendus

— Code Flask complet avec endpoints et logique de génération PDF.

— Un PDF d'étude de marché généré automatiquement pour au moins deux produits différents.

— Optionnel : graphiques et comparaison détaillée dans le PDF.

— Une brève explication de votre approche et des prompts utilisés pour le LLM.

Note : Cette partie du TP est conçue pour que vous réfléchissiez de manière autonome et appliquiez les concepts appris précédemment. Vous pouvez vous inspirer des étapes précédentes, mais l'implémentation finale doit être votre propre travail.

