

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

```
1 # Créer l'environnement virtuel
2 python -m venv venv
3 # Linux / Mac
4 source venv/bin/activate
5 # Windows
6 venv\Scripts\activate
7
8 # Installer FastAPI, Uvicorn et Jinja2 pour templates
9 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

```
1 from fastapi import FastAPI, Form, Request
2 from fastapi.responses import HTMLResponse
3 from fastapi.templating import Jinja2Templates
4 import requests
5
6 app = FastAPI()
7 templates = Jinja2Templates(directory="templates")
8
9 def call_llm(prompt: str) -> str:
10     response = requests.post(
11         "http://localhost:11434/api/generate",
12         json={"model": "llama3", "prompt": prompt}
13     )
14     return response.json()["response"]
```

```

15
16 @app.get("/", response_class=HTMLResponse)
17 def home(request: Request):
18     return templates.TemplateResponse("index.html", {"request": request
19                                     })
19
20 @app.post("/analyse_market", response_class=HTMLResponse)
21 def analyse_market(request: Request, produit: str = Form(...), secteur:
22     str = Form(...)):
23     prompt = f"Fais une étude de marché synthétique pour {produit}
24         dans le secteur {secteur}, inclut concurrents, tendances et
25         points forts/faibles."
26     texte = call_llm(prompt)
27     return templates.TemplateResponse("result.html", {"request": request
28                                                         , "contenu": texte})

```

4. Templates HTML

index.html

```

1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4     <meta charset="UTF-8">
5     <title>Analyse de marché </title>
6 </head>
7 <body>
8     <h1>Analyse de marché automatique</h1>
9     <form action="/analyse_market" method="post">
10         Produit: <input type="text" name="produit"><br>
11         Secteur: <input type="text" name="secteur"><br>
12         <input type="submit" value="Analyser">
13     </form>
14 </body>
15 </html>

```

result.html

```

1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4     <meta charset="UTF-8">
5     <title>Résultat de l'analyse</title>
6 </head>
7 <body>
8     <h1>Résultat de l'analyse de marché </h1>
9     <pre>{{ contenu }}</pre>

```

```
10 <a href="/">Retour</a>
11 </body>
12 </html>
```

5. Génération de PDF

```
1 from reportlab.lib.pagesizes import A4
2 from reportlab.pdfgen import canvas
3 from fastapi.responses import FileResponse
4
5 def generate_pdf(content, filename="rapport.pdf"):
6     c = canvas.Canvas(filename, pagesize=A4)
7     c.setFont("Helvetica-Bold", 16)
8     c.drawString(50, 800, "tude de march automatis e")
9     c.setFont("Helvetica", 12)
10    y = 780
11    for line in content.split('\n'):
12        c.drawString(50, y, line)
13        y -= 15
14    c.save()
15    return filename
16
17 @app.get("/download_pdf")
18 def download_pdf():
19     filename = generate_pdf("Exemple de contenu")
20     return FileResponse(filename, media_type='application/pdf', filename="rapport.pdf")
```

6. Lancer le serveur

```
1 uvicorn main:app --reload
2 # 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 bibliothèques 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.