Guide d’installation de FastAPI et Chatbot avec RAG

Étape 1 : Installation de VS Code et Python

1. **Télécharger et installer VS Code** : Lien de téléchargement
2. **Installer Python** (si ce n'est pas encore fait) : Lien de téléchargement

    - Vérifiez l'installation avec :

        ```
        python --version
        ```

    - Si vous utilisez **Windows**, assurez-vous d'ajouter Python au **PATH** lors de l'installation.

Étape 2 : Création d’un environnement virtuel

Ouvrez **VS Code**, puis ouvrez un **terminal intégré** (**Ctrl** + ` sous Windows/Linux).
Exécutez les commandes suivantes :

- **Sous Windows (Terminal VS Code ou `cmd`)** :

    ```
    python -m venv venv
    venv\Scripts\activate
    ```

- **Sous macOS/Linux (Terminal VS Code ou `bash`)** :

    ```
    python3 -m venv venv
    source venv/bin/activate
    ```

Vous devriez voir `(venv)` devant la ligne de commande, indiquant que l'environnement est activé.

Étape 3: Organisation des fichiers du projet

Dans **VS Code**, créez la structure suivante :

```
/MyChatBot
├── /app
│    │    ├── main.py  # Fichier principal pour FastAPI
│    ├── api.py  # Contient l'API FastAPI
├── /app/documents/    # Dossiers pour vos fichiers PDF du RAG
├── .env/  # Fichier des variables d’environnement
├── REAMDE.md  # Fichier readMe
├── requirements.txt  # Liste des dépendances
```

Étape 4 : Installation des dépendances

Dans le fichier requirements.txt mettez :

```
aiofiles==23.2.1
aiohappyeyeballs==2.4.4
aiohttp==3.11.12
aiosignal==1.3.2
annotated-types==0.7.0
anyio==4.8.0
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
arrow==1.3.0
asgiref==3.8.1
asttokens==3.0.0
async-lru==2.0.4
attrs==24.3.0
audioop-lts==0.2.1
babel==2.16.0
backoff==2.2.1
bcrypt==4.2.1
beautifulsoup4==4.12.3
bleach==6.2.0
blinker==1.9.0
build==1.2.2.post1
cachetools==5.5.1
certifi==2024.12.14
cffi==1.17.1
charset-normalizer==3.4.1
chroma-hnswlib==0.7.6
chromadb==0.6.3
click==8.1.8
colorama==0.4.6
coloredlogs==15.0.1
comm==0.2.2
contourpy==1.3.1
cycler==0.12.1
dacktool==0.0.7
dataclasses-json==0.6.7
dbstream==0.1.28
debugpy==1.8.12
decorator==5.1.1
defusedxml==0.7.1
Deprecated==1.2.18
dirtyjson==1.0.8
distro==1.9.0
durationpy==0.9
environs==14.1.0
executing==2.1.0
```

```
faiss-cpu==1.10.0
fastapi==0.115.6
fastapi_cors==0.0.6
fastjsonschema==2.21.1
ffmpy==0.5.0
filelock==3.16.1
filetype==1.2.0
Flask==3.1.0
Flask-Cors==5.0.0
flatbuffers==25.1.24
fonttools==4.55.3
fqdn==1.5.1
frozenlist==1.5.0
fsspec==2024.12.0
google-api-core==2.24.1
google-auth==2.38.0
googleapis-common-protos==1.66.0
gradio==5.15.0
gradio_client==1.7.0
greenlet==3.1.1
groq==0.18.0
grpcio==1.70.0
h11==0.14.0
httpcore==1.0.7
httptools==0.6.4
httpx==0.28.1
httpx-sse==0.4.0
huggingface-hub==0.28.1
humanfriendly==10.0
idna==3.10
importlib_metadata==8.5.0
importlib_resources==6.5.2
ipykernel==6.29.5
ipython==8.31.0
ipywidgets==8.1.5
isoduration==20.11.0
itsdangerous==2.2.0
jedi==0.19.2
Jinja2==3.1.5
jiter==0.8.2
joblib==1.4.2
json5==0.10.0
jsonpatch==1.33
jsonpointer==3.0.0
jsonschema==4.23.0
jsonschema-specifications==2024.10.1
jupyter-events==0.11.0
jupyter-lsp==2.2.5
jupyter_client==8.6.3
```

```
jupyter_core==5.7.2
jupyter_server==2.15.0
jupyter_server_terminals==0.5.3
jupyterlab==4.3.4
jupyterlab_pygments==0.3.0
jupyterlab_server==2.27.3
jupyterlab_widgets==3.0.13
kiwisolver==1.4.8
kubernetes==32.0.0
langchain==0.3.18
langchain-chroma==0.2.1
langchain-community==0.3.16
langchain-core==0.3.34
langchain-groq==0.2.4
langchain-huggingface==0.1.2
langchain-text-splitters==0.3.6
langsmith==0.3.6
llama-cloud==0.1.12
llama-cloud-services==0.6.0
llama-index==0.12.16
llama-index-agent-openai==0.4.5
llama-index-cli==0.4.0
llama-index-core==0.12.16.post1
llama-index-embeddings-openai==0.3.1
llama-index-indices-managed-llama-cloud==0.6.4
llama-index-llms-openai==0.3.18
llama-index-multi-modal-llms-openai==0.4.3
llama-index-program-openai==0.3.1
llama-index-question-gen-openai==0.3.0
llama-index-readers-file==0.4.4
llama-index-readers-llama-parse==0.4.0
llama-parse==0.6.0
markdown-it-py==3.0.0
MarkupSafe==2.1.5
marshmallow==3.26.1
matplotlib==3.10.0
matplotlib-inline==0.1.7
mdurl==0.1.2
mistune==3.1.0
mmh3==5.1.0
monotonic==1.6
mpmath==1.3.0
multidict==6.1.0
mypy-extensions==1.0.0
nbclient==0.10.2
nbconvert==7.16.5
nbformat==5.10.4
nest-asyncio==1.6.0
networkx==3.4.2
```

```
nltk==3.9.1
notebook==7.3.2
notebook_shim==0.2.4
numpy==1.26.4
oauthlib==3.2.2
ollama==0.4.7
onnxruntime==1.20.1
openai==1.61.1
opentelemetry-api==1.30.0
opentelemetry-exporter-otlp-proto-common==1.30.0
opentelemetry-exporter-otlp-proto-grpc==1.30.0
opentelemetry-instrumentation==0.51b0
opentelemetry-instrumentation-asgi==0.51b0
opentelemetry-instrumentation-fastapi==0.51b0
opentelemetry-proto==1.30.0
opentelemetry-sdk==1.30.0
opentelemetry-semantic-conventions==0.51b0
opentelemetry-util-http==0.51b0
orjson==3.10.14
overrides==7.7.0
packaging==24.2
pandas==2.2.3
pandocfilters==1.5.1
parso==0.8.4
pillow==11.1.0
platformdirs==4.3.6
posthog==3.11.0
prometheus_client==0.21.1
prompt_toolkit==3.0.48
propcache==0.2.1
proto-plus==1.26.0
protobuf==5.29.3
psutil==6.1.1
pure_eval==0.2.3
pyasn1==0.6.1
pyasn1_modules==0.4.1
pycparser==2.22
pydantic==2.10.5
pydantic-settings==2.7.1
pydantic_core==2.27.2
pydub==0.25.1
Pygments==2.19.1
pyparsing==3.2.1
pypdf==5.2.0
PyPDF2==3.0.1
PyPika==0.48.9
pyproject_hooks==1.2.0
pyreadline3==3.5.4
python-dateutil==2.9.0.post0
```

```
python-dotenv==1.0.1
python-json-logger==3.2.1
python-multipart==0.0.20
pytz==2024.2
pywin32==308
pywinpty==2.0.14
PyYAML==6.0.2
pyzmq==26.2.0
referencing==0.36.1
regex==2024.11.6
requests==2.32.3
requests-oauthlib==2.0.0
requests-toolbelt==1.0.0
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rich==13.9.4
rpds-py==0.22.3
rsa==4.9
ruff==0.9.4
safehttpx==0.1.6
safetensors==0.5.2
scikit-learn==1.6.1
scipy==1.15.1
semantic-version==2.10.0
Send2Trash==1.8.3
sentence-transformers==3.4.1
setuptools==75.8.0
shellingham==1.5.4
six==1.17.0
slite==0.0.2
sniffio==1.3.1
soupsieve==2.6
SQLAlchemy==2.0.37
stack-data==0.6.3
starlette==0.41.3
striprtf==0.0.26
sympy==1.13.1
tenacity==9.0.0
terminado==0.18.1
threadpoolctl==3.5.0
tiktoken==0.8.0
tinycss2==1.4.0
tokenizers==0.21.0
tomlkit==0.13.2
torch==2.6.0
tornado==6.4.2
tqdm==4.67.1
traitlets==5.14.3
transformers==4.48.2
```

```
typer==0.15.1
types-python-dateutil==2.9.0.20241206
typing-inspect==0.9.0
typing_extensions==4.12.2
tzdata==2024.2
uri-template==1.3.0
urllib3==2.3.0
uvicorn==0.34.0
watchfiles==1.0.4
wcwidth==0.2.13
webcolors==24.11.1
webencodings==0.5.1
websocket-client==1.8.0
websockets==14.1
Werkzeug==3.1.3
widgetsnbextension==4.0.13
wrapt==1.17.2
yarl==1.18.3
zipp==3.21.0
zstandard==0.23.0
```

Dans le **terminal VS Code**, exécutez :

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

Étape 5 : Implémentation du serveur FastAPI

Dans **`app/main.py`**, ajoutez le code suivant :

```python
from fastapi import FastAPI, Request
from app.api import router

app = FastAPI()

app.include_router(router)

@app.get("/")
def read_root():
    return {"request": request}
```

Ajoutez l'API dans **`app/routes.py`** (avec votre).

```python
import re
import os
from fastapi import APIRouter, HTTPException
```

```python
from pydantic import BaseModel
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma
from langchain.chains import RetrievalQA
from langchain_groq import ChatGroq
from langchain.prompts import PromptTemplate
from langchain.memory import ConversationBufferMemory
from groq import Client
from fastapi.encoders import jsonable_encoder
import json
from langchain_community.llms import Ollama



router = APIRouter()

# Modèle de requête pour le chatbot
class QueryRequest(BaseModel):
    question: str

# Fonction de nettoyage du texte
def clean_text(text):
    text = re.sub(r'\s+', ' ', text)  # Suppression des espaces et sauts de
ligne excessifs
    return text.strip()  # Suppression des espaces en début/fin

# Liste des documents juridiques à indexer
documents = [
    {"path": "app/documents/Depression_symptoms_in_Canadian_psycholo.pdf",
"title": "Daniel L. Peluso, R. Nicholas Carleton, and Gordon J. G. Asmundson,
Depression Symptoms in Canadian Psychology Graduate Students: Do Research
Productivity, Funding, and the Academic Advisory Relationship Play a Role?,
Canadian Journal of Behavioural Science © 2011 Canadian Psychological
Association 2011, Vol. 43, No. 2, 119–127"},

    # Ajouter d'autres documents ici...
]

# Initialisation de la liste des documents traités
all_docs = []

# Chargement et traitement des documents
for doc in documents:
    pdf_path = doc["path"]
    title = doc["title"]

    if not os.path.exists(pdf_path):
```

```
        print(f"⚠ Attention : {pdf_path} non trouvé, il sera ignoré.")
        continue

    loader = PyPDFLoader(pdf_path)
    pages = loader.load()

    for page_num, page in enumerate(pages):
        cleaned_text = clean_text(page.page_content)
        all_docs.append({
            "text": cleaned_text,
            "metadata": {"source": pdf_path, "title": title, "page": page_num
+ 1}
        })

# Découpage du texte en chunks avec inclusion des métadonnées
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
docs = [
    (chunk, doc["metadata"])
    for doc in all_docs
    for chunk in text_splitter.split_text(doc["text"])
]

# Création d'objets Documents avec métadonnées
from langchain.schema import Document
final_docs = [
    Document(page_content=chunk, metadata=metadata) for chunk, metadata in
docs
]

# Chargement du modèle d'embeddings
persist_directory = "MyVectorDB1.0"
embedding_function = HuggingFaceEmbeddings(model_name="sentence-
transformers/multi-qa-MiniLM-L6-cos-v1")

# Création ou chargement de la base Chroma avec métadonnées
vectorstore = Chroma.from_documents(final_docs, embedding_function,
persist_directory=persist_directory)

# Configuration de la clé API Groq
api_key = os.getenv("GROQ_API_KEY")
if not api_key:
    raise ValueError("La clé API GROQ_API_KEY n'est pas définie.")

client = Client(api_key=api_key)

# Initialisation du modèle Groq
groq_llm = ChatGroq(model_name="llama-3.3-70b-versatile")
#ollama_llm = Ollama(model="mistral")
```

```
# Définition du prompt personnalisé
template = """
Tu es un **Psychiatre spécialisé dans le diagnostic de la dépression**.
Ton rôle est de faire un diagnostic préliminaire de la dépression chez les
patients et de les orienter vers des professionnels de la santé mentale si
nécessaire.
COmmencer par receuillir des informations générales (age, sexe) sur les
symptômes et les antécédents médicaux du patient.
Guider le patient à travers une série de questions pour évaluer son état
mental et émotionnel.
Si des informations manquent pour la précision de la réponse vous pouvez
demander à l'utilisateur de les fournir.
Donnerun diagnostic préliminaire de la dépression et recommander des
ressources pour obtenir de l'aide.

### **Contexte Psychiatrique disponible** :
{context}

### **Historique de la conversation** :
{chat_history}

### **Utilisateur** : {question}

### **Assistant Psychiatre** :
"""

prompt_template = PromptTemplate(
    input_variables=["history", "context", "question"],
    template=template,
)

# Initialisation de la mémoire de conversation
memory = ConversationBufferMemory(memory_key="chat_history",
return_messages=True, input_key="question")

# Création de la chaîne QA avec mémoire et récupération des sources
qa_chain = RetrievalQA.from_chain_type(
    llm=groq_llm,
    chain_type="stuff",
    retriever=vectorstore.as_retriever(search_kwargs={"k": 5}),  #
Récupération des 5 documents les plus pertinents
    return_source_documents=True,
    verbose=True,
    chain_type_kwargs={"prompt": prompt_template, "memory": memory, "verbose":
True}
)
```

```python
@router.post("/chatPsy")
def chat(request: QueryRequest):
    try:
        response = qa_chain({"query": request.question})  # Exécute la requête
        answer = response['result']

        # 🔍 Debugging: Afficher les sources
        print("🔍 Raw source documents:", response['source_documents'])

        # Extraction des sources avec métadonnées
        sources = []
        for doc in response['source_documents']:
            print("📝 Metadata:", doc.metadata)  # Debugging : voir les
métadonnées réelles
            sources.append({
                "source": doc.metadata.get("source", "Inconnu"),
                "title": doc.metadata.get("title", "Inconnu"),
                "page": doc.metadata.get("page", "Inconnue")
            })

        # 🔍 Debugging: Afficher la structure des sources avant retour
        print("☑ Formatted sources:", json.dumps(sources, indent=2,
ensure_ascii=False))

        return jsonable_encoder({"response": answer, "sources": sources})

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

Étape 6 : Configuration des variables d'environnement

Créez un fichier **`.env`** dans le dossier **`MyCHatBot/`** :

```
$env:GROQ_API_KEY=VOTRE_CLE_GROQ
```

Étape 7 : Lancer l'API FastAPI

Dans **VS Code (terminal)**, exécutez :

```
uvicorn app.main:app --reload
```

L'API sera accessible à **`http://127.0.0.1:8000`** .

Pour tester l'API avec Swagger UI :
Ouvrez **`http://127.0.0.1:8000/docs`** dans votre navigateur.

Étape 8 : Tester avec `curl` ou Postman

Vous pouvez tester avec **Postman** ou **cURL** :

```
curl -X 'POST' \
  'http://127.0.0.1:8000/chat' \
  -H 'Content-Type: application/json' \
  -d '{"question": "Quels sont les articles du Code du Commerce sur les
contrats ?"}'
```

Tester l'API avec Swagger UI

1. **Lancez votre API** dans le **terminal VS Code** avec la commande :

   ```
   uvicorn app.main:app --reload
   ```

2. **Ouvrez votre navigateur** et accédez à :

   ```
   http://127.0.0.1:8000/docs
   ```

3. Vous verrez **l'interface Swagger** avec toutes les routes de votre API.
4. Cliquez sur **`POST /chat`**, puis sur **"Try it out"**.
5. Dans le champ **Request Body**, entrez une question comme ceci :

   ```
   {
     "question": "Quels sont les articles du Code du Commerce sur les
   contrats ?"
   }
   ```

6. **Cliquez sur "Execute"** et observez la réponse de l'API !