Guide Complet pour Débutants: Création d'un Chatbot Juridique avec Groq API, ChromaDB et MinLM

GHaythem Ghazouani

May 15, 2025

Contents

1 Introduction

Ce guide vous montrera, étape par étape, comment créer un chatbot juridique qui utilise l'API Groq avec LangChain, ChromaDB et le modèle MinLM pour répondre à des questions juridiques basées sur un document PDF. Nous utiliserons FastAPI pour créer une API REST et Swagger pour la tester. Ce guide est conçu pour les débutants et inclut toutes les commandes à exécuter dans VSCode.

2 Installation de l'environnement de développement

2.1 Installation de VSCode

Si vous n'avez pas encore VSCode, téléchargez-le depuis `https://code.visualstudio.com/` et installez-le.

2.2 Extensions VSCode recommandées

Ouvrez VSCode et installez les extensions suivantes pour faciliter le développement:

1. Python (Microsoft)
2. Pylance (Microsoft)
3. Python Indent (Kevin Rose)
4. autoDocstring (Nils Werner)

Pour installer une extension:

1. Cliquez sur l'icône des extensions dans la barre latérale (ou appuyez sur Ctrl+Shift+X)
2. Recherchez l'extension
3. Cliquez sur "Installer"

2.3 Installation de Python

Si vous n'avez pas Python installé:

1. Téléchargez Python 3.8 ou supérieur depuis `https://www.python.org/downloads/`
2. Lors de l'installation, cochez la case "Add Python to PATH"
3. Vérifiez l'installation en ouvrant un terminal dans VSCode (Terminal → New Terminal) et tapez:

```
python --version
```

3 Création du projet

3.1 Création du dossier du projet

Ouvrez VSCode et suivez ces étapes:

1. Cliquez sur "File" → "Open Folder"
2. Créez un nouveau dossier nommé "legal_chatbot" et ouvrez-le
3. Ouvrez un terminal dans VSCode (Terminal → New Terminal)

3.2 Création de l'environnement virtuel

Dans le terminal VSCode, exécutez les commandes suivantes:

```
# Création de l'environnement virtuel
python -m venv venv

# Activation de l'environnement virtuel
# Sur Windows
venv\Scripts\activate
# Sur macOS/Linux
source venv/bin/activate
```

Vous devriez voir "(venv)" apparaître au début de la ligne de commande, indiquant que l'environnement virtuel est activé.

3.3 Installation des dépendances

Toujours dans le terminal VSCode avec l'environnement virtuel activé, installez les dépendances nécessaires:

```
# Installation des packages requis
pip install fastapi uvicorn python-multipart python-dotenv
pip install langchain langchain-groq
pip install chromadb sentence-transformers
pip install pypdf
```

3.4 Création de la structure du projet

Exécutez les commandes suivantes pour créer la structure de base du projet:

```
# Sur Windows
mkdir app
mkdir app\routes
mkdir models
mkdir utils
mkdir data
type nul > app\__init__.py
type nul > app\main.py
type nul > app\routes\__init__.py
type nul > app\routes\chatbot.py
type nul > models\__init__.py
```

```
type nul > models\chatbot.py
type nul > utils\__init__.py
type nul > utils\pdf_loader.py
type nul > .env

# Sur macOS/Linux
mkdir -p app/routes
mkdir -p models
mkdir -p utils
mkdir -p data
touch app/__init__.py
touch app/main.py
touch app/routes/__init__.py
touch app/routes/chatbot.py
touch models/__init__.py
touch models/chatbot.py
touch utils/__init__.py
touch utils/pdf_loader.py
touch .env
```

4 Configuration de l'environnement

4.1 Configuration des variables d'environnement

Dans VSCode, ouvrez le fichier '.env' et ajoutez votre clé API Groq:

```
GROQ_API_KEY=votre_cle_api_groq
```

Pour obtenir une clé API Groq:

1. Créez un compte sur `https://console.groq.com/`
2. Accédez à la section API Keys
3. Créez une nouvelle clé API
4. Copiez la clé et collez-la dans le fichier .env

5 Implémentation du chargeur de PDF

Ouvrez le fichier 'utils/pdf$_{l}$*oader.py*'*dansVSCodeetajoutezlecodesuivant* :

```
# utils/pdf_loader.py
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import HuggingFaceEmbeddings
import os
from dotenv import load_dotenv

# Charger les variables d'environnement
load_dotenv()

def load_and_process_pdf(pdf_path):
```

```
    """
    Charge un document PDF, le divise en chunks et crée une base de
     données vectorielle avec ChromaDB.

    Args:
        pdf_path (str): Chemin vers le fichier PDF à traiter

    Returns:
        Chroma: Base de données vectorielle contenant les chunks du
     document
    """
    print(f"Chargement du PDF: {pdf_path}")

    #  tape  1: Charger le PDF avec PyPDFLoader
    loader = PyPDFLoader(pdf_path)
    documents = loader.load()
    print(f"PDF chargé avec succès: {len(documents)} pages")

    #  tape  2: Diviser le texte en chunks plus petits pour un
    meilleur traitement
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,  # Taille de chaque chunk en caractères
        chunk_overlap=200,  # Chevauchement entre les chunks pour
    maintenir le contexte
        length_function=len
    )
    chunks = text_splitter.split_documents(documents)
    print(f"Document divisé en {len(chunks)} chunks")

    #  tape  3: Initialiser le modèle d'embedding MinLM
    # Nous utilisons le modèle all-MiniLM-L6-v2 qui offre un bon é
    quilibre entre performance et rapidité
    embeddings = HuggingFaceEmbeddings(
        model_name="sentence-transformers/all-MiniLM-L6-v2",
        model_kwargs={'device': 'cpu'}  # Utiliser CPU pour la
    compatibilité
    )
    print("Modèle d'embedding MinLM initialisé")

    #  tape  4: Créer une base de données vectorielle avec ChromaDB
    # ChromaDB stockera les embeddings et permettra des recherches
    sémantiques efficaces
    db_directory = "./data/chroma_db"
    vector_store = Chroma.from_documents(
        documents=chunks,
        embedding=embeddings,
        persist_directory=db_directory  # Stockage persistant des
    embeddings
    )

    # Persister la base de données pour une utilisation future
    vector_store.persist()
    print(f"Base de données vectorielle ChromaDB créée et persistée
     dans {db_directory}")

    return vector_store
```

6 Implémentation du modèle de chatbot

Ouvrez le fichier ‘models/chatbot.py‘ dans VSCode et ajoutez le code suivant:

```python
# models/chatbot.py
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory
from langchain_groq import ChatGroq
import os
from dotenv import load_dotenv

# Charger les variables d'environnement
load_dotenv()

class LegalChatbot:
    def __init__(self, vector_store):
        """
        Initialise le chatbot juridique avec une base de données vectorielle.

        Args:
            vector_store (Chroma): Base de données vectorielle ChromaDB contenant les documents
        """
        print("Initialisation du chatbot juridique...")

        # Stocker la base de données vectorielle
        self.vector_store = vector_store

        # Créer une mémoire pour stocker l'historique de conversation
        self.memory = ConversationBufferMemory(
            memory_key="chat_history",
            return_messages=True
        )

        # Vérifier que la clé API Groq est disponible
        groq_api_key = os.environ.get("GROQ_API_KEY")
        if not groq_api_key:
            raise ValueError("La clé API Groq n'est pas définie. Vérifiez votre fichier .env")

        # Définir le prompt système pour guider le modèle
        SYSTEM_PROMPT = """
        Tu es un assistant juridique expert. Ta t che est de répondre aux questions juridiques en te basant uniquement sur les documents fournis.

        Voici quelques règles à suivre:
        1. Réponds uniquement aux questions juridiques basées sur les documents fournis.
        2. Si tu ne connais pas la réponse ou si l'information n'est pas dans les documents, indique-le clairement.
        3. Cite les sections spécifiques des documents pour appuyer tes réponses.
        4. Utilise un langage clair et précis, mais reste technique lorsque c'est nécessaire.
```

```
        5. N'invente pas d'informations qui ne sont pas présentes dans les documents.

        Format de réponse:
        - Commence par une réponse directe à la question
        - Développe avec des détails pertinents
        - Cite les sources spécifiques (numéros de page, sections)
        - Si nécessaire, mentionne les limitations de ta réponse
        """

        # Initialiser le modèle LLM avec Groq
        self.llm = ChatGroq(
            groq_api_key=groq_api_key,
            model_name="llama3-70b-8192",  # Modèle LLM puissant pour les réponses juridiques
            temperature=0.2,  # Température basse pour des réponses plus précises
            system=SYSTEM_PROMPT  # Prompt système pour guider le modèle
        )
        print("Modèle LLM Groq initialisé")

        # Créer la cha ne de conversation avec récupération (RAG)
        self.chain = ConversationalRetrievalChain.from_llm(
            llm=self.llm,
            retriever=self.vector_store.as_retriever(
                search_kwargs={"k": 3}  # Récupérer les 3 chunks les plus pertinents
            ),
            memory=self.memory,
            return_source_documents=True  # Retourner les documents sources pour les citations
        )
        print("Cha ne de conversation RAG initialisée")

    async def get_response(self, query):
        """
        Obtient une réponse du chatbot pour une requête donnée.

        Args:
            query (str): La question posée par l'utilisateur

        Returns:
            dict: Dictionnaire contenant la réponse et les sources
        """
        print(f"Traitement de la requête: {query}")

        # Invoquer la cha ne de conversation de manière asynchrone
        result = await self.chain.ainvoke({"question": query})

        # Extraire les sources pour la citation
        sources = []
        for doc in result.get("source_documents", []):
            if hasattr(doc, "metadata") and "source" in doc.metadata:
                sources.append(doc.metadata["source"])
```

```
            elif hasattr(doc, "metadata") and "page" in doc.metadata:
                sources.append(f"Page {doc.metadata['page']}")

        # Retourner la réponse et les sources
        response = {
            "answer": result["answer"],
            "sources": list(set(sources))  # liminer les doublons
        }

        print("Réponse générée avec succès")
        return response
```

7 Création des routes API

Ouvrez le fichier ‘app/routes/chatbot.py‘ dans VSCode et ajoutez le code suivant:

```
# app/routes/chatbot.py
from fastapi import APIRouter, UploadFile, File, HTTPException
from fastapi.responses import JSONResponse
import os
import tempfile
from pydantic import BaseModel
from typing import List, Optional
from models.chatbot import LegalChatbot
from utils.pdf_loader import load_and_process_pdf

# Créer un router FastAPI
router = APIRouter(tags=["chatbot"])

# Modèles de données pour la validation des entrées/sorties
class Query(BaseModel):
    text: str

    class Config:
        schema_extra = {
            "example": {
                "text": "Quelles sont les obligations légales d'un employeur?"
            }
        }

class Response(BaseModel):
    answer: str
    sources: Optional[List[str]] = None

# Variable globale pour stocker l'instance du chatbot
chatbot_instance = None

@router.post("/upload-pdf", response_class=JSONResponse, summary="Télécharger un document PDF")
async def upload_pdf(file: UploadFile = File(...)):
    """
    Télécharge un document PDF et initialise le chatbot juridique.
```

```
36
37      - **file**: Fichier PDF à télécharger (format .pdf uniquement)
38
39      Retourne un message de confirmation si le chargement est réussi
        .
40      """
41      global chatbot_instance
42
43      # Vérifier que le fichier est un PDF
44      if not file.filename.endswith(".pdf"):
45          raise HTTPException(
46              status_code=400,
47              detail="Le fichier doit être au format PDF (.pdf)"
48          )
49
50      # Créer un fichier temporaire pour stocker le PDF
51      with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf")
        as temp_file:
52          # Lire le contenu du fichier téléchargé et l'écrire dans le
         fichier temporaire
53          content = await file.read()
54          temp_file.write(content)
55          temp_file_path = temp_file.name
56
57      try:
58          # Charger et traiter le PDF avec notre utilitaire
59          vector_store = load_and_process_pdf(temp_file_path)
60
61          # Initialiser le chatbot avec la base de données
        vectorielle
62          chatbot_instance = LegalChatbot(vector_store)
63
64          return {
65              "message": "Document PDF chargé avec succès et chatbot
        initialisé",
66              "filename": file.filename
67          }
68      except Exception as e:
69          # En cas d'erreur, renvoyer un message d'erreur détaillé
70          raise HTTPException(
71              status_code=500,
72              detail=f"Erreur lors du traitement du PDF: {str(e)}"
73          )
74      finally:
75          # Supprimer le fichier temporaire pour libérer de l'espace
76          if os.path.exists(temp_file_path):
77              os.unlink(temp_file_path)
78
79  @router.post("/ask", response_model=Response, summary="Poser une
        question au chatbot")
80  async def ask_question(query: Query):
81      """
82      Pose une question au chatbot juridique et obtient une réponse.
83
84      - **query**: Objet contenant le texte de la question
85
86      Retourne la réponse du chatbot et les sources utilisées.
```

```
    """
    global chatbot_instance

    # Vérifier que le chatbot a été initialisé
    if chatbot_instance is None:
        raise HTTPException(
            status_code=400,
            detail="Veuillez d'abord télécharger un document PDF via /upload-pdf"
        )

    try:
        # Obtenir une réponse du chatbot
        response = await chatbot_instance.get_response(query.text)
        return response
    except Exception as e:
        # En cas d'erreur, renvoyer un message d'erreur détaillé
        raise HTTPException(
            status_code=500,
            detail=f"Erreur lors de la génération de la réponse: {str(e)}"
        )
```

8 Création de l'application principale

Ouvrez le fichier 'app/main.py' dans VSCode et ajoutez le code suivant:

```
# app/main.py
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.routes.chatbot import router as chatbot_router

# Créer l'application FastAPI
app = FastAPI(
    title="Chatbot Juridique API",
    description="API pour un chatbot juridique utilisant Groq, ChromaDB et MinLM",
    version="1.0.0",
    docs_url="/docs",  # URL pour la documentation Swagger
    redoc_url="/redoc"  # URL pour la documentation ReDoc
)

# Configuration CORS pour permettre les requêtes cross-origin
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Autoriser toutes les origines
    allow_credentials=True,
    allow_methods=["*"],  # Autoriser toutes les méthodes
    allow_headers=["*"],  # Autoriser tous les headers
)

# Ajouter les routes du chatbot
app.include_router(chatbot_router, prefix="/api")

# Route racine
@app.get("/", tags=["root"])
```

```python
async def root():
    """
    Route racine qui renvoie un message de bienvenue.
    """
    return {
        "message": "Bienvenue sur l'API du Chatbot Juridique",
        "documentation": "/docs",
        "endpoints": {
            "upload_pdf": "/api/upload-pdf",
            "ask_question": "/api/ask"
        }
    }
```

9 Création du fichier principal

Créez un fichier 'main.py' à la racine du projet et ajoutez le code suivant:

```python
# main.py
import uvicorn

if __name__ == "__main__":
    # Démarrer le serveur uvicorn
    uvicorn.run(
        "app.main:app",  # Chemin vers l'application FastAPI
        host="0.0.0.0",  #  couter  sur toutes les interfaces
        port=8000,       # Port 8000
        reload=True      # Recharger automatiquement lors des modifications
    )
```

10 Exécution de l'application

10.1 Lancement du serveur

Dans le terminal VSCode, assurez-vous que l'environnement virtuel est activé, puis exécutez:

```
# Exécuter l'application
python main.py
```

Vous devriez voir une sortie similaire à:

```
INFO:     Started server process [12345]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

10.2 Accès à l'interface Swagger

Ouvrez votre navigateur et accédez à:

```
http://localhost:8000/docs
```

Vous verrez l'interface Swagger qui vous permettra de tester votre API.

11 Test du chatbot

11.1 Téléchargement d'un document PDF

Dans l'interface Swagger:

1. Cliquez sur l'endpoint '/api/upload-pdf'
2. Cliquez sur "Try it out"
3. Cliquez sur "Choose File" et sélectionnez un document PDF juridique
4. Cliquez sur "Execute"

11.2 Poser une question

Après avoir téléchargé un PDF:

1. Cliquez sur l'endpoint '/api/ask'
2. Cliquez sur "Try it out"
3. Dans le champ "Request body", entrez votre question:

```
{
  "text": "Quelles sont les principales obligations légales mentionnées dans le document?"
}
```

4. Cliquez sur "Execute"

12 Dépannage

12.1 Problèmes courants et solutions

1. **Erreur "ModuleNotFoundError"**

   Solution: Vérifiez que vous avez installé toutes les dépendances et que l'environnement virtuel est activé.

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

2. **Erreur "API key not found"**

   Solution: Vérifiez que votre fichier .env contient la clé API Groq correcte.

3. **Erreur lors du chargement du PDF**

   Solution: Vérifiez que le PDF n'est pas corrompu et qu'il est lisible.

4. **Erreur "Address already in use"**

   Solution: Changez le port dans le fichier main.py ou arrêtez le processus qui utilise le port 8000.

```
    # Sur Windows
    netstat -ano | findstr :8000
    taskkill /PID <PID> /F

    # Sur macOS/Linux
    lsof -i :8000
    kill -9 <PID>

```

13 Améliorations possibles

Voici quelques idées pour améliorer votre chatbot:

1. **Interface utilisateur web**

   Créez une interface web simple avec HTML, CSS et JavaScript pour interagir avec votre API.

2. **Support de plusieurs documents**

   Modifiez le code pour permettre le chargement de plusieurs documents PDF.

3. **Historique des conversations**

   Ajoutez une base de données pour stocker l'historique des conversations.

4. **Authentification**

   Ajoutez une authentification pour sécuriser votre API.

5. **Déploiement**

   Déployez votre application sur un service comme Heroku, Vercel ou AWS.

14 Conclusion

Félicitations! Vous avez créé un chatbot juridique qui utilise l'API Groq avec LangChain, ChromaDB et le modèle MinLM pour répondre à des questions basées sur un document PDF. Ce chatbot peut être utilisé pour fournir des informations juridiques basées sur des documents spécifiques.

Ce projet vous a permis d'apprendre:

- Comment utiliser FastAPI pour créer une API REST
- Comment traiter des documents PDF avec PyPDFLoader
- Comment utiliser ChromaDB pour stocker et rechercher des embeddings

- Comment utiliser le modèle MinLM pour créer des embeddings
- Comment utiliser l'API Groq pour générer des réponses
- Comment implémenter la technique RAG (Retrieval Augmented Generation)