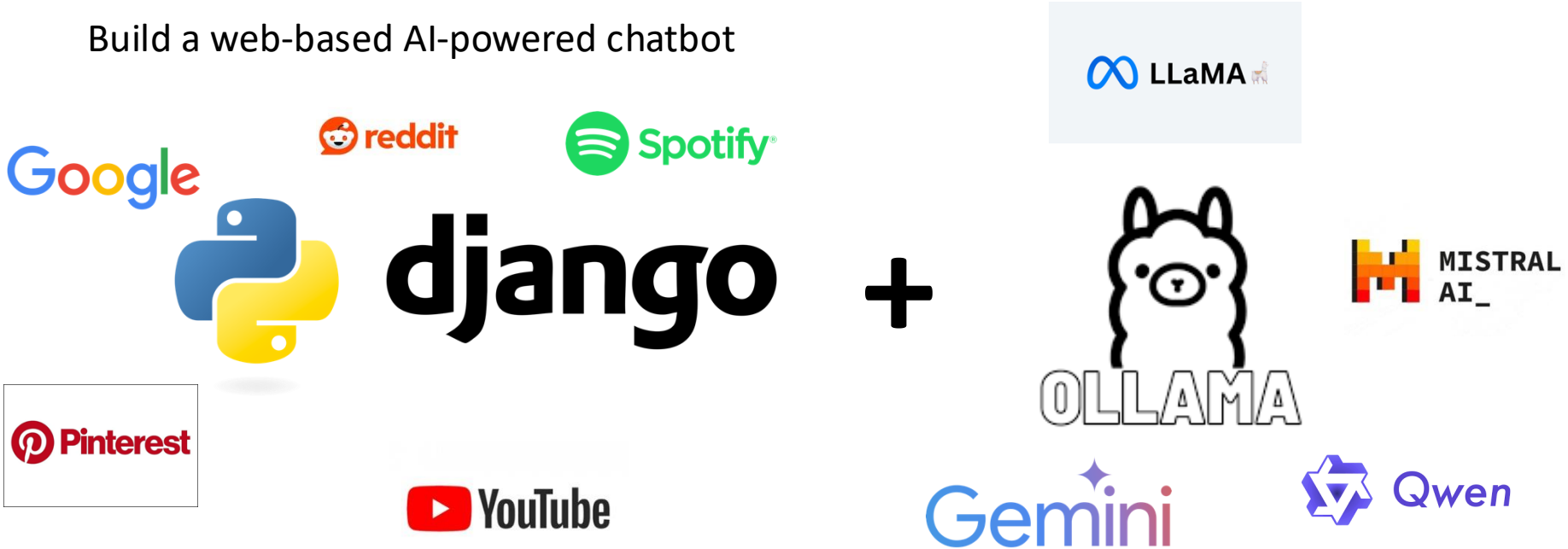
Building an AI Chatbot from Scratch

Sam Yang

September 26th, 2024

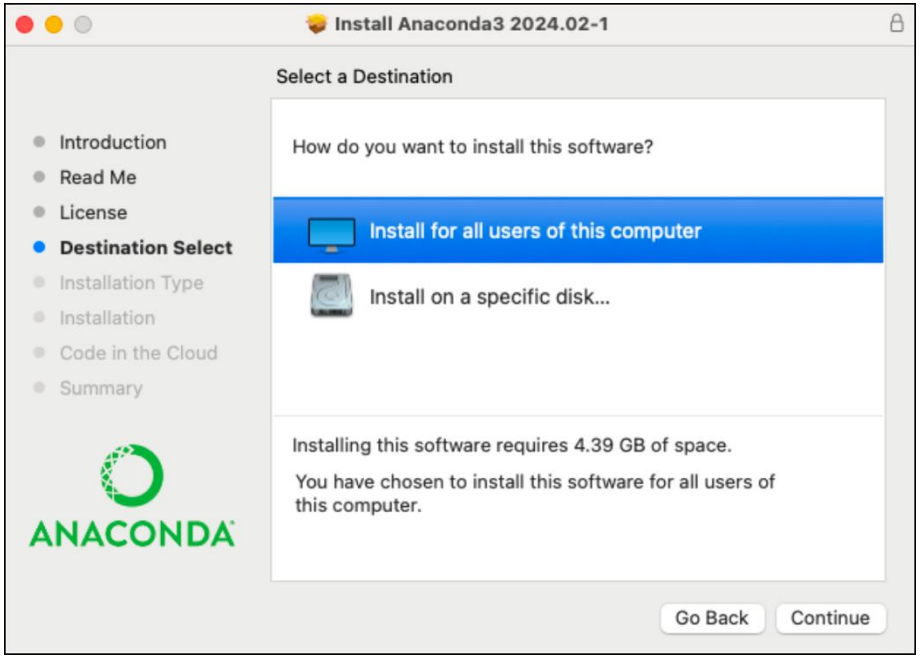
Overview

Build a web-based AI-powered chatbot



Install Anaconda

Windows: https://docs.anaconda.com/anaconda/install/windows/
Mac: https://docs.anaconda.com/anaconda/install/mac-os/
Linux: https://docs.anaconda.com/anaconda/install/linux/



Anaconda is an open-source distribution of the Python and R programming languages for data science that simplifies package management and deployment.

Create Folders

Django is a free, open-source web framework that helps developers build websites quickly and efficiently.

Create a Folder **LLM**:
mkdir LLM

Change directory to LLM:
cd LLM

Create a folder **Django_project**:
mkdir Django_project

Change directory to Django_project :
cd Django_project

Create Virtual Environment from YML File

In terminal (Anaconda Prompt on Windows), type the command below and press ENTER key:

conda env create -f environment.yml

When the environment is created, you will see:

```
done
#
# To activate this environment, use
#
#     $ conda activate LLM
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Activate the Virtual Environment:

conda activate LLM

If you see “(LLM)” at the beginning,
it means the LLM environment is activated:

```
done
#
# To activate this environment, use
#
#     $ conda activate LLM
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) samtari@Quanpengs-MacBook-Pro Chatbot % conda activate LLM
(LLM) samtari@Quanpengs-MacBook-Pro Chatbot %
```

Create Environment from requirements.txt File

In terminal (Anaconda Prompt on Windows), type the command below and press ENTER key:

conda create –-name LLM

Activate the Virtual Environment:

conda activate LLM

Install pip:

conda install pip

Install required packages:

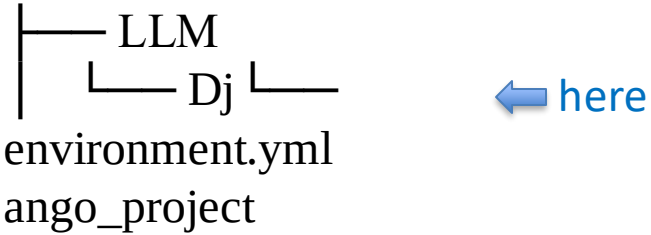
pip install –r requirements.txt

If there is a “OpenSSL” related error when creating the environment:
This is due to .dll error
go to location where you've installed Anaconda **Anaconda3>Library>bin**. search and copy following dll files
libcrypto-1_1-x64.dll
libssl-1_1-x64.dll
and paste to: **Anaconda3>DLLs**
then restart your **terminal** (or computer)
issue will get resolved.

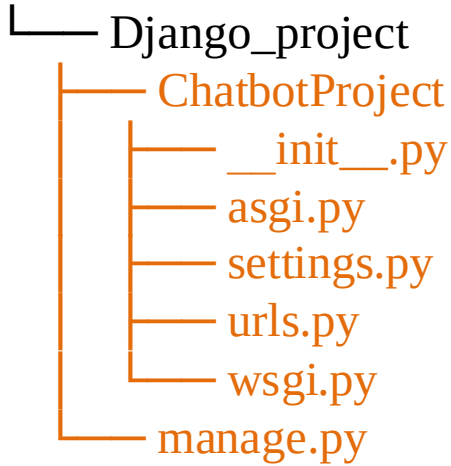
Create a Django Project

In folder Django_project, create a project named ***ChatbotProject***:

django-admin startproject ChatbotProject .



There will be some new files generated under the Django_project folder:



Verify if the Project Works:

python manage.py runserver

You will see:

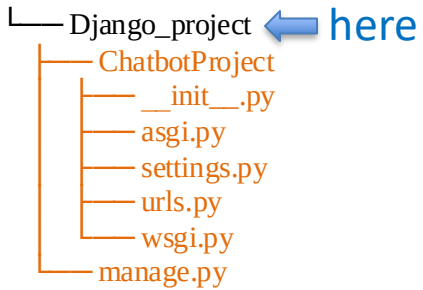
```
September 25, 2024 - 21:19:57
Django version 5.1.1, using settings 'Chatbot.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

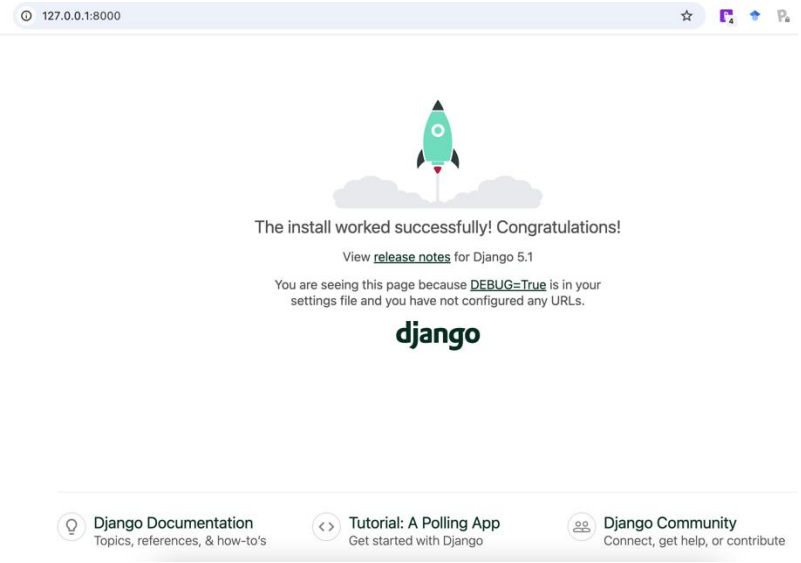
Access http://127.0.0.1:8000/ from a browser:

If you see this, congratulations! The project is built successfully!

If you want to exit:
CONTROL + Z (Windows)
COMMAND + Z (Mac)

But leaving it **on** is more convenient for the later development. Just open another terminal to continue. Don't forget to activate environment.

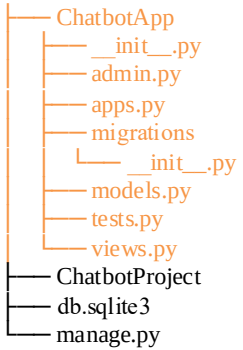


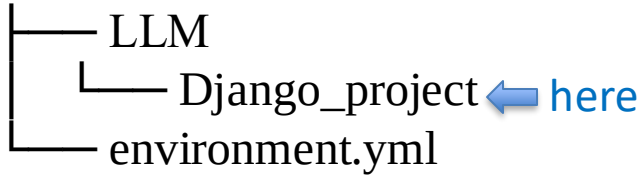


Create an App: ChatbotApp

python manage.py startapp ChatbotApp

There will be a new folder ***ChatbotApp*** and some files created:





Then, add ChatbotApp to settings

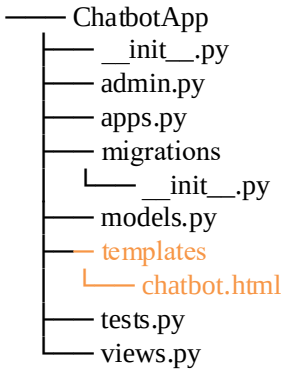
Edit ***settings.py*** file in ***ChatbotProject*** folder:
Add 'ChatbotApp', to INSTALLED_APPS list:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ChatbotApp',
]
```

Create a Template for Webpage

Create a folder ***templetes*** under ***ChatbotApp*** for templates:

Then create a file ***chatbot.html*** under ***templetes*** as a template:



Edit ***chatbot.html***

```
<!DOCTYPE html>
<html>
<head>
  <title>LLM Response</title>
</head>
<body style="height: 100%; margin: 0; display: flex; justify-content: center; align-items: center;">
  <div style="text-align: center;">
    <h1>My Chatbot</h1>
    <p>{{ response }}</p>
  </div>
</body>
</html>
```

Print “Hello World!”

Create a function:

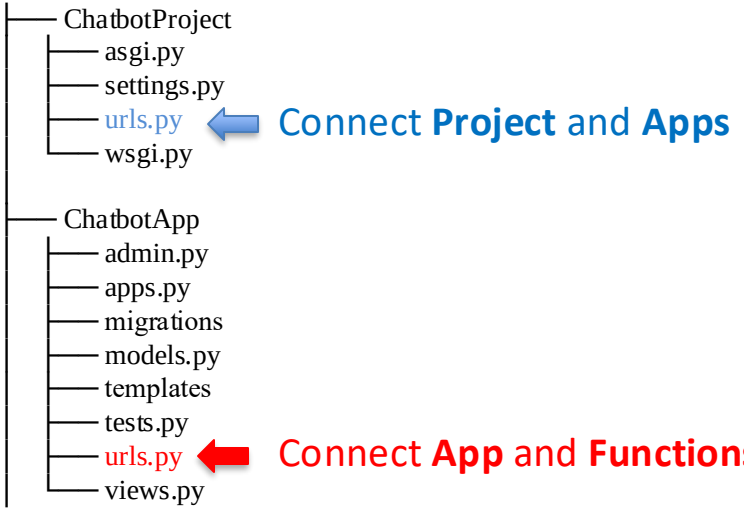
Edit the **views.py** file under **ChatbotApp** folder:

```
from django.shortcuts import render

def helloFunction(request):
    response = {"response": "Hello World!"}
    return render(request, 'chatbot.html', response)
```

Make Connections using urls.py

Edit the ***urls.py*** under the ***ChatbotProject*** folder:

```
from django.urls import path, include

urlpatterns = [
    path('', include('ChatbotApp.urls')),
]
```

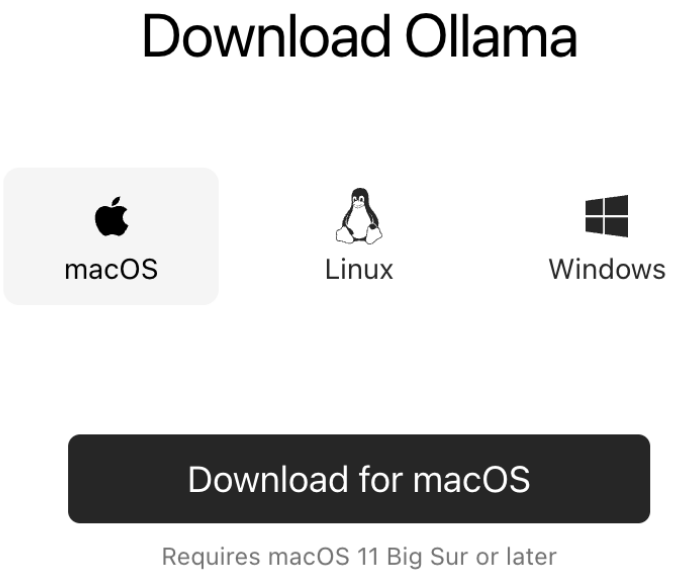


Create a ***urls.py*** under the ***ChatbotApp*** folder:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.helloFunction, name='helloFunction'),
]
```

Install Ollama

https://ollama.com/download



might need to open a new terminal

Install a Large Language Model

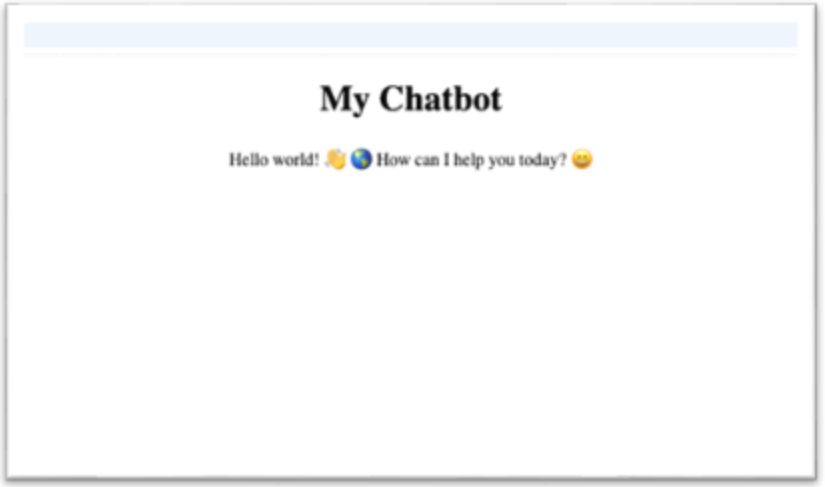
ollama pull gemma2:2b
ollama run gemma2:2b

```
(LLM) samtari@Quanpengs-MacBook-Pro Chatbot % ollama run gemma2:2b
>>> Hello
Hello! 👋

How can I help you today? 😊


>>> Send a message (/? for help)
```

First LLM Output

Edit the ***views.py*** file under ***ChatbotApp*** folder:

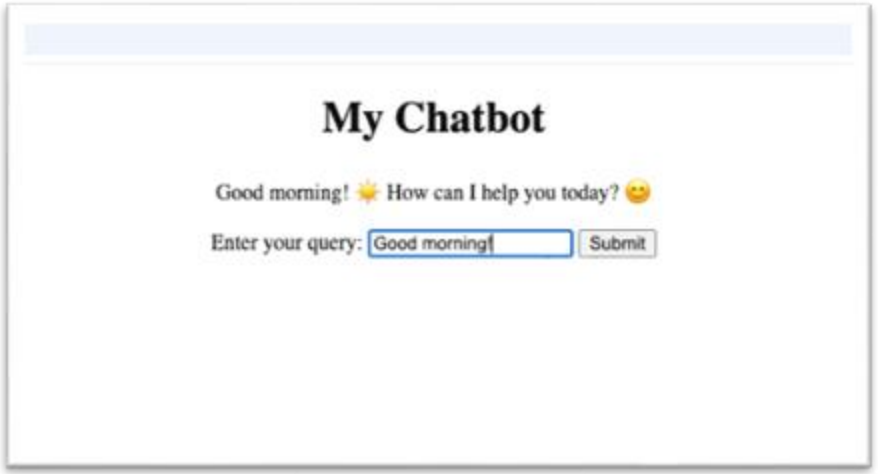
```
from django.shortcuts import render
import ollama

LLM_Model = 'gemma2:2b'

def LLM(prompt):
    response = ollama.chat(model=LLM_Model, messages=[{'role': 'user', 'content': prompt}])
    return response['message']['content']

def helloFunction(request):
    response = LLM("Hello world!")
    response = {"response": response}
    return render(request, 'chatbot.html', response)
```



Interact with Chatbot

Edit the ***views.py*** file under ***ChatbotApp*** folder:

```
from django.shortcuts import render
import ollama

LLM_Model = 'gemma2:2b'

def LLM(prompt):
    response = ollama.chat(model=LLM_Model, messages=[{'role': 'user', 'content': prompt}])
    return response['message']['content']


def helloFunction(request):
    userInput = request.GET.get('userInput')
    response = LLM(userInput)
    response = {"response": response}
    return render(request, 'chatbot.html', response)
```



Edit the ***chatbot.html*** file under ***templates*** folder:

```
<!DOCTYPE html>
<html>
<head>
    <title>LLM Response</title>
</head>
<body style="height: 100%; margin: 0; display: flex; justify-content: center; align-items: center;">
    <div style="text-align: center;">
        <h1>My Chatbot</h1>
        <p>{{ response }}</p>
        <form method="GET" action="{% url 'helloFunction' %}">
            <label for="userInput">Enter your query:</label>
            <input type="text" name="userInput" placeholder="Chat with me" required>
            <button type="submit">Submit</button>
        </form>
    </div>
</body>
</html>
```

Chat with Your Files

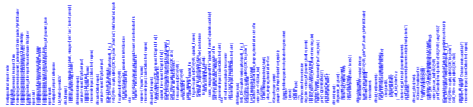
Use your documents to customize the chatbot

Create a new template ***customChatbot.html*** under ***templates*** folder

templates/customChatbot.html

```
<!DOCTYPE html>
<html>
<head>
  <title>LLM Response</title>
</head>
<body style="height: 100%; margin: 0; display: flex; justify-content: center; align-items: center;">
  <div style="text-align: center;">
    <h1>Custom Chatbot</h1>

      <h2>Train Chatbot</h2>
      <form method="POST" action="/upload" enctype="multipart/form-data">
        {% csrf_token %}
        <label for="file">Choose files to upload:</label>
        <input type="file" id="file" name="files[]" multiple required>
        <button type="submit">Upload</button>
      </form>
      <br />
      <form method="POST" action="/train">
        {% csrf_token %}
        <button type="submit">Train Chatbot</button>{{ alert }}
      </form>
    <p>{{ response }}</p>
    <form method="GET" action="{% url 'customChatbot' %}">
      <label for="userInput">Enter your query:</label>
      <input type="text" name="userInput" placeholder="Chat with me" required>
      <button type="submit">Submit</button>
    </form>
  </div>
</body>
</html>
```

Edit the ***views.py*** file under ***ChatbotApp*** folder:



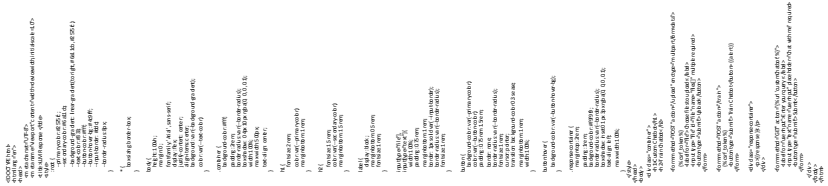
Edit the ***urls.py*** file under ***ChatbotApp*** folder:

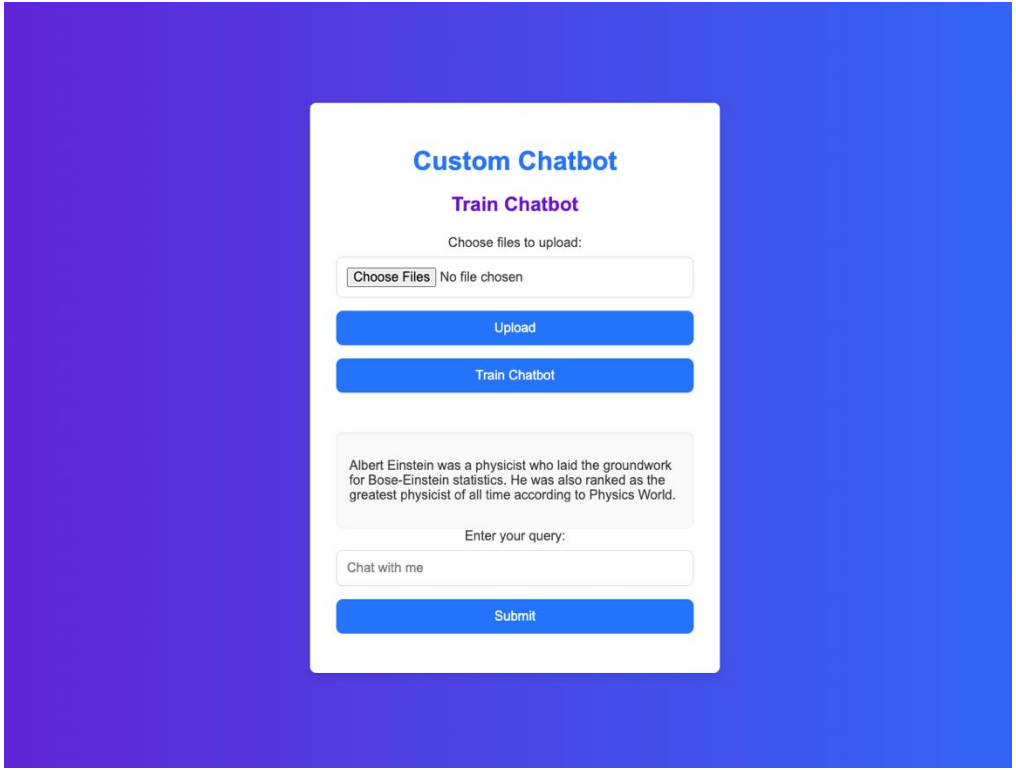
```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.helloFunction, name='helloFunction'),
    path('customChatbot', views.customChatbot, name='customChatbot'),
    path('upload', views.upload_files, name='upload_files'),
    path('train', views.train, name='train'),
]
```

Make it look nicer

Update template ***customChatbot.html*** under ***templates*** folder

templates/customChatbot.html





Bonus: Weather App

Provide suggestion on what to wear based on weather prediction

views.py

```
from django.shortcuts import render
import os
from langchain_community.llms import CTransformers
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import CTransformersEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter
import fitz
import pytesseract
from django.core.files.storage import default_storage
from langchain.prompts import PromptTemplate
from langchain.chains import ConversationalRetrievalChain
from langchain.chains.combine_documents import create_stuff_documents_chain
import requests
from django.http import HttpResponse

LLM_Model = "gem ma2-2b"

def llm_ll(prompt):
    response = client.chat(model='LLM_Model', messages=[{'role': 'user', 'content': prompt}])
    return response['message']['content']

def the lief_section(request):
    user_input = request.GET.get('user-input')
    response = LLM(user_input)
    response = {"response": response}
    return render(request, 'chatbot.html', response)

def focus_chatbot(request):
    user_input = request.GET.get('user-input')
    response = RAG(user_input)
    response = {"response": response}
    return render(request, 'custom_chatbot.html', response)

def upload_files(request):
    if request.method == 'POST' and request.FILES.getlist('files'):
        # Get the list of uploaded files
        uploaded_files = request.FILES.getlist('files')
        # Define the directory where files will be saved
        BASE_DIR = os.path.dirname(os.path.abspath(__file__))
        DATA_PATH = os.path.join(BASE_DIR, 'data')
        # Ensure the 'data' directory exists
        if not os.path.exists(DATA_PATH):
            os.makedirs(DATA_PATH)
        # Process each file
        success_messages = []
        for uploaded_file in uploaded_files:
            file_path = os.path.join(DATA_PATH, uploaded_file.name)
            # Save the file to the 'data' directory
            with default_storage.open(file_path, 'wb+') as destination:
                for chunk in uploaded_file.chunks():
                    destination.write(chunk)
            success_messages.append(f"File '{uploaded_file.name}' uploaded successfully")
        # Combine all success messages into one alert
        alert = {'message': success_messages}
        return render(request, 'custom_chatbot.html', alert)
    else:
        # If no files were uploaded, return an error
        alert = {'error': 'No files uploaded'}
        return render(request, 'custom_chatbot.html', alert)

def RAG(user_input):
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    FAISS_PATH = os.path.join(BASE_DIR, "faiss_index")
    # Load embeddings and vector store
    embeddings = CTransformersEmbeddings(model='LLM_Model')
    # Load the FAISS index with allow_dangerous_deserialization set to True
    vector_store = FAISS.load_local(
        FAISS_PATH,
        Ollama(model='LLM_Model'),
        allow_dangerous_deserialization=True
    )
    # Set up the question prompt
    prompt = PromptTemplate(template="""

        Answer the following question only based on the given context:
        <context>
        {context}
        </context>
        Question: {input}
        """)
    # Retrieve context from the vector store
    document_chain = create_stuff_documents_chain(llm, prompt)
    retriever = vector_store.as_retriever()
    retrieval_chain = create_retrieval_chain(retriever, document_chain)
    # Perform the query
    response = retrieval_chain.invoke({"input": user_input})
    print("---SOURCE PAGE CONTENT---")
    print(response["context"])
    return response["answer"]

def find_in_pdf():
    def generate_data_store():
        documents = load_documents()
        chunks = split_text(documents)
        save_to_faiss(chunks)
        return HttpResponse({'data to C': 'success'})
    def load_documents():
        # Load documents from a local directory
        loader = DirectoryLoader(DATA_PATH, glob="*.pdf", loader_cls=UnstructuredPDFLoader)
        documents = loader.load()
        return documents
    def split_text(documents):
        # Split documents into manageable chunks
        text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=500,
            chunk_overlap=100,
            length_function=len,
            add_start_index=True,
        )
```

templates/weather.html

```
<!DOCTYPE html>
<html>
<head>
    <title>All Weather Assistant</title>
</head>
<body style="height: 100%; margin: 0; display: flex; justify-content: center; align-items: center;">
    <div style="text-align: center;">

        <script>
            document.addEventListener('DOMContentLoaded', function () {
                // Check if the ip-hidden input is empty, then fetch the IP
                const ipInput = document.getElementById('ip-hidden');
                if (!ipInput.value) {
                    fetch('https://api.ipify.org?format=json')
                        .then(response => response.json())
                        .then(data => {
                            ipInput.value = data.ip;
                        })
                        .catch(error => console.error('Error fetching IP:', error));
                }
            });
        </script>

        <h1>AI Weather Assistant</h1>

        <form method="GET" action="{% url 'weather' %}">
            <label for="ip">IP Address:</label>
            <input type="text" id="ip-hidden" name="ip_address" placeholder="IP address" value="{{ ip_address | default_if_none:'' }}" required> {{ location }}
            <br>
            <label for="tone">Enter the tone:</label>
            <input type="text" name="tone" placeholder="set a tone for response" value="{{ tone | default:'normal' }}" required>

            <button type="submit">Submit</button>
        </form>

        <div style="width: 40%; margin: 0 auto; text-align: left;">
            <p>{{ response }}</p>
        </div>

    </div>
</body>
</html>
```

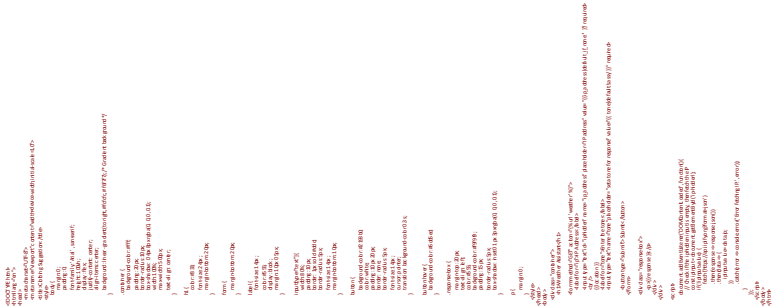
urls.py

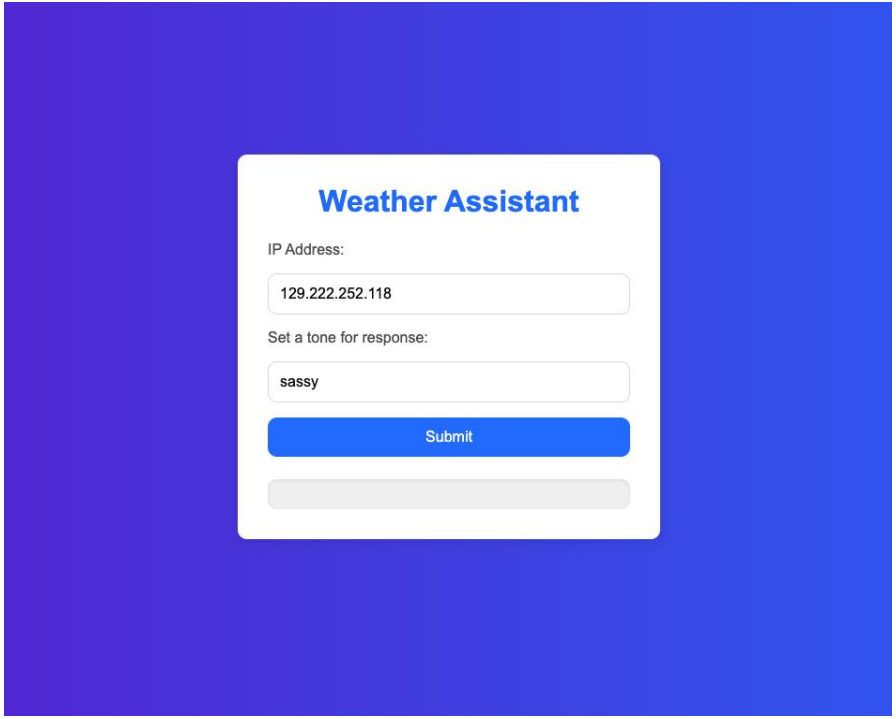
```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.helloFunction, name='helloFunction'),
    path('customChatbot', views.customChatbot, name='customChatbot'),
    path('upload', views.upload_files, name='upload_files'),
    path('train', views.train, name='train'),
    path('weather/', views.weather, name='weather'),
]
```

Make it look nicer

Update template ***weather.html*** under ***templates*** folder

templates/*weather*.html





Summary

- Used Django to build a website
- Used Ollama to load an LLM
- Customized a Chatbot
- Built a Weather App with APIs and LLM

Further Resources

- W3School: https://www.w3schools.com/
- Django: https://www.djangoproject.com/
- React: https://react.dev/
- MySQL: https://www.mysql.com/
- Neo4j: https://neo4j.com/

Supplemental Information

Port Issue

To run the *python manage.py runserver code to view on browser, there will be an error:*

Error: That port is already in use.

To solve this problem there are two ways:

1. Terminate the process previous running in the port (e.g., `8000`)
    1. *lsof -i :* `8000` (to find the **PID** of the process)
    2. *kill -9* PID

    (windows:

    1. `netstat -ano | findstr :8000`
    2. `taskkill /PID 8000 /f`)
2. Run the server on a new port:

    python manage.py runserver 8081

Access http://127.0.0.1:8000/ from a browser, now you have successfully pass information from Django to the browser