Integrating your Chatbot into a Web Interface

Skills Network

Introduction

In this lab, you will learn to set up a back-end server and integrate your chatbot into a web application.

Learning objectives

After completing this lab, you will be able to:

- Set up your back-end server
- Integrate your chatbot into your Flask server
- Communicate with the back-end using a web page

Prerequisites

This section assumes you know how to build the simple terminal chatbot explained in the first lab.

There are two things you must build to create your ChatGPT-like website:

1. A back-end server that hosts your chatbot
2. A front-end webpage that communicates with your back-end server

Without further ado, let's get started!

Step 1: Hosting your chatbot on a backend server

What is a backend server?

A backend server is like the brain behind a website or application. In this case, the backend server will receive prompts from your website, feed them into your chatbot, and return the output of the chatbot back to the website, which will be read by the user.

Hosting a simple backend server using Flask

Note: Consider using a requirements.txt file

`flask` is a Python framework for building web applications with Python. It provides a set of tools and functionalities to handle incoming requests, process data, and generate responses, making it easy to power your website or application.

Prerequisites

For all terminal interactions in this lab (such as running python files or installing packages), you will use the built-in terminal that comes with Cloud IDE. You may launch the terminal by either:

- Pressing `Ctrl + ` ` or
- By selecting Terminal --> New Terminal from the toolbar at the top of the IDE window on the right.

In your terminal, let's install the following requisites:

```
1. 1
2. 2

1. python3.11 -m pip install flask
2. python3.11 -m pip install flask_cors
```

Copied! Executed!

Setting up the server

Next, you will create a script that stores your flask server code.

To create a new Python file, Click on `File Explorer`, then right-click in the explorer area and select `New File`. Name this new file `app.py`.

Let's take a look at how to implement a simple flask server:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10

1. from flask import Flask
2. 
3. app = Flask(__name__)
4. 
5. @app.route('/')
6. def home():
7.     return 'Hello, World!'
8. 
9. if __name__ == '__main__':
10.     app.run()
```

Copied!

Paste the above code in the `app.py` file you just created and save it.

In this code:

- You import the `Flask` class from the `Flask` module.
- You create an instance of the `Flask` class and assign it to the variable `app`.
- You define a route for the homepage by decorating the `home()` function with the `@app.route()` decorator. The function returns the string 'Hello, World!'. This means that when the user visits the URL where the website is hosted, the backend server will receive the request and return 'Hello, World!' to the user.
- The `if __name__ == '__main__':` condition ensures that the server is only run if the script is executed directly, not when imported as a module.
- Finally, you call `app.run()` to start the server.

```
1. 1

1. python3.11 app.py
```

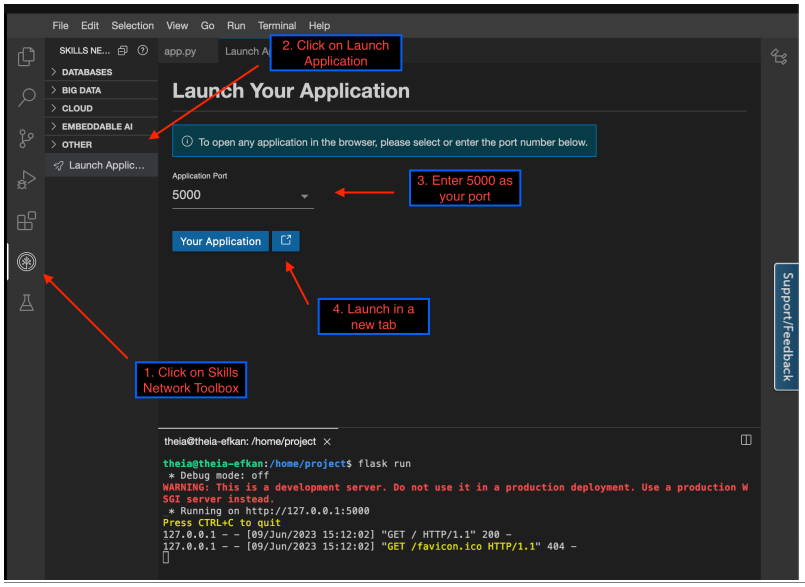
Copied! Executed!

Save this code in a Python file, for example, app.py, and run it by typing `python app.py` in the terminal. By default, Flask hosts the server at http://127.0.0.1:5000/ (which is equivalent to http://127.0.0.1:5000/).

With this command, the Flask server will start running. If you run this server on your local machine, then you can access it by visiting http://127.0.0.1:5000/ or http://localhost:5000/ in your web browser.

However, you are currently running this lab in the Skills Network Cloud. Thus, you can access your server as follows:

1. Navigate to the Skills Network Toolbox from the toolbar on the left side of the IDE
2. Click "Launch Application" in the adjacent vertical sidebar
3. Enter 5000 as your Application Port
4. Launch the application in a new browser tab

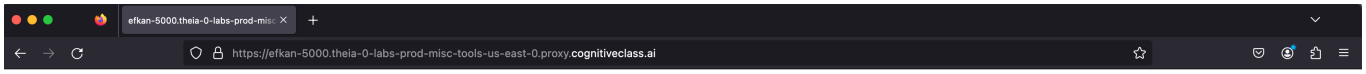


By performing the above steps, you have visited the **relative localhost URL** of the cloud server.

IMPORTANT: Throughout the rest of this lab, you will refer to this URL as <HOST>.

On visiting the localhost, you should see the "Hello, World!" message displayed.

Here's what it should look like:



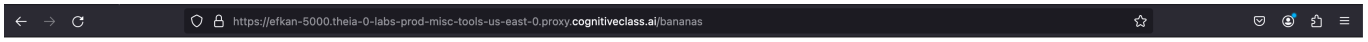
Hello, world!

Let's add the following routes to try it out:

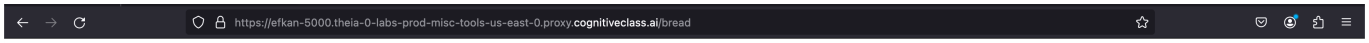
```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7

1. @app.route('/bananas')
2. def bananas():
3.     return 'This page has bananas!'
4.
5. @app.route('/bread')
6. def bread():
7.     return 'This page has bread!'
```

Copied!

Now, let's stop our app using `Ctrl + C` in the terminal and re-run with `flask run`. Then, let's visit both of these routes at http://<HOST>/bananas and http://<HOST>/bread. Here's what you should see:



This page has bananas!



This page has bread!

Okay - now that you've demonstrated how routes work, you can remove these two routes (bananas and bread) from your `app.py` as you won't be using them.

Before proceeding, you'll also add two more lines of code to your program to mitigate CORS errors - a type of error related to making requests to domains other than the one that hosts this webpage.

You'll be modifying your code as follows:

```
from flask import Flask
from flask_cors import CORS            # newly added

app = Flask(__name__)
CORS(app)                              # newly added

@app.route('/')
def home():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run()
```

Copied!

Integrating your chatbot into your Flask server

Now that you have your Flask server set up, let's integrate your chatbot into your Flask server.

As stated at the beginning, this lab assumes you've completed the first lab of this guided project on how to create your own simple chatbot.

First, you'll install the requisites

```
python3.11 -m pip install transformers torch
```

Copied!

Next, let's copy the code to initialize your chatbot from lab 1 and place it at the top of your script. You also must import the necessary libraries for your chatbot.

```
from transformers import AutoModelForSeq2SeqLM
from transformers import AutoTokenizer
```

Copied!

```
model_name = "facebook/blenderbot-400M-distill"
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
conversation_history = []
```

Copied!

Next, you'll need to import a couple more modules to read the data.

```
from flask import request
import json
```

Copied!

Before implementing the actual function though, you need to determine the structure you expect to receive in the incoming HTTP request.

Let's define your expected structure as follows:

```
{
    'prompt': 'message'
}
```

Copied!

Now implement your chatbot function. Again, you'll copy code over from your chatbot implementation from the first lab.

```
@app.route('/chatbot', methods=['POST'])
def handle_prompt():
    # Read prompt from HTTP request body
    data = request.get_data(as_text=True)
    data = json.loads(data)
    input_text = data['prompt']

    # Create conversation history string
    history = "\n".join(conversation_history)

    # Tokenize the input text and history
    inputs = tokenizer.encode_plus(history, input_text, return_tensors="pt")

    # Generate the response from the model
    outputs = model.generate(**inputs, max_length= 60)  # max_length will acuse model to crash at some point as history grows

    # Decode the response
    response = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()

    # Add interaction to conversation history
    conversation_history.append(input_text)
    conversation_history.append(response)

    return response
```

Copied!

The only new thing you've done uptil now is read the prompt from the HTTP request body. You've copied everything else from your previous chatbot implementation!

Perfect, now before testing your application, here's what the final version of your code looks like:

```
from flask import Flask, request, render_template
from flask_cors import CORS
import time
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

app = Flask(__name__)
CORS(app)

model_name = "facebook/blenderbot-400M-distill"
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
conversation_history = []

@app.route('/chatbot', methods=['POST'])
def handle_prompt():
    data = request.get_data(as_text=True)
    data = json.loads(data)
    input_text = data['prompt']

    # Create conversation history string
    history = "\n".join(conversation_history)
```

```
    # Tokenize the input text and history
    inputs = tokenizer.encode_plus(history, input_text, return_tensors="pt")

    # Generate the response from the model
    outputs = model.generate(**inputs, max_length= 60)  # max_length will cause the model to crash at some point as history grows

    # Decode the response
    response = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()

    # Add interaction to conversation history
    conversation_history.append(input_text)
    conversation_history.append(response)

    return response


if __name__ == '__main__':
    app.run()
```

Copied

Now let's test your implementation by using `curl` to make a POST request to <HOST>/chatbot with the following request body: {"prompt":"Hello, how are you today?"}.

Open a new terminal: Select terminal tab -> open new terminal

```
1
curl -X POST -H "Content-Type: application/json" -d '{"prompt": "Hello, how are you today?"}' 127.0.0.1:5000/chatbot
```

Copied Executed

Here's the output of the above code:

```
1
I am doing very well today as well. I am glad to hear you are doing well.
```

Copied

If you got a similar response, then congratulations! You have successfully created a Flask backend server with an integrated chatbot!

After you finish, press `ctrl + c` to stop the server.

Communicating with your backend using a webpage

In this section, you'll download a template chatbot webpage and configure it to make requests to your backend server.

First, let's clone a repository that has a template website and install your required libraries.
If your flask app is running, terminate it with `Ctrl + C` and run the following lines in the terminal:

```
1
2
git clone https://github.com/ibm-developer-skills-network/LLM_application_chatbot
python3.11 -m pip install -r LLM_application_chatbot/requirements.txt
```

Copied Executed

If the operations are complete with no errors, then you have successfully obtained a copy of the template repository.

The file structure of this repo should be as follows:

- ibm-chatbot-template/
  - static/
    - script.js
    - < other assets >
  - templates/
    - index.html

Let's move your flask app `app.py` to the `LLM_application_chatbot/` folder so that you can host `index.html` on your server.

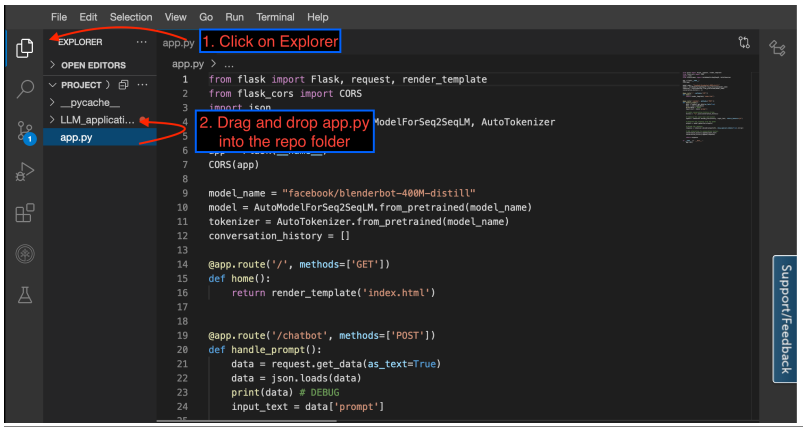
Both `app.py` and the `LLM_application_chatbot/` folder should be in `/home/project`. You can move `app.py` into `ibm_chatbot_template/` by running the following line in the terminal:

```
1
mv app.py LLM_application_chatbot/
```

Copied Executed

Alternatively, you may do the same by dragging and dropping in the IDE as follows:

1. Navigate to the Explorer from the sidebar on the left
2. Drag and drop `app.py` into `LLM_application_chatbot/` (Be careful not to drop it in a subfolder)



After adding your flask app, the file structure should be as follows:

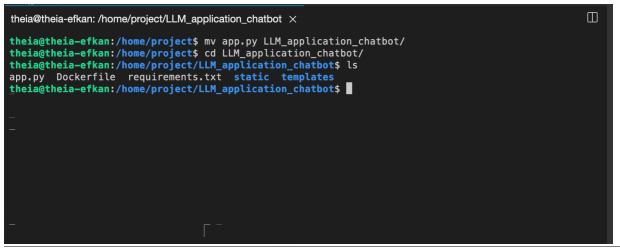
- LLM_application_chatbot/
  - app.py
  - static/
    - script.js
    - < other assets >
  - templates/
    - index.html

Let's test this by running:

```
1
2
cd LLM_application_chatbot/
ls
```

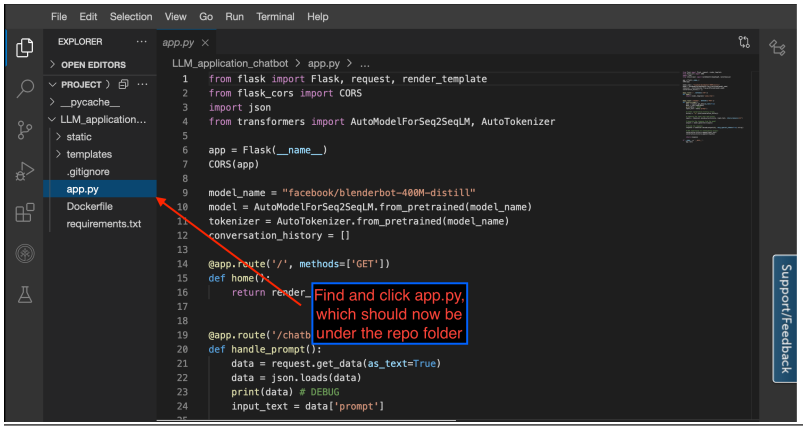
Copied Executed

You should see `app.py` in the output:



Since you've changed the location of `app.py`, you must re-open it in your editor. This is because the current tab for `app.py` tries to read from and write to app.py at a path that no longer exists.

Simply close the editor tab for `app.py`, and re-open it by clicking `app.py` in the explorer.



Now, let's modify your `app.py` so that you host index.html at <HOST>/. You can achieve this by adding the following route to your code:

```
1
2
3
@app.route('/', methods=['GET'])
def home():
    return render_template('index.html')
```

Copied

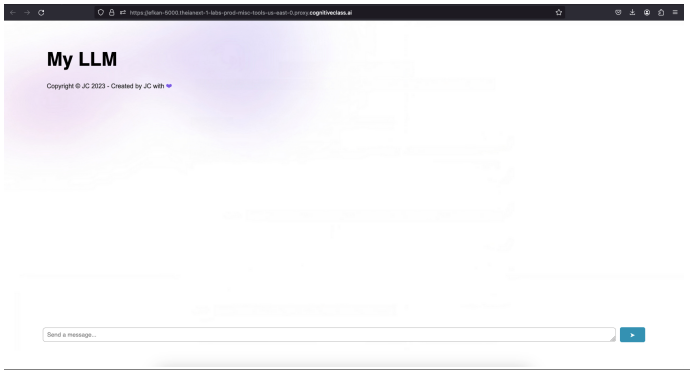
After adding the code, you can run your flask app with the following:

```
1
flask run
```

Copied Executed

Once your flask server is running, you may see the render of `index.html` by visiting <HOST>/.

Here's what you should see:



This template already has JavaScript code that emulates a chatbot interface. When you type a message and hit send, the website template does the following:

1. Enter your input message into a text bubble
2. Send your input to an endpoint (by default, this is set to www.example.com in the template)
3. Wait for the response from the endpoint and put the response in a text bubble

You will not implement such an interface as it is beside the purpose of this lab.

Instead, you will make sure that in step 2, you send the user input to the route you created for your chatbot earlier: http://127.0.0.1:5000/chatbot.

To send the input, you open `/static/script.js` and find where the endpoint is set.

```
30  const sendMessage = async (message) => {
31    // addMessage(message, 'user','user.jpeg');
32    addMessage(message, 'user','../static/user.jpeg');
33    // Loading animation
34    const loadingElement = document.createElement('div');
35    const loadingtextElement = document.createElement('p');
36    loadingElement.className = `loading-animation`;
37    loadingtextElement.className = `loading-text`;
38    loadingtextElement.innerText = 'Loading....Please wait';
39    messagesContainer.appendChild(loadingElement);
40    messagesContainer.appendChild(loadingtextElement);
41
42    async function makePostRequest(msg) {
43      const url = 'www.example.com';
44      const requestBody = {
45        prompt: msg
46      };
47
48      try {
49        const response = await fetch(url, {
50          method: 'POST',
51          headers: {
52            'Content-Type': 'application/json'
53          },
54          body: JSON.stringify(requestBody)
55        });
56
57        const data = await response.text();
58        // Handle the response data here
59        console.log(data);
60        return data;
61      } catch (error) {
62        // Handle any errors that occurred during the request
63        console.error('Error:', error);
64        return error
65      }
66    }
```

Let's change this endpoint to your chatbot route. In this example, www.example.com will replace

```
1. 1
2. 2
```

```
1. 'https://sinaez-5000.theianext-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/chatbot'
2.
```

Copied!

The URL may be different for you. Basically, copy the url in your app launnch and add `/chatbot` at the end

```
40    loadingtextElement.innerText = 'Loading....Please wait';
41    messagesContainer.appendChild(loadingElement);
42    messagesContainer.appendChild(loadingtextElement);
43
44    async function makePostRequest(msg) {
45      const url = 'https://efkan-5000.theia-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/';  // Make a POST request to this url
46      const requestBody = {
47        prompt: msg
48      };
49
```

And that should be it! Before testing your code, let's glance at the final version of your flask app:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
```

```
1. from flask import Flask, request, render_template
2. from flask_cors import CORS
3. import json
4. from transformers import AutoModelForSeq2Seq2LM, AutoTokenizer
5.
6. app = Flask(__name__)
7. CORS(app)
8.
9. model_name = "facebook/blenderbot-400M-distill"
10. model = AutoModelForSeq2Seq2LM.from_pretrained(model_name)
11. tokenizer = AutoTokenizer.from_pretrained(model_name)
12. conversation_history = []
13.
14. @app.route('/', methods=['GET'])
15. def home():
16.     return render_template('index.html')
17.
18.
19. @app.route('/chatbot', methods=['POST'])
20. def handle_prompt():
21.     data = request.get_data(as_text=True)
22.     data = eval(data)
23.     print(data) # DEBUG
24.     input_text = data['prompt']
25.
26.     # Create conversation history string
27.     history = "\n".join(conversation_history)
28.
29.     # Tokenize the input text and history
30.     inputs = tokenizer.encode_plus(history, input_text, return_tensors="pt")
31.
32.     # Generate the response from the model
33.     outputs = model.generate(**inputs)
34.
35.     # Decode the response
36.     response = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()
37.
38.     # Add interaction to conversation history
39.     conversation_history.append(input_text)
40.     conversation_history.append(response)
41.
42.     return response
43.
44. if __name__ == '__main__':
45.     app.run()
```

Copied!

Okay! Now let's test your app by stopping it first with `Ctrl + C` (if it's running) and restarting it with `flask run`.

In your terminal, you should see something like this:

```
ibm-chatbot-template — flask run — 80×24
(base) efkan@Efkans-MacBook-Pro ibm-chatbot-template % flask run
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

And now let's navigate to your homepage at <HOST>/ and try out your chatbot!

My LLM

Copyright © JC 2023 - Created by JC with ❤

Hello, my name is Jim. What is your name?

Hi jim, nice to meet you. My name is john. What do you do for fun?

I enjoy programming and kitesurfing. How about you?

I like hiking and skydiving. How interesting!

Send a message...

Congratulations! If your output is similar, then you have just created your own chatbot website!

Author

Sina Nazeri (Ph.D.) linkedin