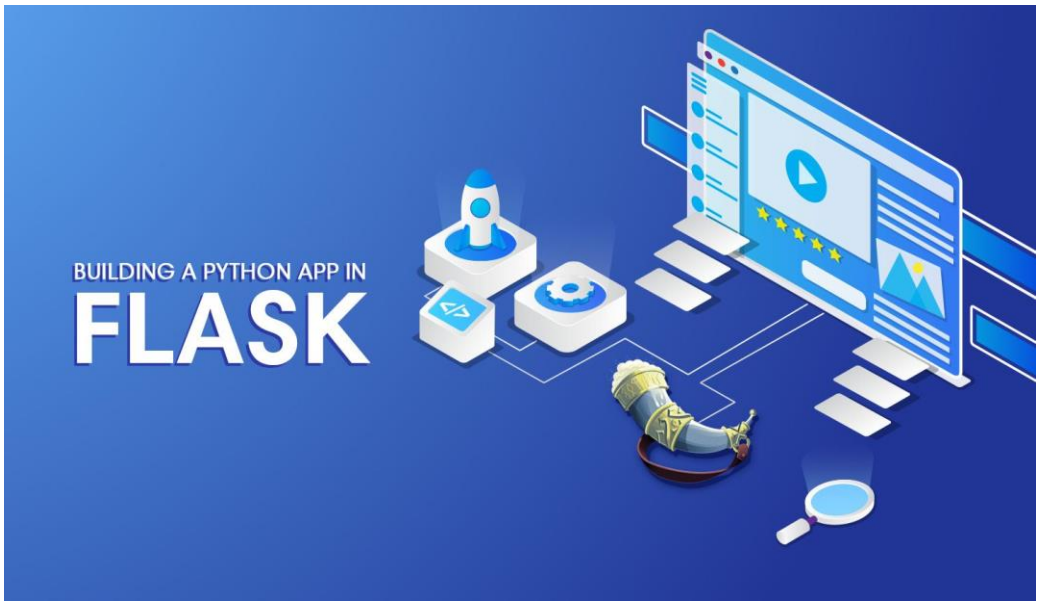
Integrating Chatbot into Web app using Flask

AUTHOR: JOHNKNOX P BOVAS

REG.NO: 961621104037

INTRODUCTION

The Flask is a Python micro-framework used to create small web applications and websites using Python. Flask works on a popular templating engine called Jinja2, a web templating system combined with data sources to the dynamic web pages.



1. *Set up Flask*: Make sure you have Flask installed. If not, install it using pip:

```
pip install Flask
```

2. *Create a Flask App*: Create a new Flask app by defining routes, templates, and views.

3. *HTML Templates*: Create HTML templates for the chat interface. You'll need an input field for user messages and a chat window to display bot responses.

4. *CSS Styling*: Apply CSS to style the chat interface.

5. *Chatbot Logic*: Implement the chatbot logic in Python. You can use a function to process user input and generate responses. The chatbot logic can be as complex as you want, depending on the scope of your project.

6. *Route for Chatbot*: Create a Flask route to handle chatbot interactions. Typically, this route should accept user input, pass it to the chatbot logic, and return the bot's response.

7. *User Input Handling*: In your HTML template, use JavaScript to capture user input and send it to the Flask route that communicates with the chatbot.

8. *Bot Response Display*: Update the chat window in the HTML template with the bot's responses. This is typically done using JavaScript to dynamically update the content.

9. *Session Management*: You may want to implement session management to maintain context between user inputs.

10. *Testing*: Test your web app thoroughly to ensure that the chatbot works as expected.

11. *Deployment*: Once everything is working as expected, deploy your Flask app to a hosting platform of your choice.

Here's a very simplified example to get you started:

```
python

from flask import Flask, request, render_template


app = Flask(__name)


# Sample chatbot logic

def chatbot_response(user_input):

    # Your chatbot logic here

    return "Bot: You said, " + user_input


@app.route('/')

def home():

    return render_template('index.html')


@app.route('/chat', methods=['POST'])

def chat():

    user_input = request.form['user_input']

    bot_response = chatbot_response(user_input)

    return bot_response
```

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

Now start developing the Flask framework based on the above ChatterBot in the above steps.

We have already installed the Flask in the system, so we will import the Python methods we require to run the Flask microserver.

Step 1: Import necessary methods of Flask and ChatterBot.

```
from flask import Flask, render_template, request
from chatterbot import ChatBot
from chatterbot.trainers import ChatterBotCorpusTrainer
```

Step 2: Then, we will initialize the Flask app by adding the below code.

```
#Flask initialisation
app = Flask(__name__)
```

Flask(name) is used to create the Flask class object so that Python code can initialize the Flask server.

Step 3: Now, we will give the name to our chatbot.

```
chatbot=ChatBot('Pythonscholar')
```

Step 4: Add training code for a chatbot.

```
# Create a new trainer for the chatbot
trainer = ChatterBotCorpusTrainer(chatbot)

# Now, let us train our bot with multiple corpus
trainer.train("chatterbot.corpus.english.greetings",
              "chatterbot.corpus.english.conversations" )
```

Step 5: We will create the Flask decorator and a route in this step.

```
@app.route("/")
def index():
        render_template("index.html")
```

The route() is a function of a Flask class used to define the URL mapping associated with the function.

Then we make an index function to render the HTML code associated with the index.html file using the render_template function.

In the next step, we will make a response function that will take the input from the user, and also, it will return the result or response from our trained chatbot.

Step 6: Create a function to take input from the user and respond accordingly.

```
@app.route("/get", methods=["POST"])
def chatbot_response():
        msg = request.form["msg"]
        response = chatbot.get_response(msg)
        return response
```

Step 7: Then, we will add the final code that will start the Flask server after interpreting the whole code.

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

The complete code will look like this:

```
from flask import Flask, render_template, request
from chatterbot import ChatBot
from chatterbot.trainers import ChatterBotCorpusTrainer

#Flask initialization
app = Flask(__name__)

chatbot=ChatBot('Pythonscholar')

# Create a new trainer for the chatbot
trainer = ChatterBotCorpusTrainer(chatbot)

# Now let us train our bot with multiple corpus
trainer.train("chatterbot.corpus.english.greetings",
              "chatterbot.corpus.english.conversations" )

@app.route("/")
```

```
def index():

        return render_template("index.html")



@app.route("/get", methods=["GET","POST"])
def chatbot_response():
    msg = request.form["msg"]
    response = chatbot.get_response(msg)
    return str(response)

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

As per Jinja2 implementation, we have to create two folders for storing HTML and CSS files; while working with the Jinja2 engine, it is necessary to make a folder with a name template to add HTML files, and for other files like CSS, javascript, or image we have to make a folder with the name static but it optional but creating template folder is compulsory.

First, we will make an HTML file called index.html inside the template folder.

```
<!DOCTYPE html>
<html>


<head>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='style.css')}}" />
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
</head>


<body>
    <div class="row">
```

```
        <div class="col-md-10 mr-auto ml-auto">
    <h1>Pythonscholar ChatBot</h1>
    <form>
        <div id="chatbox">
            <div class="col-md-8 ml-auto mr-auto">
                <p class="botText"><span>Hi! I'm Pythonscholar.</span></p>
            </div>
        </div>
        <div id="userInput" class="row">
            <div class="col-md-10">
                <input id="text" type="text" name="msg" placeholder="Message"
class="form-control">
                <button type="submit" id="send" class="btn btn-warning">Send</button>
            </div>
        </div>
    </form>
</div>
</div>

<script>
    $(document).ready(function() {
        $("form").on("submit", function(event) {
            var rawText = $("#text").val();
            var userHtml = '<p class="userText"><span>' + rawText + "</span></p>";
            $("#text").val("");
            $("#chatbox").append(userHtml);
            document.getElementById("userInput").scrollIntoView({
                block: "start",
                behavior: "smooth",
            });
            $.ajax({
                data: {
                    msg: rawText,
                },
                type: "POST",
                url: "/get",
            }).done(function(data) {
```

```
                var botHtml = '<p class="botText"><span>' + data + "</span></p>";
                $("#chatbox").append($.parseHTML(botHtml));
                document.getElementById("userInput").scrollIntoView({
                    block: "start",
                    behavior: "smooth",
                });
            });
            event.preventDefault();
        });
    });
</script>
</body>

</html>
```

We will create a style.css file, save it in the static folder, and use the below code.

```
body {
    font-family: Garamond;
}

h1 {
    color: black;
    margin-bottom: 0;
    margin-top: 0;
    text-align: center;
    font-size: 40px;
}

h3 {
    color: black;
    font-size: 20px;
    margin-top: 3px;
    text-align: center;
```

```
}
.row {
    display: flex;
    flex-wrap: wrap;
    margin-right: -15px;
    margin-left: -15px;
}

.ml-auto{
    margin-left:auto !important;
}
.mr-auto{
    margin-right:auto !important;
}

.col-md-10,.col-md-8,.col-md-4{
    position: relative;
    width: 100%;
    min-height: 1px;
    padding-right: 15px;
    padding-left: 15px;
}
.col-md-8{flex:0 0 66.666667%;max-width:66.666667%}
.col-md-4{flex:0 0 33.333333%;max-width:33.333333%}
.col-md-10{flex:0 0 83.333333%;max-width:83.333333%}

.form-control {
    background: no-repeat bottom,50% calc(100% - 1px);
    background-image: none, none;
    background-size: auto, auto;
    background-size: 0 100%,100% 100%;
    border: 0;
    height: 36px;
    transition: background 0s ease-out;
    padding-left: 0;
    padding-right: 0;
```

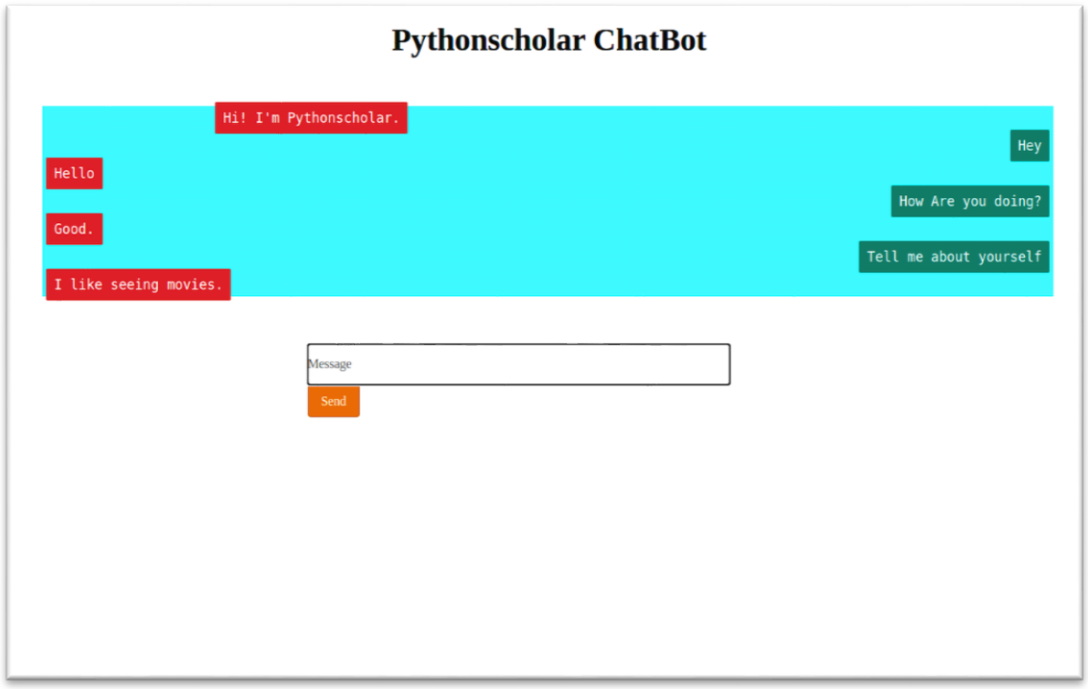
```
    border-radius: 0;
    font-size: 14px;
}
.form-control {
    display: block;
    width: 100%;
    padding: .4375rem 0;
    padding-right: 0px;
    padding-left: 0px;
    font-size: 1rem;
    line-height: 1.5;
    color: #495057;
    border:none;
    background-color: transparent;
    background-clip: padding-box;
    border-bottom: 1px solid #d2d2d2;
    box-shadow: none;
    transition: border-color .15s ease-in-out,box-shadow .15s ease-in-out;
}
.btn {
    float: left;
    text-align: center;
    white-space: nowrap;
    vertical-align: middle;
    user-select: none;
    border: 1px solid transparent;
    padding: .46875rem 1rem;
    font-size: 1rem;
    line-height: 1.5;
    border-radius: .25rem;
    transition: color .15s ease-in-out,background-color .15s ease-in-out,border-color
.15s ease-in-out,box-shadow .15s ease-in-out;
}
.btn-warning {
    color: #fff;
    background-color: #f08f00;
    border-color: #c27400;
```

```
}
.btn.btn-warning:active, .btn.btn-warning:focus, .btn.btn-warning:hover {
    box-shadow: 0 14px 26px -12px rgba(255,152,0,.42),0 4px 23px 0 rgba(0,0,0,.12),0 8px 10px -5px rgba(255,152,0,.2);
}

button, input, optgroup, select, textarea {
    margin: 0;
    font-family: inherit;
    font-size: inherit;
    line-height: inherit;
    overflow:visible;
}
#chatbox {
    background-color: cyan;
    margin-left: auto;
    margin-right: auto;
    width: 80%;
    min-height: 70px;
    margin-top: 60px;
}

#userInput {
    margin-left: auto;
    margin-right: auto;
    width: 40%;
    margin-top: 60px;
}

#textInput {
    width: 87%;
    border: none;
    border-bottom: 3px solid #009688;
    font-family: monospace;
    font-size: 17px;
}
```

```css
#buttonInput {
    padding: 3px;
    font-family: monospace;
    font-size: 17px;
}

.userText {
    color: white;
    font-family: monospace;
    font-size: 17px;
    text-align: right !important;
    line-height: 30px;
    margin: 5px;
}

.userText span {
    background-color: #009688;
    padding: 10px;
    border-radius: 2px;
}

.botText {
    color: white;
    font-family: monospace;
    font-size: 17px;
    text-align: left;
    line-height: 30px;
    margin: 5px;
}

.botText span {
    background-color: #ef5350;
    padding: 10px;
    border-radius: 2px;
}
```

```
#tidbit {
    position: absolute;
    bottom: 0;
    right: 0;
    width: 300px;
}
```

We are all set to run our Flask application.



This is our Chatbot output.

Congratulations, we have successfully built a chatbot using Python and Flask.

CONCLUSION

To conclude in this phase of our project in chatbot we integrated a chatbot into a web app with Flask can be a powerful way to provide interactive and dynamic user experiences. By following the steps outlined, you can create a seamless user experience, allowing users to interact with your chatbot through a web interface. The possibilities are vast, from creating customer support chatbots to educational tools. With the foundational structure in place, you can expand and customize your project further to meet the specific needs of your audience and provide a valuable service or experience.