Python Flask – Request Object

Akash7

In a Flask App, we have our own Webpage (Client) and a Server. The Server should process the data. The **Request**, in Flask, is an object that contains all the data sent from the Client to Server. This data can be recovered using the GET/POST Methods. POST is used when your application expects user input to be received by command or an HTTP request, while GET gets all the information before it even has a chance for submission. Using both methods at once gives perfect freedom but still requires complex UI patterns like AJAX calls etc, with most common applications being frameworks where multiple forms are necessary such as Slack Webhooks, MailgunMailserver, and eCommerce Commerce Framework. With Flask-Request class instead, we don't have any need anymore since this API allows us flexible handling of many other situations.

There are various attributes associated with the request objects. They are Form, Cookies, Args, Files, and Method.

The form is used to create and return the form data from other application codes like JSP API or POST. This attribute allows you to specify predefined custom parameters when creating HTTP requests for your service (like "GET /info") which can be given as string values without any format validation etc.

The **cookies** property contains cookie contents that will allow the client to access a piece of information about users' details through the Cookie header field of returned JSON Object Content Type Information.

Args is used to initialize a Request constructor which you have given as a parameter of the class.

Files are used to refer to file data stored in the user's cookies (whereas a method is an HTTP API for accessing those files). Actions can be chained together so as far it goes you will need different Actions when calling other methods by default like POST /items/:id or GET /API. So if we want to add multiple items to the item list then there must also be some additional actions on both end-points that do something similar and which cannot be handled easily using form objects alone – eg create a new "form" from existing ones, etc.

The method is used to create a new form element or an existing one via custom URL configuration.

Let us consider the below example of Retrieving user entered form data from Webpage and Displaying it on another web page. This Requires 3 files – app.py (Main Python File), Temp.html (That contains the Form), and result_data.html (That displays the Output). You should make sure all other required libraries have been installed for this example in order not to run out of namespaces. In the main route, we just render the Temp.html file. The HTML File contains a form that asks for Name, Email, and Phone Number. Once Submit button is pressed, it returns to **/passing** route in python, which processes the Form if the method is POST. The flask Request Method had two options – POST and GET, using which form data can be processed/Retrieved. Request.form is used to execute a batch of requests, such as checking if the user has provided any password when requested by calling display(). This method will return an object containing information about each request made during that time (whereas Requests.listing returns only objects from matching filters). Request.form is used to render the requested text in JSON format, as it allows you to use multiple forms of presentation for different content types such as emails, links, etc.

Now the data is passed to result_data.html, which displays the data as a Table.

Github Link for the Code is at : https://github.com/akashakash24/Geeks-For-Geeks/tree/Flask_Request

Python3

```python
# Importing Necessary Modules
from flask import *
app = Flask(__name__)

# Create a Main route here
@app.route('/')
def input():
    return render_template('Temp.html')

# Create other routes here.
# host/passing will be the website link
@app.route('/passing', methods=['GET', 'POST'])
def display():
    if request.method == 'POST':
        result = request.form

        # Send result data to result_data HTML file
        return render_template("result_data.html", result=result)


# main route to start with
if __name__ == '__main__':
    app.run(debug=True)
```

HTML

```html
Temp.html
<html>
<style>
body {
    text-align: center;
    background-color: green;
}
form {
    display: inline-block;
}
</style>
   <body>

      <h3>Hey Folks, Please Fill out this Form</h3>
     <form action = "/passing" method = "POST">
        <p>Name <input type = "text" name = "name" /></p>
        <p>Email <input type = "email" name = "email" /></p>
        <p>Phone Num <input type = "text" name = "phone" /></p>
        <p><input type = "submit" value = "Submit" /></p>
     </form>
     <h4>Created by Akash, Geeks for Geeks</h4>
   </body>
</html>
```

HTML

```html
result_data.html
<!doctype html>
<html>
<style>
body {
    text-align: center;
    background-color: Orange;
}
table {
    display: inline-block;
    border-collapse: collapse;
}
</style>
   <body>
     <p><strong>Registration Successful, Please see your details</strong>
     <table border = 1>
        {% for key, value in result.items() %}
           <tr>
              <th> {{ key }} </th>
              <td> {{ value }} </td>
           </tr>
        {% endfor %}
     </table>
   </body>
</html>
```

Run the file using python app.py.

C:\Personal\GFG\Oct-Dec2022\Flask>python app.py

** Serving Flask app 'app'*

** Debug mode: on*

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

** Restarting with stat*

** Debugger is active!*

** Debugger PIN: 259-675-897*

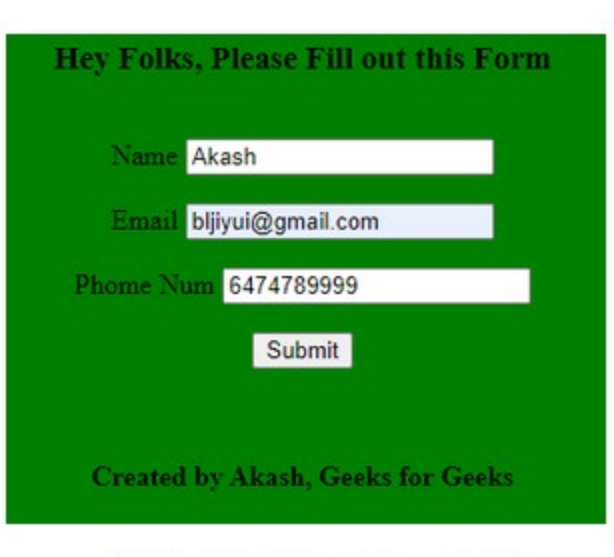

Fig 1.2 – Enter data and Press submit

Fig 1.3 – Output seen on http://127.0.0.1:5000/passing

Last Updated : 29 Mar, 2023

Article Contributed By :
Akash7
Akash7
+ Follow

Improved By : sagartomar9927
Article Tags : Picked, Python Flask, Technical Scripter 2022, Python, Technical Scripter
Practice Tags : python