**HTTP Methods**

HTTP (HyperText Transfer Protocol) methods are used to perform actions on web resources. Here’s a list of the most commonly used HTTP methods:

1. **GET**: Requests data from a specified resource.
2. **POST**: Submits data to be processed to a specified resource.
3. **PUT**: Updates a current resource with new data.
4. **DELETE**: Deletes a specified resource.
5. **HEAD**: Similar to GET, but it retrieves only the headers and not the body of the resource.
6. **OPTIONS**: Describes the communication options for the target resource.
7. **PATCH**: Partially updates a specified resource.

Example:

1.Create a **templates** directory in your project folder and place an index.html file in it with the form content.

**index.html**:

<!DOCTYPE html>

<html>

<head>

    <title>Form Example</title>

</head>

<body>

    <!-- GET Form -->

    <form method="GET" action="/get">

        <input type="text" name="get\_data" placeholder="Enter some data">

        <input type="submit" value="GET Request">

    </form>

    <br>

    <!-- POST Form -->

    <form method="POST" action="/post">

        <input type="text" name="post\_data" placeholder="Enter some data">

        <input type="submit" value="POST Request">

    </form>

    <br>

    <!-- PUT Form -->

    <form method="POST" action="/put?\_method=PUT">

        <input type="text" name="put\_data" placeholder="Enter new data">

        <input type="submit" value="PUT Request">

    </form>

    <br>

    <!-- DELETE Form -->

    <form method="POST" action="/delete?\_method=DELETE">

        <input type="text" name="delete\_data" placeholder="Enter data to delete">

        <input type="submit" value="DELETE Request">

    </form>

    <br>

    <!-- PATCH Form -->

    <form method="POST" action="/patch?\_method=PATCH">

        <input type="text" name="patch\_data" placeholder="Enter data to update">

        <input type="submit" value="PATCH Request">

    </form>

</body>

</html>

1. Create a **Flask** application to handle the form submissions.

**app.py**:

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/get', methods=['GET'])

def get\_example():

    data = request.args.get('get\_data')

    return f'Received GET data: {data}'

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

def post\_example():

    data = request.form['post\_data']

    return f'Received POST data: {data}'

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

def put\_example():

    try:

        if request.args.get('\_method') == 'PUT':

            data = request.form['put\_data']

            return f'Received PUT data: {data}'

        else:

            return 'PUT request not properly formatted'

    except Exception as e:

        return f'Error processing PUT request: {str(e)}'

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

def delete\_example():

    try:

        if request.args.get('\_method') == 'DELETE':

            data = request.form['delete\_data']

            return f'Deleted data: {data}'

        else:

            return 'DELETE request not properly formatted'

    except Exception as e:

        return f'Error processing DELETE request: {str(e)}'

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

def patch\_example():

    try:

        if request.args.get('\_method') == 'PATCH':

            data = request.form['patch\_data']

            return f'Patched data: {data}'

        else:

            return 'PATCH request not properly formatted'

    except Exception as e:

        return f'Error processing PATCH request: {str(e)}'

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Flask -Templates**

Flask uses Jinja2 as its template engine to render dynamic web pages.

* **Creating Templates**: Store your HTML templates in a directory named templates in your project folder.
* **Rendering Templates**: Use the render\_template function to render HTML templates.

Structure of Project

Set3\_1/

│

|── templates/

│ ├── index.html

|── static/

└── styles.css

├── app.py

app.py

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

    return render\_template('index.html')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

templates/

index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Simple Web Server</title>

</head>

<body>

    <h1>Welcome to the Simple Web Server!</h1>

    <p>This is a static HTML page served by Flask.</p>

</body>

</html>

Static/

Styles.css

/\* static/styles.css \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f0f0;

margin: 0;

padding: 0;

}

h1 {

color: #333;

text-align: center;

margin-top: 20px;

}

p {

color: #666;

text-align: center;

}

.container {

width: 50%;

margin: 100px auto;

padding: 20px;

background-color: #fff;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

border-radius: 8px;

}

The **Jinja2** template engine uses the following delimiters for escaping from HTML.

* {% ... %} for **Statements**
* {{ ... }} for **Expressions** to print to the template output
* {# ... #} for **Comments** not included in the template output
* # ... ## for Line Statements

In the following example, use of conditional statement in the template is demonstrated. The URL rule to the **hello()** function accepts the integer parameter. It is passed to the **hello.html** template. Inside it, the value of number received (marks) is compared (greater or less than 50) and accordingly HTML is conditionally rendered.

The Python Script is as follows −

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/hello/<int:score>')

def hello\_name(score):

return render\_template('hello.html', marks = score)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

HTML template script of **hello.html** is as follows −

<!doctype html>

<html>

<body>

{% if marks>50 %}

<h1> Your result is pass!</h1>

{% else %}

<h1>Your result is fail</h1>

{% endif %}

</body>

</html>

Note that the conditional statements **if-else** and **endif** are enclosed in delimiter **{%..%}**.

visit URL **http://localhost/hello/60** and then **http://localhost/hello/30** to see the output of HTML changing conditionally.

The Python loop constructs can also be employed inside the template. In the following script, the **result()** function sends a dictionary object to template **results.html** when URL **http://localhost:5000/result** is opened in the browser.

The Template part of **result.html** employs a **for loop** to render key and value pairs of dictionary object **result{}** as cells of an HTML table.

Run the following code from Python shell.

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/result')

def result():

dict = {'phy':50,'che':60,'maths':70}

return render\_template('result.html', result = dict)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

Save the following HTML script as **result.html** in the templates folder.

<!doctype html>

<html>

<body>

<table border = 1>

{% for key, value in result.items() %}

<tr>

<th> {{ key }} </th>

<td> {{ value }} </td>

</tr>

{% endfor %}

</table>

</body>

</html>

Here, again the Python statements corresponding to the **For** loop are enclosed in {%..%} whereas, the expressions **key and value** are put inside **{{ }}**.

After the development starts running, open **http://localhost:5000/result** in the browser to get the following output.

**Flask – Request Object**

**Sending Form Data to Template.**

 **Form**: A dictionary object containing key-value pairs of form parameters and their values. It's used to access data sent via POST requests.

python

from flask import Flask, request

app = Flask(\_\_name\_\_)

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

def submit():

name = request.form['name']

return f'Hello, {name}!'

 **args**: A dictionary object containing parsed contents of the query string, which is part of the URL after the question mark (?). It's used to access data sent via GET requests.

python

@app.route('/search', methods=['GET'])

def search():

query = request.args.get('query')

return f'Search results for: {query}'

 **Cookies**: A dictionary object holding Cookie names and values. It’s used to access cookies sent by the client.

python

@app.route('/show\_cookies', methods=['GET'])

def show\_cookies():

user\_cookie = request.cookies.get('username')

return f'Username cookie: {user\_cookie}'

 **files**: A dictionary object containing the data pertaining to uploaded files. It’s used to handle file uploads.

python

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

def upload\_file():

file = request.files['file']

return f'File uploaded: {file.filename}'

 **method**: A string representing the current request method (e.g., 'GET', 'POST', 'PUT', etc.).

python

@app.route('/request\_method', methods=['GET', 'POST'])

def request\_method():

return f'The request method is: {request.method}'

**HTTP status codes and their descriptions in Flask:**

| **Status Code** | **Name** | **Description** |
| --- | --- | --- |
| 200 | OK | The request has succeeded. |
| 201 | Created | The request has been fulfilled and resulted in a new resource being created. |
| 204 | No Content | The server successfully processed the request, but is not returning any content. |
| 301 | Moved Permanently | The requested resource has been assigned a new permanent URI. |
| 302 | Found | The requested resource resides temporarily under a different URI. |
| 400 | Bad Request | The server could not understand the request due to invalid syntax. |
| 401 | Unauthorized | The request requires user authentication. |
| 403 | Forbidden | The server understood the request, but refuses to authorize it. |
| 404 | Not Found | The server can't find the requested resource. |
| 405 | Method Not Allowed | The request method is known by the server but has been disabled and cannot be used. |
| 500 | Internal Server Error | The server encountered an unexpected condition that prevented it from fulfilling the request. |
| 502 | Bad Gateway | The server received an invalid response from the upstream server while acting as a gateway or proxy. |
| 503 | Service Unavailable | The server is not ready to handle the request. Common causes include server maintenance or overload. |