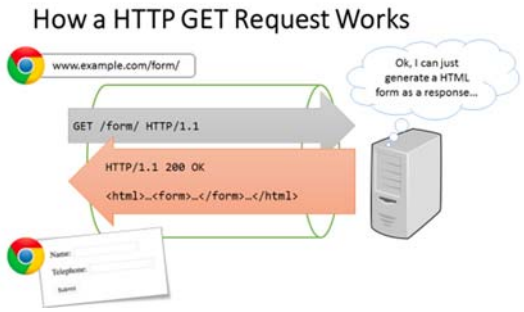
N03_02_Request_Object

In this chapter, students will learn and understand:

- Explain how the browsing of a webpage is submitted as a `"GET"` request method to a server
- Use `flask.request.method` to return a `"GET"` or `"POST"` method
- Explain how web form data is submitted as a `"POST"` request method to a server
- Use `flask.request.form` to return a multi dictionary of field names and their associated values

Section 1: Introduction

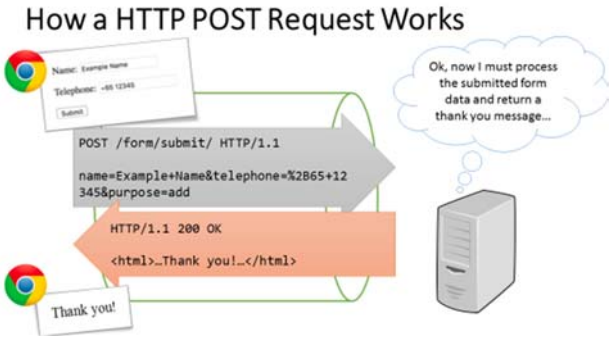
In this lesson, you will learn how a web form is generated when a user types a URL into a web browser and how form data is submitted to a server.



Suppose a client tries to access a webpage `www.example.com/form/` into the web browser to access the web form page, a `GET` request message is sent over to the server. Then what the server does is to render a template and send it over as the response to the browser. When the browser receives the response, it displays it to the user.

Difference between GET and POST requests: [HTTP Methods GET vs POST (w3schools.com)]

What the user will now see is the web form with Name, Telephone form fields and a Submit button. Now the user enters data into the form and clicks Submit.



A `POST` request message with the form data being parsed as the body of the request message is now sent over to the server. The server detects the request as a `POST` through looking at the headers of the request message.

Once the server receives the data, it will send a `HTTP 200 OK` response back to the client to acknowledge that it has received the data. In summary, this is how a client and a server will send and receive data on the HTTP protocol.

Section 2: Creating A Request Object

In this task, you will create a message board that will allow users to post messages onto the class webpage.

1. In **app.py**, import `request`.

2. Add a new function called `form` as shown below. This function will display the request object's method.

```
@app.route('/form/')
def form():
    return request.method
```

What is the default HTTP method used when a client requests for a webpage in the browser?

……..

2.1 Creating A Form using default GET method

1. Create a new template called **message_form.html**. Save it under **templates**.
   message_form.html is a webpage that consists of three fields:
   - `username`
   - `email`
   - `message`

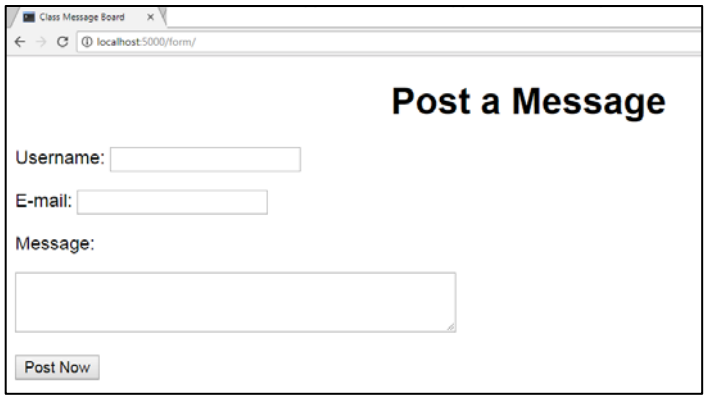
```
<!doctype html>
<html>
<head>
<title>Class Message Board</title>
</head>
<h1>Post a Message</h1>
<body>
<form action = "{{ url_for('result') }}">
        <p>Username: <input type = "text" name = "username"></p>
        <p>E-mail: <input type="email" name ="email"></p>
        <p>Message:</p>
        <p><textarea name = "message" rows="3" cols="50">
         </textarea></p>
       <p><input type="submit" value="Post Now"></p>
    </form>
</body>
</html>
```

Once the client enters the data and hits the **Post Now** button, the browser collates the form data, organises them as `"name=value"` dictionary pairs, and concatenates them together using `"&"` as the field separator. This is known as the **query string**. Then, it will send the query string over to the server using the `GET` method by default. When the control is handled over to the server, the logic written in the Python file will then process the submitted data accordingly.

2. In **app.py**, modify the `form` function to render **message_form.html**.

3. Enter `http://localhost:5000/form`.

The web form should appears as shown below.



Test the webpage by typing in some values and click **Post Now**. What do you observe? Why does this happen?

……

……

……

2.2 Submitting Form Data

1. Create a new webpage and save it as **message_result.html** under **templates** folder. This webpage will display the results of the submitted data.

Take note that no processing of data is done at this point.

```
<!doctype html>
<html>
<head>
<title>Class Message Board</title>
</head>
<body>
<h1>Message Posting</h1>
<p>Good day, {{username}}</p>
<p>Your email is: {{email}}</p>
<p>Your posted message is: {{message}}</p>
</body>
</html>
```

2. In app.py, modify the `form` function such that the data is being sent over to the server through a `POST` request and **message_result.html** is being rendered with the appropriate form data passed in.

```
@app.route('/result/')
def result():
    username = request.args.get('username')
    email = request.args.get('email')
    message = request.args.get('message')

    return render_template('message_result.html',
                           username=username,
                           email=email,
                           message=message)
```

3. Run **app.py**.

4. Enter `http://localhost:5000/form/` in your browser to refresh the web form.

5. Enter some test data. Click **Post Now**.

2.1 Creating A Form using POST method

4. Create a new template called **message_form.html**. Save it under **templates**.
   message_form.html is a webpage that consists of three fields:
   - `username`
   - `email`
   - `message`

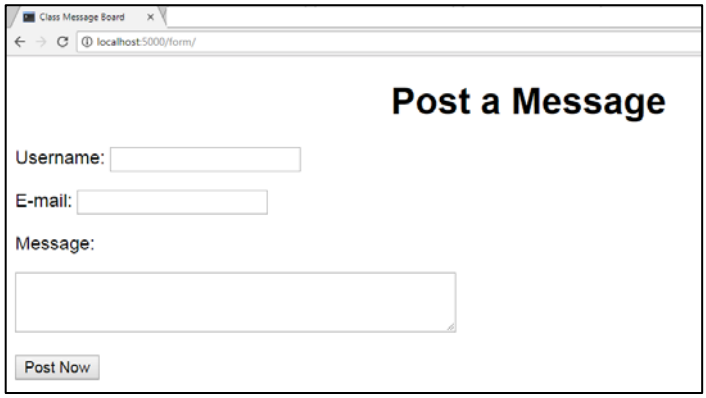
```
<!doctype html>
<html>
<head>
<title>Class Message Board</title>
<link rel="stylesheet" href="/static/css/style.css">
</head>
<h1>Post a Message</h1>
<body>
<form action = "{{ url_for('form') }}" method = "POST">
        <p>Username: <input type = "text" name = "username"></p>
        <p>E-mail: <input type="email" name ="email"></p>
        <p>Message:</p>
        <p><textarea name = "message" rows="3" cols="50">
         </textarea></p>
       <p><input type="submit" value="Post Now"></p>
    </form>
</body>
</html>
```

Once the client enters the data and hits the **Post Now** button, the browser collates the form data, organises them as `"name=value"` dictionary pairs, and concatenates them together using `"&"` as the field separator. This is known as the **query string**. Then, it will send the query string over to the server using the `POST` method specified in the form's attribute `<method>`. When the control is handled over to the server, the logic written in the Python file will then process the submitted data accordingly.

5. In **app.py**, modify the `form` function to render **message_form.html**.

6. Enter `http://localhost:5000/form`.

The web form should appears as shown below.



2.2 Submitting Form Data

6. Create a new webpage and save it as **message_result.html** under **templates** folder. This webpage will display the results of the submitted data.

Take note that no processing of data is done at this point.

```
<!doctype html>
<html>
<head>
<title>Class Message Board</title>
<link rel="stylesheet" href="/static/css/style.css">
</head>
<body>
<h1>Message Posting</h1>
<p>Good day, {{username}}</p>
<p>Your email is: {{email}}</p>
<p>Your posted message is: {{message}}</p>
</body>
</html>
```

7. In app.py, modify the `form` function such that the data is being sent over to the server through a `POST` request and **message_result.html** is being rendered with the appropriate form data passed in.

```
@app.route('/form/', methods=['GET', 'POST'])
def form():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        message = request.form['message']
        return render_template(
            'message_result.html',
            username=username,
            email=email,
            message=message)
    else:
        return render_template('message_form.html')
```

8. Run **app.py**.

9. Enter `http://localhost:5000/form/` in your browser to refresh the web form.

10. Enter some test data. Click **Post Now**.

You should see the data entered in the web form being displayed.

2.3 `flask.request` Object

The data from a client's web page is being sent to the server as a global request object. The attributes of the request object are listed below:

`flask.request`	
`form`	It is a dictionary object containing key and value pairs of form parameters and their values. **Used for retrieving values in query string for POST method.**
`args`	parsed contents of query string which is part of URL after question mark (?). **Used for retrieving values in query string for GET method.**
`method`	current request method (`GET`, `POST`, etc)
`files`	A MultiDict with files uploaded as part of a POST request. Each file is stored as `FileStorage` object. It has a `save()` function that can store the file on the filesystem.

The form data received by the triggered function can collect it in the form of a `dictionary` object and forward it to a template to render it on a corresponding web page.

In the example used above, `'/form/'` renders a web page – **message_form.html** which is first loaded onto a browser through a `HTTP GET` request. When the user enters data into the form and clicks on **Post Now**, the form data is then posted to the `'/form/'` URL through a `HTTP POST` method. This method will trigger the `form()` function.

The form data is stored as a dictionary object. The `form()` function collects the form data present in `request.form` and sends it for rendering to **message_result.html**. The template then dynamically renders the form data and displays the entered data as `{{username}}`, `{{email}}` and `{{message}}` variables on the webpage.

References:

- MOE Python Flask Web Application Starter Kit
- MOE Teacher Training 2018 - HTML and CSS
- Udemy: Python and Flask Bootcamp: Create Websites using Flask!