COSC 419 – Topics in Computer Science

Fall 2020

Recap: Flask Basic Functionality

- Last lecture, we started to look at some of the basic functionality available in Flask:
  - Using templates and passing data into templates
  - Basic routing with variables and redirects
  - Handling GET requests and specifying request types

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/myTemplate')
def templateExample():
        return render_template('hello.html')
```

What About POST Requests?

```
from flask import Flask, request, render_template
app = Flask(__name__)

@app.route('/', methods=["GET","POST"])
def root():
        if request.method == "POST":
                #Code to run if request is a POST request
```

- Like with GET requests, we can specify whether we want a route to catch ***POST requests*** or not in our @app directive using the methods parameter
- We can also check what kind of request has been sent by checking the flask `request.method` object

Reading POST Request Data

- Recall from last lecture that we use `request.args.get()` to fetch GET request data from a query string

```
myVar = request.form.get("myName")
return str(myVar)
```

- The equivalent for fetching POST request data is `request.form.get()`, which we can pass a key into to fetch the matching value in the POST request
- Like the version used for GET, this will return None (Python equivalent to `null`) if the specified key doesn't exist

When to Use GET and POST

- Although both methods send data to the server, the way in which they are sent is important:
  - GET requests pass data as a query string along with the URL in the HTTP request headers
  - POST requests pass data as a set of key-value pairs in the body of an HTTP request
- Why is this important? Because the headers of an HTTP request are not encrypted when we use HTTPS, meaning everything sent via GET request is sent in plaintext

GET Requests and Security

- This means that data sent via our GET requests isn't secure – it can be seen by anyone who happens to see our request:
  - Someone sniffing network traffic data
  - Your ISP
  - Routing servers that pass your request onwards towards it's final destination

Request URL: `http://134.122.38.246/?username=Matt&password=superSecurePassword`

Request Method: `GET`

Status Code: `200 OK`

Remote Address: `134.122.38.246:80`

Referrer Policy: `no-referrer-when-downgrade`

When is GET Appropriate?

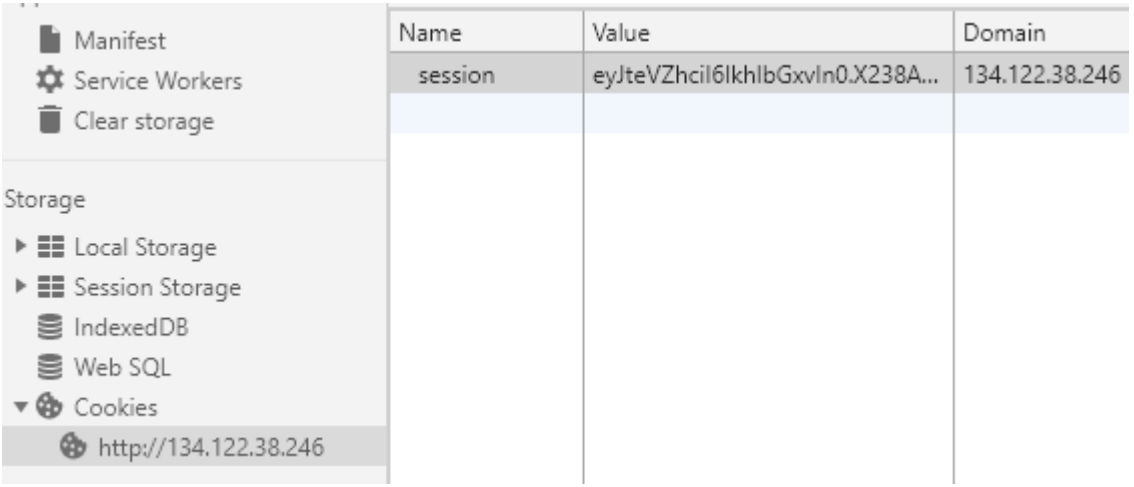
- The general rule of thumb is that if you are simply fetching data and don’t need to do so securely, a GET request is fine
  - Typical applications could be a public API, search parameters, or fetching a public web page
- If we intend to modify data on the server (i.e. write to a database), submit a form, or ensure that data is handled securely, we should use POST
  - Typical applications: submitting form inputs, passing login or other sensitive information, uploading files

Maintaining State in our Application

- So far, we’ve been working in a stateless server environment. What does this mean?
  - No state is held between requests – it is as if each request to the server is from a brand new client that has never connected before
- So what if we want to store information between requests?
  - We can use ***session variables***, which are kept between requests – we can use these to attach data to a particular user session

How Sessions in Flask Work

- Flask sessions serialize the session data, then encrypt it using a secret key that is set in the Flask application



- The encrypted and serialized data is stored on the client-side in the form of a cookie, which is sent back to the server with each request
- The server can then use the secret key to decrypt the cookie, reading the session variables

Setting Our Secret Key

- You can set the secret key for your application by passing the key to our app object:

```
app = Flask(__name__)
app.secret_key = “mySuperAwesomeKey”
```

- The key simply takes a string input of any length – it is recommended that you choose a long, secure key (i.e. not all the same character, and avoiding common keys such as ‘password’)
- The secret key must be set before trying to use sessions

Setting a Session Variable

- Before we can use sessions, we'll need to first import the session object from Flask:

```
from flask import session
```

- Then, we can simply set a session variable using the session object, which is a dictionary of key value pairs:

```
session["key"] = "value"
```

- Flask takes care of generating the cookie and sending it to the client with each response

Accessing Session Variables

- Since the session is effectively a dictionary (like request.args and request.form), we can use the same manner of accessing them with the get() function

```
myValue = session.get(“key”)
```

- As we’ve seen previously, this will return None if the specified key doesn’t exist
- You can also fetch directly from the dictionary, but it will throw an error if the key doesn’t exist

```
myValue = session[“key”]
```

Clearing Session Variables

- We can also remove variables from the session, using the built in `pop()` method:

```
session.pop("key", None)
```

- The pop function will remove a variable from the session and return it's value, or if the key is not set, a secondary default value (in this case, None)
- If no default value is set, pop will throw an error if the specified key doesn't exist – so you should always set a default, unless you want to throw an error

A Typical Session Use Case

- One of the classic uses of session variables is to maintain whether a user is logged in or not
- The user input for username and password is checked, and if it matches an existing user, a session variable `loggedIn` is set to true
- For protected pages, the `loggedIn` session variable is checked, and the user is redirected to a non-protected page if it doesn’t exist
- When a user logs out, we pop the `loggedIn` from the session

Nesting Templates in Flask

- Last lecture we began discussing templates in Flask
- Templates are simply HTML files with some additional markup that allows us to handle basic logic (if/else, loops) and mark where we want to echo data to using the double-braces notation, **{{ myVar }}**
- This alone makes templates quite powerful, but we actually do even more with them by using ***template inheritance***
  - This allows templates to be nested within other templates

The Template Block Directive

- Within a template, we can specify that we want to render another template or chunk of HTML inside using the block directive:

```
{% block myPage %} {% endblock %}
```

- This directive tells the templating engine that if this templated is ***extended***, this block will be replaced with HTML from another template
- This allows templates to be nested within one another

A Simple Example

- Consider two templates, beginning first with the ‘parent’ template:

template1.html

```
<h1>Hello</h1>
{% block insertHere %} {% endblock %}
```

- And the ‘child’ template that will extend the parent – note that the block names must be identical:

template2.html

```
{% extends "template1.html" %}
{% block insertHere %}
    <h1>World!</h1>
{% endblock %}
```

How to Render Nested Templates

- In order to render and return a nested template, we can use the `render_template` function as we did before
- Note that if we want to render the nested templates, we'll need to render the child, ***not*** the parent
  - If you render the parent, it won't fill the block with the child HTML – instead it will just be empty there
- Rendering a template that extends another template will automatically render both and insert the HTML into the appropriate block (if necessary)

Why Template Nesting is Powerful

- This might not seem like a particular fancy or powerful feature – after all, it’s just tacking together pieces of HTML, right?
- But this can save us a lot of time and effort – you can build a base template for your site (i.e. navbars, backgrounds, basic layout) and then fill it with pre-templated content – no duplication of HTML across templates
- Also has the benefit of keeping style and layout consistent across your website

Any Questions?