CSE 106

Lecture 12 – Web Frameworks (Flask)

Acknowledgement: flask.palletsprojects.com, tutorialspoint.com/flask

Fetch API: Alternative to XMLHttpRequest

- ES6 feature supported in modern browsers since 2015
- Avoids callbacks by using promises
  - let promise = fetch(url, [options])
- Calls the code in the “then” once response is received

```
fetch('https://api.github.com/users')
    .then(function (response) {
        console.log(response);
    })
    .catch(function (err) {
        console.log("Something went wrong!", err);
    });
```

Fetch API: POST

```
const data = { username: 'example' };
fetch('https://example.com/profile', {
    method: 'POST',
    headers: {
       'Content-Type': 'application/json',
    },
    body: JSON.stringify(data),
})
    .then((response) => {
        console.log('Success:', response.json());
    })
    .catch((error) => {
        console.error('Error:', error);
    });
```

Fetch API: with async await

```
let response = await fetch('https://example.com/profile');

if (response.ok) { // if HTTP-status is 200-299
  // get the response body (the method explained below)
  let json = response.json();
} else {
  alert("HTTP-Error: " + response.status);
}
```

Web Frameworks

- Provide a standard way to build web apps (library)
- Different frameworks for backend and frontend development
  - Front end: Angular, React, Vue, Ember, Backbone
  - Back end: Flask, Django, Express, Rails, Laravel, Spring
- Backend framework may provide libraries for:
  - API routing
  - Templating
  - Session management
  - Database access
  - Authentication

Flask

- Backend web framework written in Python
- Developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco
- Based on the Werkzeug WSGI toolkit and Jinja2 template engine
  - Web Server Gateway Interface (**WSGI**) - a specification for a universal interface between the web server and the web applications for Python
  - **Werkzeug** - a WSGI toolkit, which implements requests, response objects, etc.
  - **Jinja2** - a popular templating engine for Python which lets you more easily render dynamic web pages

Flask

- A “micro framework” - aims to keep the core simple but extensible
- Not opinionated, leaving decisions up you
- Doesn’t include database abstraction layer, form validation, or other features
- Supports extensions to easily integrate with existing libraries
  - Database integration
  - Form validation
  - Upload handling
  - Authentication
  - More

Flask installation

- Flask supports Python 3.6 and newer
- Recommend PyCharm (Professional is free for students)
- Install Python and pip locally
- Use a virtual environment
  - Sets up an environment with custom libraries for a specific project
  - Packages installed for one project will not affect other projects or the operating system’s packages
- Install flask with pip

```
pip install Flask
```

Flask – Hello World

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

@app.route('/')
def hello_world():
  return 'Hello World'

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

Flask - Hello World

- The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function

```
@app.route(rule, options)
```

  - The rule parameter represents URL binding with the function
  - The options is a list of parameters to be forwarded to the underlying Rule object
- The **run()** method of Flask class runs the application on the local development server

```
app.run(host, port, debug, options)
```

Flask - Routing

- The route() decorator in Flask is used to bind URL to a function
- Below URL ‘/hello’ rule is bound to the hello_world() function
- If a user visits *http://localhost:5000/hello* URL, the output of the hello_world() function will be rendered in the browser

```
@app.route('/hello')
def hello_world():
  return 'hello world'
```

Flask – Variable Rules

- Dynamically build a URL by adding a <variable-name> to URL
- Passed as a keyword argument to the function with which the rule is associated
- Go to URL/hello/Sam and it should say: “Hello Sam!”

```
@app.route('/hello/<name>')
def hello_name(name):
  return 'Hello %s!' % name
```

Flask – Variable Rules

- The default variable is a string, but it can convert to int or float

```
@app.route('/blog/<int:postID>')
def show_blog(postID):
  return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')
def revision(revNo):
  return 'Revision Number %f' % revNo
```

Flask – URL Building

- The **url_for()** function accepts the name of a function as first argument and one or more keyword arguments
- The **redirect** function redirects to a specific URL

```
@app.route('/admin')
def hello_admin():
  return 'Hello Admin’

@app.route('/guest/<guest>')
def hello_guest(guest):
  return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
  if name =='admin':
    return redirect(url_for('hello_admin'))
  else:
    return redirect(url_for('hello_guest',guest = name))
```

Flask – Static files

- Static files such as javascript or css need to be served by the server and go in the /static folder
- HTML files go in the /template folder in the Flask project

```
<html>
 <body>
  <h1>Hello World!</h1>
 </body>
</html>
```

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

Wake-up!

- https://youtu.be/nc9HTPI1vDE

Full stack with Flask – HTTP method w/ form

- Specify the HTTP methods in the route, under methods (e.g. POST)

```
<html>
   <body>
      <form action = "http://localhost:5000/login" method = "post">
         <p>Enter Name:</p>
         <p><input type = "text" name = "name" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

← Frontend

```
@app.route('/success/<name>')
def success(name):
   return 'welcome %s' % name

@app.route('/login',methods = ['POST'])
def login():
    user = request.form['name']
    return redirect(url_for('success',name = user))
```

← Backend

Full stack with Flask – HTTP method w/ AJAX

```
<div>
  <button type="button" onclick="getBookInfo()">Display Book Info</button>
    <h3>Author: <span id="author"></span> </h3>
    <h3>Title: <span id="title"></span> </h3>
</div>
<script>
  function getBookInfo() {
    const xhttp = new XMLHttpRequest();
    xhttp.open("GET", "/bookinfo", true);
    xhttp.send();
    xhttp.onload = function() {
        const response = JSON.parse(this.responseText);
        document.getElementById("author").innerHTML = response.author;
        document.getElementById("title").innerHTML = response.title;
    };
  }
</script>
```

← Frontend

```
@app.route("/bookinfo")
def bookInfo():
    book = {
            "author":"JRR Tokien",
            "title":"The Hobbit"
        }
    return json.dumps(book)
```

← Backend

Flask – Templates

- HTML file can be rendered by the render_template()

```
@app.route('/')
def index():
  return '<html><body><h1>Hello World</h1></body></html>'
```

This is cumbersome!

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

This is better

Flask – Templates

- Flask will try to find the HTML file in the templates folder, in the same folder in which this script is present

```
Application folder
   Hello.py
   templates
      hello.html
```

Flask – Templates

- Jinja2 template engine allows you to render html files with variables and code imbedded in the html file

```
<html>
 <body>
  <h1>Hello {{ name }}!</h1>
 </body>
</html>
```

← Frontend

```
@app.route('/hello/<user>')
def hello_name(user):
  return render_template('hello.html', name = user)
```

← Backend

Flask – Templates

- 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

Flask – Templates

```
<html>
  <body>
    {% if percent > 60 %}
      <h1> Your result is pass!</h1>
    {% else %}
      <h1>Your result is fail</h1>
    {% endif %}
  </body>
</html>
```

← Frontend

```
@app.route('/hello/<int:score>')
def hello_name(score):
  return render_template('score.html', percent = score)
```

← Backend

Flask – Templates

```
<body>
  <table border = 1>
    {% for key, value in result.items() %}
      <tr>
        <th> {{ key }} </th>
        <td> {{ value }} </td>
      </tr>
    {% endfor %}
  </table>
</body>
```

← Frontend

```
@app.route('/result')
def result():
  dict = {'physics':72,'chem':58,'math':81}
  return render_template('result.html', result = dict)
```

← Backend

Templates vs AJAX

- Templates
  - Easy way to directly render html file populated with data
  - Good for when you want to render the whole page with data at once
  - Requires page load for data changes
- AJAX
  - Good for when you want some content on page to update
  - Best for calling REST APIs