COSC 419
Lab 2: Setting Up Flask and Building our First Flask App

In this lab, we'll be installing and configuring the Flask framework, before starting work on our first Flask app.

Configuring WSGI

To begin, you'll first need to establish an SSH connection to your remote server. You can go back and reference the instructions in Lab 1 if you require assistance. Once you've connected, we'll run our first step of the installation: installing the Web Server Gateway Interface (WSGI) that will connect Python with Apache. Run the following command:

```
sudo yum install python3-mod_wsgi
```

Confirm the installation (You may need to enter your sudo password) and wait for it to complete. Once it is finished, we can begin configuring WSGI to interact with our web application. Navigate to the following folder:

```
/etc/httpd/conf.d
```

You may wish to log in as the root user (you can use the `su` command to do this) to navigate to this folder if you do not have permissions. Then, we'll need to create a new file, which we will call `wsgi.conf`. You may use `nano` or `vim` for this, whichever you're more comfortable doing.

Copy the following code into your `wsgi.conf` file. The lecture two notes detail what this configuration file is doing.

```
<VirtualHost *>
    WSGIDaemonProcess myapp user=apache group=apache threads=5
    WSGIScriptAlias / /var/www/html/myapp.wsgi

    <Directory /var/www/html>
        WSGIProcessGroup myapp
        WSGIApplicationGroup %{GLOBAL}
        Order deny,allow
        Allow from all
    </Directory>
</VirtualHost>
```

Setting Up Flask

We'll step away from WSGI temporarily to begin our installation of the Flask framework. Flask can be installed easily using `pip`, a package manager for the Python language. Run the following command to install Flask:

```
sudo pip3 install Flask
```

It will then download flask and the required dependencies. Once we've downloaded Flask, we can continue on with setting up WSGI and getting a barebones web page working. First, navigate to your web root at `/var/www/html`, and then create a new file called `myapp.wsgi`. Inside this file, include the following code:

```
import sys
sys.path.insert(0, '/var/www/html')
from myapp import app as application
```

Save and close the file, and then create a second file in the same folder named myapp.py. This will serve as the main application file for our Flask application. Open it up and write the following content to it:

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

@app.route('/')
def helloWorld():
      return 'Hello World!'
```

Save the file and then close it. This is our basic flask app, with a single route at the root URL (or in our case, IP address). All that is left to do is restart our server:

```
sudo service httpd restart
```

Once you've done that, try visiting your web server's IP in your web browser. If the installation went properly, you should see a 'Hello World!' message being served by Flask. Congratulations! You've successfully installed and configured your Flask web application. Now we can do something fun with it.

To prevent permissions errors when running our code in this lab, we will need to disable SELINUX. SELINUX is a secondary access control schema in CentOS and RedHat. It can be set to permissive mode with the following command:

```
sudo setenforce permissive
```

Building a One-Page Web App

For our first web app, we'll build a simple single page application. It will ask the user for an input URL via an HTML5 form, fetch the HTML from that URL, and then return a list of the top 10 keywords (by frequency) that appear there.

To start you off, I've included a Python script file called `scrape.py`. This file includes all the functions necessary to fetch HTML from a URL, clean it, remove stopwords (common words such as "and", "I", "we", etc), and return a sorted dictionary of keywords from most common to least common. The script also includes a commented-out section as an example of how to use these functions together.

You may either copy-paste the contents of the `scrape.py` file directly into your `myapp.py` file, or you may include the `scrape.py` file in your web root and then import it into your `myapp.py`.

I've also included a very basic HTML5 file for this lab. The file is called form.html and includes a very basic HTML5 text input form which takes one user input (a URL to check) and includes a space for you to return your keywords.

You may either copy/paste the contents of form.html into a new file, or copy it onto your server. You should use `form.html` as a template, and you will need to modify it slightly by adding in a variable to print out your keywords.

Your application should have one route on the home/root ('/'). In your route function, you will need to do the following (in roughly this order):

1. Check to see if there is a GET query string parameter called "url" – this means the form has been submitted.
2. If the parameter exists, read it and run it through the first scraping function to fetch the web page HTML, then run it through the rest of the scraping functions to clean it up.
3. If there was a URL to scrape and you've scraped it, now you'll want to fetch the first 10 items in the sorted dictionary produced by `getCountDict()` (the top 10 keywords), and find some way to format them in a "pretty" format.
4. Render the form.html template and return it. If there was data to scrape, include your "prettified" top 10 keywords in the template by passing the data to the template.

The information needed to complete this task is available in the lecture three notes. A hint: you'll need a basic none-check in your template to handle cases where no keyword data is sent (when the user hasn't submitted a URL yet).

Note: You can assume the user will submit a proper URL to the application. You do not need to implement input validation/checking.

Submission

There is no required submission for this lab. You will be marked directly on the state of your server, using the following schema:

Marks	Item
1 Mark	Flask configuration and Install
2 Marks	Handling GET Request parameters
1 Mark	Parse and return 10 keywords
2 Marks	Correct usage of template system to serve form AND keyword data