Tutorial: Flask

Introduction

As required by assessment 1, the student needs to write an application(of your choice of language) to perform the tasks required by the specification. One of the requirements is that the application can be rendered in a web browser by directly entering the root Public IPv4 DNS of the EC2 instance (in either https or http). As http/https are required, the application needs to renderer **HTML** to respond to the request from the user.

What is Flask?

Flask is a web framework. This means Flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website. (https://pymbook.readthedocs.io/en/latest/flask.html)

Why Flask?

This is not a Flask or web application tutorial. Instead, this showcases one way to achieve the task. The students are free to use other frameworks or programming languages, provided that the student writes the code to perform the task. Since Flask is written in Python, the following example is Python-based.

In a nutshell, you will create and reuse the HTML templates and inject the “dynamic” content to the HTML template.

The basic structure of a Flask application

-app.yaml. The app.yaml is required by the GCP app engine. We will only run the code locally, which the YAML file is not needed.

-Static/ The folder to store the files like css and js

-Templates/ The folder to store the HTML templates.

Main.py

```
# [START gae_python3_render_template]
import datetime

from flask import Flask, render_template

app = Flask(__name__)


@app.route('/')
def root():
    # For the sake of example, use static information to inflate the template.
    # This will be replaced with real information in later steps.
    dummy_times = [datetime.datetime(2018, 1, 1, 10, 0, 0),
                   datetime.datetime(2018, 1, 2, 10, 30, 0),
                   datetime.datetime(2018, 1, 3, 11, 0, 0),
                   ]

    return render_template('index.html', times=dummy_times)


if __name__ == '__main__':
    # This is used when running locally only. When deploying to Google App
    # Engine, a webserver process such as Gunicorn will serve the app. This
    # can be configured by adding an `entrypoint` to app.yaml.
    # Flask's development server will automatically serve static files in
    # the "static" directory. See:
    # http://flask.pocoo.org/docs/1.0/quickstart/#static-files. Once deployed,
    # App Engine itself will serve those files as configured in app.yaml.
    app.run(host='127.0.0.1', port=8080, debug=True)
```

In the above example, the application generates the date/time and uses the render_template() function to inject the content **times=dummy_times** with the template index.html.

Index.html Please see how the injected content and HTML are mixed and used to render the output.

```
<html>
<head>
  <title>Datastore and Firebase Auth Example</title>
  <script src="{{ url_for('static', filename='script.js') }}"></script>
  <link type="text/css" rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>

  <h1>Datastore and Firebase Auth Example</h1>

  <h2>Last 10 visits</h2>
  {% for time in times %}
    <p>{{ time }}</p>
  {% endfor %}

</body>
</html>
```

The result

How to run the sample code.

Test your web service by running it locally in a virtual environment.

Windows:
1. Create an isolated Python environment in a directory external to your project and activate it.

```
cd ..
python -m venv env env\Scripts\activate
```

2. Navigate to your project directory and install dependencies:

```
python -m pip install -r requirements
```

3. Run the application:

4. In your web browser, enter the following address:

In your terminal window, press **Ctrl+C** to exit the web server.

Mac OS/Linux:
1. Create an isolated Python environment in a directory external to your project and activate it:

2. Follow the same Steps 2-4 documented previously for the Windows platform.

The interaction/routing between Front-end and backend

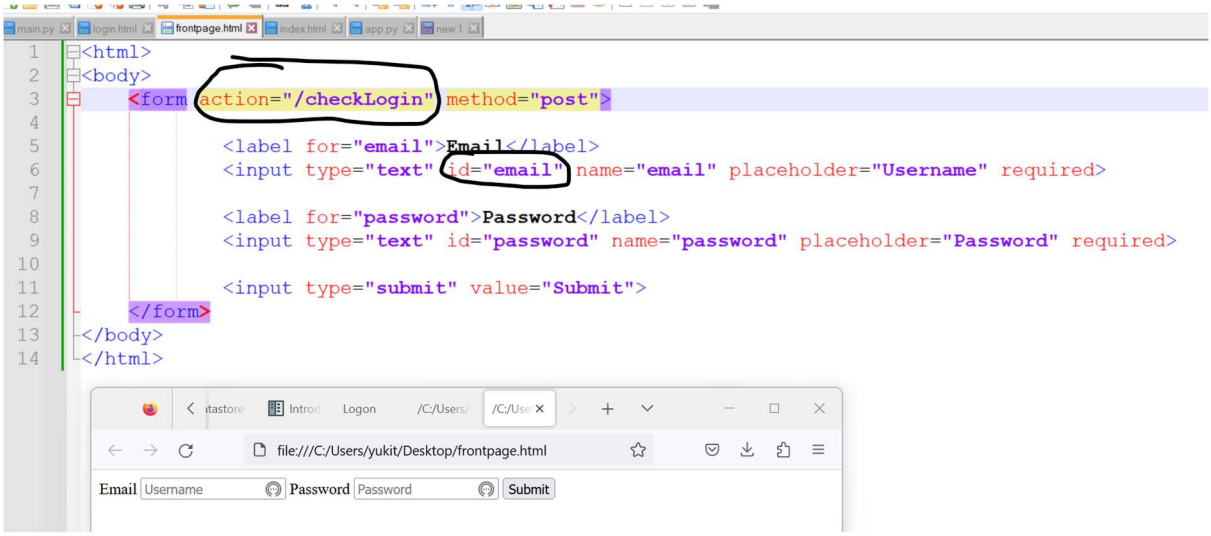
As per assessment 1, the login page is the first page you need to show.

The login page contains an “Email” text field, a “Password” field, and a “Login” button as well as a register link. When user clicks the “Login” button, it will validate if the user-entered credentials match with the information stored in the login table.

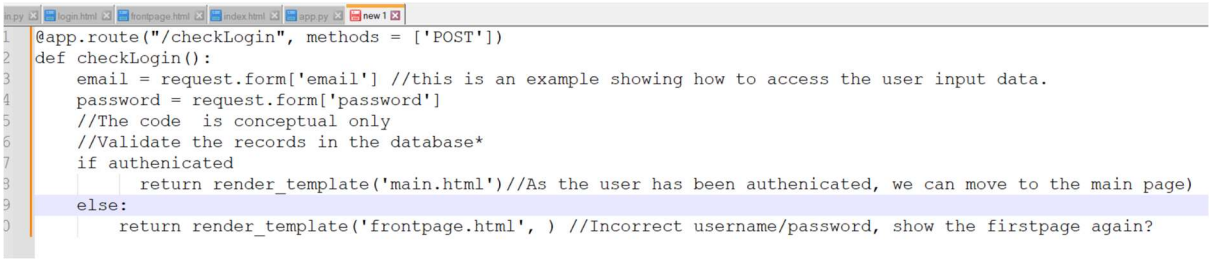
In a nutshell, we need an HTML template that captures the user input and validates the credentials at the backend function.

The following code is a **conceptual example only.**

Template\frontend.html (this could be the first page of your application)



When the user submits the form, the backend function "/checkLogin" will be triggered with the data captured from the form. In the Python code, you will need to build the checkLogin function.



```
@app.route("/checkLogin", methods = ['POST'])
def checkLogin():
    email = request.form['email'] //this is an example showing how to access the user input data.
    password = request.form['password']
    //The code  is conceptual only
    //Validate the records in the database*
    if authenicated
        return render_template('main.html')//As the user has been authenicated, we can move to the main page)
    else:
        return render_template('frontpage.html', ) //Incorrect username/password, show the firstpage again?
```