Scenario

You are a developer at a sports news website. You recently volunteered to gather sports statistics for the data analytics team. To familiarize yourself with Flask, you have decided to take your existing API call and load it to a Flask page.

Objective

This activity leverages an API key from football-data.org, with the goal of providing hands-on experience in developing a web application using the Flask framework.

Upon completing this activity, you will be able to integrate Flask code into a Python application to establish a connection with an API. Additionally, you will gain the skills to incorporate the necessary code into HTML files, enabling the retrieval of data from the Python application.

Instructions

Download and save the file Activity_Buildingyourfirstwebapps.zip on your machine and extract it to a folder where it is easily accessible. Open the folder, which contains the folder structure for the project. There are three files:

- app.py is where the Flask code will be added
- templates/index.html contains starter HTML code which will connect to your Python application (home page).
- templates/scores.html contains starter HTML code which will connect to your Python application (scores page).

For this project, you will need the **`requests`** library to make API calls, **`schedule`** to refresh data, and the **`Flask`** library to develop the Flask application. The **`socket`** library is installed by default. Open a terminal or command prompt and navigate to the folder containing the files for this project. Issue the following command:

```
pip install Flask requests schedule
```

You may get a list of messages stating "Requirement already satisfied", but if not, it will install the required libraries.

Leave the terminal or command prompt open.

Next, you'll modify the app.py file. Open app.py in an editor. If you completed Course 3 in the Microsoft Python Certificate (Automating and Scripting in Python), you can reuse your API key from that previous course.

If you did not complete the course, we'll be using the Football Data API (https://www.football-data.org/) to get current soccer match results and/or live scores. You will first need to get an API Key by heading over to the Football Data API, signing up for a free account, and grabbing your API key.

You will need to replace the text that says YOUR API KEY with your 32-character API key.

All required libraries are imported. There are three methods to examine:

`fetch_scores`: This method makes an API call to the football-scores.org API endpoint and displays the scores.

`load_index_page`: This method will be used to populate the index page.

`load_scores_page`: This method will be used to populate the scores page.

In this step, you will set up the route for the index.html page. Locate the load_index_page function. This function needs a decorator and needs to load the index.html file.

Decorate the route by adding the following immediately before the load_index_page function, replacing the # STEP 3: YOUR CODE HERE comment. You will need to use app.route to assign the URL of / to this function.

This decoration will associate a path with the function. In this case, when a website visitor connects to the root URL of the website, the load_index_page function will load. However, at the moment, there is no logic in the function. This will be a very basic site, just loading the contents of the index.html page from the components/index.html with no changes.

To add logic to generate the index page, you will need to add logic to display the index page. You will load the index page inside the function. You will need to use render_template to load the contents of index.html and return the result. The index.html file has already been provided in the components folder, and render_template has been imported already.

Within the load_index_page function, locate the # STEP 4: YOUR CODE HERE comment

When a user visits the root, they will see the results of this function.

Save the file. Switch to your terminal or command prompt window. Run the application by typing

```
python ./app.py
```

If there are no errors, this will set up a server at http://127.0.0.1:5000/ which you can visit in your browser. Your index page should display with a title "Coursera Flask Demo". Notice there is a link to the scores page, which does not work yet.

In this step, you will set up the route for the scores.html page. In app.py, locate the load_scores_page function. This function needs a decorator and needs to load the scores.html file. This will be similar to the previous step.

Decorate the route by adding the following immediately before the load_scores_page function, replacing the # STEP 5: YOUR CODE HERE comment. You will need to use app.route to assign the URL of **`/scores`** to this function.

You will also need to load the scores page. The difference is you will also provide the live_scores as an argument to the render_template. The code to load the scores into the live_scores variable has been provided.

Within the load_scores_page function, locate the # STEP 6: YOUR CODE HERE comment.

To add logic to generate the index page, you will need to add logic to display the scores page. You will load the scores page inside the function. You will need to use render_template to load the contents of scores.html and return the result. The scores.html file has already been provided in the components folder. To correctly display the page, you will also need to pass the live_scores data to render_template. It is expecting the values to be passed in argument named scores.

Within the load_scores_page function, locate the #YOUR CODE HERE comment. Replace it with your code.

Revisit your site in the browser. If you closed it, you can re-open it by running the following command from your terminal or command prompt:

```
python ./app.py
```

When you open the server in your browser (http://127.0.0.1:5000/), you should be automatically directed to the main page titled Coursera Flask Demo. The link labeled View scores! should work and direct you to a page labeled Live Soccer Scores, which should display scores on a web page.

Close the files. You have completed this exercise.

1. As a sports news website developer, why are you using Flask in this activity? Select the best answer. **1 point**

- To create a mobile app for live scores
- To analyze the sports data
- To collect data directly from sports teams
- To display live sports scores on a web page

2. Where will you add the Flask code to connect to the API and display scores? Select the best answer. **1 point**

- app.py
- scores.html
- requirements.txt
- index.html

3. What does the decorator @app.route('/') do in your Flask app? Select the best answer. **1 point**

- It hides the API key from users.
- It makes the app run faster.
- It tells Flask which function to run for the website's homepage.
- It automatically updates the scores every minute.

4. You want to display the live scores on a web page. Which built-in Flask function helps you do that? Select the best answer. **1 point**

- fetch_scores
- render_template
- app.route
- requests.get

5. In this project, what is the main role of the fetch_scores function? Select the best answer. **1 point**

- To get the live scores from the football-data.org API
- To display the scores in a user-friendly format
- To schedule updates for the live scores
- To create the HTML structure of the scores page

6. You've made substantial changes to your Flask app. How do you restart the app to see those changes on the website? Select the best answer. **1 point**

Close and reopen the app.py file.

Delete the app.py file and create a new one.

Stop the running server and run python app.py again.

Refresh the web page in your browser.