Web Apps with Bottle

Contents

Setting up

1. Create a new folder called **UsingBottle**
2. Open the folder in VS Code
3. Open a terminal and install Bottle using the command:

   pip install bottle

First App

1. Inside the UsingBottle folder create a new folder called **FirstApp**
2. Inside this folder create a new file called **first_app.py**
3. Add the following code:

```
from bottle import route, run

@route('/')
def index():
    return '<h1>My First Bottle App!</h1>'

run(host='localhost', port=8080, debug=True, reloader=True)
```

4. Go to the terminal in VS Code
5. In the terminal, change the current directory to the **FirstApp** directory by using the command:

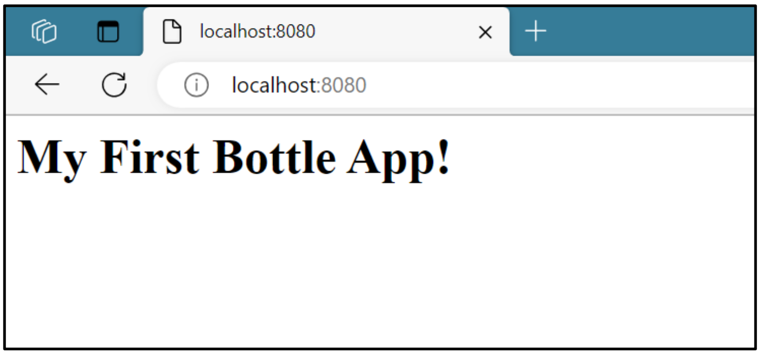
   cd FirstApp
6. Run your web app using the command:

   python first_app.py
7. Open your web browser and navigate to the page:

   localhost:8080

   NOTE: If this does not work, try 127.0.0.1:8080

   You should see something like this in your browser window:



My First Bottle App!

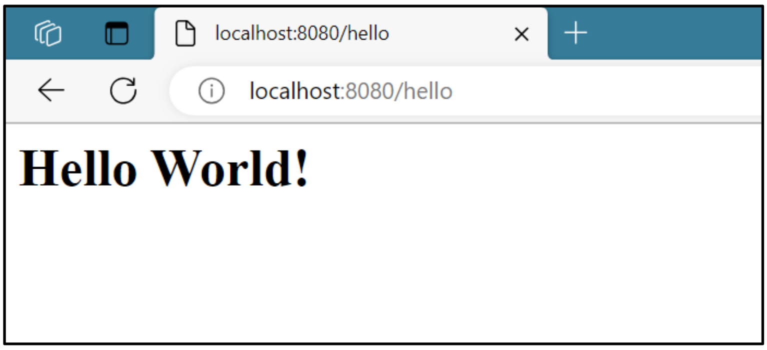
Another Page

One way of thinking about routes is that each route corresponds to a page in our website. The route gives the path to get to that page, so by adding a new route to our app we can add a new page.

1. Add the following code to your app just before the **run** function

```
@route('/hello')
def hello():
    return '<h1>Hello World!</h1>'
```

2. Save your changes and refresh your browser. The changes to your app should automatically be reloaded into your app.
3. Your web browser will look the same. Go to the address bar and navigate to:
   localhost:8080/hello
   You should now see something like this in your browser window:



Hello World!

Adding a link from another page

Now, let's add some links in the HTML to get from the home page to the hello page, and back again.

1. Modify the code in the two routes to look like the following:

```
@route('/')
def index():
    return '''<h1>My First Bottle App!</h1>
            <p><a href="/hello">Hello</a></p>'''

@route('/hello')
def hello():
    return '''<h1>Hello World!</h1>
            <p><a href="/">Back</a></p>'''
```

Using Templates

If we want to make the HTML for our pages more complicated, we need a more effective way of writing the HTML than just creating a string. This is where *templates* come in. A template gives us a way of creating the html for a page and then including that in our app.

1. Create a new folder and name it **views**
   NOTE: Make sure you follow the correct naming conventions!
2. Create a new file and name if **index.tpl**
3. Add the following HTML to the file:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First App</title>
    </head>
    <body>
        <h1>My First Bottle App!</h1>
        <p><a href="/hello">Hello</a></p>
    </body>
</html>
```

8. Modify the **import** line at the top of the python file to include **template**

```
from bottle import route, run, template
```

9. Modify the function **index** so that it looks like the following:

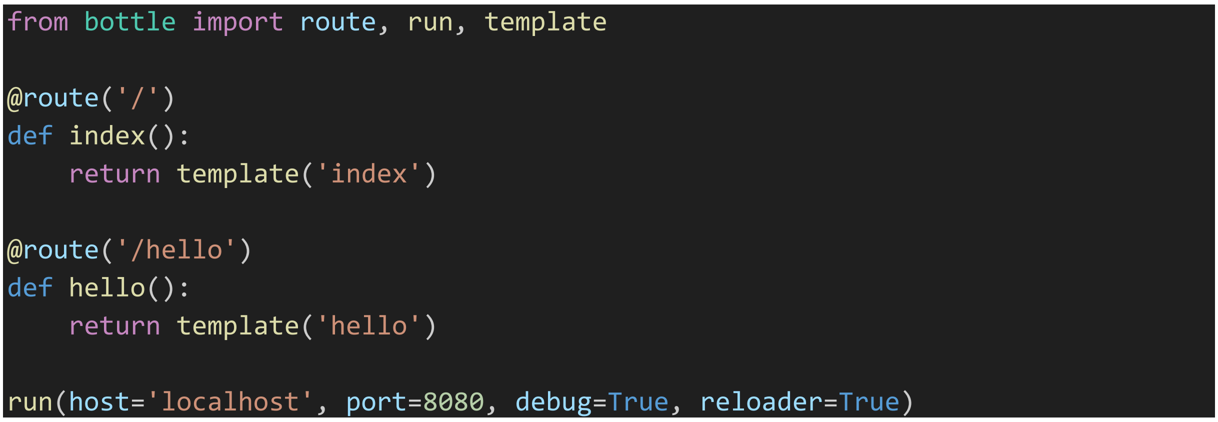
```
@route('/')
def index():
    return template('index')
```

10. Now go back to the browser and go to:
    localhost:8080
    It should look the same as previously, and the link should work the same way. If you open the *page source* though, you should see the HTML that you added to the template file.
11. Create another template file called **hello.tpl** in the **views** folder.
12. Add some html to this file such that the *Hello World!* page will appear as per previously.
13. Update the *hello* route so that the new template is called.

You should have the following html in your **hello.tpl** page:

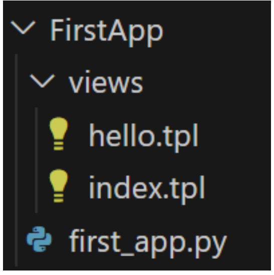
```
<!DOCTYPE html>
<html>
    <head>
        <title>My First App</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <p><a href="/">Back</a></p>
    </body>
</html>
```

And the code in **first_app.py** should look something like this:



```
from bottle import route, run, template

@route('/')
def index():
    return template('index')

@route('/hello')
def hello():
    return template('hello')

run(host='localhost', port=8080, debug=True, reloader=True)
```

You should also have a folder structure something like this:



```
FirstApp
  views
    hello.tpl
    index.tpl
  first_app.py
```

NOTE: The icons you have may be different, but the structure should be the same.

Using Variables in your HTML

Wouldn't it be way more interesting if we could get our app to say hello to different people instead of just the world?

Well, it turns out that we can add variables and Python code to our templates to make them way more flexible.

To start with, modify *<h1>* content in the **hello.tpl** file so that it reads as:

```
<h1>Hello {{ name }}</h1>
```

Note the curly brackets in this line (the {{ and }}). These indicate that the content within the brackets is a Python command, in this case a parameter.

Save the file and refresh the web app in your browser, making sure you navigate to the hello page.

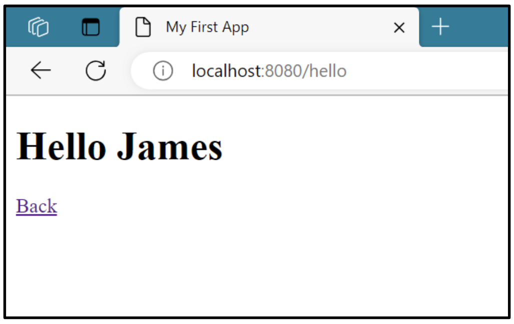
You should get a message along the lines of **Error: 500 Internal Server Error**. If you look at the exception information, you will notice an error message such as *NameError("name 'name' is not defined")*. This indicates that we have tried to use a parameter in our template but have not defined the parameter when we called the template.

Let's fix the app to provide a parameter to the html template.

Modify the line of code that calls the *hello* template to the following:

```
    return template('hello', name='James')
```

Once again, the web ap should reload once you save the file. Go to your web browser and refresh the page. You should get something like the following:



Hello James

Back

Using a List in your HTML

We can use Python code inside our template to dynamically print the values of a list in our page, regardless of how long the list is.

To start with, let's create another page:

1. In your **views** folder create a new file and save it as **customers.tpl**
2. Add the following HTML to the page:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First App</title>
    </head>
    <body>
        <h1>Customer List</h1>
        <p><a href="/">Home</a></p>
        <p>
            <ul>
                <li>John Smith</li>
            </ul>
        </p>
    </body>
</html>
```

3. Add a new route to your app called **customers** and call this template by adding the following code to the **first_app.py** file.

```
@route('/customers')
def customers():
    return template('customers')
```

4. Save the app file and go to the page **localhost:8080/customers** and see if your page loads.

Now that we have hooked up a new web page, we can add a list of names to our page.

5. Start by adding a list of names to the **customers** function in the **first_app.py** file. Your function should look something like this:

```
@route('/customers')
def customers():
    customer_list = ['James Smith', 'John Chalmers', 'Jane Peterson']
    return template('customers', customers=customer_list)
```

Now we can add some Python code to our template to loop through the list and print out each customer name. To indicate that we want to start some Python code, we use the **%** symbol like in the following code:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First App</title>
    </head>
    <body>
        <h1>Customer List</h1>
        <p><a href="/">Home</a></p>
        <p>
            <ul>
                % for name in customers:
                <li>{{ name }}</li>
                % end
            </ul>
        </p>
    </body>
</html>
```

In this case, we are going to loop through the customer list and print each customer name. Note the **% end** to indicate that we have finished the Python loop.

Using a List of Dictionaries

We can improve the readability of our code and the structure of our HTML by using a dictionary to store the information about each customer, and then use a table to present the results in our *customers* page.

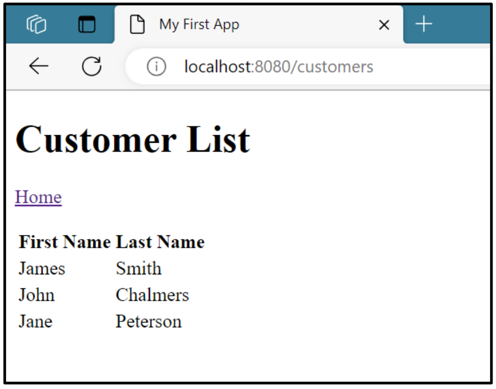
Modify the list of customers in the **customers** function with the following code:

```
    customer_list = [
        {'first_name': 'James', 'last_name': 'Smith'},
        {'first_name': 'John', 'last_name': 'Chalmers'},
        {'first_name': 'Jane', 'last_name': 'Peterson'}
    ]
```

We can now use this to create a table in our HTML by replacing the HTML for the list with the following:

```
            <table>
                <tr>
                    <th>First Name</th>
                    <th>Last Name</th>
                </tr>
                % for customer in customers:
                <tr>
                    <td>{{ customer['first_name'] }}</td>
                    <td>{{ customer['last_name'] }}</td>
                </tr>
                % end
            </table>
```

Your **customers** page should now look something like this:



Customer List

Home

First Name	Last Name
James	Smith
John	Chalmers
Jane	Peterson

Adding Styling and Images

To add content that is not going to changed we need to add **static** files to our web app.

1. Create a new folder in your **FirstApp** folder called **static**
2. Inside the **static** folder create a new file called **style.css**
3. Add some simple CSS to the file such as:

```
body {
    background: tan;
}
```

4. Add a link to the **index.tpl** file by adding the following line to the **<head>** section of the HTML:

```
        <link type="text/css" href="/static/style.css" rel="stylesheet">
```

Try refreshing the home page by going to the page **localhost:8080** in your web browser. Note that the CSS has not taken effect on our page. To fix this, we need to create static resources for our page.

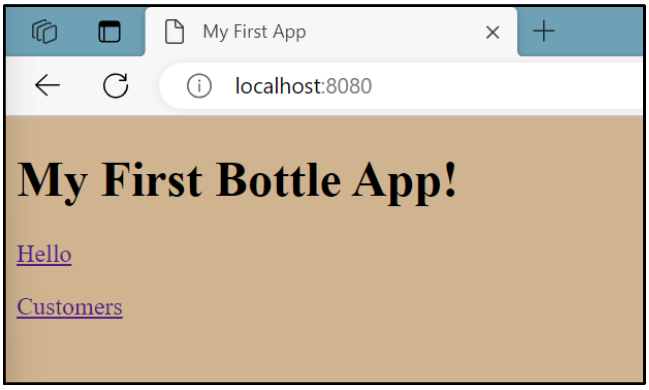
5. Import the **static_file** function at the start of our **first_app.py** file:

```
from bottle import route, run, template, static_file
```

6. Add a route to the static files by adding the following code:

```
@route('/static/<filename>')
def static(filename):
    return static_file(filename, root='./static')
```

Now go back to the browser and refresh your page. The CSS should now be taking effect and your page should look something like this:



My First Bottle App!

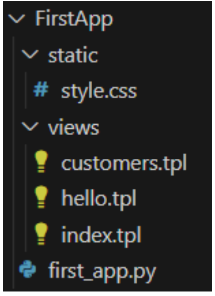
Hello

Customers

Now add a link to the CSS file to the other template pages and check that the CSS is working for all of them.

If we want to add images to our page, we need to add them to the static folder as well and then include them in the HTML.

Buy now, your folder structure should look something like this:



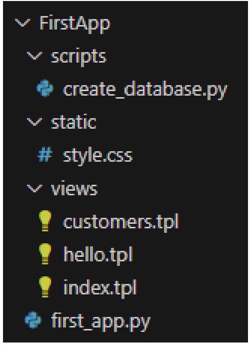
```
FirstApp
  static
    style.css
  views
    customers.tpl
    hello.tpl
    index.tpl
  first_app.py
```

Creating a new Database

The first thing we are going to do is add a script to create a new database. To keep our project neat, we are going to add a new folder to our project for the python scripts.

1. Add a new folder called **scripts**
2. Add a new file to the **scripts** folder called **create_database.py**. This is the file where we are going to write the code to create a new, empty database.

Your folder structure should now look something like this:



```
FirstApp
  scripts
    create_database.py
  static
    style.css
  views
    customers.tpl
    hello.tpl
    index.tpl
  first_app.py
```

3. Now open the **create_database.py** file and add the following code:

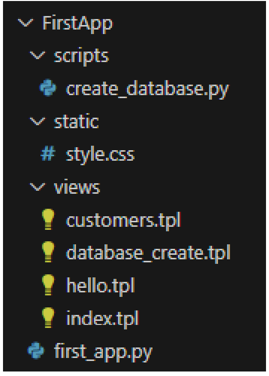
```
import sqlite3

def create_empty_database(database_name):
    conn = sqlite3.connect(database_name)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()

    print('Creating empty database')
    # Update database settings
    print('Update PRAGMA to support foreign keys')
    cursor.execute('PRAGMA foreign_keys = ON')

    # Drop tables if they exist to allow new tables to be created
    print('Drop tables if they exist')
    cursor.execute('DROP TABLE IF EXISTS Player;')
    cursor.execute('DROP TABLE IF EXISTS Team;')

    # Create tables
    print('Create Team table')
    cursor.execute('''CREATE TABLE Team
                            (team_id INTEGER PRIMARY KEY,
                            name varchar(20) NOT NULL,
                            colour varchar(10));''')

    print('Create Player table')
    cursor.execute('''CREATE TABLE Player
                            (player_id INTEGER PRIMARY KEY,
                            first_name varchar(20) NOT NULL,
                            last_name varchar(20) NOT NULL,
                            team_id INTEGER,
                            FOREIGN KEY (team_id) REFERENCES 
Team(team_id));''')

    conn.commit()
    cursor.close()
    conn.close()
    print('Empty database created')
```

We also need a view that we can call when we have created the database.

4. Inside the **views** folder add a new file called **database_create.tpl**
5. Update this file with the following HTML:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First App</title>
        <link type="text/css" href="/static/style.css" rel="stylesheet">
    </head>
    <body>
        <h1>My First Bottle App!</h1>
        <p>Creating a new database</p>
        <p><a href="/">Return to the home page</a></p>
    </body>
</html>
```

Your folder structure should now look something like this:



```
FirstApp
  scripts
    create_database.py
  static
    style.css
  views
    customers.tpl
    database_create.tpl
    hello.tpl
    index.tpl
  first_app.py
```

Now we need to add a link to our **index.tpl** page so that we can add the correct route to our web app.

6. Add the following HTML to your **index.tpl** file:

```
        <p><a href="/create_database">Create a new database</a></p>
```

Finally, we need to add a route to our web app.

7. Open the file **first_app.py**
8. Import the create_database Python script by adding the following line at the start of your code:

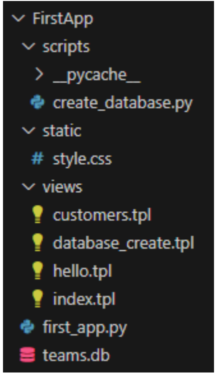
```python
import scripts.create_database as create_db
```

9. Add a route to your code (below the route to customers, but before the route for static files:

```python
@route('/create_database')
def create_database():
    create_db.create_empty_database('teams.db')
    return template('database_create')
```

10. Finally, we can run our app to see if the database script works! Run the web app and you should get a new database file created that will have two empty tables.

Your folder structure should now look something like this:



```
⌄ FirstApp
  ⌄ scripts
    › __pycache__
      create_database.py
  ⌄ static
    # style.css
  ⌄ views
      customers.tpl
      database_create.tpl
      hello.tpl
      index.tpl
    first_app.py
    teams.db
```

Inserting some Data

Now that we have an empty database, lets add some data to the tables.

1. Let's start by creating a new python script in our **scripts** folder called **insert_data.py**
2. Add the following python to the **insert_data.py** file:

```
import sqlite3

def insert_sample_data(database_name):
    conn = sqlite3.connect(database_name)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()

    print('Inserting data into database')

    print('Inserting data into Team table')
    cursor.execute('INSERT INTO Team (team_id, name, colour) VALUES (1,
"Dolphins", "blue");')
    cursor.execute('INSERT INTO Team (team_id, name, colour) VALUES (2,
"Sharks", "green");')
    cursor.execute('INSERT INTO Team (team_id, name) VALUES (3, "Whales");')

    print('Inserting data into Team table')
    cursor.execute('INSERT INTO Player (player_id, first_name, last_name,
team_id) VALUES (11, "James", "Smith", 1);')
    cursor.execute('INSERT INTO Player (player_id, first_name, last_name,
team_id) VALUES (12, "Sandra", "Peters", 2);')
    cursor.execute('INSERT INTO Player (player_id, first_name, last_name,
team_id) VALUES (13, "Richard", "Jones", 1);')

    conn.commit()
    cursor.close()
    conn.close()
    print('Data inserted into database')
```

3. Add a new template in the **views** folder called **database_insert.tpl**
4. Add the following HTML to the new template:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First App</title>
        <link type="text/css" href="/static/style.css" rel="stylesheet">
    </head>
    <body>
        <h1>My First Bottle App!</h1>
        <p>Inserting sample data in the database</p>
        <p><a href="/">Return to the home page</a></p>
    </body>
</html>
```

5. Add a new link to the **index.tpl** file by adding the following line:

```
        <p><a href="/insert_data">Insert sample data in database</a></p>
```

Finally, we need to add a route to our web app.

6. Open the file **first_app.py**
7. Import the **insert_data** Python script by adding the following line at the start of your code:

```
import scripts.insert_data as db_insert
```

8. Add a route to your code (below the route to customers, but before the route for static files:

```
@route('/insert_data')
def insert_data():
    db_insert.insert_sample_data('teams.db')
    return template('database_insert')
```

9. Finally, we can run our app to see if the database script works! Run the web app and you should be able to add some sample data to your database file.

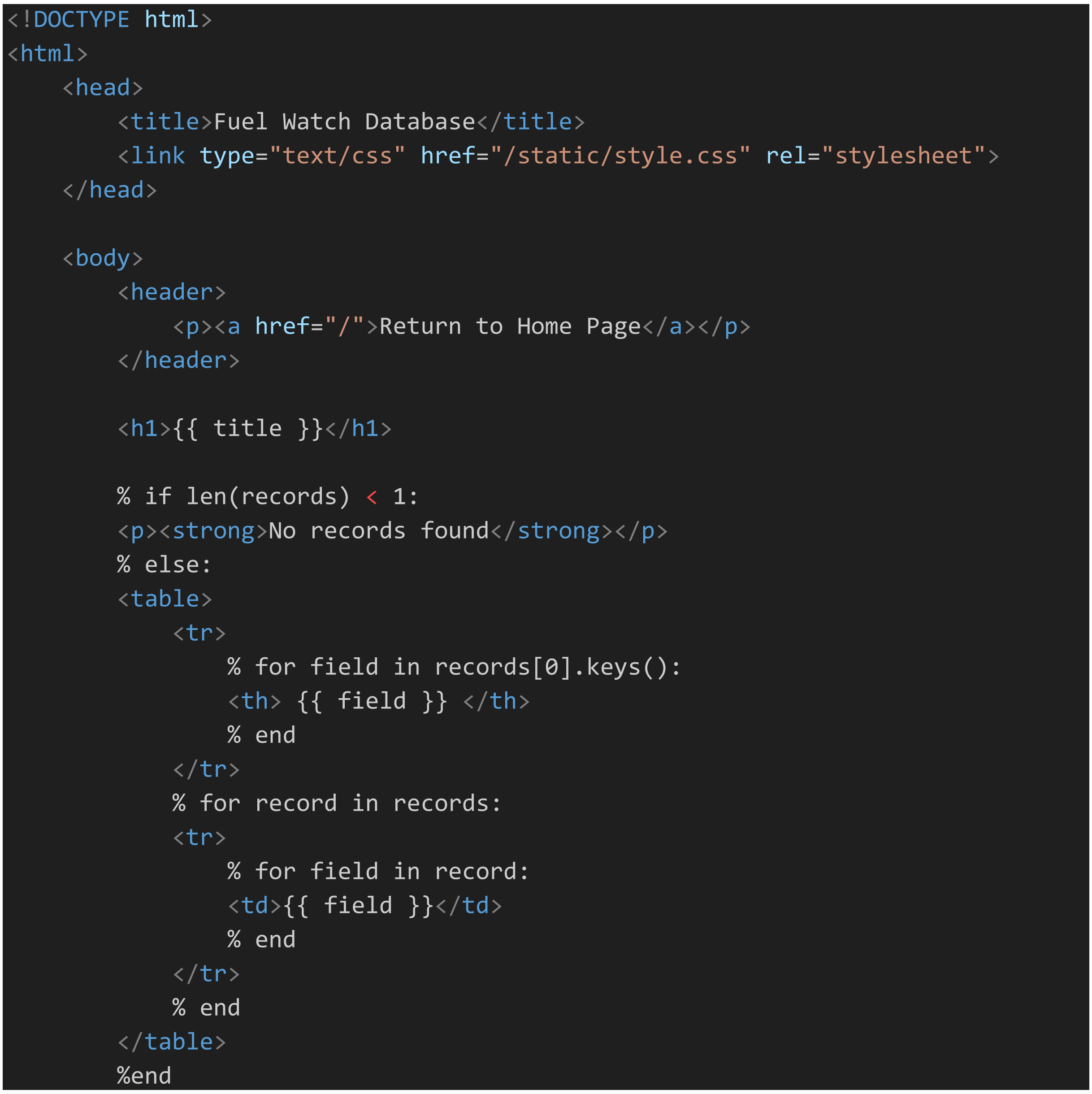
Selecting Data

Now that we have created a database and added some sample data, it would be nice to be able to run some queries on that data!

The first thing we are going to need to do is create a new template that will allow us to display the results of a query. To do this, we are going to do something similar to the **customers** template that we created earlier, where we will build an HTML table based on the contents of a dictionary passed into the template.

The first thing to do is to create a new template to present the results of our query. This template will assume that each record being returned is a dictionary, with the field names as the key.

1. Create a new template file in the **views** folder called **results.tpl**
2. Place the following code in the results template. This code will check to see if any records have been returned, and if they have will build an HTML table to present the results to the user.



```
<!DOCTYPE html>
<html>
    <head>
        <title>Fuel Watch Database</title>
        <link type="text/css" href="/static/style.css" rel="stylesheet">
    </head>

    <body>
        <header>
            <p><a href="/">Return to Home Page</a></p>
        </header>

        <h1>{{ title }}</h1>

        % if len(records) < 1:
        <p><strong>No records found</strong></p>
        % else:
        <table>
            <tr>
                % for field in records[0].keys():
                <th> {{ field }} </th>
                % end
            </tr>
            % for record in records:
            <tr>
                % for field in record:
                <td>{{ field }}</td>
                % end
            </tr>
            % end
        </table>
        %end
        <footer>
            <p><a href="/">Return to Home Page</a></p>
        </footer>
    </body>
</html>
```

Note that this template expects two parameters:

- *title* – a heading to describe the query to the user
- *records* – the results of a query.

If the query does not return any results, then the records list will be empty, so the template will display that no records have been found. Otherwise the template will build and HTML table based on the results.

Now we have built the template, we need to add a route to our web app that will run a query.

3. Go to the file **first_app.py**
4. The first thing we will need to do is import the sqlite libraries so we can use them. At the top of the file, add the line:

```
Import sqlite3
```

5. Now we need to add the information for our route. In this case, we are going to get all the information about the teams in the database. Add the following code before the static route:

```
@route('/select_all_teams')
def select_all_teams():
    conn = sqlite3.connect('teams.db')
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()

    query = 'SELECT * FROM Team'
    cursor.execute(query)
    result = cursor.fetchall()

    cursor.close()
    conn.close()

    return template('results', records=result, title='All Teams')
```

Note that this code will:

- Open a connection to the database using a row factory
- Run a select query
- Close the connection to the database
- Call the template, with the results of the query and a title to display to the user passed in as parameters

Finally, we need to add a link to the index page to call the new route.

6. Open the file **index.tpl**

7. Add the following line to your page:

```
        <p><a href="/select_all_teams">Select all teams in the
database</a></p>
```

Now you should be able to run your web app and display all the teams in the database.

To add more queries to your web app, simply add a new route for each query, updating the sql each time. For example, if you want to add a query to list all the players and their team name, you could add the following route to the file **first_app.py**:

(NOTE: Don't forget to add a link to your index page to call the route.)

```
@route('/select_players_and_team')
def select_all_teams():
    conn = sqlite3.connect('teams.db')
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()

    query = '''
            SELECT Player.first_name, Player.last_name, Team.name
            FROM Player, Team
            WHERE Player.team_id = Team.team_id
            '''
    cursor.execute(query)
    result = cursor.fetchall()

    cursor.close()
    conn.close()

    return template('results', records=result, title='All Teams')
```