Project 1: Rainy Day

Released	Tuesday, July 3
Due	Thursday, July 12 at 6pm EST

Objectives

- Become more comfortable with Python.
- Gain experience with Flask.
- Learn to use SQL to interact with databases.
- Learn to make API calls and parse the results of those calls.
- Enable others to leverage your data via your own API.

Getting Help

If you need help while working on the project, feel free to take advantage of any or all of the following resources:

- Attend one or more of the course's **sections**.
- Ask questions on **Slack**.
- Attend **office hours**.

Overview

In this project, you'll build a web page that will allow users to query weather information. Users will be able to register for your website and then log in using their username and password. Once they log in, they will be able to search for cities and towns, have the option to "check in" to the location they've searched for, and see current and/or recent historical weather data. You'll also use a third-party API by Dark Sky, a popular service for up-to-the-minute weather data, to get current weather information that you will display to users. Finally, users will be able to query for basic information about the cities and towns (such as

latitude, longitude, and number of check-ins by users) programmatically via your website's API.

Getting Ready

First, log into CS50 IDE and ensure everything is up to date:

```
$ update50
```

Then, after that finishes executing, download the distribution code for this assignment.

```
$ cd
$ wget http://cdn.cs50.net/web/2018/summer/projects/1/project1.zip
$ unzip project1.zip
$ rm -rf project1.zip
$ cd project1
```

You should see that the directory contains four files initially. You'll add at least one, and possibly more, as you proceed through this project.

```
README.md   application.py   requirements.txt   zips.csv
```

Getting Started

PostgreSQL

For this project, you'll need to set up a PostgreSQL database to use with our application. It's possible to set up PostgreSQL locally on your own computer, but for this project, we'll use a database hosted by Heroku, an online web hosting service.

1. Navigate to https://www.heroku.com/, and create an account if you don't already have one.
2. On Heroku's Dashboard, click **New** and choose **Create new app**.
3. Give your app a name, perhaps using your GitHub username (e.g., `project1-USERNAME`), and click **Create app**.
4. On your app's *Overview* page, click the **Configure Add-ons**button.

5. In the “Add-ons” section of the page, type in and select **Heroku Postgres**.
6. Choose the **Hobby Dev - Free** plan, which will give you access to a free PostgreSQL database that will support up to 10,000 rows of data. Click **Provision**.
7. Now, click the **Heroku Postgres :: Database** link.
8. You should now be on your database’s overview page. Click on **Settings**, and then “View Credentials.” This is the information you’ll need to log into your database. You can access the database via Adminer, filling in the server (the *Host* in the credentials list), your username (the *User*), your password, and the name of the database, all of which you can find on the Heroku credentials page.

Alternatively, if you are already comfortable with using SQL at the command line, you can run

```
$ psql URI
```

on the command line, where the `URI` is the link provided in the Heroku credentials list.

Flask

To try running your project now as a Flask application:

1. In a terminal window, navigate into your `project1` directory.
2. Run `pip3 install -r requirements.txt` in your terminal window to make sure that all of the necessary Python packages (Flask and SQLAlchemy, for instance) are installed.
3. Set the environment variable `FLASK_APP` to be `application.py`. To do this, type:

```
$ export FLASK_APP=application.py
```

1. You may optionally want to set the environment variable `FLASK_DEBUG` to `1`, which will activate Flask’s debugger

and will automatically reload your web application whenever you save a change to a file.
2. Set the environment variable `DATABASE_URL` to be the URI of your database, which you should be able to see from the credentials page on Heroku.
3. Run `flask run` to start up your Flask application.
4. If you navigate to the URL provided by `flask` in the terminal (look for a line in your output of the form `Running on:`), you should see the text `"Project 1: TODO"`!

Dark Sky API

Dark Sky is a popular mobile app for up-to-the-minute weather information, and we'll be using their API in this project to get access to weather data for the cities and towns we're searching for.

1. Go to https://darksky.net/dev/register and fill out the short form to apply for an API key. When given the option, select the free version, which will allow you 1,000 API calls per day (which should be more than enough for you to test!)
2. After filling out the form, you should receive an email from Dark Sky; click the link in that email to be redirected to complete registration and be redirected to https://darksky.net/dev/account.
3. You should then see your API key, which will be a series of about 25 alphanumeric characters. You'll always be able to access this key by logging in to your Dark Sky developer account, but you may also wish to write it down somewhere.
4. You can now use that API key to make requests to the Dark Sky API, documented here. In particular, Python code like the below

```
import requests
weather = requests.get("https://api.darksky.net/forecast/KEY/42.37,-71.11").json()
print(weather["currently"])
```

where `KEY` is your API key, will give you the *current* weather data (only a small subset of what your API call actually returns) for latitude 42.37, which is 42.37° N; and longitude -71.11, which is 71.11° W (this being more commonly known as Cambridge, MA). In particular, you might see something like this:

```
{'time': 1530641402, 'summary': 'Clear', 'icon': 'clear-day', 'n
earestStormDistance': 45, 'nearestStormBearing': 336, 'precipInt
ensity': 0, 'precipProbability': 0, 'temperature': 93.56, 'appar
entTemperature': 101.11, 'dewPoint': 71.31, 'humidity': 0.49, 'p
ressure': 1020.16, 'windSpeed': 4.64, 'windGust': 7.18, 'windBea
ring': 229, 'cloudCover': 0.22, 'uvIndex': 8, 'visibility': 10,
'ozone': 304.82}
```

To make it look a little bit nicer, we could include the `json` module and use a function therein called `dumps`, which allows us to "pretty-print" the JSON to make it more visually appealing if being viewed by human eyes.

```
import requests, json
weather = requests.get("https://api.darksky.net/forecast/KEY/42.
37,-71.11").json()
print(json.dumps(weather["currently"], indent = 2))
{
  "time": 1530641402,
  "summary": "Clear",
  "icon": "clear-day",
  "nearestStormDistance": 45,
  "nearestStormBearing": 336,
  "precipIntensity": 0,
  "precipProbability": 0,
  "temperature": 93.56,
  "apparentTemperature": 101.11,
  "dewPoint": 71.31,
  "humidity": 0.49,
  "pressure": 1020.16,
  "windSpeed": 4.64,
  "windGust": 7.18,
  "windBearing": 229,
  "cloudCover": 0.22,
  "uvIndex": 8,
  "visibility": 10,
  "ozone": 304.82
}
```

We'll defer to you to review the documentation to see what these key-value pairs mean, exactly, with respect to Dark Sky's output.

Requirements

Alright, it's time to actually build your web application! Here are the requirements:

- **Registration**: Users should be able to register for your website, providing (at minimum) a username and password.
- **Login**: Users, once registered, should be able to log in to your website with their username and password.
- **Logout**: Logged in users should be able to log out of the site.
- **Import**: Provided for you in this project is a file called `zips.csv`, which is a file in CSV format of all ZIP codes in the United States that have a population of 15,000 or more (there are 7375 such ZIP codes; apologies if your own hometown or ZIP code is not included as a result!) as well as some other information such as location names and short codes, latitude and longitude, and population. In a Python file called `import.py` separate from your web application, write a program that will take the information in this CSV and import it into your PostgreSQL database. You will first need to decide what table(s) to create, what columns those tables should have, and how they should relate to one another. Run this program by running `python import.py` to import the books into your database, and submit this program with the rest of your project code.
- **Search**: Once a user has logged in, they should be taken to a page where they can search for a location. Users should be able to type a ZIP code or the name of a city or town. After performing the search, your website should display a list of possible matching results, or some sort of message if there were no matches at all. If

the user typed in only partial information, your search page should find matches for those as well!

- **Location Page**: When users click on a location from the results of the search page, they should be taken to a page for that location, with details about the location coming from your database: the name of the location, its ZIP code, its latitude and longitude, its population, and the number of check-ins and the written comments that users have left for the location on your website.
- **Check-In Submission**: On the location page, users should be able to submit a "check-in", consisting of a button that allows them to log a visit, as well as a text component where the user can provide comments about the location. Users should not be able to submit more than one check-in for the same location or edit a comment they have previously left. Users should only be able to submit a check-in if they are logged in.
- **Dark Sky Weather Data**: On your location page, you should also display information about the current weather, displaying minimally the time of the weather report, the textual weather summary (e.g. "Clear"), temperature, dew point, and humidity (as a percentage). You can display more information if you wish.
- **API Access**: If users make a GET request to your website's `/api/<zip>` route, where `<zip>` is a ZIP code, your website should return a JSON response containing (at a minimum) the name of the location, its state, latitude, longitude, ZIP code, population, and the number of user check-ins to that location. The resulting JSON should follow the format; the order of the keys is not important, so long as they are all present:

```
{
    "place_name": "Cambridge",
    "state": "MA",
    "latitude": 42.37,
    "longitude": -71.11,
    "zip": "02138",
    "population": 36314,
```

```
    "check_ins": 1
}
```

- If the requested ZIP code isn't in your database, your website should return a 404 error.
- In `README.md`, include a short write-up describing your project, what's contained in each file, and (optionally) any other additional information the staff should know about your project.
- If you've added any Python packages that need to be installed in order to run your web application, be sure to add them to `requirements.txt`!

Beyond these requirements, the design, look, and feel of the website are up to you! You're also welcome to add additional features to your website, so long as you meet the requirements laid out in the above specification!

Hints

- Think carefully about how you might want to store ZIP codes in your database table!
- At minimum, you'll probably want at least one table to keep track of users, one table to keep track of locations, and one table to keep track of check-ins to those locations. But you're not limited to just these tables, if you think others would be helpful.
- In terms of how to "log a user in," recall that you can store information inside of the `session`, which can store different values for different users. In particular, if each user has an `id`, then you could store that `id` in the session (e.g., in `session["user_id"]`) to keep track of which user is currently logged in.

FAQs

- Reload periodically to see if any FAQs arise!

How to Submit

Step 1 of 2

1. Log into CS50 IDE and open a terminal window, if not open already.
2. Submit your project by executing:

```
$ cd ~/workspace/project1
$ submit50 web50/2018/summer/project1
```

Step 2 of 2

Fill out this form!

Congratulations! You've completed Project 1.