Homework 6: Saving Data with a Flask Backend

Due Friday 3/8 @4pm on Courseworks.

What to submit:

- 2 uploads titled:
  - Problem1.pdf
  - Hw6_app.zip

- Submit them separately to Courseworks.

Problem 1. Running a Flask Project

Install Flask on your computer, download and unzip our example flask project.
Run the code on your machine. At the terminal within the "/people" folder execute the python code as "python server.py"
Install other packages you might need (render_template, Response, request, jsonify)

In a web browser, visit http://127.0.0.1:5000/

You should see a page that simply says "Hello World". If you have done this, the Flask project is currently running on your machine. Answer the following questions about how the code works in a pdf called Problem1.pdf.

1. How many webpages does this code serve?
    a. Show us screen shots of all of them.
2. In hello.html, what syntax makes the <name> show up in the HTML?
3. In people.html, which line(s) imports people.js?
    a. This line has two variables in it. What are they? What do they represent?
4. In people.html, which line(s) get the data sent from the server, put it into JavaScript and parses the string into JSON?
5. In people.js, what function contains the ajax call?
6. What route does that function send data to?
7. What data does it send?
8. If everything goes perfectly on the server side, what line(s) of code will execute on the front end (copy them here)?
9. If something goes wrong on the server side, what line(s) of code will execute on the front end?
10. On the server, what does the add_name function do (verbalize it in a few sentences, don't copy the code)?
11. If all goes well, what does the add_name function return? Show us an example.
12. If you refresh the web page, will the changes you made to the data still be there? Why or why not?
13. If you restart the server will the changes you made to the data still be there? Why or why not?

Problem 2. Saving paper sales on the server

Start a new Flask project to be the backend behind the log_sales application from HW5. As long as the server is running, the changes you make to the sales will persist even when you close and open the webpage (or refresh the page).

You may adapt the example code if you wish.

1. On the server, you must have 3 variables that store all the data. (see the following page)
2. You must have a route for "/infinity" that renders a template called cu-paper-infinity.html (similar to 1_log_sales.html)
    a. This file must contain all your static HTML and your CSS
    b. It may contain JavaScript variables, but no functions.
3. Your infinity.html file must load/include an infinity.js file
4. In cu-paper-infinity.js you must have three functions:
    a. **var display_sales_list = function(sales){…}**
        i. This takes in an array of sales and displays them on the UI. This contains no ajax calls.
    b. **var save_sale = function(new_sale){ …}**
        i. this takes one sale in the following format (note: there is no id field):

```
{
        "salesperson": salesperson,
        "client": client,
        "reams": reams
}
```

        ii. It must make an ajax call to the server.
        iii. If the server was successful, the UI updates accordingly:
            1. The new sale appears at the top of the list of sales
            2. The if client (if any) is in the autocomplete.
    c. **var delete_sale = function(id){ …}**
        i. this takes in one sales id (for example, 1 or 3)
        ii. it must make an ajax call to the server.
        iii. The sales array is returned to the JavaScript ajax call.
        iv. If the server was successful, the UI updates accordingly.
    d. You may have other JavaScript functions to add click handles when the document is ready, and process inputs, etc. All the functionality from HW5 is still required.
5. On the server, you must have two routes (and function) that don't render pages, but do process data. Call them:
    a. "save_sale"
        i. It takes in a sale (like in b.i),
        ii. It adds a unique id
        iii. It updates the data

iv. It returns two things: all the sales and all the clients

b. “delete_sale”
    i. It takes in an id of a sale
    ii. It updates the data
    iii. It returns all the sales.

Variables on the server

```
current_id = 4

sales = [
    {
        "id": 1,
        "salesperson": "James D. Halpert",
        "client": "Shake Shack",
        "reams": 1000
    },
    {
        "id": 2,
        "salesperson": "Stanley Hudson",
        "client": "Toast",
        "reams": 4000
    },
    {
        "id": 3,
        "salesperson": "Michael G. Scott",
        "client": "Computer Science Department",
        "reams": 10000
    },
]


clients = [
    "Shake Shack",
    "Toast",
    "Computer Science Department",
    "Teacher's College",
    "Starbucks",
    "Subsconsious",
    "Flat Top",
    "Joe's Coffee",
    "Max Caffe",
    "Nussbaum & Wu",
    "Taco Bell",
];
```

Problem 3. Saving the PPC on the server

In the same Flask application from Problem 2, extend it to have another page for the drag and drop party planning committee UI. However, how every time a name is dropped to a new list, it will save. If you close and open the webpage the changes you made to the PPC and non-PPC will persist.

Again, you may adapt the example code if you wish.

1. On the server, you must have 2 variables that store all the data. (see the following page)
2. You must have a route for "/ppc" that renders a template called ppc.html (similar to 2_ppc.html from HW5)
    a. This file must contain all your static HTML and your CSS
    b. It may contain JavaScript variables, but no functions.
3. Your ppc.html file must load/include an ppc.js file
4. In ppc.js you must have three functions:
    a. **var display_lists = function(non_ppc, ppc){…}**
        i. This takes in two arrays and displays them on the UI. This contains no ajax calls.
    b. **var move_to_ppc = function(name){ …}**
        i. this takes one name in the following format:

```
{
        "name": name
}
```

        ii. It must make an ajax call to the server.
        iii. On the server, the non_ppc_people array and ppc_people arrays must update accordingly.
        iv. The both updated arrays are returned to the JavaScript ajax call
        v. If the server was successful, the UI updates accordingly.
    c. **var move_to_non_pcc = function(name){ …}**
        i. this takes in one name (for example "Angela") in the same format as move_to_pcc
        ii. it must make an ajax call to the server.
        iii. On the server, the non_ppc_people array and ppc_people arrays must update accordingly.
        iv. The both updated arrays are returned to the JavaScript ajax call
        v. If the server was successful, the UI updates accordingly.
    d. You may have other JavaScript functions to add click handles when the document is ready, and process inputs, etc. All the functionality from HW5 is still required.
5. On the server, create the route(s) that process the ajax calls.

Variables on the server

```
non_ppc_people = [
"Phyllis",
"Dwight",
"Oscar",
"Creed",
"Pam",
"Jim",
"Stanley",
"Michael",
"Kevin",
"Kelly"
]


ppc_people = [
"Angela"
]
```