Homework 5: Saving Data on a Server with Flask

Warm up: due Friday 2/16 @ 11:59pm on Courseworks
Main: due Tuesday 2/20 @ 11:59pm on Courseworks.

Warm-up:

What to submit:

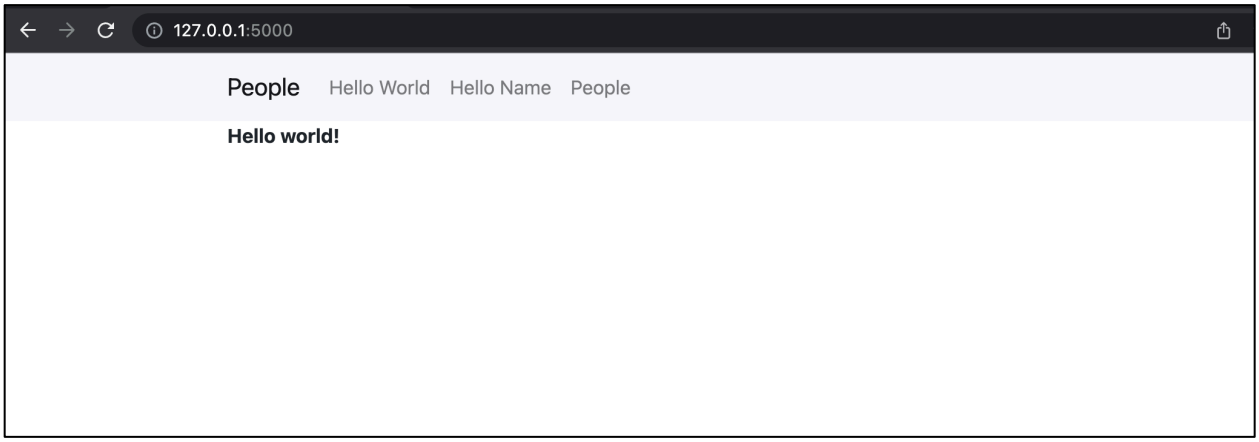
1. A pdf containing your screenshots.

Problem 1. Running a Flask Project

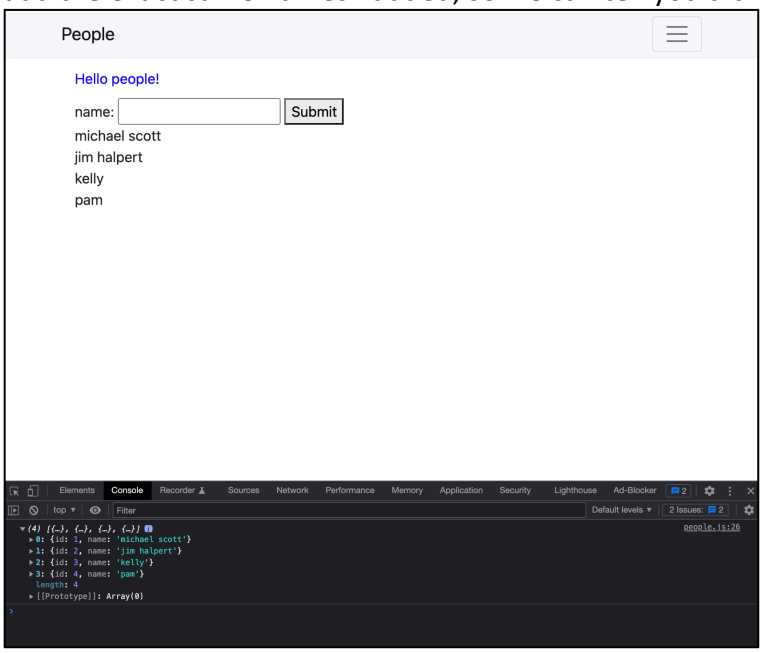
Install Flask on your computer, download and unzip our example flask project (people.zip) from the course website.
Unzip the file, and open a new terminal within the "/people" folder.
At the terminal in the "/people" folder, execute the python code as "python server.py"
Look at the error messages to see if you need to install any packages (if you've never run flask before, you probably do).
Install other packages you might need (flask render_template, Response, request, jsonify).
On a mac, usually the command "pip install flask" will work to install necessary package.

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

It should look like this:



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. In “templates/layout.html”, change the word “People” in the navbar to “Columbia Paper Infinity”.
    a. Show a screenshot of http://127.0.0.1:5000/ where we can see this change.
2. Go to http://127.0.0.1:5000/hello/<your name here>.
    a. Show a screenshot of the UI that has your name on it.
3. Go to http://127.0.0.1:5000/people
    a. Add some names
    b. Refresh the page and make sure the names are there (but don’t restart the server)
    c. If new data is still there after the refresh, rejoice! You did it! You can now save data on the server!
    d. Find the get_and_save_name() function in static/people.js
    e. When the ajax function is successful, add a console.log() statement that displays the data the server sends back.
    f. Take a screenshot that shows the data that was logged, kinda like this: (do not add the exact same names I added, so we can tell you did it yourself
    g. 

4. In static/main.css, change the color of the Hello People div to be any color that is not blue
    a. Show a screenshot of http://127.0.0.1:5000/people with your new color

Main:

What to submit:

- A folder called **infinity_*UNI*** (put your UNI in the UNI placeholder) that contains all your files. You may zip the folder if you like.
- A link to a Youtube Video demonstrating the functionally:
    - First show all the main original features still work. Show the user:
        - adding a sale (including the autocomplete feature)
        - deleting a sale with the X button,
        - deleting a sale with the trash can,
        - seeing an error when there are no reams entered
    - Then show that saving data works.
        - Add a bunch of data, then refresh the page – the page should go blank for a half second while the page reloads, then all the data will be there.
        - Also show that when you open a new browser tab and go to the URL, the new data displays there too.
    - Show autocomplete is updated.
        - Enter a name of a new company – "Dunder Mifflin"
        - Show that after you enter the name, it appears in the autocomplete when you enter the next sale.

Problem 1. Saving paper sales on the server

Start a Flask project to be the backend behind the log_sales application from HW4. You may want to start by copying your warm up application

1. There should be two pages to this site.
    a. A welcome page at the "/" route, which renders a template called "welcome.html"
    b. A log sales pages at the "/infinity" route that renders a template called "log_sales.html"
2. There should be a layout.html template with a navbar with links to the welcome page and the log_sales page.
3. Both pages need to extend the layout.html template
4. The welcome page doesn't have much. Just some text that says
    a. "Welcome to Columbia Paper infinity! Remember to log all your sales on this site, even ones you make on the phone."
    b. You may style this however you want.
    c. Put any CSS that you use in static/main.css
    d. Put any JS that you use in static/welcome.js (you probably won't use any JS here)
5. The log sales page must have all the same functionality as it did in HW4. Additionally, the data will be stored on the server, so that when you reload the page, it will still be present. (However, if you restart the flask server, new data will disappear)
6. For the new /infinity page,

a. Put all CSS in static/main.css
b. Put almost all JS that you use in static/log_sales.js
c. In the html file, remember to extend layout.html so you can see the nav bar. This means you will have to get rid of any <head> information from your old log_sales.html. Much of it you can probably delete, most of that should already be in layout.html, except including log_sales.js (see next question)
d. Your log_sales.html file must load/include an log_sales.js file

7. The server must send two variables when it renders the log_sales.html page: the list of sales, and the list of clients.
8. The log_sales.html page must read in those variables in a <script> tag (see the warm up for an example of this)
9. Log_sales.html and log_sales.js may no longer have any other variables that represent the data (i.e. delete your data representation from the old log_sales.js)
10. On the server (in server.py), you must have 3 variables that store all the data. (see the following page)
    a. current_id
    b. sales
    c. clients
11. In log_sales.js you must have three functions:
    a. **function display_sales_list (sales){…}**
        i. This takes in an array of sales and displays them on the UI. This contains no ajax calls.
    b. **function save_sale (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. **function delete_sale (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.
12. On the server, you must have two routes (and functions) that don't render pages, but do process data. Call them:

a. “save_sale”
    i. It takes in a sale (like in part 11.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",
];
```