Homework 4: JavaScript Widgets

Warm up: due Friday 2/11 @ 11:59pm on Courseworks
Main: due Tuesday 2/15 @ 11:59pm on Courseworks.

Warm-up:

<u>*What to submit:*</u>
A folder called **warmup_*UNI*** (put your UNI in the UNI placeholder) that contains files with the following titles:

- dd_warmup.html
- dd_warmup.js
- dd_warmup.css (optional)

You may zip the warmup folder if you like.

<u>*Problems:*</u>

1. Write a short sentence describing your participation on 2/9. (unless instructed otherwise by your section TA)
2. Create a simple drag and drop interface using the JQuery draggable and droppable widgets.
    a. There are two blue divs. One says “Non-PPC”, the other says “PPC”. You may hard code these in HTML.
    b. Under Non-PPC there are two divs with people’s names :“Phyllis” and “Angela.” For this problem, you may hard code these in HTML. For Main Problem 2, you will need to create these divs them dynamically in JavaScript.
    c. When you hover over either of the name divs, the background color should turn light yellow and the mouse cursor should turn into the “move” cursor
        i. Move cursor looks like this: ✥
    d. Widget CSS customization 1: when you drag a name div, it should still be light yellow and have a move cursor.
    e. Widget CSS customization 2: When you drag a name div and drop it on the blue PPC div, it should:
        i. Console.log() the name of person. (We recommend storing the name in HTML element using the data-* attributes. For example: <div data-name=”Phyllis”>
        ii. The div should stay put (it is successfully dropped).
    f. When you drag a name div and drop it anywhere other than the blue PPC div it should revert back to its location. Don’t code this yourself. Look for a property of the draggable or droppable widget that will do this for you.
    g. We highly recommend looking at the https://jqueryui.com/droppable/ examples and documentation.

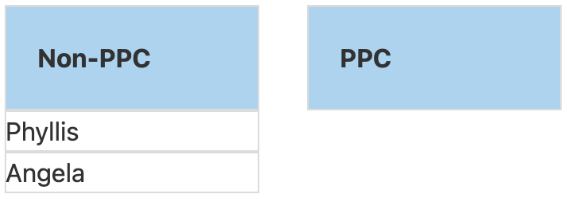
The site should be implemented in a folder called dd_warmup_UNI that has the following files:

- One HTML file called dd_warmup.html
- One JS file called dd_warmup.js
- (Optional) a CSS file called dd_warmup.css

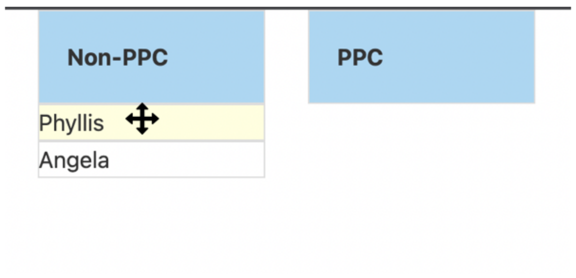
Here are JS + Bootstrap you will need for all the problems in this homework (main and warm up)

```
<!-- JQuery -->
<script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="https://code.jquery.com/ui/1.12.0/jquery-ui.min.js"></script>
<link rel="stylesheet" href="http://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">

<!-- bootstrap same as before -->

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
```

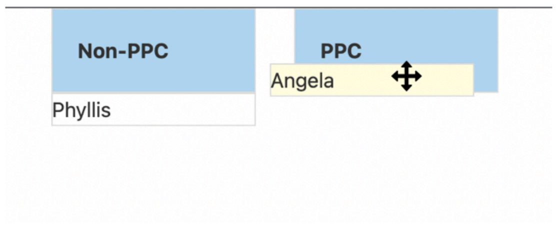
Original state:



When the mouse is hovering on a name div



While dragging a name div



Main:

What to submit:

- A folder called **log_sales_*UNI*** (put your UNI in the UNI placeholder) that contains files with the following titles:
    - log_sales.html
    - log_sales.js
    - log_sales.css (optional)
- A folder called **ppc_*UNI*** (put your UNI in the UNI placeholder) that contains files with the following titles:
    - ppc.html
    - ppc.js
    - ppc.css (optional)

You may zip the warmup folder if you like

Problem 1.
Logging Paper Sales with an Autocomplete Widget.

You a building a website called Columbia Paper Infinity for the salespeople at Columbia Paper to log their paper sales.

The interface to log sales should allow users to enter a sale, see all previously entered sales, and delete sales. It will all be implemented in HTML, CSS, Bootstrap, JavaScript, and JQuery. It will not have a back-end, so the results won't actually save if you close the page and open it again. Focus on the implementation of features, rather than design.

The requirements for the site are as follows:

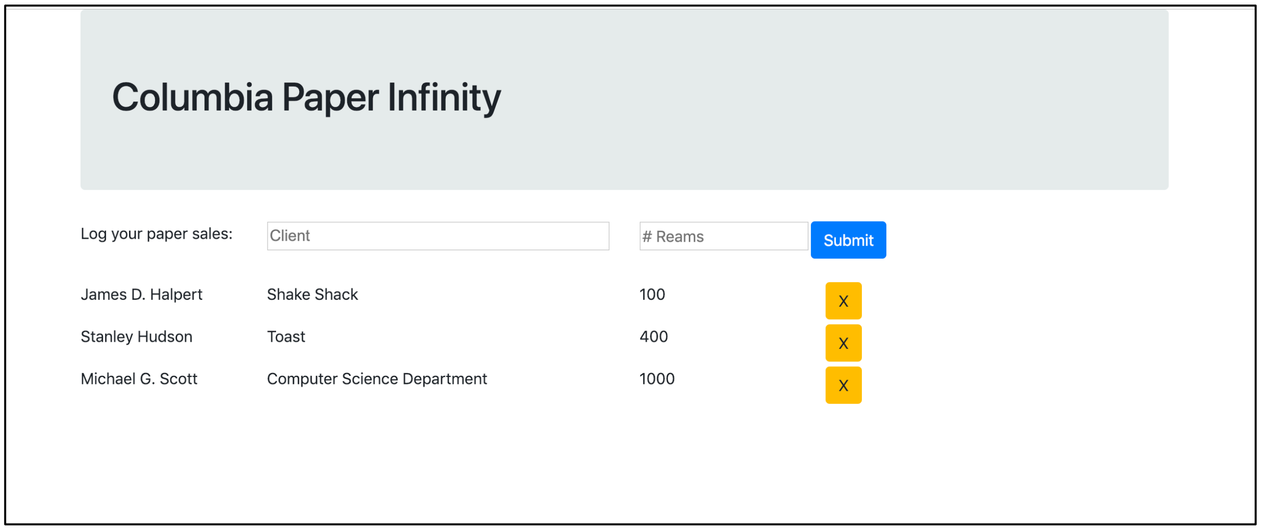
1. Write a short sentence describing your participation on 2/14. (unless instructed otherwise by your section TA)
2. The site should be implemented in a folder called log_sales_UNI that has the following files:
   - One HTML file called log_sales.html
   - One JS file called log_sales.js
   - (Optional) a CSS file called log_sales.css
3. At the top of the page should be a large header that says "Columbia Paper Infinity"
4. The user must see two text boxes: one to enter the client to whom the paper was sold, and another to enter the number of reams of paper.
   The client text entry widget should have the word "client" as placeholder text.
   The reams text entry widget should be short and have the word "# reams" as placeholder text.
5. The client box must have JQuery autocomplete populated with the names of clients from this file:
   - http://coms4170.cs.columbia.edu/2022-spring/hw/hw4/js/clients.js
   - Copy the data from this file into your own js file (log_sales.js)
   - Use the JQuery autocomplete feature. If conflicts with bootstrap, so you may have to not to bootstrap for these text inputs (however, you will still be able to use bootstrap for the other parts of the page.)
6. If the user enters a client name that is not in the autocomplete, add it so that it will be in the autocomplete the next time they type the same name.
7. Beneath the client entry box, there must be a list of previously entered sales records. The information in the records must align with the grid structure with the input boxes above them. The sales records must start with this data:
   - http://coms4170.cs.columbia.edu/2022-spring/hw/hw4/js/sales.js
   - Copy the data from this file into your own js file (log_sales.js)
   - You may not change the data format.
   - No <table>'s.
8. When the user enters a new sale, a new record must show up as the first record in the list, directly below the input boxes. It must include the salesperson's name. Hard code

the name of the salesperson in JavaScript with a const variable. That will be the only person logging sales in this assignment. This must be implemented in the Model + View/Controller style demonstrated in class. This means that:

- NOTE: When a change is made to the underlying data (the model), you do not make changes to the UI (the view) directly. Instead, you first update the json data (the model), then update the appearance (the view). This way the model and the view are always in sync. The easiest way to update he view from the model is to remove the entire list from the UI and then regenerate the entire list (with the updates) from the updated data. You could do clever things with adding and deleting elements, but removing and regenerating is actually an easy and great way to do this.

9. For each sales record, there must be a delete button which will remove it from the list of sales records. This must be implemented in the Model + View/Controller style: first delete the data from the json data (the model) then regenerate the view from the updates json data. Do not just delete the elements of the UI.
10. After the new entry is added, the interface must go back to a state where the user can immediately start typing a new sale. This means:
    - The text boxes must clear
    - The cursor must go to the client input box.
11. There must be two ways to enter a sales record:
    1. Pressing a "submit" button near the text input fields
    2. Pressing "enter" in the "# reams" text input
12. If either the client field or the reams field is empty when the user tries to submit, three things must happen:
    1. The data must not submit
    2. The webpage must throw produce a warning in the UI to identify each error next to the field where it should be corrected. Do not use an alert() box.
    3. The webpage must move the cursor to the first field that was empty (so that the user can easily type there)
    4. Once the data is correctly submitted, the warnings should disappear.
13. If the “# reams” is not a number (but is non-empty), the webpage must warn the user that the number of reams is not a number in a manner consistent with the warning produced in the previous problem. The data must not submit, and the cursor should go back to the reams field. Do not clear the data in either field.

Here is a screenshot of my design.
You do not have to copy it.
It should only serve as a guide to what we expect.



Columbia Paper Infinity

James D. Halpert	Shake Shack	100	X
Stanley Hudson	Toast	400	X
Michael G. Scott	Computer Science Department	1000	X

Problem 2:

Editing the Party Planning Committee with Drag & Drop

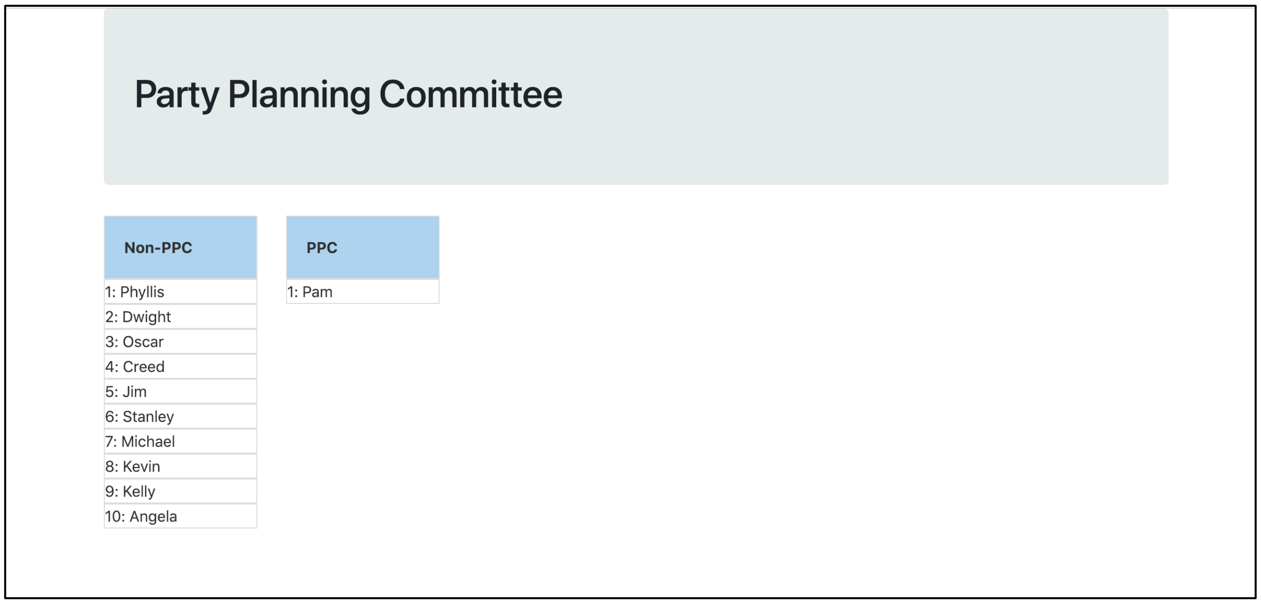
You a building a website called Party Planning Committee for the employee at Columbia Paper to keep track of who is on the party planning committee by moving people on and off with a drag and drop interface. It will all be implemented in HTML, CSS, Bootstrap, JavaScript, and JQuery. It will not have a back-end, so the results won't actually save if you close the page and open it again. Focus on the implementation of features, rather than graphic design. Graphic design should be minimal.

The requirements for the site are as follows:

1. The site should be implemented as
   - One HTML file called ppc.html
   - One JS file called ppc.js
   - (Optional) one CSS file called ppc.css
2. At the top of the page should be a large header that says "Party Planning Committee"
3. The PPC site needs to display two lists:
   - People **not on** the PPC:
     1. In its default state it should show all the employees.
     2. You can copy that from here:
     3. http://coms4170.cs.columbia.edu/2022-spring/hw/hw4/js/employees.js
   - People **on** the PPC.
     1. In its default state, the party planning committee is empty.
4. Each list must have a div at the top of the list with the label "Non-PPC" or "PPC". This label should be large enough to serve as a drop target. They should be big enough so that it's not painful to try to drop things there.
5. Using JQuery Draggable and Droppable events, implement an interface where the user can drag names from the Non-PPC list to the head of the of the PPC list, and when they drop it, the name will get added to the end of the PPC list. The reverse must also be true: names from the PPC list can be dragged to the header of the non-PPC list to move them off the PPC.
   This must be implemented in the Model + View/Controller style demonstrated in class and described in the note in the previous problem.
6. To cue that an element is draggable, implement the UI such that when a draggable element is hovered over, its background turns light yellow, and the cursor changes to the "move" cursor
7. While the item is being dragged, the background should also be light yellow, and the cursor should still be the "move" cursor.
8. While the item is being dragged, it should look like it is on top of all the other elements on the page, so it is fully visible.

9. While the item is being dragged, the drop target should turn a darker shade of whatever color you made it.
10. When the item is dragged over the drop target, the drop target should turn an even darker shade.
11. If an item is dropped anywhere other than an appropriate drop target for that item, it should revert back to the place where the user started dragging it.

Here is a screenshot of my design.
You do not have to copy it.
It should only serve as a guide to what we expect.



Party Planning Committee

Non-PPC	PPC
1: Phyllis	1: Pam
2: Dwight	
3: Oscar	
4: Creed	
5: Jim	
6: Stanley	
7: Michael	
8: Kevin	
9: Kelly	
10: Angela	