Homework 5: JavaScript Widgets

Due Friday 2/22 @4pm on Courseworks.

What to submit:

- 4 separate files with the following titles:
    - 1_log_sales.html
    - 1_log_sales.js
    -
    - 2_ppc.html
    - 2_ppc.js

- Submit them separately to Courseworks, do **not** zip them up.

Problem 1.
Logging Paper Sales with 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. Design should be minimal.

The requirements for the site are as follows:

1. The site should be implemented as an
    - One HTML file called 1_log_sales.html (all CSS goes here)
    - One JS file called 1_log_sales.js
2. At the top of the page should be a large header that says "Columbia Paper Infinity"
3. 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.
4. The client box must have JQuery autocomplete populated with the names of clients from this file:
    - http://coms4170.cs.columbia.edu/2019-spring/assignments/hw5/js/clients.js
    - Copy the data from this file into your own js file.
    - 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.)
5. 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.
6. 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 this data:
    - http://coms4170.cs.columbia.edu/2019-spring/assignments/hw5/js/sales.js
    - Copy the data from this file into your own js file.
    - You may not change the data format.
    - No <table>'s.
7. 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. That will be the only person logging sales in this assignment. This must be implemented in the Model + View/Controller style demonstrated in class.

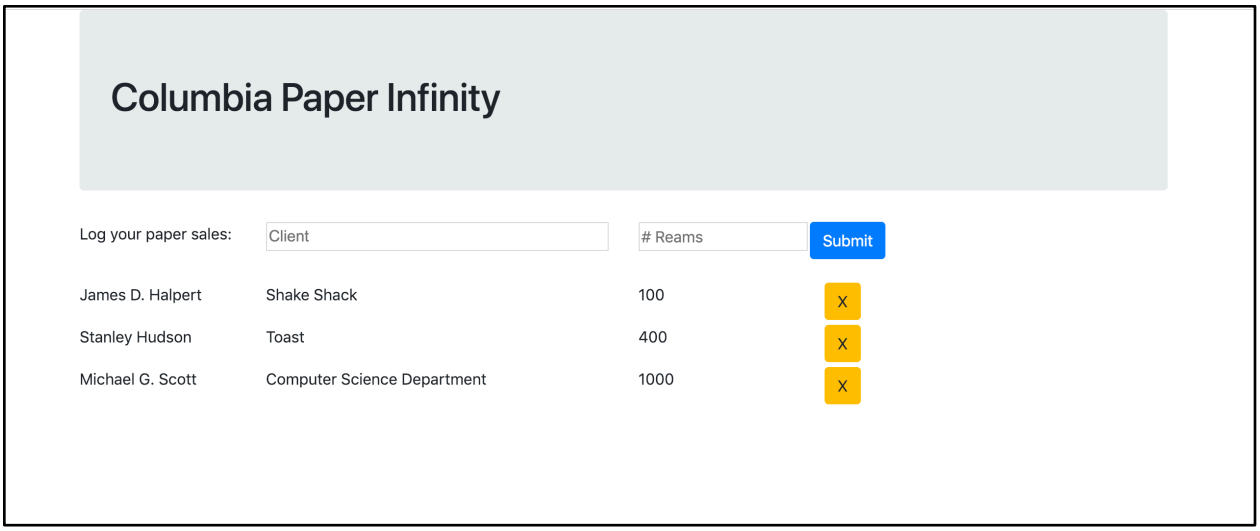
8. For each sales record, there must be a delete button which will remove it from the list of sales records. This must be implemented in the Model + View/Controller style demonstrated in class.
9. After the new entry is added, the interface must go back to a state where the user can immediately start typing a new sale. This means:
    - The text boxes must clear
    - The cursor must go to the client input box.
10. 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
11. 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 an alert warning the user of the first field that was empty
    3. The webpage must move the cursor to the first field that was empty (so that the user can easily type there)
12. If the “# reams” is not a number, the webpage must throw an alert warning the user that the number of ream is not a number. The data must not submit, and the cursor should go back to the reams field. Do not clear the data in either field.

Here are JS + Bootstrap you will need:

<!-- 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">

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.



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 design. Design should be minimal.

The requirements for the site are as follows:

1. The site should be implemented as
    - One HTML file called 2_ppc.html (all CSS goes here)
    - One JS file called 2_ppc.js
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 all the employees.
        2. You can copy that from here:
        3. http://coms4170.cs.columbia.edu/2019-spring/assignments/hw5/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.
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.** ~~of yellow~~.

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 animate 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.