

Homework 4: JavaScript Widgets and MVC

Warm up: due Friday 2/14 ❤️ @ 11:59pm on Courseworks

(**grace period until Sunday 2/16 at 11:59pm**)

Main: due Tuesday 2/18 @ 11:59pm on Courseworks.

Warm-up:

What to submit:

1. A zip file of a folder called jquery_dd_warmup_UNI that has the following files:
 - Jquery_dd_warmup.html
 - Jquery_dd_warmup.js
2. A link to a short (30 sec) YouTube video showing you using your site and explaining how it works, showing code, when necessary (either with the Developer Tools or by showing you code in the code editor). Be sure to show all the features listed.

Problems:


1. Copy and paste the example code for basic drag and drop from the JQuery documentation: at the <https://jqueryui.com/droppable/>
 - a. Get it to run locally on your computer.
 - b. You will have to copy and paste the links we'll use for jquery in this class. (without the css file, the the example won't render with the right styles.)

<!-- 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>
```

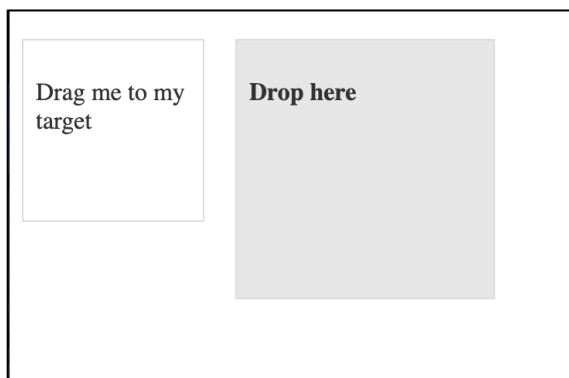
```
<link rel="stylesheet" href="http://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
```

- c. Separate this file into two files:
 - i. Jquery_dd_warmup.html
 - ii. Jquery_dd_warmup.js
 - iii. (you will have to link the .js file to the .html file)
- d. Widget Customization #1: When the draggable object is hovered over, the cursor should change to the "move" cursor. You can Google the move cursor or read about it here: https://www.w3schools.com/cssref/pr_class_cursor.php This should be fairly straightforward (one line of code)
 - i. Move cursor looks like this: 
- e. Widget Customization #2: when the draggable object is dragged, but not dropped on a valid drop target, it should revert back to the position that it

started at. Google it or explore the other jQuery drag and drop examples. This should be fairly straightforward (~1 line of code)

2. Make a short video showing you use the application with each of the features. You do NOT need to explain how you implemented it for this video.
 - a. Hover over the draggable object so we can see the move cursor.
 - b. Drag and drop the draggable object so that it reverts.
 - c. Drag the object the droppable object around slowly for a bit so we can see it change colors when the draggable object can be dropped.
 - d. Drag and drop the draggable object so that it successfully drops (and does not revert).

For reference: this is what the base code should look like:



Problem 1.

MVC Paper Sales with Drag & Drop and Autocomplete Widgets.

Main:

What to submit:

1. A zip file of 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)
2. A link to a short (2-3 min) YouTube video showing you using your site and explaining how it works, showing code, when necessary (either with the Developer Tools or by showing you code in the code editor). Be sure to show all the features listed.
 - **UPDATE: In your video, you must say your first and last name in the beginning of the video. Also write your name in the code (or website) so we can see it in the video.** (This is so the TAs can easily know who's video is who's when grading each person's video)

You are 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. 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
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/2024-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.)
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 they type the same name.
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 start with this data:
 - <http://coms4170.cs.columbia.edu/2024-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.
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 with a const variable. That will be the only person logging new sales in this assignment. This must be implemented in the Model + View/Controller style demonstrated in class. This means that:
 - 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 the 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.
 - Note: You could do more clever things with adding and deleting single elements, but removing and regenerating the entire list is easy and great way to do this.
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: first delete the data from the json data (the model) then regenerate the view from the updated json data. Do not just delete the elements of the UI.
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 the "submit" button
 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 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.
12. 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.
13. There needs to be a div that is labeled trash that users can drag and drop sales rows into to delete them.
- When the user hovers over any row in table, the background should turn lightyellow and the cursor should turn into the move cursor
 - The user should be able to drag a row in the table around the screen.
 - When the user has dragged a row to the droppable part of the trash div, the trash div should turn yellow to provide feedback that it is possible to drop it here.
 - When the user drops it, the data must be deleted in the Model + View/Controller style, just like it was in the delete button: first delete the data from the json data (the model) then regenerate the view from the updated json data. Do not just delete the elements of the UI.
 - If the user stops dragging the row before it reaches the trash, the row should revert to its original position. It should not stay where ever it was dropped.
 - We recommend making the trash div about as wide as a row to make it an easy drop target.

Here are JQuery + 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 href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
```

Here is a screenshot of my design.

You do not have to copy it.

It should only serve as a guide to what design quality we expect: just basic readability and layout.

Columbia Paper Infinity

trash

Log your paper sales:

	<input type="text" value="Client"/>	<input type="text" value="# Reams"/>	<input type="button" value="Submit"/>
James D. Halpert	Shake Shack	100	<input type="button" value="X"/>
Stanley Hudson	Toast	400	<input type="button" value="X"/>
Michael G. Scott	Computer Science Department	1000	<input type="button" value="X"/>