Individual programming project – part 1: building an interactive to-do list web page

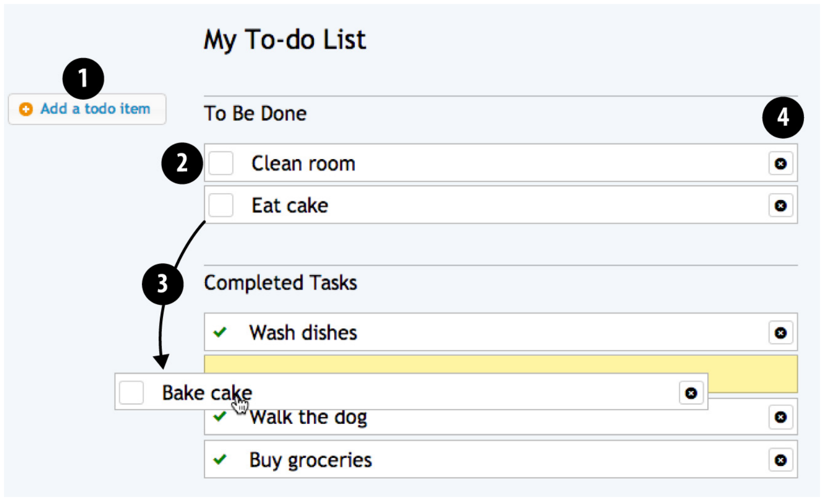
Part 1 of the individual programming project helps you get familiarize with JavaScript and jQuery for building the front-end of a web application. In particular, you will learn how to use the jQuery library and the jQueryUI plug-in in building a highly interactive web page in just a few steps. For this purpose, this part of the programming project walks you through the construction of a basic to-do list web page. Successfully completing the to-do list web page will contribute **5%** to your total mark of NWEN304.

1. An overview of the to-do list web page

Your to-do list web page will let users do the following

- Add a new **task to the list of to-dos.** You'll achieve that by adding a jQuery UI button that, when clicked, opens a jQuery UI dialog box.
- **Mark a task as complete.** Each to-do list item will have a checkbox to the left of the task name. Clicking this box will automatically remove the to-do list item from the *To Be Done* list and move it to the *Completed Tasks* list.
- **Remove tasks from either list by clicking a Delete button**. To allow complete removal of a task, you'll supply a Delete button that removes it from the page.

A schematic view of the web page to be developed is shown below.



2. Things to do

2.1 *Download web page templates*

Download the web page template, including the jQuery library, jQueryUI plug-in and necessary CSS files, from our course website (all files needed are provided together through a zip file). You are highly recommended to use our template to finish your lab project. Our tutors cannot provide technical guidance if you choose NOT to follow our template. In this case, your work will be marked purely by the required functionalities implemented on your web page.

2.2 *Add a button to your web page*

In a text editor, open the file *index.html* that you can find in the provided template. This file already contains links to the necessary CSS, jQuery, and jQueryUI files. However, it doesn't yet load the *todo.js* file — the file you'll add the programming to. Therefore, the first thing is to link to *todo.js* in your web page. This can be done easily by using the following.

```
<script src="todo.js"></script>
```

Next, let's add a button to the web page. Locate the HTML comment *<!-- add button here -->* and replace it with the following HTML:

```
<button id="add-todo">Add to-do item</button>
```

It is now the time to format the newly added button and make sure that it looks nice. Open the *todo.js* file in your text editor, and, inside the *$(document).ready()* function, type the following

```
$('#add-todo').button({ icons: { primary: "ui-icon-circle-plus" } });
```

The above code gives a little + icon to the left of the text "Add to-do item" on the button. For your information, jQuery UI lets you place two icons on a button, one on the left (the primary icon) and one on the right (the secondary icon). One icon is good enough for this button.

2.3 *Add a dialog box*

We have now added a button. But when the button is clicked, nothing will happen. To make the button useful, we want to add a dialog box. Locate the comment *<!-- add dialog box here -->* in the *index.html* file and replace it with the following HTML:

```
<div id="new-todo" title="Add to-do item">
        <form>
                <p>
                        <label for="task">Task Name:</label>
                        <input type="text" name="task" id="task">
        </p>
    </form>
</div>
```

With the help of jQueryUI, we can turn any chunk of HTML into a dialog box. For this purpose, add the JavaScript code below to *todo.js*:

$('#new-todo').dialog(**{ modal : true, autoOpen : false** });

In the code above, the *modal* option requires a user to close the dialog box in order to do anything else with the web page. Meanwhile, *autoOpen* is set to false to make sure that the dialog box will not be opened immediately upon loading the web page in a browser.

Since the dialog box will not be opened by default, we should open it when the button added in 2.2 is clicked by user. For this purpose, make the following changes to *todo.js*:

```
$('#add-todo').button({
    icons: { primary: "ui-icon-circle-plus" }}).click(
        function() {
            $('#new-todo').dialog('open');
        });
```

If you can open the dialog box now by clicking the button, we shall move on to add more functionalities to the dialog box. In particular, we would like to add two buttons to the dialog box, one for adding a new task to the to-do list and one for cancelling the addition of tasks. For this purpose, add the following code to *todo.js*:

```
$('#new-todo').dialog({
    modal : true, autoOpen : false,
    buttons : {
        "Add task" : function () { },
        "Cancel" : function () { $(this).dialog('close'); }
    }
});
```

In the above, we added two buttons to the dialog box, one reads as “Add task” which does nothing at the moment. The other button has the text “Cancel” and will close the dialog box whenever clicked.

2.4 *Add new tasks to the to-do list*

This step is very important for us to build a working to-do list. To achieve this goal, we need to retrieve the task entered by the user first and check whether the entered task is valid. This can be easily done by making the following changes to *todo.js*:

```
"Add task" : function () {
        var taskName = $('#task').val();
        if (taskName === "") { return false; }
},
```

This code simply checks to make sure the *taskName variable* doesn't contain an empty string. The *return false* part just exits this function, with the result that the dialog box stays open. Next, let's construct the HTML for the newly added task by using the JavaScript code below.

```
"Add task" : function () {
        var taskName = $('#task').val();
        if (taskName === '') { return false; }
        var taskHTML = '<li><span class="done">%</span>';
        taskHTML += '<span class="delete">x</span>';
        taskHTML += '<span class="task"></span></li>';
        var $newTask = $(taskHTML);
        $newTask.find('.task').text(taskName);
},
```

It is interesting to note that jQuery can convert HTML directly to jQuery object by using $(), as demonstrated in the code above. We want the process of adding new tasks visually appealing. We also want to close the dialog box once a new task is added. For this purpose, add the code below to the "Add task" function.

```
        $newTask.hide();
        $('#todo-list').prepend($newTask);
        $newTask.show('clip',250).effect('highlight',1000);
        $(this).dialog('close');
```

This step is almost finished. However there is one small issue – the text input field in the dialog box still remembers the last task user entered when you open the dialog box again. Please add code to solve this problem.

2.5 *Mark tasks as complete*

Any pending tasks in the to-do list eventually should be marked as complete. This feature is not currently supported by your web page. To support this important feature, add the following JavaScript code (i.e. a click event handler) to *todo.js*:

```
$('#todo-list').on('click', '.done', function() {
        var $taskItem = $(this).parent('li');
        $taskItem.slideUp(250, function() {
                var $this = $(this);
                $this.detach();
        });
});
```

Can you figure out the logic of the code above? Please give it a try. Here jQuery's detach() method is used to remove the selected HTML element or elements from the page, but doesn't completely get rid of them. In other words, the list item is gone from the list, but it still exists in memory. You can then move the detached element to another list. This is very easy by adding two lines of code:

```
        $('#completed-list').prepend($this);
        $this.slideDown();
```

2.6 *Support drag and drop*

To support drag and drop, we can use the *sortable feature* provided by jQueryUI. To use this feature, we need to make the to-do list and complete list as sortable. Use the following scripts to make necessary changes to *index.html* and *todo.js*:

```
        <ul id="todo-list" class="sortlist">
        <ul id="completed-list" class="sortlist">

        $('.sortlist').sortable();
```

After these changes, we can re-order items in a list but we cannot drag items from one list to another. To enable drag and drop across two different lists, we have to connect the two lists together. This requires us to make further changes to *todo.js*:

```
$('.sortlist').sortable({
        connectWith : '.sortlist',
        cursor : 'pointer',
        placeholder : 'ui-state-highlight',
        cancel : '.delete,.done'
});
```

The *connectWith* option lets you connect one list with another. Since both the two lists on our web page are associated with the same class name, we can specify the class name to connect them together. In addition, the *cursor* option changes the cursor to a pointer when a list item is being dragged; the *placeholder* option highlights the space in the list where a user can drop an item; and the *cancel* option identifies elements on a list item that won't work as handles for dragging the item (can't drag a task by either its "delete this task" icon, or its "mark as done" box).

2.7 *Delete tasks*

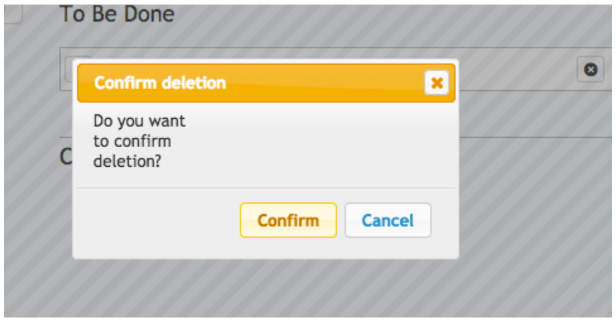
The to-do list will become very long over time if no item in the list can be deleted. To implement the deletion function, add a new event handler as shown below to *todo.js*:

```
$('.sortlist').on('click','.delete',function() {
        $(this).parent('li').effect('puff', function() { $(this).remove(); });
});
```

Can you imagine what is going on in the code above? jQueryUI's *effect()* method applies one of its many effects to the element. In this case, the *puff effect* makes an element grow in size, fade away, and disappear. The function inside the *effect()* method is a callback function that runs once the effect is done.

2.8 *Confirm task deletion*

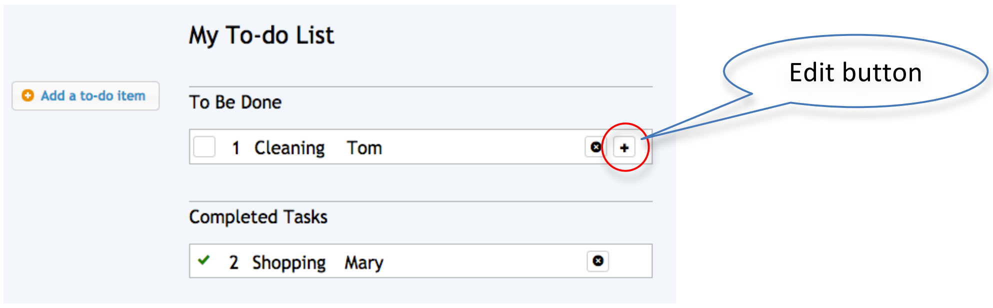
Currently, when a user clicks the Delete button, the task is gone forever. You could add a modal dialog box (as shown below) that asks the user to confirm the action. If the user clicks "Confirm", the task is removed; if "No", the task stays as is. No code will be provided for this step. Please write your own JavaScript code in *todo.js* and also make necessary changes to *index.html*.

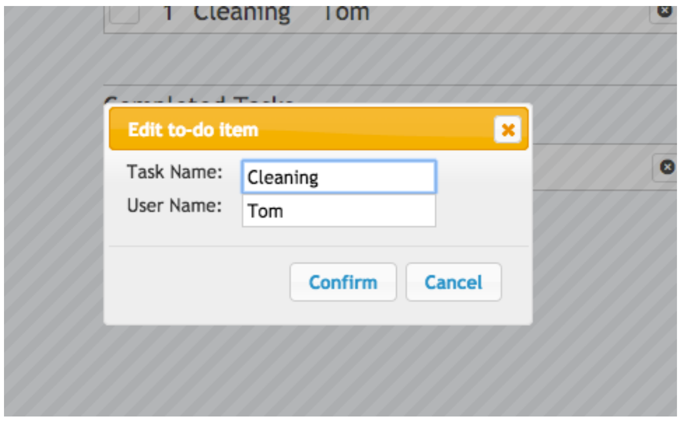


Do you want to confirm deletion?

2.9 *Edit existing task items*

Implement a new functionality to edit existing task items in the *To Be Done* list only. Particularly, as highlighted in the figure below, each task item is associated with a new button for the editing function. Once the button is clicked, a new dialog box will be displayed (one example is also given below) for users to update the task name/description based on the existing values. Notice that in the example provided, users can also update their names for any existing task items in the to do list. This functionality is NOT compulsory for this step. Upon confirmation of the update through the dialog box, the corresponding task item in the *To Be Done* list will also be updated accordingly.





3. Marking guide

This lab project will be marked out of 100%. Below please find a summary of the distribution of marks over different steps.

Steps	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
Weight	0%	10%	10%	10%	10%	10%	10%	15%	25%

The main functionalities to check for each step are:

- 2.2: New button has been added successfully with desirable visual appearance.

- 2.3: When the newly added button is clicked, a dialog box with two buttons will be opened for adding new tasks. When the “Cancel” button of the dialog box is clicked, the dialog box will be closed with no change to the to-do list.
- 2.4: New tasks can be added successfully to the to-do list by using the dialog box.
- 2.5: Tasks in the to-do list can be marked as complete and moved to the complete list.
- 2.6: Tasks can be dragged and dropped within each list and also across the to-do list and complete list.
- 2.7: Tasks can be deleted from the to-do list and complete list.
- 2.8: A dialog box will be opened for confirmation when user wants to delete a task.
- 2.9: A new edit button is made available for each task item in the to-do list. Once an edit button is clicked, a dialog box will be displayed to help users edit the corresponding task item in the to-do list based on existing values.

4. Things to know for submission

Only online submission is allowed. Please submit the completed web page together with all necessary resources (i.e. CSS files, images, js files, jQuery library, etc.) in a single zip file. After submission, please download your web page again from the submission system and test its functionality (using the Google Chrome Web browser). Please note that ultimately you are responsible for ensuring that you have submitted your work that executes properly on our system.