4.3.9 Save the Edited Task

On our journey to update/edit a task, we first had to load the task's information into the form. The next bit of logic will involve saving the task and changing the necessary DOM elements.

PAUSE

Because we're using the same form, how do we know when a new task is being created versus an old task being updated?

We know it's a new task if the form has a `data-task-id` attribute on it.

Hide Answer

The `data-task-id` attribute that we added to the form will serve two purposes. One, it keeps track of which task we're editing. Two, its existence lets us know that, yes, a task is being edited in the first place. A handy way of knowing if an element has a certain attribute or not is to use the `hasAttribute()` method.

Add the following lines to your existing `taskFormHandler()` function above the declaration for `var taskDataObj` at the end:

```
var isEdit = formEl.hasAttribute("data-task-id");
console.log(isEdit);
```

With the DevTools console open, first create a new task, then edit a task and click the Submit button. (Remember, every time we submit, we run the `taskFormHandler` function.) When we submit the form to create a new task, `isEdit` will be `false`, and `false` will display in the console. When we submit after editing, `isEdit` will be `true`, and `true` will display in the console. Awesome—this means we can use the same form handler for new and old tasks!

Let's now do two things. First, delete the `console.log()` beneath `isEdit`. Then, replace `createTaskEl(taskDataObj);` with the following block of code, which wraps the `createTaskEl()` call and `taskDataObj` variable in an if/else statement. Make sure that it follows the `isEdit` variable initialization:

```
// PUT THIS BELOW `var isEdit = ...` in `taskFormHandler()`

// has data attribute, so get task id and call function to complete ed
if (isEdit) {
  var taskId = formEl.getAttribute("data-task-id");
  completeEditTask(taskNameInput, taskTypeInput, taskId);
}
// no data attribute, so create object as normal and pass to createTas
else {
  var taskDataObj = {
    name: taskNameInput,
    type: taskTypeInput
  };

  createTaskEl(taskDataObj);
}
```

This way, `createTaskEl()` will only get called if `isEdit` is `false`. If it's `true`, we'll call a new function, `completeEditTask()`, passing it three arguments: the name input value, type input value, and task ID.

We'd best create that function now:

```
var completeEditTask = function(taskName, taskType, taskId) {
  console.log(taskName, taskType, taskId);
};
```

Console log the parameters to verify that our function is working and getting the data it needs. Restart the app, create a task, select edit to bring it into the form fields, then submit. If all went well, the `taskNameInput`, `taskTypeInput`, and `taskId` will be logged to the console.

We'll use `taskId` to find the correct `<li>` element and use `taskName` and `taskType` to update the `<li>` element's children accordingly.

Replace the `console.log()` in `completeEditTask()` with the following code:

```
// find the matching task list item
var taskSelected = document.querySelector(".task-item[data-task-id='"

// set new values
taskSelected.querySelector("h3.task-name").textContent = taskName;
taskSelected.querySelector("span.task-type").textContent = taskType;

alert("Task Updated!");
```

Also reset the form by removing the task ID and changing the button text back to normal:

```
formEl.removeAttribute("data-task-id");
document.querySelector("#save-task").textContent = "Add Task";
```

By removing the `data-task-id` attribute, we ensure that users are able to create new tasks again. Try it out in the browser, switching between making tasks and editing existing tasks. If anything seems off, check the console for errors. An error like `Uncaught TypeError: Cannot set property 'textContent' of null`, for instance, suggests one of the selectors is wrong.

One opportunity for a bug lies in the `querySelector`s in the code above, if you were to make a minor typo. `h3.task-name` isn't the same as `h3 .task-name`: the former looks for `<h3>` elements with a class of `task-name`, whereas the latter looks for elements with a class of `task-name` that are descendent elements of an `<h3>` element. The latter would return `null`, and since there is no `textContent` property on `null`, the browser itself would report an error message in the console.