To Do List Assignment Walkthrough

Warning! Spoiler Alert! For those who want to work on the To Do List assignment without any help, **DO NOT** read the rest of this unit. It contains detailed guideline on how to build the basic features of the application.

Important note for graduated students

The old API with URL **https://altcademy-to-do-list-api.herokuapp.com/** has been deprecated, please change your code to use the new API **https://fewd-todolist-api.onrender.com/**.

========== End of note ==========

The Walkthrough

Right, you're stuck, and I am here to help. Building a web application from scratch can be a daunting task. But if we take baby steps, we can accomplish it without breaking a sweat. Let's first take a look at the assignment outline.

> "You will be building a To Do List web application that is connected to the ATDAPI service. It will have persistent data so that every time you visit the page, the list of tasks is the same as you left it. Use the example requests in the previous chapters as reference for making requests to ATDAPI."

And the minimum requirement we need to fulfill:

1. A list of tasks rendered in the DOM based on data from the ATDAPI server.
2. Each task has a description, a remove button, and a mark complete/active button.
3. An input element and a button that lets user add a new task.

So, we need to build a To Do List app that is connected to an API server. The app's data is permanently stored on the API server. Which means all the Tasks of our To Do List app are on the server. When we create, delete, or edit a task, it needs to be reflected on the server as well. This app will involve DOM manipulation, AJAX, and some basic HTML/CSS.

Setting Up Environment

The best way to start any project is to setup your base environment. We need an HTML, CSS and JavaScript file all linked up and working. Let's get to work.

In our HTML file, we need to link up the CSS file and the JavaScript file. Also, add Bootstrap and jQuery CDN links in there. We will be using Bootstrap 3 and jQuery 3.

Add a basic bootstrap structure and a heading. So we have something to look at when we open up the app.

```
copy dex.html-->
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>To Do List</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <link href="index.css" rel="stylesheet" type="text/css" />
  <script src="index.js"></script>
 </head>
 <body>
  <div class="container">
   <div class="row">
    <div class="col-xs-12">
     <h2>To Do List</h2>
    </div>
   </div>
  </div>
 </body>
</html>
```

In our CSS file, make a simple CSS definition so that we know it's working. We will delete it later.

```
copy ex.css*/
h2 {
 color: blue;
}
```

Finally, in the JavaScript file, console log a message.

```
copy ex.js
console.log("js is working today")
```

Check Point

Open up the index.html file with Google Chrome browser and make sure you see

1. A heading that says "To Do List".
2. The heading is blue.
3. A message in the console that says "js is working today"

Connecting To The API Server

The key of this assignment is to build a web application that connects to an API server. So the next thing we should do is to make sure we can make AJAX requests successfully.

The API service we are using is called ATDAPI, it is an API for managing To Do List tasks. We can see its API structure on https://fewd-todolist-api.onrender.com/.

To make sure we are making successful requests, let's make a GET request to get all the tasks belonging to the user with ID 1, which should already have some tasks created. In your index.js file, add the following.

```
copy ₹({
 type: 'GET',
 url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=1',
 dataType: 'json',
 success: function (response, textStatus) {
  console.log(response);
 },
 error: function (request, textStatus, errorMessage) {
  console.log(errorMessage);
 }
});
```

Check Point

You should be able to see the response being logged in the console. It will be an object containing a list of tasks.

```
copy :s: [{
  id: 175,
  content: 'stuff',
  completed: false,
  due: null,
  created_at: '2018-05-02T02:12:24.942Z',
  updated_at: '2018-05-02T02:12:24.942Z'
 },{
  id: 93,
  content: 'Do Laundry',
  completed: false,
  due: null,
  created_at: '2018-04-05T01:39:58.500Z',
  updated_at: '2018-04-14T14:12:15.817Z'
 },{
  id: 154,
  content: 'Hello',
  completed: false,
  due: null,
  created_at: '2018-04-14T14:44:18.955Z',
  updated_at: '2018-04-14T14:44:18.955Z'
 }]
}
```

Injecting The Tasks Into The DOM

Great, we will keep using the user ID 1 as our API key for now, since it already contains some tasks, which we need in order to render tasks into the DOM.

To dynamically add HTML content into the DOM, we can use the jQuery append method. The append method needs a target node in the HTML to append to. So we will add a div in the index.html file that will house our list of tasks. Add the following after the h2 heading element.

```
copy d='todo-list'>
</div>
```

We need to stop here for a moment and review some crucial concept. Recall that JavaScript code are run based on their order in the HTML file. Our index.js file is linked to our index.html file in the `<head>` element. Which means currently, all the code in our index.js file will be executed before any of the HTML elements in the body are created.

If we try to reference any of the `<div>` elements in index.js, we will get an error, because none of them are created yet. To solve, this, we should wrap our JavaScript code in a jQuery ready function. So that our programs only execute after the DOM is ready.

```
copy :ument).ready(function(){
 console.log('dom ready');

 $.ajax({
  type: 'GET',
  url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=1',
  dataType: 'json',
  success: function (response, textStatus) {
   console.log(response);
  },
  error: function (request, textStatus, errorMessage) {
   console.log(errorMessage);
  }
 });
});
```

Check Point

If you try to refresh the webpage, you should see:

1. The message 'dom ready' printed in the console.
2. The GET response printed in the console.

Now we are ready to inject the tasks as HTML into the DOM. To make this as painless as possible, first, we need study the structure of the response data we got from the GET request.

```
copy :s: [{
  id: 175,
  content: 'stuff',
  completed: false,
  due: null,
  created_at: '2018-05-02T02:12:24.942Z',
  updated_at: '2018-05-02T02:12:24.942Z'
 },{
  id: 93,
  content: 'Do Laundry',
  completed: false,
  due: null,
  created_at: '2018-04-05T01:39:58.500Z',
  updated_at: '2018-04-14T14:12:15.817Z'
 },{
  id: 154,
  content: 'Hello',
  completed: false,
  due: null,
  created_at: '2018-04-14T14:44:18.955Z',
  updated_at: '2018-04-14T14:44:18.955Z'
 }]
}
```

The response data is an object that has only one key value pair. The key is "tasks" and the value is an array of objects containing data of each task.

To get to the tasks array, we can use the dot notation like this `response.tasks`. Then, we should be able to iterate the individual tasks using a `forEach` loop. Before doing any DOM injection, let's make sure we can iterate each task successfully.

```
copy :ument).ready(function(){
  $.ajax({
    type: 'GET',
    url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=1',
    dataType: 'json',
    success: function (response, textStatus) {
      response.tasks.forEach(function (task) {
        console.log(task.content);
      })
    },
    error: function (request, textStatus, errorMessage) {
      console.log(errorMessage);
    }
  });
});
```

We are only console logging the content property of each task. Also notice we have removed the initial console log message that prints 'dom ready'. It is important to clean up unnecessary code if we don't need them anymore. This reduces clutter and chances of bugs and makes our life much easier.

Check Point

After refreshing the page. You should see the following in the console.

1. The content of each task.

```
copy
Do Laundry
Hello
```

> Note: the tasks you get might be different to mine. As the database will change overtime with users interacting with the Demo app.

Now we are ready to inject the tasks into the DOM as HTML content. We will use a basic `<p>` element to contain the content of each task for now. We will use the jQuery append method and concatenate the content of each task with the HTML string.

```
copy :ument).ready(function(){
  $.ajax({
    type: 'GET',
    url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=1',
    dataType: 'json',
    success: function (response, textStatus) {
      response.tasks.forEach(function (task) {
        $('#todo-list').append('<p>' + task.content + '</p>');
      })
    },
    error: function (request, textStatus, errorMessage) {
      console.log(errorMessage);
    }
  });
});
```

Check Point

After refreshing the page, you should see:

1. Task contents appear in the HTML after the AJAX comes back successfully.

Demo App

You can see the working walkthrough app up to this check point on https://repl.it/@altcademy/To-Do-List-Assignment-Walkthrough-Part-1.

Create A New Task

Cool, we will let the tasks display as is for now. We will come back to add remove and complete button later on. Because we need to work on creating new tasks first, and start using our own api key for our app.

First, let's go to ATDAPI to create a new user ID. Mine is **2**. We will use this ID as our api key from this point forward. You should get your own user ID and use that instead.

When we work on new features, we should attempt it in baby steps. And we should make sure the most crucial part of the new feature is working first. For creating new tasks, obviously, it will be to create a new task by making an AJAX request. Let's do that.

```
copy ×({
  type: 'POST',
  url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=2',
  contentType: 'application/json',
  dataType: 'json',
  data: JSON.stringify({
    task: {
      content: 'Do something fun'
    }
  }),
  success: function (response, textStatus) {
    console.log(response);
  },
  error: function (request, textStatus, errorMessage) {
    console.log(errorMessage);
  }
});
```

We have already gone through how to make AJAX requests so I won't bore you with the details here. If you need a refresher, go back to the previous units. Above is an POST request that will create a new task. The content of this new task is 'Do something fun'. Once the request comes back as successful, the response will be logged in the console.

Check Point

Refresh the page and check the console for the response object of the above POST request.

The response is a task object containing the new task's information. It is just the same as the other tasks we received when we made the GET request to get all tasks. Now that we have the AJAX working, we can go on to implement the HTML and JavaScript that will let our user create a new task on our page. In our HTML, we need

an input element for the content and a button to fire the function that makes the AJAX request. Add the following after the `.todo-list` div.

```
<form id="create-task">
 <input id="new-task-content" type="text" placeholder="new task" required/>
 <button>Create</button>
</form>
```

We wrapped our input and button in a form element so we can make use of the required field on the input element. This makes sure the user cannot submit with empty content.

Next, we need to create a function that will make the POST request, we will call it createTask. It needs to get the value from the HTML input element and add it into the data field of the post request.

```
var createTask = function () {
 $.ajax({
  type: 'POST',
  url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=2',
  contentType: 'application/json',
  dataType: 'json',
  data: JSON.stringify({
   task: {
    content: $('#new-task-content').val()
   }
  }),
  success: function (response, textStatus) {
   console.log(response);
  },
  error: function (request, textStatus, errorMessage) {
   console.log(errorMessage);
  }
 });
}
```

Great, time to link the function to the HTML. When a user clicks the button, a *submit* event is fired on the form element. We will use our createTask function in the callback function when the submit event is fired.

```
$('#create-task').on('submit', function (e) {
 e.preventDefault();
 createTask();
});
```

The default behaviour of when a form fires the submit event, is to reload the page. That's why we need to add the preventDefault function call in the callback function. You can try creating a new task now. The AJAX request will log the response in the console.

Check Point

Make sure the form works and a new task is created. Check the console for the response.

Display The New Task

Awesome, slowly but surely, we are getting there. Now that we can create new tasks on the api server, we should reflect the new state of our tasks on the page as well. There are two ways we can do this.

The first way is to make a GET request to get all the tasks once again and show them on the page. This ensures we are always getting the latest data.

The second way is to use the response we got from the POST request and inject that to the end of the existing tasks. This assumes that our page's data is in sync with the server's data and we are displaying the most up to date data.

Personally, I would go with the former. Because this requires less work on the front end and we can be certain 100% that we are showing the most up to date data. At the cost of making one more AJAX request. And for beginners, this is what I would recommend as you have one less thing to worry about.

To achieve this, we need to abstract the GET request into a function and call it in the success callback of our POST request. We will name this function getAndDisplayAllTasks. And remember to call this function at the end of your JavaScript program so the tasks are displayed on page load.

```
copy cument).ready(function(){
 var getAndDisplayAllTasks = function () {
  $.ajax({
   type: 'GET',
   url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=2',
   dataType: 'json',
   success: function (response, textStatus) {
    response.tasks.forEach(function (task) {
     $('#todo-list').append('<p>' + task.content + '</p>');
    })
   },
   error: function (request, textStatus, errorMessage) {
    console.log(errorMessage);
   }
  });
 }

 var createTask = function () {
  $.ajax({
   type: 'POST',
   url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=2',
   contentType: 'application/json',
   dataType: 'json',
   data: JSON.stringify({
    task: {
     content: $('#new-task-content').val()
    }
   }),
   success: function (response, textStatus) {
    getAndDisplayAllTasks();
   },
   error: function (request, textStatus, errorMessage) {
    console.log(errorMessage);
   }
  });
 }

 $('#create-task').on('submit', function (e) {
  e.preventDefault();
  createTask();
 });

 getAndDisplayAllTasks();

});
```

Now try creating a new task. You will see a bug. After creating a new task, a list of the latest tasks are appended to the end of the existing tasks. They are duplicates.

The solution to this is, we need to clear out all the existing tasks in the HTML before injecting the latest tasks. We can use the jQuery empty method for this. We will use it on the `.todo-list` div.

```
copy etAndDisplayAllTasks = function () {
 $.ajax({
  type: 'GET',
  url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=2',
  dataType: 'json',
  success: function (response, textStatus) {
   $('#todo-list').empty(); // Add this line
   response.tasks.forEach(function (task) {
    $('#todo-list').append('<p>' + task.content + '</p>');
   })
  },
  error: function (request, textStatus, errorMessage) {
   console.log(errorMessage);
  }
 });
}
```

Smooth as silk. One more user experience tweak before we move on. After the POST request is successful, we should clear out the value of the input element so the user can create a new task again without having to delete the existing value. This just makes it nicer for the user.

```
copy reateTask = function () {
 $.ajax({
type: 'POST',
  url: 'https://fewd-todolist-api.onrender.com/tasks?api_key=2',
  contentType: 'application/json',
  dataType: 'json',
  data: JSON.stringify({
   task: {
    content: $('#new-task-content').val()
   }
  }),
  success: function (response, textStatus) {
   $('#new-task-content').val(''); // Add this line
   getAndDisplayAllTasks();
  },
  error: function (request, textStatus, errorMessage) {
   console.log(errorMessage);
  }
 });
}
```

Check Point

You should be able to create a new task and display the latest state of all the tasks in the HTML.

Demo App

You can see the working walkthrough app up to this check point on https://repl.it/@altcademy/To-Do-List-Assignment-Walkthrough-Part-2.

Removing A Task

We only have two more features to add now, removing a task and marking a task as complete or active. We will start with removing a task. As for all things, we should make sure the AJAX request works first.

Pick one of the tasks belonging to your user ID for the following step. You can see all your tasks by visiting this link directly on your browser. https://fewd-todolist-api.onrender.com/tasks?api_key=2. Choose a task and make a note of its id number. The one I choose is 178.

The AJAX request for deleting a task is such.

```
copy ({
 type: 'DELETE',
  url: 'https://fewd-todolist-api.onrender.com/tasks/5?api_key=2',
  success: function (response, textStatus) {
    console.log(response);
  },
  error: function (request, textStatus, errorMessage) {
    console.log(errorMessage);
  }
});
```

Add it to the end of your program and reload the page. The task you choose should be deleted. The response message `{ success: true }` should be printed in the console. Reload the page again and you will see that the task is no longer showing up.

Now that the AJAX is working. we can link it up to the HTML. This one, however, is a bit trickier than the features we did before. Each task should have its own delete button. And each delete button should call a function that will delete the task from the api server the button belongs to. We need to find a way to pass the ID of the task to the delete function.

This can be done by dynamically adding data attribute to the HTML string when we inject the tasks to the DOM. In the forEach function in getAndDisplayAllTasks, modify the following.

```
copy nse.tasks.forEach(function (task) {
  $('#todo-list').append('<div class="row"><p class="col-xs-8">' + task.content + '</p><button class="delete" data-id="' + task.id + '">Delete</button>');
});
```

We have added a button to each task. The button has an data-id attribute that contains the task ID. We also wrapped the task inside a .row div and gave the paragraph element a column size. This should add a bit of structure to our page.

Now we just need to work on the deleteTask function and link it to all the buttons. The deleteTask function will be similar to the addTask button. Inside the deleteTask function, we will make an AJAX request to delete the specific task. Then we will call getAndDisplayAllTasks to render the latest state of the tasks.

```
copy eleteTask = function (id) {
  $.ajax({
 type: 'DELETE',
    url: 'https://fewd-todolist-api.onrender.com/tasks/' + id + '?api_key=2',
    success: function (response, textStatus) {
      getAndDisplayAllTasks();
    },
    error: function (request, textStatus, errorMessage) {
      console.log(errorMessage);
    }
  });
}
```

Finally, we have to link the deleteTask function to all the delete buttons. We added a class `.delete` to all the delete buttons. We will use that as the selector for the click event. Add the following after the deleteTask function.

```
copy ument).on('click', '.delete', function () {
  console.log($(this).data('id'))
});
```

Because our tasks are dynamically added to the DOM, we cannot add event listeners to the delete buttons themselves. Instead we need to add the event listener to the document object and add the CSS selector to target the delete buttons. I deliberately omitted the function call to deleteTask because I want to test that this is working first.

We can get the data-id attribute of the button element by using the jQuery `data()` method. This method needs to be called on the `this` variable because the `this` variable refers to the HTML element that fired the click event, which is the button. Wrapping the `this` variable in a jQuery function turns it into a jQuery object so we can proceed to use the jQuery methods.

Reload the page and click on any of the delete buttons. The ID of the corresponding task should be logged in the console.

After this, we just need to call the deleteTask function in the event listener callback and our tasks will be deleted upon request.

```
copy ument).on('click', '.delete', function () {
  deleteTask($(this).data('id'));
});
```

Check Point

Task should be deleted and removed from the page after clicking the delete button.

Demo App

You can see the working walkthrough app up to this check point on https://repl.it/@altcademy/To-Do-List-Assignment-Walkthrough-Part-3.

Mark A Task As Complete

We are nearing the end of the walkthrough. We have one more feature to punch out. And we will break it into two parts. Let's first add the feature of marking a task as complete. Again, we will start with the AJAX request. The URL to marking a task as complete is `/tasks/:id/mark_complete?api_key=your_user_id`. Pick a random task from your tasks and use its ID for the following PUT request. I am using task with ID 176. Make sure you are using a task belonging to your user ID.

```
copy ‹({
 type: 'PUT',
  url: 'https://fewd-todolist-api.onrender.com/tasks/4/mark_complete?api_key=2',
  dataType: 'json',
  success: function (response, textStatus) {
    console.log(response);
  },
  error: function (request, textStatus, errorMessage) {
    console.log(errorMessage);
  }
});
```

You should get the following response.

```
copy
  task: {
    id: 176,
    content: 'Wash dirty dishes',
    completed: true,
    due: null,
    created_at: '2018-05-04T02:33:15.718Z',
    updated_at: '2018-05-04T04:33:30.051Z'
  }
}
```

Note the completed property is now `true`. Awesome 😎.

The rest of the procedures is similar to adding the remove button. We need to modify the HTML content we inject into the DOM in the getAndDisplayTasks function to add a checkbox to let the user toggle complete and active. Wrap the PUT request in a function. And link the checkbox to the function through an event listener. Here is the code.

In the getAndDisplayAllTasks function, modify the following. This will add a input checkbox to the end of each task. It has the data-id attribute just like the delete button.

```
copy ›nse.tasks.forEach(function (task) {
  $('#todo-list').append('<div class="row"><p class="col-xs-8">' + task.content + '</p><button class="delete" data-id="' + task.id + '">Delete</button><input type="checkbox" class="mark-complete" data-id="' + task.id + '" ' + (task.completed ? 'checked' : '') + '>');
});
```

Add a markTaskComplete function.

```
copy ıarkTaskComplete = function (id) {
  $.ajax({
 type: 'PUT',
    url: 'https://fewd-todolist-api.onrender.com/tasks/' + id + '/mark_complete?api_key=2',
    dataType: 'json',
    success: function (response, textStatus) {
      getAndDisplayAllTasks();
    },
    error: function (request, textStatus, errorMessage) {
      console.log(errorMessage);
    }
  });
}
```

Link the checkbox to the markTaskComplete function. The checkbox will fire a change event whenever a user checks or un-checks the checkbox. We need to listen to the change event and get the checked state of the checkbox. If the checkbox is in a checked state, that means the user wants to mark it as completed. We should call the markTaskComplete function.

```
copy :ument).on('change', '.mark-complete', function () {
  console.log(this.checked);
});
```

We are taking baby steps again. Make sure the above code console logs the checked state of the checkboxes when you check and un-check them. If it is working. We can add the function call to markTaskComplete.

```
copy :ument).on('change', '.mark-complete', function () {
 if (this.checked) {
    markTaskComplete($(this).data('id'));
  }
});
```

Awesome. Reload the page and check away.

The checkbox is awfully close to the delete button. Why don't we give the delete button a small right margin in the CSS.

```
copy :e {
  margin-right: 5px;
}
```

Check Point

Each task has a checkbox that lets the user mark the task as complete. The tasks are reloaded and the latest state of the tasks are displayed.

Mark A Task As Active

For this last part, we just need to create the relevant AJAX request, wrap it in a function, and add it to the callback of the event handler of the checkboxes. It is exactly the same as marking a task as completed. Create the following function named markTaskActive.

```
copy ıarkTaskActive = function (id) {
  $.ajax({
 type: 'PUT',
    url: 'https://fewd-todolist-api.onrender.com/tasks/' + id + '/mark_active?api_key=2',
    dataType: 'json',
    success: function (response, textStatus) {
      getAndDisplayAllTasks();
    },
    error: function (request, textStatus, errorMessage) {
      console.log(errorMessage);
    }
  });
}
```

Call this function in the else statement of the callback function to the change event of the checkboxes.

```
copy cument).on('change', '.mark-complete', function () {
 if (this.checked) {
   markTaskComplete($(this).data('id'));
 } else {
   markTaskActive($(this).data('id'));
 }
});
```

And you're done. If the checkbox changes from un-checked to checked, we will send a request to the server to mark the task as completed. If the checkbox changes from checked to un-checked, we will send a request to the server to mark the task as active.

Check Point

Un-checking a checkbox will mark a task as active and refresh the list of tasks.

Demo App

You can see the working walkthrough app up to this check point on https://repl.it/@altcademy/To-Do-List-Assignment-Walkthrough-Part-4.

We have finished all the basic requirements for this assignment.

1. A list of tasks rendered in the DOM based on data from the ATDAPI server.
2. Each task has a description, a remove button, and a mark complete/active button.
3. An input element and a button that lets user add a new task.

You can beautify the app with some CSS or attempt the bonus feature.

- A toggle to show Active/Complete/All tasks only.

A hint on the bonus feature. Try filtering the list of tasks based on the "completed" property in the success callback of the getAndDisplayAllTasks function. Once you got that working. Create a set of HTML elements that lets the user toggle whether he/she wants to see All/Complete/Active tasks.

Extract the user's filtering preference either directly from the HTML elements or have the toggle elements update a state in your JavaScript program.

Finally, base your filter method in getAndDisplayAllTasks on the toggle state. If the user wants to see All tasks, don't filter out any tasks. If the user only wants to see completed tasks, filter out the active tasks. If the user only wants to see active tasks, future out the completed tasks.

😭😐😑😀😍

Rate this content

MARK AS COMPLETE 🎉

© All rights reserved by Altcademy, Inc.

This material is copyrighted. You are not allowed to upload, post, publish, reproduce, transmit or distribute in any way any component of the website itself or create derivative works with respect thereto, as the website is copyrighted under applicable laws.

Motivational Quote

"Knowledge is power"

– Francis Bacon