## Documentation:

### Goals:

The goals of this document are making an easy to understand ToDo list. With Time and Point keeping functionality. This aids me in calculating my room and board. I also intend for it to be easily added to my portfolio website (it is the link button). I hope to create a modular design so I can add and improve upon it.

### Description:

Upon opening the website you are greeted with a login window. You login with the credentials User: timo, Pass: pi . Once logged in you can click any of the nav orbs at the top. The first in indoor chores, the second is outdoor chores, the third is calculation, and the forth is a link to my hopefully modular website portfolio. When you click on either the indoor or outdoor orbs, you can access the predetermined list of tasks. You then have 3 option buttons: 1st you have the read option which you can view the current data of that task, 2nd you have the update option where you can change the values of the task, 3rd you can delete the task when it is no longer relevant, the 4th you can duplicate the task. When you click on the calculator button, you then have the options to “read” or “Total”. The read button lists all the tasks and the assigned points. The total button displays the total number of points you have. Each time the total, read, update, delete, or duplicate buttons are clicked, you then have the option to close the info bar.

## Rubric:

* Interface is Authentic, Professional, Balanced Interface design is authentic, looks professional, is balanced across the web page
  + This is done through CSS. I copied some of your CSS, and my own from a project in 215. This was done so that I can add this project to my portfolio project with minimal difficulty.
  + \*{color: whitesmoke;} header {width: 100%; height: 20vh; background-color: 24059; color: whitesmoke; display: flex; align-items: center; justify-content: center; flex-direction: column;}
* <link rel="stylesheet" href="To\_Do.css" />
* Interface links to functional jQuery jQuery is included in application source
  + This was done through using script src.
* <script src="./lib/jquery-3.7.1.js"></script>
* <script src="To\_Do\_array.js"></script>
* <script src="To\_Do\_Code.js"></script>
* Events are Effective, jQuery jQuery is used to bind events to event handlers<br/> The event handling mechanism works to achieve goals.
  + I used on click events to define each button pressed.
* $(".displayLine button").on("click", (e) => {
* const target = $(e.target);
* handler(target[0]);
  + Below is the main event handler
* const handler = (element) => {
* let elementClass = element.className;
* let indexOftodo = $(`button.${elementClass}`).index(element);
* if (elementClass == "view") {
* console.log("View called");
* viewHandler(indexOftodo);
* } else if ((elementClass == "update")) {
* console.log("update called");
* updateHandler(indexOftodo);
* } else if (elementClass == "delete") {
* console.log("removed called");
* removeHandler(indexOftodo);
* }else if (elementClass == "duplicate") {
* console.log("duplicate called");
* duplicateHandler(indexOftodo);
* }
* else if (elementClass == "Alltotals") {
* console.log("All Totals called");
* AllTotalsHandler(indexOftodo);
* }
* else if (elementClass == "Finaltotal") {
* console.log("Final Total called");
* FinalTotalHandler(indexOftodo);
* }
* };
* DOM Access, jQuery The DOM is accessed with jQuery
  + Changed elements on the web page, used so often should not even be mentioned.
* let info = $(`#ShowInfo${index}`); // Target the specific ShowInfo box
* info.html(`
* Task name: <input id="task\_name" value="${todo\_list[index].task}">
* Time Estimation: <input id="Time\_Est" value="${todo\_list[index].Time\_Est}">
* Time Taken: <input id="Time\_Took" value="${todo\_list[index].Time\_Took}">
* Completed:
* <select id="Completed">
* <option value="true" ${todo\_list[index].Completed ? 'selected' : ''}>True</option>
* <option value="false" ${!todo\_list[index].Completed ? 'selected' : ''}>False</option>
* </select>
* Points given: <input id="Points" value="${todo\_list[index].Points}">
* <button class="save">Save</button>
* `);
* DOM Dynamically Built, jQuery The DOM is dynamically updated using jQuery calls
  + Is done to refresh the page, also done very often.
* info.find(".save").on("click", () => {
* console.log("Save button clicked");
* // Update the todo\_list with new values
* todo\_list[index].task = $("#task\_name").val();
* todo\_list[index].Time\_Est = $("#Time\_Est").val();
* todo\_list[index].Time\_Took = $("#Time\_Took").val();
* todo\_list[index].Completed = $("#Completed").val() === 'true';
* todo\_list[index].Points = $("#Points").val();
* console.log("Updated todo\_list:", todo\_list[index]);
* // Clear the info div after saving
* setTimeout(function () {
* info.html("");
* }, 1000);
* createPanel({ target: { id: "Inside" } }); // Refresh the panel
* });
* Documentation: Program Code and Project Description .js file is fully annotated, explaining the code, not just stating that a call is being used. <br/> There is a document describing how the application works and how to use it, and the goal of the applicant
  + This is done through out the program. I probably could have done more of it.
* // Get the todo lists from both APIs and concatenate them
* let insideList = ToDoInsideAPI();
* let outsideList = ToDoOutsideAPI();
* todo\_list = [...insideList, ...outsideList]; // Combine both lists
* // Iterate through the todo\_list and display each task with its points
* todo\_list.forEach((todo) => {
* info.append(`
* <p>${todo.task}: ${todo.Points} points</p>
* `);
* });

* Style, jQuery In at least one instance, style is modified or applied using jQuery
  + This is done throughout the program. For me it is mostly used when there is text to add.
* info.append(`
* <p><strong><em>Total Points: ${totalPoints}</em></strong></p>
* `);

## Links:

Pages: <https://tmatis.github.io/jQueryInterests/>

Repo: <https://github.com/TMatis/jQueryInterests>