**A PROJECT REPORT**

**ON**

**INTERNSHIP PROJECT 2**

**CREATE A TO-DO LIST USING HTML PROGRAMMING LANGUAGE**

**IN THE DOMAIN OF**

**FRONT-END WEB DEVELOPMENT**

**BY**

**MALLAVARAPU LAKSHMI PRAHARSHA**

**DATE:19-04-2024**

**TABLE OF CONTENTS**

**S. No Topic Page no.**

1. Abstract 3

2. Objectives 4-6

3. Introduction 7

4. Methodology 8-12

5. Code 13-19

6. Results and Outputs 20-25

7. Conclusion 26

**ABSTRACT**

This abstract outlines the creation of a comprehensive To-Do list utilizing HTML as the front-end language and JavaScript with jQuery as the backend. The To-Do list aims to provide users with a seamless experience for managing tasks, including functionalities to add new tasks and edit existing ones. In the front end, HTML will structure the interface, offering an intuitive layout for users to interact with. Utilizing JavaScript with jQuery as the backend enhances the list's functionality, enabling dynamic features such as task addition and editing. Users can easily input new tasks through an interactive interface, with JavaScript handling the logic behind storing and displaying these tasks. Furthermore, the integration of jQuery streamlines the process, allowing for efficient manipulation of DOM elements and enhancing user interactivity. Overall, the combination of HTML, JavaScript, and jQuery offers a robust solution for creating a user-friendly To-Do list with advanced functionalities, empowering users to manage their tasks effectively.

**OBJECTIVE**

The following objectives are taken into account while Creating a To Do list using the HTML Programming Language:

1. User-Friendly Interface
2. Accessibility
3. Responsive Design
4. Clear Task Representation
5. Task Management Functionalities
6. Data Persistence

**User-Friendly Interface:**

The foremost objective in designing a To-Do list using HTML programming language is to craft a user-friendly interface that fosters intuitive interaction and effortless navigation. Users should seamlessly engage with the list, effortlessly adding, editing, and deleting tasks with minimal friction. Achieving this entails employing clear visual cues, intuitive controls, and logical layout organization to enhance usability. By prioritizing simplicity and efficiency in design, users can effectively manage their tasks without unnecessary cognitive load, resulting in a more satisfying and productive user experience.

**Accessibility:**

An essential objective when creating a To-Do list using HTML programming language is to prioritize accessibility, ensuring that all users, including those with disabilities, can effectively engage with the interface. This involves employing semantic HTML elements to structure content in a meaningful way, aiding screen readers in accurately interpreting and conveying information to users. Additionally, providing alternative text for non-text elements, such as images or icons, enables users relying on assistive technologies to comprehend and navigate the interface seamlessly. By adhering to accessibility best practices, such as implementing keyboard navigation and ensuring sufficient color contrast, the To-Do list becomes inclusive, empowering all users to manage their tasks independently and efficiently. This commitment to accessibility not only aligns with ethical considerations but also enhances the usability and reach of the To-Do list, fostering a more inclusive digital experience for all individuals.

**Responsive Design:**

A crucial objective in crafting a To-Do list using HTML programming language is to implement responsive design principles, enabling the interface to adapt seamlessly to diverse screen sizes and devices. By prioritizing responsiveness, users can access and manage their tasks effortlessly across various platforms, including desktop computers, tablets, and smartphones. This entails utilizing fluid layouts, flexible grids, and media queries to dynamically adjust the layout and content presentation based on the device's screen dimensions. Additionally, optimizing touch-friendly interactions and ensuring readability on smaller screens enhances the user experience on mobile devices. By embracing responsive design, the To-Do list becomes accessible and user-friendly across a wide range of devices, empowering users to stay organized and productive regardless of their preferred device or screen size.

**Clear Task Representation:**

A fundamental objective in designing a To-Do list using HTML programming language is to ensure clear task representation, enhancing user comprehension and organization. Each task should be distinctly presented, incorporating essential details such as task name, due date (where applicable), and priority level to facilitate effective task management. Employing styling techniques to differentiate between completed and pending tasks enhances visual clarity, aiding users in swiftly identifying their current progress and remaining obligations. Utilizing contrasting colors, icons, or text formatting for completed tasks versus pending ones can effectively communicate task status at a glance. Additionally, employing intuitive visual cues, such as checkboxes or strike-through text, or removing if the task is completed further enhances user understanding and engagement. By prioritizing clear task representation, the To-Do list optimizes user productivity and satisfaction, fostering a streamlined and efficient task management experience.

**Task Management Functionalities:**

A pivotal objective in developing a To-Do list using HTML programming language is to furnish essential task management functionalities that facilitate seamless organization and productivity. This encompasses incorporating features such as adding new tasks, marking tasks as complete, editing existing tasks, and deleting tasks, ensuring they are easily accessible and intuitively usable. The addition of new tasks should be a straightforward process, allowing users to input task details effortlessly through a clear and intuitive interface. Once tasks are added, users should have the ability to mark them as complete, providing a sense of accomplishment and progress tracking. Editing existing tasks should be intuitive, enabling users to modify task names, due dates, priorities, or any other relevant details with ease.

Furthermore, the deletion of tasks should be straightforward, allowing users to remove tasks they no longer need or want to track. These functionalities should be readily accessible through intuitive controls, such as buttons or context menus, ensuring a frictionless user experience. By prioritizing these task management functionalities and ensuring they are easily accessible and intuitive to use, the To-Do list empowers users to efficiently organize and manage their tasks, ultimately enhancing productivity and satisfaction.

**Data Persistence:**

A critical objective in developing a To-Do list using HTML programming language is to implement robust data persistence mechanisms, ensuring that users' tasks are reliably saved and accessible across sessions. This involves implementing a mechanism to store task data securely, so users can seamlessly resume task management activities even after refreshing the page or navigating away from it. One approach to achieve data persistence is by leveraging local storage capabilities provided by modern web browsers. Using local storage allows task data to be stored directly on the user's device, enabling quick and efficient access without the need for server communication. This method is well-suited for smaller-scale applications and provides a seamless user experience, particularly for single-user scenarios. By implementing robust data persistence mechanisms, such as local storage or backend server integration, the To-Do list ensures that users' tasks are securely saved and accessible across sessions, ultimately enhancing the reliability and usability of the application.

**INTRODUCTION**

Creating a To-Do list using a combination of HTML, CSS, JavaScript (JS), jQuery, and Bootstrap offers a versatile and powerful solution for organizing tasks effectively. This project combines the markup and styling capabilities of HTML and CSS with the dynamic behavior and interactivity provided by JavaScript and jQuery. Additionally, leveraging Bootstrap allows for rapid prototyping and responsive design implementation, ensuring compatibility across various devices and screen sizes.

HTML forms the foundation of the To-Do list, providing the structure and layout of the user interface. It defines the elements such as input fields, buttons, and containers needed to input, display, and manipulate tasks. CSS stylesheets complement HTML by enhancing the visual presentation, including typography, colors, layout, and responsiveness. With CSS, we can create visually appealing designs and ensure a consistent user experience across different platforms.

JavaScript adds functionality to the To-Do list, enabling dynamic interactions such as adding new tasks, marking tasks as complete, and editing or deleting tasks. By utilizing JavaScript, users can interact with the To-Do list in real time, making it more engaging and efficient. Integrating jQuery into our project simplifies DOM manipulation and event handling, streamlining development and enhancing code efficiency. By leveraging jQuery's extensive library of functions and plugins, we can expedite the implementation of interactive features and animations, enriching the user interface and overall user experience.

Bootstrap, a popular front-end framework, further enhances the project by providing pre-designed components and utilities for building responsive and mobile-friendly web applications. With Bootstrap, developers can easily create responsive layouts, navigation bars, buttons, and other UI elements, saving time and effort in the design process.

In this project, we will explore how to integrate these technologies to create a feature-rich and user-friendly To-Do list application. By leveraging the strengths of HTML, CSS, JavaScript, jQuery, and Bootstrap, we aim to deliver a seamless and intuitive task management experience for users across different devices and platforms. Let's dive into the implementation details and create a robust To-Do list that meets the needs of modern users.

**METHODOLOGY**

In front-end web development, methodology refers to the systematic approach or framework used to plan, design, and build the user interface and user experience of a website or web application. It encompasses various processes, techniques, and best practices aimed at ensuring efficient development, optimal performance, and user satisfaction.

When creating a To-Do list using HTML, CSS, JavaScript (JS), jQuery, and Bootstrap, it's essential to adopt a structured methodology to ensure efficiency, maintainability, and user satisfaction throughout the development process. Here's a detailed overview of the methodology to be considered:

**Requirement Analysis:**

Requirement Analysis is the foundational stage of developing a To-Do list using HTML, CSS, JavaScript, jQuery, and Bootstrap, where the focus is on comprehensively understanding the project's goals and user needs. Firstly, it involves defining the core features and functionalities that the To-Do list should encompass. These typically include task addition, editing, deletion, and ensuring responsiveness across various devices to accommodate a diverse user base. Furthermore, it's crucial to delve into user personas to gain insights into the specific needs and preferences of the target audience. Understanding the demographics, behavior patterns, and pain points of users enables developers to tailor the To-Do list application accordingly, ensuring it aligns with users' expectations and enhances their productivity. For instance, personas may range from busy professionals needing quick task management to students requiring detailed organization features. By conducting thorough requirement analysis and identifying user personas, developers can lay a solid foundation for the To-Do list project, guiding subsequent design and development decisions. This approach ensures that the resulting application not only meets functional requirements but also addresses the unique needs and preferences of its intended users, ultimately leading to a more satisfying and effective user experience.

**Wireframing and Prototyping:**

In the Wireframing and Prototyping phase of developing a To-Do list using HTML, CSS, JavaScript, jQuery, and Bootstrap, the focus is on creating visual representations of the user interface and user experience. Wireframes serve as skeletal outlines, depicting the layout and structure of the To-Do list without detailed design elements. They help stakeholders and developers visualize the hierarchy of content, placement of elements, and flow of interactions.

Prototypes, on the other hand, are interactive representations of the To-Do list, allowing users to simulate interactions and navigate through different screens or states. They provide a more realistic preview of how the final product will look and behave, enabling stakeholders to provide feedback and make informed decisions before proceeding with development. Tools like Adobe XD, Sketch, Figma, or even traditional methods like pen and paper or digital painting software can be used to create wireframes and prototypes. These tools offer features for designing layouts, adding interactive elements, and testing usability. During this phase, it's essential to consider the user experience and ensure that the To-Do list is intuitive and easy to use. Elements such as task input fields, buttons for adding, editing, and deleting tasks, as well as feedback mechanisms for indicating task status, should be incorporated into the wireframes and prototypes. By creating wireframes and prototypes, developers can iteratively refine the design of the To-Do list, gathering feedback from stakeholders and users to ensure alignment with project goals and user needs. This approach minimizes the risk of misunderstandings and reduces rework during the development phase, ultimately leading to a more successful and user-friendly product.

**Responsive Design Planning:**

In the Responsive Design Planning phase of developing a To-Do list using HTML, CSS, JavaScript, jQuery, and Bootstrap, the primary objective is to ensure that the application is accessible and user-friendly across a wide range of devices and screen sizes. Responsive design is essential for providing a consistent and optimized user experience regardless of the device being used, whether it's a desktop computer, tablet, or smartphone. To achieve responsive design, Bootstrap's grid system and responsive utility classes are invaluable tools. Bootstrap offers a flexible and powerful grid system based on a 12-column layout, allowing developers to create responsive layouts that adapt seamlessly to different screen sizes. By utilizing grid classes such as `col-md`, `col-sm`, and `col-xs`, developers can define the width of content columns relative to the screen width, ensuring that content is appropriately sized and positioned across devices.

Additionally, Bootstrap provides responsive utility classes that enable developers to show or hide content based on screen size, adjust spacing and alignment, and control the visibility of elements. These utility classes, such as `d-none`, `d-md-block`, `ml-auto`, and `mr-auto`, empower developers to fine-tune the layout and appearance of the To-Do list for optimal responsiveness. During the planning phase, it's essential to consider the specific breakpoints at which the layout should adapt to different screen sizes. This involves identifying key breakpoints based on common device widths and defining the layout behavior at each breakpoint. By strategically planning the responsive design strategy and leveraging Bootstrap's grid system and utility classes, developers can create a fluid and adaptable layout for the To-Do list that ensures a seamless user experience across devices. This approach enhances accessibility and usability, allowing users to interact with the To-Do list effectively regardless of the device they're using.

**HTML Structure:**

In the HTML Structure phase of developing a To-Do list using HTML, CSS, JavaScript, jQuery, and Bootstrap, the primary goal is to construct a well-organized and semantically meaningful HTML structure based on the wireframes and prototypes created earlier. Semantic HTML elements are utilized to enhance clarity and accessibility, ensuring that the To-Do list is easily navigable and understandable by both users and assistive technologies.

The To-Do list HTML structure typically comprises several key sections, each serving a distinct purpose in facilitating task management:

**1.** **Task Input Section:** This section includes input fields and buttons for users to add new tasks to the list. Semantic HTML elements such as `<form>`, `<input>`, and `<button>` are utilized to create a structured and accessible input interface. Additionally, labels and placeholders provide descriptive cues for users, enhancing usability and clarity.

**2.** **Task Display Section:** Here, the existing tasks in the To-Do list are displayed, typically in the form of a list or grid. Semantic elements like `<ul>` and `<li>` are employed to structure the list of tasks, with each task represented as an individual list item. Clear and concise task descriptions, along with any associated metadata such as due dates or priority levels, are included to provide context and aid comprehension.

**3.** **Task Controls and Modals:** This section houses controls and buttons for performing actions on tasks, such as marking tasks as complete, editing or deleting tasks, and managing task details. Semantic elements such as `<button>` and `<div>` are used to create interactive controls, while modal dialogs may be employed for editing task details or confirming actions. These elements are designed to be accessible and intuitive, facilitating seamless task management for users.

By structuring the HTML of the To-Do list in a logical and semantically meaningful manner, developers can enhance clarity, accessibility, and usability for all users. Semantic HTML elements provide valuable context and organization, while clear task descriptions and intuitive controls streamline the task management experience. This approach ensures that the To-Do list is not only functional but also user-friendly and accessible across a variety of devices and platforms.

**CSS Styling:**

In the CSS Styling phase of developing a To-Do list using HTML, CSS, JavaScript, jQuery, and Bootstrap, the objective is to enhance the visual appearance and usability of the application while maintaining consistency and alignment with the project's branding and requirements. Bootstrap's predefined styles and components serve as a valuable starting point for achieving a modern and consistent design. By leveraging Bootstrap's extensive CSS framework, developers can easily apply styles to various elements of the To-Do list, ensuring a cohesive and professional appearance across different sections and components. Bootstrap's utility classes, such as spacing, typography, and color classes, offer flexibility and efficiency in styling, enabling developers to achieve desired visual effects with minimal effort. However, customization may be necessary to tailor the styling of the To-Do list to specific project requirements and branding guidelines. This involves overriding Bootstrap's default styles and adding custom CSS rules to achieve the desired look and feel. Customization may include adjustments to colors, fonts, spacing, borders, and other visual properties to create a unique and branded experience for users.

Furthermore, attention should be paid to responsive design considerations during the CSS styling phase. Bootstrap's responsive classes and breakpoints can be utilized to ensure that the To-Do list adapts gracefully to different screen sizes and devices, providing an optimal viewing and interaction experience for users on desktops, tablets, and smartphones. By applying CSS styles thoughtfully and leveraging Bootstrap's predefined styles and components, developers can enhance the visual appeal and usability of the To-Do list while maintaining consistency and alignment with project requirements. Customization allows developers to tailor the styling to match the project's branding and achieve a unique and branded experience for users. Additionally, responsive design considerations ensure that the To-Do list remains accessible and user-friendly across a variety of devices and screen sizes.

**JavaScript Functionality:**

In the JavaScript Functionality phase of developing a To-Do list using HTML, CSS, JavaScript, jQuery, and Bootstrap, the primary objective is to implement dynamic interactions and task management features to enhance the usability and functionality of the application. This involves utilizing JavaScript or jQuery to handle user input, manipulate tasks, and manage events, ensuring a seamless and responsive user experience. Vanilla JavaScript offers the flexibility and power to create interactive features from scratch, while jQuery provides a simplified syntax and cross-browser compatibility, making it easier to manipulate the DOM and handle events. Developers can choose the appropriate approach based on project requirements and familiarity with the respective technologies.

The JavaScript functionality typically includes:

**1.** **Task Addition:** Implement functionality to add new tasks to the To-Do list when users submit the task input form. This involves capturing user input, creating new task elements dynamically, and appending them to the task display section.

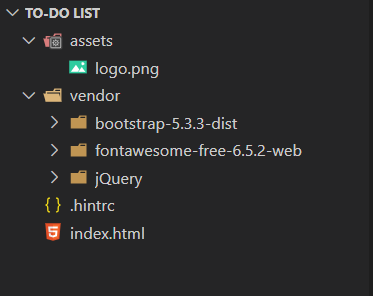
**2.** **Task Manipulation:** Enable users to perform actions on tasks, such as marking tasks as complete, editing task details, or deleting tasks. This involves adding event listeners to task elements, responding to user interactions, and updating the task list accordingly.

**3.** **Error Handling and Validation:** Implement robust error handling and validation mechanisms to maintain data integrity and prevent unexpected behavior. This includes validating user input to ensure it meets specified criteria, handling edge cases gracefully, and providing informative error messages to users when necessary.

**4.** **Event Handling:** Manage user interactions and respond to events such as clicks, keypresses, or form submissions. This involves attaching event listeners to relevant DOM elements, handling events with appropriate callback functions, and updating the UI in response to user actions.

By implementing JavaScript functionality effectively, developers can create a dynamic and interactive To-Do list that enhances user engagement and productivity. Whether using vanilla JavaScript or jQuery, attention to detail in error handling, validation, and event handling ensures a robust and reliable user experience. This approach empowers users to manage their tasks efficiently while maintaining data integrity and usability standards.

**File Structure:**

****

**CODE**

**index.html:**

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="UTF-8">

        <meta name="viewport" content="width=device-width, initial-scale=1.0">

        <title>Major Project | TODO List</title>

        <link href="vendor/bootstrap-5.3.3-dist/css/bootstrap.css" rel="stylesheet">

        <link href="vendor/fontawesome-free-6.5.2-web/css/all.css" rel="stylesheet">

        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">

    </head>

    <body style="background-color: #FBFCFC;">

        <nav class="navbar navbar-expand-lg navbar-light" style="background-color: #FEF9E7;">

            <div class="container-fluid">

                <a href="#" class="navbar-brand">

                    <img src="assets/logo.png" class="img-fluid" alt="logo" width="120"  height="40" style="margin: 10%;">

                </a>

                <h1>TASK TRACKER: STAY ORGANIZED WITH OUR TO-DO LIST</h1>

                <button type="button" class="navbar-toggler" data-bs-toggle="collapse" data-bs-target="#navbar">

                    <i class="bi bi-list"></i>

                </button>

                <div class="collapse navbar-collapse" id="navbar">

                    <div class="navbar-nav ms-auto">

                    </div>

                </div>

            </div>

        </nav><br><br>

        <div class="container p-5" style="background-color: #FEF5E7; border-radius: 10px;">

            <div class="mb-3">

                <button type="button" class="btn btn-outline-primary" onclick="showAddTaskModal()">Add Task</button>

            </div>

            <div class="d-flex justify-content-center">

                <div class="col-sm-12 col-md-12 col-lg-12">

                    <div class="card">

                        <div class="card-body">

                            <table class="table">

                                <thead class="text-center">

                                    <th>#</th>

                                    <th>Task/Description</th>

                                    <th>Responsible</th>

                                    <th>ETA(Time Zone)</th>

                                    <th>Action</th>

                                </thead>

                                <tbody class="text-center" id="taskTableBody">

                                </tbody>

                            </table>

                        </div>

                    </div>

                </div>

            </div>

        </div>

        <div class="modal fade" id="addTaskModal" data-bs-backdrop="static" dats-bs-keyboard="false" tabindex="-1" aria-labelledby="addTaskModalLabel" aria-hidden="true">

            <form id="taskInputForm">

                <div class="modal-dialog">

                    <div class="modal-content">

                        <div class="modal-header">

                            <h5 class="modal-title" id="addTaskModalLabel">Add Task</h5>

                            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

                        </div>

                        <div class="modal-body">

                            <div class="mb-1">

                                <label for="addTaskTextArea" class="form-label">Task/Description</label>

                                <textarea class="form-control" id="addTaskTextArea" name="taskDescription" rows="3" placeholder="Add Your Task or Description"></textarea>

                            </div>

                            <div class="mb-1">

                                <label for="addTaskResponsible" class="form-label">Responsible</label>

                                <input type="text" class="form-control" id="addResponsiblePerson" name="taskResponsiblePerson" rows="3" placeholder="Add the Responsible Person's Name">

                            </div>

                            <div class="mb-1">

                                <label for="addTaskResponsible" class="form-label">ETA</label>

                                <input type="datetime-local" class="form-control" id="addETA" name="taskETA" rows="3" placeholder="Click to add time">

                            </div>

                        </div>

                        <div class="modal-footer">

                            <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>

                            <button type="button" class="btn btn-primary" onclick="addTask()">Add Task</button>

                        </div>

                    </div>

                </div>

            </form>

        </div>

        <div class="modal fade" id="updateTaskModal" data-bs-backdrop="static" dats-bs-keyboard="false" tabindex="-1" aria-labelledby="updateTaskModalLabel" aria-hidden="true">

            <form id="taskUpdateForm">

                <div class="modal-dialog">

                    <div class="modal-content">

                        <div class="modal-header">

                            <h5 class="modal-title" id="editTaskModalLabel">Edit Task</h5>

                            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

                        </div>

                        <div class="modal-body">

                            <div class="mb-1">

                                <label for="editTaskTextArea" class="form-label">Task/Description</label>

                                <textarea class="form-control" id="editTaskTextArea" name="taskDescription" rows="3" placeholder="Add Your Edited Task or Description"></textarea>

                            </div>

                            <div class="mb-1">

                                <label for="addTaskResponsible" class="form-label">Responsible</label>

                                <input type="text" class="form-control" id="editResponsiblePerson" name="taskResponsiblePerson" rows="3" placeholder="Edit to add the Responsible Person's Name">

                            </div>

                            <div class="mb-1">

                                <label for="addTaskResponsible" class="form-label">ETA</label>

                                <input type="datetime-local" class="form-control" id="editETA" name="taskETA" rows="3" placeholder="Click to edit time">

                            </div>

                            <input type="hidden" id="editIndex" name="taskIndex">

                        </div>

                        <div class="modal-footer">

                            <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>

                            <button type="button" class="btn btn-primary" onclick="updateTask()">Add Task</button>

                        </div>

                    </div>

                </div>

            </form>

        </div>

        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>

        <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js" integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM8ODewa9r" crossorigin="anonymous"></script>

        <script src="vendor/bootstrap-5.3.3-dist/js/bootstrap.js"></script>

        <script>

            createHtmlfromStorage();

            function showAddTaskModal(){

                $("#addTaskModal").modal('show');

            }

            function addTask(){

                console.log('Add Task Clicked');

                $("#addTaskModal").modal('hide');

                var dataArr = $("#taskInputForm").serializeArray();

                var taskObject = new Object();

                var storageObjectArr = [];

                var storageObject = localStorage.getItem('taskStorage');

                for(var i in dataArr){

                    var name = dataArr[i]['name'];

                    var value = dataArr[i]['value'];

                    taskObject[name] = value;

                }

                /\* Convert the object into JSON and store it in the Local storage. If there are existing tasks, push the new one to them \*/

                if(storageObject != null && storageObject != undefined && storageObject != '') {

                    storageObjectArr = JSON.parse(storageObject);

                    storageObjectArr.push(taskObject);

                }else{

                    storageObjectArr.push(taskObject);

                }

                localStorage.setItem('taskStorage', JSON.stringify(storageObjectArr));

                createHtmlfromStorage()

                $("#taskInputForm").trigger('reset');

            }

            function createHtmlfromStorage(){

                var storageObjectArr = [];

                var storageObject = localStorage.getItem('taskStorage');

                var storageObjectArr = JSON.parse(storageObject);

                var html = '';

                console.log(storageObject);

                if(storageObject != null && storageObject != undefined && storageObject != '') {

                    if(storageObjectArr && storageObjectArr.length > 0) {

                        for(let i in  storageObjectArr){

                            var date = new Date(storageObjectArr[i]['taskETA'])

                            html = html + '<tr>'

                                        +'<td>'+ (parseInt(i)+1) +'</td>'

                                        +'<td>'+ storageObjectArr[i]['taskDescription'] +'</td>'

                                        +'<td>'+ storageObjectArr[i]['taskResponsiblePerson'] +'</td>'

                                        +'<td>'+ date.toUTCString() +'</td>'

                                        +'<td><i class="bi bi-check-circle-fill" onclick="markAsDone('+i+')"></i><i class="bi bi-pencil-square" onclick="editTask('+i+')"></i></td></tr>'

                        }

                    }else{

                        html =  '<tr><td colspan="5">No task added yet to display!</td></tr>';

                    }

                }

                $("#taskTableBody").html(html);

            }

            function markAsDone(index){

                var storageObjectArr = [];

                var storageObject = localStorage.getItem('taskStorage');

                if(storageObject != null && storageObject != undefined && storageObject != ''){

                    storageObjectArr = JSON.parse(storageObject);

                    storageObjectArr.pop(index);

                }

                localStorage.setItem('taskStorage',JSON.stringify(storageObjectArr));

                createHtmlfromStorage();

            }

            function editTask(index){

                var storageObjectArr = [];

                var storageObject = localStorage.getItem('taskStorage');

                if(storageObject != null && storageObject != undefined && storageObject != ''){

                    storageObjectArr = JSON.parse(storageObject);

                    $("#editTaskTextArea").val(storageObjectArr[index]['taskDescription']) ;

                    $("#editResponsiblePerson").val(storageObjectArr[index]['taskResponsiblePerson']) ;

                    $("#editETA").val(storageObjectArr[index]['taskETA']) ;

                    $("#editIndex").val(index) ;

                    $("#updateTaskModal").modal('show');

                }

            }

            function updateTask(){

                $("#updateTaskModal").modal('hide');

                var dataArr = $("#taskUpdateForm").serializeArray();

                var taskObject = new Object();

                var storageObjectArr = [];

                var storageObject = localStorage.getItem('taskStorage');

                for(var i in dataArr){

                    var name = dataArr[i]['name'];

                    var value = dataArr[i]['value'];

                    taskObject[name] = value;

                }

                /\* Convert the object into JSON and store it in the Local storage. If there are existing tasks, push the new one to them \*/

                if(storageObject != null && storageObject != undefined && storageObject != '') {

                    storageObjectArr = JSON.parse(storageObject);

                    storageObjectArr[taskObject['taskIndex']] = taskObject;

                }

                localStorage.setItem('taskStorage', JSON.stringify(storageObjectArr));

                createHtmlfromStorage()

            }

        </script>

    </body>

</html>

The provided code is an HTML file with embedded JavaScript and CSS styles. It is a simple to-do list application that allows users to add, edit, and mark tasks as done. The application uses the browser's local storage to store the tasks.

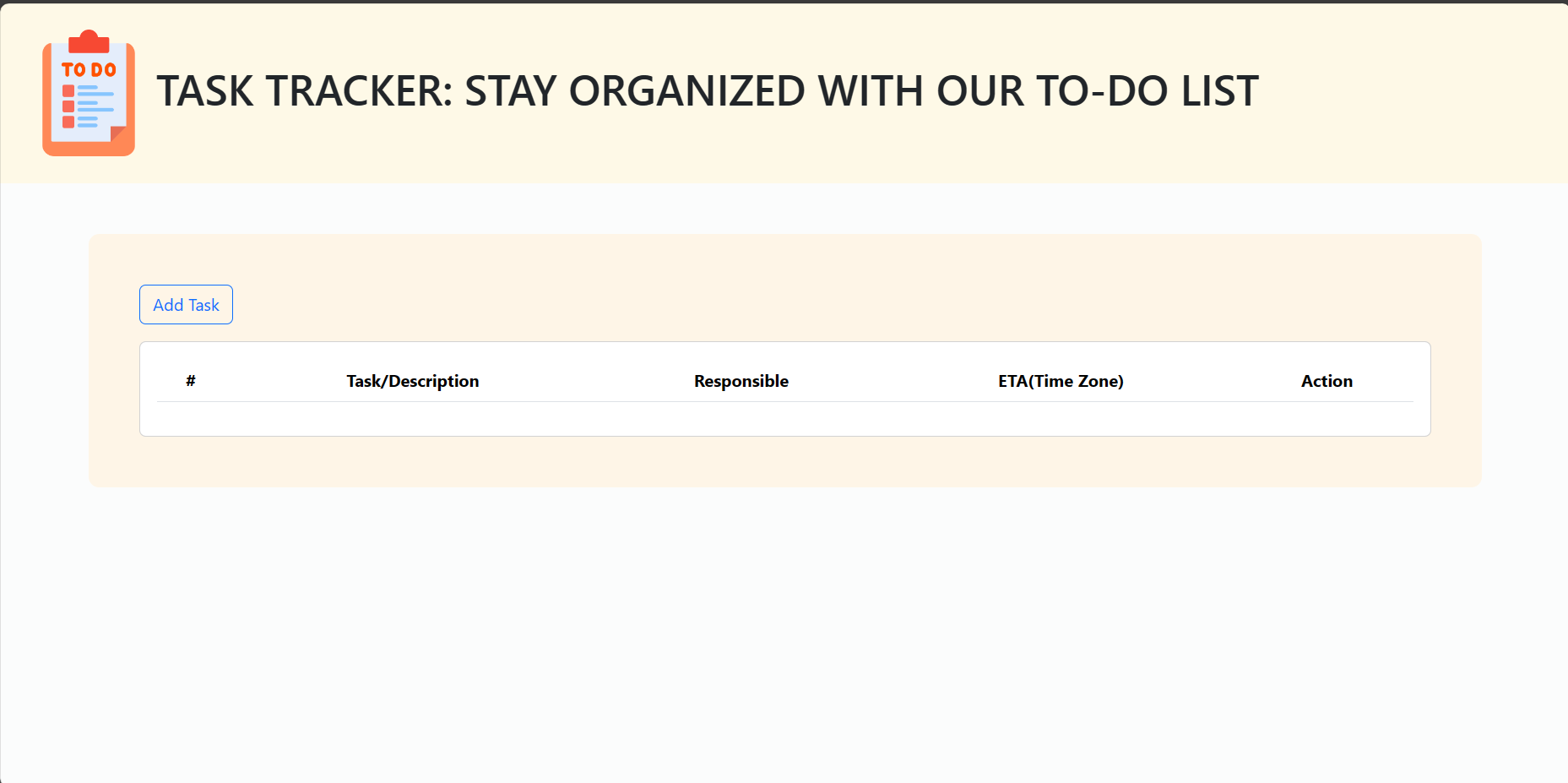
The HTML structure includes a navigation bar, a container for the to-do list, and two modals for adding and editing tasks. The navigation bar contains a logo and a title, and the to-do list container includes a table that displays the tasks. The add task modal contains input fields for the task description, responsible person, and estimated time of completion (ETA). The edit task modal contains the same input fields as the add task modal, as well as a hidden input field for the task index.

The JavaScript code handles the functionality of the application. It includes functions for creating HTML from the local storage, showing the add task modal, adding a task, creating HTML from the local storage, marking a task as done, editing a task, and updating a task. The createHtmlfromStorage function retrieves the tasks from the local storage and generates the HTML for the to-do list table. The showAddTaskModal function displays the add task modal, and the addTask function adds a new task to the local storage and updates the to-do list table. The markAsDone function removes a task from the local storage and updates the to-do list table, and the editTask function displays the edit task modal with the selected task's details. The updateTask function updates the selected task in the local storage and updates the to-do list table.

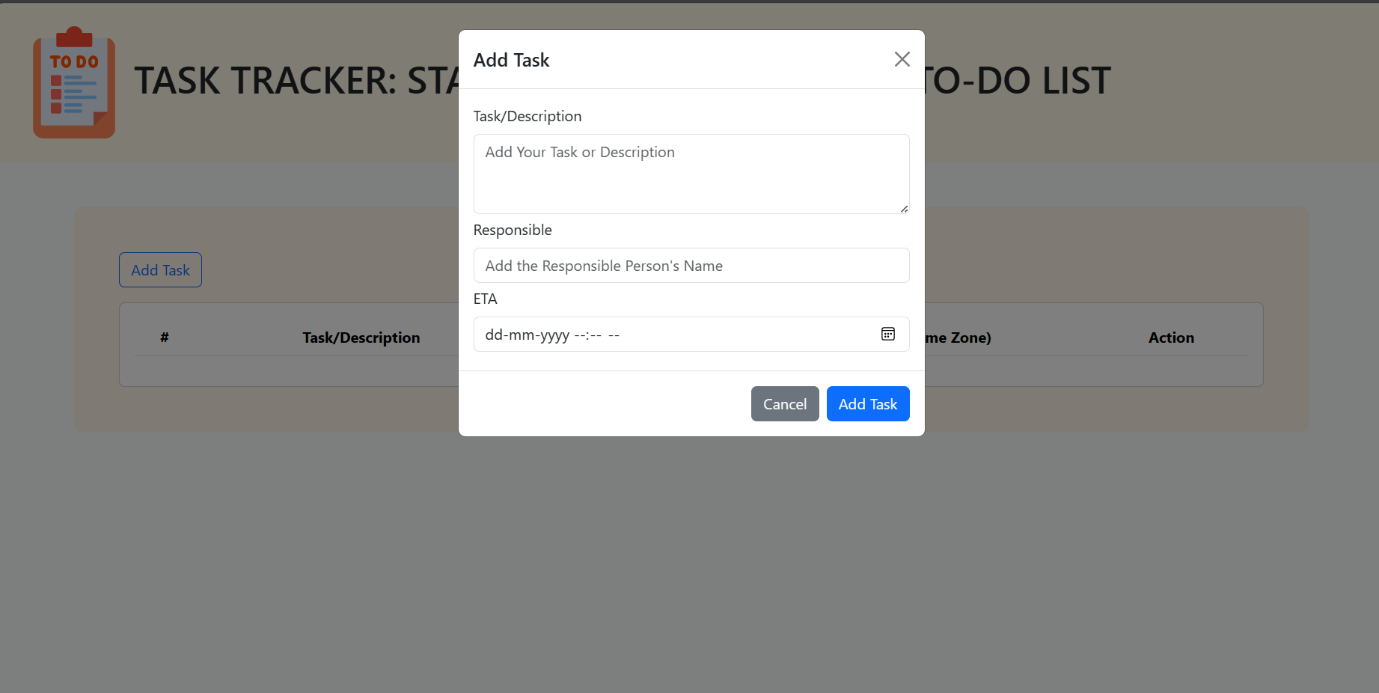
The CSS styles include styles for the navigation bar, to-do list container, and modals. The navigation bar has a light yellow background color, and the to-do list container has a light orange background color. The modals have a white background color and a shadow effect. The table in the to-do list container has a striped background color for alternate rows.

**RESULTS AND OUTPUTS**

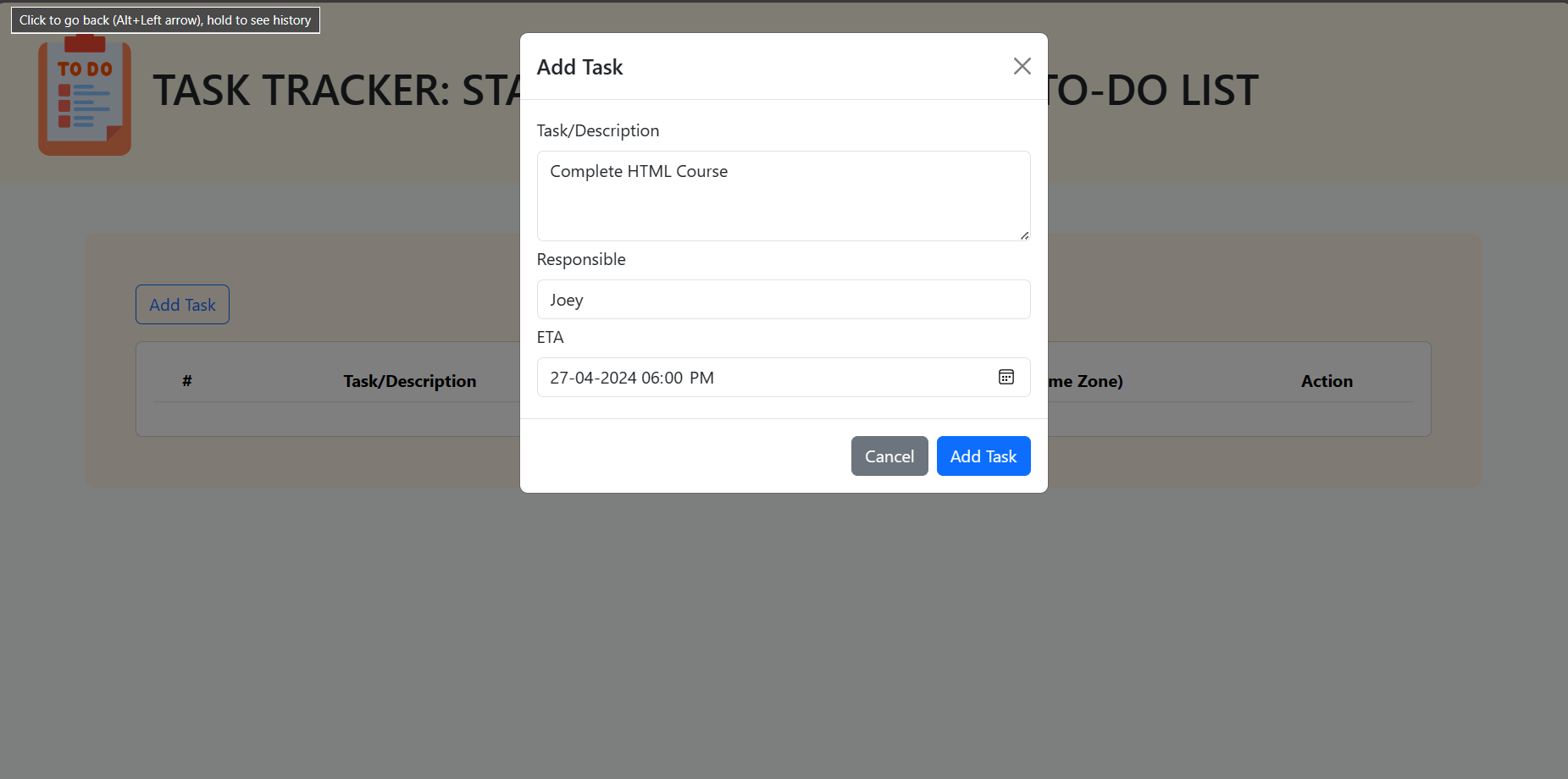
**HOME PAGE:**

****

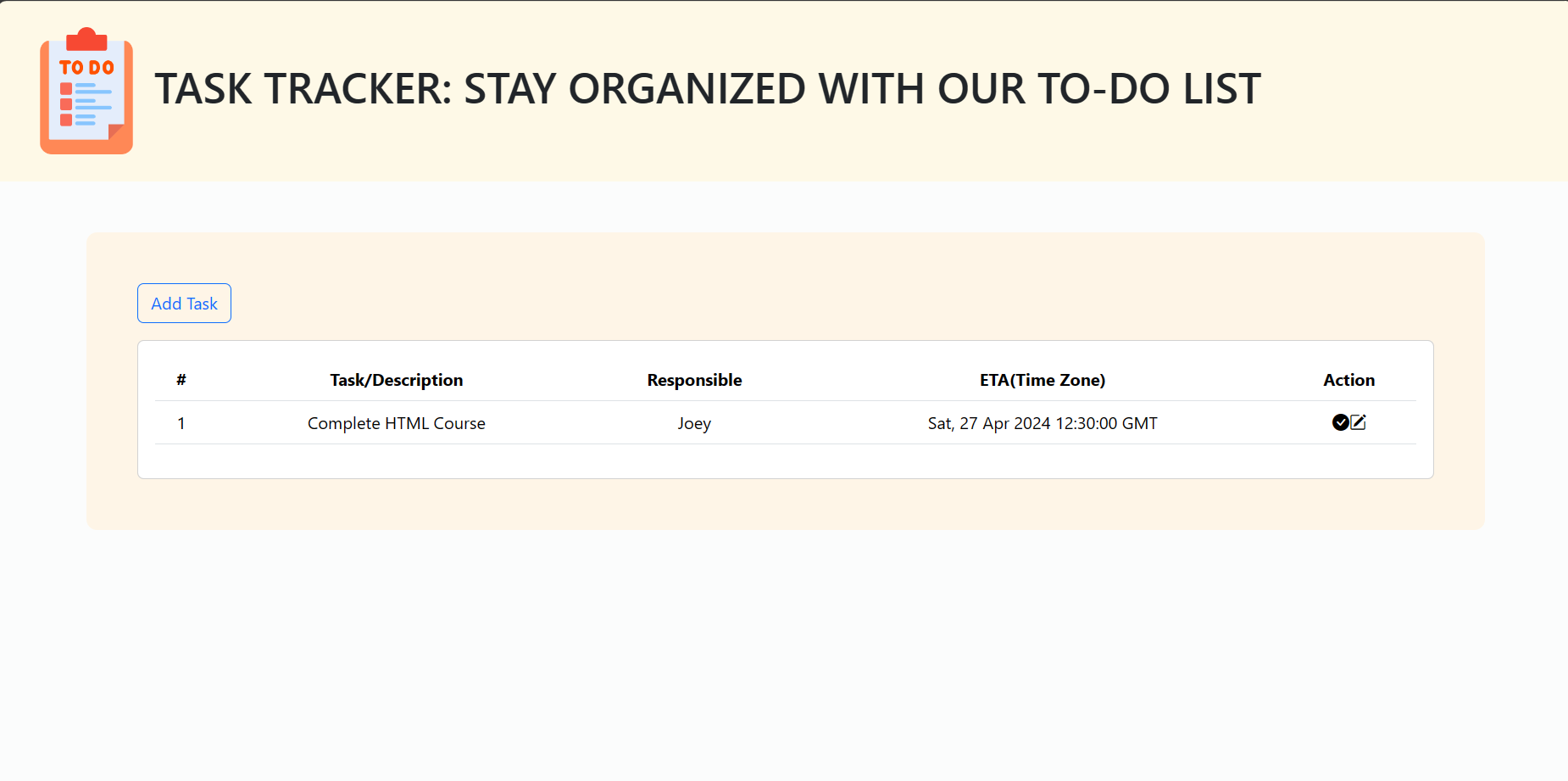
**ADD TASK FORM:**

****

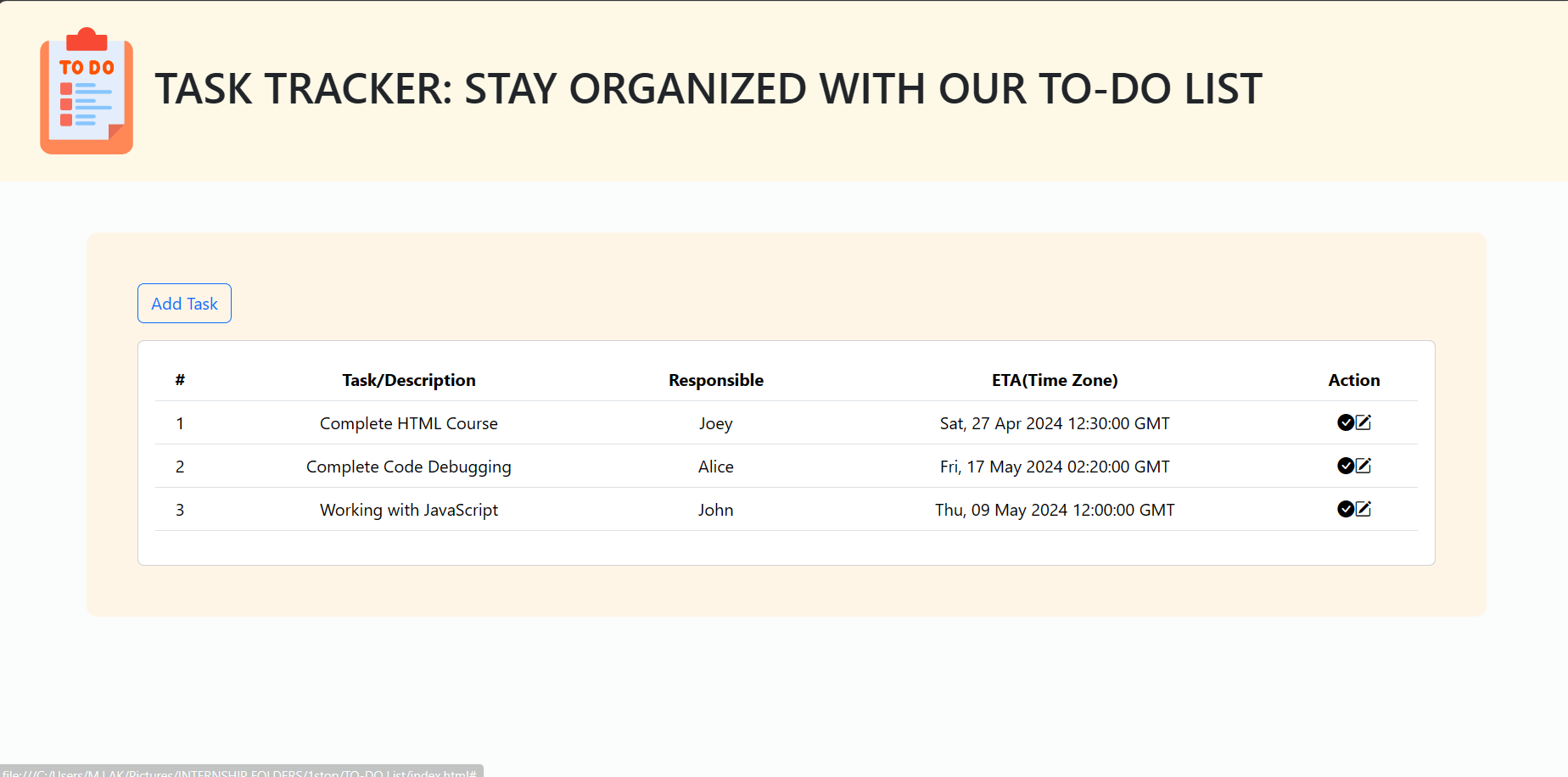
**FILLING THE DETAILS IN THE ADD TASK FORM:**

****

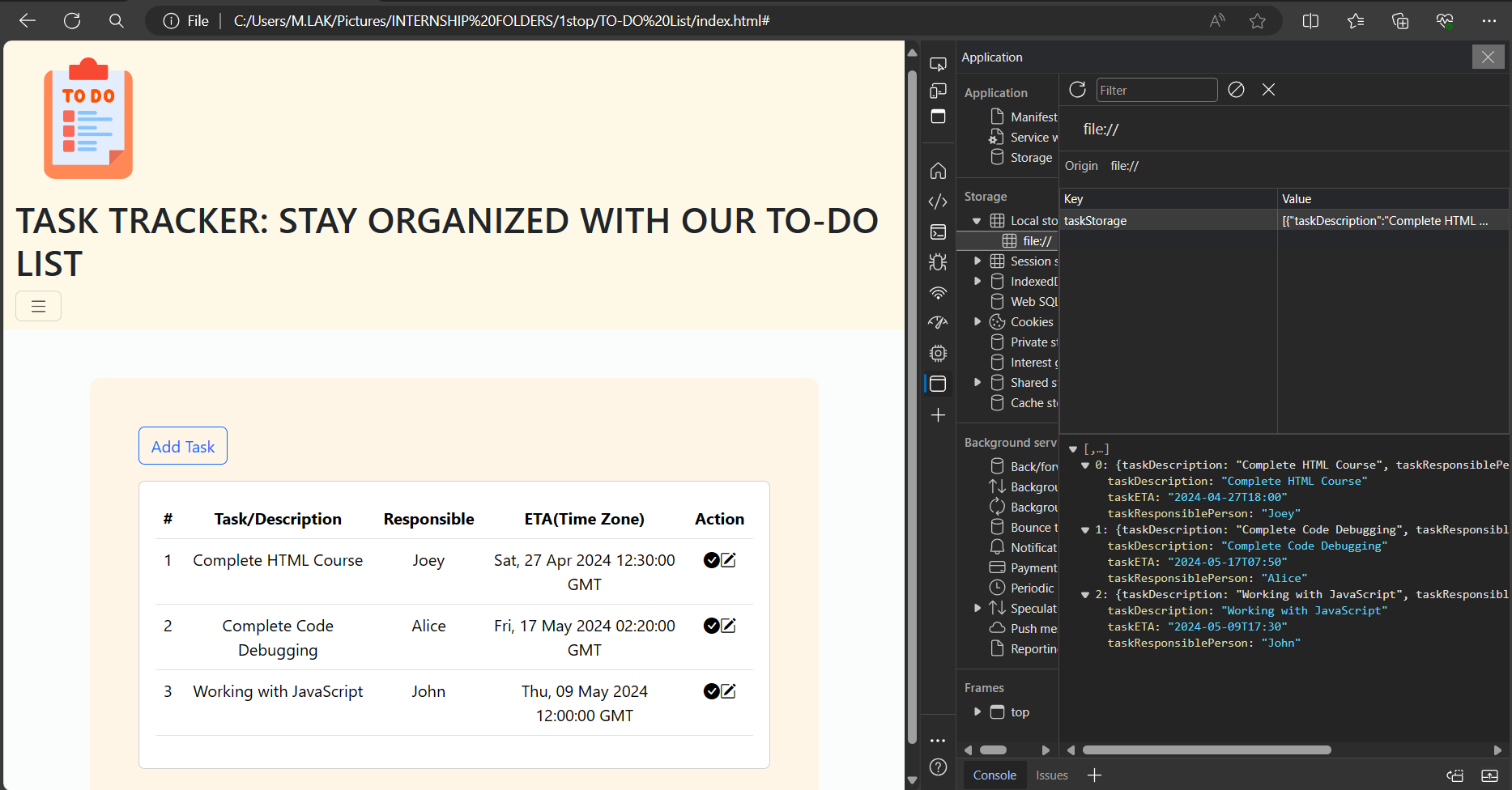
**DISPLAYING THE TASK DETAILS ON THE SCREEN:**

****

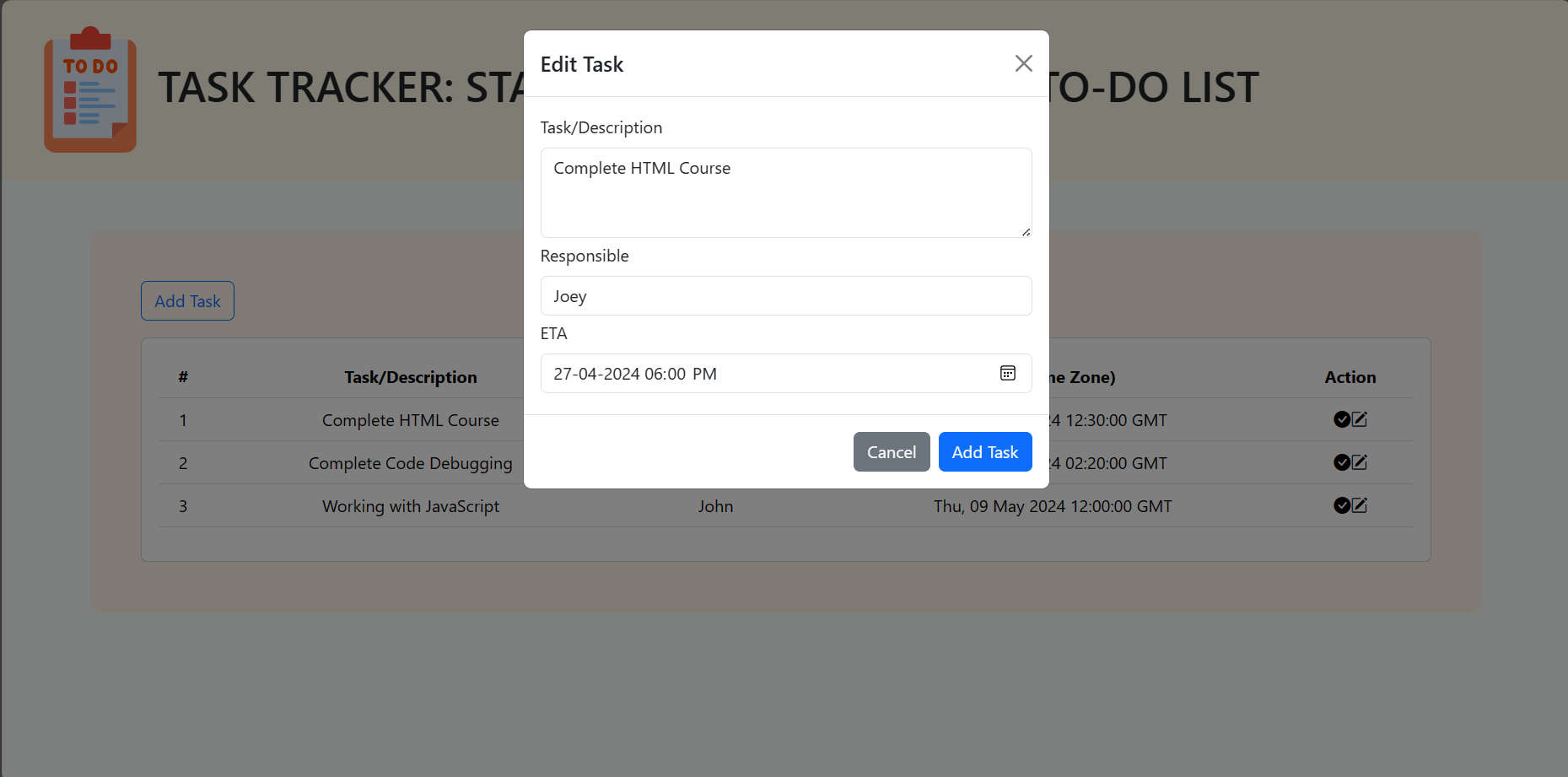
**MORE TASKS ADDED AND DISPLAYED ON THE SCREEN:**

****

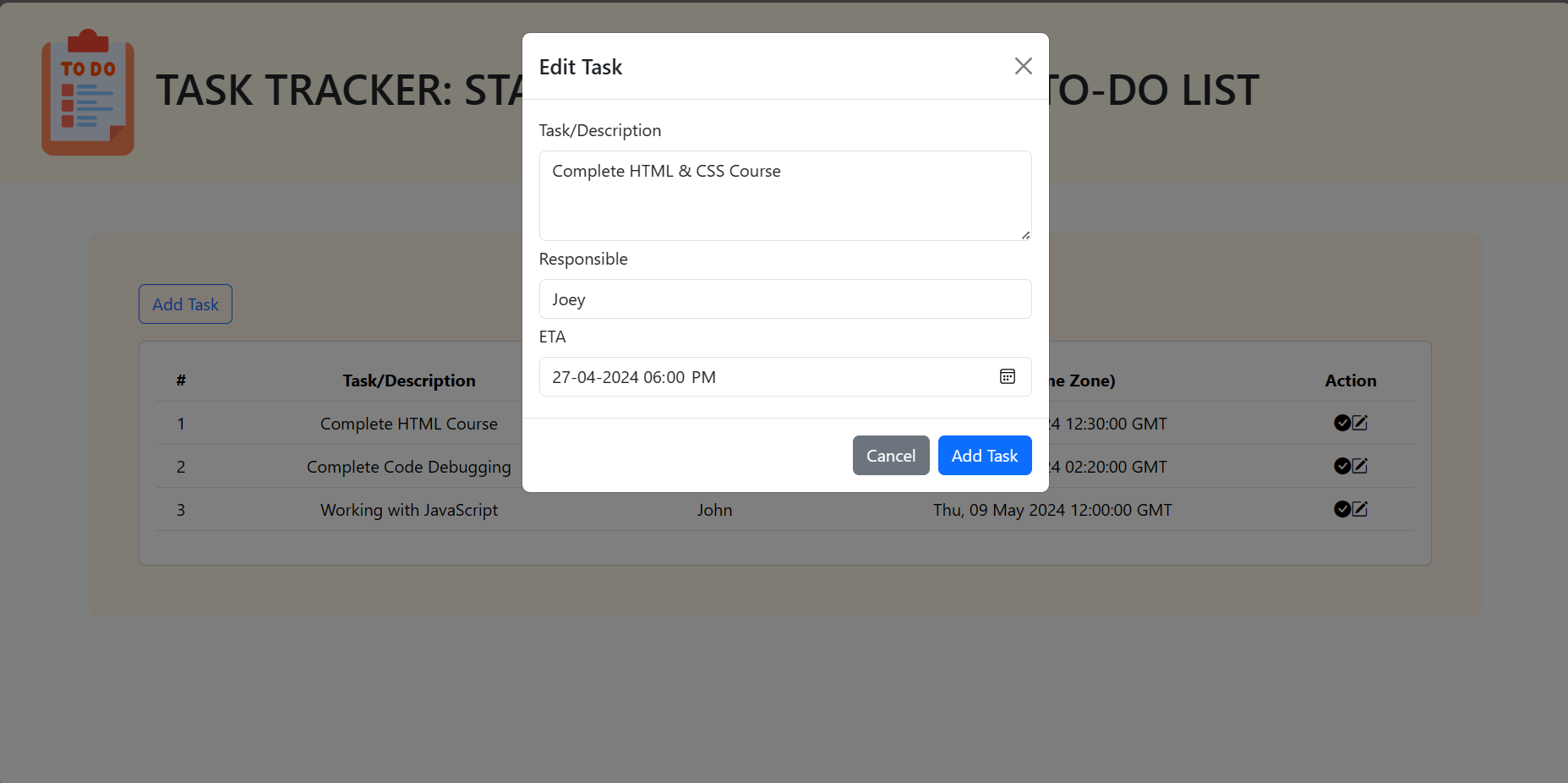
**LOCAL STORAGE OF THE ADDED TASKS:**

****

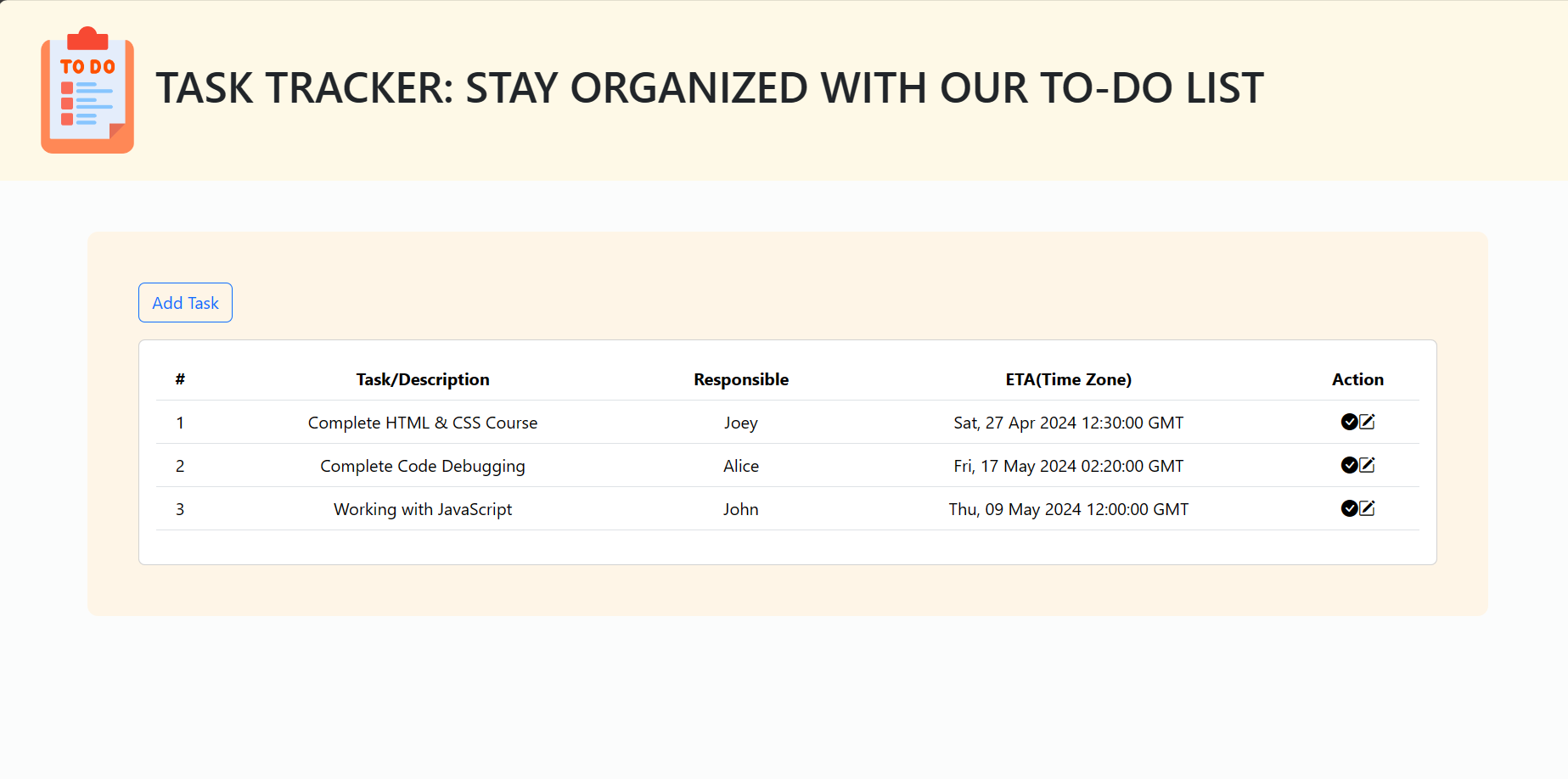
**EDITING OF A TASK DETAILS:**

****

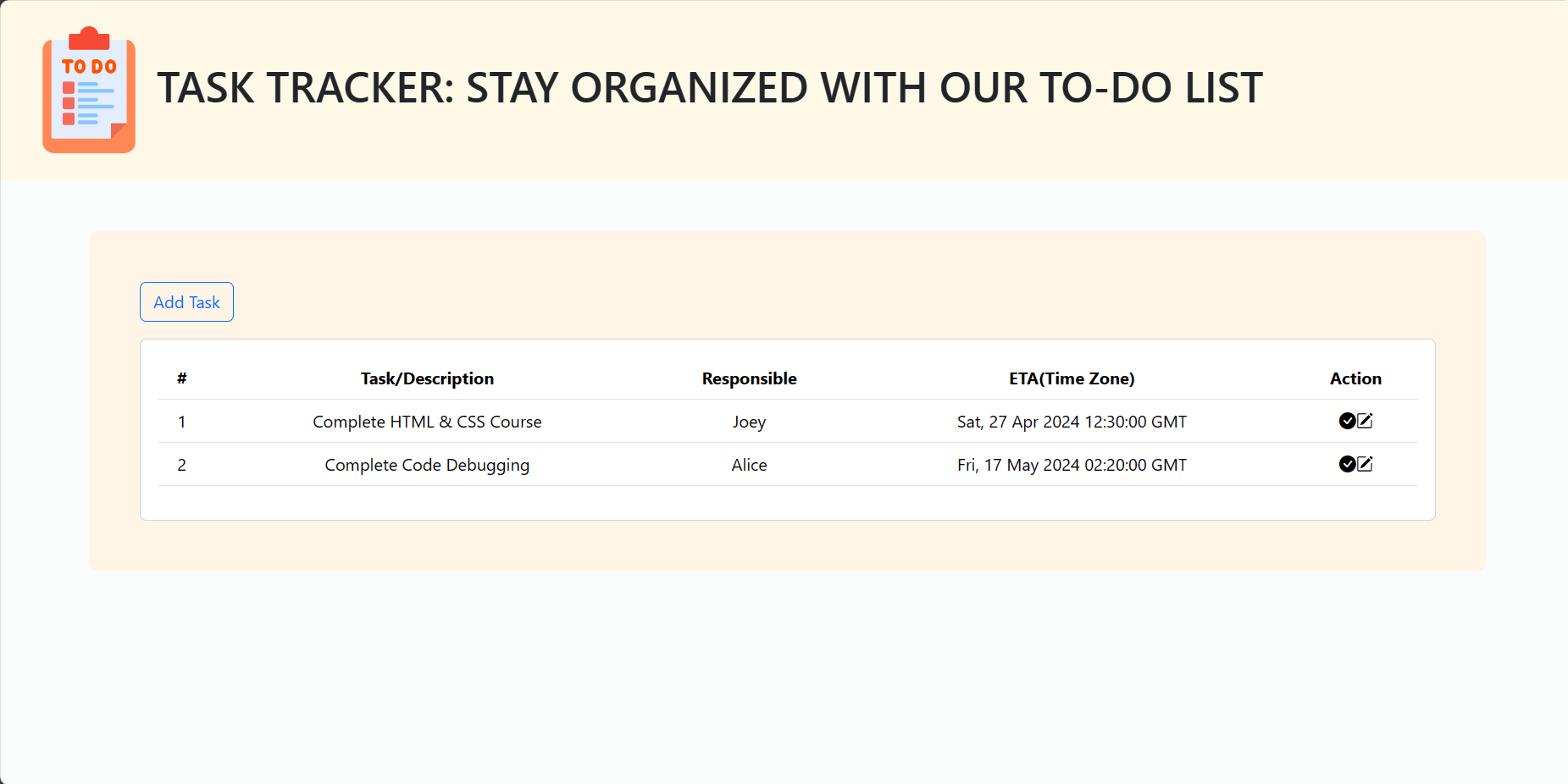
**EDITED THE TASK/DESCRIPTION DETAILS IN THE FORM:**

****

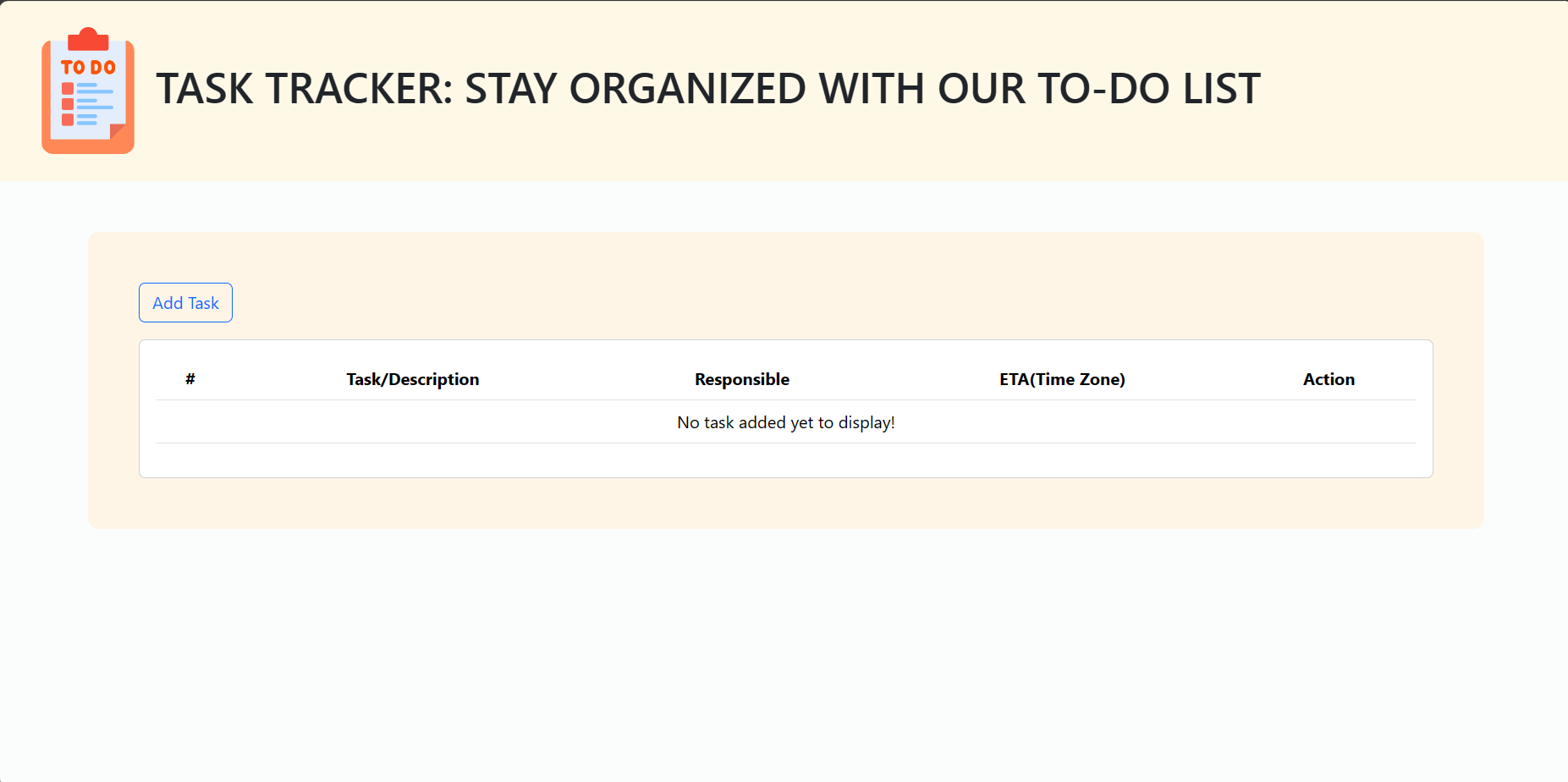
**REFLECTED THE EDITINGS ON THE WEBPAGE DISPLAY:**

****

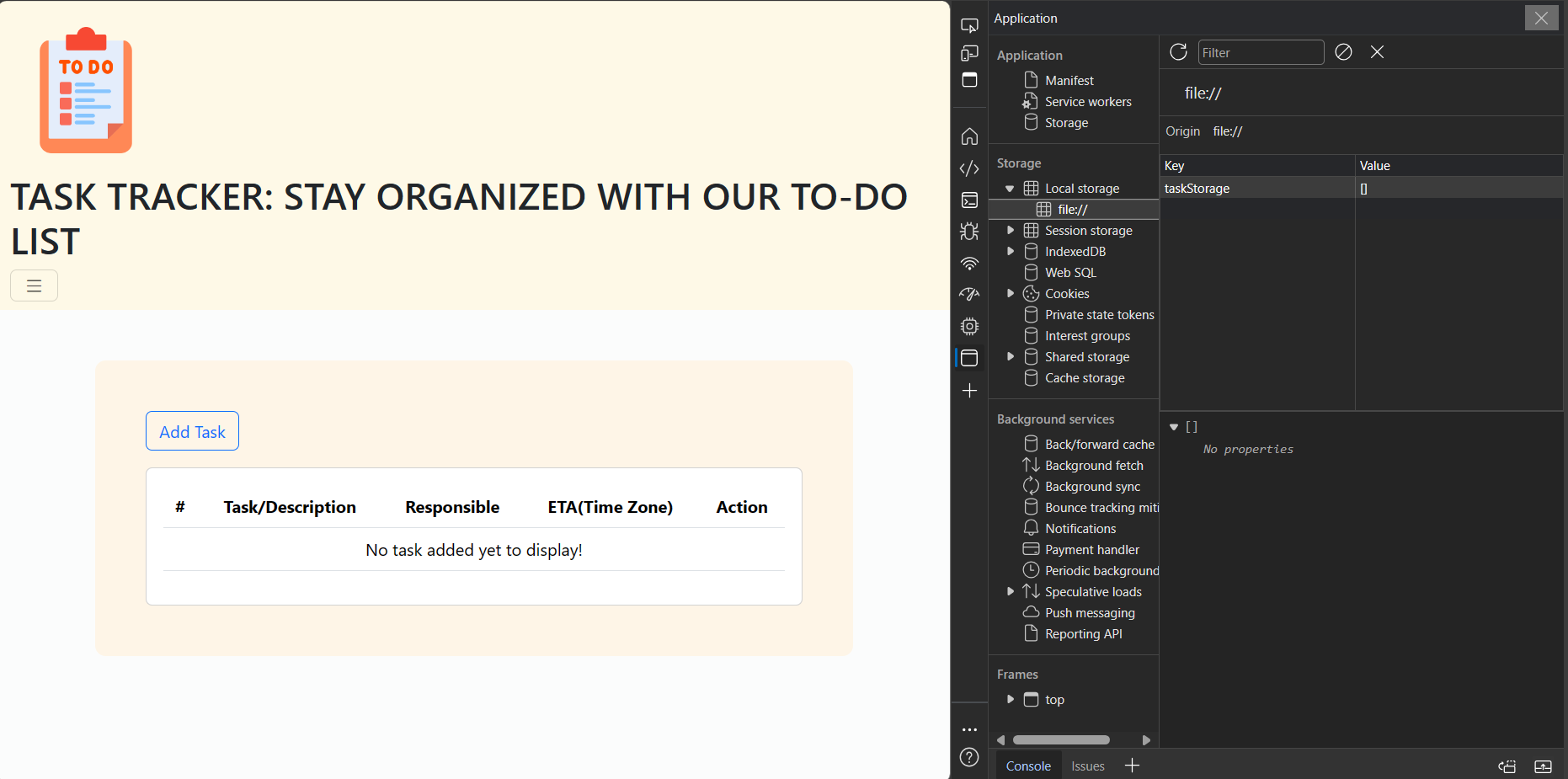
**TASK COMPLETED AND REMOVING FROM DISPLAY:**

****

**NO TASKS ARE PRESENT TO DISPLAY:**

****

**EMPTY LOCAL STORAGE:**

****

**CONCLUSION**

In conclusion, the development of a To-Do list using HTML, CSS, JavaScript, and jQuery has resulted in a versatile and user-friendly task management solution. By leveraging these technologies, we have successfully created a responsive and interactive To-Do list that empowers users to organize their tasks efficiently. Through careful planning, wireframing, and prototyping, followed by meticulous implementation of HTML structure, CSS styling, and JavaScript functionality, we have ensured a seamless user experience across various devices and screen sizes. Additionally, the utilization of jQuery has simplified DOM manipulation and event handling, contributing to the overall efficiency of the application. Moving forward, continued iteration and refinement based on user feedback will further enhance the To-Do list's usability and effectiveness, cementing its role as a valuable tool for personal and professional task management.