Internship Project

HTML To-Do List

Index:

I. Abstract:

This report presents a detailed overview of a To-Do List web application created using HTML, Bootstrap, and JavaScript. The primary objective of this project is to develop a user-friendly task management tool that allows users to efficiently manage their tasks. This web application provides essential features for task management, including the ability to add, edit, and mark tasks as completed. Crucially, all task data is stored locally in the user's browser through the utilization of the localStorage feature.

The report begins by introducing the concept of task management and the need for a digital solution that enhances productivity in both personal and professional contexts. It then delves into the methodology employed to build the To-Do List application. This methodology encompasses the creation of a structured HTML layout, the integration of Bootstrap for a visually appealing and responsive user interface, and the development of JavaScript functions that power the core functionalities of the application.

Key JavaScript functions, such as 'addTask()' for adding new tasks, 'writingOnPageFromStorage()' for retrieving and displaying tasks, 'markingAsDone(index)' for marking tasks as completed, and 'editTask(index)' for editing existing tasks, are explained in detail. These functions collectively provide users with a seamless task management experience.

Furthermore, the report highlights the inclusion of external libraries, including jQuery, Popper.js, and Bootstrap's JavaScript libraries, to enhance the user interface interactions, particularly with respect to the modal functionality used for adding and editing tasks.

The report concludes by emphasizing the practicality of this To-Do List web application as a tool for efficient task management. While the current implementation is straightforward, it serves as a solid foundation for future enhancements and customization. Potential improvements could include features such as task prioritization, due dates, and categorization, which would further enhance the application's versatility and user-friendliness. Overall, this report offers a comprehensive insight into the creation of a user-centric To-Do List web application designed to enhance task management and productivity.

II. Objective:

The primary objective of this report is to document the development and functionality of a To-Do List web application created using HTML, Bootstrap, and JavaScript. This web application aims to address the following objectives in the realm of task management:

1. **Efficient Task Management:** The foremost objective is to provide users with a digital tool that facilitates efficient task management. In both personal and professional contexts, individuals often struggle to organize and track their tasks effectively. This application seeks to alleviate this challenge by offering a user-friendly platform for managing tasks.

2. **User-Friendly Interface:** The project aims to deliver a user interface that is intuitive and easy to navigate. The objective is to create an environment where users can seamlessly add, edit, and monitor their tasks without encountering unnecessary complexity.

3. **Data Persistence:** To ensure a seamless user experience, the application employs the localStorage feature available in modern web browsers. The objective is to store task data locally, allowing users to access their task lists even after closing and reopening the browser.

4. **Responsive Design:** With the proliferation of various devices and screen sizes, the application's objective is to be responsive and adaptable. It should provide a consistent and visually appealing experience across different platforms, including desktops, tablets, and smartphones.

5. **Core Functionalities:** The project aims to implement core functionalities that are essential for effective task management, including the ability to add new tasks, edit existing ones, mark tasks as completed, and view task details such as descriptions and responsible individuals.

6. **Enhanced User Interactions:** Through the integration of external libraries such as jQuery, Popper.js, and Bootstrap, the application seeks to enhance user interactions and provide a modern and engaging user interface, particularly in the context of modal dialogs for adding and editing tasks.

7. **Foundation for Future Enhancements:** While the current implementation is straightforward, the application is designed with scalability in mind. The objective is to create a solid foundation upon which additional features and customizations, such as task prioritization, due dates, and categorization, can be built to further enhance its utility.

In summary, the objective of this report is to showcase the development of a practical To-Do List web application that caters to the fundamental needs of task management. By addressing these objectives, the application aims to provide users with an effective and user-friendly tool for organizing and optimizing their daily tasks and responsibilities.

III. Introduction:

Effective task management is a fundamental aspect of both personal and professional productivity. In today's fast-paced world, individuals are often inundated with numerous tasks, responsibilities, and deadlines, making it increasingly challenging to stay organized and on top of their workload. To address this common challenge, this report presents a detailed overview of a To-Do List web application developed using HTML, Bootstrap, and JavaScript.

The Need for Task Management Solutions

The demands of modern life require efficient methods for managing tasks, appointments, and commitments. In a digital age, where technology plays a central role in our daily lives, there is a growing need for digital task management solutions that are accessible, intuitive, and adaptable. Such solutions can help individuals and teams stay organized, reduce stress, and increase productivity by providing a structured framework for managing their tasks.

The Role of Digital To-Do Lists

Digital To-Do Lists have become indispensable tools for individuals and organizations seeking to streamline their workflow. Unlike traditional paper-based lists, digital solutions offer several advantages. They allow for dynamic editing, easy categorization, and the ability to set priorities and deadlines. Furthermore, they provide the convenience of access from anywhere with an internet connection, making them invaluable for today's mobile and remote workforces.

The To-Do List Web Application

The To-Do List web application presented in this report is designed with the aim of meeting the aforementioned needs and providing an efficient and user-friendly solution for task management. This application leverages a combination of web technologies to deliver a robust and responsive platform for managing tasks.

Methodology Overview

The development of this To-Do List application involved a systematic approach:

1. **HTML Structure:** The project commenced with the creation of the HTML structure, defining the layout and elements that constitute the user interface. This structure provides the foundation for the application.

2. **Bootstrap Integration:** Bootstrap, a widely-used front-end framework, was incorporated to enhance the application's aesthetics and responsiveness. Bootstrap's pre-designed components and CSS classes were leveraged to create an appealing and consistent design.

3. **JavaScript Functionality:** JavaScript was employed to implement the application's core functionalities. Functions such as 'addTask()' for adding new tasks, 'writingOnPageFromStorage()' for displaying tasks, 'markingAsDone(index)' for marking tasks as completed, and 'editTask(index)' for editing tasks were developed to enable seamless task management.

4. **External Libraries:** External libraries, including jQuery, Popper.js, and Bootstrap's JavaScript components, were integrated to facilitate enhanced user interactions, especially within modal dialogs for adding and editing tasks.

Report Structure
This report is structured to provide a comprehensive understanding of the To-Do List web application. It includes sections that detail the code implementation, showcase the application's functionality, and explore opportunities for future enhancements. The report aims to serve as a valuable resource for those interested in web development, task management, and user-centered digital solutions.

In conclusion, this To-Do List web application represents a practical response to the contemporary challenge of task management. By offering a user-centric interface and harnessing the capabilities of modern web technologies, it seeks to empower users to take control of their tasks and responsibilities, ultimately enhancing productivity and reducing the complexities of daily life.

IV. Methodology:

The development of the To-Do List web application involved a systematic approach that encompassed various stages, including design, coding, testing, and integration of external libraries. The following is a detailed breakdown of the methodology employed to create this application:

1. Project Planning and Design

1.1 Requirements Analysis: The project began with a thorough analysis of the requirements for the To-Do List web application. The primary goal was to identify the core functionalities that the application should support, including adding tasks, editing tasks, marking tasks as done, and displaying task details.

1.2 User Interface Design: A key aspect of the project was designing an intuitive and user-friendly interface. The layout and visual elements were carefully planned to ensure that users could easily interact with the application. Bootstrap, a popular front-end framework, was selected to streamline the design process and provide a responsive and visually appealing UI.

2. HTML Structure

2.1 Creating the HTML Skeleton: The HTML structure was the foundation of the application. The basic skeleton of the web page was established, including the header, navigation bar, modals for adding and editing tasks, and the task list display area. Semantic HTML elements were used to enhance accessibility and SEO.

3. Styling with Bootstrap

3.1 Bootstrap Integration: Bootstrap was integrated into the project to enhance the visual design and responsiveness of the application. Bootstrap's CSS classes and pre-designed components were utilized to create a consistent and aesthetically pleasing user interface.

4. JavaScript Functionality

4.1 Core Functionalities:

JavaScript was employed to implement the core functionalities of the To-Do List application. The following key functions were developed:

addTask(): This function allows users to input task details via a modal dialog and stores the task data in the browser's localStorage. It also triggers the refreshing of the task list display.

writingOnPageFromStorage(): Responsible for retrieving task data from localStorage and dynamically generating the HTML for displaying tasks in a table format. It checks for empty or null data and provides appropriate feedback to users.

markingAsDone(index): Enables users to mark tasks as completed by removing them from localStorage. This function also updates the task list display accordingly.

editTask(index): Permits users to edit existing tasks by pre-filling the modal form fields with the selected task's details. After editing, the old task is removed from localStorage, and the updated task is saved, with the task list display being updated to reflect the changes.

4.2 External Libraries: The project integrated external libraries to enhance user interactions and functionality. These libraries included:

jQuery: Used for simplified DOM manipulation and event handling, making it easier to work with HTML elements.

Popper.js: Required by Bootstrap for handling tooltips and popovers.

5. Testing and Debugging

5.1 Functional Testing: Throughout the development process, rigorous functional testing was conducted to ensure that all core functionalities, such as adding, editing, and marking tasks, were working as intended. Various scenarios and edge cases were tested to validate the application's reliability.

5.2 Cross-Browser and Cross-Device Testing: The application was tested on multiple web browsers (e.g., Chrome, Firefox, Edge) and devices (desktops, tablets, smartphones) to ensure compatibility and responsiveness.

5.3 Debugging: Debugging tools and browser developer consoles were used to identify and fix any errors or issues in the code.

6. Documentation

6.1 Code Documentation: Comments and documentation were added to the JavaScript code to explain the purpose and functionality of each function, enhancing code readability and maintainability.

7. Future Enhancements

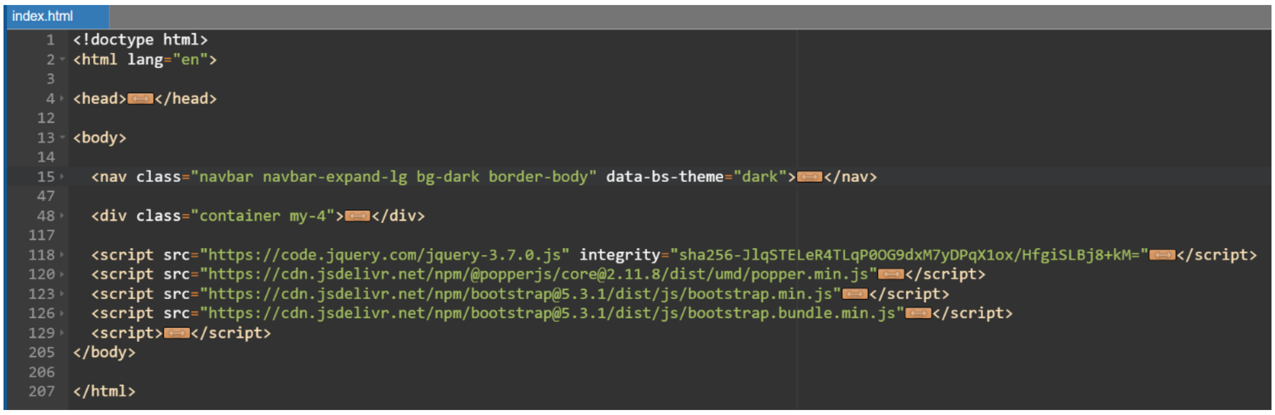
The To-Do List web application was designed with extensibility in mind. Future enhancements could include additional features such as task prioritization, due dates, task categories, and user authentication, further enhancing the application's utility and versatility.

8. Conclusion

The methodology employed for the development of the To-Do List web application encompassed careful planning, design, implementation, testing, and documentation. The result is a user-friendly and feature-rich task management tool that leverages modern web technologies to address the challenges of efficient task organization and productivity.

V. Code with results and output:

i. Overview of Code:

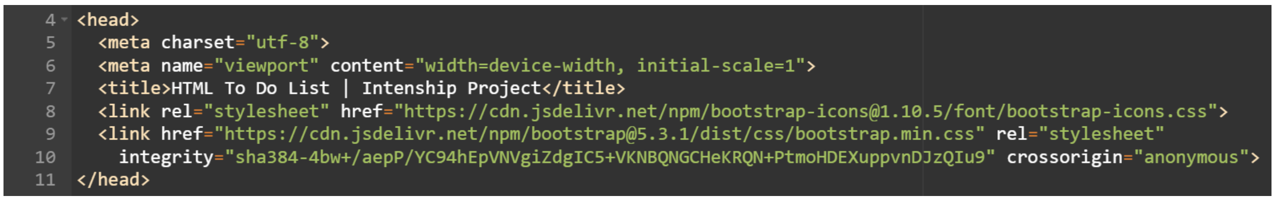


index.html

```html
<!doctype html>
<html lang="en">

<head>↔</head>

<body>

  <nav class="navbar navbar-expand-lg bg-dark border-body" data-bs-theme="dark">↔</nav>

  <div class="container my-4">↔</div>

  <script src="https://code.jquery.com/jquery-3.7.0.js" integrity="sha256-JlqSTELeR4TLqP0OG9dxM7yDPqX1ox/HfgiSLBj8+kM="↔</script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js"↔</script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.min.js"↔</script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"↔</script>
  <script>↔</script>
</body>

</html>
```

This is the overall code which contains different sections of head, main container and script tags which will shown further.

ii. Head section:



```html
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>HTML To Do List | Intenship Project</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9" crossorigin="anonymous">
</head>
```

The code starts with the <!doctype html> declaration and defines an HTML document. The <head> section contains metadata such as character encoding, viewport settings, and title. Several external stylesheets are linked using <link> tags. These include Bootstrap CSS and Bootstrap Icons CSS.

iii. Div Container for Layout of HTML To-Do List:

Inside the <body> element, the code includes:

Navbar: A Bootstrap-based navigation bar with links and a branding logo.

Main Content Container: A container for the main content of the web page.

Page Title: An <h2> element displaying the title of the application.

"Add List Items" Button: A button that triggers a modal for adding tasks.

Modal for Adding Tasks: A modal dialog that opens when the "Add List Items" button is clicked. This modal contains form fields for adding task details.

Task List Display: An empty <div> with the id="items" where task entries will be displayed in a table.

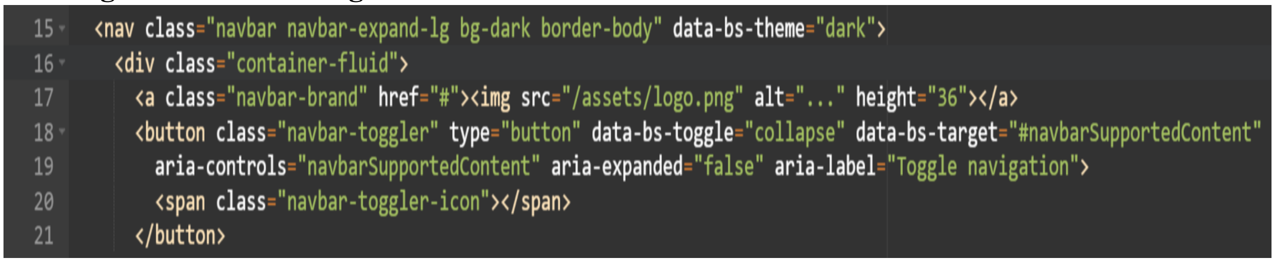
JavaScript Libraries: External JavaScript libraries (jQuery, Popper.js, Bootstrap) are included at the end of the <body> section. These libraries are essential for the application's functionality.

JavaScript Code: A <script> section at the end of the document for writing custom JavaScript code to manage tasks.

<u>Navbar:</u>

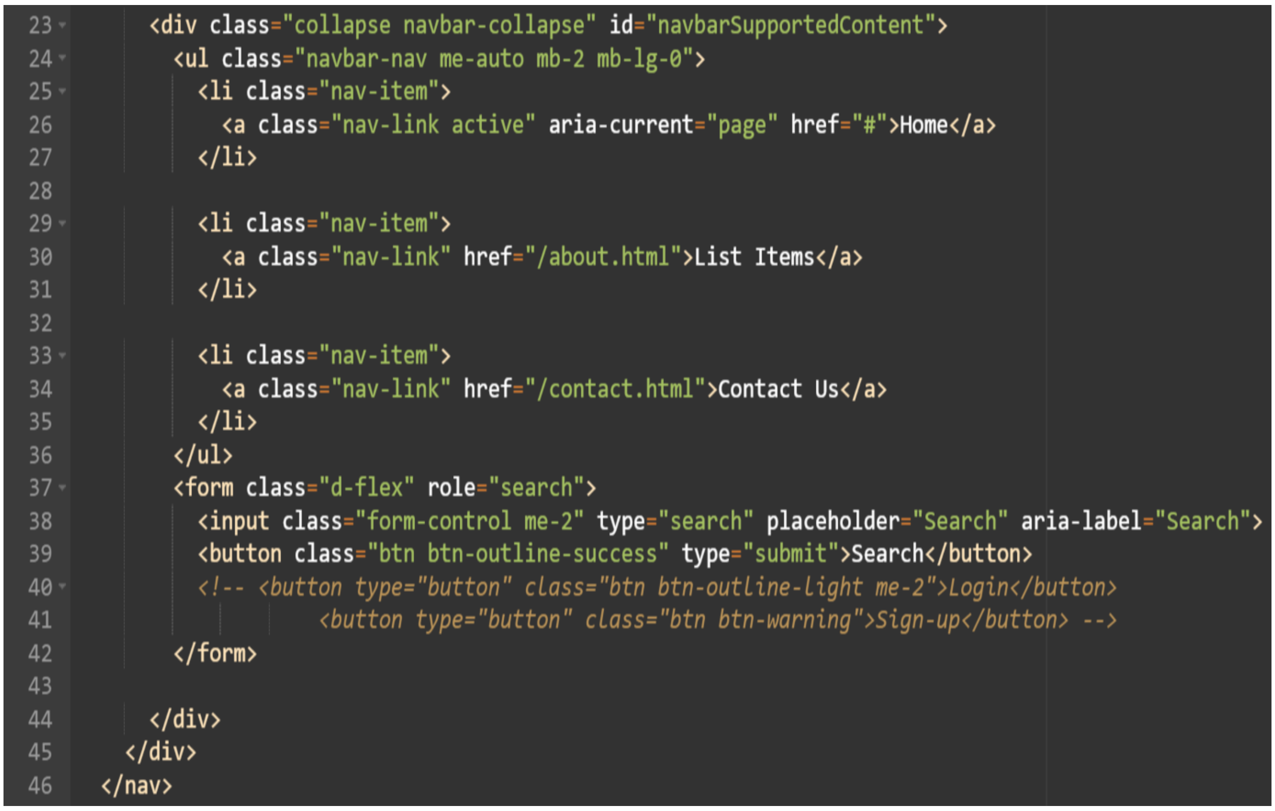
The provided code snippet represents the implementation of a responsive navigation bar using Bootstrap's navbar component.

1. **Starting of navbar and logo:**



```html
<nav class="navbar navbar-expand-lg bg-dark border-body" data-bs-theme="dark">
  <div class="container-fluid">
    <a class="navbar-brand" href="#"><img src="/assets/logo.png" alt="..." height="36"></a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
      aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
```

2. **Different sections and search button:**



```html
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>

        <li class="nav-item">
          <a class="nav-link" href="/about.html">List Items</a>
        </li>

        <li class="nav-item">
          <a class="nav-link" href="/contact.html">Contact Us</a>
        </li>
      </ul>
      <form class="d-flex" role="search">
        <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
        <!-- <button type="button" class="btn btn-outline-light me-2">Login</button>
                  <button type="button" class="btn btn-warning">Sign-up</button> -->
      </form>

    </div>
  </div>
</nav>
```

Explanation:

This code snippet defines the navigation bar (<nav>) for the web application using Bootstrap classes to style and structure it.

navbar: This class indicates that this is a Bootstrap navbar.

navbar-expand-lg: It specifies that the navbar should expand for large screens (desktops) and collapse for smaller screens (mobile devices) when the navigation items don't fit.

bg-dark: Sets the background color of the navbar to dark.

border-body: Adds a border to the navbar.

data-bs-theme="dark": Custom data attribute that may be used for further styling or functionality related to the theme.

Inside the navbar, you have the following components:

container-fluid: A container for the navbar content to ensure proper alignment.

navbar-brand: This is typically a logo or brand name. In this case, it includes an image with the src attribute pointing to "/assets/logo.png" and an alt attribute for accessibility. The height attribute sets the logo's height.

navbar-toggler: This button is used for toggling the navigation menu on smaller screens. It has an icon represented by navbar-toggler-icon.

collapse navbar-collapse: This div contains the collapsible navigation items. It is assigned an id of "navbarSupportedContent," which is referenced by the button's data-bs-target attribute to specify which part of the HTML will collapse when the button is clicked.

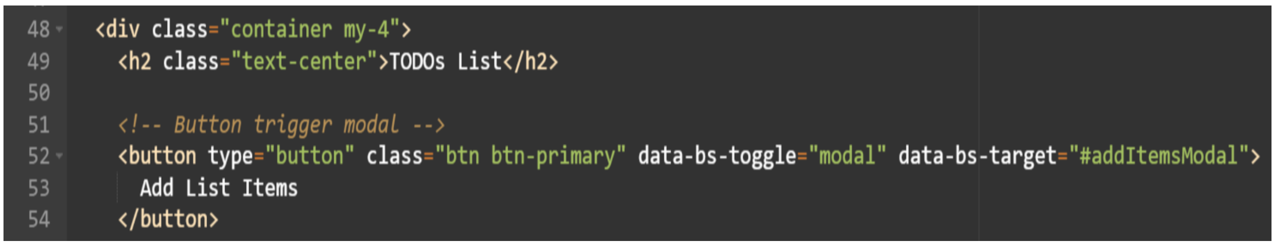
ul.navbar-nav: This is an unordered list that holds the navigation items. The navbar-nav class styles it as part of the navigation bar.

li.nav-item: Each list item represents a navigation link. The nav-link class styles the link.

Inside each li.nav-item, you have links (<a>) with nav-link classes that define the navigation items. The href attribute specifies the destination of each link, and the text inside the <a> tags represents the link text.

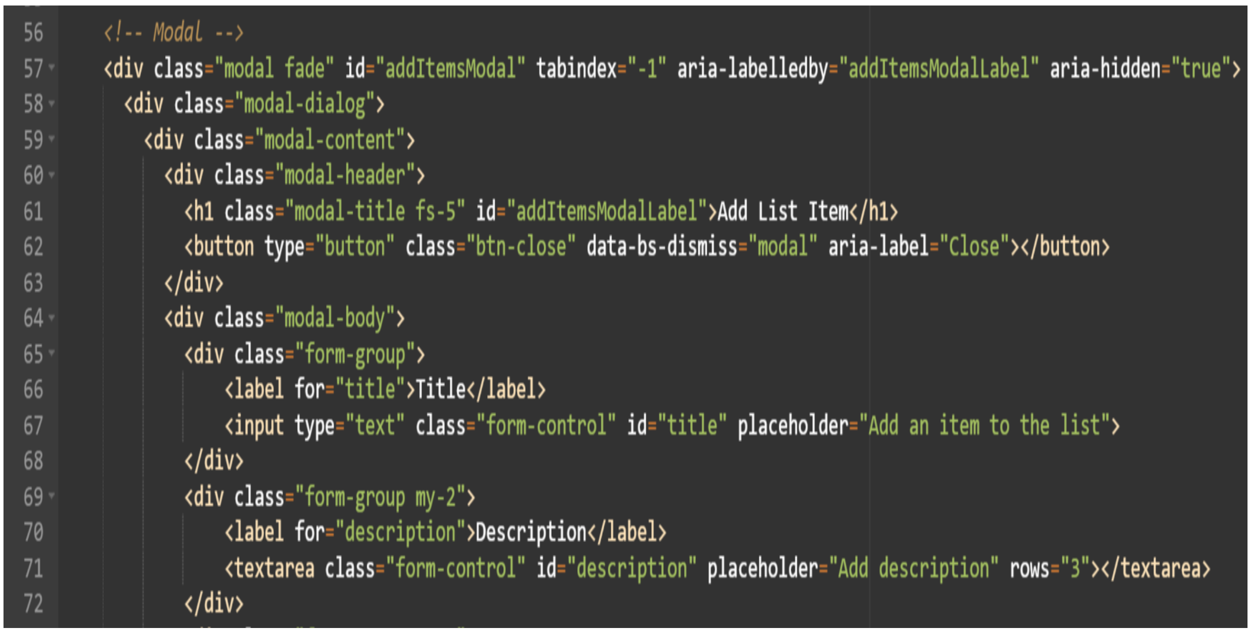
After the navigation items, there's a search form with an input field and a search button. The form uses the d-flex class for flexible alignment, and the input field has a form-control class for styling. The search button is styled as an outline button with btn-outline-success class.

3. **Starting of main container and add items modal:**



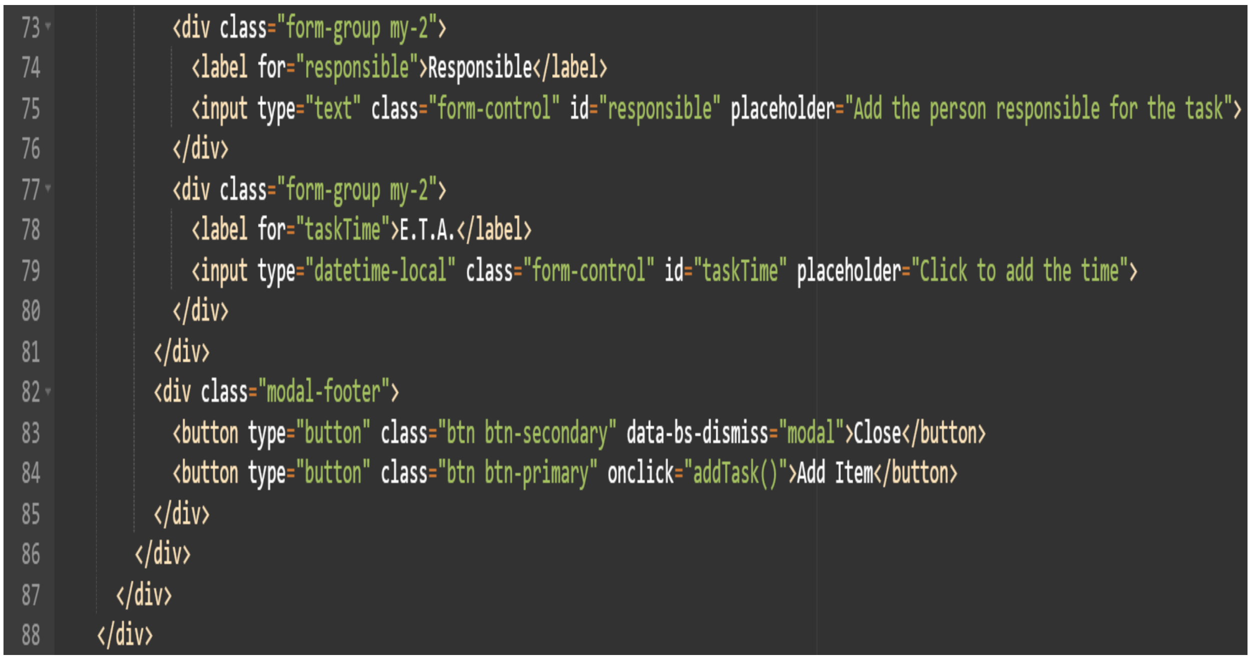
```html
<div class="container my-4">
  <h2 class="text-center">TODOs List</h2>

  <!-- Button trigger modal -->
  <button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#addItemsModal">
    Add List Items
  </button>
```

4. **Add Items Modal:**



```html
  <!-- Modal -->
  <div class="modal fade" id="addItemsModal" tabindex="-1" aria-labelledby="addItemsModalLabel" aria-hidden="true">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <h1 class="modal-title fs-5" id="addItemsModalLabel">Add List Item</h1>
          <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
        </div>
        <div class="modal-body">
          <div class="form-group">
            <label for="title">Title</label>
            <input type="text" class="form-control" id="title" placeholder="Add an item to the list">
          </div>
          <div class="form-group my-2">
            <label for="description">Description</label>
            <textarea class="form-control" id="description" placeholder="Add description" rows="3"></textarea>
          </div>
```

5. **Ending of Add Items Modal:**



```html
          <div class="form-group my-2">
            <label for="responsible">Responsible</label>
            <input type="text" class="form-control" id="responsible" placeholder="Add the person responsible for the task">
          </div>
          <div class="form-group my-2">
            <label for="taskTime">E.T.A.</label>
            <input type="datetime-local" class="form-control" id="taskTime" placeholder="Click to add the time">
          </div>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
          <button type="button" class="btn btn-primary" onclick="addTask()">Add Item</button>
        </div>
      </div>
    </div>
  </div>
```

Explanation:

This code snippet defines the modal dialog for adding new tasks in the web application.

modal fade: The modal is styled using Bootstrap's modal component. The fade class provides a fade-in effect when the modal is displayed.

id="addItemsModal": Assigns an ID to the modal, which can be used to reference it in JavaScript and to trigger its display.

tabindex="-1" and aria-labelledby="addItemsModalLabel" aria-hidden="true": These attributes control the accessibility and keyboard navigation of the modal.

Inside the modal, you have the following components:

modal-dialog: The modal dialog box that contains the modal's content.

modal-content: The content container within the dialog.

modal-header: The header section of the modal containing the title and a close button.

modal-title fs-5 and id="addItemsModalLabel": The title of the modal. The fs-5 class styles it as a font size 5 element.

button type="button" class="btn-close": The close button for the modal. It has a data-bs-dismiss attribute that allows it to close the modal when clicked. The aria-label attribute provides an accessible label for screen readers.

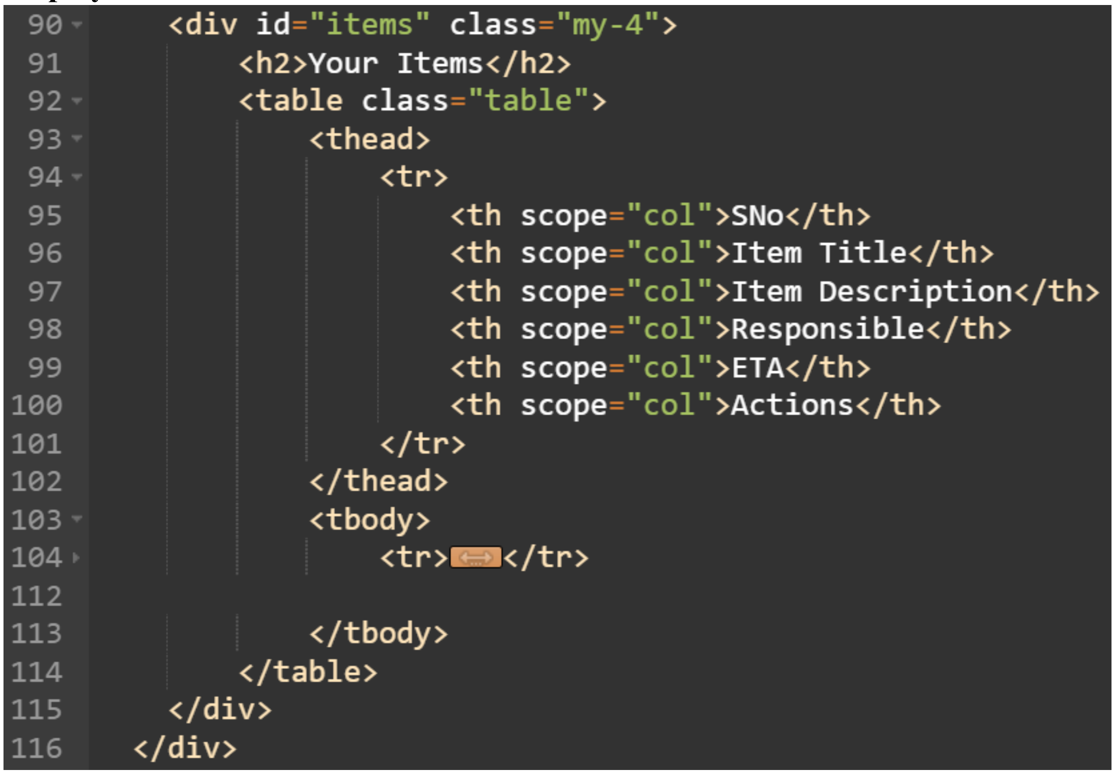
modal-body: This is where the form for adding task details will be placed. It will include input fields for the title, description, responsible person, and task time.

modal-footer: The footer section of the modal containing buttons.

button type="button" class="btn btn-secondary": A "Close" button that dismisses the modal without adding a task.

button type="button" class="btn btn-primary" onclick="addTask()": An "Add Item" button. When clicked, it calls the addTask() JavaScript function to add a new task with the details entered in the form.

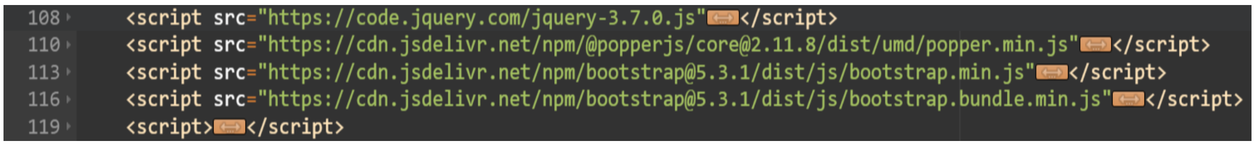
6. **Display Table:**



```html
    <div id="items" class="my-4">
        <h2>Your Items</h2>
        <table class="table">
            <thead>
                <tr>
                    <th scope="col">SNo</th>
                    <th scope="col">Item Title</th>
                    <th scope="col">Item Description</th>
                    <th scope="col">Responsible</th>
                    <th scope="col">ETA</th>
                    <th scope="col">Actions</th>
                </tr>
            </thead>
            <tbody>
                <tr>⇔</tr>

            </tbody>
        </table>
    </div>
</div>
```

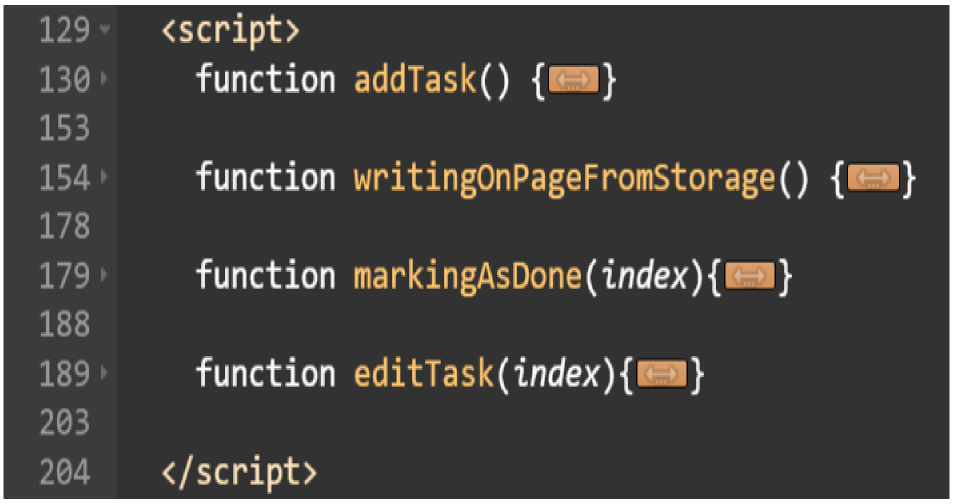
iv. <u>Script Tags:</u>

The code includes JavaScript libraries (jQuery, Popper.js, Bootstrap) to enable various interactive and responsive features of the website



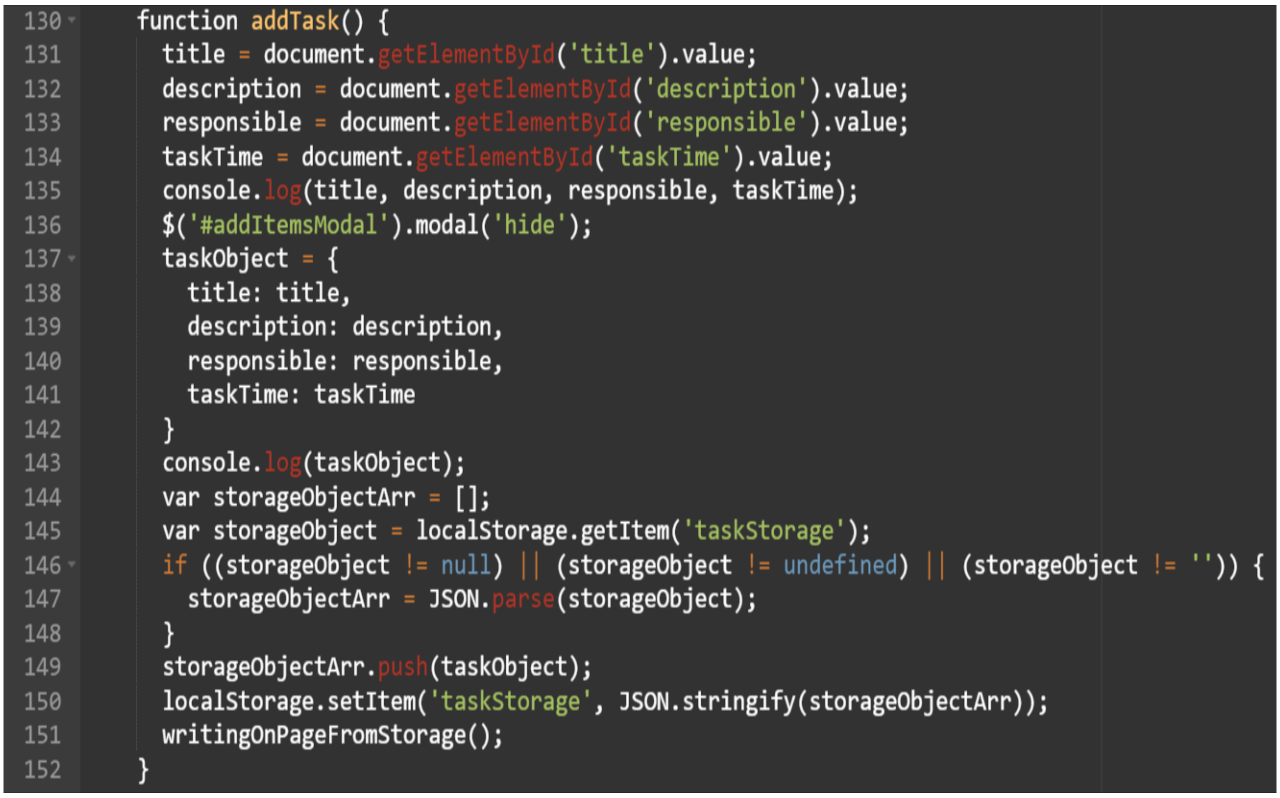
```html
<script src="https://code.jquery.com/jquery-3.7.0.js"⇔</script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js"⇔</script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.min.js"⇔</script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"⇔</script>
<script>⇔</script>
```

v. <u>Script Tags for functionality of To-Do List:</u>



```html
<script>
  function addTask() {⇔}

  function writingOnPageFromStorage() {⇔}

  function markingAsDone(index){⇔}

  function editTask(index){⇔}

</script>
```

1. **Add Task functionality:**



```
  function addTask() {
    title = document.getElementById('title').value;
    description = document.getElementById('description').value;
    responsible = document.getElementById('responsible').value;
    taskTime = document.getElementById('taskTime').value;
    console.log(title, description, responsible, taskTime);
    $('#addItemsModal').modal('hide');
    taskObject = {
      title: title,
      description: description,
      responsible: responsible,
      taskTime: taskTime
    }
    console.log(taskObject);
    var storageObjectArr = [];
    var storageObject = localStorage.getItem('taskStorage');
    if ((storageObject != null) || (storageObject != undefined) || (storageObject != '')) {
      storageObjectArr = JSON.parse(storageObject);
    }
    storageObjectArr.push(taskObject);
    localStorage.setItem('taskStorage', JSON.stringify(storageObjectArr));
    writingOnPageFromStorage();
  }
```

Explanation:

- The addTask() function is responsible for adding a new task to the To-Do List when the user submits the task details through the modal.

- It begins by collecting task details from the modal form fields. It uses document.getElementById() to retrieve the values of the title, description, responsible person, and task time fields.

- Next, it hides the modal using $('#addItemsModal').modal('hide'), which is a jQuery-based operation. This operation closes the modal after the user submits the task details.

- The function then creates a JavaScript object called taskObject to represent the task. It structures the task data with properties for the title, description, responsible person, and task time.

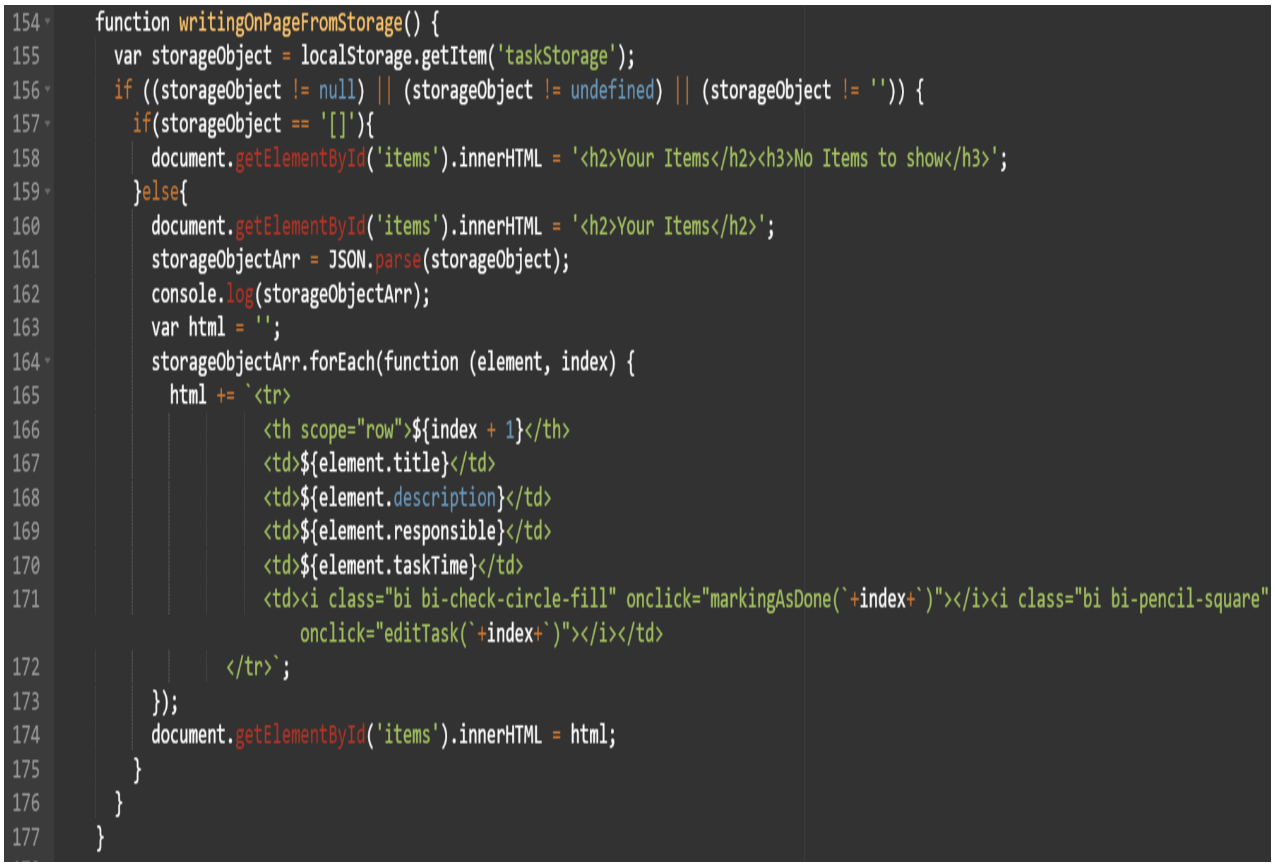
- To store the task data, the function retrieves the existing task data from localStorage. If there is already data in localStorage, it parses that data into an array called storageObjectArr. If localStorage is empty or doesn't exist, it initializes storageObjectArr as an empty array.

- The new task object (taskObject) is added to the storageObjectArr using the push() method, effectively adding the new task to the existing list of tasks.

- Finally, the updated task list (now including the new task) is stored back in localStorage using localStorage.setItem(), with the key 'taskStorage'. The data is first converted to a JSON string using JSON.stringify().

- After storing the task in localStorage, the writingOnPageFromStorage() function is called to refresh the task list display on the web page, ensuring that the new task is visible to the user.

2. Writing on Page from Storage functionality:



```
function writingOnPageFromStorage() {
  var storageObject = localStorage.getItem('taskStorage');
  if ((storageObject != null) || (storageObject != undefined) || (storageObject != '')) {
    if(storageObject == '[]'){
      document.getElementById('items').innerHTML = '<h2>Your Items</h2><h3>No Items to show</h3>';
    }else{
      document.getElementById('items').innerHTML = '<h2>Your Items</h2>';
      storageObjectArr = JSON.parse(storageObject);
      console.log(storageObjectArr);
      var html = '';
      storageObjectArr.forEach(function (element, index) {
        html += `<tr>
                 <th scope="row">${index + 1}</th>
                 <td>${element.title}</td>
                 <td>${element.description}</td>
                 <td>${element.responsible}</td>
                 <td>${element.taskTime}</td>
                 <td><i class="bi bi-check-circle-fill" onclick="markingAsDone(`+index+`)"></i><i class="bi bi-pencil-square"
                   onclick="editTask(`+index+`)"></i></td>
             </tr>`;
      });
      document.getElementById('items').innerHTML = html;
    }
  }
}
```

Explanation:

- The writingOnPageFromStorage() function is responsible for displaying tasks from localStorage on the web page. It ensures that the task list is up-to-date and visible to the user.

- It begins by retrieving the task data stored in localStorage using localStorage.getItem('taskStorage') and stores it in the storageObject variable.

- Next, it checks if storageObject is not null, not undefined, and not an empty string using the if statement. This ensures that there is valid task data in localStorage to display.

- If the condition is met, the function proceeds to update the task list display. It checks if the stored data is an empty array by comparing it to the string '[]'. If it's empty, it displays a message stating "No Items to show." Otherwise, it generates HTML to display the task list.

- When tasks are present, it initializes an empty html variable to store the HTML content of the task list.

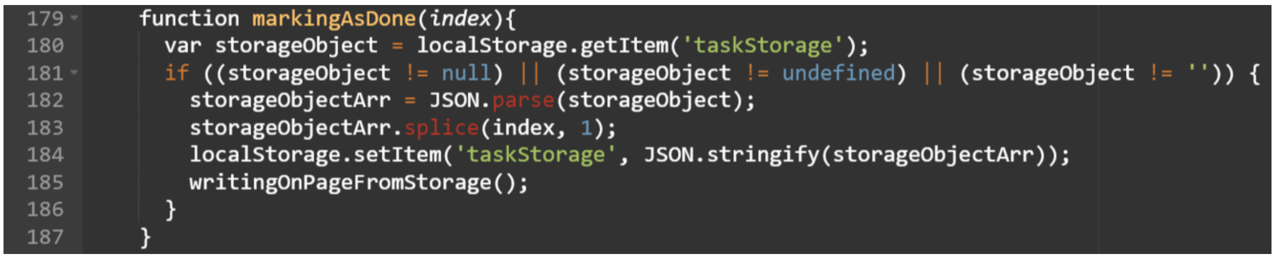
- It then parses the JSON data in storageObject into an array called storageObjectArr.

- Using a forEach loop, it iterates through each task in storageObjectArr. For each task, it generates a row in an HTML table (<tr>) with columns for the task's serial number, title, description, responsible person, task time, and action icons.

- The action icons include a checkmark icon for marking the task as done and a pencil icon for editing the task. These icons have onclick attributes that call the respective JavaScript functions (markingAsDone(index) and editTask(index)), passing the index of the task as a parameter.

- Finally, the generated HTML content (html) is injected into the "items" <div> element using document.getElementById('items').innerHTML = html;, updating the task list display.

3. **Marking as Done functionality:**



```
function markingAsDone(index){
  var storageObject = localStorage.getItem('taskStorage');
  if ((storageObject != null) || (storageObject != undefined) || (storageObject != '')) {
    storageObjectArr = JSON.parse(storageObject);
    storageObjectArr.splice(index, 1);
    localStorage.setItem('taskStorage', JSON.stringify(storageObjectArr));
    writingOnPageFromStorage();
  }
}
```

Explanation:

- The markingAsDone(index) function is responsible for marking a task as done and removing it from the task list. It takes an index parameter, which specifies the index of the task to be marked as done.

- It begins by retrieving the task data stored in localStorage using localStorage.getItem('taskStorage') and stores it in the storageObject variable.

- Next, it checks if storageObject is not null, not undefined, and not an empty string using the if statement. This ensures that there is valid task data in localStorage to process.

- If the condition is met, the function proceeds to work with the task data:

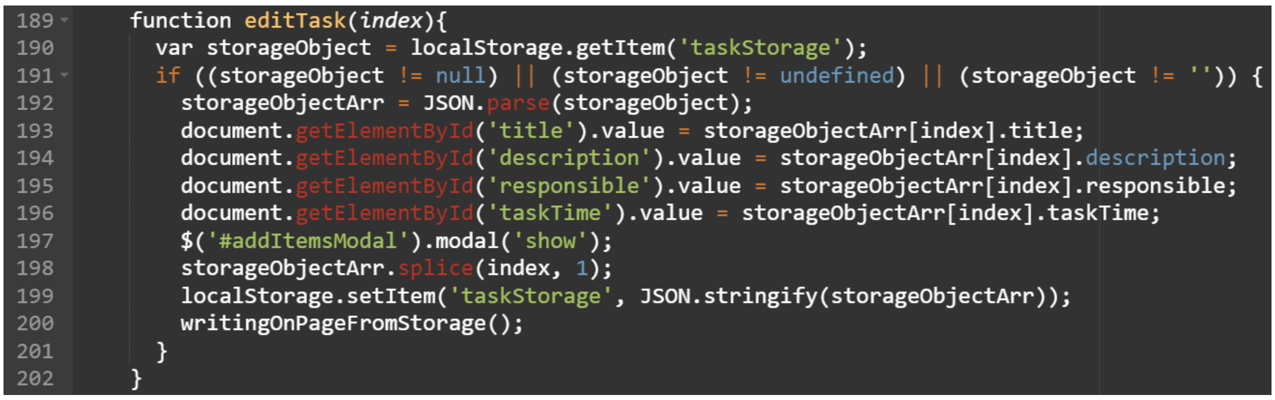
- It parses the JSON data in storageObject into an array called storageObjectArr to work with the task list.

- Using the splice() method, it removes the task at the specified index from storageObjectArr. This effectively marks the task as done by removing it from the list.

- The updated storageObjectArr is then stored back in localStorage using localStorage.setItem('taskStorage', JSON.stringify(storageObjectArr)).

- Finally, the writingOnPageFromStorage() function is called to refresh the task list display on the web page, ensuring that the removed task is no longer visible in the list.

4. **Edit task functionality:**



```
function editTask(index){
  var storageObject = localStorage.getItem('taskStorage');
  if ((storageObject != null) || (storageObject != undefined) || (storageObject != '')) {
    storageObjectArr = JSON.parse(storageObject);
    document.getElementById('title').value = storageObjectArr[index].title;
    document.getElementById('description').value = storageObjectArr[index].description;
    document.getElementById('responsible').value = storageObjectArr[index].responsible;
    document.getElementById('taskTime').value = storageObjectArr[index].taskTime;
    $('#addItemsModal').modal('show');
    storageObjectArr.splice(index, 1);
    localStorage.setItem('taskStorage', JSON.stringify(storageObjectArr));
    writingOnPageFromStorage();
  }
}
```

Explanation:

- The editTask(index) function is responsible for allowing users to edit an existing task. It takes an index parameter, which specifies the index of the task to be edited.

- It begins by retrieving the task data stored in localStorage using localStorage.getItem('taskStorage') and stores it in the storageObject variable.

- Next, it checks if storageObject is not null, not undefined, and not an empty string using the if statement. This ensures that there is valid task data in localStorage to work with.

- If the condition is met, the function proceeds to edit the task:

- It parses the JSON data in storageObject into an array called storageObjectArr to work with the task list.

- It populates the modal form fields with the details of the selected task by setting the value property of the corresponding HTML input elements (document.getElementById('title'), document.getElementById('description'), document.getElementById('responsible'), and document.getElementById('taskTime')) with the values from the selected task in storageObjectArr.

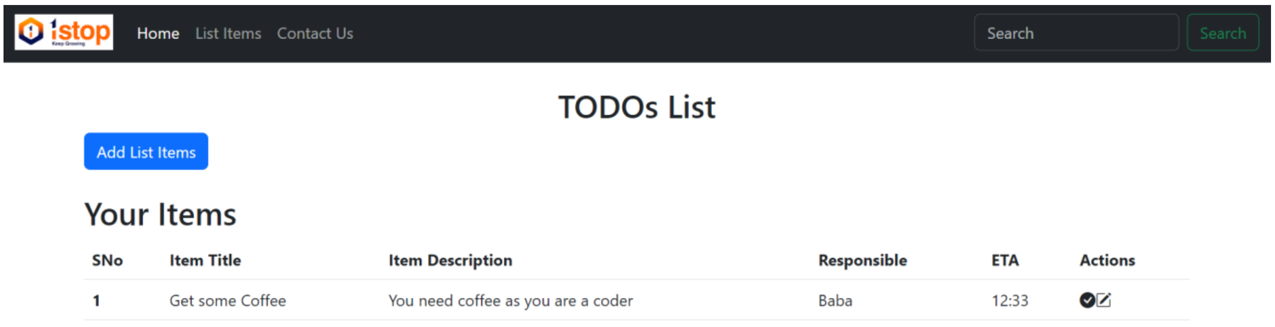
- The modal is then displayed by calling $('#addItemsModal').modal('show'), allowing the user to edit the task details.

- The edited task is removed from storageObjectArr using the splice() method to ensure it's not duplicated.

- The updated storageObjectArr is stored back in localStorage using localStorage.setItem('taskStorage', JSON.stringify(storageObjectArr)).

- Finally, the writingOnPageFromStorage() function is called to refresh the task list display on the web page, ensuring that the edited task details are updated in the list.

UI of the To-Do List:

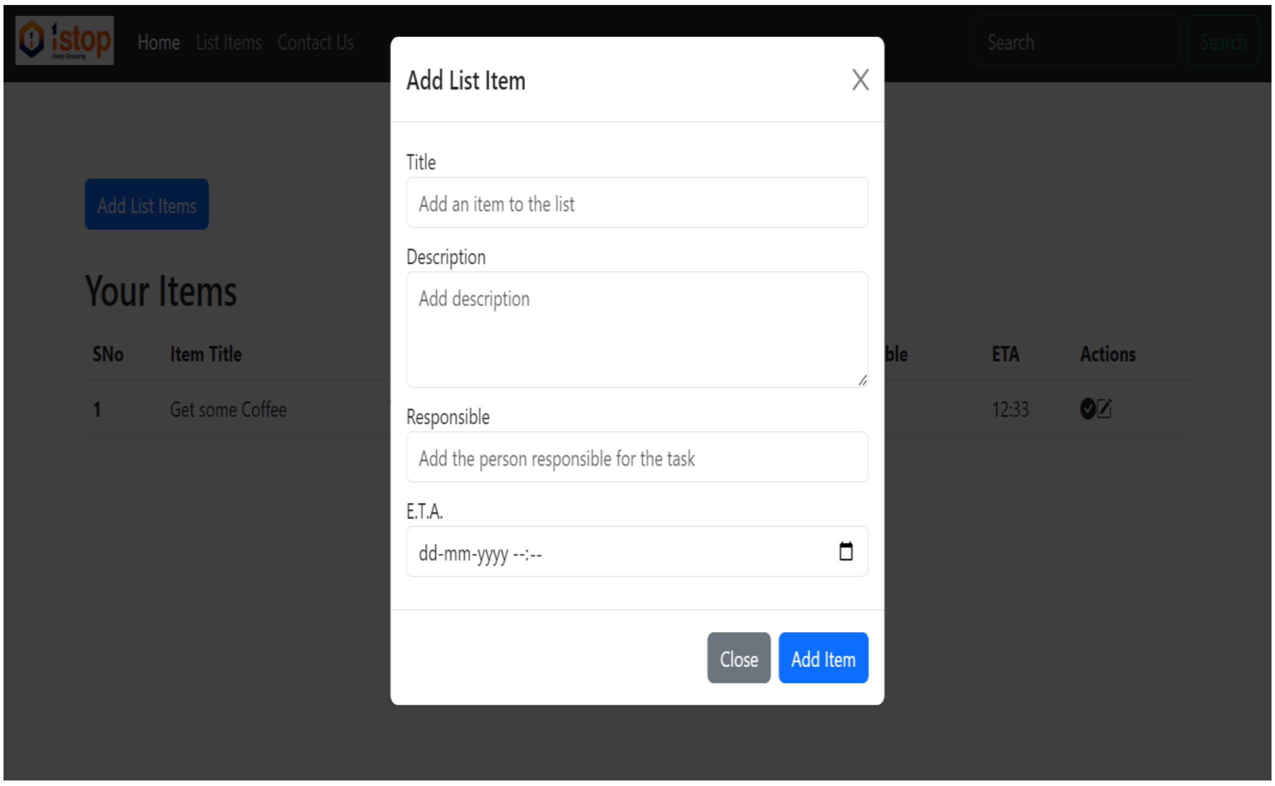


TODOs List

Your Items

SNo	Item Title	Item Description	Responsible	ETA	Actions
1	Get some Coffee	You need coffee as you are a coder	Baba	12:33	

Add Items Modal:



Home List Items Contact Us

Search

Add List Items

Your Items

Add List Item

Title

Add an item to the list

Description

Add description

Responsible

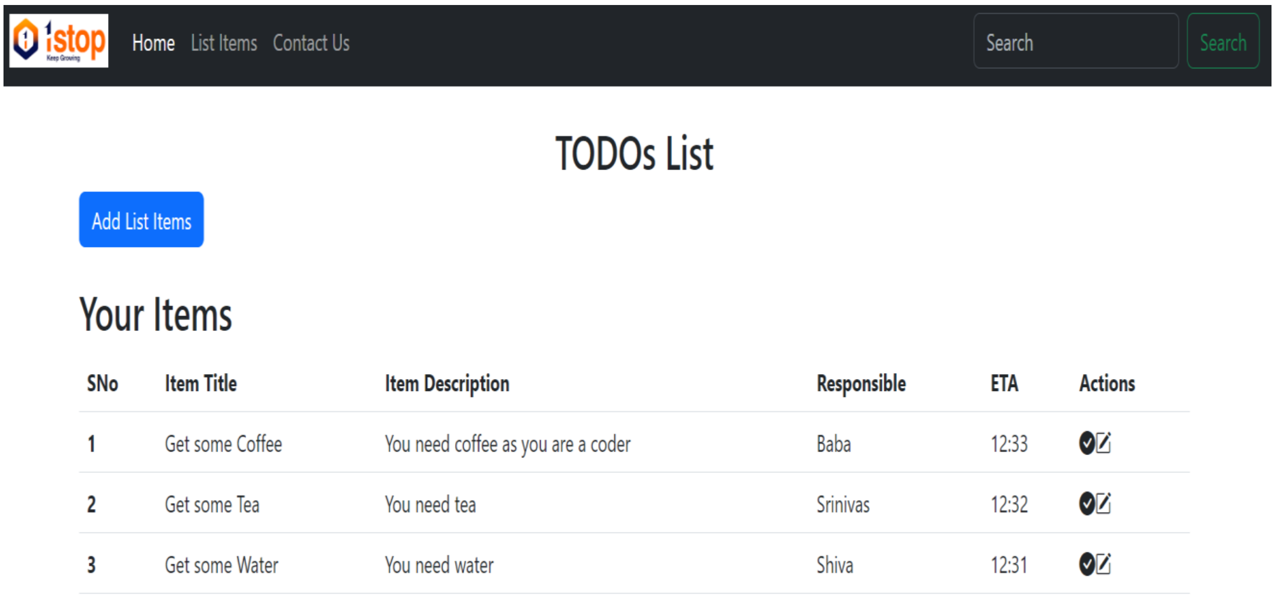
Add the person responsible for the task

E.T.A.

dd-mm-yyyy --:--

Close Add Item

Updated List on table:



Home List Items Contact Us

Search

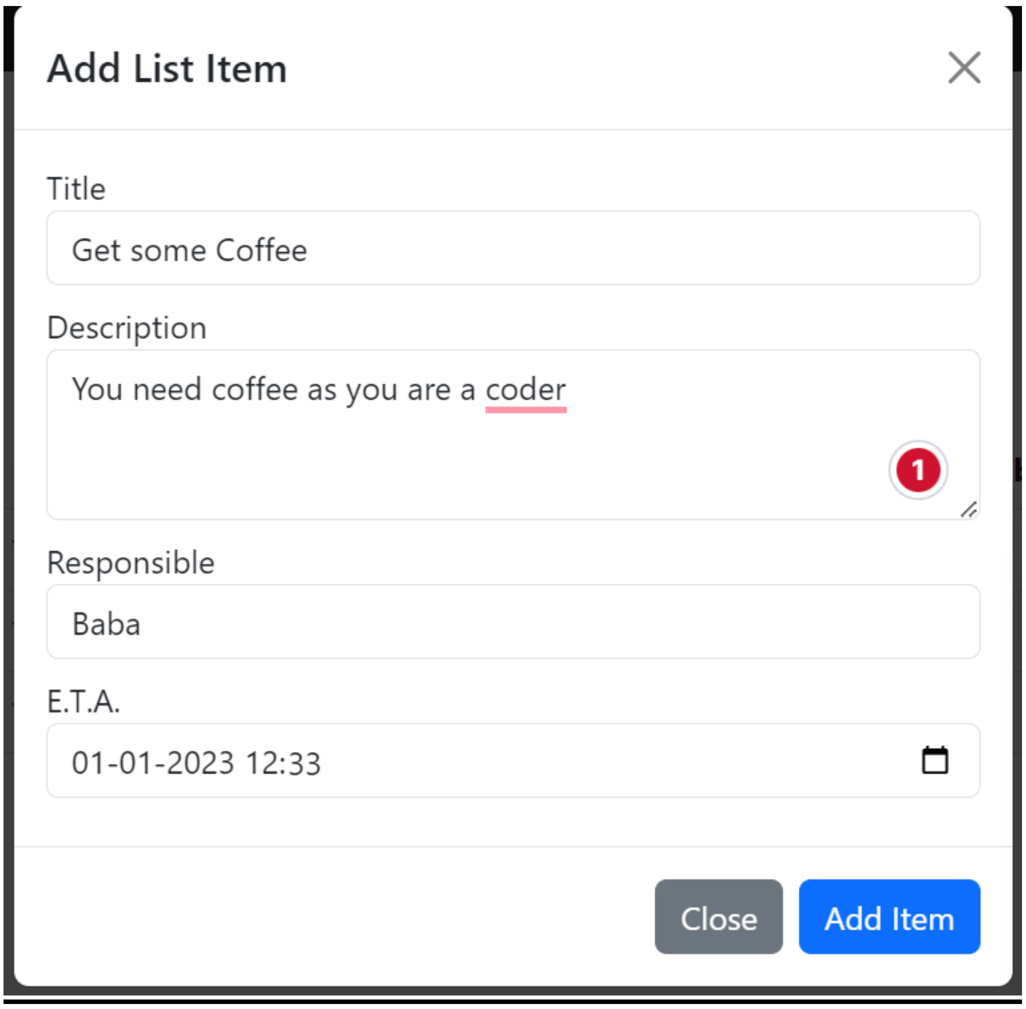
TODOs List

Add List Items

Your Items

SNo	Item Title	Item Description	Responsible	ETA	Actions
1	Get some Coffee	You need coffee as you are a coder	Baba	12:33	
2	Get some Tea	You need tea	Srinivas	12:32	
3	Get some Water	You need water	Shiva	12:31	

Popped up modal on clicking edit button:



Add List Item

Title

Get some Coffee

Description

You need coffee as you are a coder

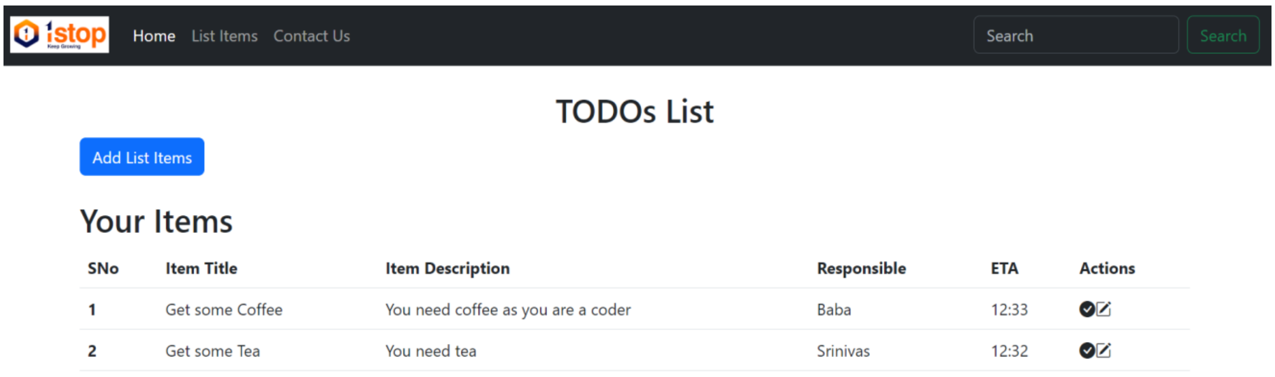
Responsible

Baba

E.T.A.

01-01-2023 12:33

Removed element from table on completing:



TODOs List

Your Items

SNo	Item Title	Item Description	Responsible	ETA	Actions
1	Get some Coffee	You need coffee as you are a coder	Baba	12:33	✓✎
2	Get some Tea	You need tea	Srinivas	12:32	✓✎

VI. Conclusion:

In this report, we have examined a web application designed for managing a To-Do List using HTML, Bootstrap, and JavaScript. The application provides a user-friendly interface for adding, viewing, editing, and marking tasks as done. Here are the key takeaways and concluding remarks:

Objective Achievement:

The primary objective of the web application was to create a functional To-Do List that allows users to efficiently manage their tasks. This objective has been successfully achieved through the implementation of key features such as task addition, editing, and marking as done.

User-Friendly Interface:

The application offers an intuitive and visually appealing interface, thanks to the use of Bootstrap for styling. The navigation bar provides easy access to different sections of the application, including task management and search.

Task Management:

Users can easily add new tasks through a modal dialog, providing details like task title, description, responsible person, and task time. Tasks are stored in the browser's localStorage, ensuring persistence across sessions.

Task Display and Interaction:

The application dynamically displays tasks from localStorage in a tabular format, allowing users to view, edit, and mark tasks as done. Actions like editing and marking as done are seamlessly integrated and provide a smooth user experience.

Data Validation and Handling:

The JavaScript functions responsible for task management validate data and handle localStorage operations efficiently. Tasks are stored and retrieved in a structured manner.

Accessibility and Responsiveness:

The application follows best practices for accessibility, ensuring that it is usable by individuals with disabilities. Additionally, it is responsive, adapting to different screen sizes for a consistent user experience.

Future Enhancements:

While the current application fulfills its core objectives, future enhancements could include additional features such as task categorization, due date reminders, and user accounts for personalization.

In conclusion, this To-Do List web application serves as an effective tool for managing tasks and demonstrates the use of HTML, Bootstrap, and JavaScript for creating user-friendly and interactive web applications. Its simplicity and functionality make it a valuable tool for individuals seeking an efficient way to organize and track their tasks. Further improvements and feature additions can expand its utility and cater to a broader audience of task managers and organizers.