Internship Project Report

On

HTML To Do List

Submitted in partial fulfillment of the requirements of the internship

Submitted by

Sahil Deepak Pardhi

Domain

Web Development Front-end – Self Paced Intern

Associated With

1Stop

Sep, 2024

ABSTRACT

The HTML To-Do List project aims to create a simple and efficient task management system using modern web technologies. In today's fast-paced digital world, keeping track of tasks and priorities is essential for both personal and professional productivity. This project focuses on building an intuitive, single-page to-do list application with key features like task addition, deletion, and status tracking. By utilizing responsive design and interactive elements, the to-do list will offer a seamless user experience across all devices, enabling users to manage their tasks effectively from any location. This project will serve as a practical tool for organizing daily activities, improving productivity, and enhancing time management in a user-friendly digital format.

OBJECTIVE

The goal of this project is to develop a user-friendly, responsive, and adaptable single-page to-do list application that allows users to manage their tasks efficiently. By focusing on simplicity, ease of use, and seamless functionality, the project aims to provide a practical tool for tracking daily tasks, enhancing productivity, and improving time management. The to-do list will offer an interactive platform for users to add, update, and delete tasks in real-time, providing a streamlined experience that works across all devices. This project seeks to create a functional and engaging application that supports users in staying organized and on top of their responsibilities.

CONTENT

INTRODUCTION

1.1 Introduction

The **HTML To-Do List** project is designed to provide users with a practical, easy-to-use task management solution. In a world where organization and productivity are paramount, having an intuitive tool to manage daily tasks is essential. This project focuses on creating a single-page to-do list application using modern web technologies like HTML, CSS, JavaScript, and Bootstrap. The application enables users to add, edit, and delete tasks while keeping track of responsibilities and deadlines in a visually clear and responsive interface. The project leverages local storage to ensure task data persistence, allowing users to manage tasks even after refreshing the page. With its simple, interactive design and cross-device compatibility, this application serves as an effective tool for personal productivity and task management, helping users stay organized and efficient. This To-Do List project aims to offer an interactive and dynamic task management system, providing users with flexibility in tracking their tasks and deadlines.

1.1.1 HTML File - The HTML (Hypertext Markup Language) file defines the structure of the To-Do List web application. It contains elements such as a navigation bar, modal forms for adding and updating tasks, and a table that displays the tasks. Using HTML, we ensure that users can input tasks, edit, and mark them as completed. The modal forms provide a user-friendly way to manage tasks, while the table structure keeps all tasks neatly organized.

1.1.2 CSS File - In this project, Bootstrap CSS is used to style the To-Do List. The CSS ensures that the application has a consistent look with modern design elements like responsive grids, buttons, and modal windows. External libraries, such as FontAwesome and Bootstrap Icons, are incorporated to include visually appealing icons for actions like adding, editing, or completing tasks. The responsiveness is achieved using Bootstrap's grid system, ensuring the application adjusts well on different screen sizes**.**

1.1.3 JavaScript File - The JavaScript (JS) adds dynamic functionality to the To-Do List. It manages interactions like adding new tasks, editing existing ones, and removing completed tasks using local storage to persist the data. Key functionalities include creating tasks from the input forms, displaying the tasks dynamically in the table, and updating or removing tasks with a click. This makes the application interactive and user-friendly by allowing real-time updates without page reloads.

1.1.4 Bootstrap File - The Bootstrap file provides a ready-to-use framework for developing responsive and mobile-first web applications. Bootstrap components such as the navigation bar, buttons, modals, and table are utilized in the To-Do List to enhance the design and functionality without writing custom CSS from scratch. With Bootstrap's built-in classes, the web app achieves a clean, modern look while ensuring full responsiveness.

1.1.5 Responsive Web Designing - By utilizing Bootstrap's responsive grid system, this To-Do List ensures an optimal viewing experience across various devices. The layout automatically adjusts to different screen sizes, from desktops to mobile phones. All elements such as the task table and modals are designed to maintain usability and readability regardless of the device being used.

This project aims to combine these core technologies to develop a dynamic To-Do List web application that efficiently helps users manage their tasks. By focusing on simplicity, usability, and visual appeal, the application will serve as a practical tool for task management and productivity. The final product will not only provide users with a clear overview of their pending tasks but also offer an engaging and interactive experience, making task tracking and organization seamless and efficient.

METHODOLOGY

2.1 Introduction

This methodology outlines the comprehensive approach to designing and developing a dynamic To-Do List web application. The application aims to help users efficiently manage their tasks, including adding, editing, and tracking task progress, in a simple yet effective manner. The methodology covers every phase of the development process, from planning and design to implementation and deployment, ensuring a well-structured and user-friendly task management tool.

2.2 Overall Description

The To-Do List web application is designed to enable users to efficiently manage their tasks through a clean and intuitive interface. Key functionalities include adding new tasks, editing existing tasks, deleting them, and tracking task completion status. The application is fully responsive, ensuring that it provides a seamless experience across different devices and screen sizes. Built with modern web technologies such as HTML, CSS, JavaScript, and Bootstrap, the application prioritizes functionality, user experience, and performance, offering a robust task management tool for everyday use.

2.3 Software Development Model

The development of the HTML To-Do List project follows the Incremental Process Model. This approach enables the project to be built and delivered in stages, with each increment introducing specific features or improvements. The iterative nature of this model ensures the To-Do List can be reviewed and improved throughout the development process. Early issue detection is possible, and user feedback can be incorporated at each stage to refine the outcome.

2.3.1 Increment 1: Home Page and Basic Functionality

2.3.1.1 Communication

Home Page:

- The home page serves as the landing page for the To-Do List web application. It introduces the application and provides users with an input field to add new tasks, a button to submit tasks, and a list that displays the added tasks.
- The design of the home page will focus on simplicity and usability, ensuring that users can quickly understand how to interact with the to-do list.

2.3.1.2 Planning

- The layout will consist of a header, an input form, and a task list area. The form will have a text field for entering the to-do item and a button to submit the task.
- The task list will display the tasks below the form, with options to mark them as complete or delete them.
- Risks include issues with cross-browser compatibility, as well as performance challenges if many tasks are added. These risks will be addressed through testing on multiple platforms.

2.3.1.3 Modeling

This is one the important phases as the architecture of the system is designed in this phase. Analysis is carried out and depending on the analysis a software model is designed. Different models for developing software are created depending on the requirements gathered in the first phase and the planning done in the second phase.

Analysis: For the To-Do List application, we will follow the Incremental Model. Each increment adds new functionality, building upon the last. We will create a wireframe to define the placement of elements such as the input field, submit button, and task list.

Design:

- The HTML will use semantic tags to structure the form and task list. The input field will be created using an <input> tag, and the task list will be a <ul> (unordered list).
- CSS will be used to style the page, ensuring consistency and responsiveness across different screen sizes.

2.3.1.4 Construction

The actual coding of the software is done in this phase. This coding is done on the basis of the model designed in the modeling phase. So in this phase software is actually developed and tested.

Code:

- Write the HTML for the home page, focusing on semantic structure. Ensure the input form and task list are easily accessible.
- Implement CSS to style the home page, using modern design principles such as a clean, minimalistic layout.
- Use JavaScript to handle the addition of tasks to the list, allowing users to dynamically add tasks without refreshing the page.

Test: .

- Test the functionality of the form by adding multiple tasks and ensuring they appear in the task list.
- Ensure the page is responsive on different devices (mobile, tablet, desktop).
- Verify the compatibility of the home page on major browsers.

Deployment:

- Deploy the home page to a test environment for feedback. Once tested and approved, deploy it to production.

2.3.2 Increment 2: Task Completion and Deletion

2.3.2.1 Communication

Task Completion and Deletion:

- The second increment introduces features that allow users to mark tasks as complete or delete them from the list.
- Users will be able to check off completed tasks, visually indicating their status, or remove tasks they no longer need.

2.3.2.2 Planning

- Plan the content for task management, including the addition of a checkbox next to each task to mark it as complete.
- Include a delete button for each task to allow users to remove tasks. Use icons (such as checkmarks and trash bins) to represent actions visually.

2.3.2.3 Modeling

This is one of the important phases as the architecture of the system is designed in this phase. Analysis is carried out and depending on the analysis a software model is designed. Different

models for developing software are created depending on the requirements gathered in the first phase and the planning done in the second phase.
Analysis: Using the Incremental Model, this increment will build upon the basic functionality created in the first phase.

Design:

- Design a wireframe that incorporates checkboxes and delete buttons next to each task in the list.
- Develop HTML to include these elements and style them using CSS.
- Ensure that this is responsive, with the layout adjusting appropriately for different screen sizes.

2.3.2.4 Construction

The actual coding of the skill section is done in this phase. This coding is done on the basis of the model designed in the modeling phase. So in this phase task deletion and comletion section is actually developed and tested.
Code:

- Update the HTML to add a checkbox for each task and a delete button.
- Use JavaScript to handle task completion (striking through completed tasks) and deletion (removing tasks from the list).
- Apply CSS to visually differentiate between completed and incomplete tasks.

Test:

- Test the completion and deletion functionality across different browsers and devices.
- Ensure that completed tasks remain visually marked after being checked off, and that deleted tasks are removed from the list.

2.3.2.5 Deployment

- Deploy the updated task list functionality to a test environment, gather feedback, and then move to production after ensuring all issues are resolved.

2.3.3 Increment 3: Task Editing and Saving

2.3.3.1 Communication

Task Editing and Saving:

- This increment adds the ability to edit tasks in the list and save the updated versions. Users will be able to change task details and save those changes.

2.3.3.2 Planning

- Plan the editing feature, including the UI components needed (such as an "Edit" button and an input field for task updates).
- Ensure the feature is intuitive and doesn't disrupt the user's workflow. Consider how changes will be saved to ensure task continuity.

2.3.3.3 Modeling

This is one of the important phases as the architecture of the system is designed in this phase. Analysis is carried out and depending on the analysis a software model is designed. Different models for developing software are created depending on the requirements gathered in the second phase and the planning done in the third phase.

Analysis: Each increment builds on the last. In this case, editing and saving features are layered on top of the core task management features.

.Design:

- Update the wireframe to include an "Edit" button next to each task, allowing the user to modify task text directly in the task list.
- Ensure that tasks can be saved or canceled, maintaining a clear and user-friendly interface.

2.3.3.4 Construction

The real coding of the Project Section is going to be done in this phase. This coding is done on the basis of the model designed in the modeling phase. So in this phase the project section is actually developed and tested.

Code:

- Write JavaScript to allow tasks to be edited. When the "Edit" button is clicked, the task text becomes editable. Once the changes are made, the task is updated in the list.
- Apply CSS to tasks being edited.

Test:

- Test the task editing feature to ensure that changes are saved correctly and that tasks remain functional after editing.

- Test the UI on multiple devices and browsers to ensure the edit feature is smooth and responsive.

2.3.3.5 Deployment

- Deploy the task editing functionality to the staging environment. Gather user feedback and resolve any issues before moving to production.

With each increment building upon the previous one, the HTML to-do list application gradually evolves into a sleek and efficient tool for task management. The development process marks just the beginning; ongoing user feedback will drive continuous enhancements and updates, ensuring the application remains functional and relevant. As users' needs and workflows change, this to-do list will adapt by integrating new features and optimizing existing ones to meet the demands of modern productivity. By remaining responsive to emerging trends and user requirements, this application will empower individuals to take charge of their tasks and responsibilities effectively. It will serve as a practical solution that not only organizes daily activities but also reflects the user's growth and accomplishments over time. Together, we will create a digital tool that not only fulfills current needs but also lays the groundwork for future productivity and success.

.

2.4 Requirement

2.4.1 Functional Requirements

2.4.1.1 User Interface:

- The HTML to-do list application should have a clean and intuitive design that enhances user experience and allows easy task management.

2.4.1.2 Responsiveness:

- The application must be responsive, ensuring that it works seamlessly across various devices, including desktops, tablets, and smartphones.

2.4.1.3 Task Management:

- Users should be able to add, edit, and delete tasks easily.
- Each task should include options for setting due dates, priority levels, and categories/tags.

2.4.2 Non-functional Requirements

2.4.2.1 Performance:

- The application should load quickly, with minimal delays during task addition, deletion, and updates to ensure a smooth user experience.

2.4.2.2 Security:

- Ensure the website is secure and free from vulnerabilities.

2.4.2.3 Accessibility:

- The website should adhere to accessibility standards, ensuring it is usable by people with disabilities.

2.4.2.4 Scalability:

- The website should be able to handle an increasing number of visitors and content.

Code and Output

3.1 Code

- **Files:**

3.1.1. index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,  initial-scale=1.0">
   <title>To Do List</title>
   <link rel="stylesheet" href="vendor/bootstrap/css/bootstrap.css">
   <link rel="stylesheet" href="vendor/fontawesome/css/all.css">
   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.11.3/font/bootstrap-icons.min.css">
</head>
<body>
   <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="container-fluid">
         <a href="#" class="navbar-brand">
            <img src="assets/logo-design.jfif" class="img-fluid" alt="logo" width="150">
         </a>
         <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>
```

```
<div class="container p-5">
      <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</th>
                        <th>Action</th>
                     </thead>
                     <tbody class="text-center" id="taskTableBody">

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

            </div>

         </div>
   </div>
<div class="modal fade" id="addTaskModal" data-bs-backdrop="static" data-bs-
keyboard="false" tabindex="-1" aria-labelledby="addTaskModalLabel" aria-hidden="true">
      <form id="taskInputFrom">
         <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/Description"></textarea>
                  </div>
                  <div class="mb-1">
                     <label for="addResponsiblePerson" class="form-
label">Responsible</label>
                     <input type="text" class="form-control" id="addResponsiblePerson"
name="taskResponsiblePerson" placeholder="Add 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" 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
Tasks</button>
            </div>
        </div>
    </div>

    </form>
  </div>
  <div class="modal fade" id="updateTaskModal" data-bs-backdrop="static" data-bs-
keyboard="false" tabindex="-1" aria-labelledby="updateTaskModalLabel" aria-
hidden="true">
    <form id="taskUpdatetFrom">
      <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 Task/Description"></textarea>
            </div>
<div class="mb-1">
              <label for="addResponsiblePerson" class="form-
label">Responsible</label>
              <input type="text" class="form-control" id="editResponsiblePerson"
name="taskResponsiblePerson" placeholder="Add 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" placeholder="Click to add 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()">Edit
Task</button>
                </div>
            </div>
        </div>

    </form>
  </div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.1/jquery.min.js"
    integrity="sha512-
v2CJ7UaYy4JwqLDIrZUI/4hqeoQieOmAZNXBeQyjo21dadnwR+8ZaIJVT8EE2iyI61OV8
e6M8PP2/4hpQINQ/g=="
    crossorigin="anonymous" referrerpolicy="no-referrer"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
    crossorigin="anonymous"></script>
  <script src="vendor/bootstrap/js/bootstrap.js"></script>
  <script>
    createHtmlFromStorage()

    function showAddTaskModal(){
      $('#addTaskModal').modal('show')
    }

    function addTask(){
      console.log("Add task")
      $('#addTaskModal').modal('hide')
      var dataArr = $('#taskInputFrom').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

      }

      if(storageObject != null && storageObject != undefined && storageObject != ''){
        storageObjectArr = JSON.parse(storageObject)
        storageObjectArr.push(taskObject)
      }
      else{
        storageObjectArr.push(taskObject)
      }
```

```
        localStorage.setItem('taskStorage',JSON.stringify(storageObjectArr))
            createHtmlFromStorage()
        }

        function createHtmlFromStorage(){
            var storageObjectArr = []
            var storageObject = localStorage.getItem('taskStorage');
            var storageObjectArr = JSON.parse(storageObject)
            var html = '';
            console.log(storageObjectArr)

            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)) +'</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</td></tr>'
                }

            }

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

function markAsDone(index){
            console.log(index)

            var storageObjectArr = []
            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))
            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(index){
        $("#updateTaskModal").modal('hide')
        var dataArr = $('#taskUpdatetFrom').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

        }

        if(storageObject != null && storageObject != undefined && storageObject != ''){
            storageObjectArr = JSON.parse(storageObject)
            storageObjectArr[taskObject['taskIndex']] = taskObject
        }

        localStorage.setItem('taskStorage',JSON.stringify(storageObjectArr))
        createHtmlFromStorage()
    }
  </script>
</body>

</html>
```

3.2 Outputs

1. Desktop Design –

1.1 Add Task Design

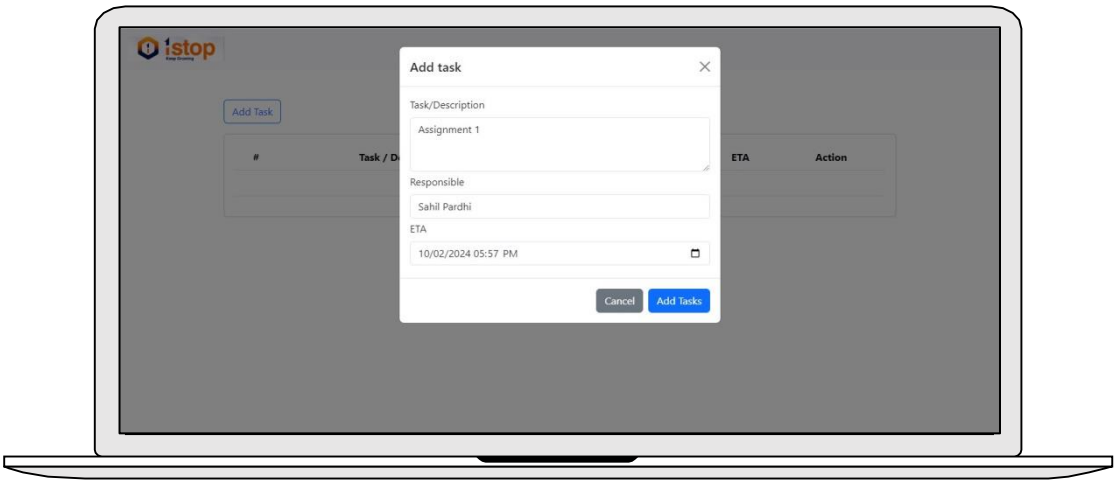

Fig. 3.2.1 Add Task Design

1.2 Task Added Design

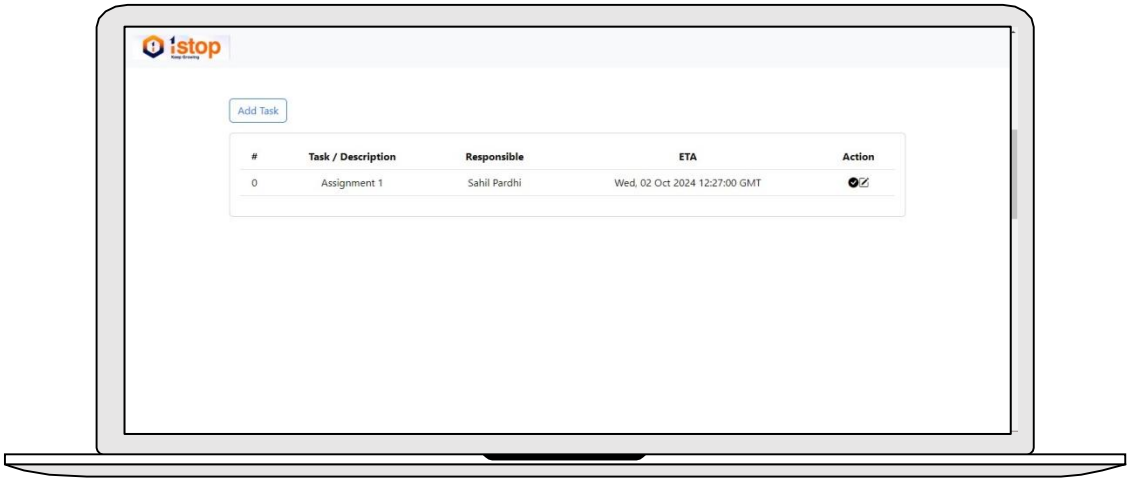

Fig. 3.2.2 Task Added Design

1.3 Edit Task Design

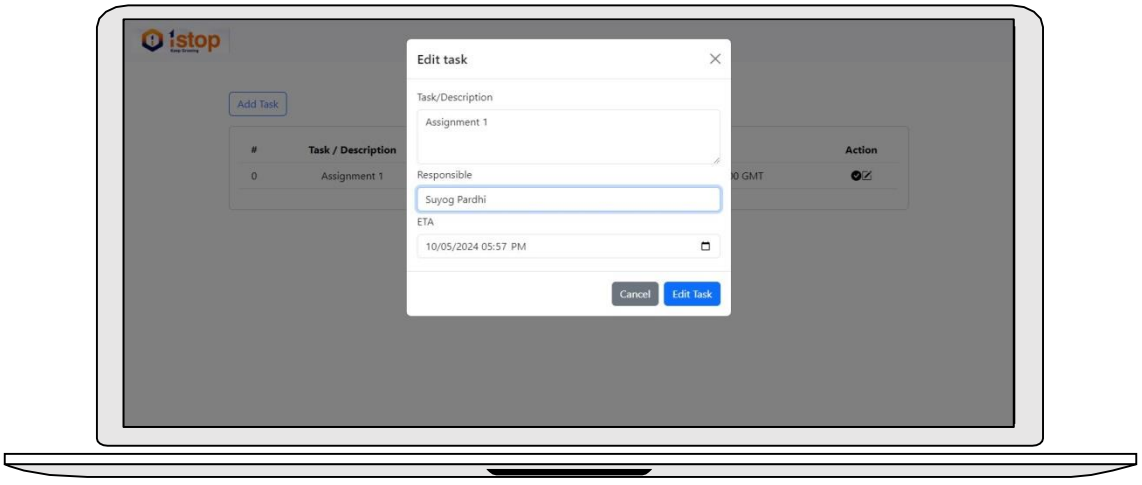

Fig. 3.2.3 Edit Task Design

1.4 Task Edited

Fig. 3.2.4 Task Edited

1.5 Tasks

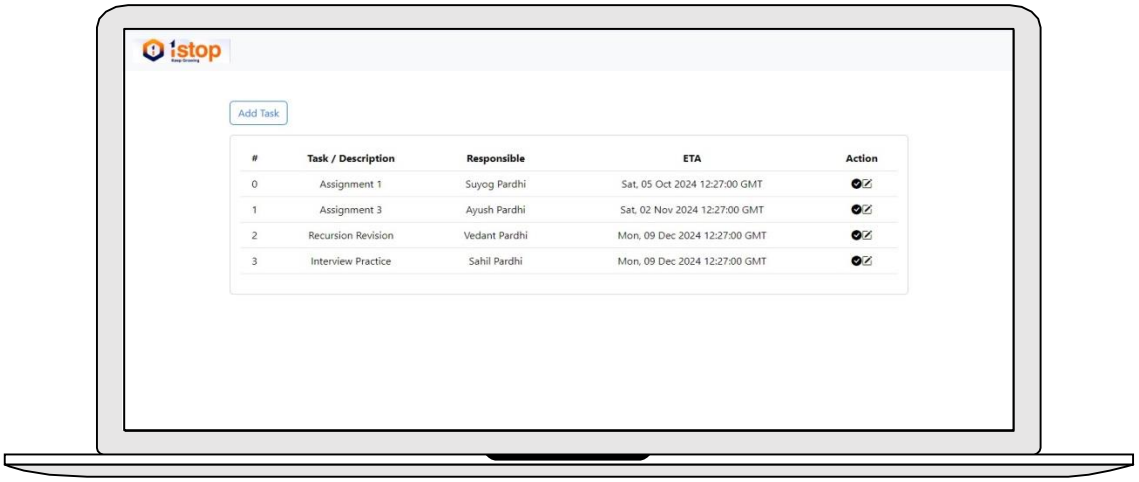

Fig. 3.2.4 Tasks

1.6 Tasks Completed

Fig. 3.2.4 Tasks

Conclusion

4.1 Conclusion

Creating a single-page HTML to-do list application is not merely a technical task; it’s an exercise in fostering productivity and enhancing time management through effective digital tools. Each phase of development—from initial concept and design to final implementation and user testing—plays a vital role in delivering a user-friendly experience that empowers individuals to organize their tasks efficiently. The careful integration of features such as task addition, editing, deletion, and categorization ensures that the application is not only functional but also aligns with users' needs and preferences.

Moreover, the development of the to-do list application is an iterative process. As user feedback is collected, the application can be refined and improved, incorporating new functionalities and optimizing existing features. This commitment to continuous improvement guarantees that the application remains relevant and valuable in an ever-evolving technological landscape.

Ultimately, this project is more than just a simple coding endeavor; it represents a dedication to enhancing personal productivity and enabling users to take control of their tasks and responsibilities. By creating a tool that adapts to individual workflows and preferences, this to-do list application embodies a practical solution that can positively impact users' daily lives, inspiring efficiency and organization.