To-Do List Application

UI/UX Design Fundamentals - Christ University

Submitted By:
Team Members:-

Full Name & Roll Number:
ALLAN PREM VARGHESE-2462030
ABEL FRANCIS-2462005

College Email id:
allan.prem@btech.christuniversity.in
abel.francis@btech.christuniversity.in

Instructor Name:
Ms. Nagaveena

Index:

1. Abstract:

This project is a simple, dynamic To-Do List Application designed to help users manage their tasks. It features the ability to add, complete, edit, and delete tasks, along with a filtering mechanism to view all, active, or completed tasks. The application leverages client-side storage (Local Storage) to persist tasks across sessions, making it a fully functional, self-contained utility. The interface is clean, modern, and responsive, thanks to the integration of the Bootstrap 5 framework.

2.Objectives:

- ➔ To build a fully functional web-based To-Do List application using vanilla JavaScript (via jQuery).
- ➔ To implement CRUD (Create, Read, Update, Delete) operations for managing tasks.
- ➔ To utilize Local Storage for task persistence.
- ➔ To provide a clean and responsive User Interface (UI) using Bootstrap 5.
- ➔ To include filtering capabilities to display tasks based on their completion status.

3. Scope of the Project:

The scope of this project is to implement a client-side To-Do List application. It covers:

- Task input and creation.
- Displaying the list of tasks.
- Marking tasks as complete or incomplete.
- Editing and deleting individual tasks.
- Filtering the displayed list by status ("All," "Active," "Completed").
- Saving and loading the task list to/from the user's browser (Local Storage).

It does not include:

- A backend database or user authentication.
- Advanced features like drag-and-drop sorting or task categories.

4. Tools & Technologies Used:

Tool/Technology	Purpose
jQuery	Simplified DOM traversal and manipulation, event handling.
Bootstrap	Responsive design, pre-built components (Card, Form, Buttons, List Group).
Web Local Storage	Storing and retrieving the task array on the client-side.
HTML5, CSS3, JavaScript	Core structure, styling, and application logic.

5. HTML Structure Overview:

- The structure utilizes Bootstrap's grid and components for a centralized, modern layout:

- Page Wrapper: A primary centers the application and provides vertical margin.

- Card Component: The entire application resides within a to give it a distinct, elevated look, complete with a primary-colored header.

- Input Area: An Input Group is used for the text input (#taskInput) and the Add Task button (#addTaskBtn).

- Filter Buttons: A set of three buttons (.filter-btn) is provided for filtering, using Bootstrap's outline button styles and a data-filter attribute to manage state in JavaScript.

- Task List: The tasks are rendered inside an unordered list element, <ul class="list-group" id="taskList">, where each task will be an <li> element styled as a Bootstrap List Group Item.

6. CSS Styling Strategy:

- The custom CSS is minimal, primarily leveraging the extensive styling provided by Bootstrap.

- Body Styling: Sets a light background (#f4f7fc) and uses a modern, system-font stack (font-family: system-ui).

- Task Completed Style: The core custom style, .task-completed, is a class applied by JavaScript to completed task text, providing a clear visual cue with text-decoration: line-through and color: gray.

- Action Button Spacing: A small adjustment for margin between the Edit and Delete buttons (.task-actions button).

- Active Filter Reinforcement: Adds a light blue box-shadow to the active filter button (.filter-btn.active) to make its selection visually unambiguous, complementing Bootstrap's default active styling.

7. Key Features:

- Task CRUD: Full functionality to Create, Read (render), Update (edit/toggle complete), and Delete tasks.

- Persistent Storage: Tasks are stored in the browser's Local Storage under the key "todo_tasks_v1", ensuring they remain available even after the browser is closed and reopened.

- Dynamic Rendering: The renderTasks() function dynamically generates the list HTML based on the current state of the tasks array, ensuring the UI always reflects the data.

- Toggle Completion: Users can mark tasks as complete via a checkbox, which applies the .task-completed style.

- Task Filtering: Users can click "All," "Active," or "Completed" filter buttons to immediately update the view.

- Keyboard Support: Pressing the Enter key in the input field adds a new task.

- Client-Side Task Editing: Editing is handled via the native JavaScript prompt() function for quick text modification.

8. Challenges Faced & Solutions:

Challenge	Solution
Identifying the correct task to modify.	Each task object is given a unique `id` (`Date.now()`). This `id` is stored on the list item using the `data-id` attribute, allowing event handlers to reliably find the corresponding object in the `tasks` array using `$(this).closest("li").data("id")`.
Efficiently updating the UI after every change.	The `renderTasks()` function is called after *every* change (add, delete, edit, filter). It clears the list and regenerates the entire view, ensuring all state changes (including filters) are accurately reflected.
Handling events for dynamically added elements.	jQuery's `$(document).on(event, selector, handler)` syntax is used for all dynamic elements (checkboxes, edit/delete buttons). This event delegation ensures that listeners work for elements that are added to the DOM *after* the initial page load.

9. Outcome:

- The result is a reliable, easy-to-use To-Do List application. It provides a good demonstration of using jQuery for dynamic front-end logic and Bootstrap for a professional, responsive presentation, coupled with Local Storage for data persistence. The code is modular, with clear functions for each key operation (`addTask`, `setTaskCompleted`, `deleteTask`, etc.), making it clean and maintainable.

10. Future Enhancements:

- **Task Reordering:** Implement drag-and-drop functionality (using a library like jQuery UI Sortable or a modern equivalent) to allow users to prioritize tasks visually.

- **Task Priority/Due Dates:** Add fields for assigning a priority level (e.g., high, medium, low) or a due date.

- **Better Editing UI:** Replace the native `prompt()` with a more integrated, in-line editing input field for a smoother user experience.

- **Code Modernization:** Refactor the JavaScript to use native ES6+ features (e.g., `fetch`, template literals, modern DOM manipulation) instead of relying solely on jQuery, aligning with current web development best practices.

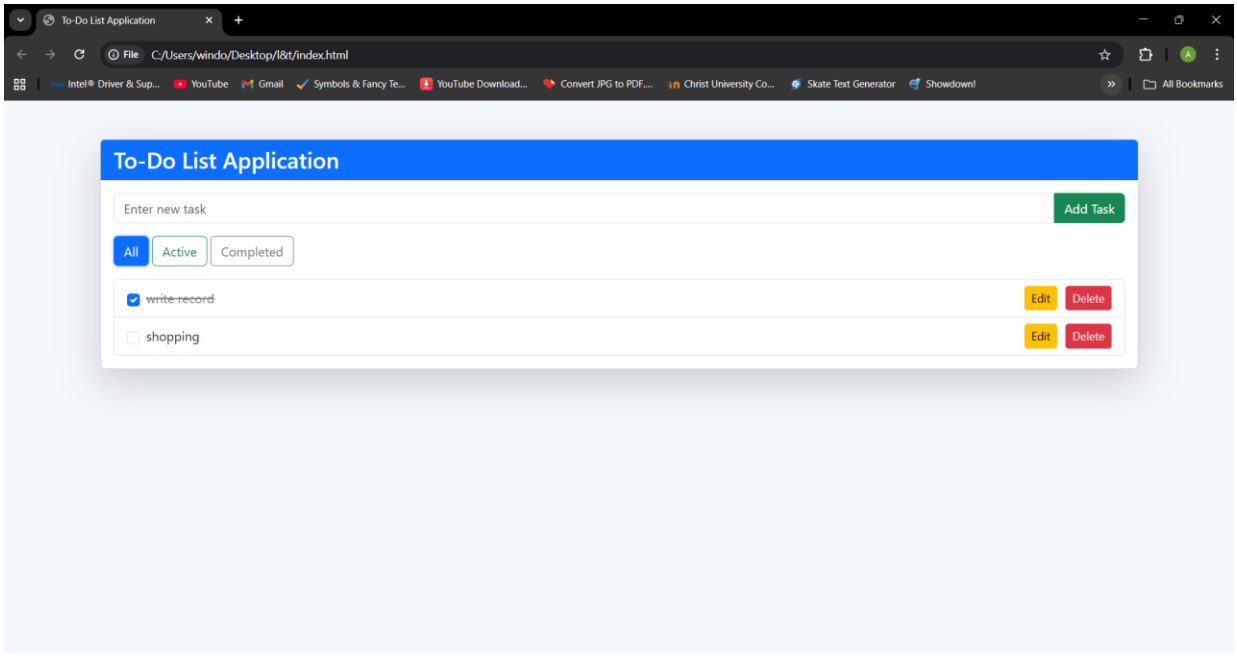
11.Sample Code:

This is a sample code not the actual code:-

```
// Render list according to tasks[] and currentFilter
function renderTasks() {
  const $list = $("#taskList").empty();

  tasks.forEach((task) => {
    // apply filter
    if (currentFilter === "active" && task.completed) return;
    if (currentFilter === "completed" && !task.completed) return;

    const $li = $(`
      <li class="list-group-item d-flex justify-content-between
align-items-center" data-id="${task.id}">
        <div class="d-flex align-items-center">
          <input type="checkbox" class="form-check-input me-2
toggle-complete">
          <span class="task-text"></span>
        </div>
        <div class="task-actions">
          <button class="btn btn-sm btn-warning edit-task">Edit</button>
          <button class="btn btn-sm btn-danger
delete-task">Delete</button>
```

```
        </div>
      </li>
    `);

    $li.find(".task-text").text(task.text).toggleClass("task-completed", task.completed);
    $li.find(".toggle-complete").prop("checked",  task.completed);

    $list.append($li);
  });
}
```

12. Screenshots of Final Output



13. Conclusion:

This project successfully delivers a robust, user-friendly To-Do List Application. By effectively combining the structural power of HTML5, the styling prowess of Bootstrap, and the dynamic capabilities of jQuery and Local Storage, it provides a solid foundation for a practical, client-side data management tool. It achieves all its core objectives and serves as an excellent example of essential front-end development principles.

14. References:

L&T LMS:

https://learn.lntedutech.com/Landing/MyCourse