Objective :

Primary objective of *To-Do List application* is to create a user-friendly application with clear visual cues that allows to manage tasks effectively. It includes categorizing tasks (total, pending, done, deleted), adding tasks with validation, providing a search functionality to easily find tasks.

Features within the application :

This application incorporates the below key features :

Task overview :

- Displays the counts for total, pending, done and deleted tasks. These counts are updated dynamically as per the user actions.

Adding Tasks :

- Input field for entering task names
- "Add Task" button, enabled only when the input field meets certain criteria:
    1. Task name must atleast have 10 characters
    2. Task name must not have any trailing and leading spaces
- After a task is added it is initially added to 'Pending' category
- After adding a task, the input field is emptied
- Uses "uuidv4" to generate unique IDs for each task
- Upon adding a task, success toast message is displayed

Searching Tasks :

- Search bar helps in filtering the tasks as per the necessity
- Filtering is not limited, applies to all categories i.e., total, pending, done, deleted

Task Management :

1. Pending/Done tasks -
    - Checkboxes to mark tasks as "Pending" or "Done"
    - Displays createdAt timestamp
    - Upon task completion/pending status change, displays success/warning messages
2. Deleted tasks -
    - "Delete" button moves tasks to delete section
    - Deleted tasks are shown with a ~~line-through~~ effect
    - Displays deletedAt timestamp
    - "Restore" button moves the tasks to "Pending" list

    - The deleted tasks also contribute to the total tasks

Data Persistence :

- All the tasks are saved to *localStorage* so they are preserved even after refresh sessions

Approaches used within the application :

To achieve each feature the below approaches were used :

Task overview :

- TaskStatistics recieves the tasks and deletedTasks as props and calculates count for each category through array methods like filter and length, to display the counts dynamically

Adding Tasks :

- AddTaskForm manages the input field and the "Add Task" button
- The **handleChange** function updates the state of taskName and calls **validInput** function to check if the input is valid, it sets **isButtonEnabled** state and **errorMessage** state based on the criteria rules
- The **handleAddTask** function adds a new task to the tasks array

Searching Tasks :

- The **filteredTasks** and **filteredDeletedTasks** are created using filter method and the includes method to check if the task name matches searchQuery and are rendered

Task Management :

1. Pending/Completed tasks - the **handleCheckBox** function updates the done status of a task, **handleDeleteTask** filters the array to remove deleted task and add the task to deletedTasks array
2. Deleted tasks - the **restoreDeleteTask** function filters the deletedTasks array to remove the restored task and adds it back to tasks array

Data persistence :

- The **useEffect** hook is used to save all the tasks to localStorage whenever they are updated such that data is persisted on refresh sessions.

Visual Cues :

- Tailwind CSS is used to style the task categories

Toast Notifications :

- To provide feedback on user-actions like adding, pending, done and deleted "react-toastify" library is used

Technical Memorandum :

- React functional components and react hooks (useState, useEffect, useCallBack) are used for state management
- Typescript ensures type safety
- uuid library is used for generating unique IDs for each task
- react-toastify library is used for toast notifications on user-actions
- moment library is used for providing a clean format for timestamps
- Tailwind CSS is used for styling