Step 1: Set Up the Project

1.1 Create a Project Folder

Create a folder for your project:

```
mkdir task-manager-ts
cd task-manager-ts
```

1.2 Initialize the Project

Initialize `npm` and install dependencies:

```
npm init -y
npm install typescript ts-node --save-dev
```

1.3 Create `tsconfig.json`

Generate a `tsconfig.json` file to configure TypeScript:

```
npx tsc --init
```

Modify the `tsconfig.json` file to include the following settings:

```
{
  "compilerOptions": {
        "target": "ES6",
        "module": "CommonJS",
        "strict": true,
        "outDir": "./dist",
        "rootDir": "./src"
  },
  "include": ["src/**/*"]
}
```

1.4 Folder Structure

Organize your project as follows:

```
/task-manager-ts
  ├── dist/      (Compiled JavaScript files)
  ├── src/
  │       ├── models/
  │       │       └── Task.ts
  │       ├── utils/
  │       │       └── Storage.ts
  │       └── index.ts
  ├── package.json
  └── tsconfig.json
```

Step 2: Define the Task Model

2.1 Create an Interface for Tasks

In `src/models/Task.ts`, define an interface for tasks:

```
export interface Task {
  id: string;
  title: string;
  completed: boolean;
}
```

2.2 Create a Task Class

Add a class to represent tasks:

```
export class TaskManager {
  private tasks: Task[] = [];

  // Add a task
  addTask(title: string): void {
        const newTask: Task = {
        id: Math.random().toString(36).substr(2, 9), // Generate a random ID
        title,
        completed: false,
        };
        this.tasks.push(newTask);
        console.log(`Task added: ${title}`);
  }
```

```
  // Mark a task as completed
  completeTask(id: string): void {
        const task = this.tasks.find((t) => t.id === id);
        if (task) {
        task.completed = true;
        console.log(`Task completed: ${task.title}`);
        } else {
        console.log("Task not found");
        }
  }

  // Delete a task
  deleteTask(id: string): void {
        const index = this.tasks.findIndex((t) => t.id === id);
        if (index !== -1) {
        const deletedTask = this.tasks.splice(index, 1)[0];
        console.log(`Task deleted: ${deletedTask.title}`);
        } else {
        console.log("Task not found");
        }
  }

  // Get all tasks
  getTasks(): Task[] {
        return this.tasks;
  }
}
```

Step 3: Add Utility for Local Storage

3.1 Create a Generic Storage Utility

In `src/utils/Storage.ts`, create a generic utility to save and load data from local storage:

```
export class Storage<T> {
  private key: string;

  constructor(key: string) {
        this.key = key;
  }
```

```
  // Save data to local storage
  save(data: T[]): void {
        localStorage.setItem(this.key, JSON.stringify(data));
  }

  // Load data from local storage
  load(): T[] {
        const data = localStorage.getItem(this.key);
        return data ? JSON.parse(data) : [];
  }
}
```

Step 4: Implement the Main Logic

4.1 Write the Main Application Code

In `src/index.ts`, implement the main logic:

```
import { TaskManager } from "./models/Task";
import { Storage } from "./utils/Storage";

// Initialize TaskManager and Storage
const taskManager = new TaskManager();
const taskStorage = new Storage<Task>("tasks");

// Load tasks from local storage
const savedTasks = taskStorage.load();
savedTasks.forEach((task) => taskManager.addTask(task.title));

// Example usage
taskManager.addTask("Learn TypeScript");
taskManager.addTask("Build a Task Manager App");
taskManager.completeTask(savedTasks[0]?.id || ""); // Complete the first task
taskManager.deleteTask(savedTasks[1]?.id || ""); // Delete the second task

// Save tasks to local storage
taskStorage.save(taskManager.getTasks());

// Log all tasks
console.log("All tasks:", taskManager.getTasks());
```

Step 5: Compile and Run the Code

5.1 Compile TypeScript to JavaScript

Run the TypeScript compiler to generate JavaScript files in the `dist` folder:

npx tsc

5.2 Run the Compiled Code

Use Node.js to run the compiled JavaScript:

node dist/index.js

Step 6: Test the App

1. Run the app using `node dist/index.js`.
2. Check the console output to verify that tasks are being added, completed, and deleted.
3. Open your browser's developer tools or use a tool like `localStorage` viewer to confirm that tasks are being saved to local storage.

Step 7: Extend the App

To make the app more interactive, you can integrate it with a frontend framework like React or Vue.js. Here's how you can extend it:

7.1 Add a Frontend

- Use HTML and JavaScript to create a UI for adding, completing, and deleting tasks.
- Fetch tasks from local storage and display them dynamically.

7.2 Use Generics for Reusability

- Extend the `Storage` class to handle other types of data (e.g., users, settings).

7.3 Add Validation

- Use TypeScript's type system to validate user input before adding tasks.

Key Takeaways

- **Classes** : Used to model tasks and manage their lifecycle.
- **Interfaces** : Defined the structure of tasks.
- **Generics** : Created reusable components like the `Storage` utility.
- **Local Storage** : Persisted tasks across sessions.
- **Compilation** : Compiled TypeScript to JavaScript for execution.