**Task Management App**

**Description:**

In this project, you will be building a Task Management App, a tool that allows users to add, view, and delete tasks. Each task will have a title, a due date, and a category. This project is built using React and TypeScript and it demonstrates concepts such as handling form inputs, maintaining application state, and implementing UI components.  
Live Demo: <https://64a5aad5ad2f6c0db9e48841--astonishing-flan-a9b9c0.netlify.app/>

**Project Setup:**

1. Open your terminal, navigate to the folder where you want to create your project, and type **npx create-vite task-manager --template react-ts**. This will create a new project folder with a React and TypeScript template.
2. Navigate into your new project folder with **cd task-manager**.
3. Install the necessary dependencies for this project. Run **npm install react-hook-form zod @hookform/resolvers zod**.
4. Also install the library/framework used for styling the componets (bootstrap, Tailwind CSS, Styled-components)

**Steps to Commit Changes to Git/GitHub in Visual Studio Code:**

1. Open Visual Studio Code.
2. Open your project folder (via File > Open Folder).
3. Click on the Source Control icon in the Activity Bar.
4. Click on Publish to GitHub and choose whether the repository should be public or private.
5. Select the files you want to include in the GitHub repository and click OK.
6. When you add new features or make changes, go to the Source Control tab, click Stage All (the + icon), add a commit message, and then click the dropdown arrow next to Commit. Select Commit and Push.
7. Confirm the successful push by checking your GitHub account.

**Coding Steps:**

**Step 1: Define the Task Model**

Define a TypeScript interface for a task in your App component file. Each task will have the following fields:

* **id**: a unique identifier for each task. This will be a number.
* **title**: a string that represents the task's name.
* **dueDate**: a Date object that represents the task's due date.
* **category**: a string that represents the task's category.

This interface will be used in the **useState** hook in the App component to type-check your tasks array (**useState<Task[]>([])**). It will also be used in the **TaskList** component, as it receives an array of tasks as a prop. You can store this Interface in a **Task.ts** file or store it in the files it will be used in.

**Step 2: File Structure**

1. Inside the **src** directory, create a new folder called **components**.
2. Inside the **components** folder, create four files: **TaskForm.tsx**, **TaskList.tsx**, **TaskFilter.tsx**, and **App.tsx**.
3. Create a new file in the **src** directory called **categories.ts**.
   1. const categories = ["Work", "Personal", "School"] as const;
   2. export default categories;

**Step 3: Implementing Components**

**TaskForm Component:**

This component is a form for the user to input new tasks. It includes fields for the title, due date, and category of the task.

1. Begin by importing the necessary dependencies and the categories file.
2. Define the schema for form validation using zod. It ensures that the title is a string between 3 and 50 characters, the due date is a valid date, and the category is a valid option from your categories list.
3. Define the **TaskFormData** type which is inferred from the zod schema.
4. Define your props interface for the component. This includes an **onSubmit** function which takes an argument of the **TaskFormData** type. This function is used to handle the form data on submission.
5. In the functional component **TaskForm**, use the **useForm** hook from **react-hook-form**. Pass your schema to the zodResolver to validate the form data. This will return various properties including the **register** function, **handleSubmit**, **reset**, and **formState**.
6. In the form element, handle the form submission with the **onSubmit** function passed as a prop. After the form is submitted, reset the form.
7. For each form field (title, due date, and category), render an appropriate input element:
8. For the title, render an input of type text. Use the **register** function from the **useForm** hook to bind this input to the form state. If there's a validation error, display the error message below this input.
9. For the due date, render an input of type date. Again, use the **register** function to bind this input to the form state. If there's a validation error, display the error message below this input.
10. For the category, render a select element with options for each category. Use the **register** function to bind this select to the form state. If there's a validation error, display the error message below this select.
11. Finally, render a submit button for the form.

**TaskList Component:**

This component is a table that lists all the current tasks.

1. Start by defining your **Task** and **Props** interfaces. The **Task** interface represents the shape of a task, and the **Props** interface represents the props that this component will receive: an array of tasks and a function to delete a task.
2. In the functional component **TaskList**, first handle the case where there are no tasks. If the tasks array is empty, render a paragraph saying "No tasks yet."
3. If there are tasks, render a table with a row for each task. The table should have a header with labels for title, due date, and category. Each row should display the task's title, due date, and category, as well as a delete button.
4. For each task, render a table row with cells for the title, due date, and category. The due date should be formatted to a local date string.
5. The last cell should contain a Delete button. When this button is clicked, it should call the **onDelete** function (passed as a prop) with the id of the current task.

**TaskFilter Component:**

This component is a select dropdown for the user to filter tasks by category.

1. Define your props interface. This component will receive a **onSelectCategory** function which takes a category string as its argument.
2. In the functional component **TaskFilter**, render a select element. When the selected option changes, call the **onSelectCategory** function with the new value.
3. The select element should contain an option for each category in your categories array. The value of each option should be the category string.

**Step 4: Implementing the App Component**

In the **App.tsx** file, create a functional component named **App**.

1. Define the initial state for the tasks array using the **useState** hook. The initial state is an empty array and the state value will be an array of tasks (**useState<Task[]>([])**).
2. Create a **handleAddTask** function to update the tasks state when a new task is added. This function should take a task object (minus the id “**task: Omit<Task, “id”>”)** as its argument, create a new task object with a unique id and the current date and time as the due date, and add the new task to the tasks array. Also, Adjust dueDate to start of day in local time zone **“task.dueDate.setHours(24);”**
3. Create a **handleDeleteTask** function to update the tasks state when a task is deleted. This function should take a task's id as its argument and remove the corresponding task from the tasks array.
4. In the return statement, render the **TaskForm** and **TaskList** components and pass the necessary props to them.

**\*Step 5: Bonus: Store and Retrieve tasks from localStorage\***

After implementing all of these, you will have a fully functioning Task Management App built with React and TypeScript. Remember to commit your changes after each step with appropriate commit messages to track your progress efficiently.

Finally, create a README.md file for this project. The README should contain a description of the project, instructions on how to set it up, Screenshot/Video and any other relevant information.

Happy coding!