EXPERIMENT 2

Aim

Build a Website based on React Hooks (useEffect, useContext, custom hooks)

Theory

Objective

The main objective of this project is to **design and implement a modern Task Manager** that goes beyond traditional keyboard/mouse input by introducing **voice-based control**. This enhances **accessibility** and makes the application more engaging and futuristic.

Specific objectives include:

- To use **React Hooks** (useState, useEffect, useContext, custom hooks) for state management, side effects, and reusability.
- To apply **Tailwind CSS** for responsive layouts, dark mode, and a clean UI without custom CSS.
- To integrate the **Web Speech API** for adding and managing tasks using voice commands.
- To implement a **custom hook (useSpeechRecognition)** for handling speech input and returning recognized commands.
- To store tasks persistently in **localStorage** using a reusable useLocalStorage hook.
- To provide a **progress bar** for visual tracking of task completion.
- To give **voice feedback** (e.g., "Task added successfully", "All tasks completed 🎉") for better user interaction.
- To ensure cross-device **responsiveness** (mobile, tablet, desktop) and accessibility.

Introduction

In modern web development, the focus has shifted towards building applications that are not just functional, but also accessible, responsive, and interactive. Traditional task management apps rely heavily on manual input, which may be less efficient and can limit accessibility for differently-abled users.

By leveraging the Web Speech API, developers can create voice-driven interfaces where users interact naturally using speech. This bridges the gap between humans and machines, making apps more inclusive and user-friendly.

React, with its Hooks system, provides a structured way to manage application state and side effects. Features like useContext and custom hooks enable cleaner code, global state sharing,

and reusability. Combined with Tailwind CSS, developers can rapidly prototype visually consistent, mobile-first UIs with built-in dark mode support.

This project — a Voice Controlled Task Manager — is a hands-on demonstration of integrating these technologies into a practical application. It combines manual task management (via forms and checkboxes) with voice-based task addition and feedback, includes real-time progress visualization via a progress bar, and enhances user engagement with spoken feedback messages.

By blending React Hooks, Tailwind CSS, and the Web Speech API, the project highlights how modern front-end tools can be combined to create applications that are accessible, intelligent, and responsive to real-world usage needs.

Prerequisites

1. **Basic Knowledge of React.js** – Understanding components, JSX, and props.
2. **React Hooks** – Familiarity with useState, useEffect, useContext, and custom hooks.
3. **JavaScript ES6+ Concepts** – Arrow functions, destructuring, template literals, etc.
4. **Tailwind CSS** – Knowledge of utility-first CSS classes for responsive and clean UI.
5. **Web Speech API** – Basic understanding of Speech Recognition and Speech Synthesis.
6. **LocalStorage** – To persist tasks across page reloads.
7. **VS Code / IDE Setup** – For writing and testing React code.
8. **Node.js and npm** – For project setup, package installation, and running the React app.

1. React Hooks

React Hooks eliminate the need for class components by allowing developers to use state and lifecycle features inside functional components.

useState → manages local state (e.g., tasks, theme, progress).
useEffect → runs side effects like syncing tasks with localStorage or updating the progress bar dynamically.
useContext → shares the global task list across multiple components/pages without prop drilling.
useReducer → organizes state updates using actions like ADD_TASK, TOGGLE_TASK, REMOVE_TASK.

2. Custom Hooks

Custom hooks encapsulate repetitive logic into reusable modules:

useLocalStorage → manages saving/loading tasks to browser storage.
useSpeechRecognition → wraps the Web Speech API and returns recognized commands.
useVoiceFeedback → uses the Speech Synthesis API to provide spoken messages to the user.

This modular structure makes the app extensible, readable, and reusable.

3. Tailwind CSS (Utility Classes)

Tailwind is used for styling and responsiveness:
Utility classes: bg-gray-800, text-white, rounded-lg.
Responsive design: sm:, md:, lg: breakpoints.
Dark mode: dark:bg-gray-900.
Interactive states: hover:bg-blue-600.

This removes the need for separate CSS files and ensures a consistent design system.

Key Outcomes

- Built a **responsive, multi-page task manager** with React Hooks and Tailwind CSS.
- Integrated **Web Speech API** for **voice-controlled task management**.
- Added **voice feedback** for motivation and accessibility.
- Implemented **progress tracking** with a visual **progress bar**.
- Used **Context API** for centralized state and **custom hooks** for reusability.
- Ensured **responsiveness and dark mode** across devices.

Learning Outcomes

- Learned how to **integrate Web APIs (speech recognition & synthesis)** into React.
- Gained hands-on experience with **React Hooks** for state and lifecycle management.
- Practiced **Context API + Reducer** for scalable state management.
- Mastered using **Tailwind utility classes** for clean, responsive UI design.
- Enhanced knowledge of **accessible app design** through voice features.

Code and explanation

1. Adding Tasks Manually

```
const addTask = () => {
  if (input.trim() !== "") {
    dispatch({ type: "ADD_TASK", payload: input });
    setInput("");
  }
};
```

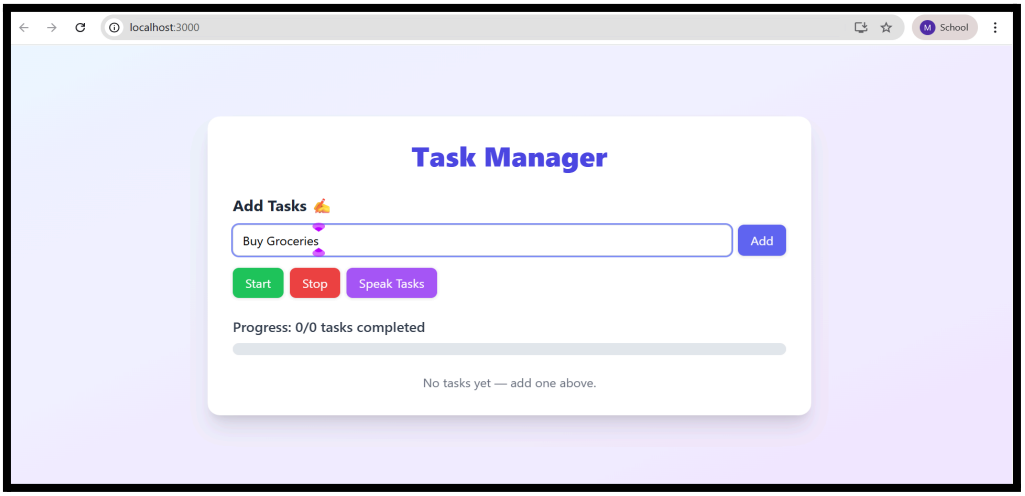


Fig 1.1 - Adding tasks Manually

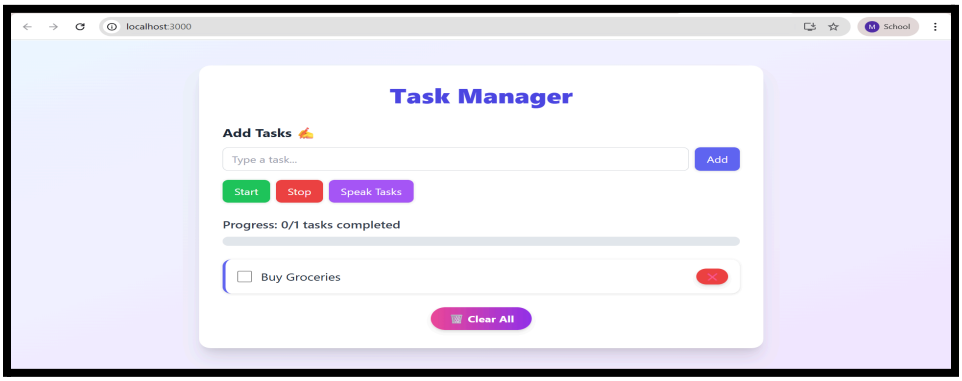


Fig 1.2 - Tasks Added

2. Adding Tasks through voice control (30% extra)

Adding tasks through voice control makes use of the Web Speech API's SpeechRecognition interface. Instead of typing into an input field, the user speaks a command such as:

- *"Finish assignment"*
- *"Buy groceries"*

The application listens for these commands, transcribes the speech into text, and extracts the task description. This task is then added to the task list in the same way as manual input.

This improves accessibility (hands-free interaction for differently-abled users) and enhances user experience by making the application faster and more engaging.

```
import React, { useState } from "react";
import { useTasks } from "../context/TaskContext";

const VoiceControls = () => {
  const { tasks, dispatch } = useTasks();
  const [listening, setListening] = useState(false);
  const [input, setInput] = useState("");

  const startListening = () => {
    const recognition = new (window.SpeechRecognition ||
window.webkitSpeechRecognition)();
    recognition.lang = "en-US";
    recognition.start();

    recognition.onresult = (event) => {
      const transcript = event.results[0][0].transcript;
      dispatch({ type: "ADD_TASK", payload: transcript });
    };

    recognition.onend = () => setListening(false);
    setListening(true);
  };

  const stopListening = () => {
```

```
    setListening(false);
  };
```

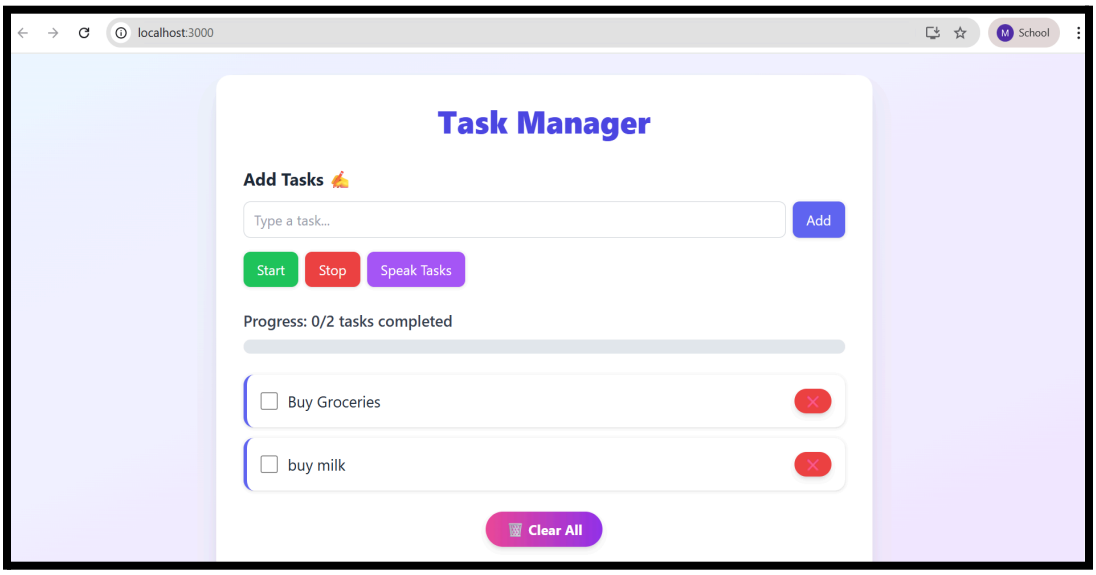


Fig 2.1- Adding Tasks through voice input

3. Voice Feedback (30% extra)

Voice feedback is implemented using the Web Speech API's Speech Synthesis interface.

- It enables the application to "speak back" to the user with natural-sounding speech.
- This provides auditory confirmation in addition to visual updates.
- Improves accessibility (useful for visually impaired users).
- Enhances user motivation and engagement (e.g., congratulatory messages when tasks are completed).

```
const readTasks = () => {
 if ("speechSynthesis" in window) {
   const pending = tasks.filter(t => !t.completed).map(t =>
t.text);
   const text =
     pending.length === 0
         ? "You have no pending tasks. Great job!"
          : `You have ${pending.length} pending tasks:
${pending.join(", ")}`;
```

```
      const utterance = new SpeechSynthesisUtterance(text);
       speechSynthesis.speak(utterance);
    }
  };
```

4. Progress Tracker

A progress tracker bar visually represents how many tasks the user has completed compared to the total tasks. It enhances motivation and provides instant feedback.

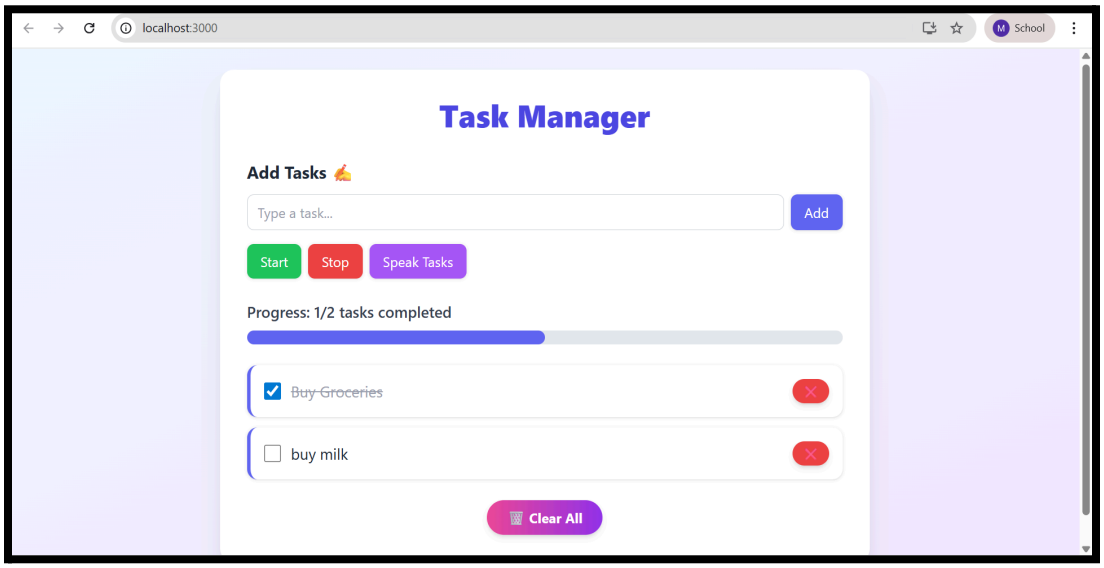
- Numerical feedback → e.g., *"Progress: 2/5 tasks completed"*.
- Visual feedback → a progress bar that fills as tasks are marked completed.

```
import React from "react";
import { useTasks } from "../context/TaskContext";

const ProgressTracker = () => {
  const { tasks } = useTasks();
  const completed = tasks.filter(t => t.completed).length;
  const total = tasks.length;
  const progress = total === 0 ? 0 : Math.round((completed /
total) * 100);

  return (
    <div className="my-6">
      <p className="text-lg font-semibold text-gray-700 mb-2">
        Progress: {completed}/{total} tasks completed
      </p>
      <div className="w-full bg-gray-200 rounded-full h-4">
        <div
          className="bg-indigo-500 h-4 rounded-full
transition-all duration-300"
          style={{ width: `${progress}%` }}
        ></div>
      </div>
    </div>
  );
};
```

```
export default ProgressTracker;
```



- **Voice Controls (30% Extra)**

Voice controls allow users to interact with the application entirely through speech. In this project, the system has two essential functions:

Voice Input – using the Web Speech API's SpeechRecognition feature, the app listens to what the user says and converts it into text. Whatever is spoken is directly written into the task list. Unlike advanced natural language processing, this system takes a direct transcription approach, recording speech exactly as it is said.

Voice Feedback – using the Speech Synthesis API, the app then "speaks back" the recorded task, confirming that the input has been captured. This closes the loop by giving users both visual confirmation (text on screen) and auditory confirmation (spoken feedback).

Together, these two features form a two-way interaction loop:

User → App (Voice Input): The user speaks a task.

App → User (Voice Feedback): The app repeats the task back aloud.

1. Voice Input (Speech-to-Text)

The Voice Input part relies on the SpeechRecognition interface of the Web Speech API.

When activated, it continuously listens for user speech, processes the audio, and converts it into textual data.

Recognition Language – set to "en-US" for English input.

Transcript – the spoken words are captured as a transcript.

Direct Recording – the transcript is not parsed or modified; it is directly added to the task list exactly as spoken.

Example:

1. User says: “buy milk” → Task list shows: buy milk.
2. User says: “buy groceries” → Task list shows: buy groceries.

This demonstrates how speech can replace traditional typing, even in its most basic form.

Significance:

- Provides a hands-free way to interact with the app.
- Makes the task manager usable in multitasking scenarios (e.g., when the user’s hands are busy).
- Acts as a foundation for more advanced voice command systems in the future.

2. Voice Feedback (Text-to-Speech)

The Voice Feedback part uses the SpeechSynthesis interface of the Web Speech API. Once a transcript has been written into the task list, the application generates spoken confirmation.

SpeechSynthesisUtterance – a text string is wrapped in an utterance object.

Playback – the browser’s built-in speech engine converts this text into natural-sounding speech.

Customization – properties like pitch, rate, and volume can be adjusted to make the voice output clearer and more natural.

Example:

1. If the user says: “buy milk” → App responds: “Task added: buy milk”.
2. If the user says: “attend meeting” → App responds: “Task added: attend meeting”.

Significance:

- Provides instant confirmation that the system has heard correctly.
- Enhances accessibility, especially for users with visual impairments who may not rely on screen text.
- Improves user confidence by ensuring that spoken input was successfully processed.

1. Accessibility and User Experience Impact

Voice controls make the task manager more inclusive and user-friendly.

- Accessibility → People with disabilities (e.g., limited motor skills or vision loss) can still use the app effectively.
- Engagement → Voice interaction feels more natural and interactive than typing.
- Efficiency → Users can add tasks faster without relying on a keyboard.

Innovation → Demonstrates how React Hooks can be integrated with Web APIs to produce a modern, speech-driven interface.

2. Overall Importance

- Even though the functionality is simple (speech-to-text recording and spoken feedback), voice controls:
- Turn the task manager into a speech-based productivity tool.
- Showcase integration of Web Speech API with React Hooks.
- Provide both input and output modalities, ensuring a closed communication loop.
- Lay the groundwork for advanced features like command recognition, continuous dictation, and multilingual support.

Conclusion

The **Voice Controlled Task Manager** successfully combines React Hooks, Tailwind CSS, and Web APIs to create a powerful, interactive, and accessible web application. Unlike conventional task managers, it enables users to manage tasks through both **manual input and voice commands**, making it **hands-free and user-friendly**.

The integration of a **progress bar** ensures real-time visual tracking, while **voice feedback** motivates and guides the user. The use of **custom hooks and Context API** ensures clean, reusable, and scalable code.

This experiment demonstrates how **modern React practices** and **browser-native APIs** can be combined to create innovative applications that are **responsive, accessible, and future-ready**.