# Practical Number 1

## AIM:

Build responsive and interactive UIs using Tailwind CSS.

## THEORY:

1. **React**
   React is a JavaScript library developed by Facebook for building fast and interactive user interfaces. It is component-based, which means the UI is broken into small reusable pieces. React handles updates efficiently using its virtual DOM, ensuring that the page reloads only the parts that change.

2. **React Hooks (`useState`)**
   Hooks are special functions in React that let developers use state and lifecycle features in functional components. In this practical, we used the `useState` hook to manage the state of input fields (such as team names, members, and tasks) and dynamically update the UI whenever the user entered new information.

3. **Tailwind CSS**
   Tailwind CSS is a utility-first CSS framework that provides ready-to-use classes for styling. Instead of writing custom CSS files, developers can directly use predefined classes for spacing, colors, typography, layout, and responsiveness. This allows rapid development of clean, modern, and responsive UIs. For example, classes like `p-4`, `bg-blue-600`, or `rounded-lg` style elements instantly.

### What is Tailwind CSS?

Tailwind CSS is a utility-first CSS framework that provides low-level utility classes to build custom designs directly in your HTML. Unlike traditional CSS frameworks that provide pre-designed components, Tailwind allows developers to compose their design by combining small, reusable utility classes. This approach increases development speed, improves consistency, and enhances customization.

### Responsive Design with Tailwind CSS

Responsive design means creating web interfaces that adapt smoothly to various screen sizes

(desktop, tablet, mobile). Tailwind CSS offers responsive variants of its utility classes to control styles based on screen breakpoints. These breakpoints are mobile-first, meaning styles apply to all screen sizes unless overridden by larger breakpoints.

Responsive design means creating web interfaces that adapt smoothly to various screen sizes (desktop, tablet, mobile). Tailwind CSS offers responsive variants of its utility classes to control styles based on screen breakpoints. These breakpoints are mobile-first, meaning styles apply to all screen sizes unless overridden by larger breakpoints.

**Typical breakpoints in Tailwind include:**

● sm: — Small screens (≥ 640px)

● md: — Medium screens (≥ 768px)

● lg: — Large screens (≥ 1024px)

● xl: — Extra large screens (≥ 1280px)

● 2xl: — 2x Extra large screens (≥ 1536px)

Using these prefixes, you can specify styles that apply only at certain screen widths, e.g., md:text-lg applies text-lg font size only on medium screens and larger.

**Interactive UIs with Tailwind CSS**

Tailwind CSS supports interactive UI states through pseudo-class variants, such as:

● hover: — styles on mouse hover

● focus: — styles when an element is focused (e.g., input fields)

● active: — styles when an element is active (e.g., a button click)

● disabled: — styles for disabled elements

By combining these variants with utility classes, you can easily create buttons, links, and inputs that respond visually to user interaction without writing custom CSS.

## CODE:-

App.jsx ↓M, M | App.css | tailwind.config.js ↓M

src > App.css > .logo:hover

```css
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}
```

```css
@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}
```

App.jsx ↓M, M | App.css | tailwind.config.js ↓M | main.jsx ↓M

src > App.jsx > App

```jsx
import React, { useState } from "react";

export default function App() {
  const [teams, setTeams] = useState([]);
  const [tasks, setTasks] = useState([]);

  // Form states
  const [teamName, setTeamName] = useState("");
  const [teamMembers, setTeamMembers] = useState("");
  const [taskName, setTaskName] = useState("");
  const [selectedTeam, setSelectedTeam] = useState("");

  // Add team
  const handleAddTeam = () => {
    if (!teamName.trim() || !teamMembers.trim()) return;
    const membersArray = teamMembers.split(",").map(m => m.trim());
    setTeams([...teams, { name: teamName, members: membersArray }]);
    setTeamName("");
    setTeamMembers("");
  };
```

```
// Add task
const handleAddTask = () => {
  if (!taskName.trim() || !selectedTeam) return;
  const team = teams.find(t => t.name === selectedTeam);
  setTasks([...tasks, { task: taskName, team }]);
  setTaskName("");
  setSelectedTeam("");
};

return (
```

```
return (
  <div className="min-h-screen bg-gradient-to-br from-blue-50 to-blue-100 flex flex-col items-center py-8">
    <h1 className="text-4xl font-extrabold text-blue-700 mb-8">Collaboration App</h1>

    <div className="grid grid-cols-1 md:grid-cols-2 gap-8 w-full max-w-5xl px-4">
      {/* Add Team */}
      <div className="bg-white p-6 rounded-2xl shadow-lg">
        <h2 className="text-2xl font-semibold text-gray-800 mb-4">Add New Team</h2>
        <input
          type="text"
          placeholder="Team Name"
          value={teamName}
          onChange={(e) => setTeamName(e.target.value)}
          className="border border-gray-300 p-2 rounded-lg w-full mb-3"
        />
        <input
          type="text"
          placeholder="Members (comma separated)"
          value={teamMembers}
          onChange={(e) => setTeamMembers(e.target.value)}
          className="border border-gray-300 p-2 rounded-lg w-full mb-3"
        />
        <button
          onClick={handleAddTeam}
          className="bg-blue-600 text-white px-4 py-2 rounded-lg hover:bg-blue-700 w-full"
        >
          Add Team
        </button>
      </div>
```

```
{/* Add Task */}
<div className="bg-white p-6 rounded-2xl shadow-lg">
  <h2 className="text-2xl font-semibold text-gray-800 mb-4">Add New Task</h2>
  <input
    type="text"
    placeholder="Task Name"
    value={taskName}
    onChange={(e) => setTaskName(e.target.value)}
    className="border border-gray-300 p-2 rounded-lg w-full mb-3"
  />
  <select
    value={selectedTeam}
    onChange={(e) => setSelectedTeam(e.target.value)}
    className="border border-gray-300 p-2 rounded-lg w-full mb-3"
  >
    <option value="">Select Team</option>
    {teams.map((team, index) => (
      <option key={index} value={team.name}>
        {team.name}
      </option>
    ))}
  </select>
  <button
    onClick={handleAddTask}
    className="bg-green-600 text-white px-4 py-2 rounded-lg hover:bg-green-700 w-full"
  >
    Add Task
  </button>
</div>
</div>
```

```jsx
        {/* Task List */}
        <div className="bg-white mt-10 p-6 rounded-2xl shadow-lg w-full max-w-5xl">
          <h2 className="text-2xl font-semibold text-gray-800 mb-4">Tasks</h2>
          {tasks.length === 0 ? (
            <p className="text-gray-500">No tasks added yet.</p>
          ) : (
            <div className="space-y-4">
              {tasks.map((t, index) => (
                <div key={index} className="bg-gray-50 p-4 rounded-lg shadow-sm">
                  <p className="font-bold text-lg text-gray-700">{t.task}</p>
                  <p className="text-sm text-gray-600 mb-2">Team: {t.team.name}</p>
                  <ul className="list-disc list-inside text-gray-600 text-sm">
                    {t.team.members.map((member, i) => (
                      <li key={i}>{member}</li>
                    ))}
                  </ul>
                </div>
              ))}
            </div>
          )}
        </div>
      </div>
    );
  }
```

App.jsx ↓M, M  # index.css 3 ×

src > # index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

App.jsx ↓M, M | main.jsx ↓M | index.css 3 | tailwind.config.js ↓M

src > main.jsx

```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

```

App.jsx ↓M, M | main.jsx ↓M | eslint.config.js | tailwind.config.js ↓M

eslint.config.js > ...

```jsx
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import { defineConfig, globalIgnores } from 'eslint/config'

export default defineConfig([
  globalIgnores(['dist']),
  {
    files: ['**/*.{js,jsx}'],
    extends: [
      js.configs.recommended,
      reactHooks.configs['recommended-latest'],
      reactRefresh.configs.vite,
    ],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
      parserOptions: {
        ecmaVersion: 'latest',
        ecmaFeatures: { jsx: true },
        sourceType: 'module',
      },
    },
    rules: {
      'no-unused-vars': ['error', { varsIgnorePattern: '^[A-Z_]' }],
    },
  },
])
```

App.jsx ↓M, M | index.html ↓M | main.jsx ↓M | eslint.config.js | tailwind.config.js

index.html > ...

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Collaboration App</title>
  </head>
  <body class="bg-gray-50 text-gray-900">
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

**OUTPUT:-**