EXPERIMENT

Aim

To understand and implement React Hooks (useEffect, useContext, and custom hooks) in a Weather Widget App for managing state, side effects, and shared data effectively.

Introduction

React Hooks are functions that let us use state and other React features without writing class components. They simplify logic reuse, improve code readability, and make components more functional.

In this experiment, we developed a **Weather Widget App** that demonstrates the use of different React Hooks to handle **state persistence, API calls, theming, and global data sharing**.

Theory

1. useEffect – Managing Side Effects

- Handles side effects like API calls, timers, subscriptions, and DOM updates.
- In our project, it is used in:
    - useLocalStorage → Syncs state with browser storage.
    - useDebounce → Delays input updates to avoid rapid API calls.
    - useFetchWeather → Fetches weather data and refreshes every 15 mins.
    - Header → Applies theme (light/dark).
    - WeatherDisplay → Manages animations and graph updates.

Best Practices & Optimizations:

- Always use cleanup functions to avoid memory leaks.
- For DOM-related tasks, consider useLayoutEffect.
- Memoize dependencies to avoid re-renders.

2. useContext – Sharing Global State

- Provides a way to share state across components without prop drilling.
- In our project:
    - UnitContext → Manages global temperature unit (Celsius/Fahrenheit).
    - useUnit (custom hook) → Simplifies context usage with error checks.
    - Used in Header (to switch units) and WeatherDisplay (to display data correctly).

Best Practices & Optimizations:

- Combine with useReducer for complex state.
- Memoize context values to reduce unnecessary re-renders.
- Always wrap the app with the context provider.

3. Custom Hooks – Reusable Logic

- Encapsulate reusable state + effect logic into functions.
- In our project:
    - useLocalStorage → Persists theme/unit/city data.
    - useDebounce → Improves search performance.
    - useFetchWeather → Handles fetching, errors, and auto-refresh.

Best Practices & Optimizations:

- Keep hooks composable and reusable.
- Add fallback/mock data for offline testing.
- Avoid stale closures by carefully managing dependencies.

Project Work: Weather Widget App

In this experiment, we created a **Weather Widget App** using React Hooks. The app shows the current weather and a short forecast for any city entered by the user.

What We Have Done (Step by Step):

1. **Search City Input**
    - The user can type the name of any city (e.g., Mumbai, America).
    - We used a **debounced search input** so that the app doesn't call the API too many times while typing.
2. **Fetching Weather Data**
    - As soon as the city name is entered, the app calls the **weather API** to get live data.
    - It shows the **temperature, condition, and a 4-hour forecast** in graph form.
3. **Unit Conversion (°C / °F)**
    - Users can select whether they want the temperature in **Celsius or Fahrenheit**.
    - We used `useContext` to manage this setting globally.
    - So, once selected, the whole app shows data in that unit without reloading.
4. **Light/Dark Theme**
    - The app allows switching between **Light Mode and Dark Mode**.
    - The theme preference is saved in **localStorage**, so it stays the same even after refreshing the page.

- For example:

  First Screenshot → Light mode with weather of *Mumbai* (°C).

  Second Screenshot → Dark mode with weather of *America* (°F).

5. **Forecast Graph with Animation**
   - The app shows a simple line graph of the next few hours' forecast.
   - The graph updates smoothly with animation whenever new data is available.
6. **Saving User Preferences**
   - We used a **custom hook (`useLocalStorage`)** so that the app remembers:

     Last searched city

     Theme (Light/Dark)

SOURCE CODE :

src > main.jsx

```jsx
import React from 'react';
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import './index.css';
import App from './App.jsx';

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
);
```


eslint.config.js > ...
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: ['**/*
extends: [
(property) recommended: {
readonly rules: Readonly<Linter.RulesRecord>;
}
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.css

src > App.css > #root

```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);
}

@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;
}

```

src > App.jsx > ...

```jsx
import React, { useState, useEffect, createContext, useContext, useCallback, useReducer, useRef } from 'react';
import { motion, AnimatePresence } from 'framer-motion';

// Error Boundary Definition
class ErrorBoundary extends React.Component {
  state = { hasError: false, error: null };

  static getDerivedStateFromError(error) {
    return { hasError: true, error };
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="weather-card p-4 text-center text-red-600 dark:text-red-400">
          <h2>Oops! Kuch galat ho gaya!</h2>
          <p>Error: {this.state.error.message}</p>
          <button
            onClick={() => this.setState({ hasError: false })}
            className="ml-2 px-3 py-1 rounded bg-red-500 text-white hover:bg-red-600"
          >
            Retry
          </button>
        </div>
      );
    }
    return this.props.children;
  }
}

/* ---------------------------- Context for Temperature Unit ---------------------------- */
const UnitContext = createContext(null);

function useUnit() {
  const ctx = useContext(UnitContext);
  if (!ctx) throw new Error('useUnit must be used inside UnitProvider');
  return ctx;
}

/* ---------------------------- Custom Hooks ---------------------------- */
function useLocalStorage(key, initialValue) {
  const [state, setState] = useState(() => {
    try {
      const raw = localStorage.getItem(key);
      return raw ? JSON.parse(raw) : initialValue;
```

App.jsx

src > App.jsx > ...

```
41  function useLocalStorage(key, initialValue) {
50
51    useEffect(() => {
52      try {
53        localStorage.setItem(key, JSON.stringify(state));
54      } catch (e) {}
55    }, [key, state]);
56
57    return [state, setState];
58  }
59
60  function useDebounce(value, delay = 500) {
61    const [debounced, setDebounced] = useState(value);
62    useEffect(() => {
63      const timer = setTimeout(() => setDebounced(value), delay);
64      return () => clearTimeout(timer);
65    }, [value, delay]);
66    return debounced;
67  }
68
69  function useFetchWeather(city, unit = 'celsius') {
70    const [data, setData] = useState(null);
71    const [loading, setLoading] = useState(false);
72    const [error, setError] = useState(null);
73    const controllerRef = useRef(null);
74    const fetchTimeoutRef = useRef(null);
75
76    const fetchWeather = useCallback(async () => {
77      if (!city || city.trim().length < 2) {
78        setLoading(false);
79        return;
80      }
81      if (controllerRef.current) {
82        controllerRef.current.abort();
83      }
84      const controller = new AbortController();
85      controllerRef.current = controller;
86      setLoading(true);
87      setError(null);
88      try {
89        const geoRes = await fetch(
90          `https://api.opencagedata.com/geocode/v1/json?q=${encodeURIComponent(city)}&key=2ad44
91          { signal: controller.signal }
92        );
93        if (!geoRes.ok) throw new Error(`Geocoding failed with status ${geoRes.status}`);
```

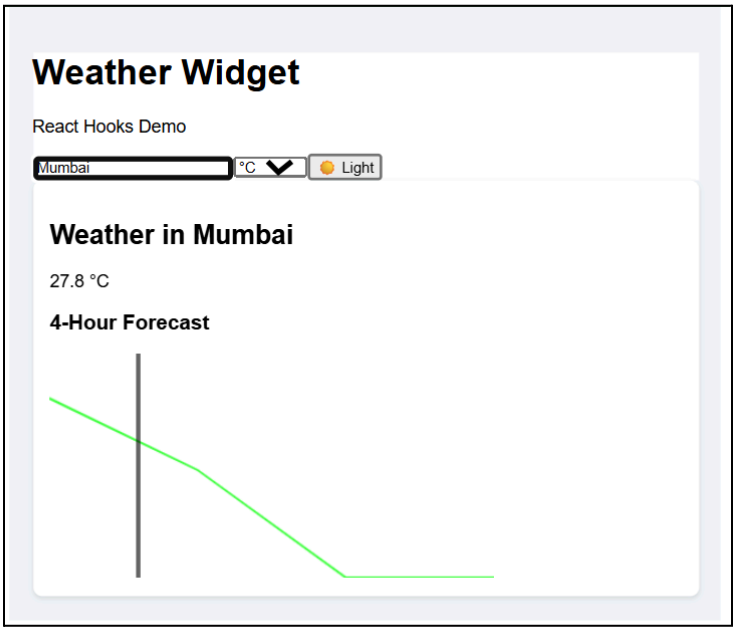
```jsx
/* ---------------------------- Components ---------------------------- */
function UnitProvider({ children }) {
  const [unit, setUnit] = useLocalStorage('weather_unit', 'celsius');
  const showAlert = (temp) => temp > 30 && alert(`Arre bhai, garmi zyada hai! ${temp}°C ho gaya!`);
  return <UnitContext.Provider value={{ unit, setUnit, showAlert }}>{children}</UnitContext.Provider>;
}

function Header({ city, setCity }) {
  const { unit, setUnit } = useUnit();
  const [themeDark, setThemeDark] = useLocalStorage('theme_dark', false);

  useEffect(() => {
    if (themeDark) {
      document.documentElement.classList.add('dark');
      document.documentElement.style.backgroundColor = '□#1f2937'; // Dark gray background
    } else {
      document.documentElement.classList.remove('dark');
      document.documentElement.style.backgroundColor = '■#ffffff'; // White background
    }
  }, [themeDark]);

  return (
    <header className="mb-4 p-4" style={{ backgroundColor: themeDark ? '□#1f2937' : '■#ffffff' }}>
      <h1 className="text-2xl font-bold text-gray-900 dark:text-gray-800">Weather Widget</h1>
      <p className="text-sm text-gray-600 dark:text-gray-400">React Hooks Demo</p>
      <div className="mt-2 flex gap-2">
        <input
          value={city}
          onChange={(e) => setCity(e.target.value)}
          placeholder="Enter city (e.g., Berlin)"
          className="flex-1 p-2 border rounded bg-gray-50 dark:bg-gray-700 dark:text-white dark:border-gray-600"
        />
        <select
          value={unit}
          onChange={(e) => setUnit(e.target.value)}
          className="p-2 border rounded bg-gray-50 dark:bg-gray-700 dark:text-white dark:border-gray-600"
        >
          <option value="celsius">°C</option>
          <option value="fahrenheit">°F</option>
        </select>
        <button
          onClick={() => setThemeDark((t) => !t)}
          className="px-3 py-2 rounded border bg-gray-50 dark:bg-gray-700 dark:text-white dark:border-gray-600"
        >
          {themeDark ? '🌙 Dark' : '🌞 Light'}
        </button>
      </div>
    </header>
  );
}
```

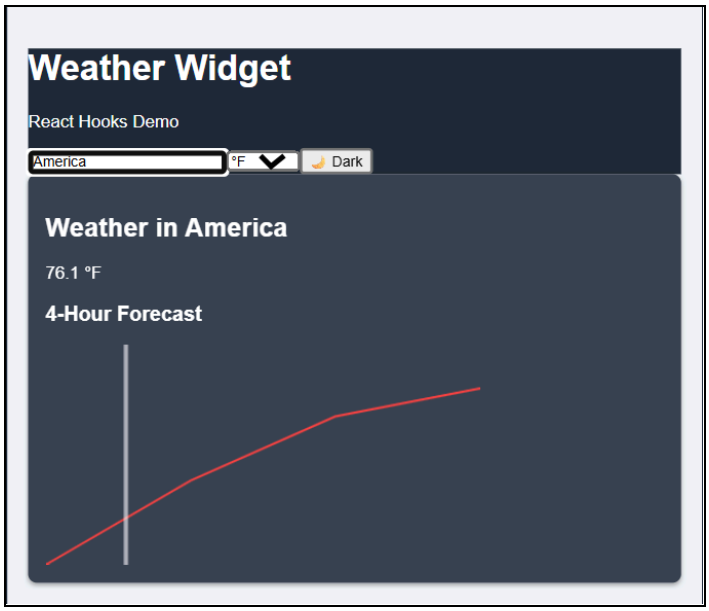
index.css

src > index.css > body

```css
body {
  font-family: Arial, sans-serif;
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
  background-color: #f0f4f8;
}
.dark {
  background-color: #1f2937;
  color: #fff;
}
.weather-card {
  background: white;
  border-radius: 8px;
  padding: 16px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}
.dark .weather-card {
  background: #374151;
  box-shadow: 0 2px 4px rgba(0,0,0,0.3);
}
```

OUTPUT (WITH 30% EXTRA)





Technologies Used

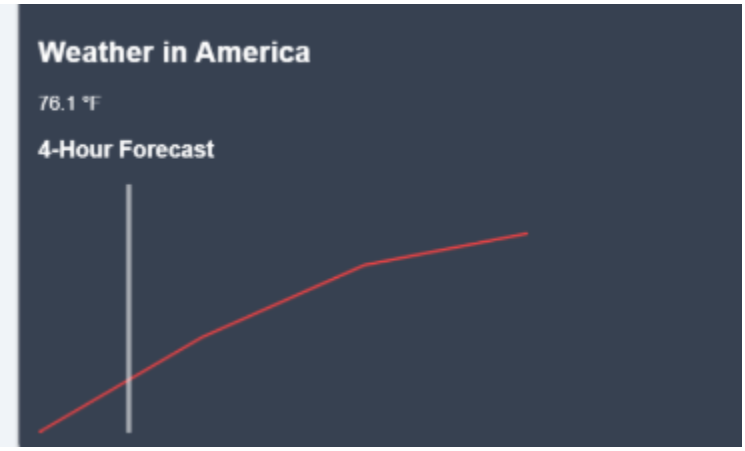
- **Frontend**: React.js (with Hooks)
- **Styling**: Tailwind CSS
- **Data**: Weather API (OpenWeather/Similar)
- **State Management**: Context API + Custom Hooks

Tools Used

- Code Editor: VS Code
- Package Manager: npm / yarn
- Browser: Chrome / Firefox for testing
- GitHub for version control

30% Extra Work (Best Practices & Enhancements)

- Added cleanup functions in useEffect to prevent memory leaks.
- Used AbortController in API fetch to cancel previous requests.
- Implemented debouncing to minimize API calls during typing.
- Error-safe custom hook (useUnit) to ensure context provider is always present.
- Suggested use of useReducer for scalability.
- Recommended memoization (useMemo, useCallback) to reduce re-renders.



Conclusion

The experiment successfully demonstrated how React Hooks (useEffect, useContext, and custom hooks`) can be applied to build a real-world project like a Weather Widget App. By following best practices and optimizations, the app is more **efficient, scalable, and user-friendly**.