

Experiment No 2

Aim : Experiment based on React Hooks (useEffect, useContext, custom hooks)

Theory :

React Hooks are a modern feature introduced in React that enable developers to use state and lifecycle methods directly inside functional components, without needing to write class components. This results in cleaner, more readable, and more reusable code. Hooks promote the functional programming style in React and make stateful logic easier to share across components.

Hooks Used in This Project

1.useState

- Purpose: Manages component-level state.
- Example: Used for toggling between light and dark mode themes, storing form inputs, or managing task lists.
- Advantage: Provides a simple API for updating and tracking state in functional components.

2.useEffect

- Purpose: Handles side effects such as API calls, DOM manipulation, and subscriptions.
- Example: Fetching data from the backend (Express + MongoDB) when the component mounts.
- Advantage: Keeps side-effect logic tied to the lifecycle of components without needing lifecycle methods.

3.useContext

- Purpose: Provides global state management across multiple components.
- Example: Sharing theme settings (light/dark) or authentication details across the entire application.
- Advantage: Eliminates “prop drilling,” allowing data to be accessed by deeply nested components without manually passing props.

4.useCallback

- Purpose: Optimizes performance by memoizing functions so they are not recreated unnecessarily on each render.

- Example: Used to avoid re-rendering of child components when passing down callback functions.
- Advantage: Improves efficiency in components with expensive operations or frequent re-renders.

Key Patterns Demonstrated

1. Custom Hooks

- Example: `useForm()` and `useTasks()` encapsulate logic for handling form input and communicating with backend APIs.
- Benefit: Promotes reusability, separation of concerns, and cleaner components.

2. Context API

- Centralizes global application state such as theme settings or user session.
- Replaces external state management libraries in simpler applications.

3. Functional Components

- Built as pure functions returning JSX.
- Paired with hooks for modular, testable, and maintainable code.

Extra Features (30% Enhancement)

● Express + MongoDB Integration

- Backend created using **Express.js** to define RESTful API routes.
- Connected to **MongoDB** for persistent data storage (e.g., tasks, users, or notes).
- Frontend React app communicates with the backend using `fetch` or `axios` inside `useEffect`.
- Demonstrates a **full-stack approach**, extending beyond client-side development.

● API Routes

- Expose CRUD operations (GET, POST, PUT, DELETE) for managing data.
- Example: `/api/tasks` route retrieves and updates task data stored in MongoDB.

● Tailwind CSS Dark/Light Mode

- Theme switching implemented with Tailwind's utility classes and CSS custom properties.
- Supports user preference persistence (e.g., storing theme in local storage or context).

Code :

1.App.jsx(Used useTheme hook for light & dark mode)

```
import React from 'react';
import { ThemeProvider } from '../contexts/ThemeContext';
import { Header } from '../components/Header';
import { TaskForm } from '../components/TaskForm';
import { TaskCard } from '../components/TaskCard';
import { TaskStats } from '../components/TaskStats';
import { TaskFilter } from '../components/TaskFilter';
import { LoadingSpinner } from '../components/LoadingSpinner';
import { ErrorMessage } from '../components/ErrorMessage';
import { ConnectionStatus } from '../components/ConnectionStatus';
import { useTasks } from '../hooks/useTasks';
import { useTheme } from '../contexts/ThemeContext';

const AppContent: React.FC = () => {

}

function App() {
  return (
    <ThemeProvider>
      <AppContent />
    </ThemeProvider>
  );
}

export default App;
```

2.useState and useEffect(Used for time component)

```
import { useState, useEffect } from 'react';

export function useCurrentTime() {
  const [currentTime, setCurrentTime] = useState(new Date());

  useEffect(() => {
    const timer = setInterval(() => {
      setCurrentTime(new Date());
    }, 1000);

    // Cleanup function
    return () => {
      clearInterval(timer);
      console.log('Clock timer cleaned up');
    };
  }, []);

  return currentTime;
}
```

3.useContext(used for creating and managing state globally)

```
import React, { createContext, useContext, useState, useEffect, ReactNode } from 'react';
import { ThemeContextType } from '../types';

const ThemeContext = createContext<ThemeContextType | undefined>(undefined);

export const useTheme = () => {
  const context = useContext(ThemeContext);
  if (!context) {
    throw new Error('useTheme must be used within a ThemeProvider');
  }
  return context;
};

interface ThemeProviderProps {
  children: ReactNode;
}

export const ThemeProvider: React.FC<ThemeProviderProps> = ({ children }) => {
  const [theme, setTheme] = useState<'light' | 'dark'>(() => {
    const saved = localStorage.getItem('theme');
    return (saved as 'light' | 'dark') || 'light';
  });

  useEffect(() => {
    localStorage.setItem('theme', theme);
    document.documentElement.classList.toggle('dark', theme === 'dark');
  }, [theme]);

  const toggleTheme = () => {
    setTheme(prev => prev === 'light' ? 'dark' : 'light');
  };
};
```

4.useTask(custom hook to manage tasks)

```
import { useState, useEffect } from 'react';
import { Task, TaskStats } from '../types';
import { apiService, ApiTask } from '../services/api';

// Transform API task to frontend task format
const transformApiTask = (apiTask: ApiTask): Task => ({
  id: apiTask._id,
  _id: apiTask._id,
  title: apiTask.title,
  description: apiTask.description,
  completed: apiTask.completed,
  createdAt: new Date(apiTask.createdAt),
  completedAt: apiTask.completedAt ? new Date(apiTask.completedAt) : undefined,
});

export function useTasks() {
  const [tasks, setTasks] = useState<Task[]>([]);
  const [filter, setFilter] = useState<'all' | 'pending' | 'completed'>('all');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [stats, setStats] = useState<TaskStats>({
    total: 0,
    completed: 0,
    pending: 0,
    completionRate: 0,
  });

  // Fetch tasks from API
  const fetchTasks = async () => {
    try {
      setLoading(true);
      const apiTasks = await apiService.getTasks();
      const tasks = apiTasks.map(transformApiTask);
      setTasks(tasks);
      const stats = tasks.reduce(
        (acc, task) => {
          acc.total++;
          if (task.completed) acc.completed++;
          else acc.pending++;
          return acc;
        },
        { total: 0, completed: 0, pending: 0 }
      );
      setStats(stats);
    } catch (error) {
      setError(error instanceof Error ? error.message : 'An error occurred');
    } finally {
      setLoading(false);
    }
  };

  // Initial fetch
  fetchTasks();
}
```

5.useForm(custom hook to manage form)

```
import { useState, useCallback } from 'react';

interface UseFormProps<T> {
  initialValues: T;
  validate: (values: T) => Partial<Record<keyof T, string>> | null;
  onSubmit: (values: T) => void;
}

export function useForm<T extends Record<string, any>>>({
  initialValues,
  validate,
  onSubmit,
}: UseFormProps<T>) {
  const [values, setValues] = useState<T>(initialValues);
  const [errors, setErrors] = useState<Partial<Record<keyof T, string>>>({});
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleChange = useCallback((name: keyof T, value: any) => {
    setValues(prev => ({ ...prev, [name]: value }));
    // Clear error for this field when user starts typing
    if (errors[name]) {
      setErrors({});
    }
  }, [errors]);

  const handleSubmit = useCallback(async (e: React.FormEvent) => {
    const { values, validate, onSubmit, initialValues } = useFormProps;
    const errors = validate(values);
    if (!errors) {
      onSubmit(values);
    }
  }, [values, validate, onSubmit, initialValues]);

  const reset = useCallback(() => {
    setValues(initialValues);
    setErrors({});
  }, [initialValues]);

  return {
    values,
    errors,
    isSubmitting,
    handleChange,
    handleSubmit,
    reset,
  };
}
```

6.Database + Backend(using express to connect to MongoDB)

```
import express from 'express';
import mongoose from 'mongoose';
import cors from 'cors';
import dotenv from 'dotenv';
import taskRoutes from './routes/tasks.js';

dotenv.config();

const app = express();
const PORT = process.env.PORT || 3001;

// Middleware
app.use(cors());
app.use(express.json());

// Connect to MongoDB
mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/react-hooks-demo')
  .then(() => {
    console.log('MongoDB connected');
  })
  .catch((error) => {
    console.error('MongoDB connection error:', error);
  });

// Routes
app.use('/api/tasks', taskRoutes);

// Health check endpoint
app.get('/api/health', (req, res) => {
  res.json({ status: 'ok' });
});

// Error handling middleware
app.use((error, req, res, next) => {
  res.status(500).json({ error: 'Internal Server Error' });
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

The screenshot shows the MongoDB Compass interface. On the left, a list of collections is visible, including 'tasks'. The main panel displays the 'tasks' collection with two documents. The first document is for 'React' (description: 'Frontend', completed: false) and the second is for 'Node.js' (description: 'Backend', completed: true). Both documents have a 'createdAt' timestamp of 2025-08-13T04:54:58.705+08:00.

Output :

React Hooks Demo

Task Management with useEffect, useContext & Custom Hooks

🕒 01:52:56 🌙

Add New Task

Enter task title...

Enter task description (optional)...

+ Add Task

All (0) Pending (0) Completed (0)


0 Total Tasks

0 Completed

0 Pending

0% Complete

Tasks (0)



No tasks yet. Create your first task!

React Hooks Demo

Task Management with useEffect, useContext & Custom Hooks

🕒 01:54:31 🌙

Add New Task

mongodb

Enter task description (optional)...

+ Add Task

All (2) Pending (1) Completed (1)

2 Total Tasks

1 Completed

1 Pending

50% Complete

Tasks (2)

✓ javascript

easy

📅 Created 17/09/2025 ⌚ Completed 17/09/2025

🗑

○ Learnt React Hooks

very awesome

📅 Created 17/09/2025

🗑

Conclusion :

The project successfully demonstrates how React Hooks such as `useState` and `useEffect` can be used to fetch and display API data in a clean, responsive UI built with Tailwind CSS. Beyond the core implementation, an additional **30% extra features** were integrated, including **dark/light mode switching**, an **animated loading spinner** for better user feedback, and an **enhanced responsive UI with smooth transitions**. These improvements not only fulfill the base requirements but also elevate the overall user experience, making the application more interactive, modern, and user-friendly.