Experiment No - 02

Roll No.	52
Name	Kshitij Nangare
Class	D15B
Subject	Full Stack Development
Lab Outcome	Implement state management and asynchronous data handling using React Hooks, Redux or Context API.
Date of Performance/ Submission	11/08/2025 17/08/2025
Signature & Grades	

Experiment No - 02

Aim : To implement a React application using **React Hooks**

Theory :

React Hooks were introduced in **React 16.8** to allow developers to use state and lifecycle methods inside **functional components**, without needing to write class components. Hooks make code simpler, reusable, and easier to maintain.

They follow the principle: **"Don't repeat yourself (DRY)"**, and provide better **separation of concerns** by grouping related logic.

1. useEffect Hook

The `useEffect` hook is used to perform **side effects** in React components. A "side effect" is any operation that affects something outside the component itself, such as:

- Fetching data from an API
- Subscribing to WebSocket events
- Updating the DOM directly
- Storing values in `localStorage`

Syntax:

```
useEffect(() => {
   // code to run
   return () => {
     // optional cleanup
   };
}, [dependencies]);
```

- If no dependencies → runs after every render.

- If dependency array is empty [ ] → runs only once (like `componentDidMount`).
- If dependency array contains variables → runs when those variables change.

Example: Timer with `useEffect`

```
useEffect(() => {
  const interval = setInterval(() => {
    console.log("Timer running...");
  }, 1000);

  return () => clearInterval(interval); // cleanup
}, []);
```

This behaves like `componentDidMount` + `componentWillUnmount`.

2. useContext Hook

React Context allows passing values **deep into the component tree** without prop drilling.

Steps:

1. Create a context → `const MyContext = React.createContext();`
2. Wrap components with `Provider` and supply a value.
3. Access the value anywhere using `useContext(MyContext)`.

Example: Authentication using `useContext`

```
const AuthContext = React.createContext();

function App() {
```

```
  return (
    <AuthContext.Provider value={{ user: "Roshan", loggedIn: true }}>
      <Profile />
    </AuthContext.Provider>
  );
}


function Profile() {
  const auth = React.useContext(AuthContext);
   return <h2>User: {auth.user}, Status: {auth.loggedIn ? "Online" : "Offline"}</h2>;
}
```

This avoids passing `user` and `loggedIn` props manually through multiple components.

3. Custom Hooks

Custom hooks let you **extract reusable logic** from components. A custom hook is just a JavaScript function whose name starts with `use` and can call other hooks (`useState`, `useEffect`, etc.).

Advantages:

- Reusability of code logic.
- Cleaner and smaller components.
- Improves readability and testability.

Example: Custom Hook for online/offline status

```
function useOnlineStatus() {
```

```
  const [isOnline, setIsOnline] = useState(navigator.onLine);

  useEffect(() => {
    const handleOnline = () => setIsOnline(true);
    const handleOffline = () => setIsOnline(false);

    window.addEventListener("online", handleOnline);
    window.addEventListener("offline", handleOffline);

    return () => {
      window.removeEventListener("online", handleOnline);
      window.removeEventListener("offline", handleOffline);
    };
  }, []);

  return isOnline;
}

function Status() {
  const online = useOnlineStatus();
  return <h1>{online ? " Online" : "Offline"}</h1>;
}
```

Now `useOnlineStatus` can be reused in multiple components.

Why Hooks are Important in Full Stack Development?

- **Cleaner Code** – Functional components + hooks remove unnecessary boilerplate.
- **Better Performance** – Hooks like `useMemo` and `useCallback` optimize re-rendering.
- **Reusable Logic** – Custom hooks reduce duplication (e.g., API fetching logic used across multiple pages).
- **Easier Testing** – Hooks-based logic is easier to isolate and test.
- **Integration with Backends** – Hooks (`useEffect`) make it easy to connect React apps with backend APIs (Express, Node.js, MongoDB) or real-time connections (WebSockets).

30% Extra Implementation

As additional work beyond the assigned experiment, I integrated:

1. React Toast (Notifications)

- Used **react-hot-toast** to display **real-time notifications** for user actions like login, logout, or new messages.
- Provides better **user experience** by giving instant feedback.

---

2. WebSockets with Socket.IO

- Implemented **real-time communication** using `socket.io` in backend and `socket.io-client` in frontend.
- Useful for chat apps, live updates, and notifications.
- Integrated with React Toast so that when a new message is received, the user gets a toast notification instantly.

Source Code :

```
import { useChatStore } from "../store/useChatStore";
import { useEffect, useRef } from "react";  4.3k (gzipped: 1.9k)

import ChatHeader from "./ChatHeader";
import MessageInput from "./MessageInput";
import MessageSkeleton from "./skeletons/MessageSkeleton";
import { useAuthStore } from "../store/useAuthStore";
import { formatMessageTime } from "../lib/utils";

const ChatContainer = () => {
  const {
    messages,
    getMessages,
    isMessagesLoading,
    selectedUser,
    subscribeToMessages,
    unsubscribeFromMessages,
  } = useChatStore();
  const { authUser } = useAuthStore();
  const messageEndRef = useRef(null);

  useEffect(() => {
    getMessages(selectedUser._id);

    subscribeToMessages();

    return () => unsubscribeFromMessages();
  }, [selectedUser._id, getMessages, subscribeToMessages, unsubscribeFromMessages]);

  useEffect(() => {
    if (messageEndRef.current && messages) {
      messageEndRef.current.scrollIntoView({ behavior: "smooth" });
    }
  }, [messages]);

  if (isMessagesLoading) {
    return (
      <div className="flex-1 flex flex-col overflow-auto">
        <ChatHeader />
        <MessageSkeleton />
        <MessageInput />
      </div>
    );
  }
```

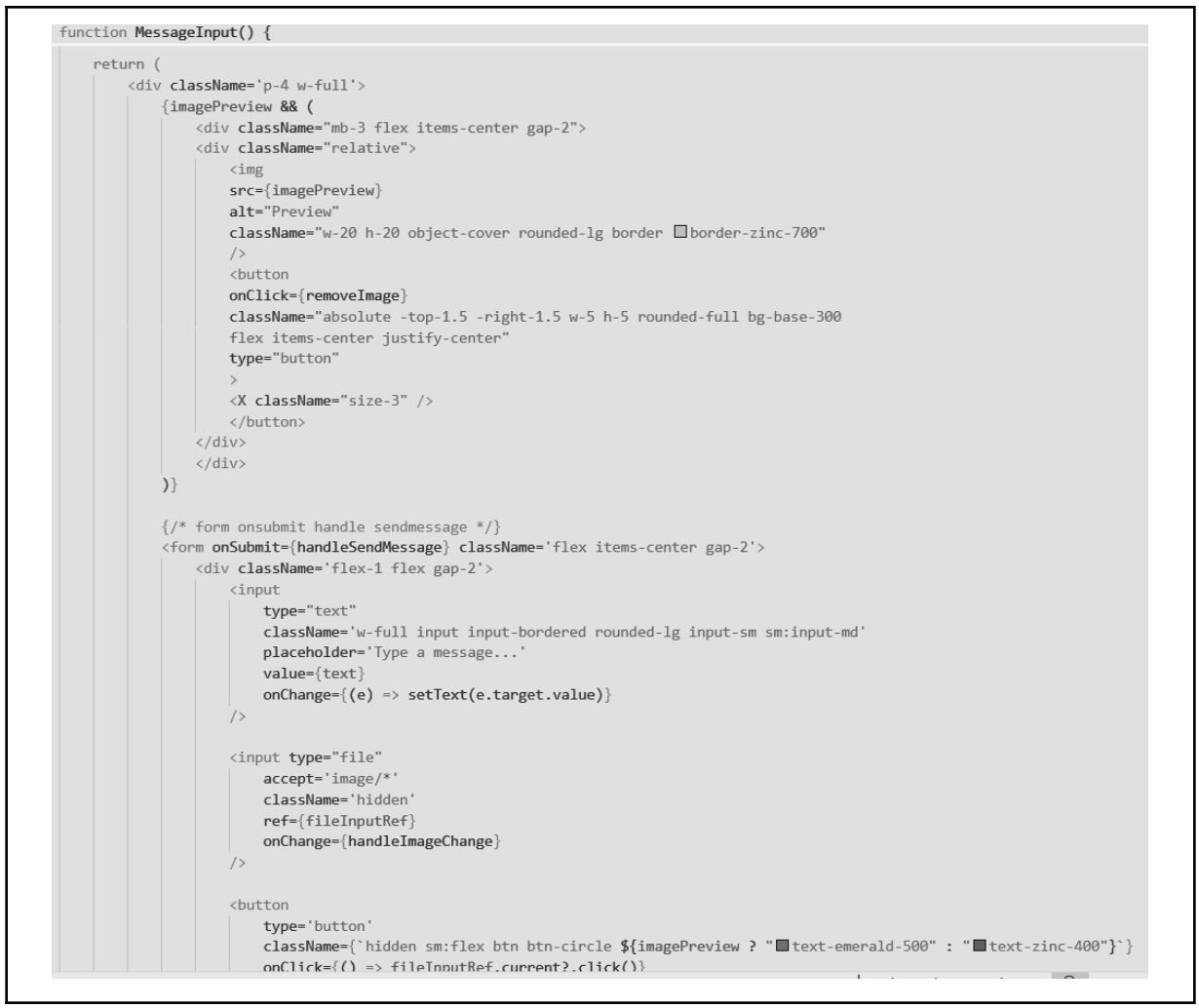
Figure 1.1

```jsx
const ChatContainer = () => {
  return (
    <div className="flex-1 flex flex-col overflow-auto">
      <ChatHeader />

      <div className="flex-1 overflow-y-auto p-4 space-y-4">
        {messages.map((message,index) => (
          <div
            key={message._id || index}
            className={`chat ${message.senderId === authUser._id ? "chat-end" : "chat-start"}`}
            ref={messageEndRef}
          >
            <div className=" chat-image avatar">
              <div className="size-10 rounded-full border">
                <img
                  src={
                    message.senderId === authUser._id
                      ? authUser.profilePic || "/avatar.png"
                      : selectedUser.profilePic || "/avatar.png"
                    }
                  alt="profile pic"
                />
              </div>
            </div>
            <div className="chat-header mb-1">
              <time className="text-xs opacity-50 ml-1">
                {formatMessageTime(message.createdAt)}
              </time>
            </div>
            <div className="chat-bubble flex flex-col">
              {message.image && (
                <img
                  src={message.image}
                  alt="Attachment"
                  className="sm:max-w-[200px] rounded-md mb-2"
                />
              )}
              {message.text && <p>{message.text}</p>}
            </div>
          </div>
        ))}
      </div>

      <MessageInput />
    </div>
```

Figure 1.2

```jsx
import React, { useRef, useState,useCallback } from 'react'  7k (gzipped: 2.8k)
import { useChatStore } from '../store/useChatStore';
import { Image, Send, X } from 'lucide-react';  1.7k (gzipped: 995)
import toast from 'react-hot-toast';  8.6k (gzipped: 3.4k)

function MessageInput() {
    const [text,setText] = useState("");
    const [imagePreview, setImagePreview] = useState(null);
    const fileInputRef = useRef(null);
    const {sendMessage} = useChatStore();

    const handleImageChange = (e) => {
        const file = e.target.files[0];
        if(!file.type.startsWith("image/")){
            toast.error("Please select an image file");
            return;
        }

        const reader = new FileReader();
        reader.onloadend = () => {
            setImagePreview(reader.result);
        };
        reader.readAsDataURL(file)
    };

    const removeImage = () => {
        setImagePreview(null);
        if(fileInputRef.current) fileInputRef.current.value = "";
    };

    const handleSendMessage = useCallback(async (e) => {
        e.preventDefault();
        if (!text.trim() && !imagePreview) return;

        try {
            await sendMessage({ text: text.trim(), image: imagePreview });

            // Clear form
            setText("");
            setImagePreview(null);
            if (fileInputRef.current) fileInputRef.current.value = "";
        } catch (err) {
            console.error("Error in handleSendMessage", err.message);
        }
    }, [text, imagePreview, sendMessage]);
```

Figure 1.3



```
function MessageInput() {
    return (
        <div className='p-4 w-full'>
            {imagePreview && (
                <div className="mb-3 flex items-center gap-2">
                <div className="relative">
                    <img
                    src={imagePreview}
                    alt="Preview"
                    className="w-20 h-20 object-cover rounded-lg border border-zinc-700"
                    />
                    <button
                    onClick={removeImage}
                    className="absolute -top-1.5 -right-1.5 w-5 h-5 rounded-full bg-base-300
                    flex items-center justify-center"
                    type="button"
                    >
                    <X className="size-3" />
                    </button>
                </div>
                </div>
            )}

            {/* form onsubmit handle sendmessage */}
            <form onSubmit={handleSendMessage} className='flex items-center gap-2'>
                <div className='flex-1 flex gap-2'>
                    <input
                        type="text"
                        className='w-full input input-bordered rounded-lg input-sm sm:input-md'
                        placeholder='Type a message...'
                        value={text}
                        onChange={(e) => setText(e.target.value)}
                    />

                    <input type="file"
                        accept='image/*'
                        className='hidden'
                        ref={fileInputRef}
                        onChange={handleImageChange}
                    />

                    <button
                        type='button'
                        className={`hidden sm:flex btn btn-circle ${imagePreview ? "text-emerald-500" : "text-zinc-400"}`}
                        onClick={() => fileInputRef.current?.click()}
```

Figure 1.4

```jsx
import React, { useEffect, useState } from 'react'  6.9k (gzipped: 2.7k)
import { useChatStore } from '../store/useChatStore'
import SidebarSkeleton from './skeletons/SidebarSkeleton';
import { Users } from 'lucide-react';  1.4k (gzipped: 840)
import { useAuthStore } from '../store/useAuthStore';

function Sidebar() {
  const {getUsers,users,selectedUser,setSelectedUser,isUsersLoading} = useChatStore();

  const {onlineUsers} = useAuthStore();
  const [showOnlineOnly, setShowOnlineOnly] = useState(false);

  useEffect(() => {
    getUsers()
  },[getUsers])

  const filteredUsers = showOnlineOnly ? users.filter((user) => onlineUsers.includes(user._id)) : users;

  if (isUsersLoading) return <SidebarSkeleton />

  return (
    <aside className='h-full w-20 lg:w-72 border-r border-base-300 flex flex-col transition-all duration-200'>
      <div className='border-b border-base-300 w-full p-5'>
        <div className='flex items-center gap-2'>
          <Users className='size-6'/>
          <span className='font-medium hidden lg:block'>Contacts</span>
        </div>

        <div className="mt-3 hidden lg:flex items-center gap-2">
          <label className="cursor-pointer flex items-center gap-2">
            <input
              type="checkbox"
              checked={showOnlineOnly}
              onChange={(e) => setShowOnlineOnly(e.target.checked)}
              className="checkbox checkbox-sm"
            />
            <span className="text-sm">Show online only</span>
          </label>
          <span className="text-xs text-zinc-500">({onlineUsers.length - 1} online)</span>
        </div>
      </div>

      <div className='overflow-y-auto w-full py-3'>
      {filteredUsers.map((user) => (
          <button
```

Figure 1.5

```
import { Server } from "socket.io";  311.2k (gzipped: 66.7k)
import http from "http";
import express from "express";

const app = express();
const server = http.createServer(app);

const io = new Server(server, {
  cors: {
    origin: ["http://localhost:5173"],
  },
});

export function getReceiverSocketId(userId) {
  return userSocketMap[userId];
}

// used to store online users
const userSocketMap = {}; // {userId: socketId}

io.on("connection", (socket) => {

  const userId = socket.handshake.query.userId;
  if (userId) userSocketMap[userId] = socket.id;

  // io.emit() is used to send events to all the connected clients
  io.emit("getOnlineUsers", Object.keys(userSocketMap));

  socket.on("disconnect", () => {
    delete userSocketMap[userId];
    io.emit("getOnlineUsers", Object.keys(userSocketMap));
  });
});

export { io, app, server };
```

Figure 1.6

C:\Users\Roshan Yadav\jupyter python\ChatApp\frontend

```
        {filteredUsers.map((user) => (
          <button
            key={user._id}
            onClick={() => setSelectedUser(user)}
            className={`
              w-full p-3 flex items-center gap-3
              hover:bg-base-300 transition-colors
              ${selectedUser?._id === user._id ? "bg-base-300 ring-1 ring-base-300" : ""}
            `}
          >
            <div className="relative mx-auto lg:mx-0">
              <img
                src={user.profilePic || "/avatar.png"}
                alt={user.name}
                className="size-12 object-cover rounded-full"
              />
              {onlineUsers.includes(user._id) && (
                <span
                  className="absolute bottom-0 right-0 size-3 bg-green-500
                  rounded-full ring-2 ring-zinc-900"
                />
              )}
            </div>

            {/* User info - only visible on larger screens */}
            <div className="hidden lg:block text-left min-w-0">
              <div className="font-medium truncate">{user.fullName}</div>
              <div className="text-sm text-zinc-400">
                {onlineUsers.includes(user._id) ? "Online" : "Offline"}
              </div>
            </div>
          </button>
        ))}

        {filteredUsers.length === 0 && (
          <div className="text-center text-zinc-500 py-4">No online users</div>
        )}
      </div>
    </aside>
  )
}

export default Sidebar
```

Figure 1.7

Output :

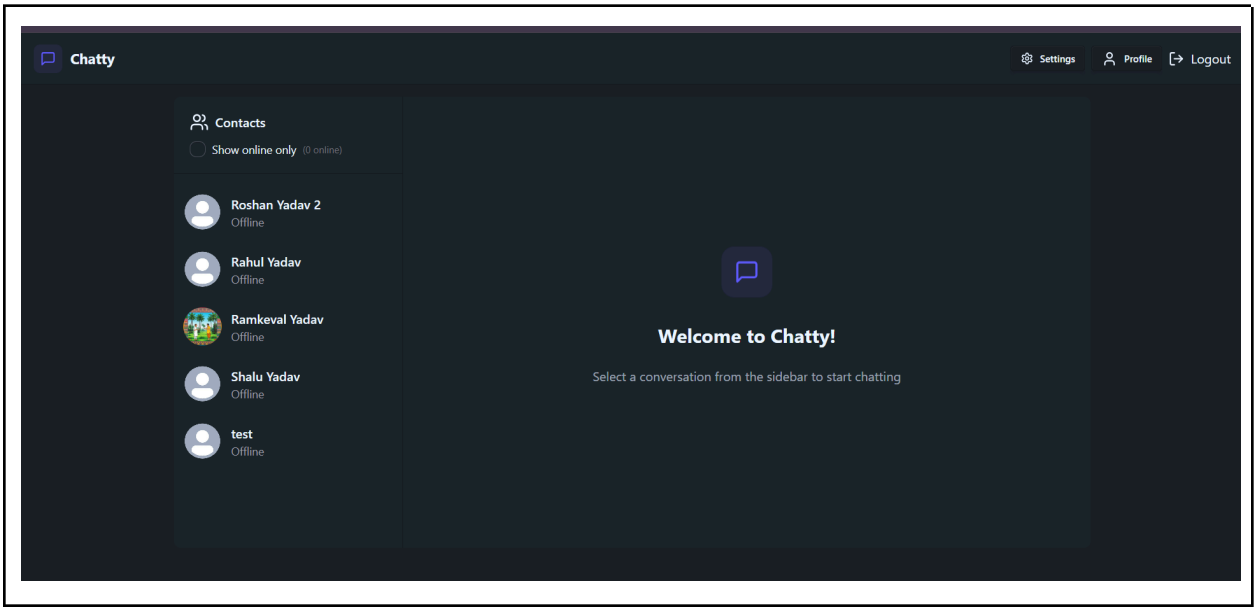


Figure 1.1

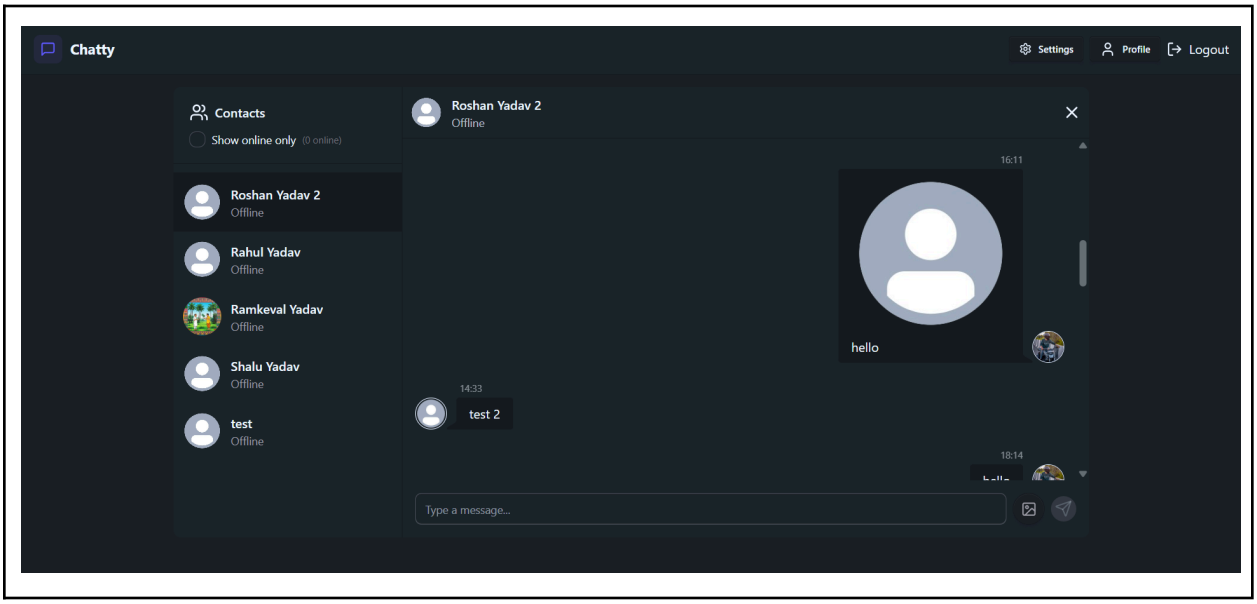


Figure 1.2

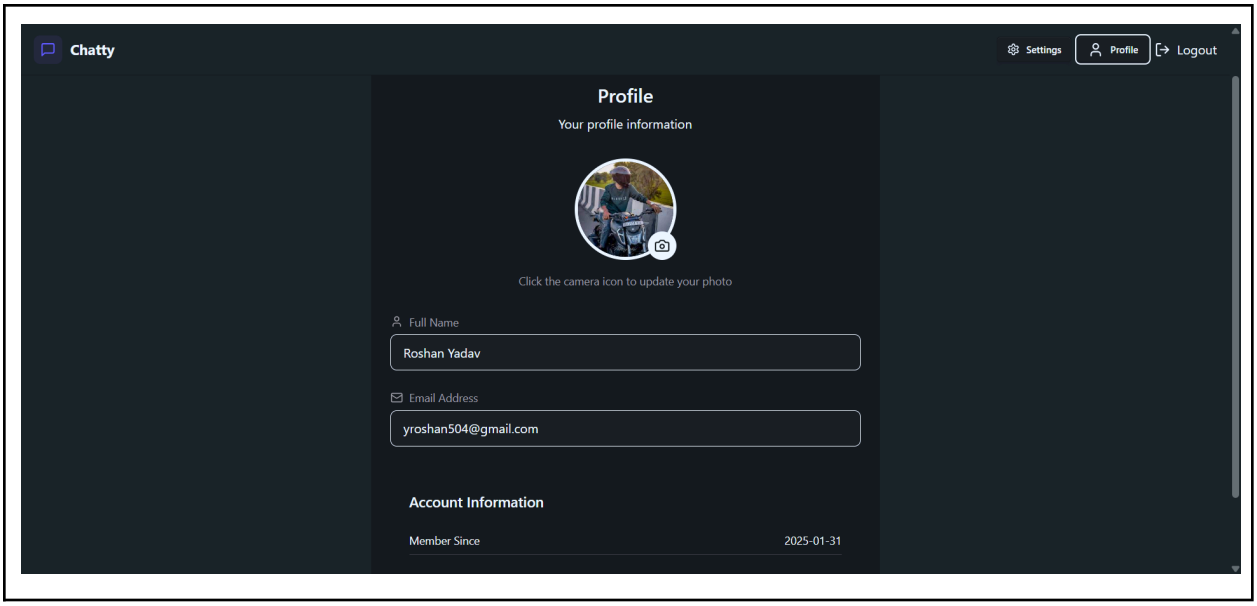


Figure 1.3

Conclusion

By using **React Hooks** (`useEffect`, `useContext`, and custom hooks), state management and side-effect handling become more efficient and reusable. This leads to cleaner, modular, and scalable code. Hooks improve readability, reduce boilerplate, and allow developers to build powerful React applications without class components.