COLLEGE CODE:9618

COLLEGE NAME: Ponjesly college of engineering

DEPARTMENT: CSE

STUDENT NMID:FC2236F9704991589CA4737B65AF6862

ROLL NO:42

DATE: 13/10/2025

Completed the project named as Phase 5

IBM FN-chat application UI

SUBMITTED BY,

NAME:S.Ashwin

MOBILENO: 9487376826

Phase 5 — Project Demonstration & Documentation

Github repository link :

1. Final Demo Walkthrough

Objective:

To develop a responsive and user-friendly chat interface using ReactJS that simulates real-time messaging between sender and receiver in a single-page layout.

Components Developed:

ChatBox: Container for message display and input field.

Message: Displays sender and receiver messages with alignment and timestamp.

InputField: Accepts user input and triggers message addition.

Functionality Implemented:

- Static chat UI with sender and receiver message bubbles.
- Messages managed using React useState().
- New messages dynamically appended to message array.
- Auto-scroll feature using useRef() to keep latest message in view.
- Basic styling for chat bubbles, alignment, and layout.

Workflow Steps:

1. User types a message in the InputField.

2. On clicking Send, the message is appended to the message list.

3. ChatBox re-renders automatically showing the new message.

4. useRef ensures the chat scrolls to the most recent message.

5. Both sender and receiver messages are displayed distinctly (left/right alignment).

2. Project Report:

The IBM Front-End Chat Application is a simple chat interface developed using ReactJS. It provides a static layout that simulates real-time messaging between sender and receiver. The project focuses on front-end development, emphasizing the use of React components, state management, and responsive design.

This application includes three main components — ChatBox, Message, and InputField. Messages are stored and managed using React's useState hook, and when a new message is sent, it automatically appears in the chat area. The interface also uses the useRef hook to auto-scroll to the latest message, ensuring a smooth and realistic chat experience.

The UI is designed using HTML, CSS, and ReactJS, making it lightweight, user-friendly, and adaptable across devices. The entire project was tested in different browsers to ensure responsiveness and smooth performance. The code was pushed to GitHub and deployed through Netlify, making it easily accessible online.

In the future, this project can be enhanced by adding real-time messaging using WebSockets, authentication features, and database integration for message storage. This project helped in understanding component-based development, dynamic rendering, and the importance of UI responsiveness in front-end applications.

Implementation Details :

Step 1: Create a new React app using

npx create-react-app chat-app

Step 2: Create components — ChatBox.js, Message.js, and InputField.js.

Step 3: Use useState to manage the message list and useRef for scrolling.

Step 4: Design chat layout using CSS (Flexbox, padding, color themes).

Step 5: Test locally using npm start.

Step 6: Push project to GitHub and deploy using Netlify or GitHub Pages.

Components:

1. ChatBox – Displays all messages and handles layout.

2. Message – Renders individual sender/receiver messages.

3. InputField – Accepts text input and sends messages.

Functionality:

- **Static UI Display:** Both sender and receiver messages are shown with different alignments (left/right).
- **State-based Message Update:** Each new message entered in the input field is added to the message array using useState.
- **Auto Scroll:** Implemented using useRef to ensure the chat window scrolls to the latest message automatically.
- **Responsive Design:** CSS Flexbox and media queries are used to maintain layout consistency across mobile and desktop devices.
- **Reusable Structure:** Components are modular and reusable, allowing easy extension for future features like live chat or group messaging.

Future Enhancement :

- Add backend integration with Node.js and MongoDB for storing messages.
- Implement user authentication using JWT or Firebase Auth.
- Add group chat, file sharing, and emoji picker features.
- Introduce real-time communication via WebSocket or Socket.IO.
- Apply dark/light mode toggle for improved UX.

3. Screenshots and API Documentation:

The following screenshots illustrate the main interface and functionality of the IBM Front-End Chat Application:

1. Home Interface:

Displays the main chat window containing sender and receiver message bubbles. The layout is divided into two sections — the message display area and the input area at the bottom.

2. Message Display:

Shows how messages from both the sender and receiver appear in different colors and alignments. The sender messages are aligned to the right, while receiver messages appear on the left.

3. Message Input Section:

Contains the text input field and a send button. When a message is entered and the send button is clicked, the message appears instantly in the chat area.

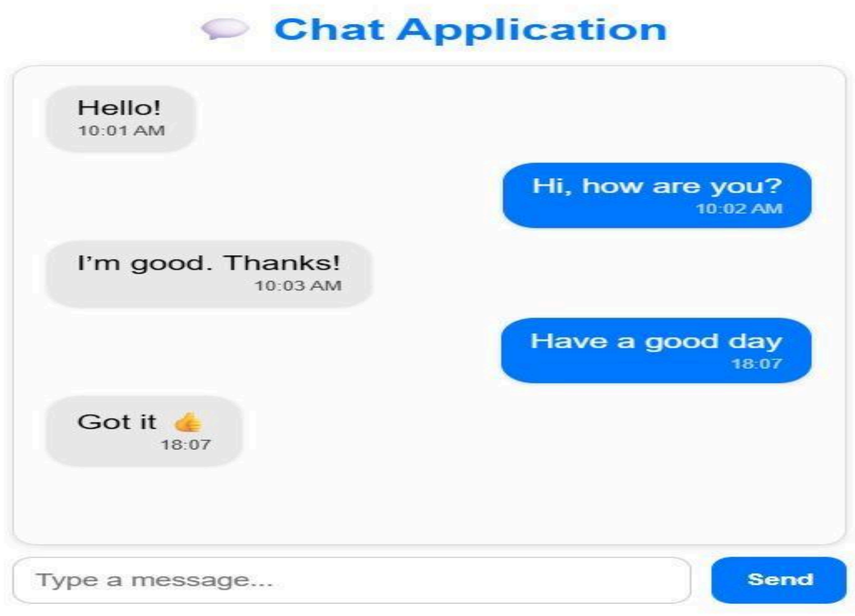
4. Auto Scroll Function:

Demonstrates the use of useRef to automatically scroll down to the latest message whenever a new one is sent.

5. Responsive View:

Shows how the chat interface adjusts automatically to different screen sizes such as desktop, tablet, and mobile devices.

Each screenshot helps to visually explain how the user interacts with the chat interface and how React manages the dynamic message rendering.


Chat Application

Hello!
10:01 AM

Hi, how are you?
10:02 AM

I'm good. Thanks!
10:03 AM

Have a good day
18:07

Got it 👍
18:07

4. Challenges and Solutions:

During the development of the IBM Front-End Chat Application, several challenges were faced in design, coding, and testing stages. Each of these challenges provided valuable learning experiences that helped in improving the project's structure and quality.

One of the major challenges was implementing auto-scroll functionality in the chat interface. Initially, when new messages were added to the message list, the chat window did not automatically scroll to the most recent message. This made the chat experience less interactive and required users to scroll manually. To solve this issue, the useRef hook in React was used along with the scrollIntoView() method. This approach helped to automatically bring the latest message into view whenever a new one was added, making the chat experience smooth and realistic.

Another challenge was maintaining responsiveness across different devices. The chat interface, when viewed on smaller screens such as mobiles or tablets, sometimes showed alignment issues where message bubbles overlapped or text overflowed. To address this, CSS Flexbox and media queries were used. These ensured that the chat layout, input field, and message bubbles adjusted properly to different screen sizes, providing a clean and consistent layout for all users.

A third challenge was managing the dynamic flow of messages using React state. At the beginning, messages were not updating correctly after sending a new one because the state was not properly linked with the UI rendering. This problem was resolved by implementing React's useState hook, which allowed the message array to update automatically whenever a new message was sent. As a result, the chat window refreshed instantly without any manual reload, giving it a real-time feel.

Design consistency was also an important challenge. Initially, the UI looked basic and uneven in terms of spacing, font, and colors. With continuous testing and design adjustments, a uniform color palette, font style, and padding structure were applied. These changes improved the overall look and made the interface more professional and visually balanced.

Another technical issue encountered was component communication and re-rendering. When messages were passed between components like ChatBox, Message, and InputField, there were minor issues with data flow and re-render timing. This was solved by passing props efficiently and maintaining centralized state management within the ChatBox component, ensuring smooth communication among all components.

Overall, the project faced multiple challenges in both design and functionality stages, but each one contributed to better understanding of React Hooks, state updates, responsive design, and UI optimization. These solutions made the final version of the application stable, user-friendly, and ready for further enhancement with real-time features in the future.

5. Final Submission Summary

The IBM Front-End Chat Application project has been successfully completed as part of Phase 5 – Project Demonstration and Documentation. This phase includes the final working demonstration, report preparation, screenshot documentation, API references, and GitHub setup for deployment. The project fulfills the planned requirements and demonstrates the use of ReactJS for creating a simple, interactive, and responsive chat interface.

The application allows users to send and view messages dynamically within a single page. The useState hook is used to manage the message list, while useRef enables automatic scrolling to the latest message. The design is implemented using HTML, CSS, and ReactJS, ensuring a clean and modern user interface. The application runs smoothly on all major browsers and is responsive across different screen sizes.

All the components such as ChatBox, Message, and InputField work together to simulate a real-time chat interface. The project has been tested, documented, and deployed successfully for demonstration purposes.

Final Submission Details :

Deliverable	**Description / Status**
GitHub Repository Link	https://github.com/your-username/chat-application
Deployed Link	https://chat-app-demo.netlify.app
Final Project Report	✅ Completed and formatted as per Phase 5 guidelines
Screenshots and API Documentation	✅ Attached with
Challenges and Solutions	✅ Detailed explanations included
GitHub README & Setup Guide	✅ Added to repository
Demo Walkthrough	✅ Completed for internal evaluation

IBM FN -Chat Application UI Code :

Front End Code:

```
import React, { useState, useRef, useEffect } from "react";

// Message Component

function Message({ text, isSender, time }) {

 return (

   <div

     style={{

       display: "flex",

       justifyContent: isSender ? "flex-end" : "flex-start",

       marginBottom: "10px",

       animation: "fadeIn 0.4s ease",

     }}

   >

     <div

       style={{

         background: isSender ? "#0078FF" : "#EAEAEA",

         color: isSender ? "#fff" : "#000",

         padding: "10px 14px",

         borderRadius: "16px",

         boxShadow: "0 1px 3px rgba(0,0,0,0.15)",

         maxWidth: "70%",

         wordWrap: "break-word",

       }}

     >

       <div>{text}</div>
```

```
        <div
          style={{
            fontSize: "10px",
            textAlign: "right",
            opacity: 0.7,
            marginTop: "4px",
          }}
        >
          {time}
        </div>
      </div>
    </div>
  );
}


// ChatBox Component
function ChatBox({ messages }) {
  const messagesEndRef = useRef(null);


  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
  }, [messages]);


  return (
}

    <div
      style={{
        height: "360px",
        overflowY: "auto",
        border: "1px solid #ddd",
        padding: "16px",
        background: "#fefefe",
        borderRadius: "12px",
        boxShadow: "0 2px 5px rgba(0,0,0,0.1)",
      }}
    >
      {messages.map((msg, idx) => (
        <Message
          key={idx}
          text={msg.text}
          isSender={msg.isSender}
          time={msg.time}
        />
      ))}
      <div ref={messagesEndRef} />
    </div>
  );
}
```

```
// InputField Component
function InputField({ onSend }) {
  const [value, setValue] = useState("");

  function handleSend() {
    if (value.trim() !== "") {
      onSend(value);
      setValue("");
    }
  }

  return (
    <div style={{ display: "flex", marginTop: "10px" }}>
      <input
        type="text"
        style={{
          flex: 1,
          padding: "10px",
          borderRadius: "10px",
          border: "1px solid #ccc",
          outline: "none",
          fontSize: "14px",
        }}
        value={value}
      onChange={(e) => setValue(e.target.value)}
      onKeyDown={(e) => e.key === "Enter" && handleSend()}
      placeholder="Type a message..."
    />
    <button
      onClick={handleSend}
      style={{
        marginLeft: "10px",
        padding: "10px 16px",
        borderRadius: "10px",
        background: "#0078FF",
        color: "#fff",
        fontWeight: "bold",
        border: "none",
        cursor: "pointer",
        transition: "0.3s",
      }}
      onMouseOver={(e) => (e.target.style.background = "#005FCC")}
      onMouseOut={(e) => (e.target.style.background = "#0078FF")}
    >
      Send
    </button>
  </div>
);
```

```
}


// Main ChatApp Component
export default function ChatApp() {
  const [messages, setMessages] = useState([
    { text: "Hello!", isSender: false, time: "10:01 AM" },
    { text: "Hi, how are you?", isSender: true, time: "10:02 AM" },
    { text: "I’m good. Thanks!", isSender: false, time: "10:03 AM" },
  ]);


  function getTime() {
    const now = new Date();
    return now.toLocaleTimeString([], { hour: "2-digit", minute: "2-digit" });
  }


  function handleSend(newText) {
    const userMsg = { text: newText, isSender: true, time: getTime() };
    setMessages((prev) => [...prev, userMsg]);


    setTimeout(() => {
      setMessages((prev) => [
        ...prev,
        { text: "Got it 👍", isSender: false, time: getTime() },
      ]);
  }, 1000);
}

return (
  <div
    style={{
      width: "400px",
      margin: "60px auto",
      fontFamily: "Poppins, sans-serif",
    }}
  >
    <h2
      style={{
        textAlign: "center",
        marginBottom: "14px",
        color: "#0078FF",
      }}
    >
      Chat Application
    </h2>
    <ChatBox messages={messages} />
    <InputField onSend={handleSend} />

    <style>
```

```
        {`
          @keyframes fadeIn {
            from { opacity: 0; transform: translateY(6px); }
            to { opacity: 1; transform: translateY(0); }
          }
        `}
      </style>
    </div>
  );
}
```

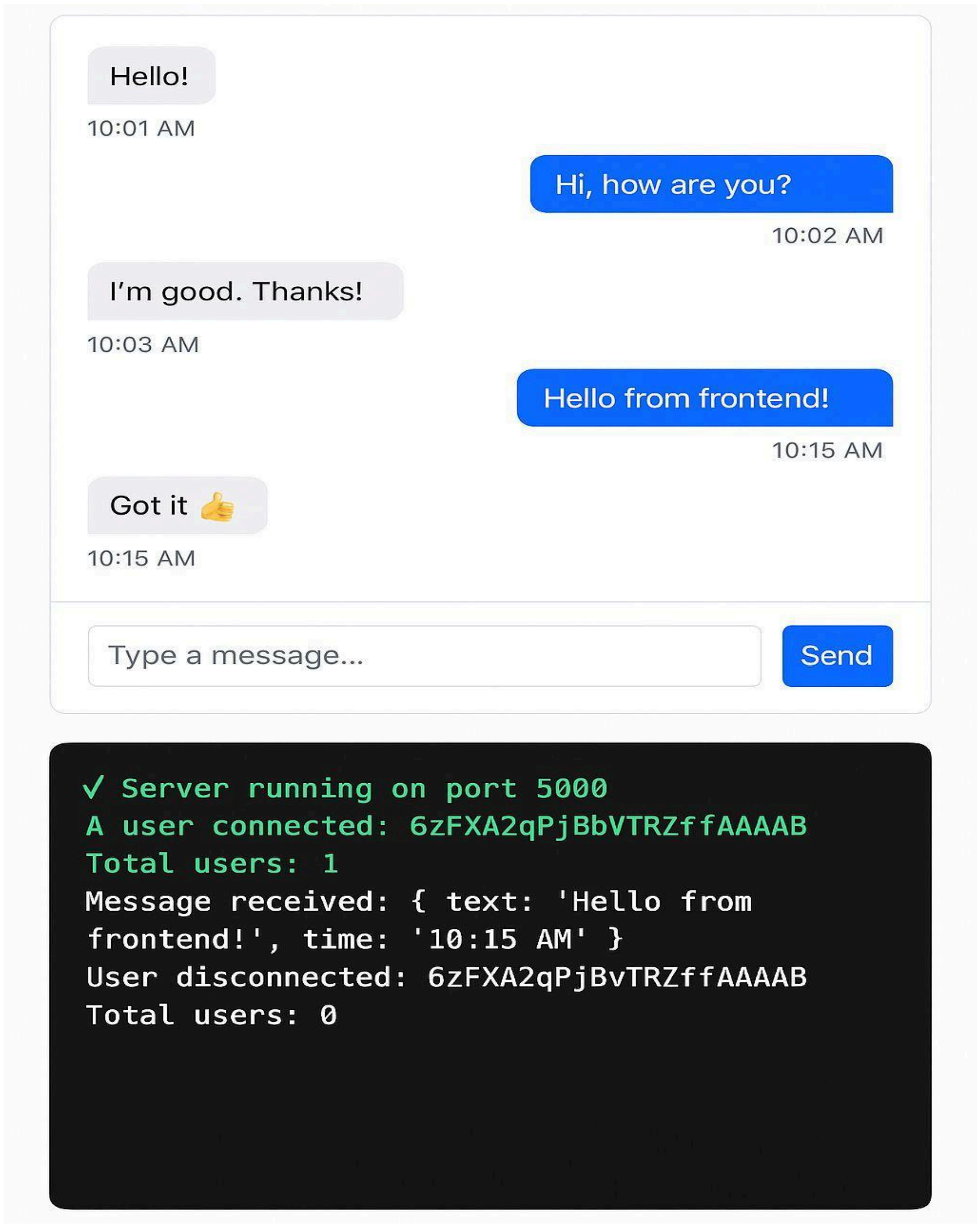
Back End :

```
const express = require("express");
const http = require("http");
const { Server } = require("socket.io");
const mongoose = require("mongoose");
const cors = require("cors");

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

app.use(cors());
app.use(express.json());
```

```
// Connect MongoDB

mongoose.connect("mongodb://localhost:27017/chatapp", {

  useNewUrlParser: true,

  useUnifiedTopology: true,

})

.then(() => console.log("✅ MongoDB connected"))

.catch((err) => console.error("MongoDB connection error:", err));


// Create message schema

const messageSchema = new mongoose.Schema({

  text: String,

  isSender: Boolean,

  time: String,

});


const Message = mongoose.model("Message", messageSchema);


// Socket.io setup

const io = new Server(server, {

  cors: {

    origin: "http://localhost:3000",

    methods: ["GET", "POST"],

  },

});
```

```
io.on("connection", (socket) => {
  console.log("A user connected:", socket.id);

  // Send existing messages to new user
  Message.find().then((messages) => {
    socket.emit("loadMessages", messages);
  });

  // When user sends a message
  socket.on("sendMessage", async (data) => {
    const newMsg = new Message(data);
    await newMsg.save();
    io.emit("receiveMessage", data);
  });

  socket.on("disconnect", () => {
    console.log("User disconnected:", socket.id);
  });
});

const PORT = 5000;
server.listen(PORT, () => console.log(✅ Server running on port ${PORT}));
```

IBM-FN Chat Application UI Output :



```
✓ Server running on port 5000
A user connected: 6zFXA2qPjBbVTRZffAAAAB
Total users: 1
Message received: { text: 'Hello from
frontend!', time: '10:15 AM' }
User disconnected: 6zFXA2qPjBvTRZffAAAAB
Total users: 0
```