## Experiment No. 8

**Aim:** Enable real-time communication via WebSocket

**Code :**

backend > JS server.js > ...

```
34
35    //routes
36    app.get('/',(req,res)=>{
37        res.send('hello from server !');
38    })
39
40    app.use("/api/v1/auth",AuthRoutes);
41    app.use("/api/v1/user",UserRoutes);
42    app.use("/api/v1/chat", chatRoutes);
43    app.use("/api/v1/group",GroupRoutes);
44
45    //running:
46    const port=process.env.PORT;
47    const mongourl=process.env.MONGO_URI;
48    app.listen(port, async()=>{
49        try {
50            const connection=await mongosse.connect(mongourl);
51            if(connection)
52            {
53                console.log(`Server running on http://localhost:${port}`);
54                console.log('MongoDB connected successfully')
55            }
```

**Fig 1:** API endpoints defined in server.js using REST principles with mongoose connection

```
export function attachWebSocketServer(server) {

                    const history = await Chat.find({ sosId }).sort({ createdAt: 1 });
                    ws.send(JSON.stringify({
                        type: "chat_history",
                        payload: history.map(msg => ({
                            sender: msg.senderType,
                            text: msg.message,
                            timestamp: msg.createdAt,
                        }))
                    }));
                }

            } catch (e) {
                console.error("Failed to process message:", e);
                ws.send(JSON.stringify({ type: "error", message: "Invalid message format" }));
            }
        });

        ws.on("close", () => {
            if (userId && connectedUsers.get(userId)?.dataIntervalId) {
                clearInterval(connectedUsers.get(userId).dataIntervalId);
            }
            connectedUsers.delete(userId);
            console.log(`❌ User disconnected: ${userId}`);
        });

        ws.on("error", (err) => console.error("WS Error:", err));
    });

    console.log("🚀 WebSocket Server is running.");
}
```

**Fig. 2:** ws server connection setup in backend

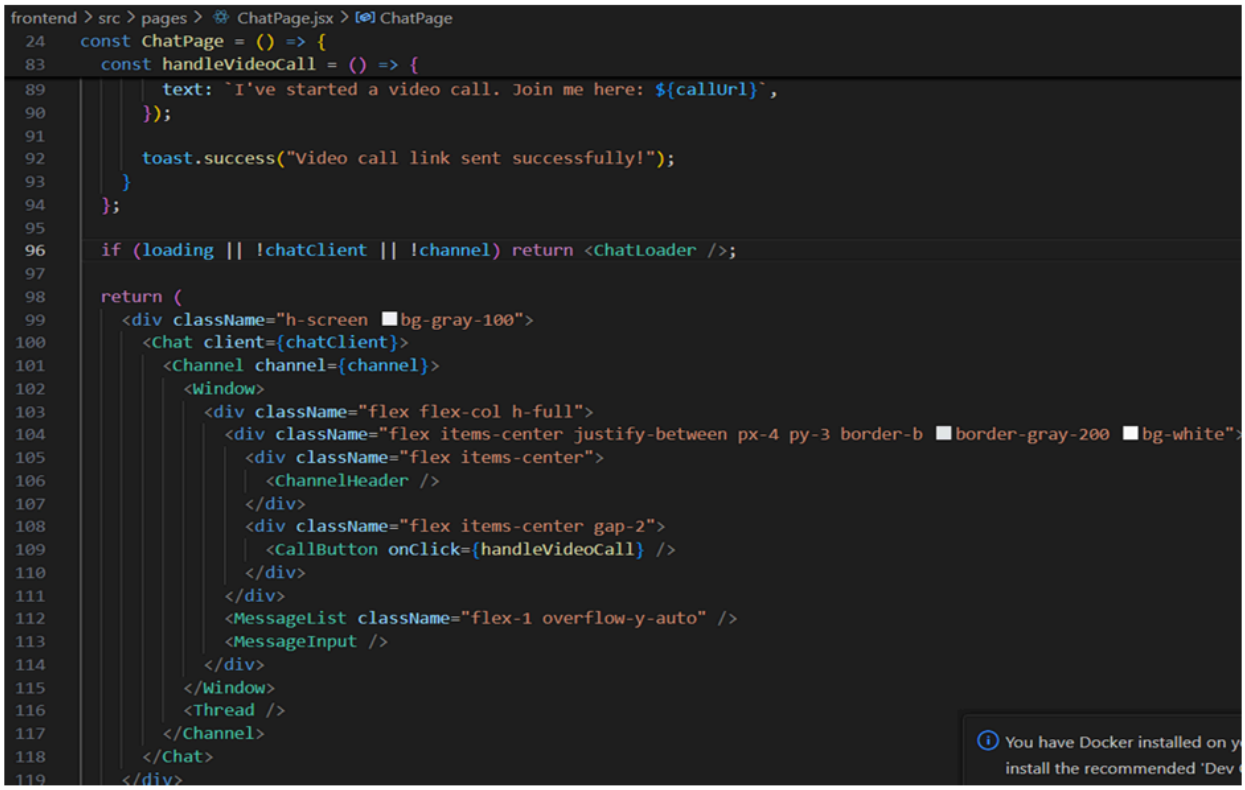


**Fig 3:** Frontend Chatscreen UI setup.

**Output:**

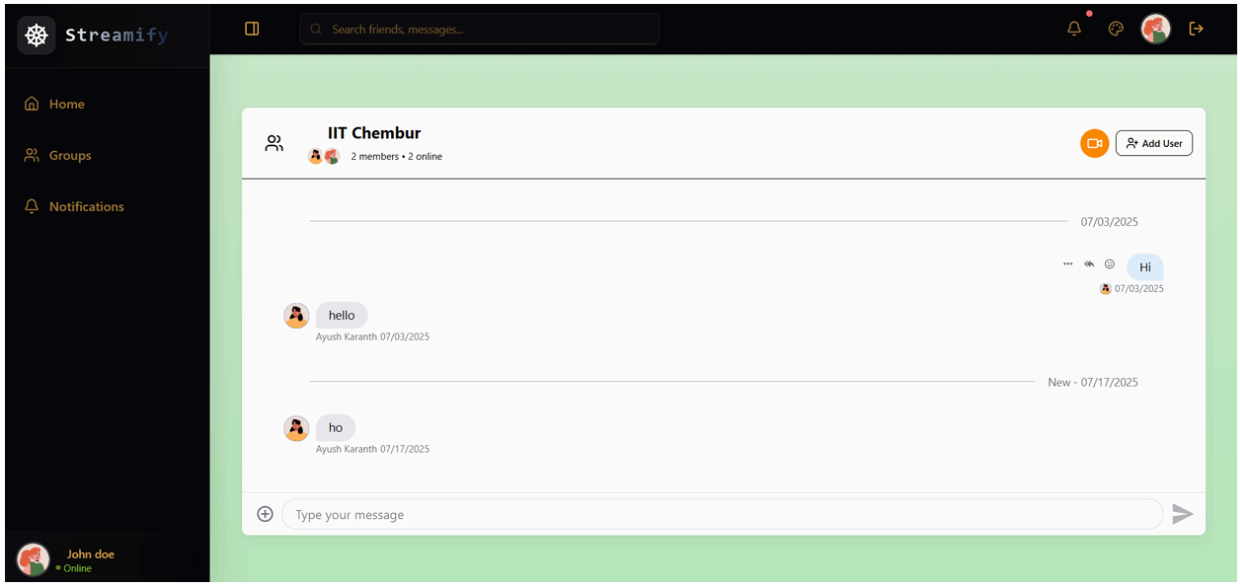


**Fig 4:** Group Chat Screen

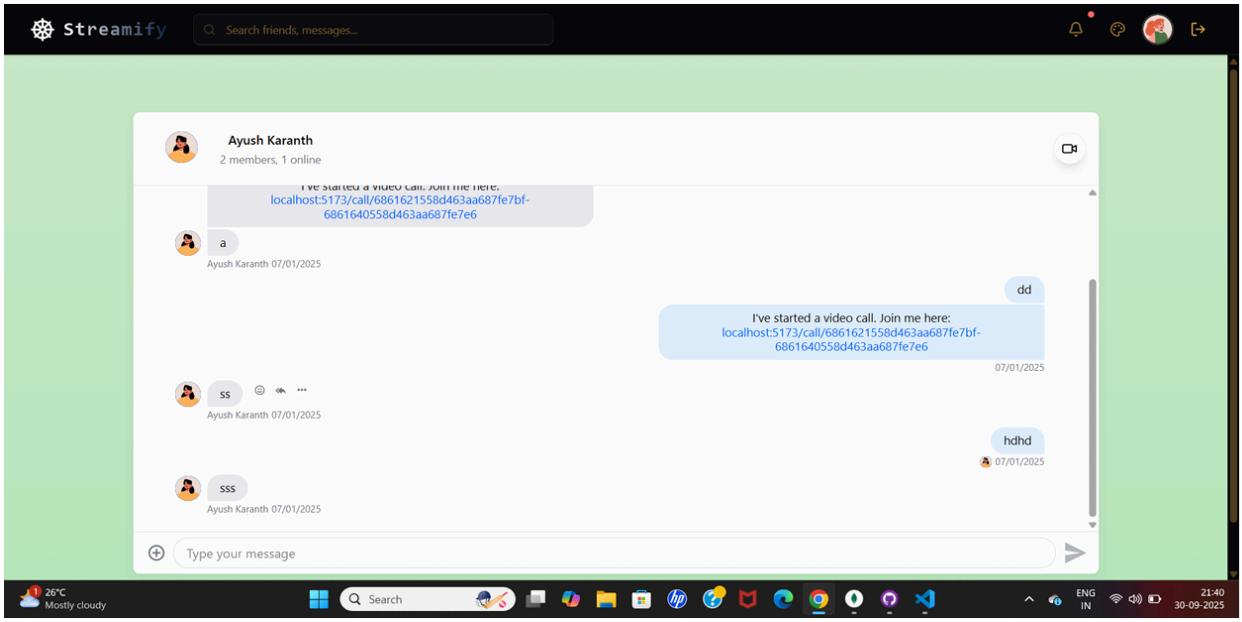


**Fig 5:** One to One chat screen.

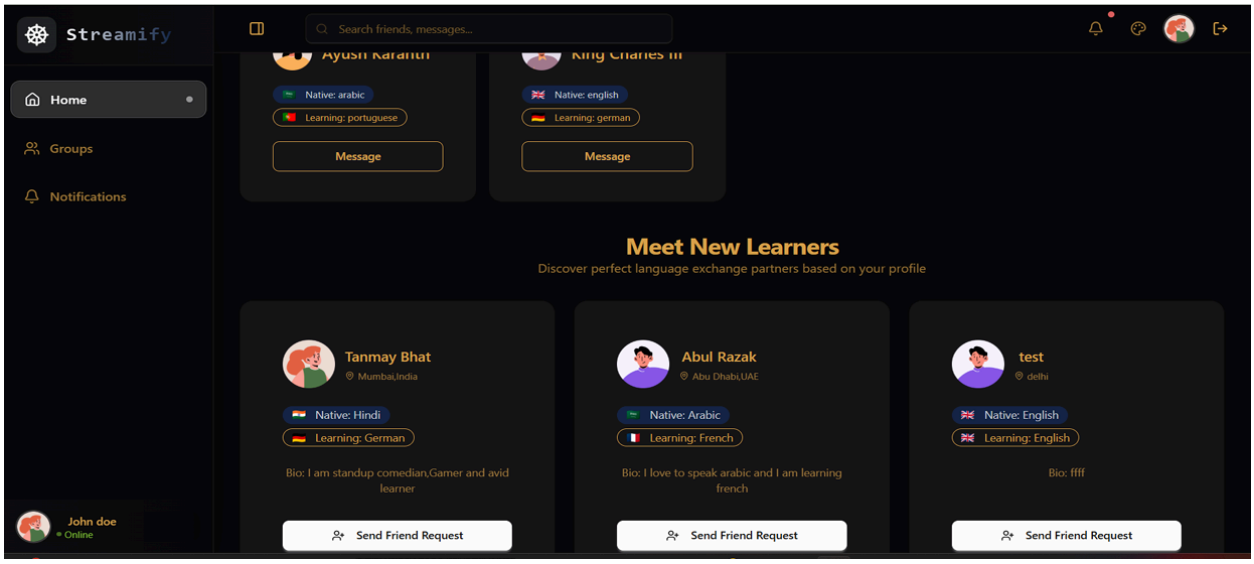


**Fig 6**: Home Page of Application

## Conclusion :

In conclusion, enabling real-time communication via **WebSocket** provides a fast and efficient way to exchange data between clients and servers. Unlike traditional HTTP requests, WebSockets maintain a persistent connection, allowing instant two-way communication. This makes them ideal for chat apps, live notifications, gaming, and real-time data updates. Overall, WebSockets enhance user experience by delivering seamless, low-latency, and interactive communication in modern web applications.