Công ty cổ phần VCCorp

BÁO CÁO TUẦN 3

Tìm hiểu về websocket và dựng một hệ thống chat dùng websocket

Tác giả: Ngô Minh Đức

Người hướng dẫn: Anh Ngô Văn Vĩ

Hà Hội, 6/2025

Tóm tắt nội dung

Báo cáo này trình bày những nội dung quan trọng về khái niệm, đặc điểm, cấu trúc, các phương pháp và ứng dụng của Websocket

MỤC LỤC

Chương 1: Giới thiệu chung về Socket

Socket là một điểm cuối (endpoint) của một liên kết giao tiếp hai chiều giữa hai tiến trình (process) chạy trên mạng. Nó đóng vai trò như một "cổng giao tiếp" cho phép các ứng dụng hoặc tiến trình truyền và nhận dữ liệu qua mạng.Việc nắm vững Prompt Engineering không chỉ giúp tiết kiệm thời gian và chi phí mà còn góp phần tăng hiệu suất phát triển ứng dụng và khai thác tối đa sức mạnh của AI.

Socket thường được kết hợp với một địa chỉ IP và một số hiệu cổng (port) cụ thể để định danh vị trí giao tiếp. Các hệ điều hành như Windows, Linux đều hỗ trợ socket và nó được sử dụng trong nhiều ngôn ngữ lập trình như C, Java, Python, Node.js,... Socket có thể được chạy đồng thời (multi-socket), giúp cải thiện hiệu suất hệ thống.

Có 2 loại Socket là Stream Socket (TCP Socket) và Datagram Socket (UDP Socket). Trong đó: Stream Socket (TCP Socket) là loại socket dựa trên giao thức TCP (Transmission Control Protocol), một giao thức hướng kết nối. Điều này có nghĩa là trước khi có thể truyền dữ liệu, cần phải thiết lập một kết nối ổn định giữa client và server. TCP đảm bảo rằng dữ liệu được truyền đi một cách đầy đủ, đúng thứ tự và đáng tin cậy, đồng thời có cơ chế xác nhận phản hồi (acknowledgement) và phát hiện lỗi để kiểm soát luồng dữ liệu. Do tính ổn định và đáng tin cậy,

Stream Socket thường được sử dụng trong các ứng dụng yêu cầu độ chính xác cao như truyền tệp, gửi email, hoặc các giao dịch tài chính – ngân hàng. Trong mô hình client-server, server sẽ luôn trong trạng thái lắng nghe kết nối, trong khi client là phía chủ động yêu cầu kết nối. Datagram Socket (UDP Socket) ngược lại dựa trên giao thức UDP (User Datagram Protocol), một giao thức không hướng kết nối. Loại socket này cho phép gửi dữ liệu mà không cần thiết lập kết nối trước, dẫn đến việc truyền tải cực kỳ nhanh chóng. Tuy nhiên, UDP không đảm bảo rằng dữ liệu sẽ đến nơi, đến đúng thứ tự, hay đến nguyên vẹn. Gói tin có thể bị trễ, mất mát hoặc đến sai thứ tự mà không có cơ chế kiểm soát hay xác nhận nào. Chính vì vậy, Datagram Socket phù hợp với các ứng dụng thời gian thực như video streaming, voice call, hoặc các trò chơi trực tuyến – nơi tốc độ truyền tải được ưu tiên hơn độ chính xác tuyệt đối.

Chương 2: Khái niệm của WebSocket

WebSocket là một giao thức truyền thông mạng hoạt động trên TCP, hỗ trợ giao tiếp hai chiều (full-duplex) giữa client và server thông qua một kết nối duy nhất. Mặc dù WebSocket được thiết kế chủ yếu cho web, nó vẫn có thể sử dụng trong các ứng dụng desktop hoặc di động

Chương 3: Đặc điểm

Quy trình handshake (bắt tay)

- Kết nối WebSocket bắt đầu bằng một HTTP request đặc biệt từ client.
- Nếu server chấp nhận, kết nối được nâng cấp từ HTTP lên WebSocket.

Client gửi request:

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Version: 13

Origin: http://example.com

Server trả lời:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

Sau khi kết nối: Client và server có thể gửi dữ liệu cho nhau bất cứ lúc nào, không cần thiết lập lại kết nối như HTTP.

Chương 4: Cấu trúc cơ bản

WebSocket mang lại nhiều lợi thế vượt trội trong các ứng dụng yêu cầu giao tiếp theo thời gian thực. Một trong những điểm mạnh lớn nhất của WebSocket là khả năng duy trì kết nối liên tục giữa client và server. Sau khi hoàn tất bước bắt tay ban đầu (handshake), WebSocket thiết lập một kết nối TCP ổn định mà không cần tái khởi động lại kết nối cho mỗi yêu cầu, giúp tiết kiệm đáng kể chi phí xử lý và độ trễ.

Một lợi ích quan trọng khác là độ trễ rất thấp. Ngay sau khi kết nối được thiết lập, dữ liệu có thể được gửi và nhận tức thì mà không cần các vòng lặp HTTP như ở RESTful API. Đây là yếu tố then chốt đối với các ứng dụng real-time như trò chuyện trực tuyến (chat), trò chơi nhiều người chơi, hoặc hệ thống theo dõi số liệu trực tiếp (real-time dashboard).

Bên cạnh đó, WebSocket giúp tiết kiệm băng thông đáng kể. Khác với HTTP, nơi mỗi request/response đi kèm header cồng kềnh, WebSocket chỉ sử dụng header rất nhỏ (khoảng 2 byte sau khi kết nối), giúp giảm tải cho mạng, đặc biệt trong các ứng dụng có lượng trao đổi dữ liệu cao hoặc liên tục.

Từ phía lập trình viên, WebSocket cung cấp API đơn giản, dễ tiếp cận, đặc biệt trong JavaScript. Các phương thức và sự kiện như send(), onmessage, onopen, onclose, onerror đều dễ sử dụng và không đòi hỏi thêm thư viện phụ trợ. Điều này rút ngắn thời gian phát triển và triển khai tính năng. Chính vì vậy, WebSocket ngày càng được ứng dụng rộng rãi trong các hệ thống như ứng dụng trò chuyện, hệ thống thông báo thời gian thực, trò chơi online, cập nhật thị trường chứng khoán, hoặc bất kỳ nền tảng nào cần phản hồi nhanh.

Chương 5: Nhược điểm

Mặc dù mang lại nhiều lợi ích, WebSocket vẫn tồn tại một số hạn chế kỹ thuật đáng lưu ý. Trước hết, khả năng tương thích với proxy hoặc tường lửa chưa thực sự hoàn thiện. Một số hệ thống mạng sử dụng tường lửa hoặc proxy HTTP không cho phép hoặc giới hạn lưu lượng WebSocket, do đó có thể gây gián đoạn hoặc ngắt kết nối trong môi trường mạng bảo mật cao.

Ngoài ra, WebSocket không tận dụng tốt cơ chế session có sẵn trong nhiều framework web phổ biến như Hibernate, Spring MVC hay các framework dựa trên mô hình request/response truyền thống. Do WebSocket hoạt động như một kết nối TCP liên tục, không phải là các HTTP request độc lập, nên các cơ chế như session scope, filter hay interceptor hoạt động không hiệu quả hoặc không áp dụng được như trong REST API.

Cuối cùng, việc scale hệ thống sử dụng WebSocket cũng phức tạp hơn so với các hệ thống dựa trên RESTful API. Do kết nối phải được duy trì liên tục giữa client và server, các giải pháp như load balancing, clustering hoặc scaling ngang cần được thiết kế cẩn thận. Một số máy chủ không hỗ trợ tốt số lượng kết nối đồng thời lớn, hoặc cần triển khai thêm các hệ thống trung gian như WebSocket gateway hoặc message broker (như Redis, Kafka, RabbitMQ) để đảm bảo khả năng mở rộng.

Chương 6: Một số thành phần của WebSocket

6.1 Các thuộc tính chính

Thuộc tính	Mô tả
readyState	Trạng thái hiện tại của kết nối WebSocket: - 0 (CONNECTING): Đang trong quá trình kết nối - 1 (OPEN): Kết nối đã mở và sẵn sàng giao tiếp - 2 (CLOSING): Kết nối đang trong quá trình đóng - 3 (CLOSED): Kết nối đã bị đóng hoặc không thể mở được

bufferedAmount	Số byte dữ liệu đã được gửi (queued) nhưng vẫn đang chờ truyền tới server

6.2 Các sự kiện chính

Sự kiện	Handler	Mô tả
Open	Onopen	Khi kết nối WebSocket được mở thành công
Message	Onmessage	Khi nhận dữ liệu từ server
Error	Onerror	Khi có lỗi trong giao tiếp
Close	Onclose	Khi có lỗi trong giao tiếp

6.3 Các phương thức chính

Phương thức	Mô tả
Send(data)	Gửi dữ liệu đến server (dạng string, ArrayBuffer, hoặc blob)
close()	Đóng kết nối hiện tại với server

Chương 7: Dựng một hệ thống chat sử dựng WebSocket

Link source github: https://github.com/theducminh/B-o-c-o-th-c-t-p-c-ng-ty-VCCorp/tree/test/B-o-c-o-th-c-t-p-c-ng-ty-VCCorp/ChatWeb

7.1 Giới thiệu

Hệ thống chat real-time cho phép nhiều người dùng giao tiếp với nhau tại cùng thời điểm. Dự án này sử dụng Socket.IO để giao tiếp hai chiều giữa client và

server, kết hợp ExpressJS, MongoDB cho backend, và React Native (Expo) cho frontend.

7.2 Kiến trúc hệ thống

- Backend:
  • ExpressJS xử lý các API REST như tạo user, tạo room, lấy danh sách room.
  • MongoDB lưu trữ thông tin người dùng, phòng chat và tin nhắn.
  • Socket.IO xử lý các kết nối real-time.
- Frontend:
  • Được xây dựng bằng React Native với Expo CLI.
  • Giao diện gồm 3 màn hình chính: Login, RoomList, Chat.
  • Sử dụng Socket.IO client để giao tiếp với server theo thời gian thực.

7.3 Cài đặt Backend

- Thư mục `backend` chứa mã nguồn backend.
- Các tệp chính:
  • server.ts: khởi tạo app, socket, kết nối MongoDB.
  • app.ts: thiết lập Express app và middleware.
  • socket.ts: xử lý logic kết nối socket.
  • models/: chứa schema của User, Room, Message.
  • routes/: định nghĩa các API như /users, /rooms.

-Các API chính:

- GET /api/users: Đăng nhập với username
- POST /api/rooms: Tạo room chat
- GET /api/rooms/:userId: Lấy danh sách room user tham gia

-Socket Events

- register: mapping userId <-> socketId
- joinRoom: tham gia vào room chat
- sendMessage: gửi tin nhắn tới server
- receiveMessage: gửi tin nhắn tới client

- Cài đặt và chạy:

- Cài đặt phụ thuộc: npm install express mongoose socket.io cors dotenv và npm install -D typescript ts-node ts-node-dev @types/node @types/express @types/socket.io
- Kết nối MongoDB Atlas: tạo file .env có MONGO_URI=mongodb+srv://<user>:<password>@cluster.mongodb.net/chatdb
- Chạy backend trên máy: npm run dev
- Build dist (chứa file js): npm run build
- Chạy backend (bằng file js): npm start

7.4 Cài đặt Frontend

- Thư mục `frontend` chứa mã nguồn React Native.
- Cấu trúc chính:
  • App.tsx: định nghĩa Navigation Stack
  • screens/: chứa các màn hình Login, RoomList, Chat
  • types/: định nghĩa interface Room, Message, Props

- Các màn hình

- LoginScreen: Nhập username, đăng nhập
- RoomListScreen: Lọc danh sách room theo userId
- ChatScreen: Nhận/gửi tin nhắn real-time

- Triển khai và chạy:

- Cài đặt phụ thuộc:
  npx create-expo-app frontend -t expo-template-blank-typescript

  cd frontend

  npm install @react-navigation/native @react-navigation/native-stack

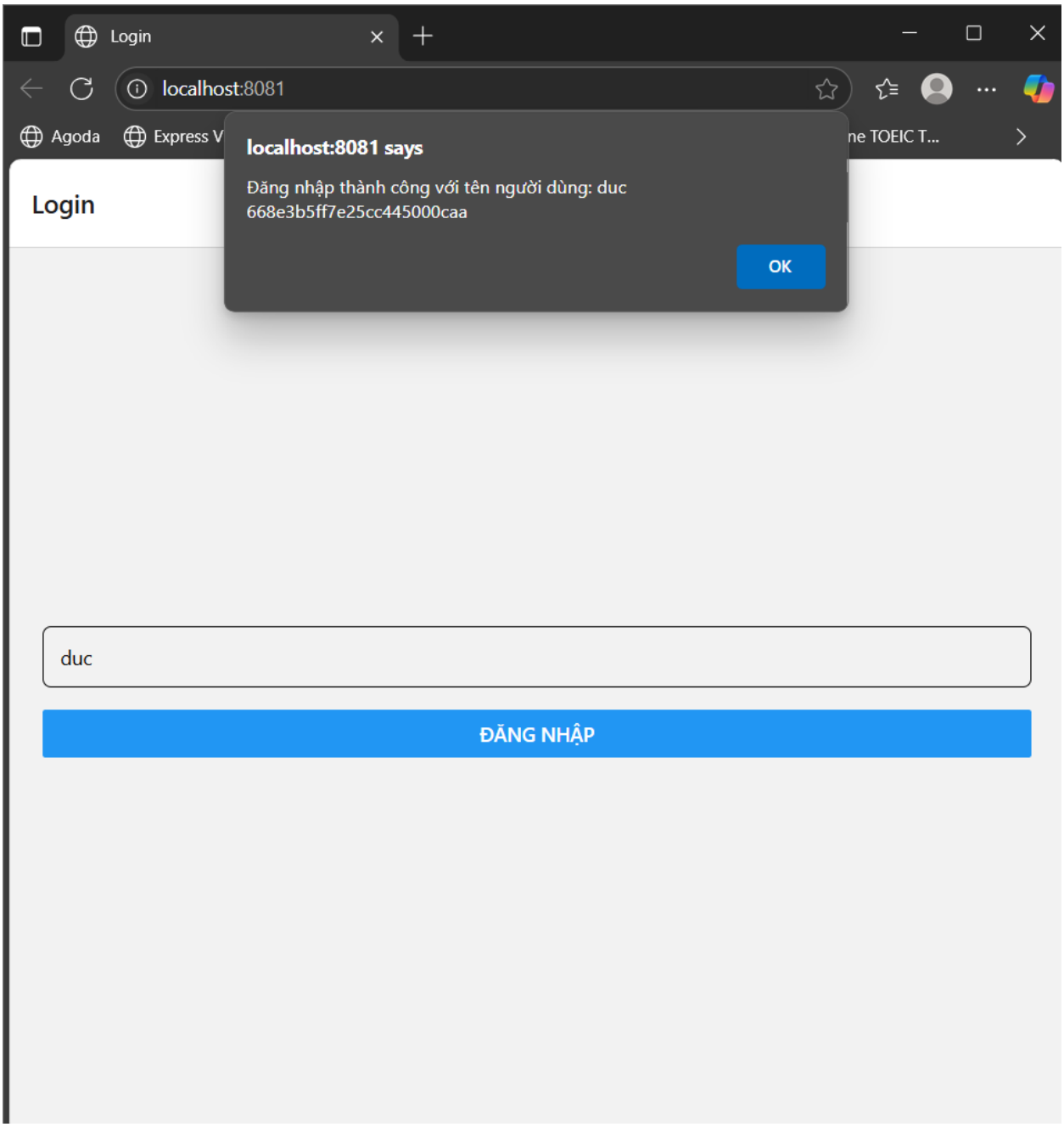
  npm install socket.io-client axios
- Chạy: cd frontend
  npx expo start --web

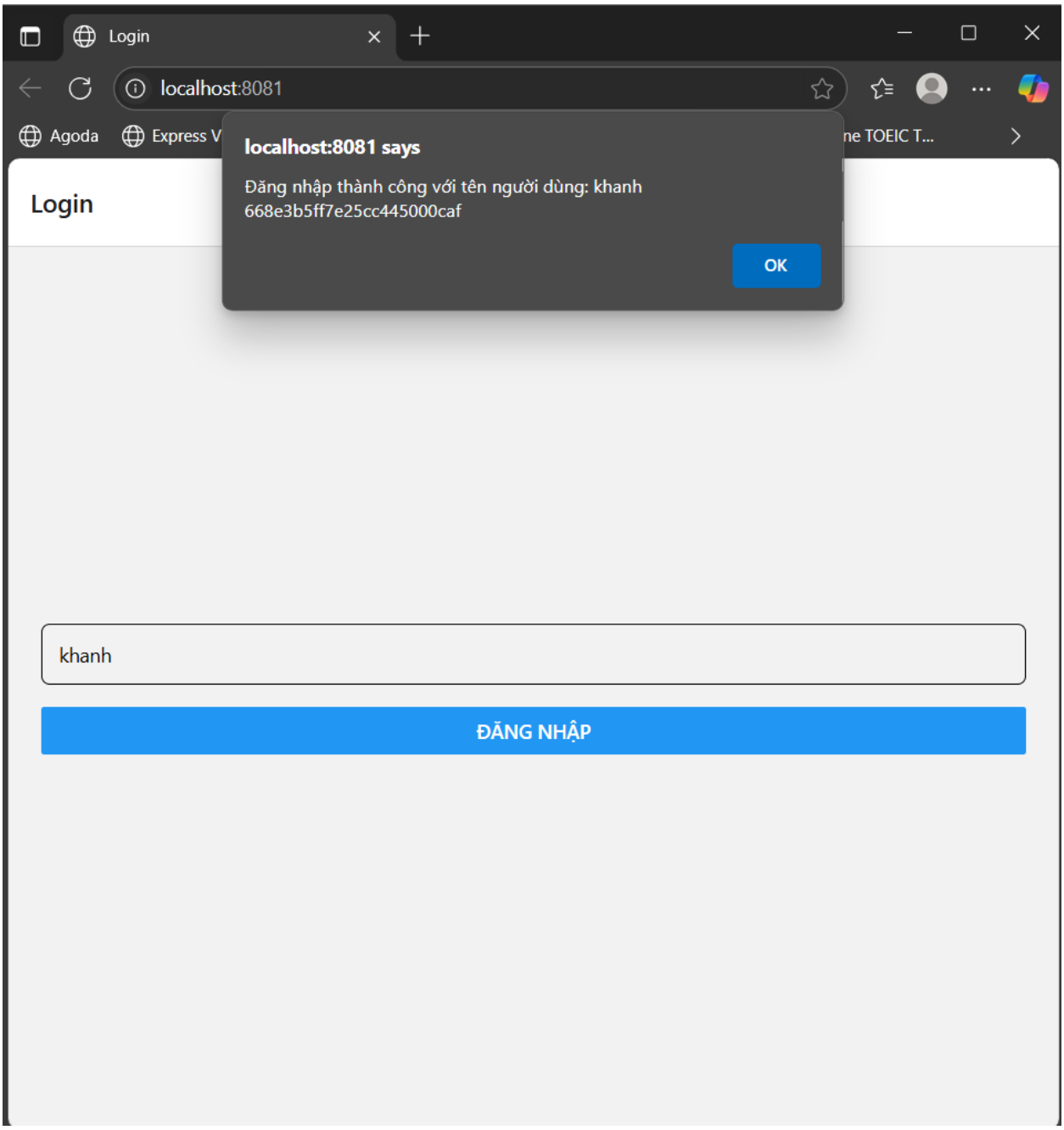
7.5 Tính năng chính

- Đăng nhập bằng username.
- Xem danh sách các phòng đã tham gia.
- Tham gia phòng và gửi/nhận tin nhắn theo thời gian thực.
- Tự động gán socket ID theo userId để gửi nhận tin nhắn.

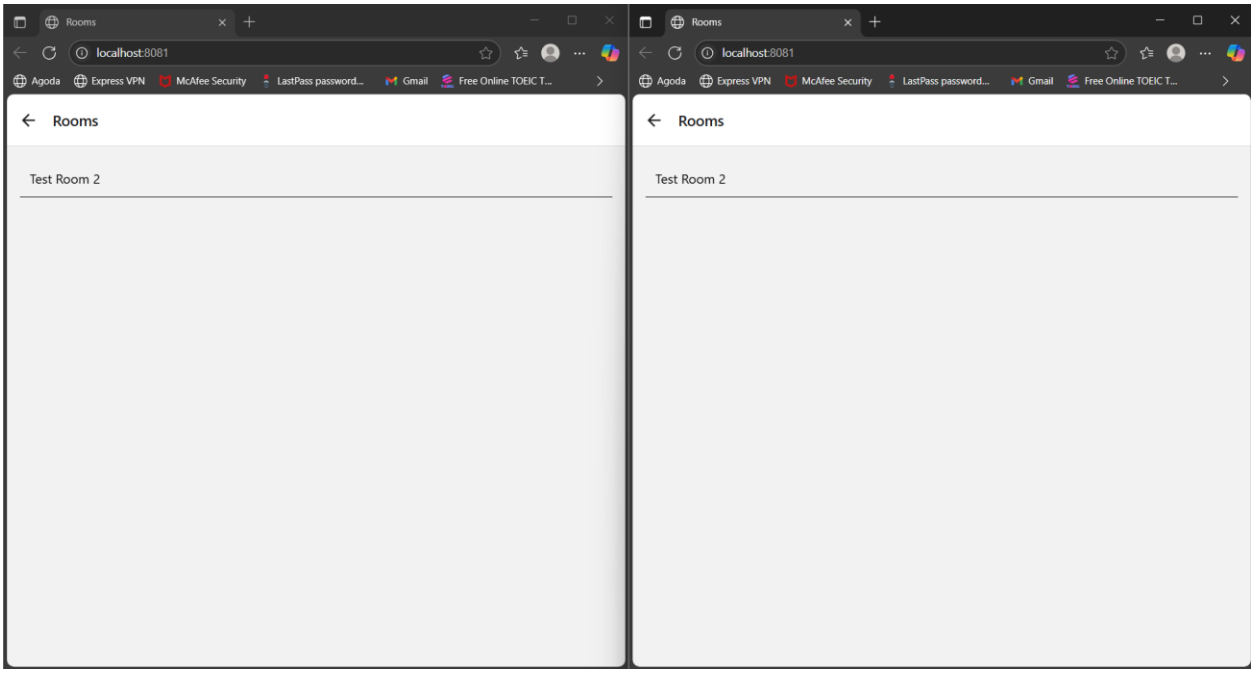
7.6 Kết quả

- Màn hình đăng nhập: là màn hình đầu tiên khi chạy hệ thống, dùng hai màn hình để đăng nhập 2 tài khoản khác nhau. Đăng nhập thành công sẽ chuyển tới màn hình Room chứa danh sách Phòng Chat mà người dùng đã tham gia

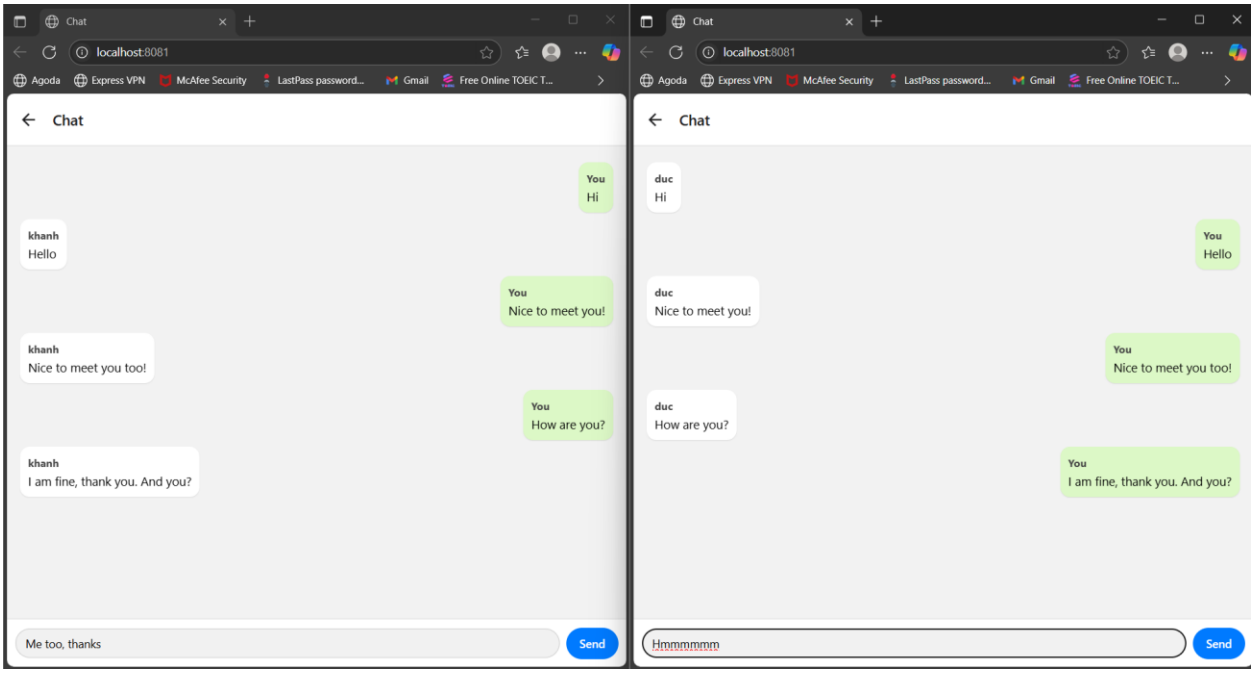

Login
localhost:8081
Agoda
Express V
ne TOEIC T...
localhost:8081 says
Đăng nhập thành công với tên người dùng: duc
668e3b5ff7e25cc445000caa
OK
Login
duc
ĐĂNG NHẬP



-Màn hình danh sách Phòng Chat tham gia: sẽ hiển thị RoomList mà người dùng tham gia sau đó chọn một Room bất kì để chuyển sang Chat Room để nhắn tin. Ở đây cả người dùng khanh và duc đều tham gia "Test Room 2" nên có click chọn "Test Room 2" ở cả hai tài khoản

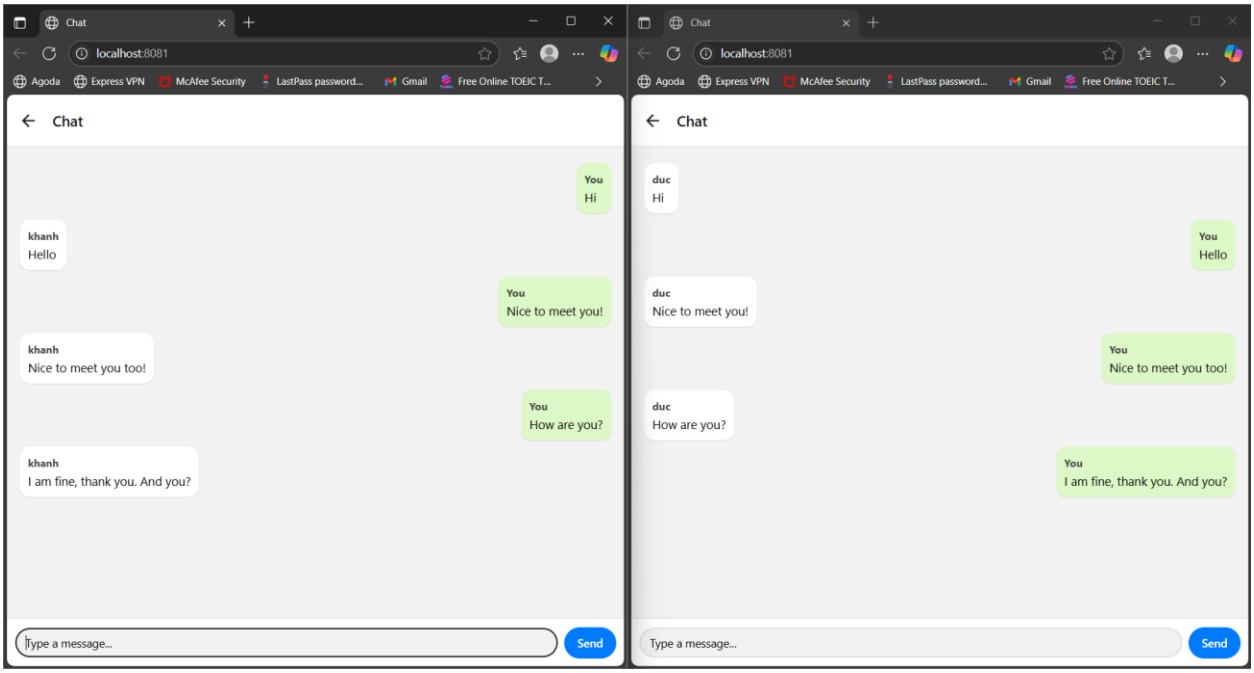


-Màn hình Chat: Hai người dùng duc và khanh có thể nhắn tin với nhau cùng lúc trong giao diện Chat này:

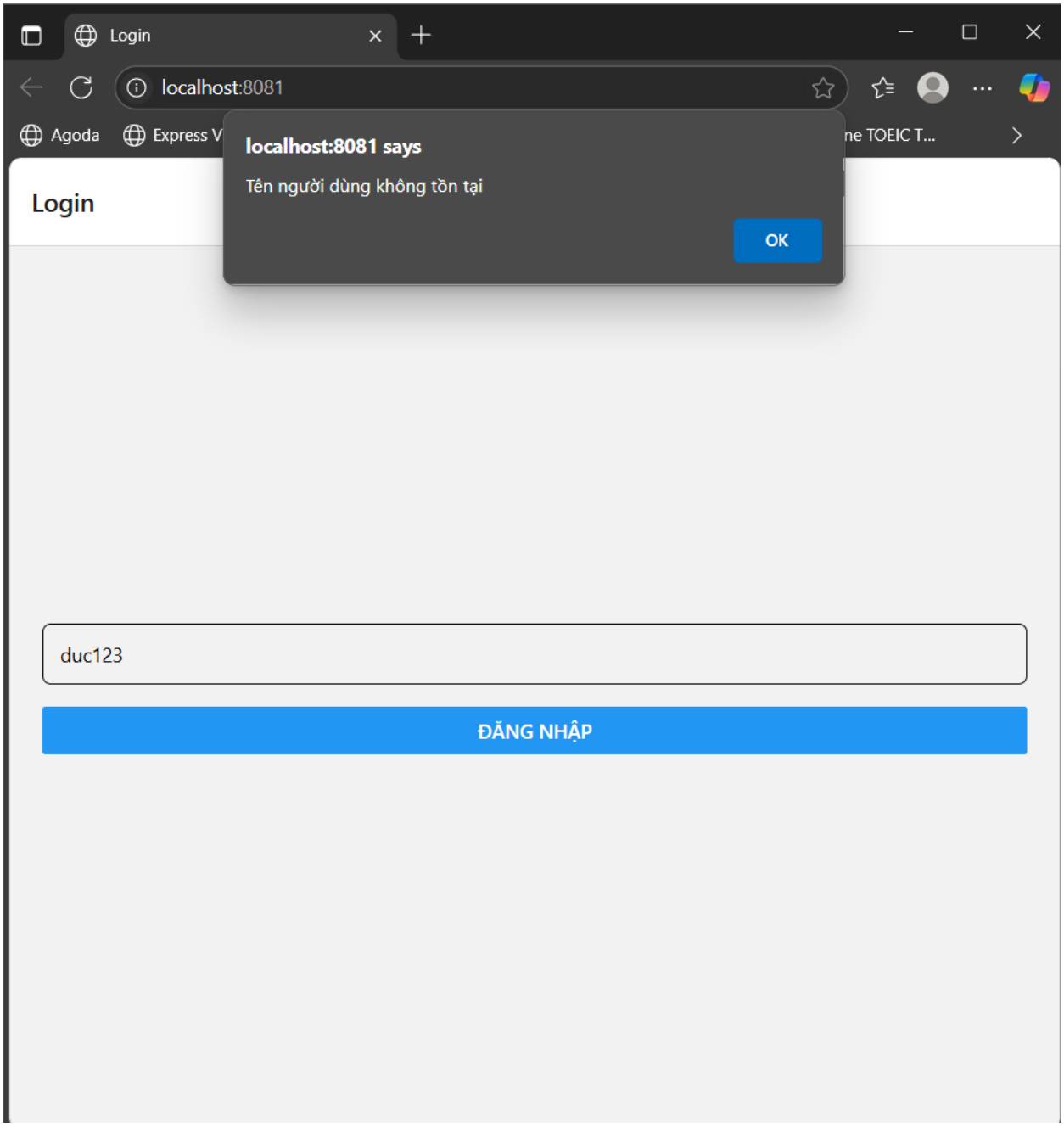


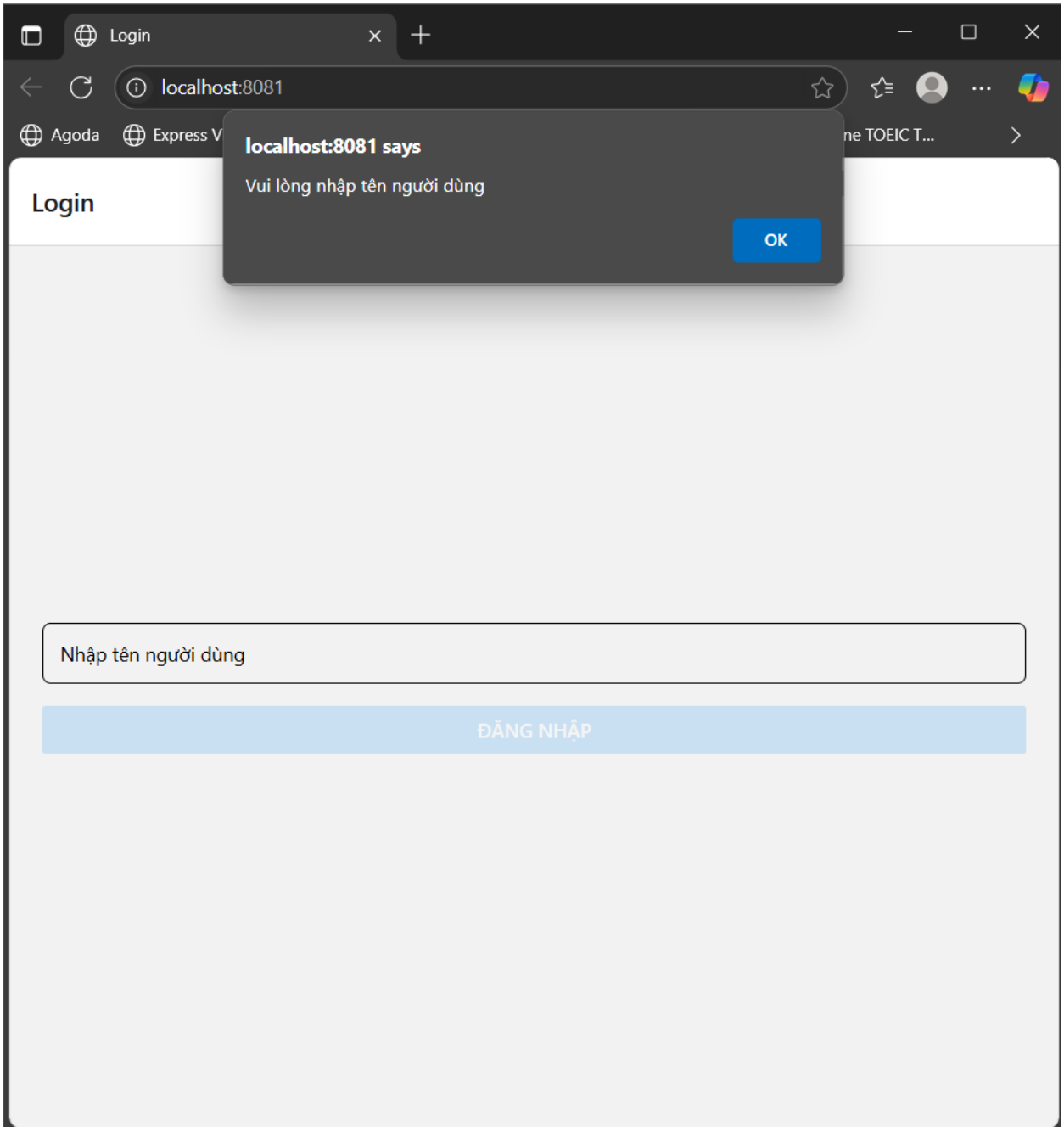
7.7 Kiểm thử

-Khi không nhập mà vẫn nhấn “Send”: không gửi tin nhắn



-Khi đăng nhập với username không tồn tại hoặc để trống: thông báo lỗi


Login
localhost:8081
Agoda
Express V
ne TOEIC T...
localhost:8081 says
Tên người dùng không tồn tại
OK
Login
duc123
ĐĂNG NHẬP


Login
localhost:8081
Agoda
Express V
ne TOEIC T...
localhost:8081 says
Vui lòng nhập tên người dùng
OK
Login
Nhập tên người dùng
ĐĂNG NHẬP

Chương 8: Kết luận

WebSocket là giải pháp tối ưu cho các ứng dụng thời gian thực trong web hiện đại, giúp tiết kiệm băng thông, giảm độ trễ và cải thiện trải nghiệm người dùng. Việc hiểu rõ sự khác biệt giữa các loại socket và cách hoạt động của WebSocket là cần thiết để thiết kế hệ thống hiệu quả, đặc biệt trong các ứng dụng yêu cầu realtime cao.