VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

MIDTERM

WEB PROGRAMMING WITH NODEJS

Instructor: **PhD. LÊ VĂN VANG**

Implementer: **HỒ MINH CHÍ TÂN – 519H0044**

TRẦN NHỰT ANH – 521H0491

HO CHI MINH CITY, 2023

VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

MIDTERM

WEB PROGRAMMING WITH NODEJS

Instructor: **PhD. LÊ VĂN VANG**

Implementer: **HỒ MINH CHÍ TÂN – 519H0044**

TRẦN NHỰT ANH – 521H0491

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENT

I would like to extend my heartfelt appreciation and gratitude to PhD. Lê Văn Vang for his invaluable contributions and guidance in the field of design and analysis of Algorithms. His expertise and dedication have played a significant role in shaping my understanding of this complex subject.

THE REPORT IS COMPLETED AT TON DUC THANG UNIVERSITY

I assure you that this is my own essay product and under the guidance of PhD. Lê Văn Vang. The research contents and results in this topic are honest and have not been published in any previous forms. The figures in the tables serving the analysis, comments and assessments were collected by the author himself from different sources specified in the references.

If any fraud is found, I am fully responsible for the content of my report. Ton Duc Thang University is not related to copyright and copyright infringement caused by me during the implementation process (if any).

Ho Chi Minh city, December, 2023.

Author

(Sign and state full name)

PART CERTIFICATION AND ASSESSMENT OF THE LECTURERS

The certification part of the instructor

Ho Chi Minh City, Date: / /2023.

(sign and state full name)

The evaluation part of the teacher rated

Ho Chi Minh City, Date: / /2023.

(sign and state full name

Contents

1. Introduction 5

1.1. What is TCP/IP in WebSocket 5

1.2. TCP/IP Features in WebSocket 5

1.3. How TCP/IP works in WebSocket 6

1.4. Some common applications of WebSocket 7

2. Pros and Cons of WebSocket 9

2.1. Pros of WebSocket 9

2.2. Cons of WebSocket 9

2.3. Compare between WebSocket and HTTP 10

3. Security in WebSocket 11

3.1. Security threats in WebSocket implementation 11

3.2. Solutions to mitigation the above threat 12

4. Implementation 13

4.1. Introduction to Node JS and socket.io 13

4.2. Implementation of the WebSocket application 13

1. Introduction

A WebSocket is a computer communications protocol that allows for real-time, two-way communication between a web browser (or other client application) and a web server. This contrasts with the traditional HTTP protocol which is a request-response protocol, meaning the client has to send a request to the server and then wait for a response.

1.1. *What is TCP/IP in WebSocket*

TCP/IP is a reliable data transfer protocol used in WebSocket. It ensures that data is transmitted in the correct order and is not lost or corrupted.

WebSocket uses TCP/IP at the transport layer of the OSI model. This layer is responsible for dividing data into smaller data packets, ensuring reliable data transmission, and controlling data traffic.

1.2. *TCP/IP Features in WebSocket*

Guaranteed Delivery

- TCP ensures all data packets reach their destination in the correct order. This is crucial for real-time applications as out-of-order or missing data packets can lead to corrupted messages or unexpected behavior.
- If a data packet gets lost or corrupted, TCP will automatically resend it until it reaches the server.

Flow control

- TCP prevents data overload by regulating the flow of information between client and server. This ensures efficient data transmission and avoids network congestion. TCP implements flow control through:
    - Window Size: The sender advertises a window size to the receiver, indicating the maximum amount of data it can buffer.
    - Flow Control Packets: The receiver can send "window update" packets to inform the sender if it has more buffer space available.
    - Congestion Signals: When the receiver becomes congested, it can send "congestion window" packets to the sender, requesting a reduction in data transmission rate.

Congestion Control

- TCP dynamically adjusts its transmission rate to avoid network congestion. This ensures smooth data delivery and prevents network performance degradation. TCP utilizes:
    - Slow Start: The sender starts with a slow transmission rate and gradually increases it until it detects congestion signals.
    - Congestion Avoidance: When congestion is detected, the sender reduces its transmission rate and enters congestion avoidance mode.
    - Fast Retransmit and Recovery: If a packet is lost due to congestion, TCP quickly retransmits it without waiting for the full timeout period.

Error Detection and Correction

- TCP includes mechanisms to identify and rectify data corruption during transmission.
    - Checksums: Each packet includes a checksum calculated based on the data content. If they differ, the packet is considered corrupted.
    - Automatic Retransmission: If a corrupted packet is detected, TCP automatically requests the sender to retransmit it.

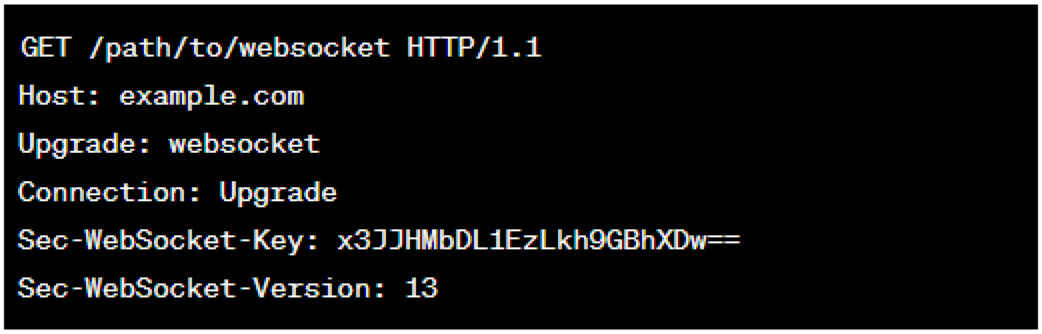
Persistent Connection

- TCP maintains a persistent connection between the client and server, allowing continuous data exchange without needing to re-establish the connection for each message. This is essential for real-time applications that require constant communication.

1.3. How TCP/IP works in WebSocket

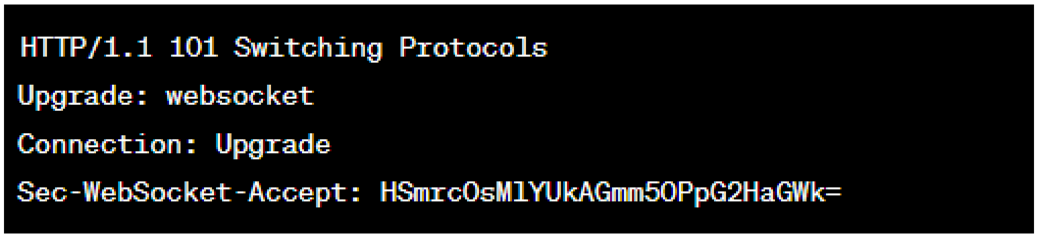
- Connection Establishment: The process begins by establishing a regular TCP connection between the client and server. The TCP/IP protocol ensures a reliable connection between devices on the network.
- WebSocket Handshake: Before using WebSocket, a handshake needs to take place. This handshake is typically a special HTTP request from the client and an HTTP response from the server, which then paves the way for the transition to a WebSocket connection.

- Example about WebSocket Handshake Process:
    - Client Request:



```
GET /path/to/websocket HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
```

    - Server Response:



```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrcOsMlYUkAGmm5OPpG2HaGWk=
```

- A WebSocket connection is established, data can be transmitted between the client and server through that TCP/IP connection. Data can be sent from both the client and server without the need to reopen the connection.

1.4. Some common applications of WebSocket

Chat Applications:

- WebSocket is extensively used in chat applications to provide instant messaging and real-time updates. It allows for bidirectional communication between users, facilitating quick message delivery without the need for constant polling.

Live Streaming:

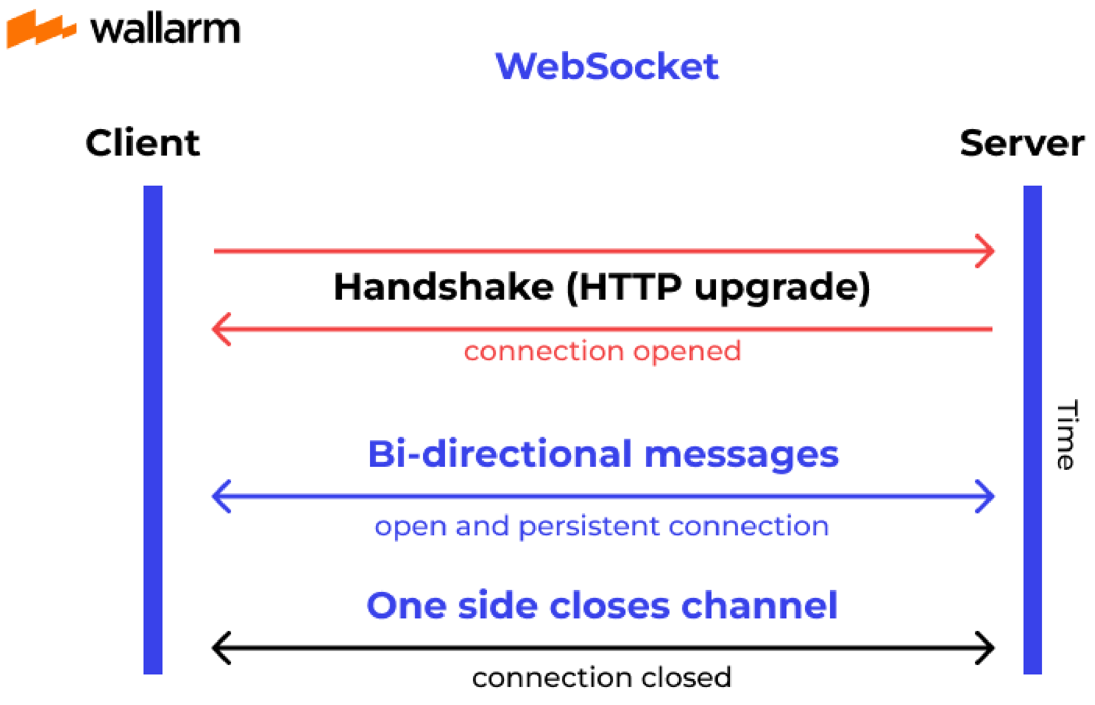
- Platforms that deliver live video or audio streaming, such as online gaming, live sports broadcasting, or video conferencing, often utilize WebSocket for low-latency, real-time communication.

Financial Trading Platforms:

- WebSocket is employed in financial trading applications where real-time updates on stock prices, currency exchange rates, and market data are critical. Traders can receive immediate notifications and make informed decisions based on the latest information.

Dashboard and Monitoring Systems**:**

- WebSocket is used in dashboard applications and monitoring systems where real-time updates on data, metrics, or system status are crucial. This is commonly seen in network monitoring, server performance tracking, and similar applications.



2. Pros and Cons of WebSocket

2.1. Pros of WebSocket

- Real-time Communication: WebSocket enables real-time, bidirectional communication between the client and server. This is particularly useful for applications that require instant updates, such as chat applications, live sports scores, and financial market updates.

- Efficiency: WebSocket uses a single, long-lived connection, eliminating the need to repeatedly open and close connections for each piece of data. This reduces latency and overhead associated with traditional HTTP connections.

- Low Latency: Due to its full-duplex nature and reduced overhead, WebSocket can achieve lower latency compared to other communication methods. This makes it suitable for applications where minimal delay is crucial.

- Reduced Network Traffic: WebSocket reduces the amount of data exchanged between the client and server by eliminating unnecessary HTTP headers in subsequent messages after the initial handshake.

- Scalability: WebSocket is well-suited for scalable applications as it allows a single server to handle a large number of simultaneous connections efficiently.

- Cross-Domain Communication: WebSocket supports cross-domain communication, allowing clients to connect to servers on different domains.

2.2. Cons of WebSocket

- Firewall and Proxy Issues: Some firewall and proxy configurations may not be WebSocket-friendly, potentially causing connection issues. WebSocket connections can be blocked or restricted by some network configurations.

- Complexity: Implementing and managing WebSocket connections can be more complex than traditional HTTP connections, especially for developers unfamiliar with real-time communication concepts.

- Lack of Universal Support: Although widely supported, not all networks, browsers, or proxy servers may fully support WebSocket. In some cases, fallback mechanisms or alternative solutions may be required.

- Security Concerns: WebSocket introduces new security considerations, such as the need to secure data during transmission and protect against potential vulnerabilities. Developers need to be aware of and address these concerns to ensure a secure WebSocket implementation.

- Stateful Connection: WebSocket maintains a stateful connection, which may pose challenges for load balancing and failover strategies in certain deployment scenarios. Ensuring session persistence or employing additional techniques may be necessary.

2.3. Compare between WebSocket and HTTP

Capabilities	WebSocket	HTTP
Connection	Establishes a single, persistent connection that remains open until closed by either client or server. This reduces overhead and improves efficiency for real-time communication.	Creates a new connection for each request and response, leading to overhead and latency
Data Transfer	Supports binary data transfer, allowing for efficient transmission of large data sets like images, audio, and video.	Data is sent in text format, limited to text messages or encoded representations of data
Latency	Real-time data exchange is inherent, reducing latency and enabling immediate updates on the client.	Polling or long-lived connections are needed for real-time updates, leading to higher latency.
Usage Scenarios	Well-suited for real-time applications such as chat applications, online gaming, financial trading platforms	Suited for traditional web applications
Stateless vs. Stateful	Maintains a stateful connection, allowing continuous	Is stateless, meaning each request is

	communication between the client and server without the need for repeated authentication	independent, and the server does not store any information about the client's state between requests
Security	Can be secured using WebSocket Secure (WSS), which is WebSocket over TLS/SSL, ensuring encrypted communication.	Can also be secured using HTTPS, providing a secure channel for data transmission.

3. Security in WebSocket

3.1. Security threats in WebSocket implementation

Cross-Site Scripting (XSS):

- XSS attacks occur when an attacker injects malicious scripts into a web application, which are then executed in the context of other users' browsers. In the case of WebSocket, if input data is not properly validated and sanitized on the server side, an attacker could inject malicious scripts into WebSocket messages.
  - Stored XSS: The malicious script is permanently stored on the target server and served to users when they access a particular page.
  - Reflected XSS: The injected script is part of the URL, and the server reflects it back to the user in the response.

Denial of Service (DoS) Attacks**:**

- Denial of Service attacks aim to disrupt the availability of a service by overwhelming it with a flood of requests. WebSocket connections can be targeted in DoS attacks, potentially leading to server resource exhaustion and unresponsiveness.
  - Volumetric Attacks: Flood the target with a massive volume of traffic to saturate its bandwidth.
  - Protocol Attacks: Exploit vulnerabilities in network protocols, leading to resource exhaustion.
  - Application Layer Attacks: Target specific applications or services to overwhelm server resources.

Man-in-the-Middle (MitM) Attacks:

- MitM attacks involve an unauthorized entity intercepting and potentially modifying the communication between a client and server. If WebSocket communication is not secured using encryption (e.g., TLS/SSL), attackers could eavesdrop on or manipulate the data being transmitted.
  - Packet Sniffing: The attacker intercepts and monitors unencrypted data packets flowing between the client and server.
  - Session Hijacking: The attacker steals session tokens or cookies to impersonate a user.
  - DNS Spoofing: The attacker provides false DNS responses to redirect users to malicious sites.

3.2. Solutions to mitigation the above threat

Cross-Site Scripting (XSS)**:**

- Input validation: Sanitize and validate all user input before sending it over the WebSocket connection. This includes escaping special characters and checking for malicious payloads.
- Output encoding: Encode all server-generated data before sending it to the client. This prevents XSS attacks that exploit vulnerabilities in the client's browser.
- Content Security Policy (CSP): Implement CSP to restrict the execution of scripts on the client-side, limiting the potential for XSS attacks.
- Use secure libraries and frameworks: Choose reliable libraries and frameworks for handling WebSocket communication that implement proper security measures.

Denial of Service (DoS) Attacks**:**

- Rate limiting: Implement rate limiting to restrict the number of messages a client can send within a specific timeframe. This can help prevent resource exhaustion attacks.
- Resource monitoring: Monitor server resources like CPU, memory, and network bandwidth to detect and mitigate DoS attacks before they overwhelm the system.

- Authentication and authorization: Implement authentication and authorization mechanisms to restrict access to the WebSocket server only for authorized users.

Man-in-the-Middle (MitM) Attacks:

- HTTPS with TLS: Use HTTPS with Transport Layer Security (TLS) to encrypt the WebSocket communication channel. This ensures data confidentiality and integrity, preventing MitM attacks from eavesdropping or tampering with messages.
- Certificate validation: Validate server certificates to ensure you're communicating with the intended server and not a malicious imposter.
- Use trusted connections: Establish connections only with trusted servers or use additional verification mechanisms to avoid connecting to compromised servers.
- Use secure web hosting providers: Choose web hosting providers that implement robust security measures and network infrastructure to minimize vulnerabilities.

4. Implementation

4.1. Introduction to Node JS and socket.io

- Node JS is an environment for running JavaScript in native machine different from it originally designed as a programming language just for running on browser using (V8 engine on Google Chrome for example)
- Socket.io is an external library for building low-latency, bidirectional and event-based communication between a client and server. The library provides developers with a set of interfaces for developing WebSocket which we have implemented in this documentation.

4.2. Implementation of the WebSocket application

The use case of this demonstration of this WebSocket application is user will talk to strangers via an anonymous chat box, but before accessing the chat box user is required to enter username only. The TechStack of this is build using Node Js for the server side and React Js for the client side, Socket.io library provide APIs both for the client side and server side for communicating to each other.

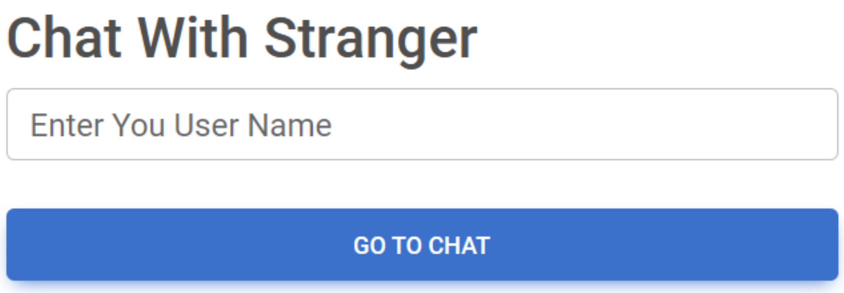


Figure 1 : Login Interface

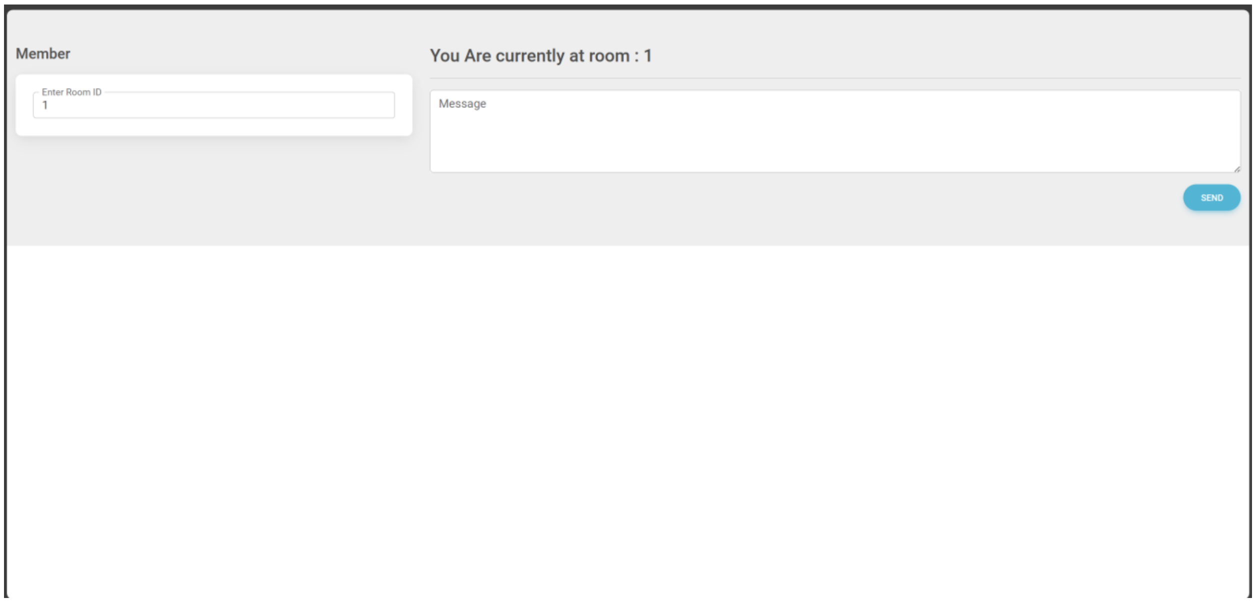
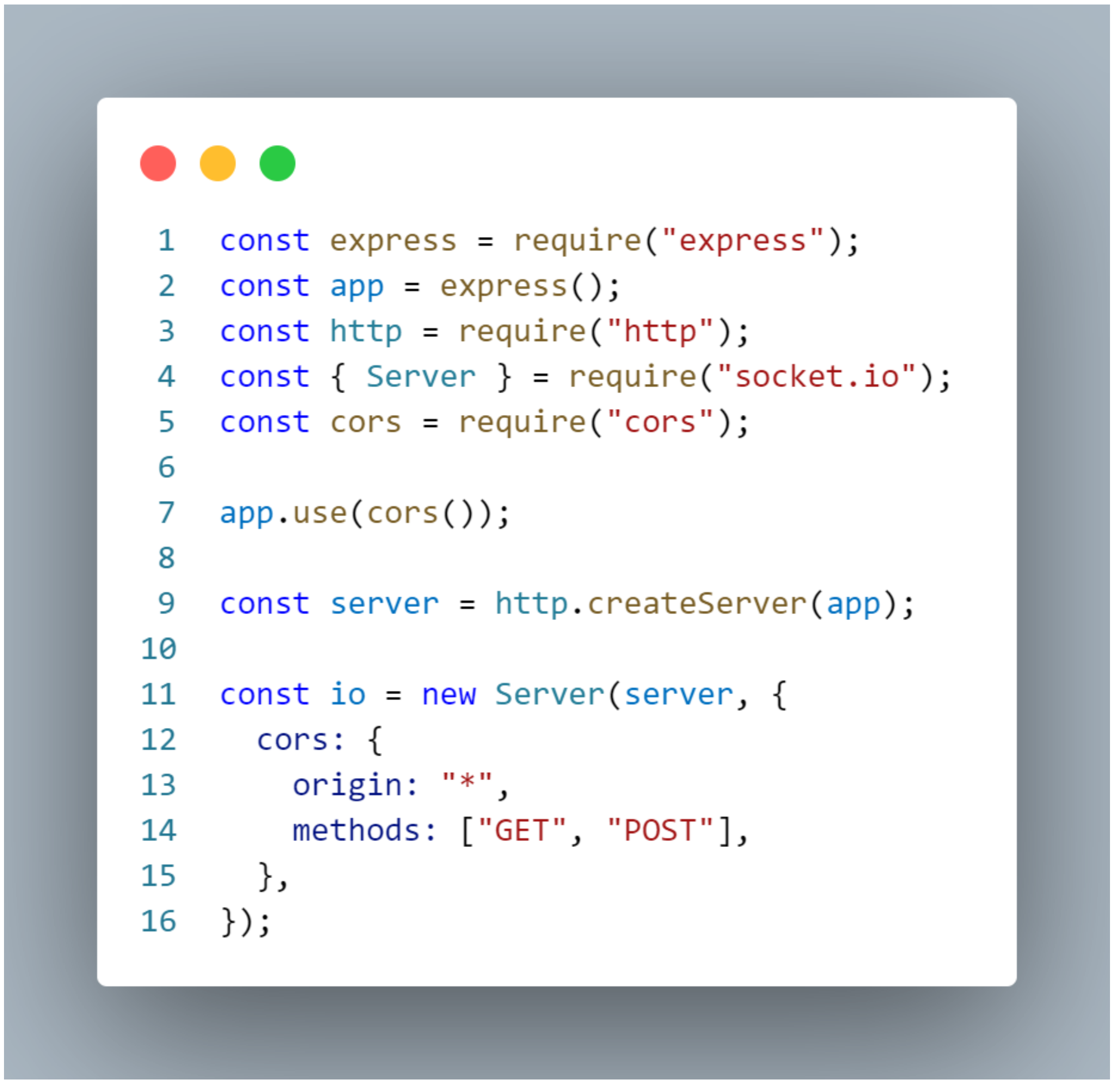


Figure 2 : Chat Box interface

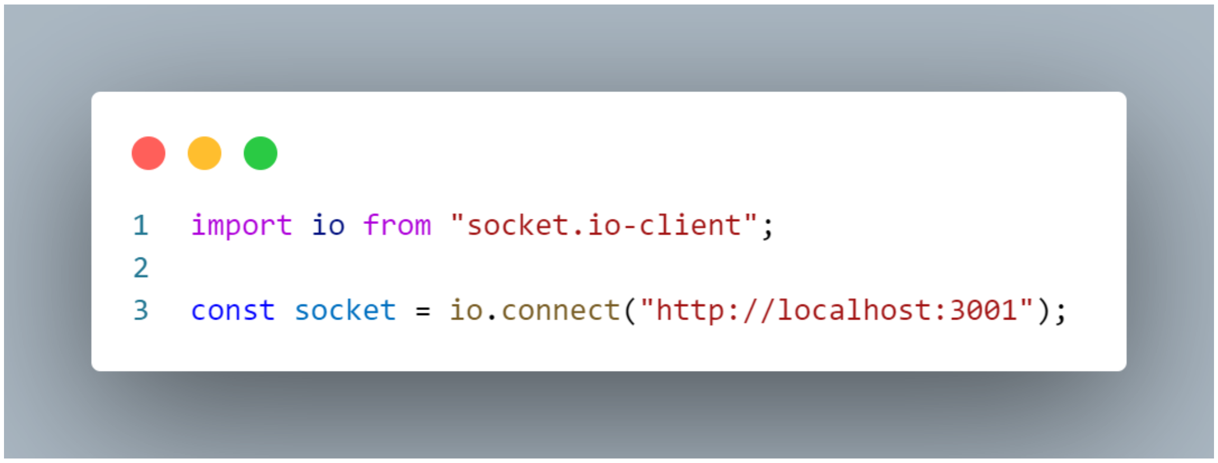
Socket.io initialization in node server



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

app.use(cors());

const server = http.createServer(app);

const io = new Server(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"],
  },
});
```

Figure 3 : Initialization of WebSocket Server

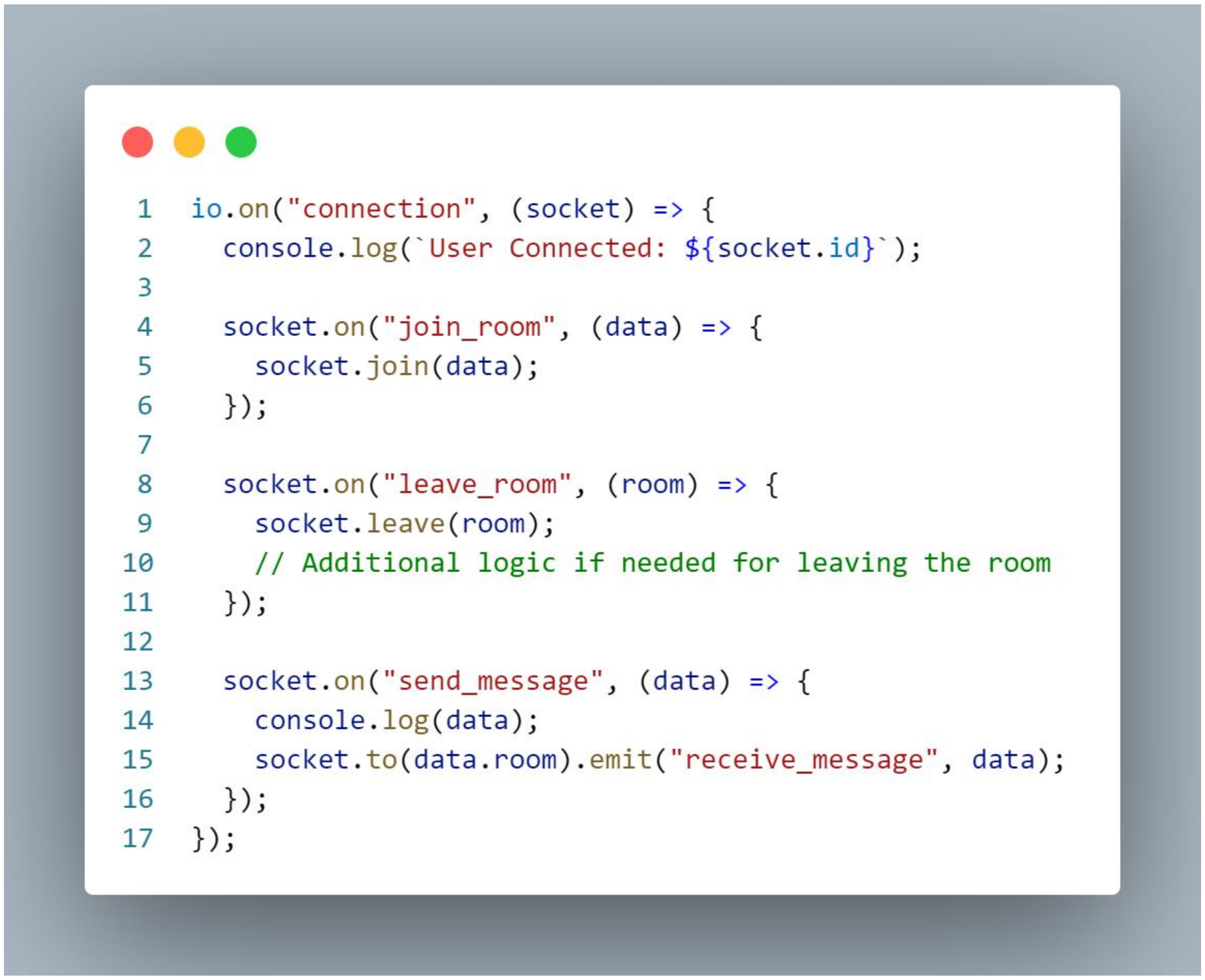
Configure the event listener for the Server for listening for any connection from the client side specifically in case is the react js client server which is connecting.



```javascript
import io from "socket.io-client";

const socket = io.connect("http://localhost:3001");
```

Figure 4 : Socket Client Initialization

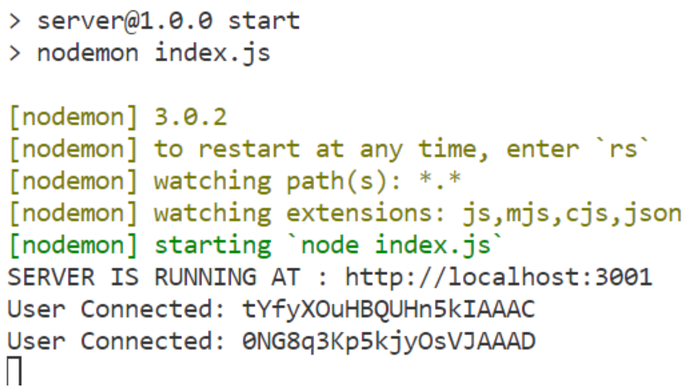
This is the WebSocket initialization in the client side for connecting the node js server side the server side is running at port 3001 in localhost.



```javascript
io.on("connection", (socket) => {
  console.log(`User Connected: ${socket.id}`);

  socket.on("join_room", (data) => {
    socket.join(data);
  });

  socket.on("leave_room", (room) => {
    socket.leave(room);
    // Additional logic if needed for leaving the room
  });

  socket.on("send_message", (data) => {
    console.log(data);
    socket.to(data.room).emit("receive_message", data);
  });
});
```

Figure 5 : Server Side Event Configuration For Listening The Client Side

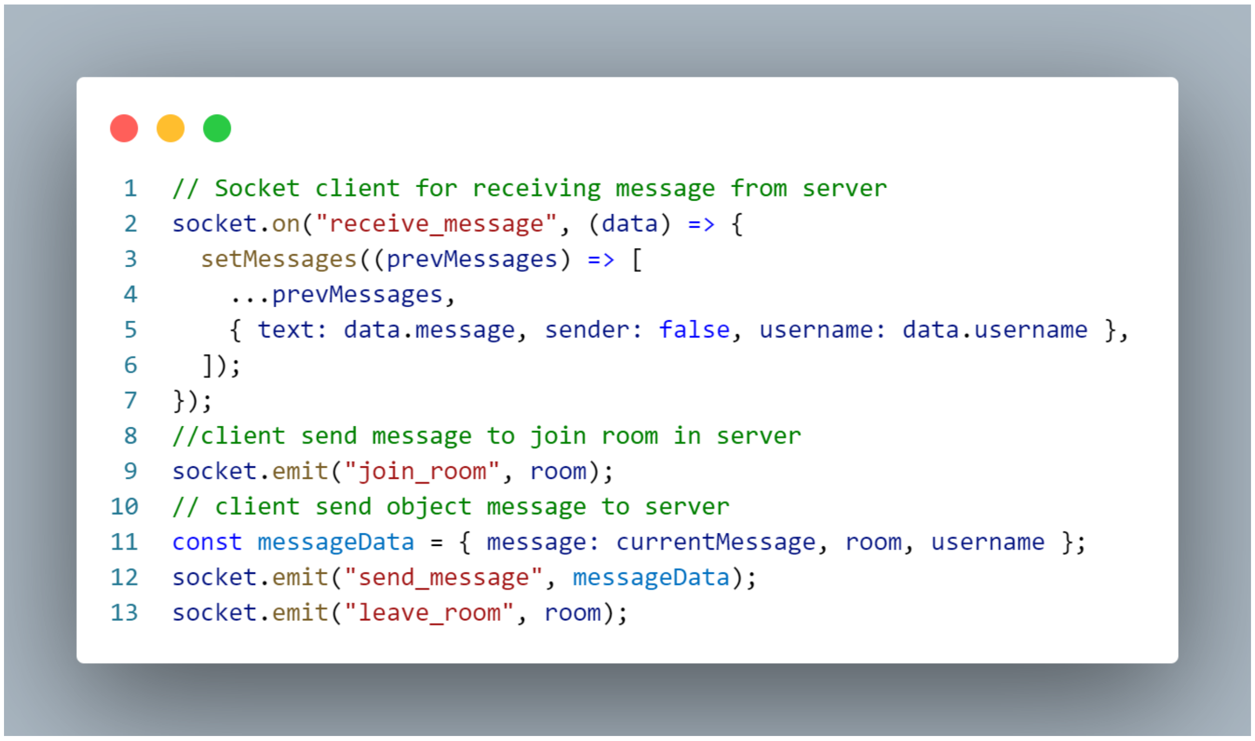
When a user is connected to that server from the "connection" event a socket instance is initialized for example 2 users connected to that WebSocket server will have 2 sockets instances



```
> server@1.0.0 start
> nodemon index.js

[nodemon] 3.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
SERVER IS RUNNING AT : http://localhost:3001
User Connected: tYfyXOuHBQUHn5kIAAAC
User Connected: 0NG8q3Kp5kjyOsVJAAAD
```

Figure 6 : User Connected Example

Based on the client emit events the server will respond to that based on the event first arguments



```
// Socket client for receiving message from server
socket.on("receive_message", (data) => {
  setMessages((prevMessages) => [
    ...prevMessages,
    { text: data.message, sender: false, username: data.username },
  ]);
});
//client send message to join room in server
socket.emit("join_room", room);
// client send object message to server
const messageData = { message: currentMessage, room, username };
socket.emit("send_message", messageData);
socket.emit("leave_room", room);
```

Figure 7 : Sending and Receive Message from Client To Server

REFERENCE