

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA CÔNG NGHỆ THÔNG TIN**



**TRẦN LÊ GIA BẢO – 520H0516  
NGUYỄN HOÀNG PHÚC KHANG – 520H0066**

**WEBSOCKET**

**MIDTERM REPORT**

**WEB PROGRAMMING WITH  
NODEJS**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA CÔNG NGHỆ THÔNG TIN**



**TRẦN LÊ GIA BẢO- 520H0516  
NGUYỄN HOÀNG PHÚC KHANG – 520H0066**

## **WEBSOCKET**

# **MIDTERM REPORT WEB PROGRAMMING WITH NODEJS**

Người hướng dẫn  
**TS. LÊ VĂN VANG**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

## LỜI CẢM ƠN

Before we finish our midterm report, we would want to sincerely thank Mr. Lê Văn Vang for his guidance and support during the writing process. The information gained throughout the course of the study serves as both a basis for the research for the midterm report and a means of preparing for what is ahead. We would want to express our gratitude to all of the instructors and staff at Ton Duc Thang University's Department of Information Technology once more for their hard work in making it possible for me to finish my midterm report.

There are occasionally a lot of mistakes in our report. We're hoping that professors will understand and offer advice on how we might get better at producing reports.

*TP. Hồ Chí Minh, ngày ... tháng ... năm 20..*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Bao*

*Trần Lê Gia Bảo*

*Khang*

*Nguyễn Hoàng Phúc Khang*

## **CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

I hereby declare that I am the principal investigator on this study, working under the scientific direction of Dr. Le Van Vang. The truthful study findings and information on this subject have never previously been released in any format. The author gathered the information in the tables for analysis, comments, and assessment from several sources, and it is made explicit in the reference section.

The Project furthermore incorporates a variety of evaluations, remarks, and statistics from other writers and organizations, all of which are annotated and referenced back to the original source.

**If any fraud is detected, I will take full responsibility for the content of my Project.** Ton Duc Thang University is not involved in copyright violations caused by me during the implementation process (if any).

*TP. Hồ Chí Minh, ngày 15 tháng 12 năm 2023*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Bao*

*Trần Lê Gia Bảo*

*Khang*

*Nguyễn Hoàng Phúc Khang*

# **WEBSOCKET**

## **TÓM TẮT**

The midterm report, written by Nguyễn Hoàng Phúc Khang and Trần Lê Gia Bảo from the Department of Information Technology at Ton Duc Thang University, focuses on WebSocket technology in the context of NodeJS web development.

## **TITLE**

### **ABSTRACT**

This paper explores the vital role that WebSocket technology plays in the current digital era, when real-time online communication is becoming more and more significant. Described as a technology that provides continuous and bidirectional communication between a client and a server, WebSocket represents a major improvement over conventional HTTP communication techniques. The research seeks to give readers a thorough grasp of the technical underpinnings of WebSocket, as well as its useful uses across a range of industries and web development applications in the future.

## MỤC LỤC

<b>DANH MỤC HÌNH VẼ .....</b>	<b>vii</b>
<b>DANH MỤC BẢNG BIỂU .....</b>	<b>viii</b>
<b>CHƯƠNG 1. INTRODUCTION AND OVERVIEW OF THE TOPIC .....</b>	<b>1</b>
1.1 Reason for Choosing the Topic .....	1
1.2 Objectives of the Topic .....	1
1.3 Brief History and Evolution of WebSocket Protocol.....	2
<i>1.3.1 Origin and Development.....</i>	<i>2</i>
<i>1.3.2 Key Features and Improvements .....</i>	<i>4</i>
<b>CHƯƠNG 2. THEORETICAL BASIS OF WEBSOCKET .....</b>	<b>5</b>
2.1 Definition: .....	5
2.2 Fetures: .....	7
2.3 Structure of WebSocket: .....	9
2.4 Advantages and Disadvantages of WebSocket.....	11
2.5 Tools Supporting WebSocket .....	13
2.6 Comparison between WebSocket and HTTP/HTTPS .....	14
<b>CHƯƠNG 3. BUILDING AN ONLINE CHAT APPLICATION .....</b>	<b>15</b>
3.1 Using Socket.io to build chat aplication .....	15
3.2 Main function in chat application .....	16
3.3 Directory organization and interface of the website .....	18
<b>CHƯƠNG 4. CONCLUSION .....</b>	<b>21</b>
4.1 Results achieved.....	21
4.2 Limitations in project .....	21

4.3 Direction of future development .....	21
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>23</b>



## DANH MỤC HÌNH VẼ

Hình 1.1 Example about chat application (Messenger) .....	3
Hình 1.2 The diagram illustrates the WebSocket connection process .....	4
Hình 2.1 Examples of Applications and Websites Using WebSocket.....	8
Hình 2.2 Client-Side WebSocket Example .....	10
Hình 2.3 Server-Side WebSocket Example .....	11
Hình 3.1 Command intall library socket.io .....	15
Hình 3.2 Using socket.io in file main.js or index.js .....	16
Hình 3.3 File io.js.....	16
Hình 3.4 All files in project.....	18
Hình 3.5 Registration page interface.....	19
Hình 3.6 Login page interface.....	19
Hình 3.7 Active account management interface .....	20
Hình 3.8 Chat interface .....	20

## **DANH MỤC BẢNG BIỂU**

Bảng 2.1 Comparison between WebSocket and HTTP/HTTPS .....	14
--	----

# CHƯƠNG 1. INTRODUCTION AND OVERVIEW OF THE TOPIC

## 1.1 Reason for Choosing the Topic

In today's digital world, the demand for real-time web communication has become increasingly vital. WebSocket technology precisely addresses this need by providing a bidirectional and continuous communication channel between client and server. The development of WebSockets has revolutionized how we interact online, making it a compelling topic of study. Its relevance spans various domains, from interactive web applications to real-time data processing, making it essential for modern web development.

### **Key Reasons for the Choice:**

- **Growing Importance in Real-Time Applications:** As real-time interactions become more critical in web applications, understanding WebSockets is crucial.
- **Technological Advancement:** WebSockets represent a significant step forward in web communication, overcoming many limitations of traditional HTTP communication.
- **Broad Applicability:** Its application in chat systems, gaming, live data feeds, and more, shows its versatility.
- **Innovation and Future Potential:** Studying WebSockets opens up possibilities for future innovations in web technologies.

## 1.2 Objectives of the Topic

The primary objectives of studying and understanding WebSockets are to gain a comprehensive understanding of its technical aspects, practical applications, and future potential.

**Main Objectives:**

**Understanding the Technical Foundations:** To delve into how WebSockets work, including the protocol's mechanics, data transfer methods, and the full-duplex communication capability.

**Exploring Practical Applications:** To examine how WebSockets are used in different real-world scenarios like instant messaging, online gaming, and financial trading platforms.

**Performance and Efficiency Benefits:** To understand the advantages of WebSockets in terms of performance and efficiency compared to traditional web communication methods.

**Future Trends and Innovations:** To explore the future prospects of WebSocket technology and its potential integration with emerging web technologies and trends.

**Security Considerations:** To study the security aspects related to WebSockets, addressing challenges and best practices in secure WebSocket communication.

Through this exploration, the goal is to acquire a thorough understanding of WebSockets and its significant role in advancing real-time communication on the web, equipping oneself with knowledge that is highly relevant in the field of web development and technology.

## **1.3 Brief History and Evolution of WebSocket Protocol**

### ***1.3.1 Origin and Development***

#### **Early 2000s: Recognizing the Need**

- ✓ **Limitations of HTTP:** During the early 2000s, as web applications grew more dynamic, the limitations of HTTP for real-time communication became increasingly apparent. HTTP, designed as a stateless request-response protocol, was inefficient for applications requiring continuous data exchange.

- ✓ **Emerging Requirements:** Applications like online gaming, financial trading platforms, and chat applications needed a more interactive and real-time communication method.



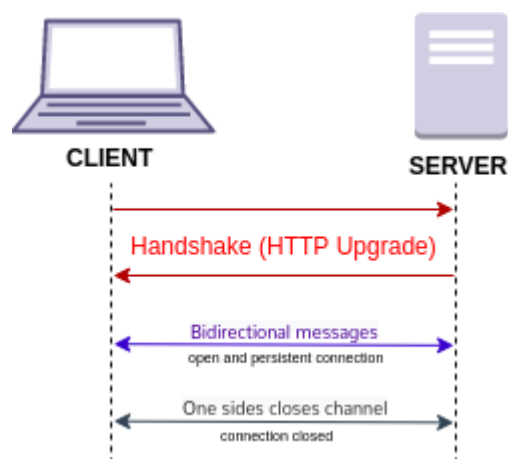
Hình 1.1 Example about chat application (Messenger)

### HTML5 and WebSocket API:

- **Inception of WebSockets:** The concept of WebSockets emerged as a part of the HTML5 initiative to enable real-time, bidirectional communication between web clients and servers. It was first referenced as TCPConnection in the HTML5 specification, as a placeholder for a TCP-based socket API. In June 2008, a series of discussions were led by Michael Carter that resulted in the first version of the protocol known as WebSocket. Before WebSocket, port 80 full-duplex communication was attainable using Comet channels; however, Comet

implementation is nontrivial, and due to the TCP handshake and HTTP header overhead, it is inefficient for small messages.

- **Standardization:** The WebSocket protocol was standardized by the Internet Engineering Task Force (IETF) as RFC 6455 in 2011. The current specification allowing web applications to use this protocol is known as WebSockets. It is a living standard maintained by the WHATWG and a successor to The WebSocket API from the W3C. The WebSocket protocol enables full-duplex interaction between a web browser (or other client application) and a web server with lower overhead than half-duplex alternatives such as HTTP polling, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the client without being first requested by the client, and allowing messages to be passed back and forth while keeping the connection open.



Hình 1.2 The diagram illustrates the WebSocket connection process

### 1.3.2 Key Features and Improvements

**Efficient Communication:** Unlike HTTP, WebSocket provides full-duplex communication, allowing data to be sent simultaneously in both directions.

**Reduced Overhead:** It minimizes the overhead of creating new TCP connections for each request, a limitation inherent in traditional HTTP.

## CHƯƠNG 2. THEORETICAL BASIS OF WEBSOCKET

### 2.1 Definition:

WebSocket is a computer communications protocol, providing a full-duplex communication channel over a single, long-lasting TCP connection. Developed as part of the HTML5 initiative, WebSocket was designed to enable web applications to maintain bidirectional communication with server-side processes, unlike the traditional request-response model of HTTP. Here are more detailed aspects of its definition:

**TCP-Based:** WebSocket relies on the Transmission Control Protocol (TCP), ensuring reliable, ordered, and error-checked delivery of streams of bytes between the client and server.

**Stateful Protocol:** Unlike HTTP, which is stateless, a WebSocket establishes a stateful connection, allowing for continuous data exchange without the need to reestablish connections for each data transfer.

**Full-Duplex Communication:** It allows data to be sent and received simultaneously, unlike HTTP where the communication is unidirectional (client to server or server to client) at any given moment.

**Low Overhead:** After the initial handshake, data frames require very little additional data overhead, making WebSocket efficient for high-frequency, low-latency applications.

**Protocol Upgrade Mechanism:** WebSocket connections are initiated through an HTTP handshake, where the client requests an upgrade to WebSocket. This makes it compatible with the HTTP protocol, leveraging existing infrastructure and ports

Explain the main process of Websocket communication in Hình 1.1 (in Chapter 1), which can be broken down into several key steps:

#### 1. Opening Handshake:

- **Client Request:** The client (usually a web browser) sends an HTTP request to a server with headers indicating a request to upgrade the

connection to WebSocket. This includes headers like **Upgrade: websocket** and **Connection: Upgrade**.

- **Server Response:** The server, if it supports WebSockets, responds with an HTTP 101 status code (Switching Protocols), acknowledging the upgrade.

## 2. Data Transfer:

- Once the handshake is complete, the HTTP connection is upgraded to a WebSocket connection.
- This established WebSocket connection allows for full-duplex communication, meaning the server and client can send messages back and forth independently and concurrently.

## 3. Data Framing:

- Data sent over a WebSocket connection is framed, which means it is packaged into small units with header information.
- These frames can carry text or binary data, and the protocol supports fragmentation, allowing large messages to be split into smaller frames.

## 4. Persisting Connection:

- The WebSocket connection remains open, allowing ongoing communication without the overhead of reestablishing connections.
- Either the client or the server can send 'ping' frames to check if the connection is still alive and 'pong' frames in response.

## 5. Closing Handshake:

- Either the client or server can initiate a connection close.
- This is done by sending a control frame with a specific opcode that denotes a close request.
- Upon receiving this, the other party acknowledges the close, and the TCP connection is terminated.



## 2.2 Fetures:

WebSocket provides an enhanced communication protocol compared to traditional HTTP. Its features are specifically designed for real-time web applications that require persistent and fast two-way communication between the client and the server. Here's a more detailed look at these features:

**Full-Duplex Communication:** WebSocket enables a two-way interactive communication session between the user's browser and a server. With this, a client and server can send messages to each other simultaneously, which is a significant enhancement over HTTP where communication is typically unidirectional for each request-response cycle.

**Low Latency:** WebSocket minimizes the overhead involved in establishing a connection by using a single TCP connection for the duration of the communication session. This significantly reduces latency because there's no need to reestablish the connection and perform a handshake for each data exchange.

**Browser Support and Compatibility:** WebSocket is widely supported in modern web browsers, which allows developers to implement real-time web applications without relying on third-party plugins or browser extensions.

**Fallback Options (Compatibility with HTTP):** WebSocket begins with an HTTP connection, which is then upgraded to a WebSocket connection through the HTTP Upgrade header. This ensures compatibility with the HTTP protocol. For environments where WebSocket is not supported, fallback mechanisms like long polling can be used.

**Low Overhead:** After the initial handshake, data frames sent over WebSocket are small and contain minimal additional data overhead, making WebSocket suitable for high-frequency data exchanges, like those needed in online gaming or stock trading applications.

**Binary and Text Data Transfer:** WebSocket allows the transfer of both text and binary data, enabling a wide range of applications, from chat applications (text) to audio/video streaming (binary).

**Secure Communication (WebSocket Secure - wss://):** WebSocket also supports TLS encryption (wss://), which provides the same level of security as HTTPS, ensuring that data transferred between the client and server is secure and encrypted.

These features are crucial in scenarios where real-time or near-real-time communication is essential:

**Interactive Games:** Multiplayer online games use WebSocket for rapid state synchronization between the server and players.

**Chat Applications:** Real-time messaging apps rely on WebSocket for instant message delivery.

**Collaborative Platforms:** Tools like Google Docs use WebSockets to allow multiple users to edit documents simultaneously.

**Financial Applications:** Stock trading platforms use WebSocket for real-time updates of stock prices and trades without refreshing the browser.

**Internet of Things (IoT):** Devices in IoT ecosystems often communicate with servers using WebSocket to get real-time updates and control messages.



Hình 2.1 Examples of Applications and Websites Using WebSocket

## 2.3 Structure of WebSocket:

### Client-Side API

*WebSocket Object:* In web browsers, the WebSocket object provides the API for creating and managing a WebSocket connection to a server, as well as for sending and receiving data on the connection.

*Constructor:* The API allows developers to create a new WebSocket instance by calling its constructor with the server's URL. This initiates the connection handshake.

*Events:* The WebSocket API provides several event listeners:

- *onopen:* Fired when the connection is successfully opened.
- *onmessage:* Fired when data is received from the server.
- *onerror:* Fired when an error occurs with the connection.
- *onclose:* Fired when the connection is closed.

*Methods:* There are also methods for sending data (*send()*) and closing the connection (*close()*).

### Server-Side Components

*WebSocket Server:* A server capable of understanding and handling WebSocket connections. It listens for incoming WebSocket connections from clients.

*Handling Connections:* The server accepts the connection request, performs the handshake, and then maintains the open connection for ongoing communication until it is closed by either the client or the server.

*Sending and Receiving Data:* The server has the ability to send data to the client and also receives data sent by the client.

### Communication Process

*Handshake:* The WebSocket handshake is an HTTP upgrade request that switches from an HTTP protocol to the WebSocket protocol. It includes a key that the server must return encoded in a response header to prove that it understands the WebSocket protocol.

*Frames*: Once the handshake is complete, data is transmitted in "frames" which may be text or binary data. Each frame consists of:

- An opcode indicating the type of frame (continuation, text, binary, close, ping, or pong).
- A payload length.
- A masking key (client to server frames must be masked).
- The payload data.

### Security Considerations

*WebSocket Secure (WSS)*: Similar to HTTPS, WSS is the WebSocket protocol over TLS/SSL, which encrypts the data transferred between the client and the server.

*Origin Verification*: During the handshake process, servers can verify the Origin header to ensure that the connection is being made from a trusted source.

*Authentication and Authorization*: WebSocket does not inherently handle authentication or authorization, which should be implemented at the application layer, possibly using existing HTTP mechanisms before the upgrade to WebSocket.

In a web browser, you might have JavaScript code like the following to create a WebSocket connection:



```

1 // Create a new WebSocket object and connect to the server
2 var ws = new WebSocket('wss://example.com/socketserver');
3
4 // Set up event handlers
5 ws.onopen = function(event) {
6   console.log('Connection opened:', event);
7   // Send a message to the server once the connection is opened
8   ws.send('Hello, server!');
9 };
10
11 ws.onmessage = function(event) {
12   console.log('Message received:', event.data);
13   // Handle incoming messages
14 };
15
16 ws.onerror = function(event) {
17   console.error('WebSocket error:', event);
18 };
19
20 ws.onclose = function(event) {
21   console.log('Connection closed:', event);
22 };
23
24 // To close the connection
25 ws.close();

```

Hình 2.2 Client-Side WebSocket Example

On the server-side, using Node.js with the ws npm package, the code might look like this:



```
1  const WebSocket = require('ws');
2
3  // Create a WebSocket server
4  const wss = new WebSocket.Server({ port: 8080 });
5
6  wss.on('connection', function connection(ws) {
7    console.log('A new client connected.');
```

Hình 2.3 Server-Side WebSocket Example

In this server-side example, a WebSocket server is created that listens on port 8080. When a client connects, the server prints a message to the console. It also sets up event listeners to handle incoming messages from clients, and to log when the connection is closed or when an error occurs.

## 2.4 Advantages and Disadvantages of WebSocket

### Advantages of WebSocket

- + *Real-Time Communication*: WebSocket provides a full-duplex communication channel, which is ideal for real-time applications such as online games, live broadcasting, and instant messaging.
- + *Low Latency*: Due to its persistent connection, WebSocket ensures that data can be sent and received with minimal delay, which is crucial for

applications where timing is critical, such as financial trading platforms.

- + *Reduced Network Traffic*: Unlike HTTP polling, WebSocket doesn't require clients to repeatedly send requests to the server to check for updates, reducing the amount of network overhead and traffic.
- + *Efficient Processing*: WebSocket's message-based communication model allows both the client and server to process and send messages independently and efficiently, without the need for a request-response paradigm.
- + *Binary Data Support*: WebSocket supports the transfer of binary data, which is essential for applications that need to send non-text content like images, audio, and other types of media.
- + *Fallback Mechanisms*: For environments where WebSocket is not supported, developers can implement fallbacks like long-polling, which allows the application to still function, albeit with higher latency.
- + *Standardized Protocol*: WebSocket is a standard protocol, ensuring consistency and reliability in its implementation across different platforms and devices.

#### Disadvantages of WebSocket

- *Browser Compatibility*: Some older browsers do not support WebSocket, which means that developers have to provide fallback mechanisms for these cases.
- *Complexity in Scaling*: Managing a large number of open WebSocket connections can be more complex than handling stateless HTTP connections, particularly when it comes to scaling and balancing the load across multiple servers.
- *Security Considerations*: WebSocket connections are persistent and can be more exposed to certain security risks, such as Cross-Site

WebSocket Hijacking (CSWSH). Proper security measures, including token-based authentication and encryption, need to be implemented.

- *Infrastructure Support*: Some proxies and firewalls are not configured to support WebSocket, which may require changes to infrastructure or additional handling to ensure WebSocket traffic is allowed and properly routed.
- *Less Mature Than HTTP*: While HTTP has been around for decades and has a vast ecosystem of tools and best practices, WebSocket is relatively newer, and its ecosystem is still growing. This can lead to fewer resources and tools compared to HTTP.
- *No Built-In Message Durability*: WebSocket does not have built-in features for message durability and delivery guarantees. If a connection drops before a message is received, the message may be lost unless additional mechanisms are implemented at the application level.
- *Protocol Restrictions*: WebSocket, being a protocol with a specific purpose, may not be suitable for all applications, especially those that do not require real-time communication.

## 2.5 Tools Supporting WebSocket

Various tools and libraries support the development and implementation of WebSocket, including:

*WebSocket APIs in Modern Browsers*: Most modern browsers have built-in support for the WebSocket API.

*Node.js Libraries*: Libraries like `ws` or `socket.io` provide robust WebSocket capabilities for server-side applications.

*Frameworks and Platforms*: Frameworks like Django (Python), Spring (Java), and others offer WebSocket integration.

*Testing Tools*: Tools like WebSocket King and Smart WebSocket Client aid in testing WebSocket implementations.

## 2.6 Comparison between WebSocket and HTTP/HTTPS

Bảng 2.1 Comparison between WebSocket and HTTP/HTTPS

	Web Socket	HTTP/HTTPS
<b>Similarities</b>	<p>+ Both support the transmission of information between the server and the client.</p> <p>+ WebSocket overcomes the high latency disadvantage of HTTP.</p>	
<b>Differences</b>	<p>- WebSocket has faster transfer speeds and lower latency.</p> <p>- <i>The operating method is different in the communication between the server and the client:</i></p> <p>+ Web Socket is a <b>two-way protocol</b> between server and client. Data can be transmitted from the server to the client or vice versa through established connections.</p> <p>- <i>Application</i></p> <p>+ WebSocket is used in most applications that require real-time communication (real-time) to transmit and receive data on a continuous and immediate connection channel.</p>	<p>- HTTP has a slower transmission speed and higher latency than Web Socket.</p> <p>+ HTTP is a one-way protocol based on TCP, a connection can be made using an HTTP request, after the connection is completed and a response is received, this process will end.</p> <p>+ HTTP is used in services or applications with RESTful design, receiving data in one direction from the server for processing =&gt; then responding =&gt; and closing the connection.</p>



## CHƯƠNG 3. BUILDING AN ONLINE CHAT APPLICATION

### 3.1 Using Socket.io to build chat application

Socket.io is a library designed for real-time web and mobile application development. Its main feature is ease of use, as it can be quickly deployed to transform websites into real-time applications such as trading platforms, online games, social networking sites, and blogs.

The Socket.io library consists of two parts:

- Client-side: includes libraries for web (JavaScript), iOS, Android.
- Server-side: written in JavaScript for the NodeJS server.

Socket.io supports multiple real-time technologies (technologies that allow for immediate communication as events happen, which are essential for real-time applications, web pages that require immediate updates to users) such as WebSocket, Flash Socket, AJAX long-polling, AJAX multipart streaming, Iframe, JSONP polling.

Using Socket.io is similar between client and server, consisting of three main parts:

- Initialize the connection (not necessarily with the server).
- Listen for events.
- Emit events.

Using socket.io

- To use Socket.io with Node.js, you first need to install it using npm as follows:

```
npm install socket.io
```

Hình 3.1 Command intall library socket.io

- After installation, it can be used within your Node.js application by including it as follows:

```
var io = require("socket.io")(server);
```

Hình 3.2 Using socket.io in file main.js or index.js

### 3.2 Main function in chat application

```

1  const express = require("express");
2  const app = express();
3  const server = require("http").Server(app);
4  const io = require("socket.io")(server);
5  const fn = require("../middlewares/function");
6  const Chat = require("../models/Chat");
7
8  const clients = {};
9
10 io.on("connection", (socket) => {
11   let accountId;
12   socket.on("login", async (id) => {
13     accountId = id;
14     // Associate the socket ID with the account ID
15     clients[accountId] = socket.id;
16     await fn.updateStatus(accountId, "Online");
17
18     let account = await fn.getAccountById(accountId);
19     socket.broadcast.emit("status", account);
20   });
21
22   socket.on("send-message", async (sendMessageObject) => {
23     const { receiverId, message } = sendMessageObject;
24
25     try {
26       let chat = new Chat({ senderId: accountId, receiverId, message });
27       await chat.save();
28
29       const socketIdReceiver = clients[receiverId];
30       const receiveMessageObject = { senderId: accountId, message };
31       // Emit the chat message to the receiver
32       io.to(socketIdReceiver).emit("receive-message", receiveMessageObject);
33     } catch (error) {
34       console.log(error);
35     }
36   });
37
38   socket.on("disconnect", async () => {
39     fn.updateStatus(accountId, "Offline");
40
41     let account = await fn.getAccountById(accountId);
42     socket.broadcast.emit("status", account);
43   });
44 });
45
46 module.exports = { app, io, express, server };
47
```

Hình 3.3 File io.js

The main function in the **io.js** file sets up a Socket.IO server integrated with an Express.js application, and it primarily handles real-time communication for a chat feature.

#### 1. Server Setup:

- Express.js is initialized (*const app = express();*).
- An HTTP server is created and passed to the Socket.IO library (*const server = require("http").Server(app);*).
- The Socket.IO server is initialized with this HTTP server (*const io = require("socket.io")(server);*).

#### 2. Middleware and Models:

- Custom middleware functions are imported (*const fn = require("./middlewares/function");*).
- The chat model is imported for database operations (*const Chat = require("./models/Chat");*)

#### 3. Connection Handling:

- A clients object is created to keep track of the connected clients.
- The *io.on("connection", (socket) => {...});* listener is set up to handle new socket connections.

#### 4. Event Listeners within the Connection:

- A login event listener (*socket.on("login", async (id) => {...});*) for when a user logs in, which associates a socket ID with the user's account ID and updates the user's status to "Online".
- A send-message event listener (*socket.on("send-message", async (sendMessageObject) => {...});*) for when a user sends a message. This listener handles the creation of a new chat message, saves it, and then emits the message to the receiver's socket.

- A disconnect event listener (*socket.on("disconnect", async () => {...});*) for when a user disconnects. It updates the user's status to "Offline" and notifies other clients of the status change.

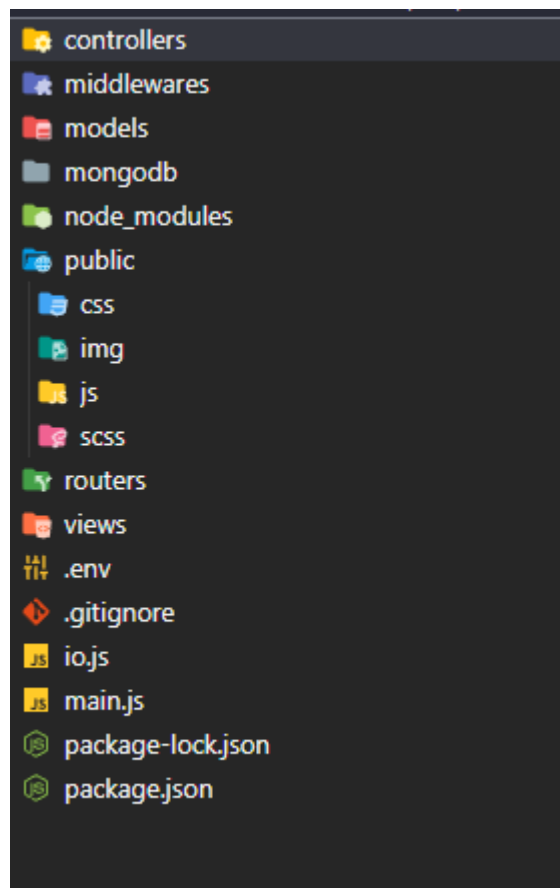
#### 5. Exporting Modules:

- The file exports the Express app, the Socket.IO io instance, the express library, and the server (*module.exports = { app, io, express, server };*).

### 3.3 Directory organization and interface of the website

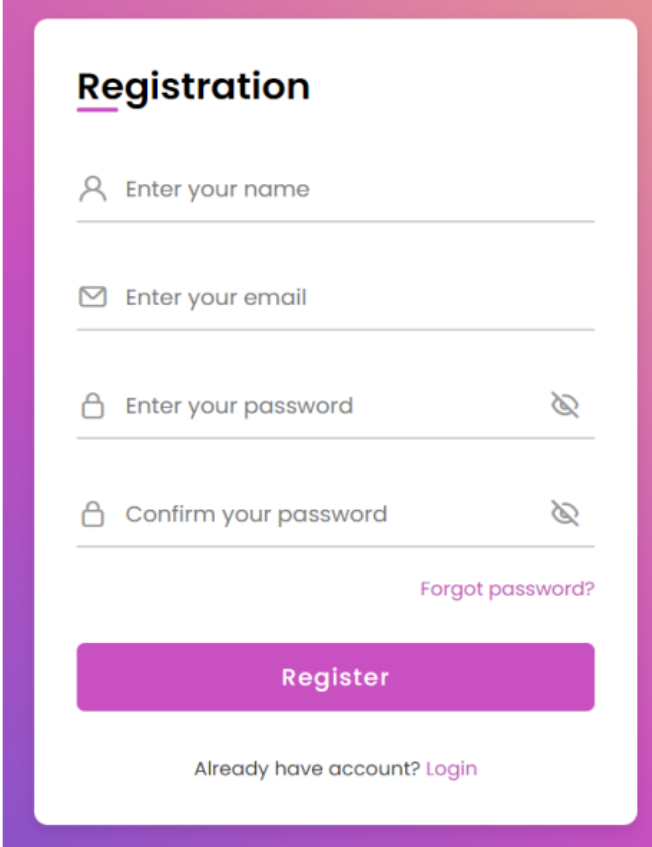
Technologies used during implementation: NodeJS, ExpressJS and socket.io library

Manage Folder



Hình 3.4 All files in project

## User Interface



The registration form is titled "Registration" with a purple underline. It contains four input fields: "Enter your name" with a person icon, "Enter your email" with an envelope icon, "Enter your password" with a lock icon and a toggle icon, and "Confirm your password" with a lock icon and a toggle icon. A "Forgot password?" link is located below the password fields. A large purple "Register" button is centered below the fields. At the bottom, it says "Already have account? [Login](#)".

**Registration**

Enter your name

Enter your email

Enter your password ☐

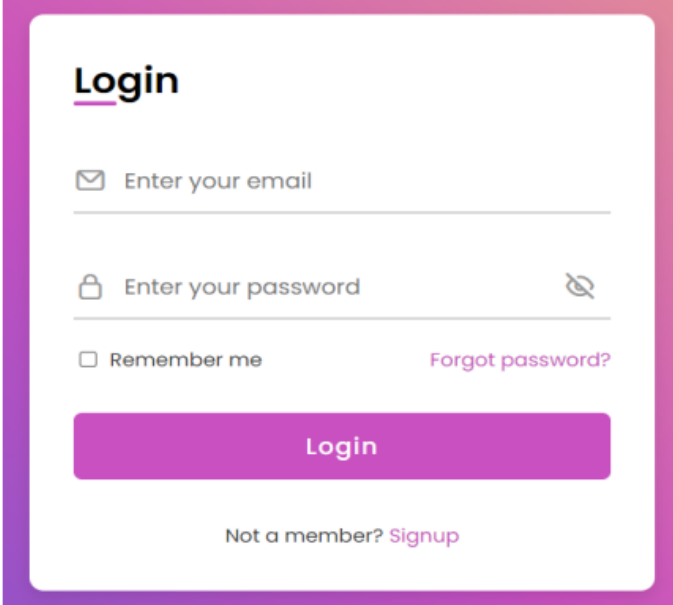
Confirm your password ☐

[Forgot password?](#)

**Register**

Already have account? [Login](#)

Hình 3.5 Registration page interface



The login form is titled "Login" with a purple underline. It contains two input fields: "Enter your email" with an envelope icon and "Enter your password" with a lock icon and a toggle icon. A "Remember me" checkbox is located below the password field. A "Forgot password?" link is located to the right of the "Remember me" checkbox. A large purple "Login" button is centered below the fields. At the bottom, it says "Not a member? [Signup](#)".

**Login**

Enter your email

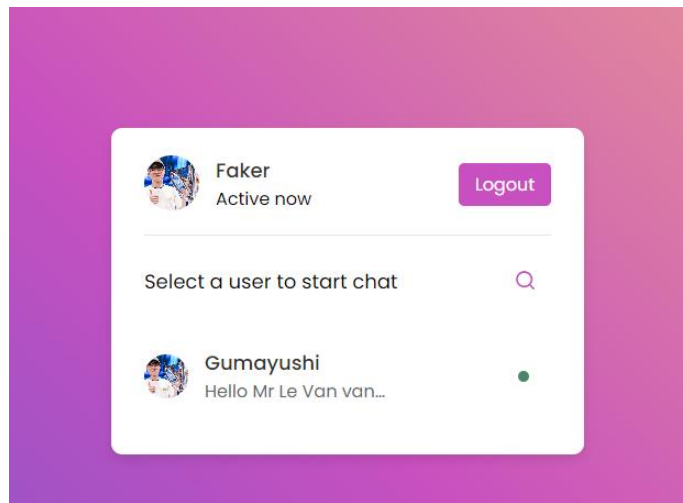
Enter your password ☐

☐ Remember me [Forgot password?](#)

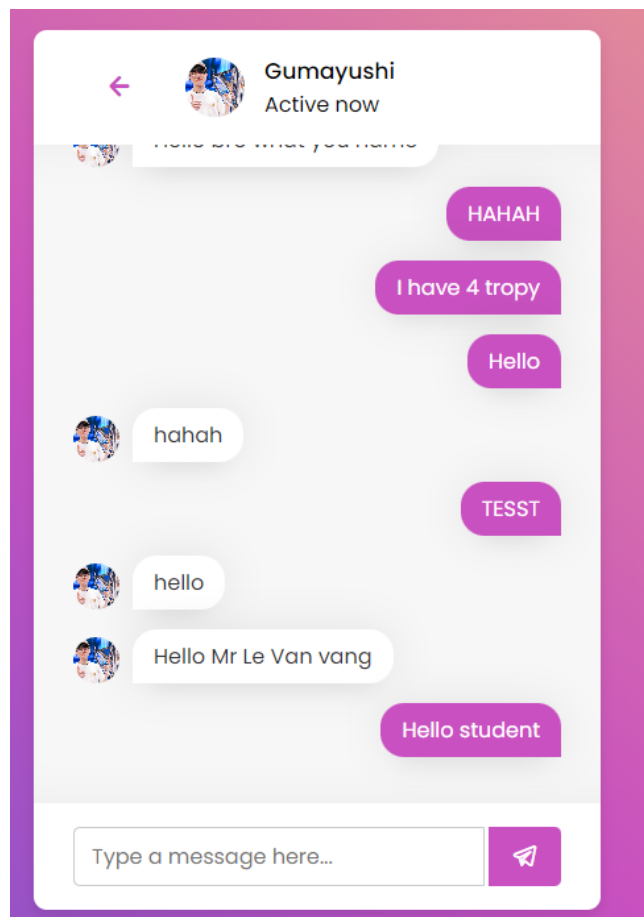
**Login**

Not a member? [Signup](#)

Hình 3.6 Login page interface



Hình 3.7 Active account management interface



Hình 3.8 Chat interface

## CHƯƠNG 4. CONCLUSION

### 4.1 Results achieved

We will gain a better understanding of how WebSocket and Socket.IO work, as well as how they facilitate real-time data communication between the server and the client

We will learn how to integrate Socket.IO with a web application built on Node.js and Express.js, leveraging the power of JavaScript on both the server and client side. Learn how to manage user sessions, recognize who is online, who has left the chat application

Learn how to package and deploy your application to a server, using tools like *npm* for managing dependencies and configuring my application

### 4.2 Limitations in project

However, during the implementation process, there are still a few small errors and unfinished items. Adjusting the functions of an online web chat can be done, such as not changing user avatars, the "Forgot password" function, etc.

In the future, we will try to fully complete these functions and continue to develop more necessary functions to complete the above application.

### 4.3 Direction of future development

**Optimize Performance:** Ensure that your application runs smoothly and reliably. Consider optimizing your source code and use monitoring tools to track and fine-tune your application's performance.

**Security:** Always implement secure user authentication and authorization. Consider implementing security measures such as SSL/TLS for WebSocket connections and address security concerns related to data sent and received via WebSocket.

**Scalability:** As your application grows, consider using scaling solutions like Redis or RabbitMQ to manage and scale WebSocket connections.

**Integrate Additional Features:** Consider adding additional features to your application, such as real-time notifications, chat functionality, or integration with third-party services like OAuth for third-party login.



## TÀI LIỆU THAM KHẢO

Tiếng Việt

<https://viblo.asia/p/websocket-la-gi-Ljy5VxkbZra>

Tiếng Anh

Socket and Websocket documentation: <https://socket.io/docs/v4/>

The difference between web socket and http :

<https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/>