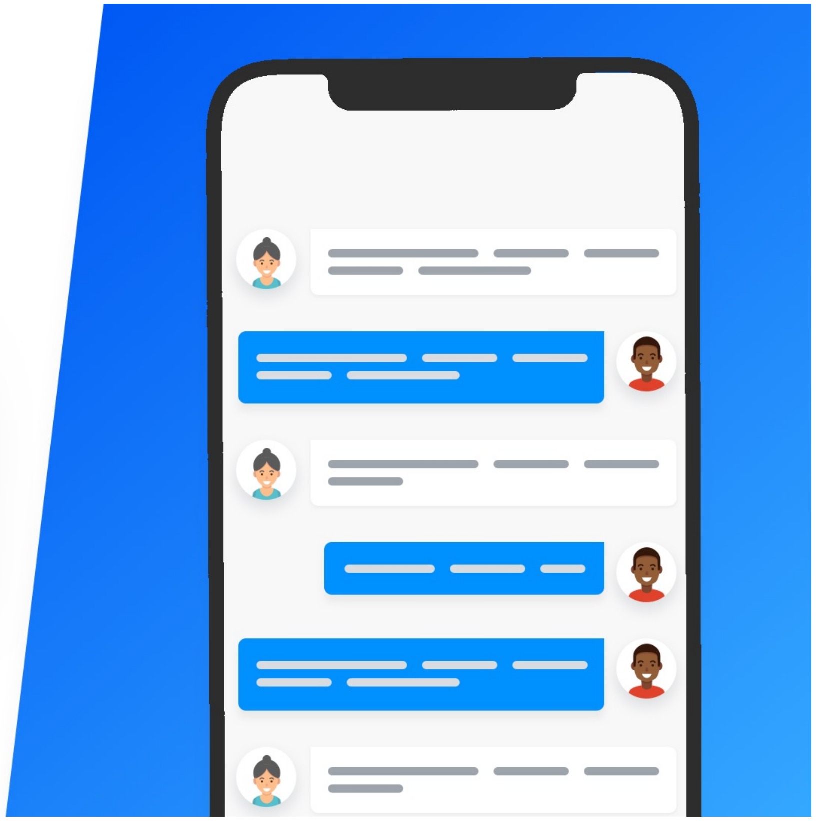
Real Time Chat Application

July 28th 2024

200535561 Dain Shin
200505619 Rakith Wikramanayake
200530585 Seahee Hong



Contents

1. Introduce the Real Time Chat

1. Introduce the Real Time Chat

What is the definition of Real time chat?

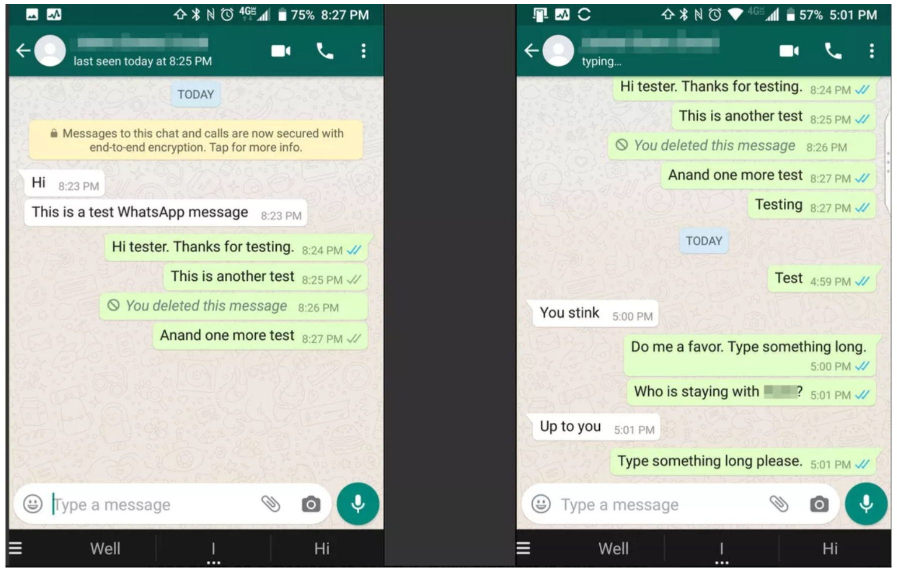
It is an online communication channel that allows you to conduct real-time conversations. It involves the transmission of live text messages from the sender to the recipient.

Real time chat market size

In 2021, the global instant messaging and chat software market was valued at $219.1 million, and it is expected to reach $538.36 million by 2031.

Why is the real time chat market growing?

- Increase in social media ·⟶ Rising need for real-time communication
  e.g., Instagram (messaging functionality), Telegram, WhatsApp
- Increased demand for remote work and virtual collaboration after COVID-19
  e.g., Slack, Teams, chatbots



75% 8:27 PM

[redacted]
last seen today at 8:25 PM

TODAY

Messages to this chat and calls are now secured with end-to-end encryption. Tap for more info.

Hi 8:23 PM

This is a test WhatsApp message 8:23 PM

Hi tester. Thanks for testing. 8:24 PM

This is another test 8:25 PM

You deleted this message 8:26 PM

Anand one more test 8:27 PM

Type a message

Well | I | Hi

57% 5:01 PM

[redacted]
typing...

Hi tester. Thanks for testing. 8:24 PM

This is another test 8:25 PM

You deleted this message 8:26 PM

Anand one more test 8:27 PM

Testing 8:27 PM

TODAY

Test 4:59 PM

You stink 5:00 PM

Do me a favor. Type something long. 5:00 PM

Who is staying with [redacted]? 5:01 PM

Up to you 5:01 PM

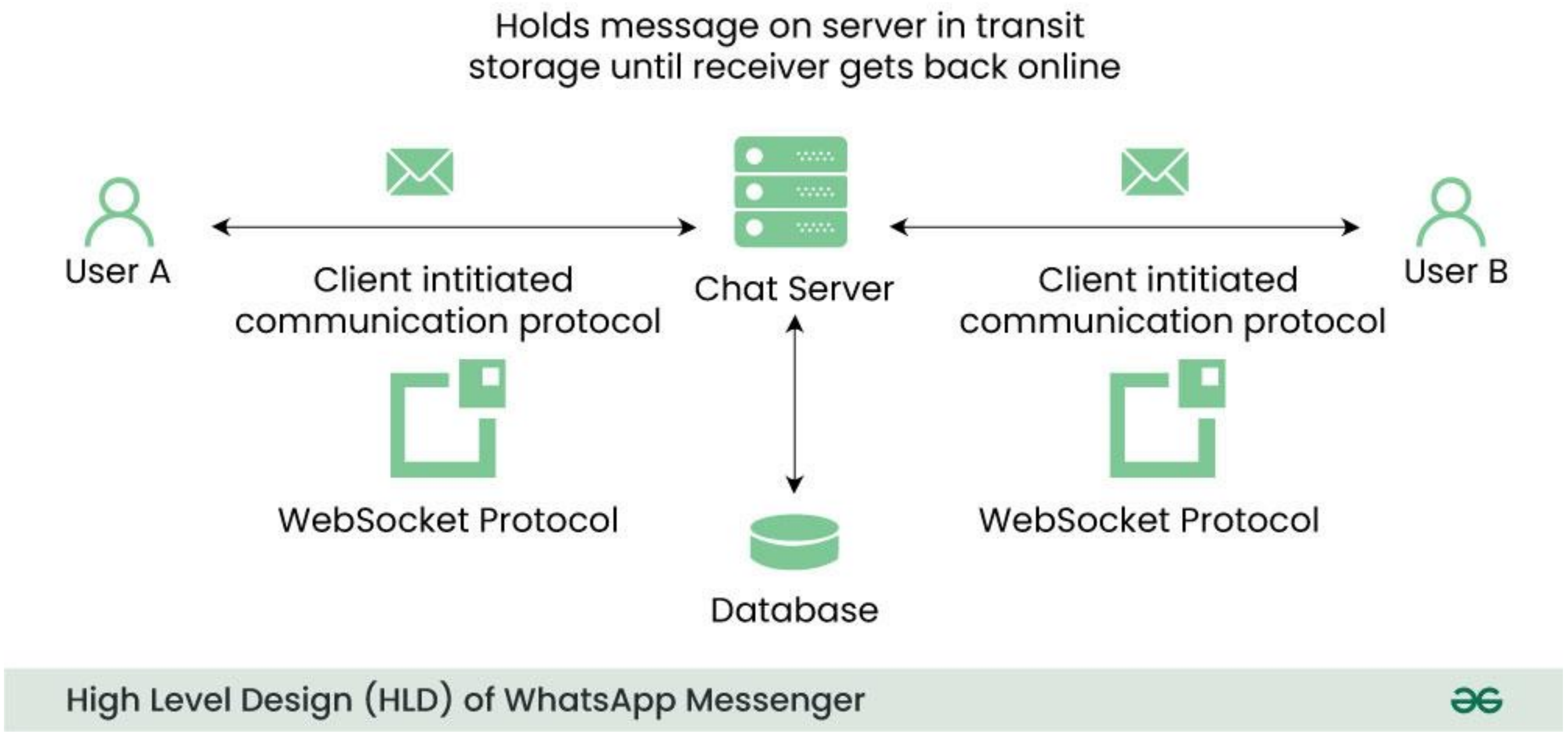
Type something long please. 5:01 PM

Type a message

Well | I | Hi

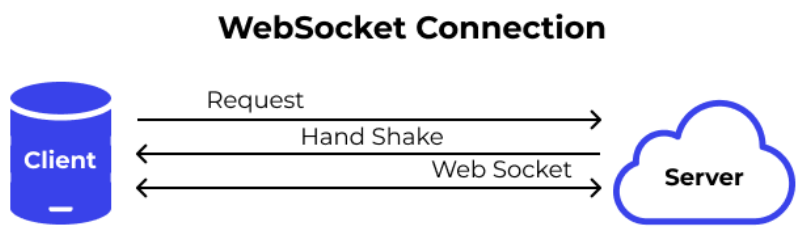
2. How the Real Time Chat works?

2-1. Real time Chat Application process

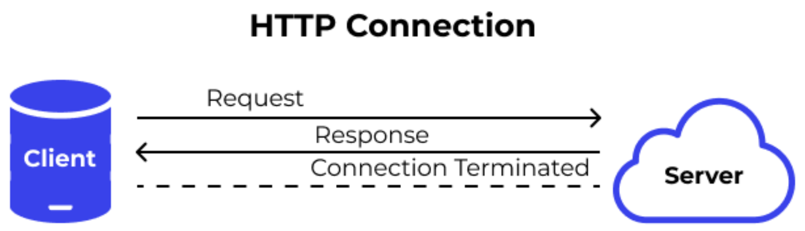

Holds message on server in transit storage until receiver gets back online
User A
Client intitiated communication protocol
Chat Server
Client intitiated communication protocol
User B
WebSocket Protocol
WebSocket Protocol
Database
High Level Design (HLD) of WhatsApp Messenger

2-2. WebSocket

WebSocket vs HTTP



VS

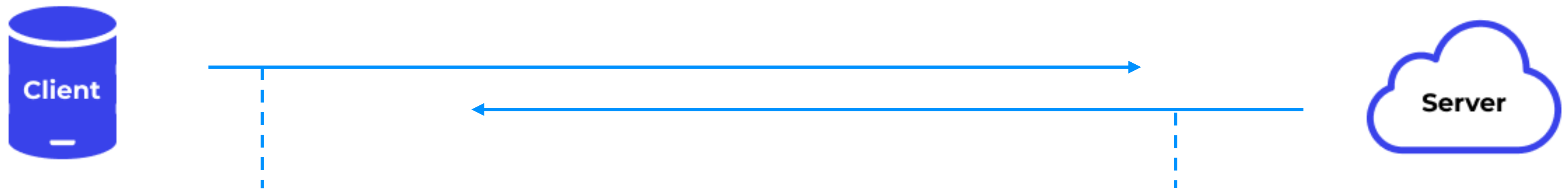


- WebSocket is a framed and bidirectional protocol.
- As WebSocket protocol is capable to support continual data transmission, it's majorly used in real-time application development.
- In WebSocket, communication occurs at both ends, which makes it a faster protocol.
- WebSocket uses a unified TCP connection and needs one party to terminate the connection. Until it happens, the connection remains active.

- HTTP is a unidirectional protocol functioning above the TCP protocol.
- HTTP is stateless and is used for the development of RESTful and SOAP applications.
- In HTTP, the connection is built at one end, making it a bit sluggish than WebSocket.
- HTTP needs to build a distinct connection for separate requests. Once the request is completed, the connection breaks automatically.

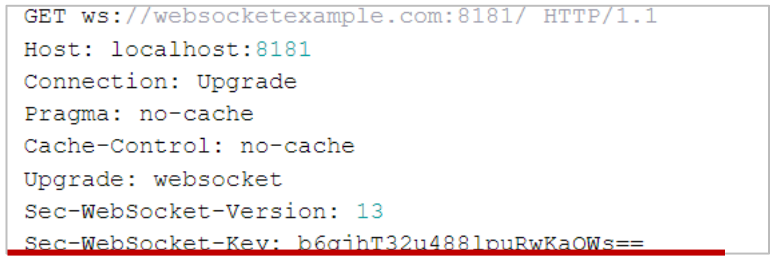
2-2. WebSocket

Hand Shake?



Request

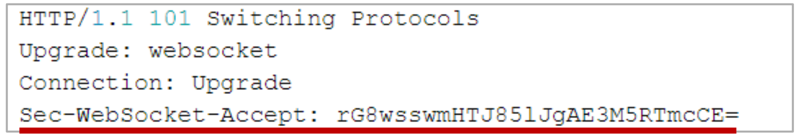
Upgrade header denotes the WebSocket handshake while the **Sec-WebSocket-Key** features Base64-encoded random value. This value is arbitrarily generated during every WebSocket handshake. Besides the above, the key header is also a part of this request.



```
GET ws://websocketexample.com:8181/ HTTP/1.1
Host: localhost:8181
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: b6gjhT32u488lpuRwKaOWs==
```

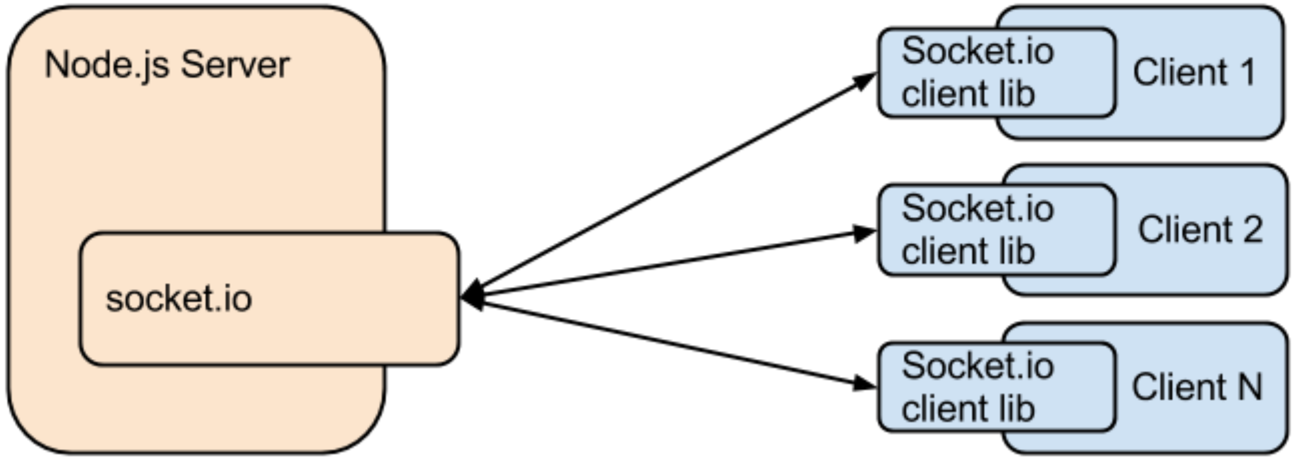
Response

The response header, **Sec-WebSocket-Accept**, features the zest of value submitted in the **Sec-WebSocket-Key** request header. This is connected with a particular protocol specification and is used widely to keep misleading information at bay. In other words, it enhances the API security and stops ill-configured servers from creating blunders in the application development.



```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: rG8wsswmHTJ85lJgAE3M5RTmcCE=
```

2-3. Socket.io



- Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server.
- The Socket.IO connection can be established with different low-level transports
- Depending on the capabilities of the browser or the network, Socket.IO will automatically pick the best available option
  1) HTTP long-polling
  2) WebSocket
  3) WebTransport

2-4. WebSocket vs Socket.io

WebSocket	Socket.io
It is the convention that is set up over the TCP connection.	It is the library to work with WebSocket.
It gives full duplex correspondence on TCP connections.	Provides occasion-based correspondence among program and worker.
Proxy and the load balancer aren't upheld in WebSocket.	An association can be set up within the sight of intermediaries and burden balancers.
It doesn't uphold broadcasting.	It underpins broadcasting.
It doesn't have a fallback option.	It underpins fallback choices.
This technology enables two-way communication that is real-time between the server and the client.	On top of WebSocket, it provides an abstraction layer that makes it easy to create real-time applications.
It needs lower memory space than Socket.io	It requires more memory space.
It is not much complex as Socket.io	It is quite more complex than WebSocket.
It does not provide features as Socket.io	It provides features such as auto-reconnect, rooms, and fallback to long polling.
It is best for simple use cases.	It is best for complex use cases.

2-5. Socket.io Advantages & Disadvantages

Advantages

1) **Connection management**
When a connection is interrupted, Socket.IO automatically attempts to reconnect according to the connection strategy that you decide

2) **Rooms**
Rooms allow you to group multiple clients together in one channel, so you can broadcast a message to all of them with just one call to Socket.IO.

3) **Multiplexing**
Socket.IO's namespacing lets you divide a single connection into multiple logical connections, which can help you to organize and manage separately the communication for different parts of your application

4) **Reduces your development and maintenance overhead**
Socket.IO takes care of a lot of functionality that you would otherwise have to implement yourself if you were using a pure WebSocket implementation

5) **Easy learning curve**
The quality of its documentation and the breadth of its developer community also mean the learning curve should be relatively shallow

Disadvantages

1) **Single region**
Socket.IO can only run in one data center or cloud region. With no fallback, failures at your cloud provider will take your messaging offline. Driving all traffic to a single geographic location also means users further away will experience higher latencies.

2) **Weak message guarantees**
Socket.IO defaults to 'at most once' delivery. In other words, it promises never to deliver your message more than once - but it doesn't rule out the chance of the message not being delivered at all.

3) **Not a drop-in replacement for WebSocket**
Socket.IO does not set out to be a pure WebSocket library. It does mean you can't easily switch to another messaging library. That could cause issues if you have non-JavaScript clients, for example.

4) **Favors the JavaScript ecosystem**
If you're working with something other than NodeJS on your backend, Socket.IO offers secondary implementations for languages such as Java and Rust. However, each of those libraries is playing catch-up with the JavaScript versions and will, most likely, drag behind in terms of features and quality.

3. Introduce Our Application

3-1. Packages

Backend	
Express	A library that makes it easy to set up a server using Node.js.
Nodemon	A tool that detects changes in the source code and reflects them without needing to stop and restart the server.
Mongoose	An ODM (Object Data Modeling) library for MongoDB.
Cors	It allows you to permit requests from specific origins or from all origins.
Dotenv	It manages environment variables more effectively.
Http	It is used to create and manage HTTP servers and clients.
Socket.io	It enables low-latency, bidirectional and event-based communication between a client and a server.

Frontend	
React	React is a declarative, efficient, and flexible JavaScript library for building user interfaces (UI).
Socket.io- client	A client-side library for socket communication.

3-2. Application Structure

Backend Setting

1) Installing NPM packages
2) Setting database
3) Setting Web socket

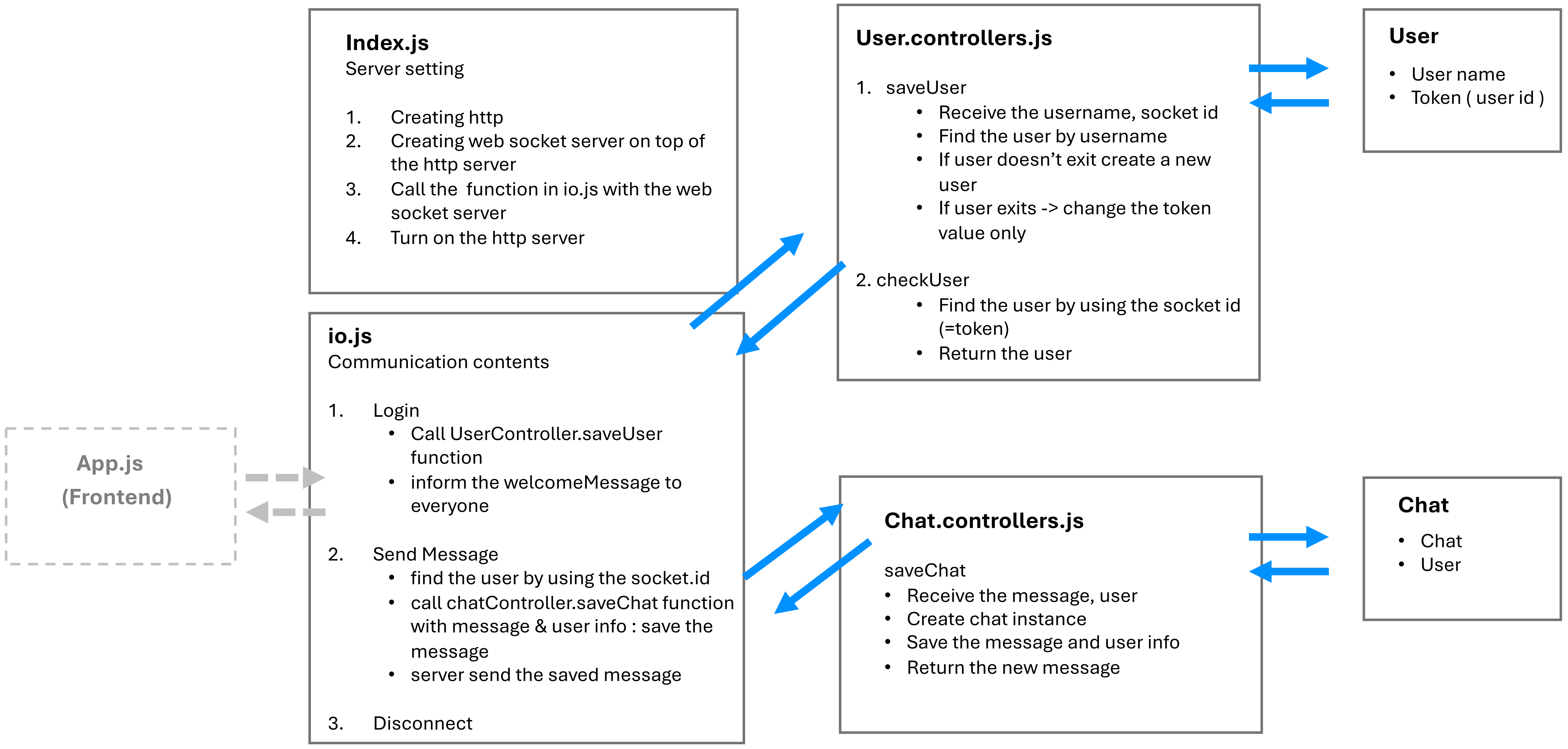
Frontend Setting

1) Installing NPM packages
2) Creating UI (React template)
3) Setting Web socket client

Functionalities

1) User login
2) Sending & Receiving message

3-2. Application Structure - Backend



Index.js
Server setting

1. Creating http
2. Creating web socket server on top of the http server
3. Call the function in io.js with the web socket server
4. Turn on the http server

io.js
Communication contents

1. Login
   - Call UserController.saveUser function
   - inform the welcomeMessage to everyone
2. Send Message
   - find the user by using the socket.id
   - call chatController.saveChat function with message & user info : save the message
   - server send the saved message
3. Disconnect

User.controllers.js

1. saveUser
   - Receive the username, socket id
   - Find the user by username
   - If user doesn't exit create a new user
   - If user exits -> change the token value only

2. checkUser
   - Find the user by using the socket id (=token)
   - Return the user

User

- User name
- Token (user id)

Chat.controllers.js

saveChat
- Receive the message, user
- Create chat instance
- Save the message and user info
- Return the new message

Chat

- Chat
- User

3-2. Application Structure - Frontend

App.js

useState

- React's useState hook allows managing state in function components
- You can set and update component state using useState
- By default, when the state value changes, a re-render occurs

```
const [existingValue, setValue] = useState<type>([defaultValue]);
const [title, setTitle] = useState<string>('');
```

useEffect

- useEffect runs when the component mounts
- It listens for message events from the server and adds messages to messageList.
- It also calls the askUserName function to prompt the user for their name

askUSerName

- Gets the user's name through a prompt
- Sends a login event to the server
- If a response is received from the server, it sets the user
- Renders the MessageContainer component, passing the message, message setting function, and message sending function as props

sendMessage

- Sendingthe message content to the server by emitting a sendMessage event

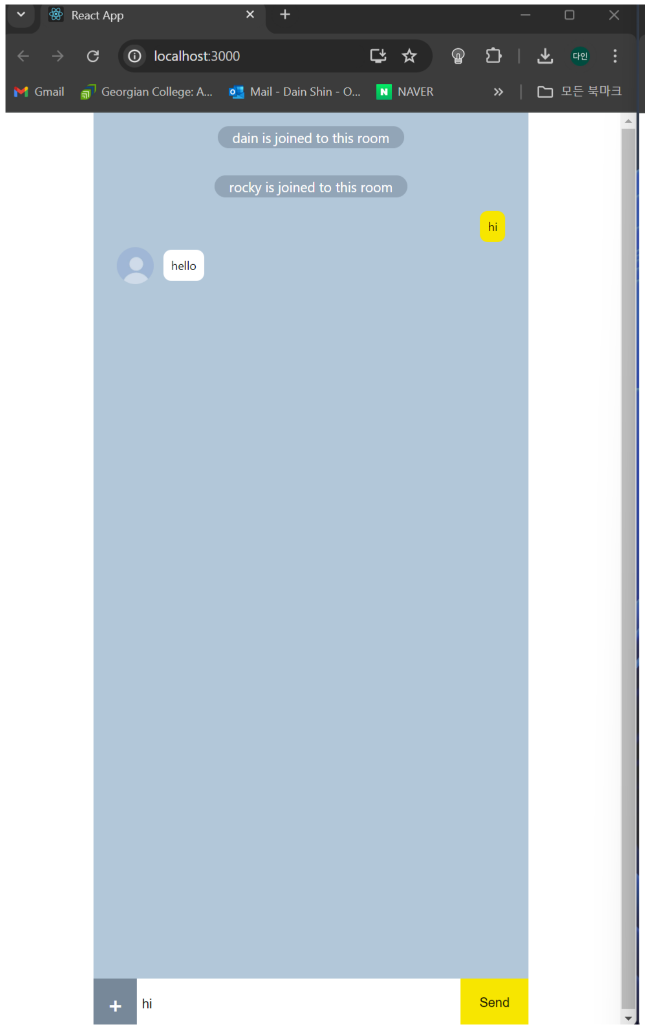
server.js

- Import the io function from the library.
- Establish a socket connection with the server.
- Set the server URL.
- Create the connection using the specified URL.

App. Application Snapshots

Deployment site:

https://main--comp2068-chatapp.netlify.app/



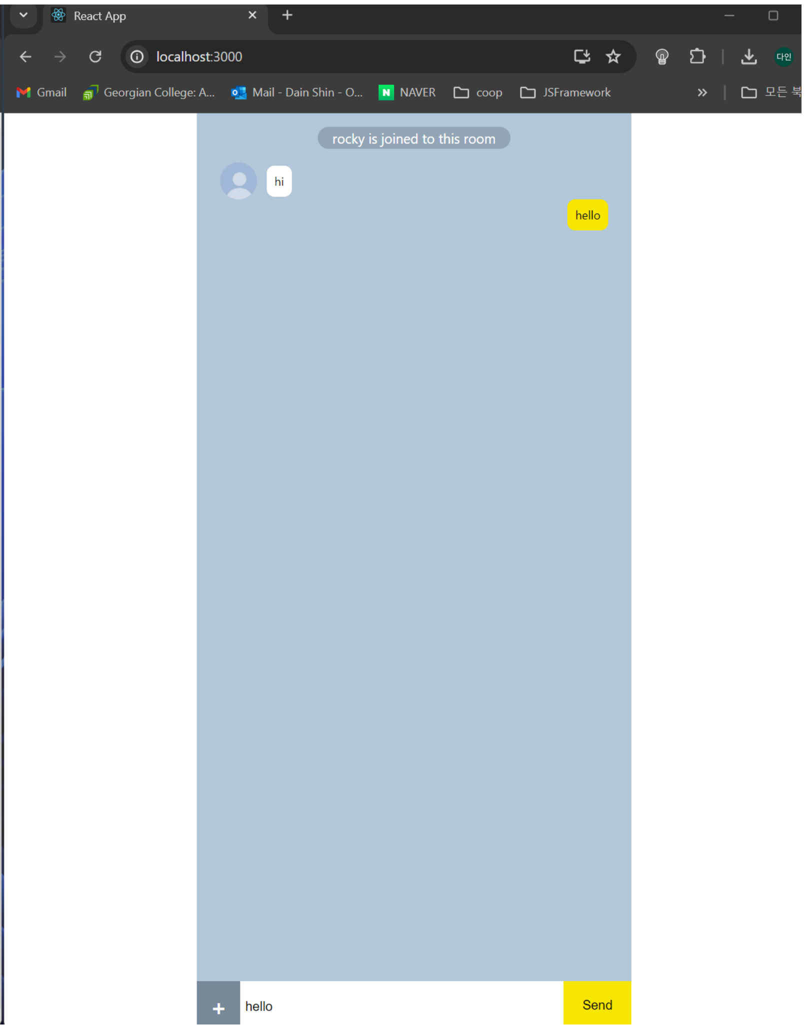
dain is joined to this room

rocky is joined to this room

hi

hello

hi



rocky is joined to this room

hi

hello

hello

3-3. Support Materials

Documents & GitHub

https://socket.io/docs

https://github.com/socketio/socket.io

https://stackoverflow.com/questions/tagged/socket.io

Video about WebSocket

https://www.youtube.com/watch?v=pnj3Jbho5Ck

https://www.youtube.com/watch?v=xTR5OflgwgU

Tutorials

https://www.tutorialspoint.com/socket.io/index.htm

4. Questions?

4. Possible Questions

Q1. Is Socket.IO still needed today?

- WebSockets are supported almost everywhere.
- However, if you use plain WebSockets for your application, you will eventually need to implement most of the features that are already included (and battle-tested) in Socket.IO, like reconnection, acknowledgements or broadcasting.

Q2. Why use WebSockets instead of HTTP polling or long polling?

- WebSockets provide continuous, bidirectional communication between the client and server, allowing for more efficient real-time data transfer.
- This reduces network traffic and latency compared to HTTP polling or long polling.

Q3. Can Socket.IO guarantee the order of messages?

- Socket.IO generally guarantees the order of messages.
- However, if message order is crucial, you can implement additional logic on both the server and client sides to manage and ensure the order of messages.

Q4. How to support video and audio calls in a real-time chat application?

- Socket.IO is primarily used for text-based messaging, while WebRTC is better suited for video and audio calls.
- We can use WebRTC to implement video and audio calls and Socket.IO for signaling.

5. Conclusion

5. Conclusion

Objective
Develop a chat application supporting real-time communication.

Technology Stack
Utilized Socket.io.

Key Features

- Real-Time Messaging :
  Send and receive messages in real-time after login.
- System and User Messages :
  Successful implementation of message differentiation and handling.

Key issue

- **Connection failing problem**
  Implemented stable messaging functionality using CORS.

Future Improvements
Enhance user interface and expand functionality.

Conclusion
Successfully implemented user login and message send/receive features.
With Socket.io's support for rooms and multiplexing, more diverse functionalities can be expected in the future.

Thank you