10. Web Socket

2023학년 2학기 웹응용프로그래밍

권 동 현

CSLAB

Contents

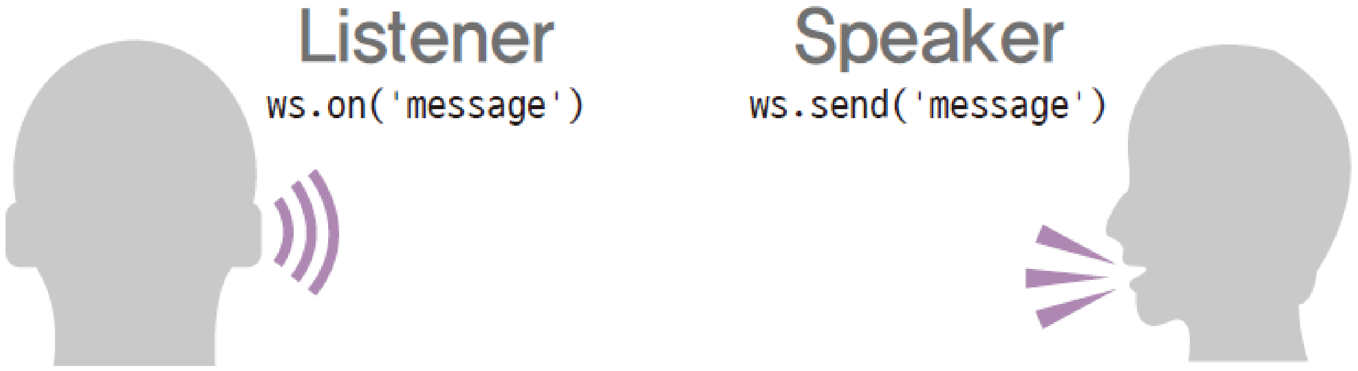
Understanding Web Socket

CSLAB

Understanding Web Socket

- WebSocket: A technology for real-time bidirectional data transfer.
  - Use the ws protocol -> Supported by the browser.
  - Most modern browsers support WebSocket.
  - Node.js can use WebSocket through packages like ws or Socket.IO.
- Before WebSocket, the polling method was used.
  - Since HTTP only allows requests from the client to the server, it periodically sends requests to the server to check for updates.
  - WebSocket requires only one connection, supports port sharing with HTTP, and has excellent performance.

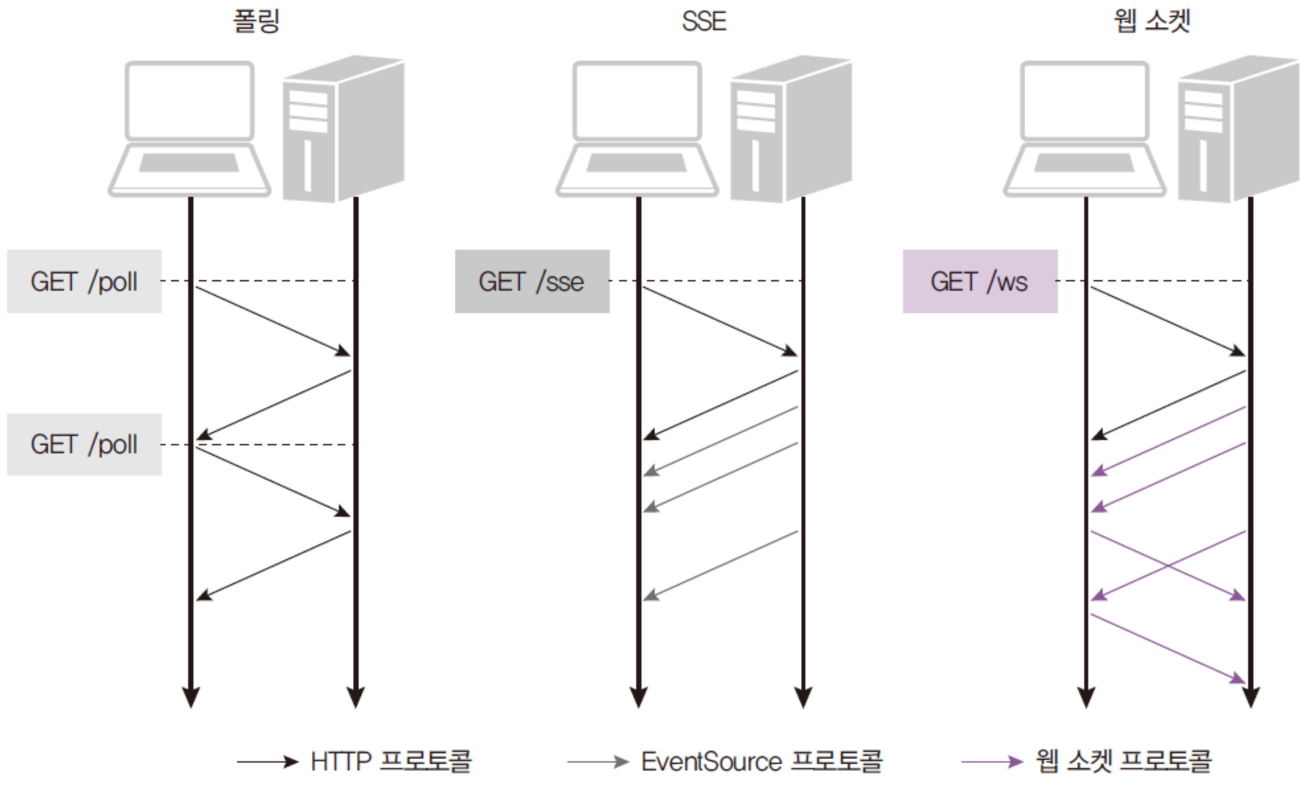
▼ 그림 12-2 웹 소켓 이미지



Server Sent Events

- SSE(Server Sent Events)
  - Using the EventSource object
  - Once connected initially, the server continuously sends data to the client
  - The client cannot send data to the server.

▼ 그림 12-3 폴링 vs. SSE vs. 웹 소켓



Web Socket based ws module

CSLAB

gif-chat Project Creation

- Create a folder named "gif-chat" and write a package.json file in it.

package.json

```json
{
  "name": "gif-chat",
  "version": "0.0.1",
  "description": "GIF 웹소켓 채팅방",
  "main": "app.js",
  "scripts": {
    "start": "nodemon app"
  },
  "author": "Zero Cho",
  "license": "ISC",
  "dependencies": {
    "cookie-parser": "^1.4.5",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "express-session": "^1.17.1",
    "morgan": "^1.10.0",
    "nunjucks": "^3.2.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.3"
  }
}
```

Create basic files

- Install the packages and create the .env, app.js, and routes/index.js files.

콘솔

```
$ npm i
```

.env

```
COOKIE_SECRET=gifchat
```

- source code: https://github.com/ZeroCho/nodejs-book/tree/master/ch12/12.2/gif-chat

Install ws module

- Install with ‘npm i ws’
  - Connect WebSocket to Express
  - ‘socket.js’ will be written later.

콘솔

```
$ npm i ws@8
```

app.js

```
const express = require('express');
const path = require('path');
const morgan = require('morgan');
const cookieParser = require('cookie-parser');
const session = require('express-session');
const nunjucks = require('nunjucks');
const dotenv = require('dotenv');

dotenv.config();
const webSocket = require('./socket');
const indexRouter = require('./routes');
...
const server = app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기 중');
});

webSocket(server);
```

socket.js

- Import ‘ws’ module
  - Connect to the Express server using ‘**new WebSocket.Server({server})**’
  - The connection event is triggered when a connection to the server is established.
  - **req.headers['x-forwarded-for'] || req.connection.remoteAddress** is a well-known method to determine the client's IP address.
  - The message, error, and close events are called when a message arrives, an error occurs, or the server connection is closed, respectively.
  - ws.OPEN signifies that the connection state is open (indicating that the connection is established).
  - Use ws.send to send messages (currently sending every 3 seconds).

socket.js

```
const WebSocket = require('ws');

module.exports = (server) => {
  const wss = new WebSocket.Server({ server });

  wss.on('connection', (ws, req) => { // 웹 소켓 연결 시
    const ip = req.headers['x-forwarded-for'] || req.socket.remoteAddress;
    console.log('새로운 클라이언트 접속', ip);
    ws.on('message', (message) => { // 클라이언트로부터 메시지 수신 시
      console.log(message.toString());
    });
    ws.on('error', (error) => { // 에러 시
      console.error(error);
    });
    ws.on('close', () => { // 연결 종료 시
      console.log('클라이언트 접속 해제', ip);
      clearInterval(ws.interval);
    });

    ws.interval = setInterval(() => { // 3초마다 클라이언트로 메시지 전송
      if (ws.readyState === ws.OPEN) {
        ws.send('서버에서 클라이언트로 메시지를 보냅니다.');
      }
    }, 3000);
  });
};
```

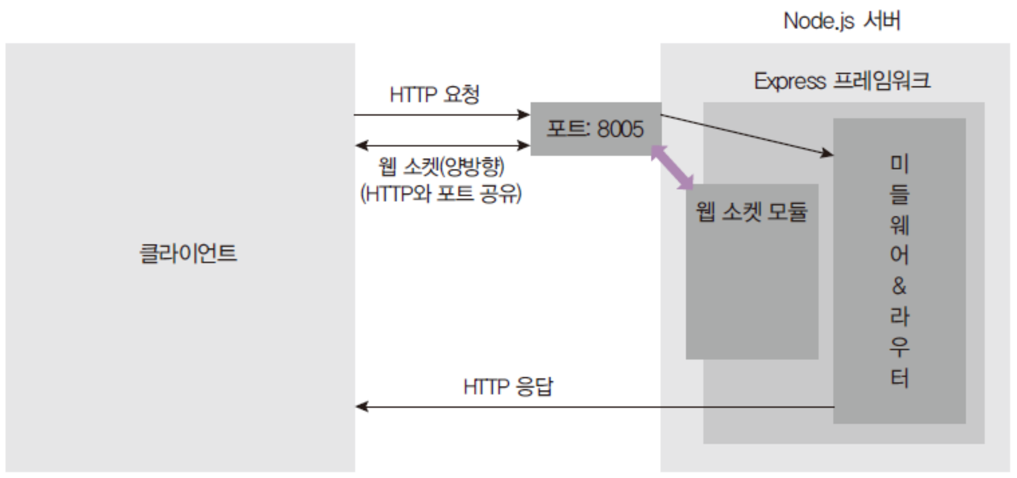
Reply to messages from the frontend

- Write the index.html and script.
  - **new WebSocket** is supported in modern browsers.
  - Input the server address as an argument.
  - The **onopen** event listener is called when a connection to the server is established.
  - The **onmessage** event listener is called when a message is received from the server.
  - The content of the server message is in **event.data**.
  - Messages can be sent to the server using **webSocket.send**.

views/index.html

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>GIF 채팅방</title>
</head>
<body>
<div>F12를 눌러 console 탭과 network 탭을 확인하세요.</div>
<script>
  const webSocket = new WebSocket("ws://localhost:8005");
  webSocket.onopen = function () {
    console.log('서버와 웹 소켓 연결 성공!');
  };
  webSocket.onmessage = function (event) {
   console.log(event.data);
   webSocket.send('클라이언트에서 서버로 답장을 보냅니다');
  };
</script>
</body>
</html>
```

그림 12-4 웹 소켓 구조도



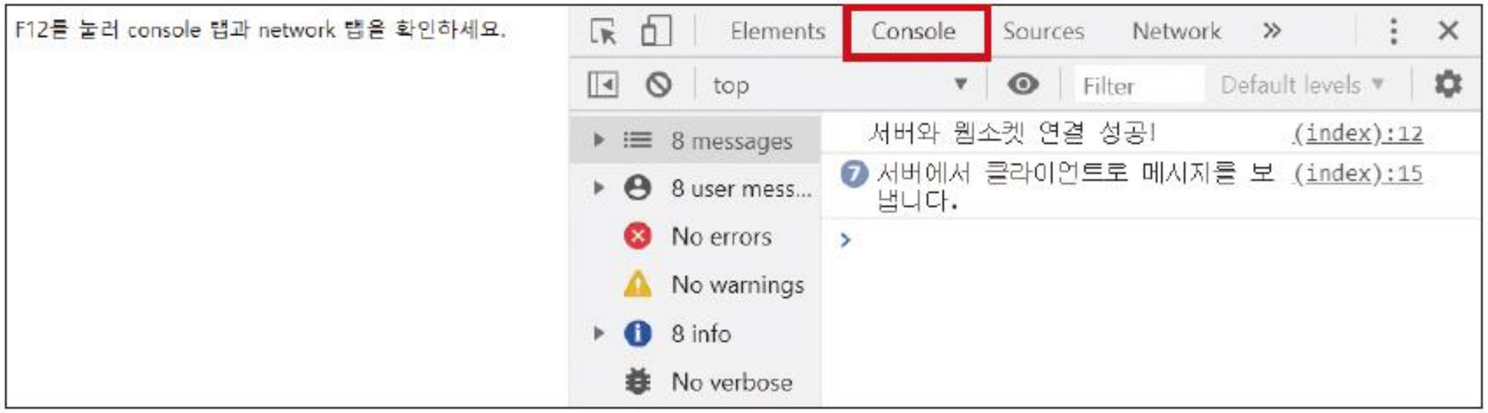
Run server

- Connect to http://localhost:8005
  - F12 console tab
  - Messages will be logged to the Node console and the browser console every 3 seconds from the moment of connection.

노드 콘솔

```
8005번 포트에서 대기 중
새로운 클라이언트 접속 ::1
클라이언트에서 서버로 메시지를 보냅니다
클라이언트에서 서버로 메시지를 보냅니다
클라이언트에서 서버로 메시지를 보냅니다
...
```

그림 12-5 Console 탭 화면



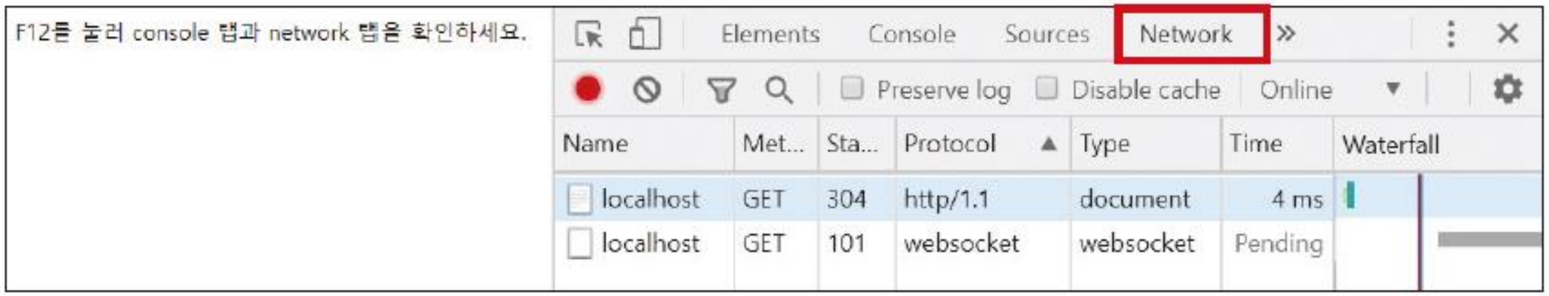
F12를 눌러 console 탭과 network 탭을 확인하세요.

```
서버와 웹소켓 연결 성공!                    (index):12
7 서버에서 클라이언트로 메시지를 보     (index):15
  냅니다.
```

Check the network request

- Open the Network tab in the developer tools.

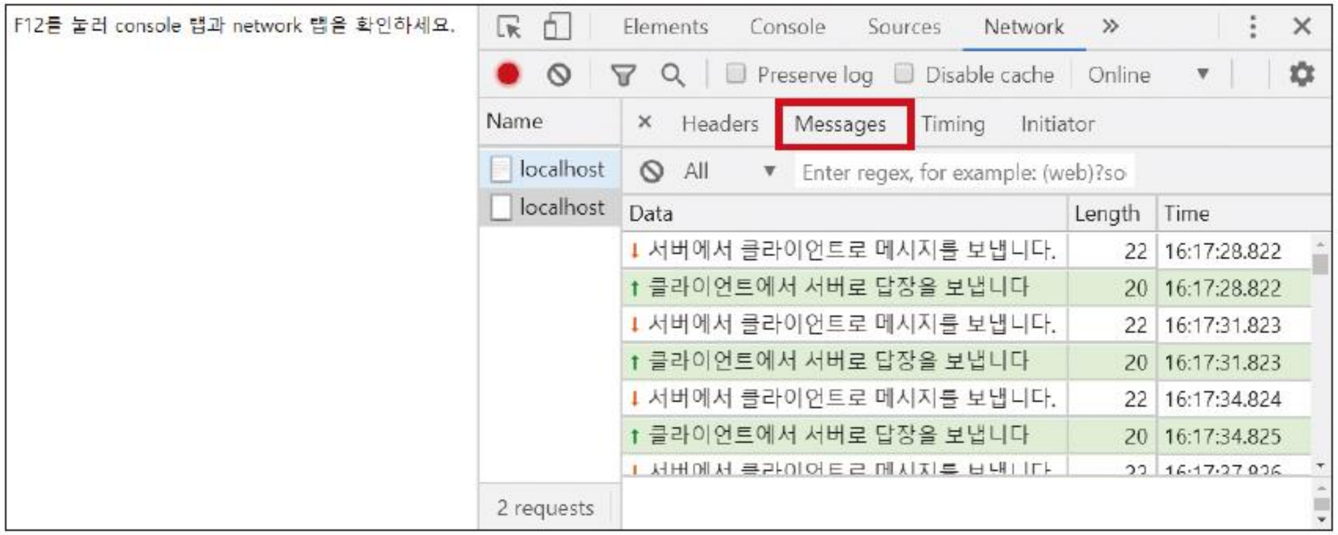
▼ 그림 12-6 Network 탭 화면



Name	Met...	Sta...	Protocol	Type	Time	Waterfall
localhost	GET	304	http/1.1	document	4 ms	
localhost	GET	101	websocket	websocket	Pending	

  - WebSocket allows continuous data exchange with just one initial request.

▼ 그림 12-7 웹 소켓 Messages 탭

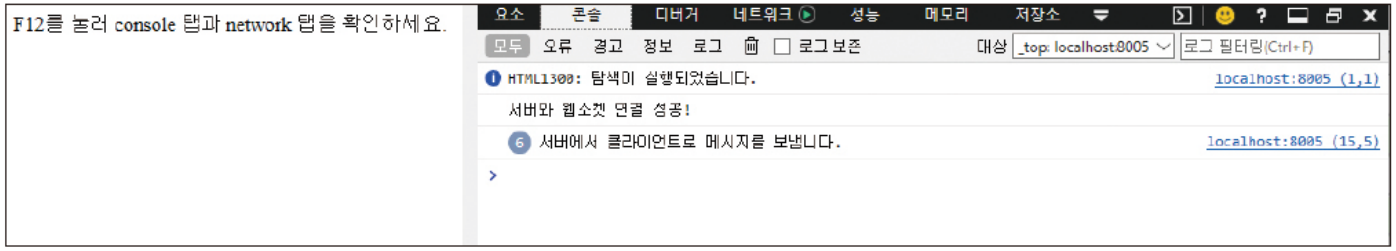


Data	Length	Time
↓ 서버에서 클라이언트로 메시지를 보냅니다.	22	16:17:28.822
↑ 클라이언트에서 서버로 답장을 보냅니다	20	16:17:28.822
↓ 서버에서 클라이언트로 메시지를 보냅니다.	22	16:17:31.823
↑ 클라이언트에서 서버로 답장을 보냅니다	20	16:17:31.823
↓ 서버에서 클라이언트로 메시지를 보냅니다.	22	16:17:34.824
↑ 클라이언트에서 서버로 답장을 보냅니다	20	16:17:34.825
↓ 서버에서 클라이언트로 메시지를 보냅니다.	22	16:17:37.826

Connect using a different browser as well

- Access http://localhost:8005 from another browser.
  - Since there are two connected browsers (clients), the amount of messages received by the server will be doubled.

그림 12-8 다른 브라우저 접속 화면



F12를 눌러 console 탭과 network 탭을 확인하세요.

```
HTML1300: 탐색이 실행되었습니다.                    localhost:8005 (1,1)
서버와 웹소켓 연결 성공!
6 서버에서 클라이언트로 메시지를 보냅니다.            localhost:8005 (15,5)
>
```

노드 콘솔

```
...
클라이언트에서 서버로 메시지를 보냅니다
클라이언트에서 서버로 메시지를 보냅니다
GET / 200 0.541 ms - 536
새로운 클라이언트 접속 ::1
클라이언트에서 서버로 메시지를 보냅니다
클라이언트에서 서버로 메시지를 보냅니다
...
```

Close one of the clients

- Close one of the browsers.
  - A message indicating disconnection will appear in the console, and the amount of messages will become one.

노드 콘솔

```
...
클라이언트에서 서버로 메시지를 보냅니다
클라이언트에서 서버로 메시지를 보냅니다
클라이언트 접속 해제 ::1
클라이언트에서 서버로 메시지를 보냅니다
클라이언트에서 서버로 메시지를 보냅니다
...
```

  - For convenience, use the Socket.IO module instead of the ws module.

Web Socket based Socket.IO

CSLAB

Install Socket.IO

- npm i socket.io
  - Replace the ws package with Socket.IO.

  - Import the Socket.IO package and connect it to the Express server. The second argument is the path (/socket.io) where the client can connect.
  - The **connection** event is triggered when a connection to the server is established, providing the socket object as a callback.
  - Access the request object through **socket.request**, and identify the socket's unique ID with **socket.id**.
  - The **disconnect** event is triggered when the connection is terminated, and the **error** event is triggered in case of an error.
  - The **reply** event is a custom event created by the user. When the **reply** event occurs on the client, it is transmitted to the server.
  - Use **socket.emit** to send messages. The first argument is the event name, and the second argument is the message.

콘솔

```
$ npm i socket.io@4
```

socket.js

```
const SocketIO = require('socket.io');

module.exports = (server) => {
  const io = SocketIO(server, { path: '/socket.io' });

  io.on('connection', (socket) => { // 웹 소켓 연결 시
    const req = socket.request;
    const ip = req.headers['x-forwarded-for'] || req.connection.remoteAddress;
    console.log('새로운 클라이언트 접속!', ip, socket.id, req.ip);
    socket.on('disconnect', () => { // 연결 종료 시
      console.log('클라이언트 접속 해제', ip, socket.id);
      clearInterval(socket.interval);
    });
    socket.on('error', (error) => { // 에러 시
      console.error(error);
    });
    socket.on('reply', (data) => { // 클라이언트로부터 메시지 수신 시
      console.log(data);
    });
    socket.interval = setInterval(() => { // 3초마다 클라이언트로 메시지 전송
      socket.emit('news', 'Hello Socket.IO');
    }, 3000);
```

Exchange messages between the server and the client

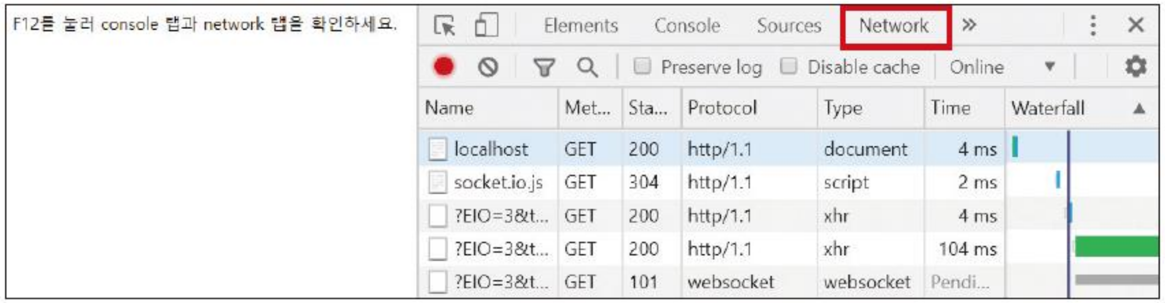
- Modify index.html:
  - Include the /socket.io/socket.io.js script (providing the io object).
  - Connect to the server address using the connect method and provide the same path as the server (/socket.io).
  - Note that the server address uses the http protocol.
  - Wait for the 'news' event from the server using the news event listener.
  - Trigger the 'reply' event with a message using socket.emit('reply', message).

```
<script src="/socket.io/socket.io.js"></script>
<script>
  const socket = io.connect('http://localhost:8005', {
    path: '/socket.io',
  });
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('reply', 'Hello Node.JS');
  });
</script>
```

Run server

- Connect to http://localhost:8005
  - You can see both WebSocket and polling connections in the Network tab of the developer tools.

▼ 그림 12-9 폴링과 웹 소켓 모두 존재



F12를 눌러 console 탭과 network 탭을 확인하세요.

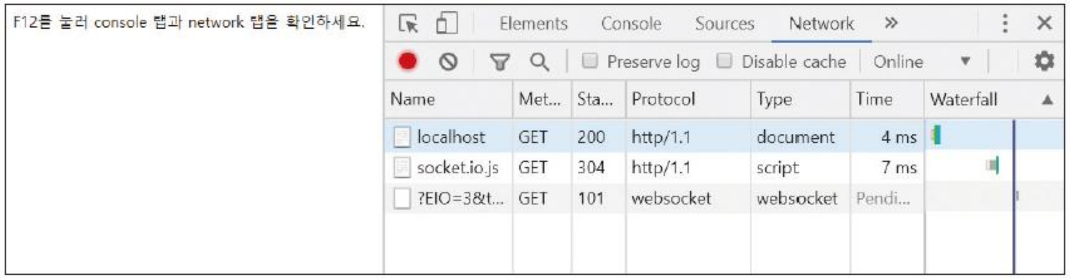
Name	Met...	Sta...	Protocol	Type	Time	Waterfall
localhost	GET	200	http/1.1	document	4 ms	
socket.io.js	GET	304	http/1.1	script	2 ms	
?EIO=3&t...	GET	200	http/1.1	xhr	4 ms	
?EIO=3&t...	GET	200	http/1.1	xhr	104 ms	
?EIO=3&t...	GET	101	websocket	websocket	Pendi...	

  - Socket.IO initially establishes a connection using the polling method (for browsers that do not support WebSocket). If WebSocket is available, it upgrades to WebSocket.
  - If you want to use WebSocket exclusively, you can achieve this by providing the transports option as follows:

index.html

```
...
<script>
  const socket = io.connect('http://localhost:8005', {
    path: '/socket.io',
    transports: ['websocket'],
  });
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('reply', 'Hello Node.JS');
  });
</script>
...
```

▼ 그림 12-10 웹 소켓만 사용 시



F12를 눌러 console 탭과 network 탭을 확인하세요.

Name	Met...	Sta...	Protocol	Type	Time	Waterfall
localhost	GET	200	http/1.1	document	4 ms	
socket.io.js	GET	304	http/1.1	script	7 ms	
?EIO=3&t...	GET	101	websocket	websocket	Pendi...	

Gif Chat-Room

Setting up the Project Structure

- Installing required packages and creating schema

콘솔

```
$ npm i mongoose multer color-hash@2
```

  - The color-hash package is used to assign colors to anonymous nicknames.

Creating Schema

- Creating Chat Room Schema (room.js) and Chat Schema (chat.js)

schemas/room.js

```
const mongoose = require('mongoose');

const { Schema } = mongoose;
const roomSchema = new Schema({
  title: {
    type: String,
    required: true,
  },
  max: {
    type: Number,
    required: true,
    default: 10,
    min: 2,
  },
  owner: {
    type: String,
    required: true,
  },
  password: String,
  createdAt: {
    type: Date,
    default: Date.now,
  },
});

module.exports = mongoose.model('Room', roomSchema);
```

schemas/chat.js

```
const mongoose = require('mongoose');

const { Schema } = mongoose;
const { Types: { ObjectId } } = Schema;
const chatSchema = new Schema({
  room: {
    type: ObjectId,
    required: true,
    ref: 'Room',
  },
  user: {
    type: String,
    required: true,
  },
  chat: String,
  gif: String,
  createdAt: {
    type: Date,
    default: Date.now,
  },
});

module.exports = mongoose.model('Chat', chatSchema);
```

Connecting Schema

- Connect the schema to index.js
  - link express with mongoose
  - secret key in the .env file

.env

```
COOKIE_SECRET=gifchat
MONGO_ID=root
MONGO_PASSWORD=nodejsbook
```

app.js

```
...
const webSocket = require('./socket');
const indexRouter = require('./routes');
const connect = require('./schemas');

const app = express();
app.set('port', process.env.PORT || 8005);
app.set('view engine', 'html');
nunjucks.configure('views', {
  express: app,
  watch: true,
});
connect();
...
```

schemas/index.js

```
const mongoose = require('mongoose');

const { MONGO_ID, MONGO_PASSWORD, NODE_ENV } = process.env;
const MONGO_URL = `mongodb://${MONGO_ID}:${MONGO_PASSWORD}@localhost:27017/admin`;

const connect = () => {
  if (NODE_ENV !== 'production') {
    mongoose.set('debug', true);
  }
  mongoose.connect(MONGO_URL, {
    dbName: 'gifchat',
    useNewUrlParser: true,
  }, (error) => {
    if (error) {
      console.log('몽고디비 연결 에러', error);
    } else {
      console.log('몽고디비 연결 성공');
    }
  });
};

mongoose.connection.on('error', (error) => {
  console.error('몽고디비 연결 에러', error);
});
mongoose.connection.on('disconnected', () => {
  console.error('몽고디비 연결이 끊겼습니다. 연결을 재시도합니다.');
  connect();
});

module.exports = connect;
```

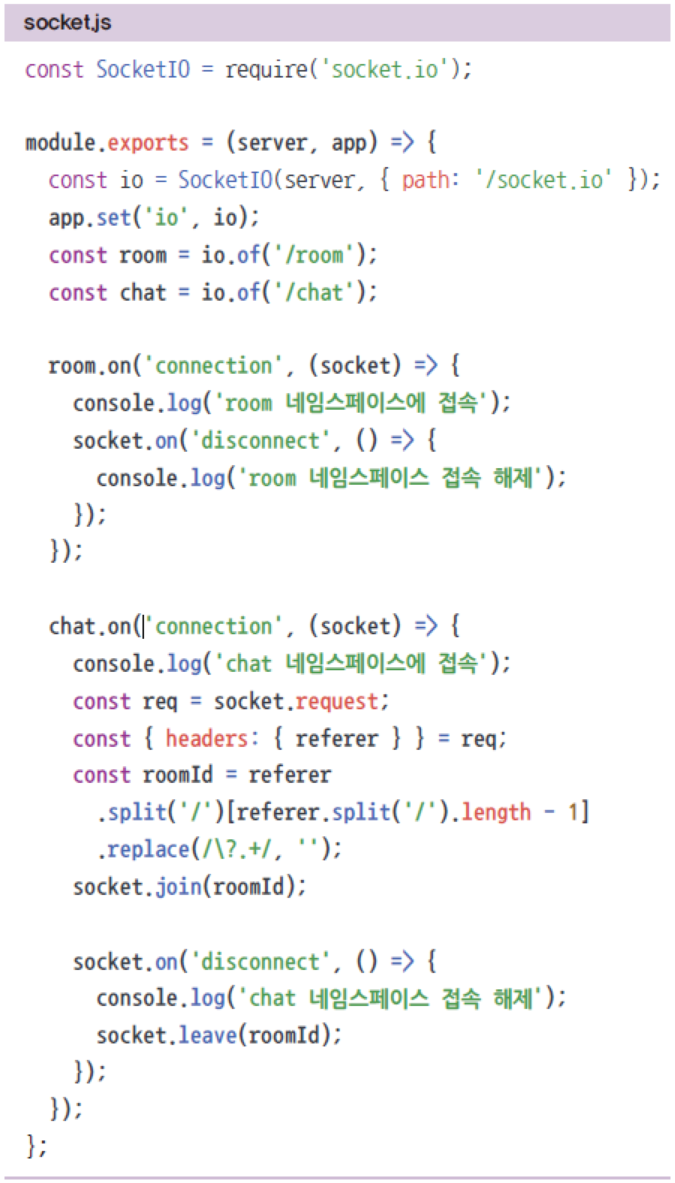
Write frontend file

- https://github.com/ZeroCho/nodejs-book/tree/master/ch12/12.4/gif-chat

- Create **views/layout.html, public/main.css, views/main.html, views/room.html, views/chat.html**
- Notice that the address of **io.connect** has changed in the code of main.html.
- **/room** in the address is a namespace (data can only be transmitted between the same namespaces)
- Connect the **newRoom** (an event to add a room to the list when a new room is created) and **removeRoom** (an event to remove a room from the list when the room is exploded) events to the socket.
- In chat.html, connect to the /chat namespace.
- Connect the **join event** (registering a system message indicating that you have entered when joining a room) and the **exit event** (registering a system message indicating that you have left when leaving the room)

Connecting socket event to socket.js

- Modify socket.js
  - Save the io object so that it can be used in the router with **app.set('io', io)**; (accessible with **req.app.get('io')**)
  - **io.of** is a method to access the namespace
  - Events can be sent separately to each namespace.
  - **req.headers.referer** contains request address
  - Extract the room ID from the request address and enter the room with **socket.join**
  - You can leave the room with **socket.leave**
  - **socket.join** and **leave** are methods prepared by Socket.IO



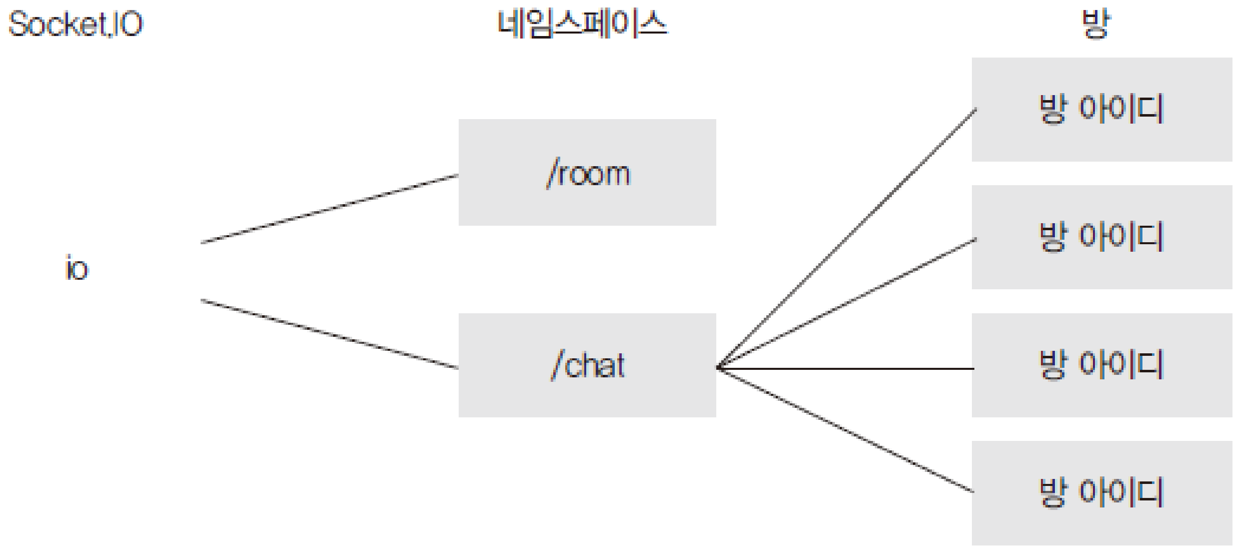
socket.js

```
const SocketIO = require('socket.io');

module.exports = (server, app) => {
  const io = SocketIO(server, { path: '/socket.io' });
  app.set('io', io);
  const room = io.of('/room');
  const chat = io.of('/chat');

  room.on('connection', (socket) => {
    console.log('room 네임스페이스에 접속');
    socket.on('disconnect', () => {
      console.log('room 네임스페이스 접속 해제');
    });
  });

  chat.on('connection', (socket) => {
    console.log('chat 네임스페이스에 접속');
    const req = socket.request;
    const { headers: { referer } } = req;
    const roomId = referer
      .split('/')[referer.split('/').length - 1]
      .replace(/\?.+/, '');
    socket.join(roomId);

    socket.on('disconnect', () => {
      console.log('chat 네임스페이스 접속 해제');
      socket.leave(roomId);
    });
  });
};
```

Understanding room

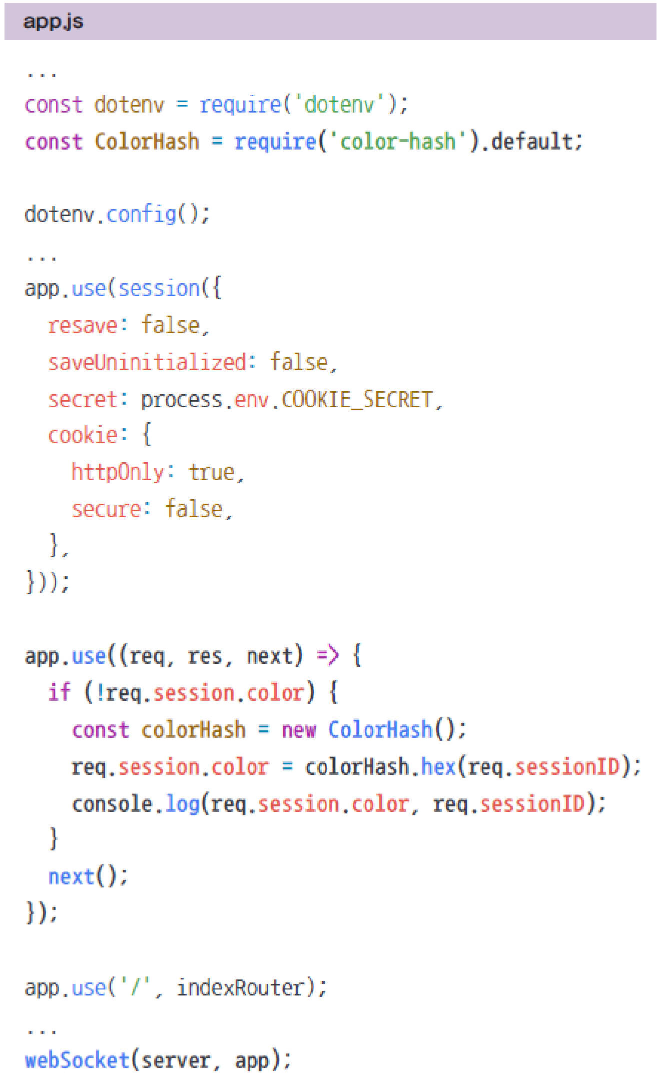
- In Socket.IO, there are namespaces and rooms under the io object.
  - The default namespace is /
  - Room is a subconcept of namespace.
  - You can only communicate within the same namespace and in the same room.

▼ 그림 12-11 네임스페이스와 방



Apply color-hash

- Since it is an anonymous chat, visitors are given a unique color ID.
  - Granted according to sessionID
  - Save color ID in session (req.session.color)



app.js

```
...
const dotenv = require('dotenv');
const ColorHash = require('color-hash').default;

dotenv.config();
...
app.use(session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
}));

app.use((req, res, next) => {
  if (!req.session.color) {
    const colorHash = new ColorHash();
    req.session.color = colorHash.hex(req.sessionID);
    console.log(req.session.color, req.sessionID);
  }
  next();
});

app.use('/', indexRouter);
...
webSocket(server, app);
```

Middleware and Socket

CSLAB

Using sessions in socket.io

- After modifying app.js, connect to Socket.IO middleware
  - Express middleware can be used in Socket.io with namespace.use

app.js

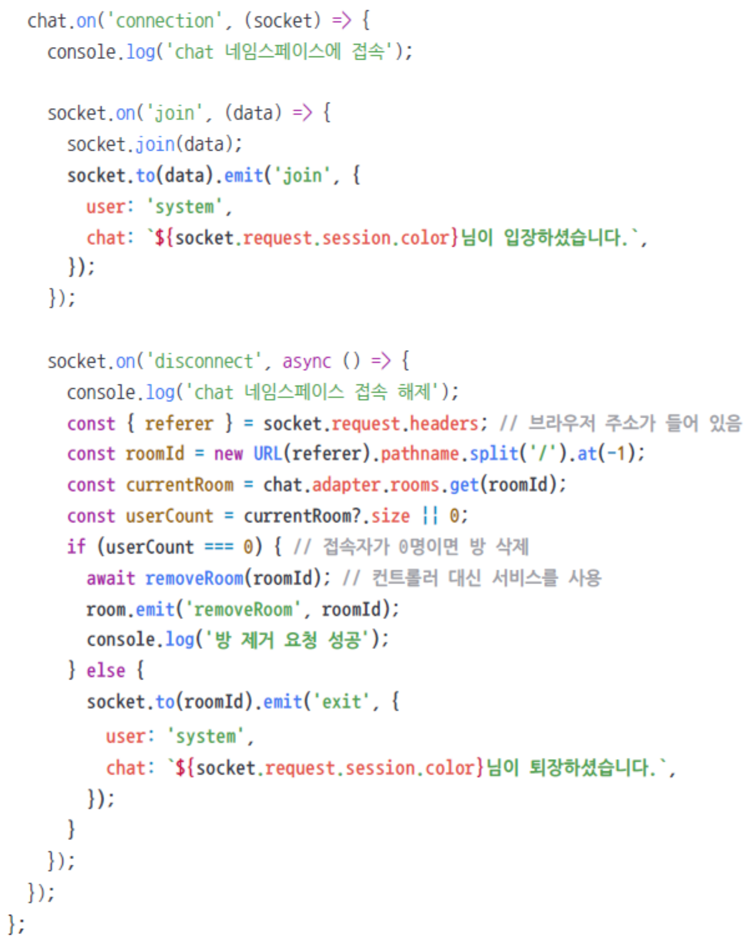
```
...
connect();

const sessionMiddleware = session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
});
...
app.use(cookieParser(process.env.COOKIE_SECRET));
app.use(sessionMiddleware);
...
const server = app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기 중');
});

webSocket(server, app, sessionMiddleware);
```

socket.js

```
const SocketIO = require('socket.io');
const { removeRoom } = require('./services');

module.exports = (server, app, sessionMiddleware) => {
  const io = SocketIO(server, { path: '/socket.io' });
  app.set('io', io);
  const room = io.of('/room');
  const chat = io.of('/chat');

  const wrap = middleware => (socket, next) => middleware(socket.request, {}, next);  ❶
  chat.use(wrap(sessionMiddleware));

  room.on('connection', (socket) => {
    console.log('room 네임스페이스에 접속');
    socket.on('disconnect', () => {
      console.log('room 네임스페이스 접속 해제');
    });
  });
```

Sending room entry and exit messages

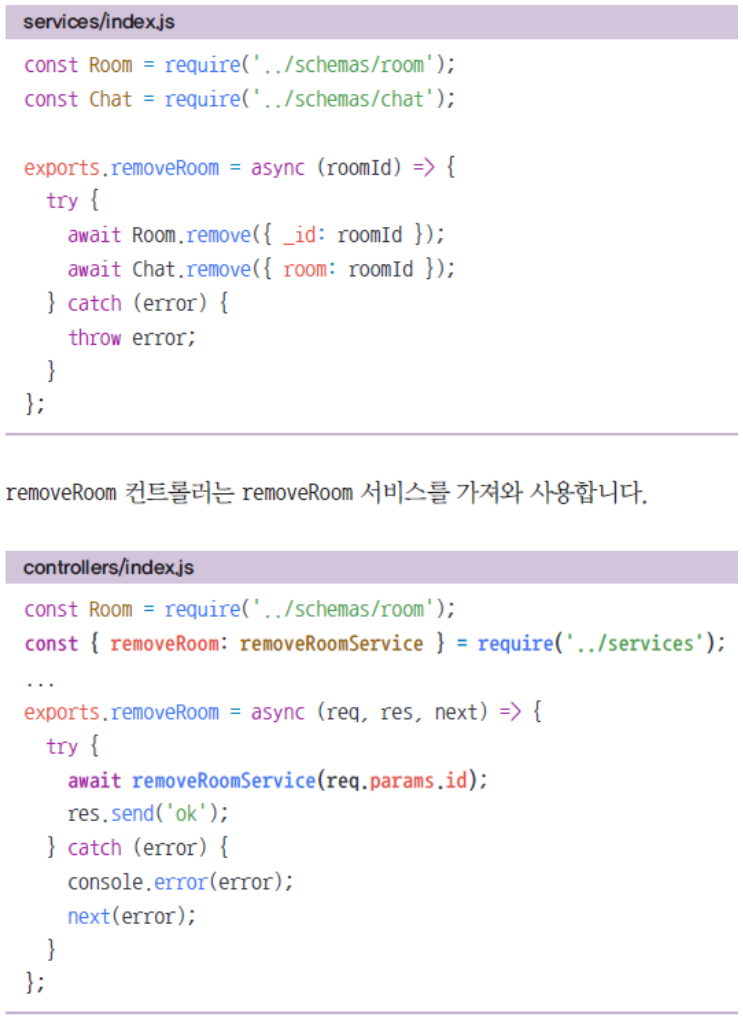
- Send data to a specific room with **to(room ID).emit(event, message)**
  - Added room explosion function when there are 0 users
  - socket.adapter.rooms.get(room ID) displays a list of socket IDs in the room.
  - Number of people in room can be determined by .size (not accurate)
  - The room explosion function is implemented separately as an express service.
  - Send request to router with axios
  - The reason for implementing it separately is because it is convenient to process DB work on the router.



```
  chat.on('connection', (socket) => {
    console.log('chat 네임스페이스에 접속');

    socket.on('join', (data) => {
      socket.join(data);
      socket.to(data).emit('join', {
        user: 'system',
        chat: `${socket.request.session.color}님이 입장하셨습니다.`,
      });
    });

    socket.on('disconnect', async () => {
      console.log('chat 네임스페이스 접속 해제');
      const { referer } = socket.request.headers; // 브라우저 주소가 들어 있음
      const roomId = new URL(referer).pathname.split('/').at(-1);
      const currentRoom = chat.adapter.rooms.get(roomId);
      const userCount = currentRoom?.size || 0;
      if (userCount === 0) { // 접속자가 0명이면 방 삭제
        await removeRoom(roomId); // 컨트롤러 대신 서비스를 사용
        room.emit('removeRoom', roomId);
        console.log('방 제거 요청 성공');
      } else {
        socket.to(roomId).emit('exit', {
          user: 'system',
          chat: `${socket.request.session.color}님이 퇴장하셨습니다.`,
        });
      }
    });
  });
};
```

Writing a Router

- Source code is https://github.com/ZeroCho/nodejs-book/tree/master/ch12/12.5/gif-chat
  - Write routes/index.js
  - GET /: Main page (room list) access router
  - GET /room: Room creation screen router
  - POST /room: Room creation request router
  - GET /room/:id room entry router
  - DELETE /room/:id remove room router



services/index.js

```
const Room = require('../schemas/room');
const Chat = require('../schemas/chat');

exports.removeRoom = async (roomId) => {
  try {
    await Room.remove({ _id: roomId });
    await Chat.remove({ room: roomId });
  } catch (error) {
    throw error;
  }
};
```

removeRoom 컨트롤러는 removeRoom 서비스를 가져와 사용합니다.

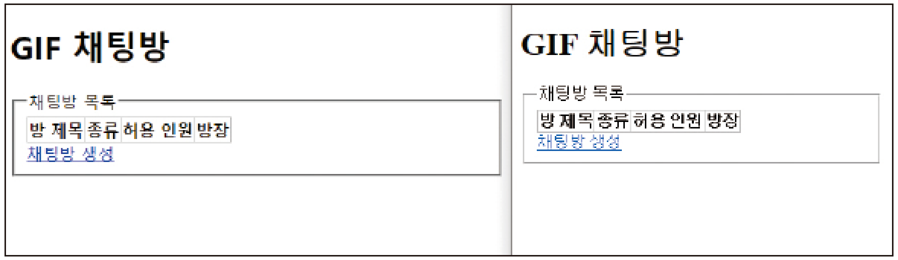
controllers/index.js

```
const Room = require('../schemas/room');
const { removeRoom: removeRoomService } = require('../services');
...
exports.removeRoom = async (req, res, next) => {
  try {
    await removeRoomService(req.params.id);
    res.send('ok');
  } catch (error) {
    console.error(error);
    next(error);
  }
};
```

Creating a room

- Run both MongoDB and server
  - Open two browsers and go to http://localhost:8005
  - Same effect as two people connected
  - Create a room

▼ 그림 12-12 크롬과 엣지 브라우저에서 접속한 화면



GIF 채팅방

채팅방 목록

방 제목	종류	허용 인원	방장

채팅방 생성

GIF 채팅방

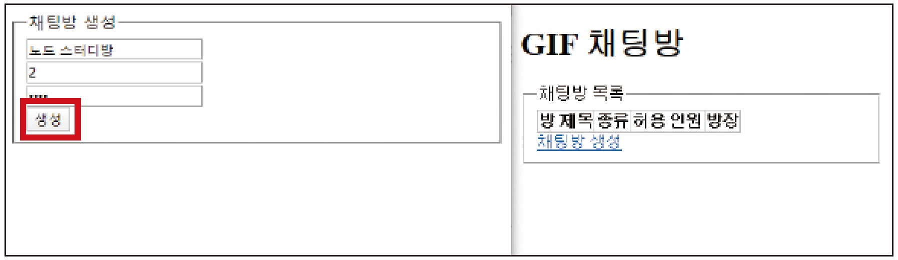
채팅방 목록

방 제목	종류	허용 인원	방장

채팅방 생성

한 브라우저에서 방을 생성해봅시다.

▼ 그림 12-13 방 생성 화면



채팅방 생성

노드 스터디방

2

****

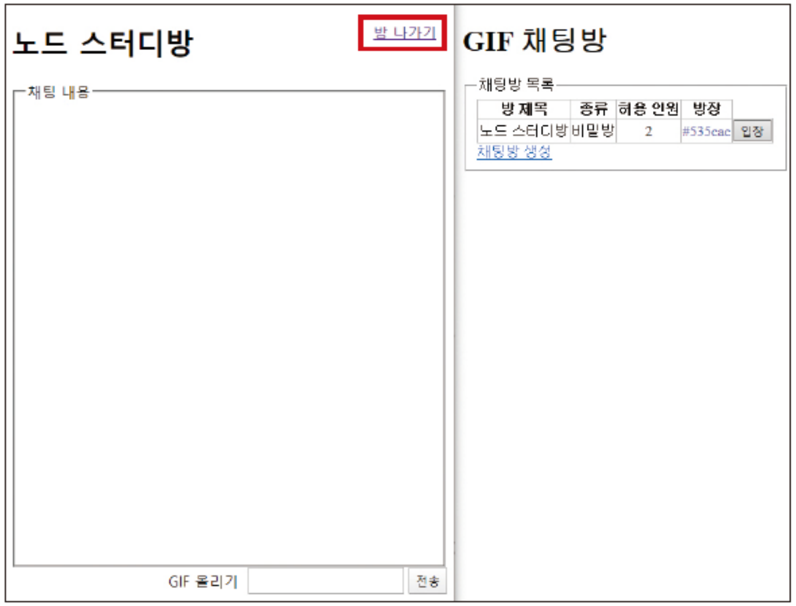
GIF 채팅방

채팅방 목록

방 제목	종류	허용 인원	방장

채팅방 생성

▼ 그림 12-14 방 생성 후 화면



노드 스터디방

GIF 채팅방

채팅방 목록

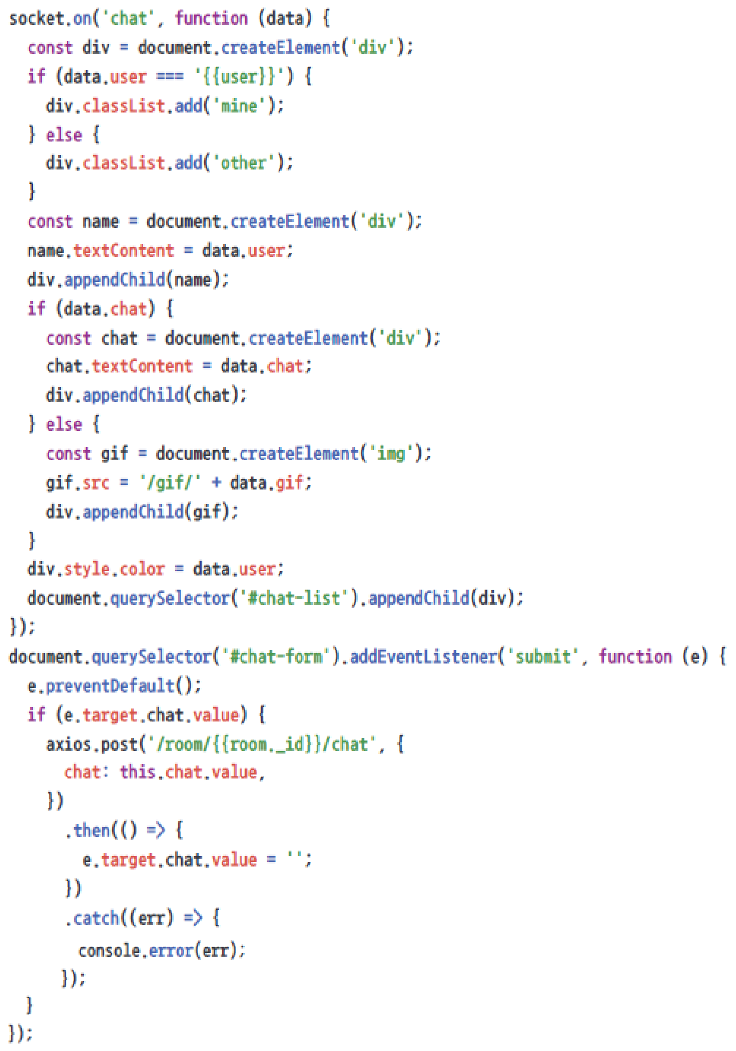
방 제목	종류	허용 인원	방장	
노드 스터디방	비밀방	2	#535cac	입장

채팅방 생성

Implementing Chat

Attaching a chat socket event listener

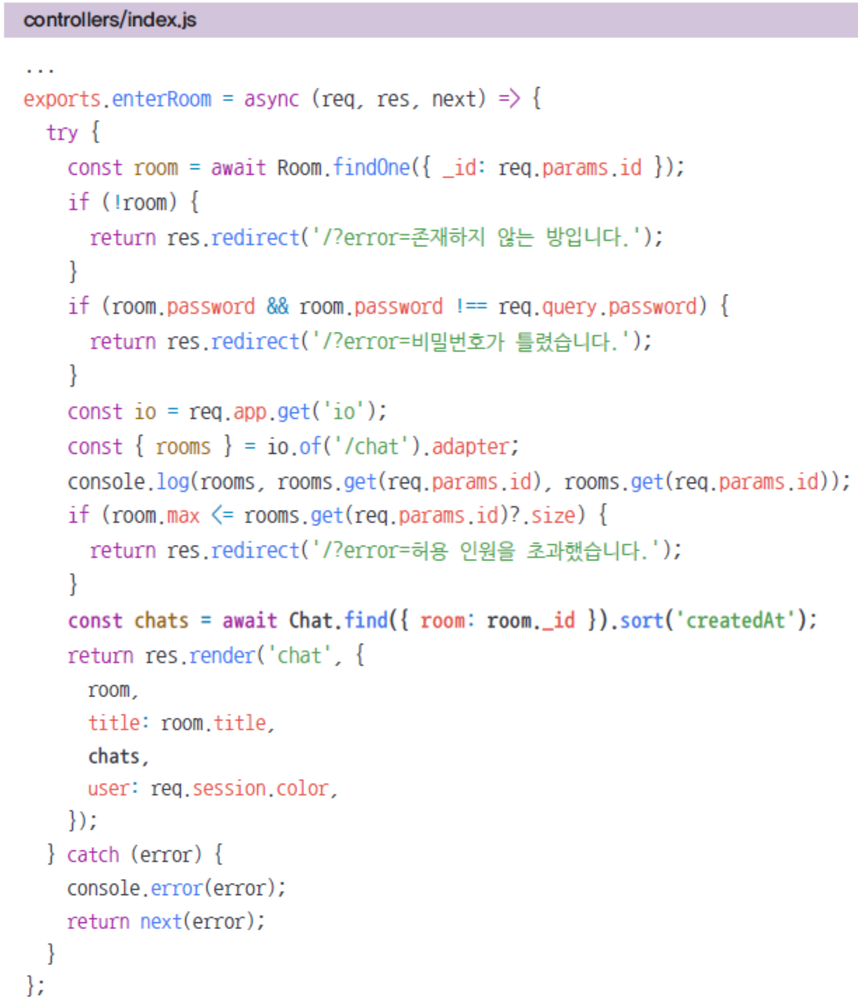
- Edit https://github.com/ZeroCho/nodejs-book/blob/master/ch12/12.6/gif-chat/views/chat.html
  - Added chat event listener. Called when a chat message is sent to a web socket
  - Rendering differently depending on event.data.user (chat sender)



```
socket.on('chat', function (data) {
  const div = document.createElement('div');
  if (data.user === '{{user}}') {
    div.classList.add('mine');
  } else {
    div.classList.add('other');
  }
  const name = document.createElement('div');
  name.textContent = data.user;
  div.appendChild(name);
  if (data.chat) {
    const chat = document.createElement('div');
    chat.textContent = data.chat;
    div.appendChild(chat);
  } else {
    const gif = document.createElement('img');
    gif.src = '/gif/' + data.gif;
    div.appendChild(gif);
  }
  div.style.color = data.user;
  document.querySelector('#chat-list').appendChild(div);
});
document.querySelector('#chat-form').addEventListener('submit', function (e) {
  e.preventDefault();
  if (e.target.chat.value) {
    axios.post('/room/{{room._id}}/chat', {
      chat: this.chat.value,
    })
      .then(() => {
        e.target.chat.value = '';
      })
      .catch((err) => {
        console.error(err);
      });
  }
});
```

Create a router to connect to the room

- If available, load and render chat



controllers/index.js

```
...
exports.enterRoom = async (req, res, next) => {
  try {
    const room = await Room.findOne({ _id: req.params.id });
    if (!room) {
      return res.redirect('/?error=존재하지 않는 방입니다.');
    }
    if (room.password && room.password !== req.query.password) {
      return res.redirect('/?error=비밀번호가 틀렸습니다.');
    }
    const io = req.app.get('io');
    const { rooms } = io.of('/chat').adapter;
    console.log(rooms, rooms.get(req.params.id), rooms.get(req.params.id));
    if (room.max <= rooms.get(req.params.id)?.size) {
      return res.redirect('/?error=허용 인원을 초과했습니다.');
    }
    const chats = await Chat.find({ room: room._id }).sort('createdAt');
    return res.render('chat', {
      room,
      title: room.title,
      chats,
      user: req.session.color,
    });
  } catch (error) {
    console.error(error);
    return next(error);
  }
};
```

Create a chat router

- Chats are saved in the DB and distributed to the room.

```
exports.sendChat = async (req, res, next) => {
  try {
    const chat = await Chat.create({
      room: req.params.id,
      user: req.session.color,
      chat: req.body.chat,
    });
    req.app.get('io').of('/chat').to(req.params.id).emit('chat', chat);
    res.send('ok');
  } catch (error) {
    console.error(error);
    next(error);
  }
};
```

routes/index.js

```
const express = require('express');
const {
  renderMain, renderRoom, createRoom, enterRoom, removeRoom, sendChat,
} = require('../controllers');

const router = express.Router();
...
router.delete('/room/:id', removeRoom);

router.post('/room/:id/chat', sendChat);

module.exports = router;
```

Chat Screen

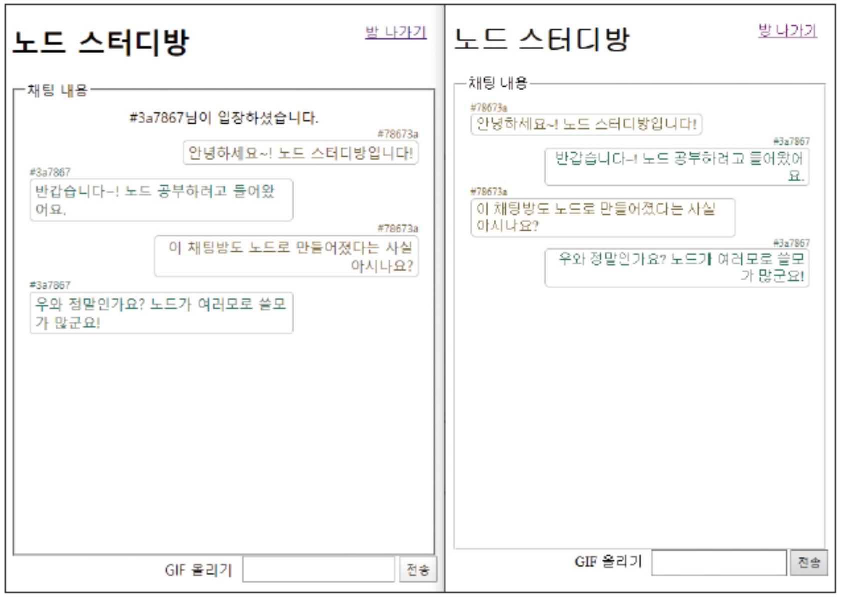


그림 12-15 실시간 채팅 화면

노드 스터디방

방 나가기

채팅 내용

#3a7867님이 입장하셨습니다.

#78673a
안녕하세요~! 노드 스터디방입니다!

#3a7867
반갑습니다~! 노드 공부하려고 들어왔어요.

#78673a
이 채팅방도 노드로 만들어졌다는 사실 아시나요?

#3a7867
우와 정말인가요? 노드가 여러모로 쓸모가 많군요!

GIF 올리기 전송

노드 스터디방

방 나가기

채팅 내용

#78673a
안녕하세요~! 노드 스터디방입니다!

#3a7867
반갑습니다~! 노드 공부하려고 들어왔어요.

#78673a
이 채팅방도 노드로 만들어졌다는 사실 아시나요?

#3a7867
우와 정말인가요? 노드가 여러모로 쓸모가 많군요!

GIF 올리기 전송

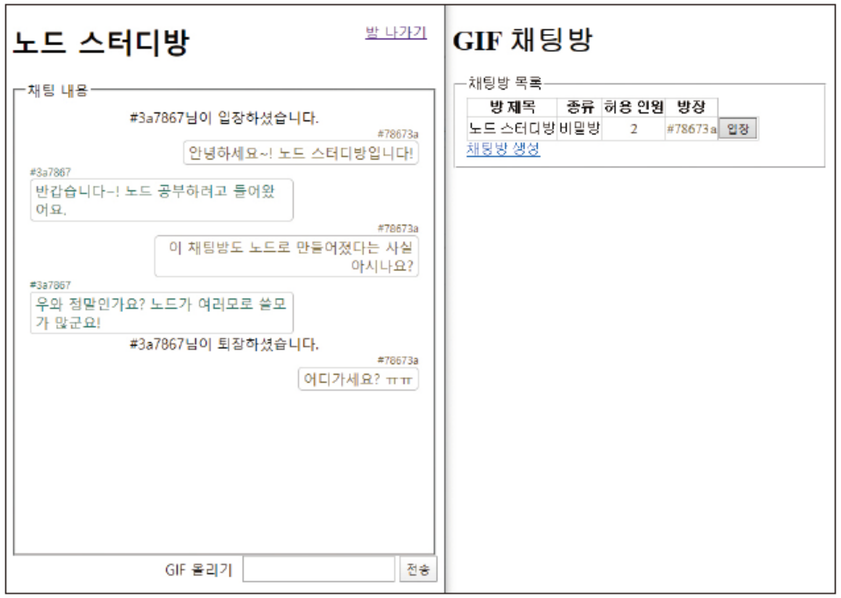


그림 12-16 퇴장 시 화면

노드 스터디방

방 나가기

채팅 내용

#3a7867님이 입장하셨습니다.

#78673a
안녕하세요~! 노드 스터디방입니다!

#3a7867
반갑습니다~! 노드 공부하려고 들어왔어요.

#78673a
이 채팅방도 노드로 만들어졌다는 사실 아시나요?

#3a7867
우와 정말인가요? 노드가 여러모로 쓸모가 많군요!

#3a7867님이 퇴장하셨습니다.

#78673a
어디가세요? ㅠㅠ

GIF 올리기 전송

GIF 채팅방

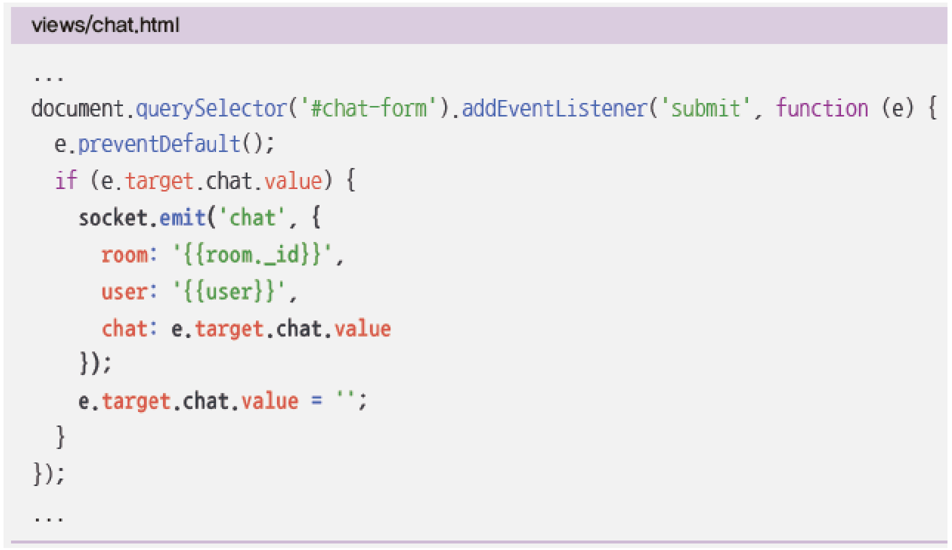
채팅방 목록

방 제목	종류	허용 인원	방장	
노드 스터디방	비밀방	2	#78673a	입장

채팅방 생성

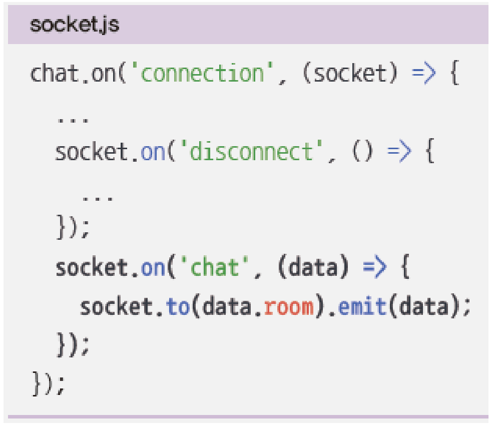
Implementing chat using only web sockets

- Chat can be sent directly via socket.emit without using DB
  - Edit chat.html, app.js



views/chat.html

```
...
document.querySelector('#chat-form').addEventListener('submit', function (e) {
  e.preventDefault();
  if (e.target.chat.value) {
    socket.emit('chat', {
      room: '{{room._id}}',
      user: '{{user}}',
      chat: e.target.chat.value
    });
    e.target.chat.value = '';
  }
});
...
```

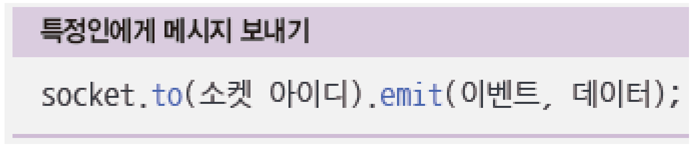


socket.js

```
chat.on('connection', (socket) => {
  ...
  socket.on('disconnect', () => {
    ...
  });
  socket.on('chat', (data) => {
    socket.to(data.room).emit(data);
  });
});
```

Other Socket.IO APIs

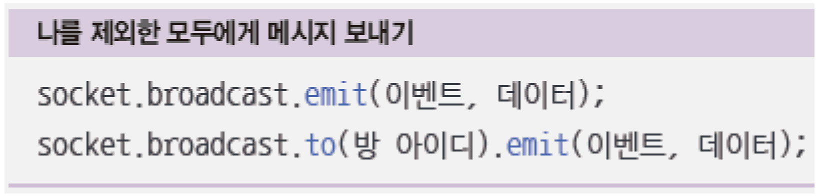
- Send a message to a specific person (used for whispers, 1:1 chat, etc.)



특정인에게 메시지 보내기

```
socket.to(소켓 아이디).emit(이벤트, 데이터);
```

- Send message to everyone except me

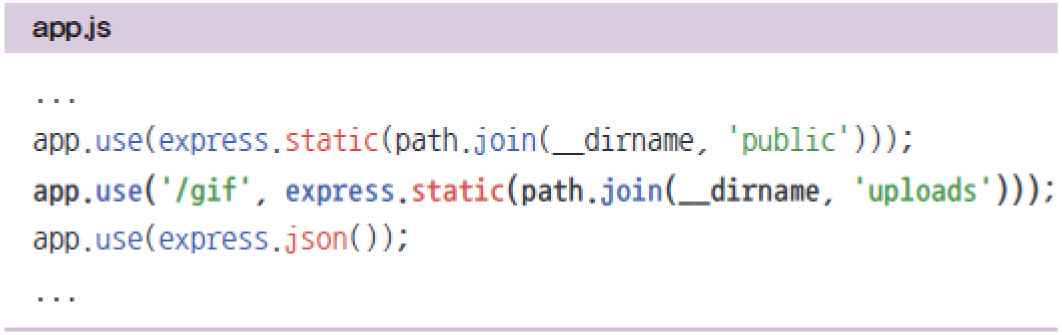


나를 제외한 모두에게 메시지 보내기

```
socket.broadcast.emit(이벤트, 데이터);
socket.broadcast.to(방 아이디).emit(이벤트, 데이터);
```

Implementing GIF transfer

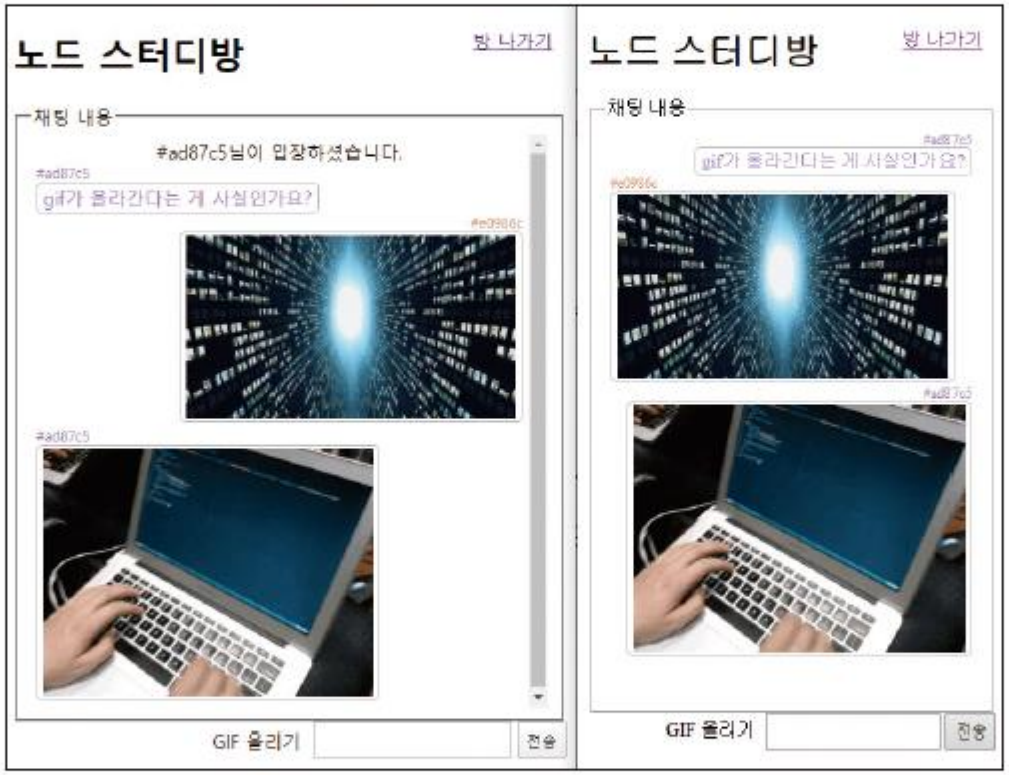
- Copy source code
  - https://github.com/ZeroCho/nodejs-book/blob/master/ch12/12.7/gif-chat/views/chat.html
  - https://github.com/ZeroCho/nodejs-book/blob/master/ch12/12.7/gif-chat/routes/index.js
  - Since it is an image upload, multer is used
  - After saving the image, spread the file path to chat data
  - Link static folder to provide images


app.js

```
...
app.use(express.static(path.join(__dirname, 'public')));
app.use('/gif', express.static(path.join(__dirname, 'uploads')));
app.use(express.json());
...
```

GIF chat screen

그림 12-17 GIF 파일 업로드 화면



노드 스터디방

방 나가기

채팅 내용

#ad87c5님이 입장하셨습니다.

#ad87c5

gif가 올라간다는 게 사실인가요?

#e0986c

#ad87c5

GIF 올리기

전송

노드 스터디방

방 나가기

채팅 내용

#ad87c5

gif가 올라간다는 게 사실인가요?

#e0986c

#ad87c5

GIF 올리기

전송