네트워크서비스 프로토콜

13주차 2~14주차

» 소프트웨어학부
» 김형균 교수

수업에 들어가며

- 지난 시간 복습
  - 퀴즈 – Top3 스타벅스 커피쿠폰
- 오늘 학습할 내용
  - 실시간 서비스
  - TCP 실시간 통신 서비스
  - Socket.IO 를 이용한 실시간 웹 서비스
  - Socket.IO 웹 서비스 실습 1
  - Socket.IO 를 이용한 실시간 웹 서비스
    - 데이터 교환
  - Socket.IO 웹 서비스 실습 2

- 오늘 학습할 내용(계속)
- 실시간 채팅 서비스 만들기
  - - 작업 폴더 및 모듈 설치
  - 1. 기초 채팅서버
  - - app.js 작성
  - - 정적 파일 제공
  - - 정적 파일 제공 : 폴더 및 파일 생성
  - - app.js 수정
  - 2. 기초 채팅 서비스
  - - app.js 수정
  - - index.html 수정
  - - index.js 작성
  - 3.채팅 기능 구현
  - - 채팅 기능 설계
  - - 이벤트 명 정의
  - - 메시지(알림)의 형식
  - - app.js 수정
  - - index.js 수정
  - - index.html 수정
  - - index.css 교체
  - - 테스트

채팅서버 만들기

실시간 서비스

- 실시간 서비스 구현하기

HTTP 통신

- 요청과 응답 기반
- 다시 요청할 때까지 변경사항 반영 안됨.

- HTTP로 실시간 서비스 작성이 힘들다.
- HTTP가 아는 다른 프로토콜 사용 → TCP

TCP 통신

- 네트워크 레이어 : Tranport Layer
- 스트림을 이용한 실시간 통신
- 소켓을 이용한 네트워크 프로그래밍

실시간 서비스

- TCP
  - 연결 지향이므로 연결 과정 필요
  - 연결 과정
    - ① 서버 소켓 생성, 준비, 대기
    - ② 클라이언트 소켓 연결, 소켓간 연결
    - ③ 데이터 교환
    - ④ 접속 끊기

TCP 실시간 통신 서비스

Node.js 교과서

net 모듈

소켓 통신을 위한 기본 모듈 : net

- var net = require('net');

TCP 서버

서버 생성

- var server = net.createServer([options][, connectionListener])

서버 함수

- server.listen(port[, host][, backlog][, callback]) : 클라이언트 접속 대기
- server.close([callback]) : 추가 접속을 받지 않는다.
- server.getConnections(callback) : 연결 갯수
- server.address() : 서버 주소

TCP 실시간 통신 서비스

- TCP 서버
  - 서버 이벤트 : net.Server
    - listening : 포트 바인딩, 접속 가능한 상태 이벤트
    - connection : 클라이언트 접속 이벤트
    - close : 서버 닫기(연결된 소켓이 없을 때만 발생)
    - error : 에러

TCP 실시간 통신 서비스

Node.js 교과서

소켓 클라이언트

소켓 생성과 연결

```
var socket = new net.Socket();
var option = {
  host = 'localhost',
  port = 3000
};
socket.connect(option, function() {
});
```

TCP 실시간 통신 서비스

Node.js 교과서

- 원격 호스트사이의 데이터 교환
  - net.Socket 이벤트
    - connect : 원격 소켓 연결 이벤트
    - data : 읽을 수 있는 데이터 도착
    - end : 원격 호스트의 소켓 종료(FIN)
    - timeout : 제한 시간 지남
    - error : 에러

- net.socket
  - net.Socket 함수, 프로퍼티
    - connect(options[, connectListener]) : 연결
    - write(data[, encoding][, callback]) : 데이터 쓰기
    - end([data][, encoding]) : 연결 종료 신호(FIN) 보내기
    - setKeepAlive([enable][, initialDelay]) : 연결 유지
    - remoteAddress, remotePort : 원격 호스트 주소와 포트

소켓 서버

socket > JS tcpserver.js > ...

```
1  var net = require('net');
2  var server = net.createServer(function(socket){
3      console.log('클라이언트 접속');
4      socket.write('Welcome to socket server');
5
6      socket.on('data',function(chunk){
7          console.log('클라이언트가 보냄 : ',
8          chunk.toString());
9      });
10
11     socket.on('end',function(chunk){
12         console.log('클라이언트 접속종료 ');
13     });
14 });
15
16 server.on('listening', function(){
17     console.log('server is listening');
18 });
19
20 server.on('close', function(){
21     console.log('server closed');
22 });
23
24 server.listen(3000);
```

소켓 클라이언트

socket > JS tcpclient.js > ...

```
1  var net = require('net');
2  var ip = '127.0.0.1';
3  var port = 3000;
4
5  var socket = new net.Socket();
6  socket.connect({host:ip, port:port}, function(){
7      console.log('서버와 연결 성공');
8      socket.write('Hello socket server');
9      socket.end();
10
11     socket.on('data',function(chunk){
12         console.log('서버가 보냄 : ',
13         chunk.toString());
14     });
15
16     socket.on('end',function(chunk){
17         console.log('서버 연결 종료 ');
18     });
19 });
```

명령 프롬프트 - node tcpserver

```
C:Wsocket>node tcpserver
server is listening
클라이언트 접속
클라이언트가 보냄 :  Hello socket server
클라이언트 접속종료
클라이언트 접속
클라이언트가 보냄 :  Hello socket server
클라이언트 접속종료
클라이언트 접속
클라이언트가 보냄 :  Hello socket server
클라이언트 접속종료
```

명령 프롬프트

```
               2개 디렉터리  361,654,792,192

C:Wsocket>node tcpclient
서버와 연결 성공
서버가 보냄 :  Welcome to socket server
서버 연결 종료

C:Wsocket>node tcpclient
서버와 연결 성공
서버가 보냄 :  Welcome to socket server
서버 연결 종료

C:Wsocket>node tcpclient
서버와 연결 성공
서버가 보냄 :  Welcome to socket server
서버 연결 종료

C:Wsocket>
```

socket > JS tcpserver.js > ...

```
1   var net = require('net');
2   var server = net.createServer(function(socket){
3       console.log('클라이언트 접속');
4       socket.write('Welcome to socket server');
5
6       socket.on('data',function(chunk){
7           console.log('클라이언트가 보냄 : ',
8           chunk.toString());
9       });
10
11      socket.on('end',function(chunk){
12          console.log('클라이언트 접속종료 ');
13      });
14  });
15
16  server.on('listening', function(){
17      console.log('server is listening');
18  });
19
20  server.on('close', function(){
21      console.log('server closed');
22  });
23
24  server.listen(3000);
```

socket > JS tcpclient.js > ...

```
1   var net = require('net');
2   var ip = '127.0.0.1';
3   var port = 3000;
4
5   var socket = new net.Socket();
6   socket.connect({host:ip, port:port}, function(){
7       console.log('서버와 연결 성공');
8       socket.write('Hello socket server');
9       socket.end();
10
11      socket.on('data',function(chunk){
12          console.log('서버가 보냄 : ',
13          chunk.toString());
14      });
15
16      socket.on('end',function(chunk){
17          console.log('서버 연결 종료 ');
18      });
19  });
```

Socket.IO 를 이용한 실시간 웹 서비스

1. 실시간 웹 서비스와 socket.io

- 실시간 서비스 작성하기
  - HTTP 방식의 한계
  - Socket
    - ✓데스트탑 애플리케이션
    - ✓모바일 애플리케이션

- 실시간 웹 서비스를 위한 다양한 기술 시도
  - ajax, polling, long poling
  - 웹 소켓

문제는 - 다양한 웹 브라우저

- socket.io : 호환되는 기술 자동 선택
- http://socket.io

Socket.IO 를 이용한 실시간 웹 서비스

1. 실시간 웹 서비스와 socket.io

- 실시간 서비스 작성하기

  - socket.io : 호환되는 기술 자동 선택
  - http://socket.io

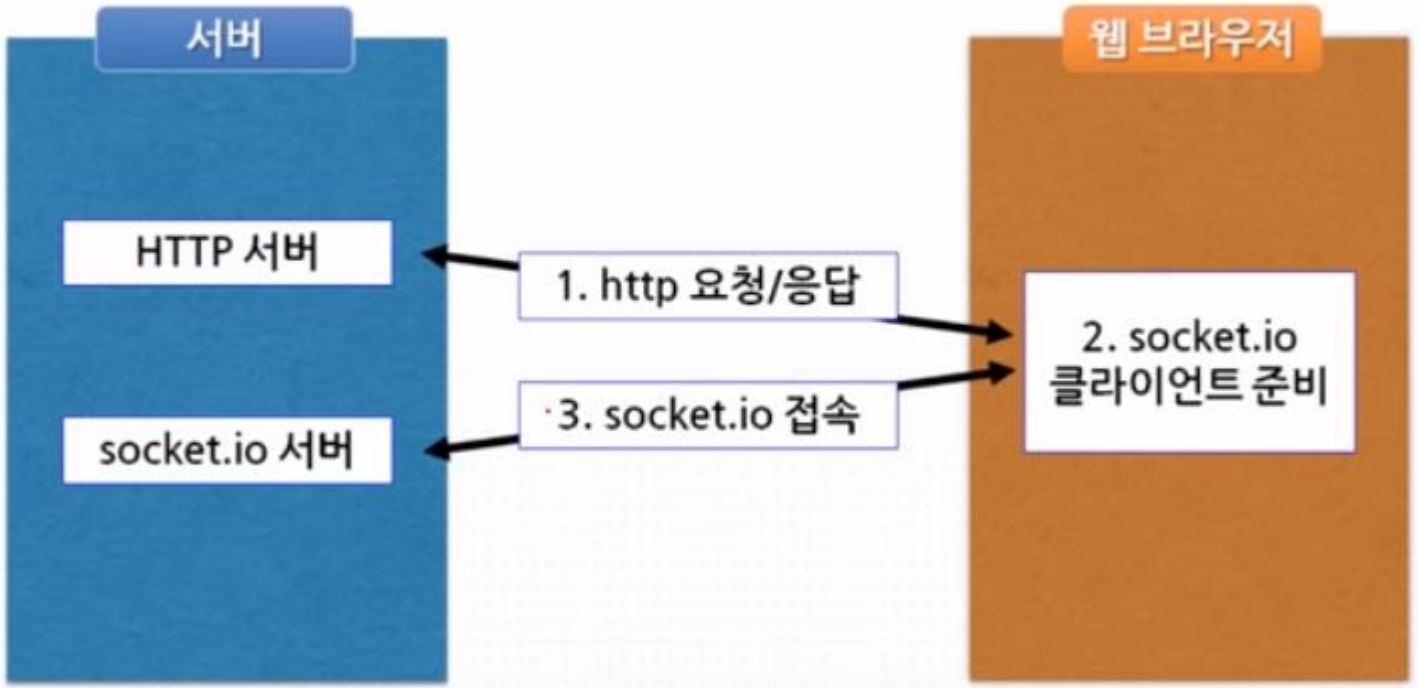
  - 설치
    - npm install socket.io

- socket.io 서버와 클라이언트

  - 서버
    - HTTP 서버
    - socket.io 서버
  - 클라이언트(웹 브라우저)
    - HTTP 클라이언트
    - socket.io 클라이언트

Socket.IO 를 이용한 실시간 웹 서비스

1. 실시간 웹 서비스와 socket.io



Socket.IO 를 이용한 실시간 웹 서비스

1. 실시간 웹 서비스와 socket.io

- 실시간 서비스를 위한 서버 준비
  - 두 서버 준비
    - 웹 서버 → http, express
    - socket.io 서버
  - socket.io 서버 생성
    - var Server = require('socket.io');
    - var io = new Server(httpServer);
  - 축약 형태
    - var io = require('socket.io')(server);
  - 연결 이벤트
    - connection

Socket.IO 를 이용한 실시간 웹 서비스

1. 실시간 웹 서비스와 socket.io

- socket.io 클라이언트
  - socket.io 클라이언트 준비
    - HTTP 서버에게 socket.io 초기화 HTML 요청
    - HTML 로딩 - 스트립트 로딩
    - socket.io 클라이언트 초기화
    - socket.io 서버 연결
  - 서버의 socket.io 클라이언트 html 응답

```
app.get('/', function(req, res) {
  res.sendFile(__dirname + '/client.html');
});
```

Socket.IO 를 이용한 실시간 웹 서비스

1. 실시간 웹 서비스와 socket.io

- socket.io 클라이언트
  - 스크립트 로딩
    - 서버 모듈 로딩 or CDN
    - <script src="/socket.io/socket.io.js"></script>
    - <script src="https://cdn.socket.io/socket.io-1.3.7.js"></script>
  - 클라이언트 소켓 클래스
    - IO(url:String, opts:Object):Socket
  - 소켓 생성, 연결
    - var socket = io();

Socket.IO 를 이용한 실시간 웹 서비스

1. 실시간 웹 서비스와 socket.io

- socket.io 클라이언트
  - socket.io 클라이언트 이벤트
    - connect : 서버와 연결
    - error : 연결 에러
    - disconnect : 연결 끊김
    - reconnect, reconnectiong, reconnect_error, ... : 재접속

서버와 연결 끊어지면 자동 재접속 시도

Socket.IO 웹 서비스 실습 1
-작업 폴더 및 모듈 설치

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\> cd socket
PS C:\socket> npm install express -s
+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in 1.849s
found 0 vulnerabilities

PS C:\socket> npm install socket.io -s
+ socket.io@2.3.0
added 45 packages from 28 contributors and audited 363 packages in 1.898s
found 0 vulnerabilities
```

Socket.IO 웹 서비스 실습 1

Node.js 교과서

서버

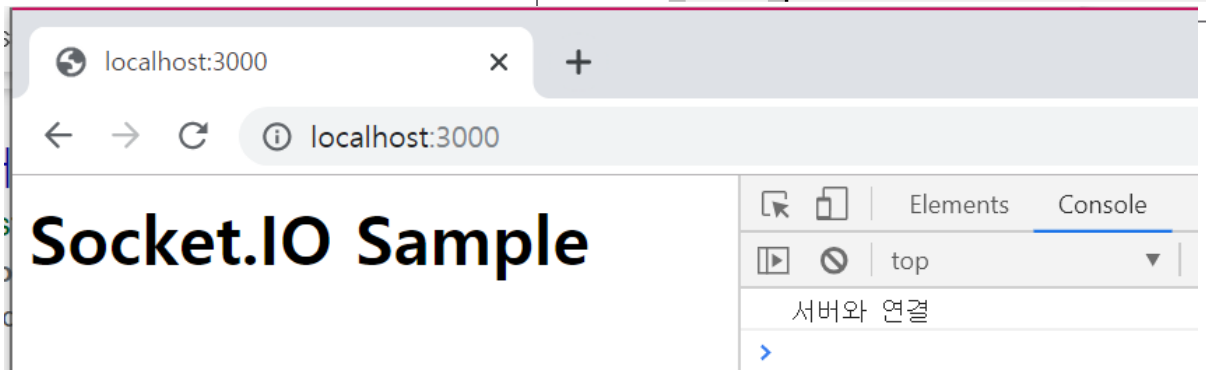
socket > JS server1.js > ...

```
1  var express = require('express');
2  var http = require('http');
3  var app = express();
4
5  var server = http.createServer(app);
6  server.listen(3000);
7
8  app.get('/', function(req,res){
9      res.sendFile(__dirname + '/client1.html');
10 });
11
12 var io = require('socket.io')(server);
13 io.on('connect', function(socket){
14     console.log('클라이언트 접속');
15 });
```

클라이언트

socket > client1.html > html

```
1  <html>
2      <head>
3          <meta charset='UTF8'>
4          <script src="/socket.io/socket.io.js"></script>
5          <script>
6              var socket = io();
7              socket.on('connect', function(){
8                  console.log('서버와 연결');
9              })
10         </script>
11     </head>
12     <body>
13         <h1>Socket.IO Sample</h1>
14     </body>
15 </html>
```



소켓 서버

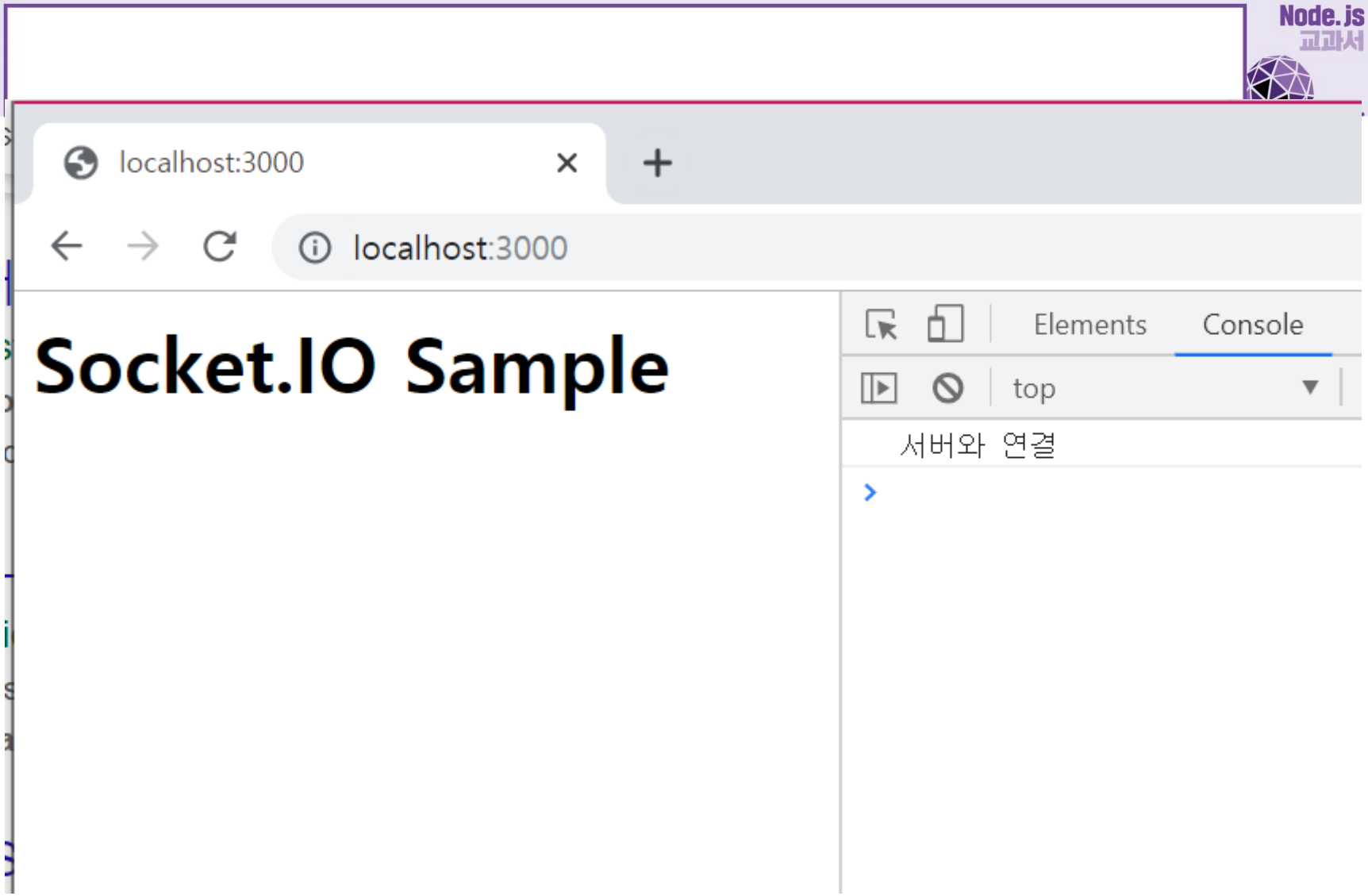
socket > JS server1.js > ...

```
1   var express = require('express');
2   var http = require('http');
3   var app = express();
4
5   var server = http.createServer(app);
6   server.listen(3000);
7
8   app.get('/', function(req,res){
9       res.sendFile(__dirname + '/client1.html');
10  });
11
12  var io = require('socket.io')(server);
13  io.on('connect', function(socket){
14      console.log('클라이언트 접속');
15  });
```

소켓 클라이언트

socket > client1.html > html

```
1  <html>
2      <head>
3          <meta charset='UTF8'>
4          <script src="/socket.io/socket.io.js"></script>
5          <script>
6              var socket = io();
7              socket.on('connect', function(){
8                  console.log('서버와 연결');
9              })
10         </script>
11     </head>
12     <body>
13         <h1>Socket.IO Sample</h1>
14     </body>
15 </html>
```


localhost:3000
localhost:3000
Socket.IO Sample
Elements
Console
top
서버와 연결

Socket.IO 를 이용한 실시간 웹 서비스 - 데이터 교환

2. 데이터 교환

- 데이터 교환
  - 메시지 주고 받기 - 이벤트 기반
    - 메시지 이벤트 정의
  - 메시지 전송
    - 이벤트 발생 : socket.emit()
    - socket.emit('EVENT', data);
  - 메시지 수신
    - 이벤트 리스너 등록 : socket.on()
    - socket.on('EVENT', function(data) {});

2. 데이터 교환

- 이벤트를 이용해서 데이터 주고 받기

보내기 / 받기

```
socket.emit('hello', {message:'Welcome'});
socket.on('howAreYou',
  function(data){
    var msg = data['message'];
  }
);
```

보내기 / 받기

```
socket.on('hello',
  function(data){
    var msg = data['message'];
  }
);
socket.emit('howAreYou', {message:'Welcome'});
```

2. 데이터 교환

- 이벤트로 메시지 주고 받기
  - 서버에 이벤트 등록 - 클라이언트에서 이벤트 발생
  - 클라이언트 이벤트 등록 - 서버에서 이벤트 발생

서버에서의 이벤트 발생

- 소켓 하나에 이벤트 발생
  socket.emit('Direct Event', [데이터]);
- 연결된 모든 소켓에 이벤트 발생
  socket.io.emit('Broadcast Event', [데이터]); // io.emit 로도 가능

Socket.IO 웹 서비스 실습 2

Node.js 교과서

서버

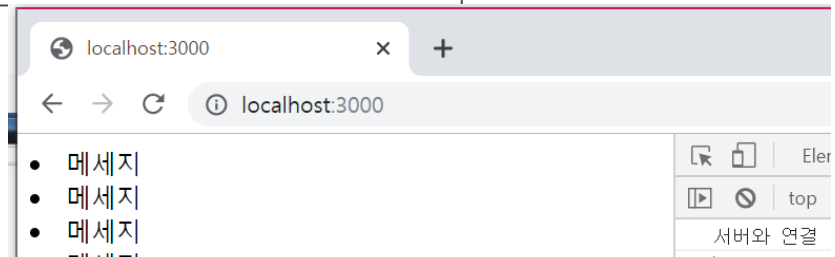
socket > JS server.js > ...

```
1  var express = require('express');
2  var http = require('http');
3  var app = express();
4
5  var server = http.createServer(app);
6  server.listen(3000);
7
8  app.get('/', function(req,res){
9      res.sendFile(__dirname + '/client.html');
10 });
11
12 var io = require('socket.io')(server);
13 io.on('connect', function(socket){
14     console.log('클라이언트 접속');
15
16     socket.on('disconnect', function(){
17         console.log('클라이언트 접속 종료');
18     });
19     setInterval(function(){
20         socket.emit('message','메세지');
21     },3000);
22 });
```

클라이언트

socket > client.html > html

```
1  <html>
2      <head>
3          <meta charset='UTF8'>
4          <script src="/socket.io/socket.io.js"></script>
5          <script>
6              var socket = io();
7              socket.on('connect', function(){
8                  console.log('서버와 연결');
9              })
10         </script>
11     </head>
12     <body>
13         <h1>Socket.IO Sample</h1>
14         <ul>test</ul>
15         <script>
16             socket.on('message', function(msg){
17                 document.writeln('<li>');
18                 document.writeln(msg);
19                 document.writeln('</li>');
20             })
21         </script>
22     </body>
23 </html>
```



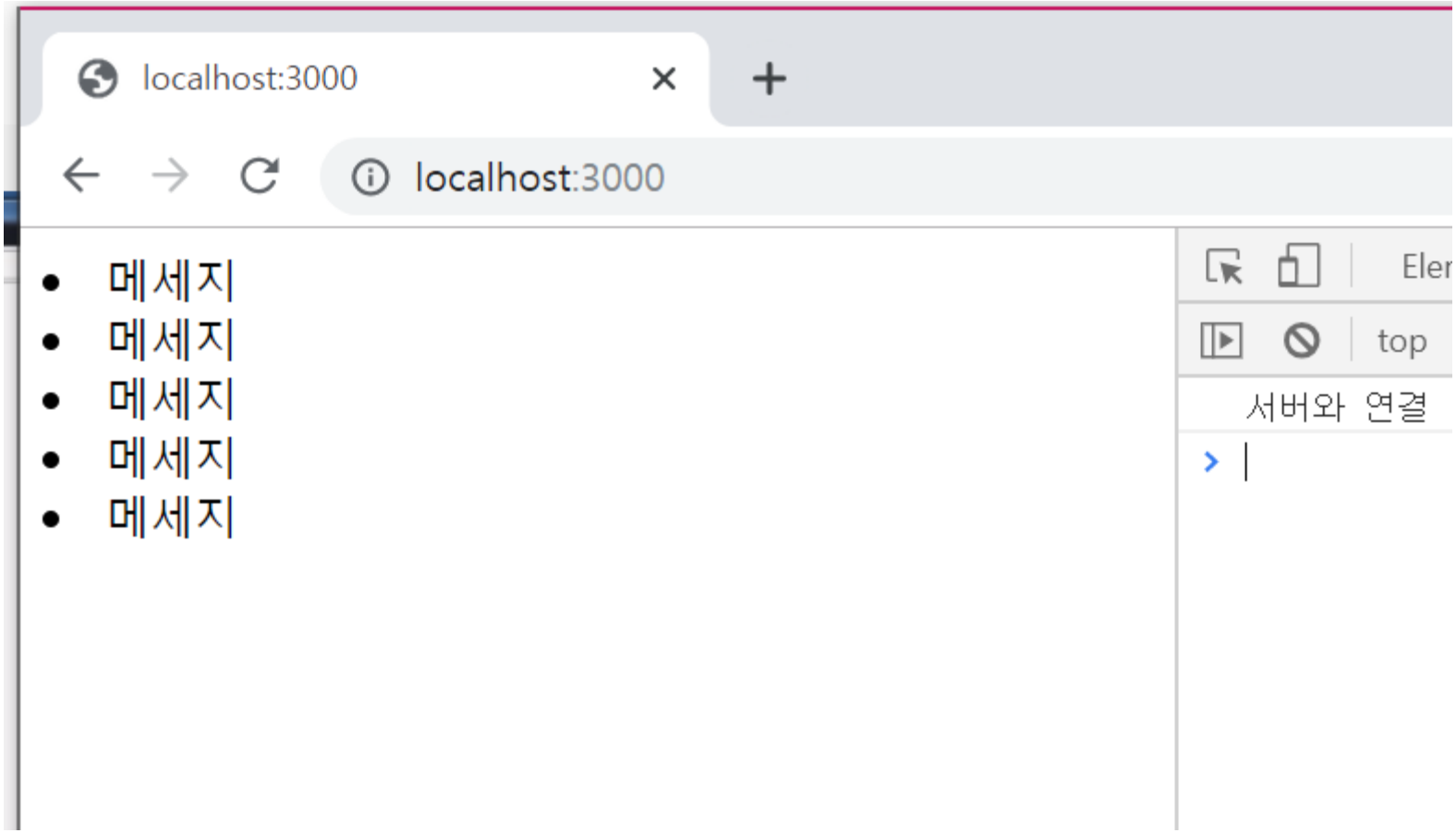
socket > JS server.js > ...

```
1  var express = require('express');
2  var http = require('http');
3  var app = express();
4
5  var server = http.createServer(app);
6  server.listen(3000);
7
8  app.get('/', function(req,res){
9      res.sendFile(__dirname + '/client.html');
10 });
11
12 var io = require('socket.io')(server);
13 io.on('connect', function(socket){
14     console.log('클라이언트 접속');
15
16     socket.on('disconnect', function(){
17         console.log('클라이언트 접속 종료');
18     });
19     setInterval(function(){
20         socket.emit('message','메세지');
21     },3000);
22 });
```

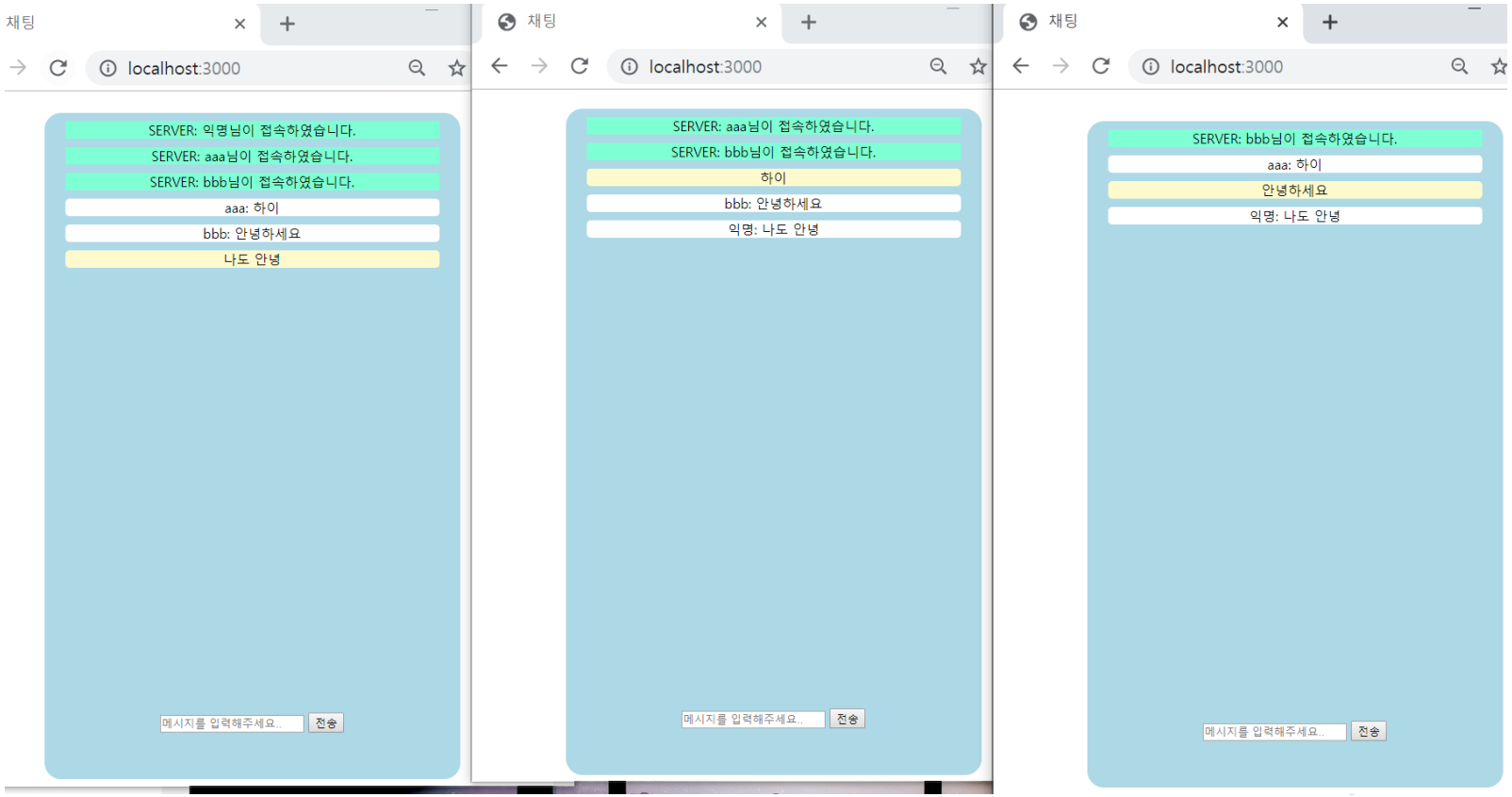
socket > client.html > html

```
1  <html>
2      <head>
3          <meta charset='UTF8'>
4          <script src="/socket.io/socket.io.js"></script>
5          <script>
6              var socket = io();
7              socket.on('connect', function(){
8                  console.log('서버와 연결');
9              })
10         </script>
11     </head>
12     <body>
13         <h1>Socket.IO Sample</h1>
14         <ul>test</ul>
15         <script>
16             socket.on('message', function(msg){
17                 document.writeln('<li>');
18                 document.writeln(msg);
19                 document.writeln('</li>');
20             })
21         </script>
22     </body>
23 </html>
```

Socket.IO 웹 서비스 실습 2



실시간 채팅 서비스 만들기



작업 폴더 및 모듈 설치

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

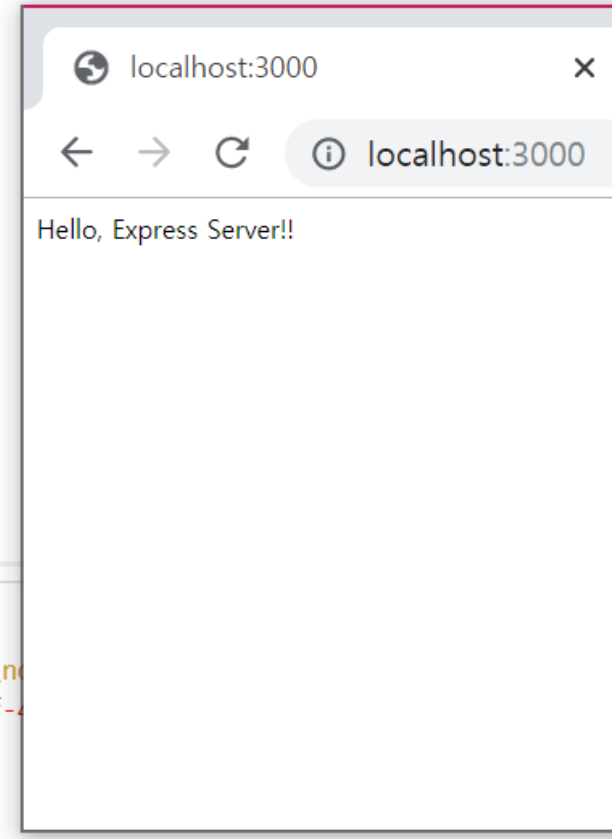
```
PS C:\> cd   chat
PS C:\ chat :> npm install express -s
+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in 1.849s
found 0 vulnerabilities

PS C:\ chat :> npm install socket.io -s
+ socket.io@2.3.0
added 45 packages from 28 contributors and audited 363 packages in 1.898s
found 0 vulnerabilities
```

1. 기초 채팅서버

» 폴더에 app.js 파일 생성

node_chat > JS app_old.js > ...

```
1   const express = require('express')
2   const socket = require('socket.io')
3   const http = require('http')
4   const app = express()
5   const server = http.createServer(app)
6   const io = socket(server)
7
8   app.get('/', function(request, response) {
9     console.log('유저가 / 으로 접속하였습니다!')
10    response.send('Hello, Express Server!!')
11  });
12
13  server.listen(3000, function() {
14    console.log('서버 실행 중..')
15  });
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

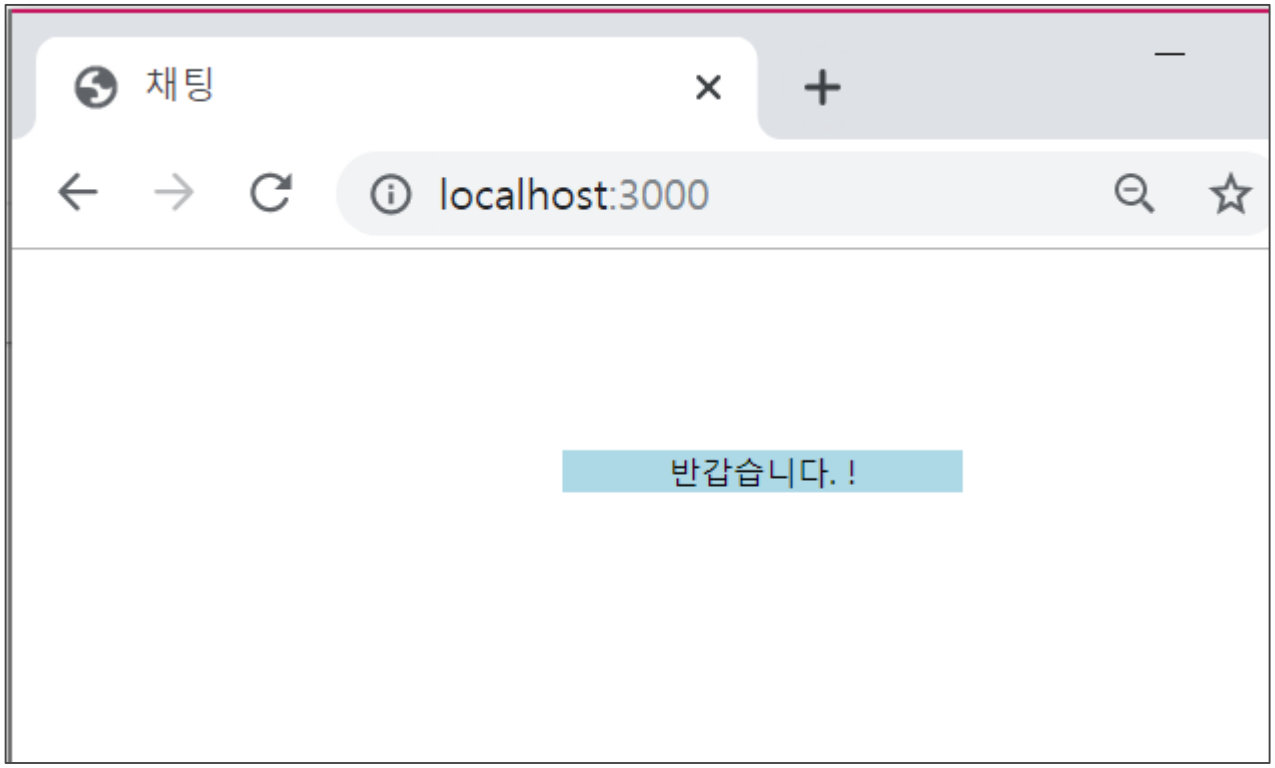
```
C:\Program Files\nodejs\node.exe --inspect-brk=46175 ..\n
Debugger listening on ws://127.0.0.1:46175/430946ab-f5ff-
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
서버 실행 중..
유저가 / 으로 접속하였습니다!
```

app.js 작성


```
1   const express = require('express')
2   const socket = require('socket.io')
3   const http = require('http')
4   const app = express()
5   const server = http.createServer(app)
6   const io = socket(server)
7
8   app.get('/', function(request, response) {
9     console.log('유저가 / 으로 접속하였습니다!')
10    response.send('Hello, Express Server!!')
11  });
12
13  server.listen(3000, function() {
14    console.log('서버 실행 중..')
15  });
```

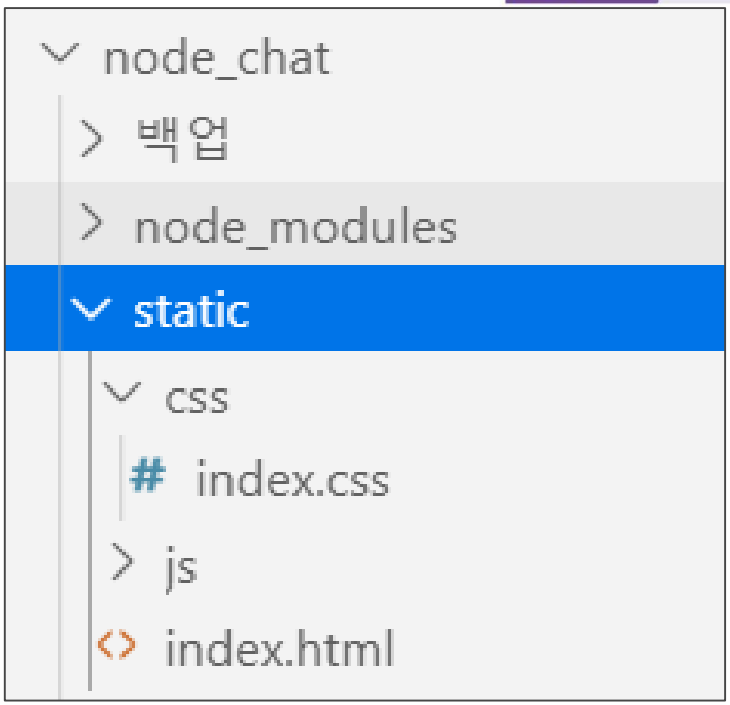
정적 파일 제공

- 1. 디렉토리 구조 잡기
- 2. HTML, CSS 예제 파일 생성
- 3. 서버 코드 수정



정적 파일 제공 : 폴더 및 파일 생성

- Chat폴더에 static 폴더 생성
- static폴더에 css폴더, js폴더 생성
- css폴더에 index.css 파일 생성
- static 폴더 안에는 **index.html** 파일을 생성

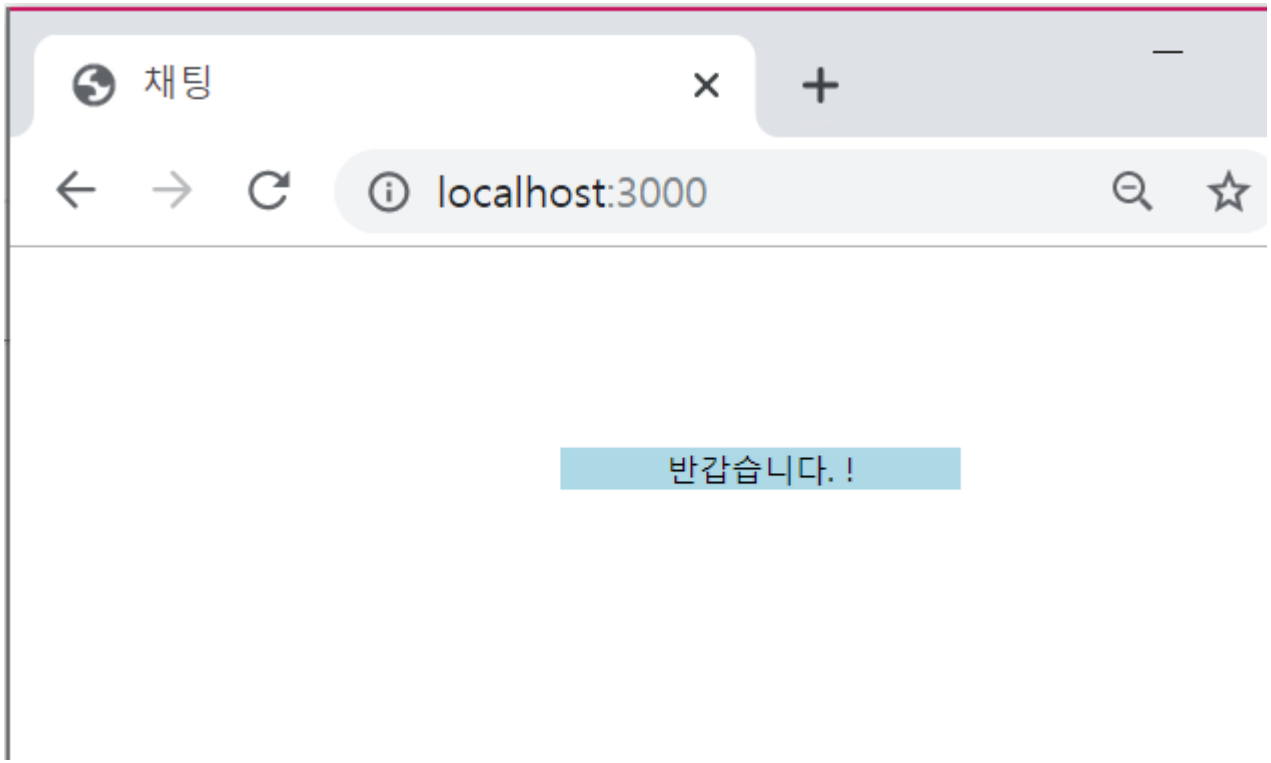


node_chat > static > css > # index.css > ...

```css
#main {
    margin: auto;
    margin-top: 100px;
    background-color: lightblue;
    text-align: center;
    width: 200px;
}
```

node_chat > static > index.html > ...

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>채팅</title>
    <link rel="stylesheet" href="/css/index.css">
  </head>
  <body>
    <div id="main">
      반갑습니다. !
    </div>
  </body>
</html>
```

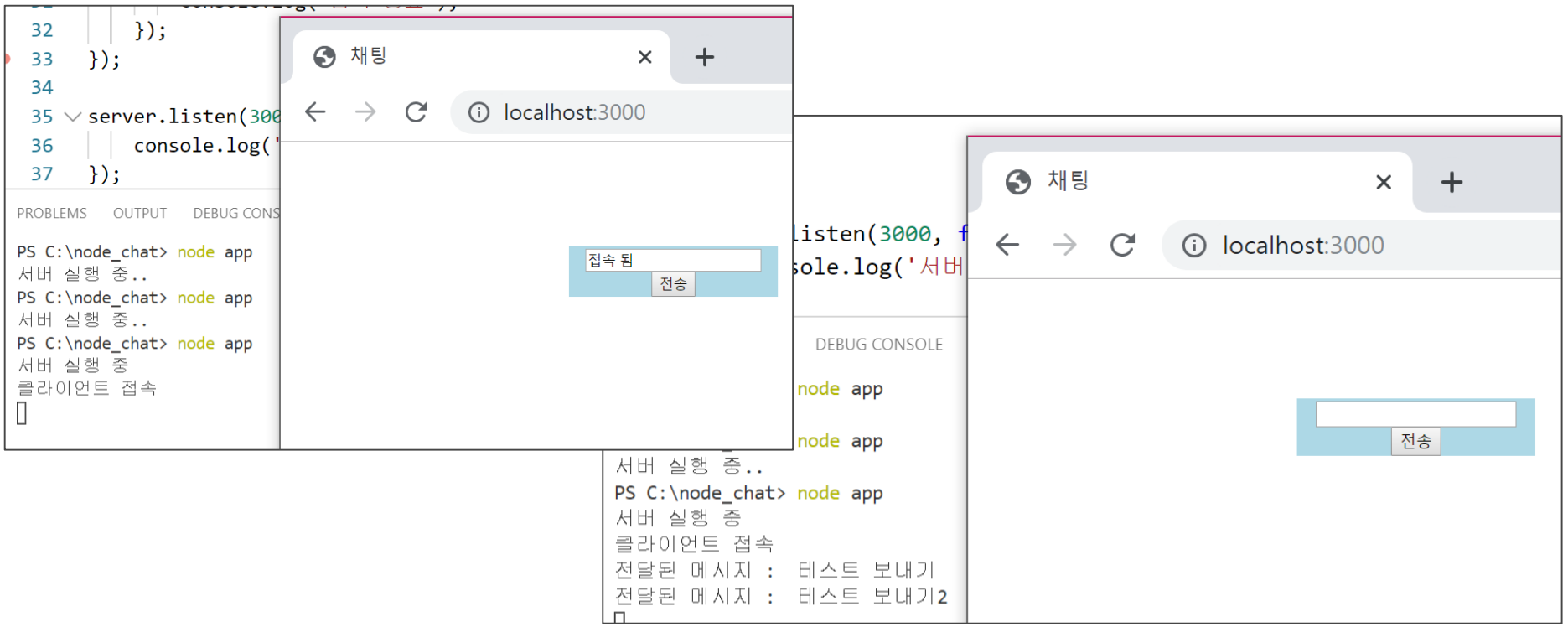

채팅
localhost:3000
반갑습니다. !

app.js 수정

```
1  const express = require('express')
2  const socket = require('socket.io')
3  const http = require('http')
4  const app = express()
5  const server = http.createServer(app)
6  const io = socket(server)
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24 server.listen(3000, function() {
25   console.log('서버 실행 중..')
26 });
```

실시간 채팅 서비스 만들기
- 2. 기초 채팅 서비스

- 1. 서버 코드 작성
- 2. HTML 수정
- 3. 클라이언트 자바스크립트 생성
- 4. 클라이언트 코드 작성
- 5. 테스트



app.js 수정

```
 9   app.use('/css', express.static('./static/css'))
10   app.use('/js',express.static('./static/js'))
11
12 > app.get('/', function(request,response){ ···
22   });
23
24   io.sockets.on('connection', function(socket){
25       console.log('클라이언트 접속');
26
27       socket.on('send', function(data){
28           console.log('전달된 메시지 : ', data.msg);
29       });
30       socket.on('disconnect', function(){
31           console.log('접속 종료');
32       });
33   });
34
35   server.listen(3000, function(){
36       console.log('서버 실행 중');
37   });
```

index.html 수정

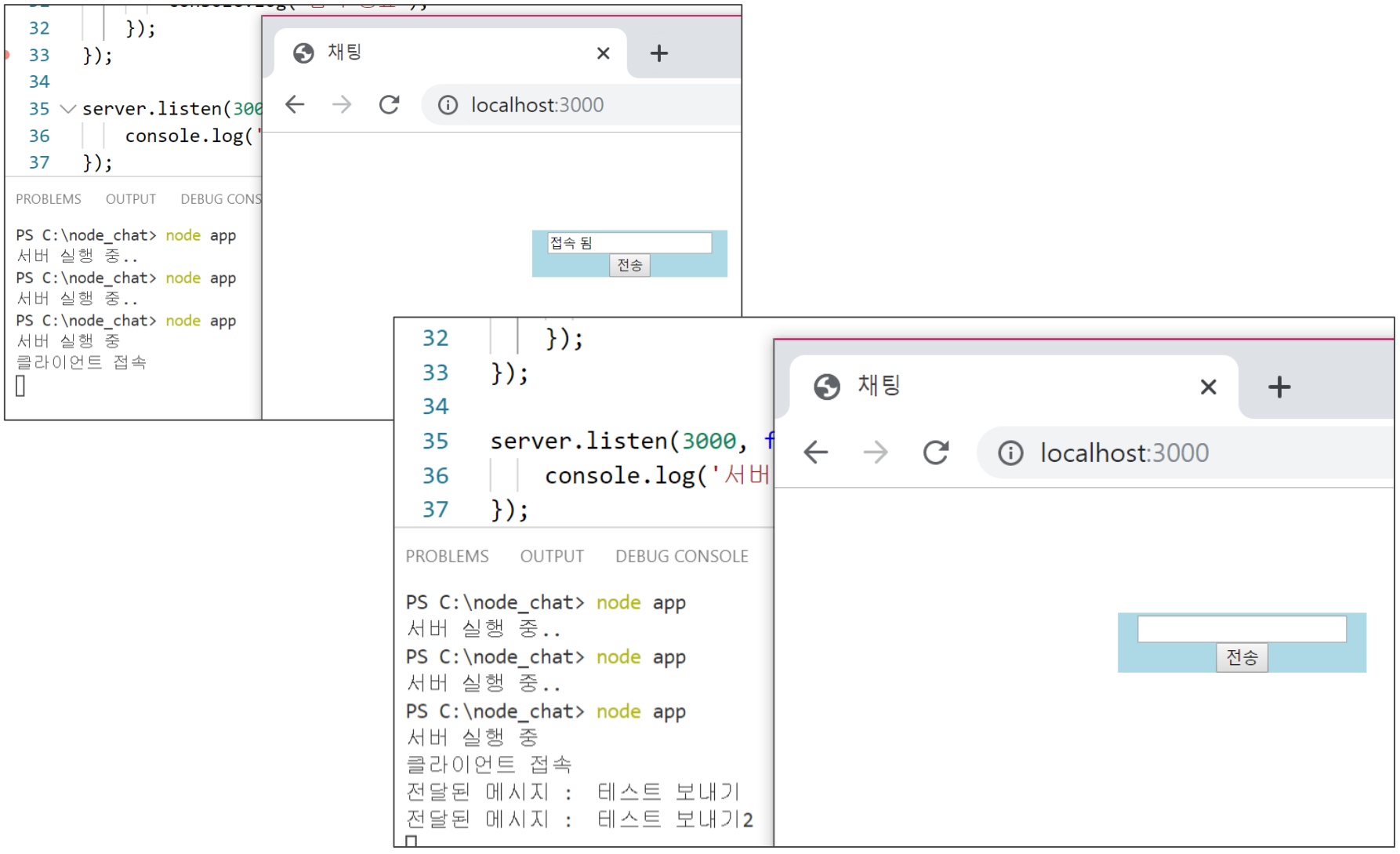
node_chat > static > index.html > ...

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8">
5       <title>채팅</title>
6       <link rel="stylesheet" href="/css/index.css">
7       <script src="/socket.io/socket.io.js"></script>
8       <script src="/js/index.js"></script>
9     </head>
10    <body>
11      <div id="main">
12        <input type="text" id="test">
13        <button onclick="send()">전송</button>
14      </div>
15    </body>
16  </html>
```

Static폴더>js폴더>index.js 작성

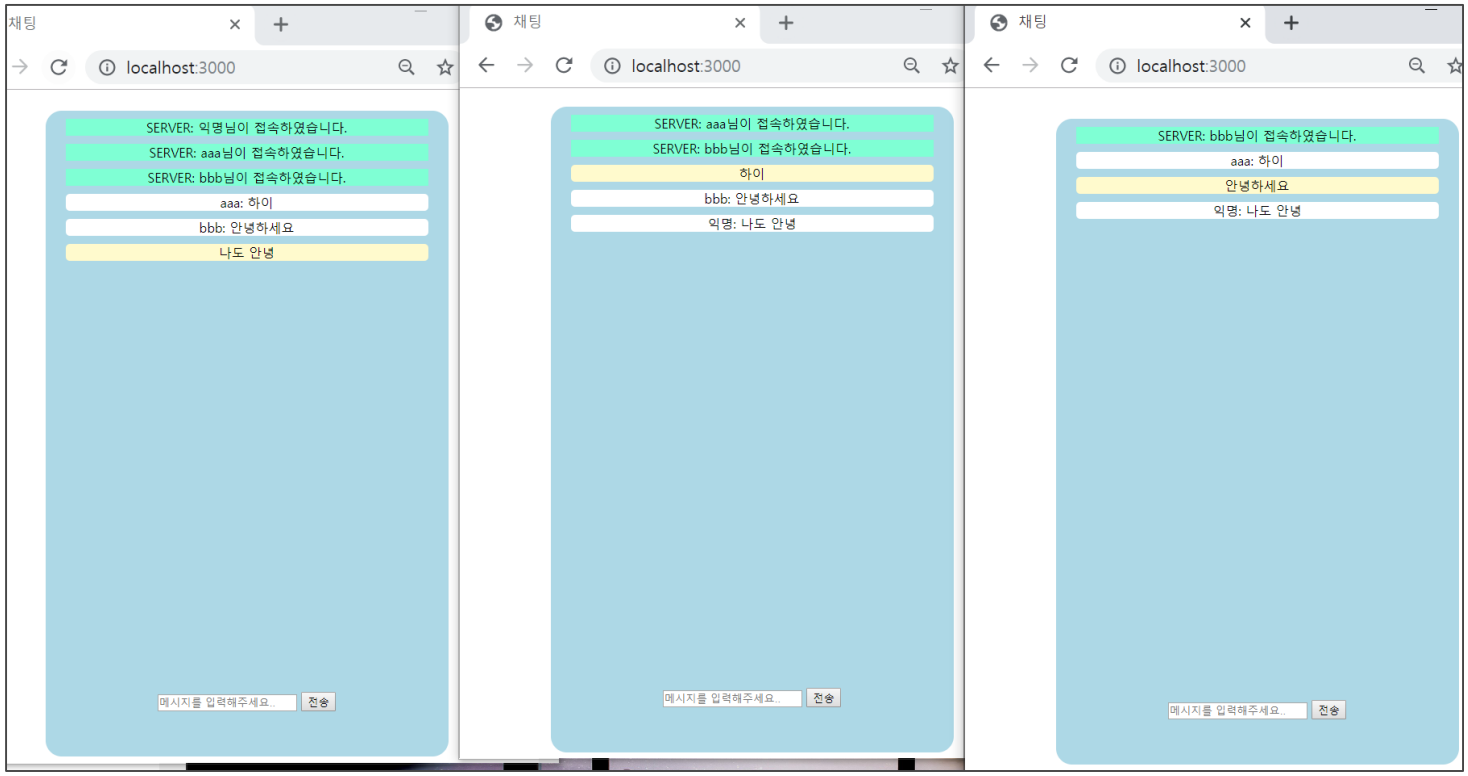
node_chat > static > js > JS index.js > ...

```
1   var socket = io()
2
3   socket.on('connect', function() {
4     var input = document.getElementById('test')
5     input.value = '접속 됨'
6   })
7
8   function send() {
9     // 입력되어있는 데이터 가져오기
10
11
12    // 가져왔으니 데이터 빈칸으로 변경
13
14
15    // 서버로 send 이벤트 전달 + 데이터와 함께
16
17  }
```

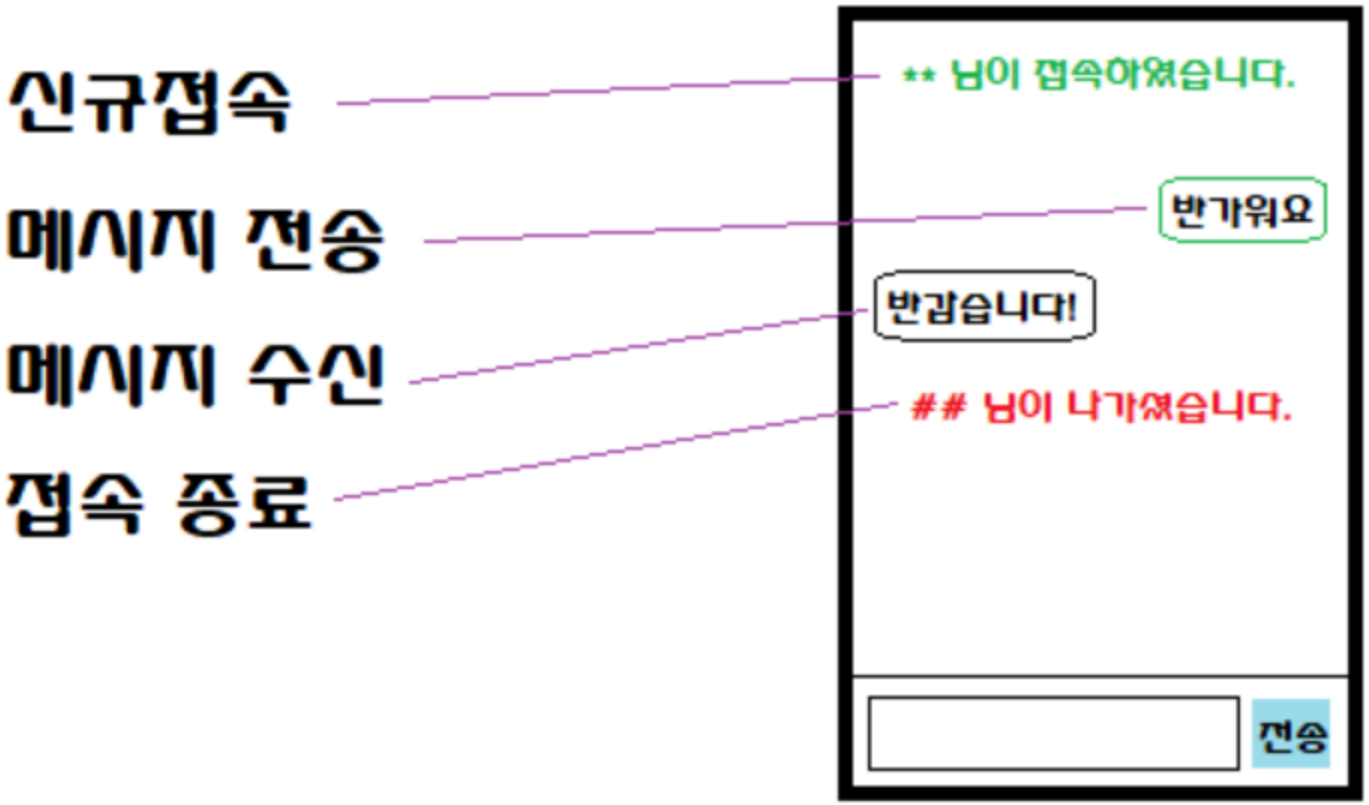

32 });
33 });
34
35 server.listen(300
36 console.log('
37 });
PROBLEMS OUTPUT DEBUG CONS
PS C:\node_chat> node app
서버 실행 중..
PS C:\node_chat> node app
서버 실행 중..
PS C:\node_chat> node app
서버 실행 중
클라이언트 접속
채팅
localhost:3000
접속 됨
전송
32 });
33 });
34
35 server.listen(3000, f
36 console.log('서버
37 });
PROBLEMS OUTPUT DEBUG CONSOLE
PS C:\node_chat> node app
서버 실행 중..
PS C:\node_chat> node app
서버 실행 중..
PS C:\node_chat> node app
서버 실행 중
클라이언트 접속
전달된 메시지 : 테스트 보내기
전달된 메시지 : 테스트 보내기2
채팅
localhost:3000
전송

실시간 채팅 서비스 만들기
3.채팅 기능 구현

- 1. 채팅 기능 설계
- 2. 서버 코드 작성
- 3. 클라이언트 코드 작성
- 4. 테스트

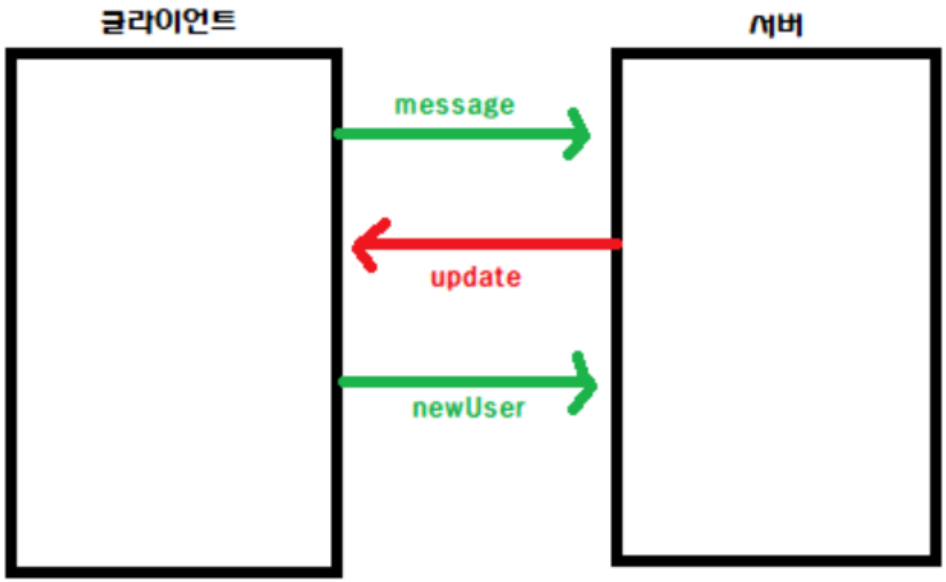


채팅 기능 설계



이벤트 명 정의

- message: 클라이언트가 서버로 메시지 전송
- update: 서버에서 받은 메시지 다른 클라이언트에게 전송(메시지 또는 정보)
- connectUser: 새로운 유저 접속을 서버에게 알림
- 접속 종료부분은 update로 통일하여 클라이언트에게 전달할 예정



메시지(알림)의 형식

- Type
  - message(기본 메시지)
  - connect(접속 알림)
  - disconnect(채팅 종료 알림) 3가지를 사용

type	메시지 유형(서버의 알림, 유저 메시지)
message	전달된 메시지 데이터
name	메시지를 전달한 유저 이름(또는 서버)

- app.js 수정: newUser 이벤트

- 클라이언트가 접속을 성공하면 클라이언트에서 newUser 이벤트를 발생
- 이벤트 발생과 함께 닉네임도 같이 서버로 전송하도록 해서 서버의 socket 안에 이름을 따로 저장해둡니다.
  - socket.name = name
  - 이 부분이 클라이언트로부터 받은 닉네임을 소켓에 저장시키는 코드입니다.
- 닉네임 정보를 받았으니 접속되어있는 다른 유저에게도 접속사실을 알려야합니다.
- io.sockets.emit('update' 데이터) 를 통해 다른 유저들에게도 알립니다.
- type은 connect로 지정하고 메시지를 전송

- app.js 수정: message 이벤트

- message 이벤트는 클라이언트에서 메시지를 입력하고 발생시키는 이벤트
  - 유저가 보내는 메시지는 모두 type이 message인 데이터로 위에서 정의
  - 타입과 메시지부분만 있으면 안되고 누가 보냈는지 알리기 위해

    data.name = socket.name 으로 소켓에 저장해두었던 이름을 데이터에 추가
- A, B, C 세 사람이 채팅을 하고 있다고 가정하고
  - A라는 사람이 메시지를 전송했으면 A 본인은 메시지를 받지 않고 B, C만 메시지를 수신
  - socket.broadcase.emit('이벤트명', 전달할데이터) 를 사용하면 본인을 제외한 나머지 유저에게 데이터를 전송
  - io.sockets.emit() = 모든 유저(본인 포함)
  - socket.broadcast.emit() = 본인을 제외한 나머지 모두

- app.js 수정


```
13 > app.get('/', function(request, response) {…
23   })
24
25   io.sockets.on('connection', function(socket) {
26     socket.on('newUser', function(name) {
27
28
29
30     })
31
32     socket.on('message', function(data) {
33
34
35
36     })
37
38     socket.on('disconnect', function() {
39
40
41
42     })
43   })
44
45   server.listen(3000, function() {
46     console.log('서버 실행 중..')
47   })
```

- index.js 수정

node_chat > static > js > JS index.js > ...

```
1   var socket = io()
2
3   socket.on('connect', function() {
4     var name = prompt('반갑습니다!', '')
5     if(!name) {
6       name = '익명'
7     }
8     socket.emit('newUser', name)
9   })
10
11  socket.on('update', function(data) {···
32  })
33
34  function send() {
35    var message = document.getElementById('test').value
36    document.getElementById('test').value = ''
37    var chat = document.getElementById('chat')
38    var msg = document.createElement('div')
39    var node = document.createTextNode(message)
40    msg.classList.add('me')
41    msg.appendChild(node)
42    chat.appendChild(msg)
43    socket.emit('message', {type: 'message', message: message})
44  }
```

- index.js 수정


```
11 socket.on('update', function(data) {
12   var chat = document.getElementById('chat')
13   var message = document.createElement('div')
14   var node = document.createTextNode(`${data.name}: ${data.message}`)
15   var className = ''
16
17   switch(data.type) {
18     case 'message':
19       className = 'other'
20       break
21     case 'connect':
22       className = 'connect'
23       break
24     case 'disconnect':
25       className = 'disconnect'
26       break
27   }
28
29   message.classList.add(className)
30   message.appendChild(node)
31   chat.appendChild(message)
32 })
```

index.html 수정

node_chat > static > index.html > ...

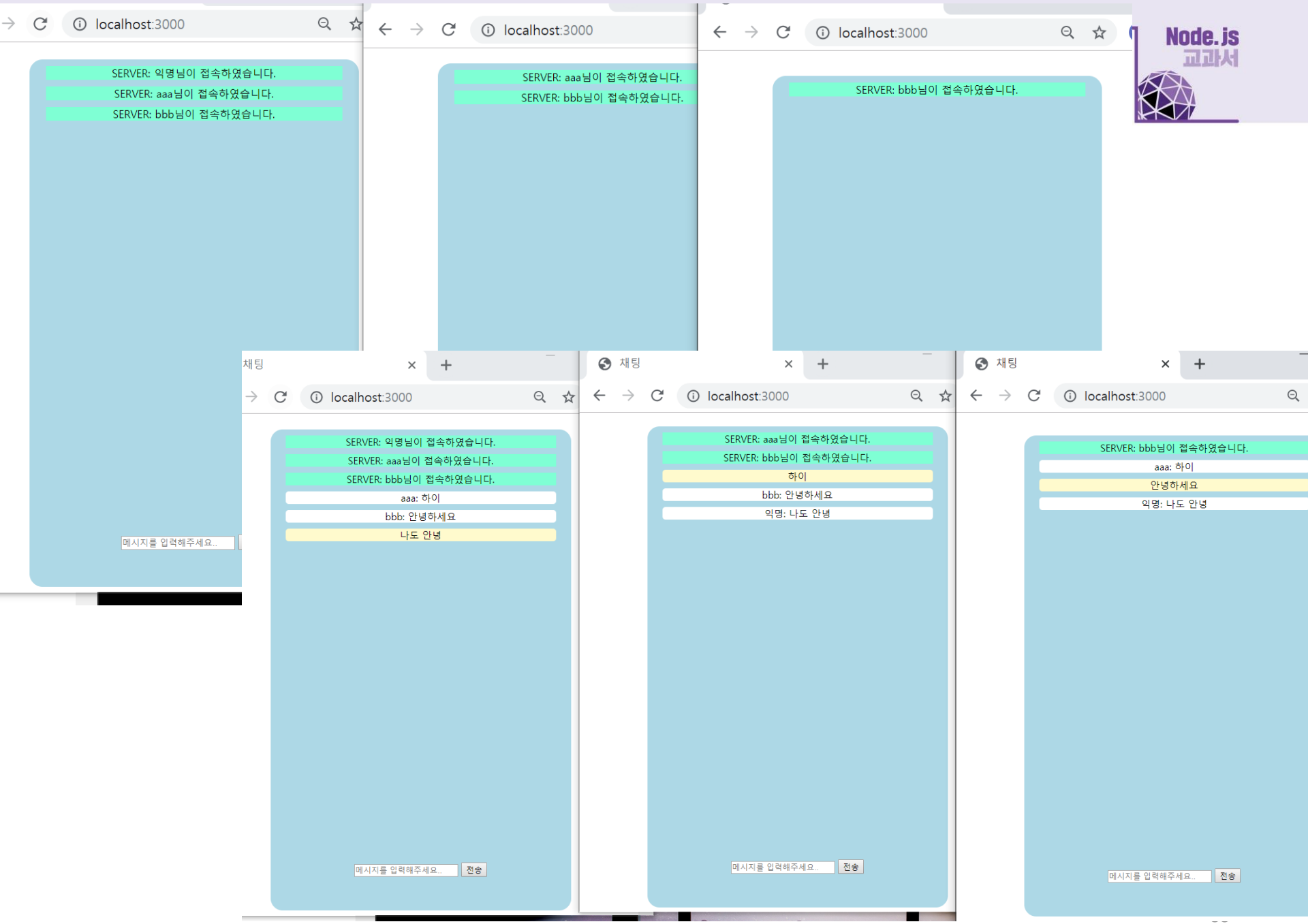
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf
    <title>채팅</title
    <link rel="stylesh
    <script src="/sock
    <script src="/js/i
  </head>
  <body>
    <div id="main">
      <input type="text" id="test">
      <button onclick="send()">전송</button>
    </div>
  </body>
</html>
```

node_chat > static > index.html > ...

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>채팅</title>
    <link rel="stylesheet" href="/css/index.css">
    <script src="/socket.io/socket.io.js"></script>
    <script src="/js/index.js"></script>
  </head>
  <body>
    <div id="main">
      <div id="chat">
        <!-- 채팅 메시지 영역 -->
      </div>
      <div>
        <input type="text" id="test" placeholder="메시지를 입력해주세요..">
        <button onclick="send()">전송</button>
      </div>
    </div>
  </body>
</html>
```

index.css 교체

- index.css 파일을 자료실에서 다운로드해서 교체


localhost:3000
SERVER: 익명님이 접속하였습니다.
SERVER: aaa님이 접속하였습니다.
SERVER: bbb님이 접속하였습니다.
localhost:3000
SERVER: aaa님이 접속하였습니다.
SERVER: bbb님이 접속하였습니다.
localhost:3000
SERVER: bbb님이 접속하였습니다.
Node.js
교과서
채팅
localhost:3000
SERVER: 익명님이 접속하였습니다.
SERVER: aaa님이 접속하였습니다.
SERVER: bbb님이 접속하였습니다.
aaa: 하이
bbb: 안녕하세요
나도 안녕
메시지를 입력해주세요..
전송
채팅
localhost:3000
SERVER: aaa님이 접속하였습니다.
SERVER: bbb님이 접속하였습니다.
하이
bbb: 안녕하세요
익명: 나도 안녕
메시지를 입력해주세요..
전송
채팅
localhost:3000
SERVER: bbb님이 접속하였습니다.
aaa: 하이
안녕하세요
익명: 나도 안녕
메시지를 입력해주세요..
전송