

# Web Socket

# WebSocket

## ❖ HTTP

- ✓ 1991년 발표된 HyperText Transfer Protocol인 HTTP는 Client-Server간 접속을 유지하지 않으며 Client-Server간 한 번에 한 방향으로만 통신이 가능한 half-duplex 방식
- ✓ 서로간에 주고받는 데이터의 양이 많아지면서 half-duplex로 인한 성능 저하는 피할 수 없으며 HTTP는 지나치게 많은 헤더 데이터를 가지고 있음
- ✓ Client(브라우저)가 요청을 보내지 않아도 Server가 데이터를 보내주는 기능의 구현에 있어서는 많은 고민이 있어왔지만 HTTP의 기본적 메커니즘 탓에 한계가 있음

## ❖ HTML5의 WebSocket

- ✓ WebSocket을 사용하면 더 이상 ActiveX를 사용하지 않고도 TCP/IP 소켓 통신을 구현할 수 있음
- ✓ 네트워크의 과부하를 줄이고 애플리케이션의 반응성을 높일 수 있음
- ✓ HTTP 헤더 크기 문제도 800byte에서 수 kbyte의 헤더 크기를 가지고 있는 HTTP와 달리 WebSocket은 수 byte 수준으로 압축이 가능
- ✓ HTTP가 적합치 않은 메세징, 트랜잭션 및 애플리케이션 특성 상 트래픽이 높고 지연시간이 낮은 환경에서 유용
- ✓ RMI(Remote Method Invocation), JMS(Java Messaging Service), XMPP(Extensible Messaging and Presence Protocol) 등이 그 예

# WebSocket

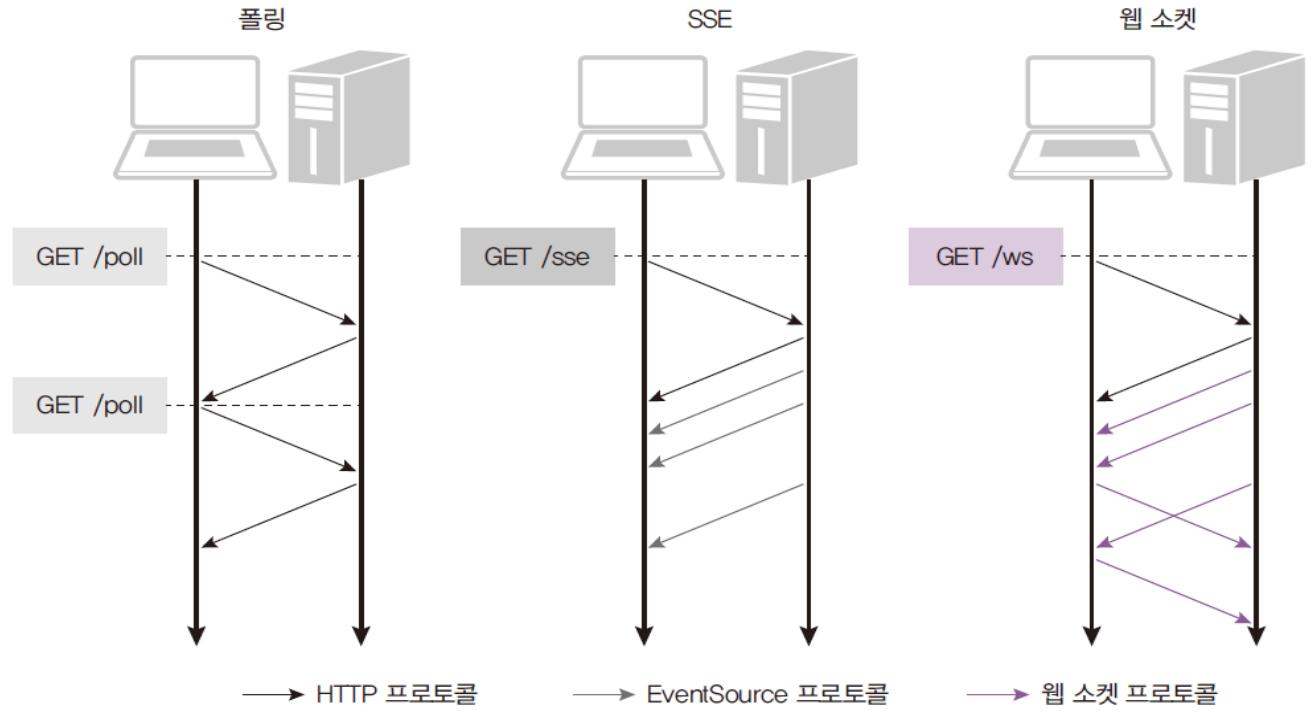
- ❖ 웹 소켓: 실시간 양방향 데이터 전송을 위한 기술
  - ✓ ws 프로토콜 사용 -> 브라우저가 지원해야 함
  - ✓ 최신 브라우저는 대부분 웹 소켓을 지원함
  - ✓ 노드는 ws나 Socket.IO같은 패키지를 통해 웹 소켓 사용 가능
- ❖ 웹 소켓 이전에는 폴링이라는 방식을 사용했음
  - ✓ HTTP가 클라이언트에서 서버로만 요청이 가기 때문에 주기적으로 서버에 요청을 보내 업데이트가 있는지 확인함
  - ✓ 웹 소켓은 연결도 한 번만 맺으면 되고, HTTP와 포트 공유도 가능하며, 성능도 매우 좋음



# WebSocket

## ❖ SSE(Server Sent Events)

- ✓ EventSource라는 객체를 사용
- ✓ 처음에 한 번만 연결하면 서버가 클라이언트에 지속적으로 데이터를 보내줌
- ✓ 클라이언트에서 서버로는 데이터를 보낼 수 없음

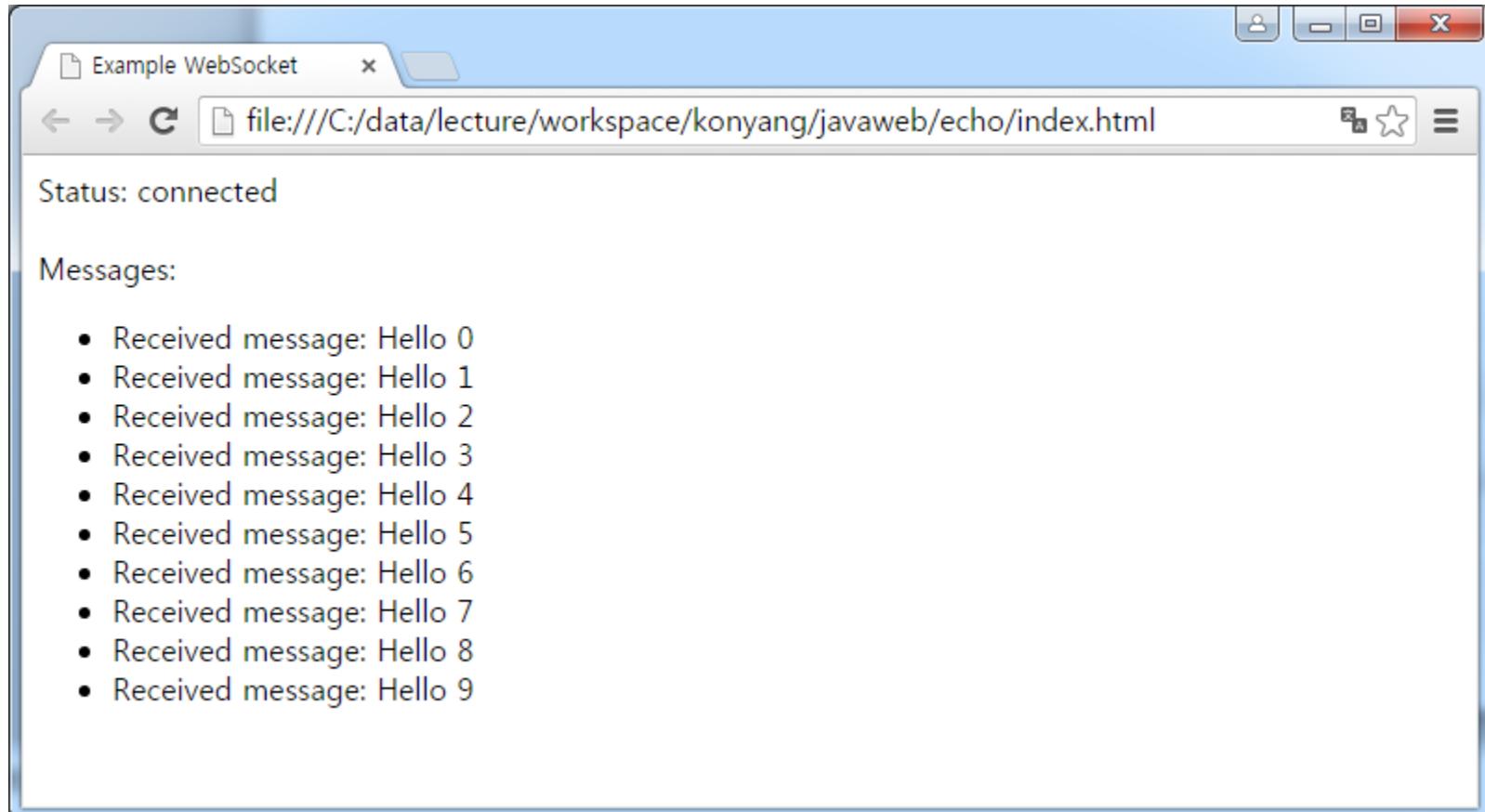


# WebSocket

## ❖ 웹 소켓을 사용하기 위한 패키지

- ✓ websocket 모듈: http 서버를 연결해야 함
- ✓ ws 모듈
- ✓ Socket.IO 모듈
  - 웹 소켓을 편리하게 사용하도록 해주는 모듈
  - IE9 와 같은 브라우저에서는 웹 소켓 대신 폴링 방식을 사용하여 전송
  - 클라이언트 측에서 웹 소켓 연결이 끊어지면 자동으로 재연결을 시도

# WebSocket – websocket 모듈



# WebSocket

- ❖ node.js 프로젝트 생성 – nodewebsocket
- ❖ package.json을 생성할 수 있도록 터미널에서 npm init 수행 – app.js 파일을 시작 파일로 설정
- ❖ websocket, express, morgan 패키지 설치
  - npm install websocket
- ❖ nodemon을 개발용으로 설치
  - npm install --save-dev nodemon

# WebSocket

## ❖ package.json 파일의 script 부분 수정

```
{  
  "name": "websokcetnode",  
  "version": "1.0.0",  
  "description": "웹소켓",  
  "main": "app.js",  
  "scripts": {  
    "start  },  
  "author": "itstudy",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.17.1",  
    "morgan": "^1.10.0",  
    "socket.io": "^2.3.0",  
    "websocket": "^1.0.32"  
  },  
  "devDependencies": {  
    "nodemon": "^2.0.4"  
  }  
}
```

# WebSocket

- ❖ **index.html 파일을 생성하고 작성**

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Example WebSocket</title>
</head>
<body>
```

# WebSocket

## ❖ index.html 파일을 생성하고 작성

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
<script type="text/javascript">
if ('WebSocket' in window) {
    var ws = new WebSocket('ws://127.0.0.1:8000', 'example-echo');
    ws.onopen = function () {
        $('#status').text('connected');
        for (var i = 0; i < 10; i++) {
            ws.send('Hello ' + i);
        }
    };
    ws.onmessage = function (evt) {
        $('#messages').append($('- ').text('Received message: ' + evt.data));
    };
    ws.onclose = function () {
        $('#status').text('connection is closed');
    };
}
else
    $('#status').text('WebSocket not supported.');
</script>

```

# WebSocket

- ❖ index.html 파일을 생성하고 작성

```
Status:<span id="status"></span><br /><br /> Messages:  
<ul id="messages"></ul>  
</body>  
</html>
```

# WebSocket

## ❖ app.js 파일을 만들고 작성한 후 실행

```
var WebSocketServer = require('websocket').server;
var http = require('http');
var fs = require('fs');

var server = http.createServer(function (req, res) {
    if(req.url == "/"){
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('Web Socket');
    }
    else if(req.url == "/index"){
        fs.readFile('index.html', function (error, data) {
            res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
            res.end(data);
        });
    }
});

server.listen(8000, function () {
    console.log('Server is listening on port 8000');
});
```

# WebSocket

## ❖ app.js 파일을 만들고 작성한 후 실행

```
wsServer = new WebSocketServer({
    httpServer: server,
    autoAcceptConnections: false
});

wsServer.on('request', function (request) {
    var connection = request.accept('example-echo', request.origin);
    connection.on('message', function (message) {
        if (message.type === 'utf8') {
            console.log('Received message: ' + message.utf8Data);
            connection.sendUTF(message.utf8Data);
        }else if (message.type === 'binary') {
            connection.sendBytes(message.binaryData);
        }
        connection.on('close', function (reasonCode, description) {
            console.log('Peer ' + connection.remoteAddress + ' disconnected.');
        });
    });
});
```

# WebSocket

❖ 브라우저에 <http://localhost:8000/index> 를 입력하고 확인

A screenshot of a web browser window. The address bar shows 'localhost:8000/index'. Below the address bar is a toolbar with icons for App, Bookmarks, iOS,업무, frontend, java, R, and 머신러닝. The main content area displays the text 'Status:connected' followed by 'Messages:' and a list of ten items, each starting with 'Received message: Hello' and a number from 0 to 9.

- Received message: Hello 0
- Received message: Hello 1
- Received message: Hello 2
- Received message: Hello 3
- Received message: Hello 4
- Received message: Hello 5
- Received message: Hello 6
- Received message: Hello 7
- Received message: Hello 8
- Received message: Hello 9

# WebSocket – ws 모듈

# WebSocket

## ❖ 패키지 추가 설치

```
npm install cookie-parser dotenv express express-session morgan nunjucks ws
```

# WebSocket

- ❖ routes 디렉토리를 생성하고 index.js 파일을 생성: /요청이 오면 처리할 내용을 작성

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

```
const router = express.Router();
```

```
router.get('/', (req, res) => {  
    res.render('index');  
});
```

```
module.exports = router;
```

# WebSocket

## ❖ socket.js 파일을 생성: 웹 소켓 로직을 작성

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

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

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

# WebSocket

## ❖ socket.js 파일을 생성: 웹 소켓 로직을 작성

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

# WebSocket

- ❖ .env 파일을 생성하고 작성

```
COOKIE_SECRET=websocket
```

# WebSocket

## ❖ app.js 파일을 생성하고 작성

```
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 app = express();
app.set('port', process.env.PORT || 8001);
app.set('view engine', 'html');
nunjucks.configure('views', {
  express: app,
  watch: true,
});
```

# WebSocket

## ❖ app.js 파일을 생성하고 작성

```
app.use(morgan('dev'));
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser(process.env.COOKIE_SECRET));

app.use(session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
}));
app.use('/', indexRouter);
```

# WebSocket

## ❖ app.js 파일을 생성하고 작성

```
app.use((req, res, next) => {
  const error = new Error(`.${req.method} ${req.url} 라우터가 없습니다.`);
  error.status = 404;
  next(error);
});

app.use((err, req, res, next) => {
  res.locals.message = err.message;
  res.locals.error = process.env.NODE_ENV !== 'production' ? err : {};
  res.status(err.status || 500);
  res.render('error');
});

const server = app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기중');
});

webSocket(server);
```

# WebSocket

- ❖ views 디렉토리를 만들고 index.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>웹 소켓</title>
</head>
<body>
<div>F12를 눌러 console 탭과 network 탭을 확인하세요.</div>
<script>
const webSocket = new WebSocket("ws://localhost:8001");
webSocket.onopen = function () {
    console.log('서버와 웹소켓 연결 성공!');
};
webSocket.onmessage = function (event) {
    console.log(event.data);
    webSocket.send('클라이언트에서 서버로 답장을 보냅니다');
};
</script>
</body>
</html>
```

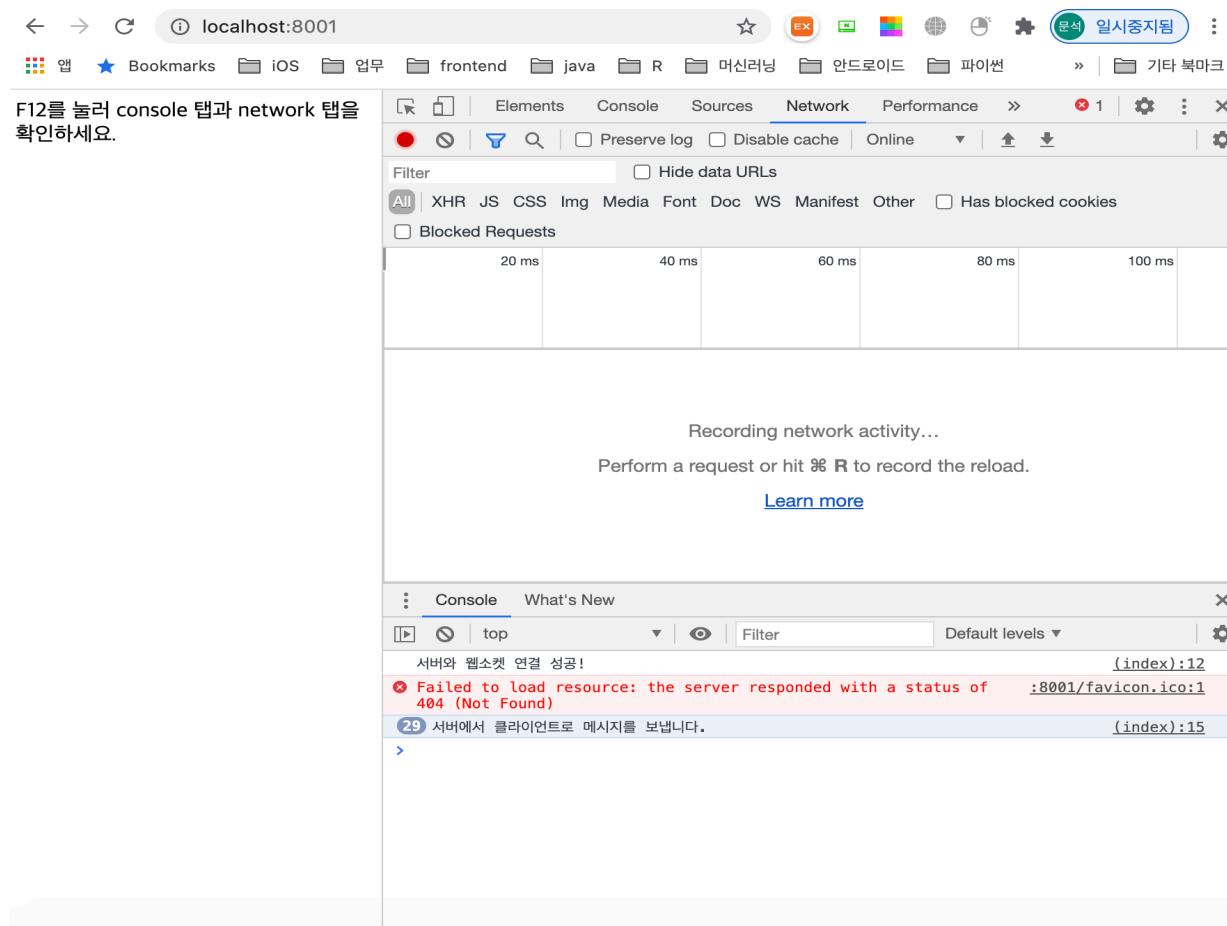
# WebSocket

- ❖ **views 디렉토리에 error.html 파일을 생성하고 작성**

```
<h1>{{message}}</h1>
<h2>{{error.status}}</h2>
<pre>{{error.stack}}</pre>
```

# WebSocket

❖ 브라우저에 localhost:8001을 입력하고 실행해서 네트워크 탭 확인



# WebSocket - socket.io 모듈

## ❖ ws 패키지 대신 Socket.IO 연결

- ✓ Socket.IO 패키지를 불러와 익스프레스 서버와 연결. 두 번째 인수는 클라이언트와 연결할 수 있는 경로(/socket.io)
- ✓ connection 이벤트는 서버와 연결되었을 때 호출, 콜백으로 소켓 객체(socket) 제공
- ✓ socket.request로 요청 객체에 접근 가능, socket.id로 소켓 고유 아이디 확인 가능
- ✓ disconnect 이벤트는 연결 종료 시 호출, error는 에러 발생 시 호출
- ✓ reply는 사용자가 직접 만들 이벤트로 클라이언트에서 reply 이벤트 발생 시 서버에 전달됨
- ✓ socket.emit으로 메시지 전달. 첫 번째 인수는 이벤트 명, 두 번째 인수가 메시지

### 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);
  });
}
```

# WebSocket - socket.io 모듈

- ❖ socket.io 패키지 설치

```
npm install socket.io
```

# WebSocket - socket.io 모듈

## ❖ socket.js 수정

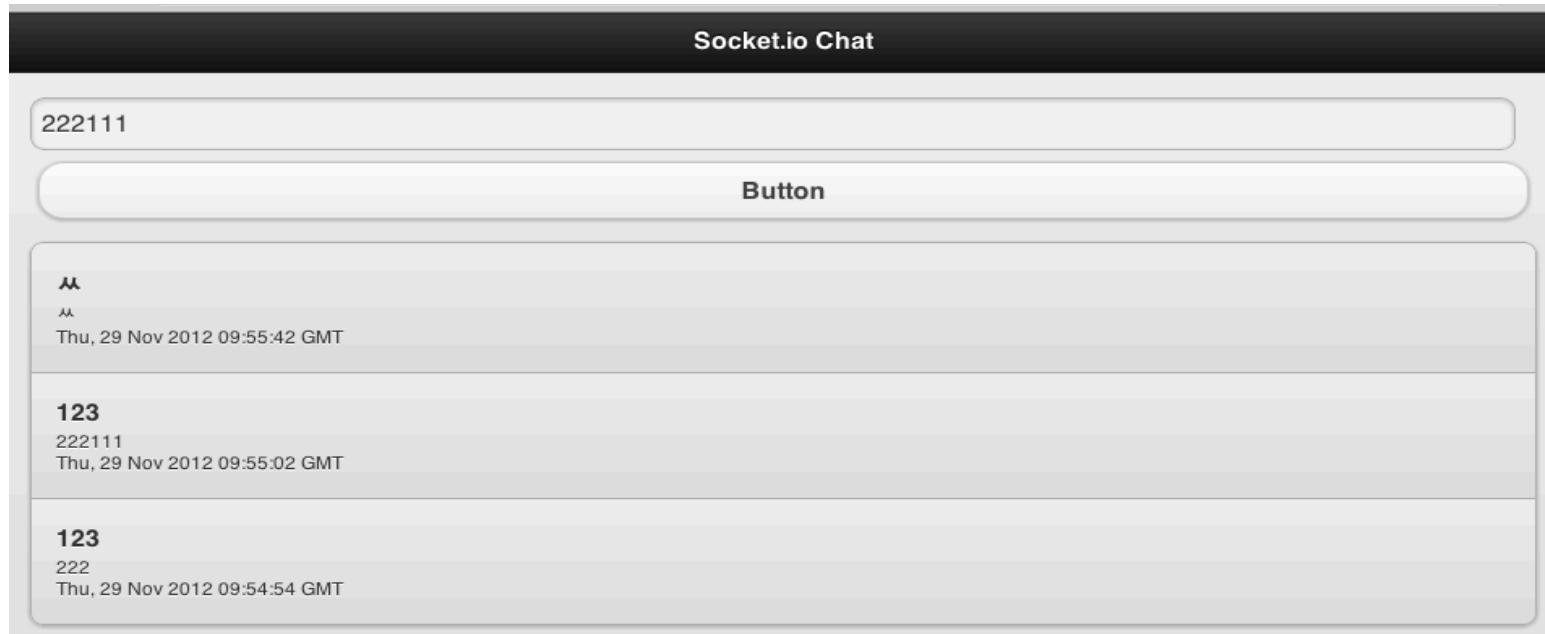
```
const SocketIO = require('socket.io');
module.exports = (server) => {
  const io = SocketIO(server, { path: '/socket.io' });
  io.on('connection', (socket) => { // 웹소켓 연결 시
    const req = socket.request;
    const ip = req.headers['x-forwarded-for'] || req.connection.remoteAddress;
    console.log('새로운 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
      console.log(data);
    });
    socket.interval = setInterval(() => { // 3초마다 클라이언트로 메시지 전송
      socket.emit('news', '안녕하세요 Socket.IO');
    }, 3000);
  });
};
```

# WebSocket - socket.io 모듈

## ❖ index.html 수정

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>웹 소켓 - SocketIO 이용</title>
</head>
<body>
<div>F12를 눌러 console 탭과 network 탭을 확인하세요.</div>
<script src="/socket.io/socket.io.js"></script>
<script>
  const socket = io.connect('http://localhost:8001', {
    path: '/socket.io',
    transports: ['websocket'],
  });
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('reply', 'Hello Node.JS');
  });
</script>
</body>
</html>
```

# WebSocket - socket.io 모듈



# 채팅

## ❖ socket.js 파일에 이벤트 처리 코드 추가

```
// message 이벤트
socket.on('message', function (data) {
    // 클라이언트의 message 이벤트를 발생시킵니다.
    io.sockets.emit('message', data);
});
```

# 채팅

## ❖ index.html 파일 수정

```
<!DOCTYPE html>
<html>
<head>
    <title>Mobile Chat</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.css"
    />
    <script src="http://code.jquery.com/jquery-1.8.2.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.js"></script>
    <script src="/socket.io/socket.io.js"></script>
```

# 채팅

## ❖ index.html 파일 수정

```
<script>
    // HTML 문서가 모두 준비되면
    $(document).ready(function () {
        // 변수를 선언합니다.
        var socket = io.connect('http://localhost:8001');
        // 이벤트를 연결합니다.
        socket.on('message', function (data) {
            // 추가할 문자열을 만듭니다.
            var output = '';
            output += '<li>';
            output += ' <h3>' + data.name + '</h3>';
            output += ' <p>' + data.message + '</p>';
            output += ' <p>' + data.date + '</p>';
            output += '</li>';
            // 문서 객체를 추가합니다.
            $(output).prependTo('#content');
            $('#content').listview('refresh');
        });
    });

```

# 채팅

## ❖ index.html 파일 수정

```
// 버튼을 클릭할 때
$('button').click(function () {
    socket.emit('message', {
        name: $('#name').val(),
        message: $('#message').val(),
        date: new Date().toUTCString()
    });
    $('#message').val('')
});
});
</script>
</head>
```

# 채팅

## ❖ index.html 파일 수정

```
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Socket.io Chat</h1>
    </div>
    <div data-role="content">
      <h3>Nick Name</h3>
      <input id="name" />
      <a data-role="button" href="#chatpage">Start Chat</a>
    </div>
  </div>
```

# 채팅

## ❖ index.html 파일 수정

```
<div data-role="page" id="chatpage">
    <div data-role="header">
        <h1>Socket.io Chat</h1>
    </div>
    <div data-role="content">
        <input id="message" />
        <button>전송</button>
        <ul id="content" data-role="listview" data-inset="true">
            </ul>
    </div>
</div>
</body>
</html>
```

# 전자칠판

# 1.프로젝트 구성

- ❖ public 디렉토리 생성

# HTML 설정

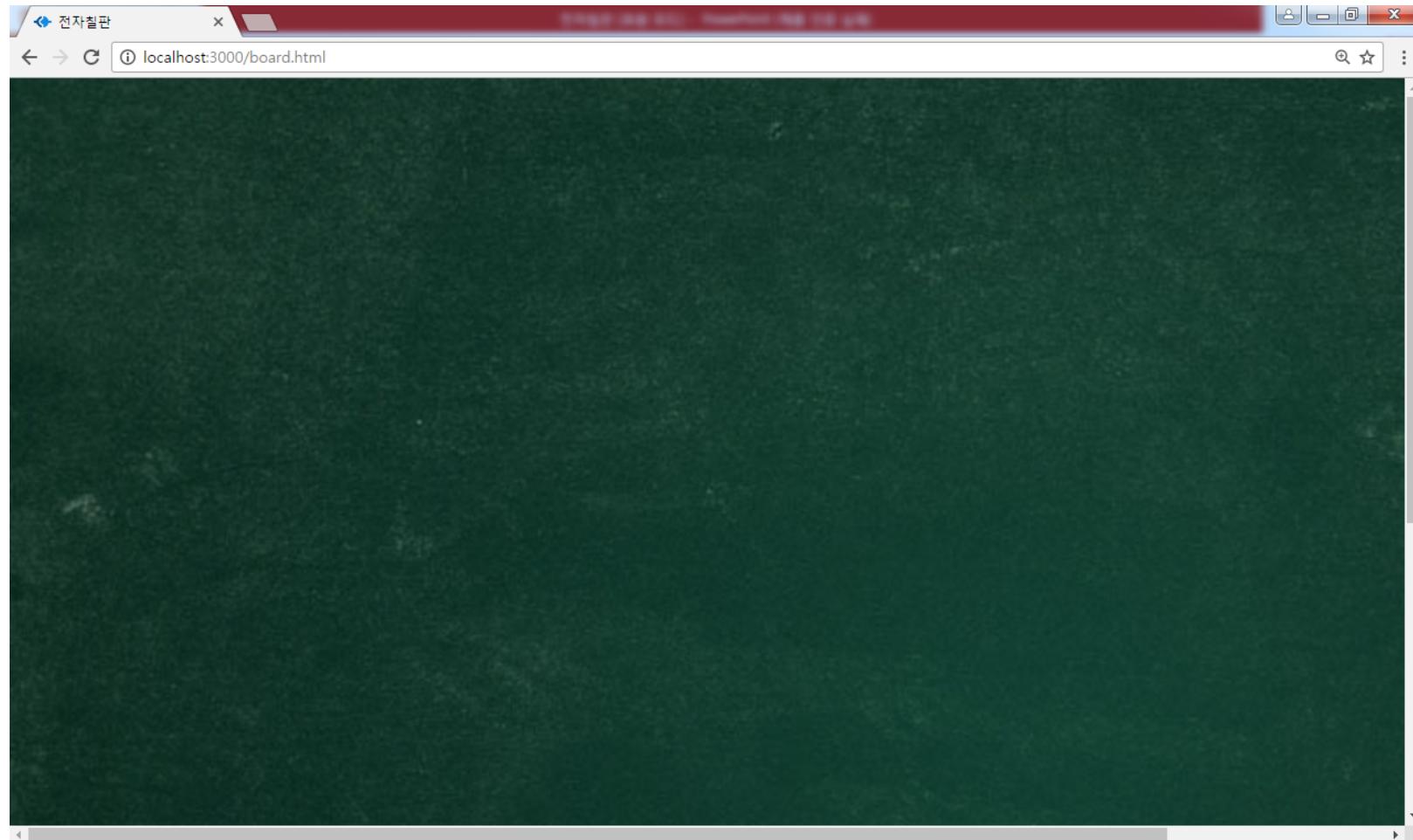
## ❖ index.html 파일 수정(스타일시트와 자바스크립트 링크 추가)

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>전자칠판</title>
<script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
<script src="js/board.js"></script>
<link rel="stylesheet" type="text/css" href="stylesheet/style.css">
</head>

<body>

</body>
</html>
```

# 캔버스



# 캔버스

- ❖ index.html 파일의 body 안에 canvas 추가

```
<canvas id = 'cv' width='860px;' height='645px;'></canvas>
```

- ❖ public 디렉토리 아래 images 디렉토리를 생성하고 blackboard.jpg 파일을 추가

- ❖ public 디렉토리 아래 stylesheet 디렉토리를 생성하고 style.css 파일을 생성하고 스타일시트 추가

```
body{  
    margin:0px;  
}  
  
#cv{  
    width:860px;  
    height:645px;  
    background-image:url('../images/blackboard.jpg');  
}
```

# 캔버스

- ❖ **public 디렉토리 안에 js 디렉토리를 만들고 board.js 파일을 생성한 후 작성**

- ✓ 사각형 그리기

```
$(function(){  
    //jQuery 이용하여 canvas element 객체 얻기  
    var ctx = $('#cv').get(0).getContext('2d');  
    ctx.fillStyle = "#FF0000";  
    ctx.fillRect(0,0,200,200);  
});
```

# 캔버스

## ✓ 원 그리기

```
$(function(){
    //jQuery 이용하여 canvas element 객체 얻기
    var ctx = $('#cv').get(0).getContext('2d');
    ctx.strokeStyle = "#FF0000";
    ctx.lineWidth=5;
    ctx.beginPath();
    ctx.arc(200,200, 100, 0,2*Math.PI);
    ctx.stroke();
});
```

# 캔버스

## ✓ 선 그리기

```
$(function(){
    //jQuery 이용하여 canvas element 객체 얻기
    var ctx = $('#cv').get(0).getContext('2d');
    ctx.strokeStyle = "#FF0000";
    ctx.lineWidth=5;
    ctx.beginPath();
    ctx.moveTo(100,100);
    ctx.lineTo(200,200);
    ctx.stroke();
});
```

# 캔버스

## ✓ 마우스 이벤트와 결합한 선 그리기

```
$(function(){
    //jQuery 이용하여 canvas element 객체 얻기
    var ctx = $('#cv').get(0).getContext('2d');
    ctx.strokeStyle = "white";
    ctx.lineWidth=5;
    ctx.beginPath();
    var drawing = false;
    $('#cv').bind('mousedown', function(e){
        drawing=true;
        ctx.moveTo(e.pageX, e.pageY);
    });
    $('#cv').bind('mousemove', function(e){
        if(drawing){
            ctx.lineTo(e.pageX, e.pageY);
            ctx.stroke();
        }
    });
    $('#cv').bind('mouseup', function(e){
        drawing=false;
    });
});
```

## 2.캔버스

### ❖ board.js 파일의 내용 수정

```
var ctx;  
$(document).ready(function() {  
    // jQuery 이용하여 canvas element 객체 얻기  
    ctx = $('#cv').get(0).getContext('2d');  
  
    // jQuery bind 이용하여 canvas에 마우스 시작,이동,끝 이벤트 핸들러 등록  
    $('#cv').bind('mousedown', draw.start);  
    $('#cv').bind('mousemove', draw.move);  
    $('#cv').bind('mouseup', draw.end);  
  
    // 기본 모양 색상 설정  
    shape.setShape();  
});
```

## 2.캔버스

```
var shape = {  
    // 기본 색상, 두께 설정  
    color : 'white',  
    width : 3,  
    // 모양 변경 메서드  
    setShape : function(color, width) {  
        if (color != null)  
            this.color = color;  
        if (width != null)  
            this.width = width;  
        ctx.strokeStyle = this.color;  
        ctx.lineWidth = this.width;  
    }  
}
```

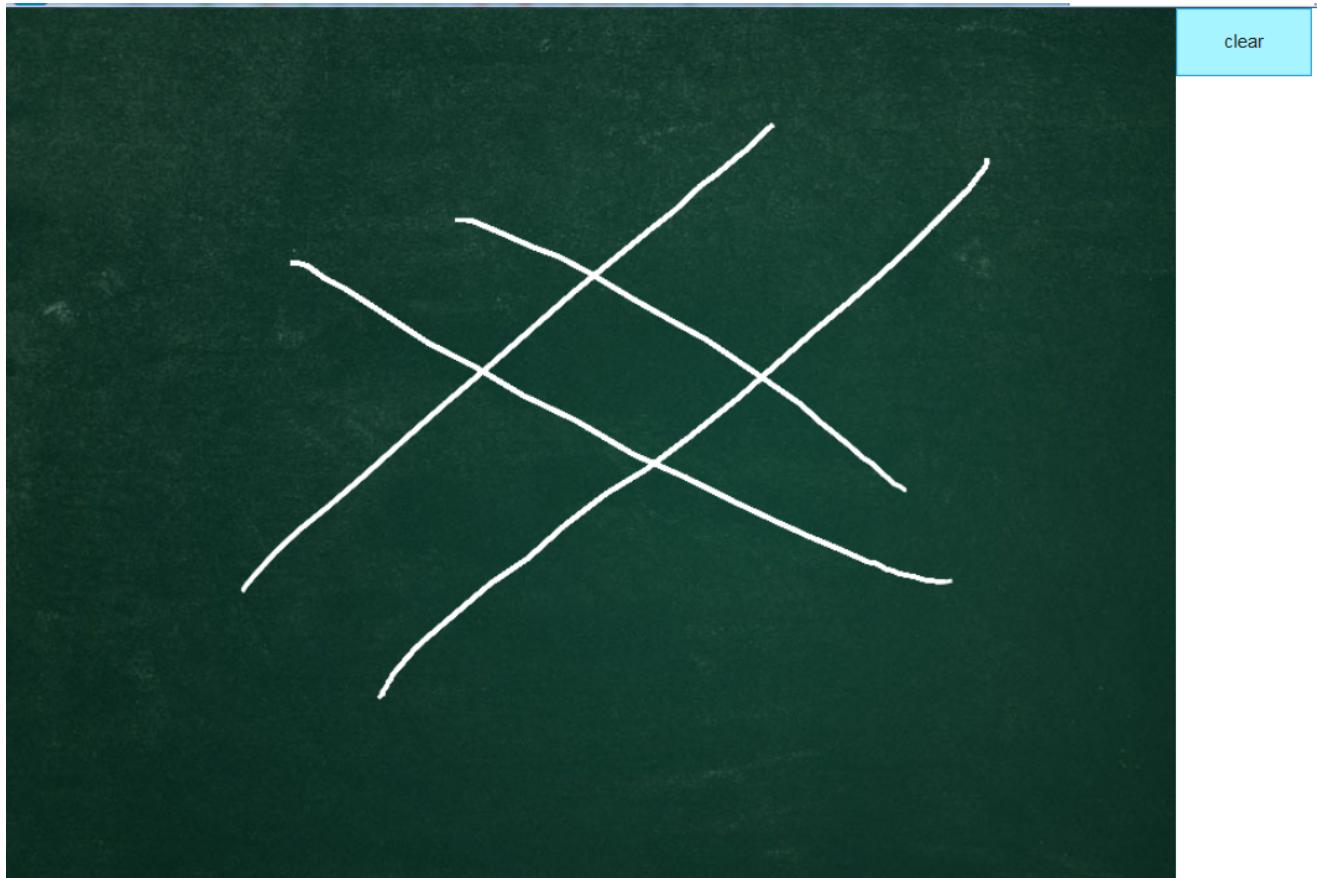
## 2.캔버스

```
// 그리기 관련
var draw = {
    drawing : null,
    start : function(e) {
        ctx.beginPath();
        ctx.moveTo(e.pageX, e.pageY);
        this.drawing = true;
    },
    move : function(e) {
        if (this.drawing) {
            ctx.lineTo(e.pageX, e.pageY);
            ctx.stroke();
        }
    },
}
```

## 2.캔버스

```
end : function(e) {  
    this.drawing = false;  
},  
}  
}
```

# 메뉴



# 메뉴

❖index.html 파일에 메뉴 영역을 추가하고 버튼 추가

```
<body>
  <canvas id='cv' width='860px;' height='645px;'></canvas>
  <div class="menu">
    <button id="clear">clear</button>
  </div>
</body>
```

# 메뉴

❖ style.css 파일에 메뉴 영역과 button의 스타일을 설정하고 cv는 수정

```
#cv{  
    width:860px;  
    height:645px;  
    background-image:url('../images/blackboard.jpg');  
    float:left;  
}  
  
.menu{  
    float:left;  
}  
  
button{  
    width:100px;  
    height:50px;  
}
```

# 메뉴

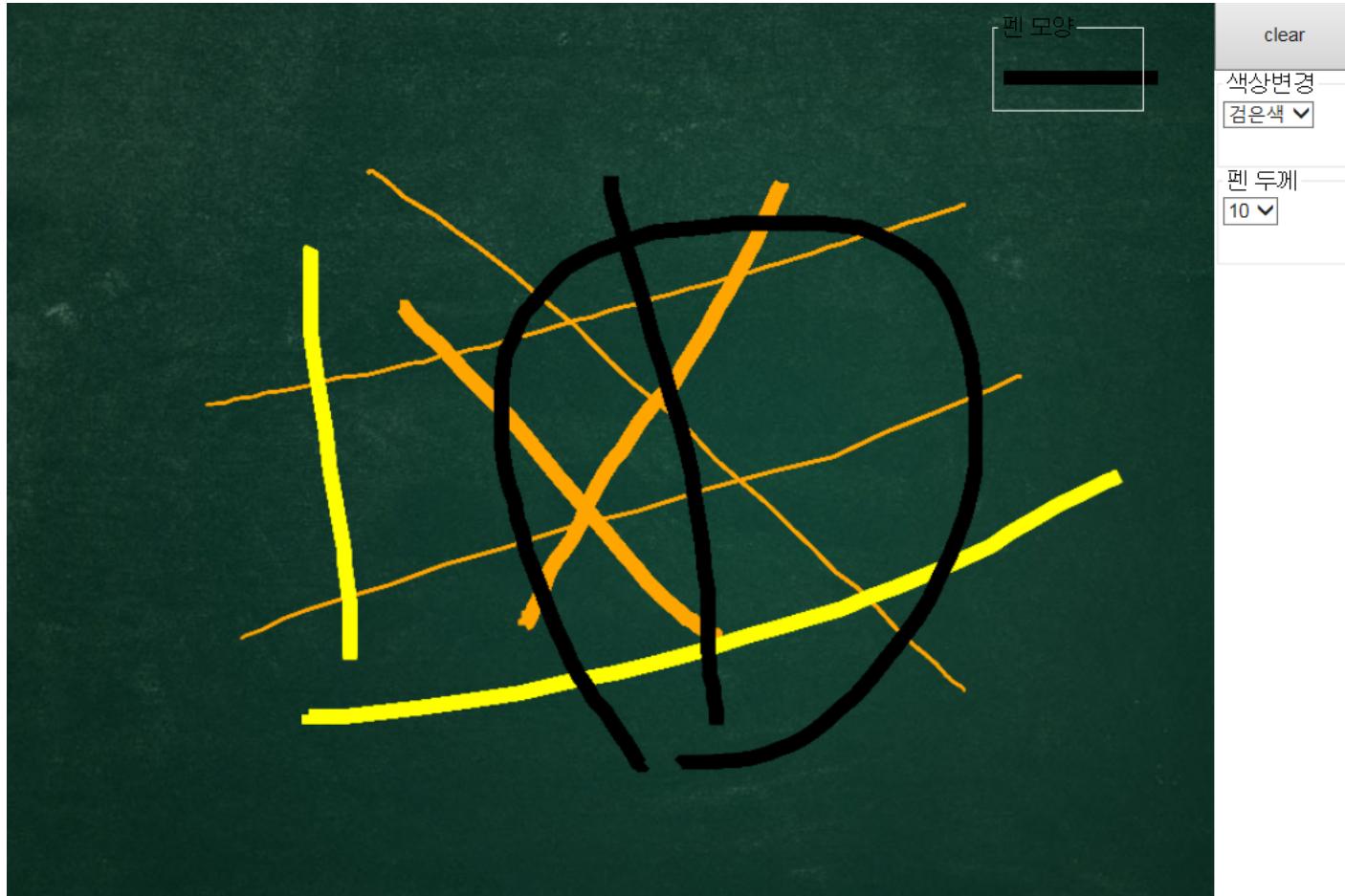
- ❖ board.js 파일의 draw 객체에 삭제 함수 추가

```
clear : function(){  
    ctx.clearRect(0,0,cv.width, cv.height);  
}
```

- ❖ board.js 파일의 document.ready 함수에 버튼의 이벤트 핸들러 지정

```
$('#clear').bind('click', draw.clear);
```

# 메뉴



# 메뉴

❖index.html 파일에 메뉴 영역에 색상변경, 펜 두께, 펜 모양 메뉴 추가

```
<div class="menu">
    <button id="clear">clear</button>
    <fieldset>
        <legend>색상변경</legend>
        <select id='pen_color'>
            </select>
    </fieldset>
    <fieldset>
        <legend>펜 두께</legend>
        <select id='pen_width'>
            </select>
    </fieldset>
    <fieldset id='pen_shape'>
        <legend>펜 모양</legend>
    </fieldset>
</div>
```

# 메뉴

## ❖ style.css 파일의 body 영역의 스타일에 추가

```
body{  
    margin:0px;  
    -webkit-user-select:none;  
}
```

## ❖ style.css 파일에 새로 추가한 영역 스타일을 설정

```
#cv_pen{  
    width:100px;  
    height:50px;  
    float:left;  
    background-image:url('images/blackboard.jpg');  
}
```

# 메뉴

```
fieldset{  
    width:100px;  
    height:60px;  
    float:left:left;  
}
```

```
#pen_shape{  
    position:absolute;  
    top:10px;  
    left:700px;  
    color:white;  
}
```

# 메뉴

- ❖ board.js 파일의 document.ready 함수에 2개 select의 value 설정 코드 추가

```
//색상 배열  
var color_map =  
[  
    {'value':'white','name':'하얀색'},  
    {'value':'red','name':'빨간색'},  
    {'value':'orange','name':'주황색'},  
    {'value':'yellow','name':'노란색'},  
    {'value':'blue','name':'파랑색'},  
    {'value':'black','name':'검은색'}  
];
```

# 메뉴

```
//색상 선택 select 설정
for(var key in color_map){
    $('#pen_color').append('<option value=' + color_map[key].value + '>' +
color_map[key].name + '</option>');
}
```

```
//두께 선택 select 설정
for(var i = 2 ; i < 15 ; i++){
    $('#pen_width').append('<option value=' + i + '>' + i + '</option>');
}
```

# 메뉴

## ❖ board.js 파일의 shape 변경

```
var shape = {  
    // 기본 색상, 두께 설정  
    color : 'white',  
    width : 3,  
  
    change : function() {  
        var color = $('#pen_color option:selected').val();  
        var width = $('#pen_width option:selected').val();  
        shape.setShape(color, width);  
    },
```

# 메뉴

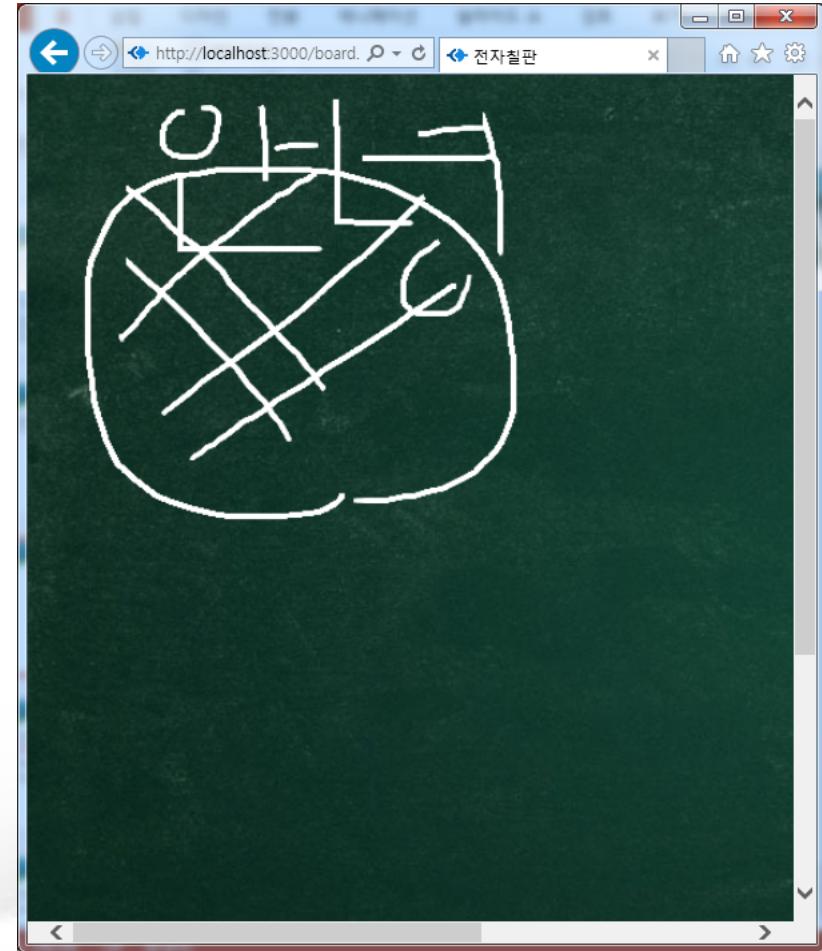
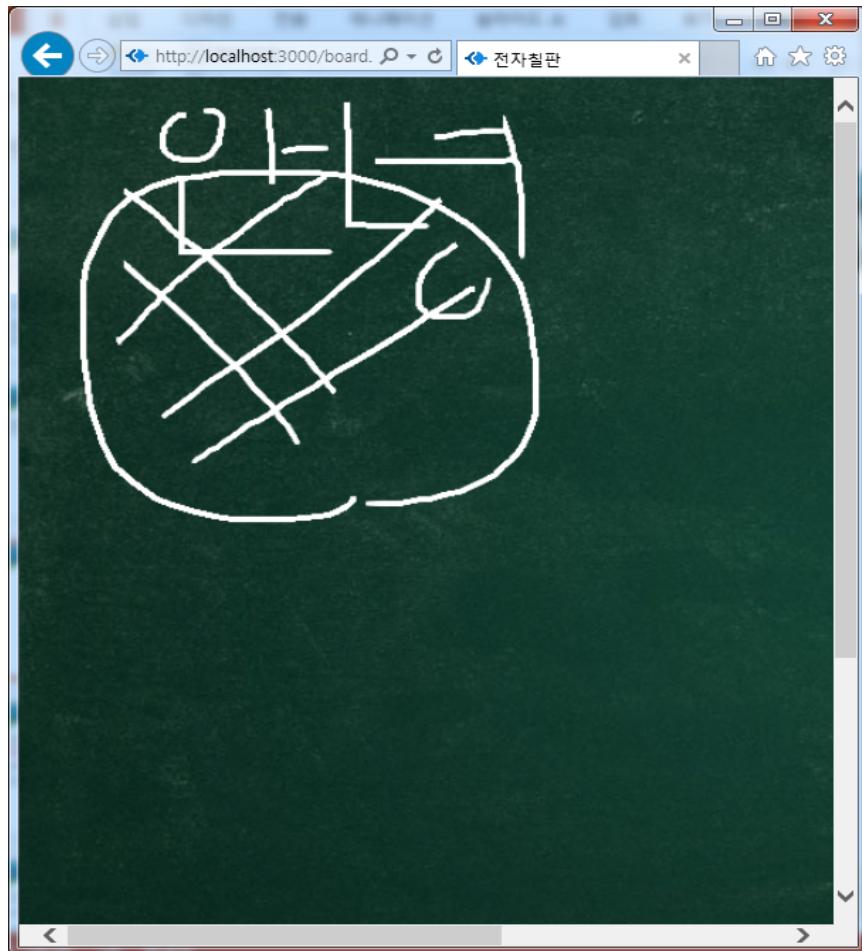
```
// 모양 변경 메서드  
setShape : function(color, width) {  
    if (color != null)  
        this.color = color;  
    if (width != null)  
        this.width = width;  
  
    ctx.strokeStyle = this.color;  
    ctx.lineWidth = this.width;  
  
    ctx.clearRect(703, 0, 860, 90);  
    ctx.beginPath();  
    ctx.moveTo(710, 55);  
    ctx.lineTo(820, 55);  
    ctx.stroke();  
}  
}
```

# 메뉴

- ❖ **board.js** 파일의 **document.ready** 함수에 **select**의 값이 변경될 때 호출되는 함수를 설정

```
$(‘select’).bind(‘change’, shape.change);
```

# WebSocket



# WebSocket

- ❖ app.js 파일에 소켓 서버 설정 코드 추가

```
var app = express();
var server = app.listen(3000);
var io = require('socket.io').listen(server);
```

- ❖ app.js 파일에서 http 서버 설정 코드 삭제

```
/*
http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
*/
```

# WebSocket

- ❖ **index.html** 파일에 **socket.io.js** 파일을 사용할 수 있도록 링크 설정

```
<script src="/socket.io/socket.io.js"></script>
```

- ❖ **board.js** 파일에 변수 선언

```
var socket;
```

- ❖ **board.js** 파일의 **document.ready** 함수에서 소켓 객체 생성

```
socket = io.connect('http://' + window.location.host);
```

# WebSocket

## ❖ board.js 파일에 msg 객체 생성

```
var msg = {
    line : {
        send : function(type, x, y) {
            console.log(type, x, y);
            socket.emit('linesend', {
                'type' : type,
                'x' : x,
                'y' : y,
                'color' : shape.color,
                'width' : shape.width
            });
        }
    }
}
```

# WebSocket

## ❖ board.js 파일의 draw 객체 수정

```
// 그리기 관련
var draw = {
    drawing : null,
    start : function(e) {
        ctx.beginPath();
        ctx.moveTo(e.pageX, e.pageY);
        this.drawing = true;
        msg.line.send('start',e.pageX,e.pageY);
    },
    move : function(e) {
        if (this.drawing) {
            ctx.lineTo(e.pageX, e.pageY);
            ctx.stroke();
            msg.line.send('move',e.pageX,e.pageY);
        }
    },
}
```

# WebSocket

```
end : function(e) {  
    this.drawing = false;  
    msg.line.send('end');  
},  
  
clear : function() {  
    ctx.clearRect(0, 0, cv.width, cv.height);  
    msg.line.send('clear');  
},  
}  
}
```

# WebSocket

- ❖ **socket.js 파일에 서버에서 메시지를 받았을 때 메시지를 전송하는 코드를 작성**

```
socket.on('linesend', function (data) {  
    console.log(data);  
    socket.broadcast.emit('linesend_toclient',data);  
});
```

# WebSocket

- ❖ **board.js 파일의 document.ready 함수에 소켓에서 이벤트를 받았을 때 이벤트 처리 코드를 작성**

```
socket.on('linesend_toclient', function (data) {  
    draw.drawfromServer(data);  
});
```

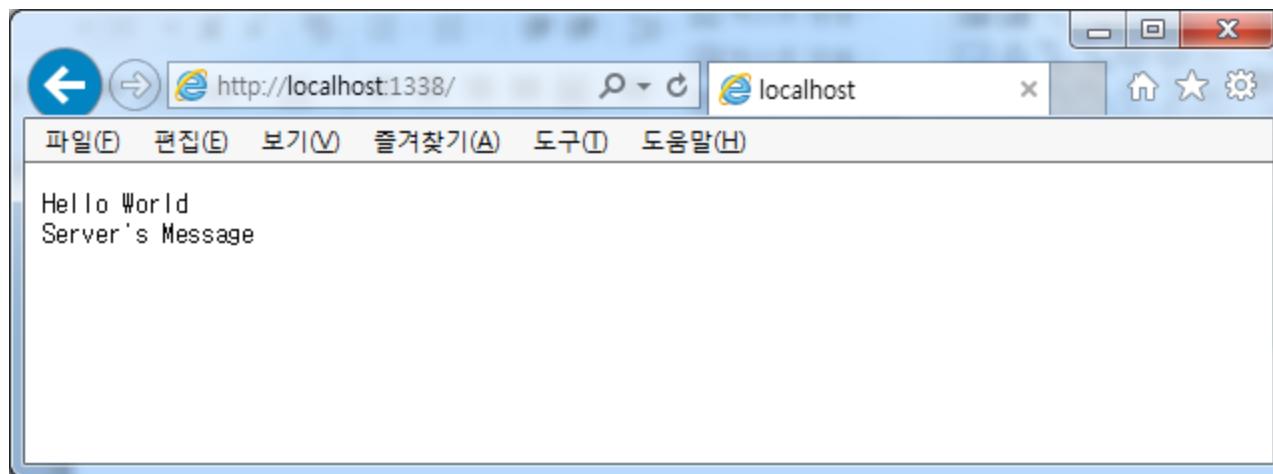
# WebSocket

## ❖ board.js 파일의 draw객체에 함수 추가

```
drawfromServer : function(data){
    if(data.type == 'start'){
        ctx.beginPath();
        ctx.moveTo(data.x,data.y);
        ctx.strokeStyle = data.color;
        ctx.lineWidth = data.width;
    }
    if(data.type == 'move'){
        ctx.lineTo(data.x,data.y);
        ctx.stroke();
    }
    if(data.type == 'end'){
    }
    if(data.type == 'clear'){
        ctx.clearRect(0, 0, cv.width,cv.height);
        shape.setShape();
    }
}
```

# Socket

# UDP 통신



# UDP 통신

## ❖ Java 프로젝트를 생성하고 UDPServer 클래스 생성

```
import java.net.*;
public class UDPServer {
    public static void main(String[] args) {
        try {
            DatagramSocket dsoc = new DatagramSocket(4445);
            byte[] data = new byte[65536];

            DatagramPacket dp = new DatagramPacket(data, data.length);
            System.out.println("서버 서비스 시작...");
            while (true) {
                dsoc.receive(dp);
                System.out.println("-----보낸 곳 주소 : "
                        + dp.getAddress().getHostAddress());
                System.out.println("자료 크기 : " + dp.getLength());
                String utf8String = new String(new String(dp.getData()).trim()).getBytes("UTF-
8"));
                System.out.println("내용 : " + utf8String);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# UDP 통신

```
String msg = "Server's Message";

InetAddress address = dp.getAddress();
int port = dp.getPort();
dp = new DatagramPacket(msg.getBytes(), msg.getBytes().length, address,
port);
dsoc.send(dp);

}

} catch (Exception e) {
    System.out.println("오류 : " + e);
}
}
```

# UDP 통신

## ❖ app.js 파일에 작성

```
var http = require('http'), client = require('dgram').createSocket('udp4'), message = 'This is my  
first message, next time you will see message from server!!';
```

```
client.on("message", function(msg, rinfo) {  
    message = msg;  
});
```

```
client.on("error", function(err) {  
    console.log("server error:\n" + err.stack);  
    server.close();  
});
```

```
http.createServer(function(req, res) {
```

# UDP 통신

```
// control for favicon
if (req.url === '/favicon.ico') {
    res.writeHead(200, {
        'Content-Type' : 'image/x-icon'
    });
    res.end();
    return;
}

var data = new Buffer('Client Buffer!!');
client.send(data, 0, data.length, 4445, 'localhost');

res.writeHead(200, {
    'Content-Type' : 'text/plain'
});
res.end('Hello World\n' + message);

}).listen(1338, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1338/');
```

# UDP 통신

- ❖ 2개의 프로젝트를 실행
- ❖ 웹 브라우저에서 1338번 포트로 접속

# TCP 통신

- ❖ Java 프로젝트를 생성하고 TCPServer 클래스 생성

```
import java.io.*;
import java.net.*;

public class TCPServer {
    public static void main(String[] args) {
        ServerSocket ss = null;
        Socket sock = null;
        try {
            ss = new ServerSocket(9999);
            while (true) {
                System.out.println("서버 대기중...");
                sock = ss.accept();
                System.out.println("접속자 정보 : " + sock.toString());
                BufferedReader in = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
                String str = in.readLine();
                System.out.println("전송된 내용 : " + str);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# TCP 통신

```
        in.close();
        sock.close();
    }
} catch (IOException e) {
    System.out.println("해당 포트 사용중!!!");
    try {
        ss.close();
    } catch (Exception ex) {
    }
}
}
```

# TCP 통신

## ❖ app.js 파일에 작성

```
var http = require('http'), client = require('net').Socket(), fromServer = "";

client.connect(9999, function() {
    console.log('Connected successfully!!');
});

// When I get 'data' from server, put 'data' in 'message' to send in response
client.on('data', function(data) {
    console.log('Recieve data: ' + data);
    fromServer = data;
});

// Add a 'close' event handler for the client socket
client.on('close', function() {
    console.log('Connection closed');
    // Close the client socket completely
    client.destroy();
});
```

# TCP 통신

```
http.createServer(function(req, res) {  
  
    // control for favicon  
    if (req.url === '/favicon.ico') {  
        res.writeHead(200, {  
            'Content-Type' : 'image/x-icon'  
        });  
        res.end();  
        return;  
    }  
  
    client.write('Who's there?\n', function() {  
        console.log('Write data successfully!!');  
    });  
});
```

# TCP 통신

```
res.writeHead(200, {  
  'Content-Type' : 'text/plain'  
});  
res.end('Hello World\n' + fromServer);  
}).listen(1338, '127.0.0.1');  
console.log('Server running at http://127.0.0.1:1337/');
```