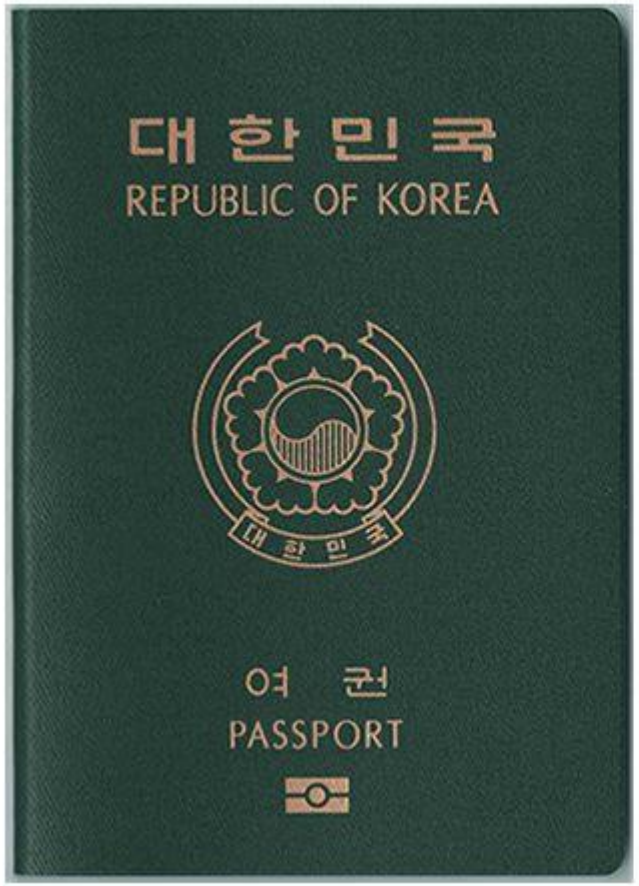
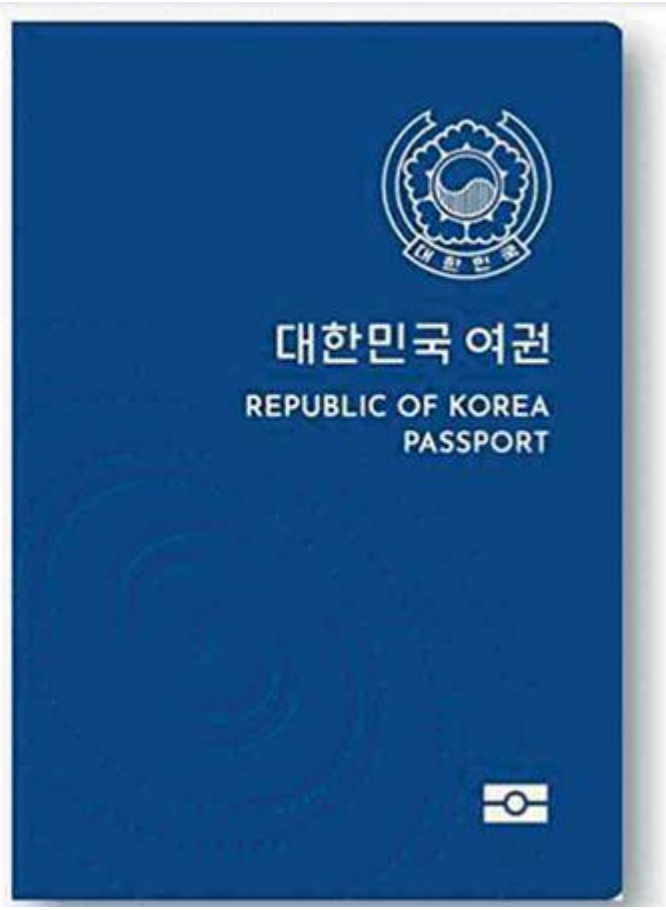
Hello,

KDT 웹 개발자 양성 프로젝트

1기! 36th

with


대 한 민 국
REPUBLIC OF KOREA
대 한 민 국
여 권
PASSPORT


대한민국 여권
REPUBLIC OF KOREA
PASSPORT

Multer 모듈로 이미지 업로드!

```
const dir = './uploads';
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, dir);
  },
  filename: function (req, file, cb) {
    cb(null, file.fieldname + '_' + Date.now());
  },
});

const limits = {
  fileSize: 1024 * 1028 * 2,
};

const upload = multer({ storage, limits });
```

```
router.post('/', login.isLogin, upload.single('img'), async (req, res) => {
  if (!fs.existsSync(dir)) fs.mkdirSync(dir);
```

작성자 : 13

dd

dd

수정 삭제

MongoDB

설치!

MongoDB Compass

Connect View Help

New connection +

Saved connections

- **tetz**
  Never

Recents

- **localhost:27017**
  2022년 9월 21일 오전 11:06
- **localhost:27017**
  2022년 9월 20일 오후 6:28

New Connection

Connect to a MongoDB deployment

FAVORITE

URI

Edit Connection String

mongodb://tetz:1234@localhost:27017/?authMechanism=DEFAULT

> Advanced Connection Options

Save | Save & Connect | Connect

New to Compass and don't have a cluster?

If you don't already have a cluster, you can create one for free using MongoDB Atlas

CREATE FREE CLUSTER

How do I find my connection string in Atlas?

If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.
See example

How do I format my connection string?

See example

암호화

Crypto

Crypto 암호화 함수 작성

- 특정 문자열을 Crypto 모듈의 sha512 방식으로 암호화 시켜주는 함수를 만들어 봅시다
- createHash() : 암호화 알고리즘
- Update() : 암호화할 문자열
- Digest() : 인코딩 방식

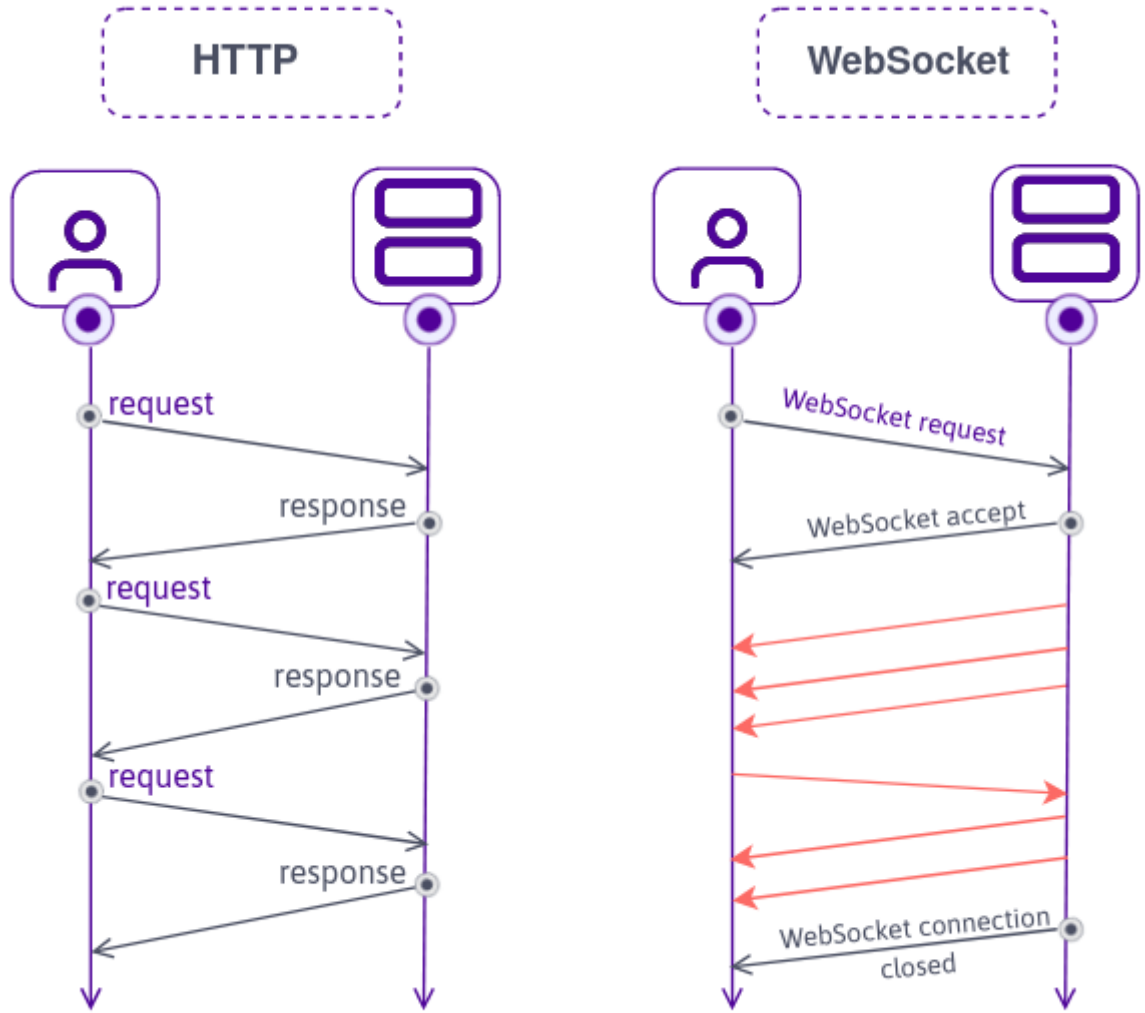
```
function createHashedPassword(password) {
  return crypto.createHash('sha512').update(password).digest('base64');
}
```

```
function verifyPassword(password, salt, userPassword) {
  const hashed = crypto
    .pbkdf2Sync(password, salt, 10, 64, 'sha512')
    .toString('base64');

  console.log('hashed', hashed);
  console.log('user pw', userPassword);
  if (hashed === userPassword) return true;
  return false;
}
```

```
hashed jB8if1ahU7Bja403dRxBb7ad865nu3xPazFHsji8WBZHoV+cE7mRGOkqAK/d1pCPdoOa0+c6cRVuSMvZHc5Wkw==
user pw jB8if1ahU7Bja403dRxBb7ad865nu3xPazFHsji8WBZHoV+cE7mRGOkqAK/d1pCPdoOa0+c6cRVuSMvZHc5Wkw==
true
```

웹 소켓

WEBSOCKETS


HTTP
request
response
request
response
request
response
WebSocket
WebSocket request
WebSocket accept
WebSocket connection
closed

프론트 코드 작성

간단한 채팅 서비스 구현

프론트 코드 작성

- 인풋 태그의 값을 입력 받아 전송 버튼을 누르면 웹소켓 서버로 전달
- 전송 버튼이 눌리면 인풋 태그는 초기화 되도록 설정

```
// 웹소켓 전역 객체 생성
var ws = new WebSocket("ws://localhost:3000");

const inputEl = document.getElementById('msg');
const chat = document.getElementById('chat');

function send() {
  const msg = inputEl.value;
  ws.send(msg);
  inputEl.value = '';
}
```

프론트 코드 작성

- 서버에서 메시지가 내려오면 해당 데이터를 받아서 chat list 에 추가!

```
// 서버로 부터 메시지를 수신한다
ws.onmessage = function (event) {
  const msgEl = document.createElement('li');
  msgEl.innerHTML = event.data;
  chat.appendChild(msgEl);
  // console.log("Server message: ", event.data);
}
```

```
// @ts-check
const express = require('express');

const router = express.Router();
const WebSocketServer = require('ws').Server;

const wss = new WebSocketServer({ port: 3000 });

wss.on('connection', (ws) => {
  // ws.send('저는 서버입니다! 들리십니까?');

  wss.clients.forEach((client) => {
    client.send(`새로운 유저가 접속 했습니다. 현재 유저 ${wss.clients.size}`);
  });

  ws.on('message', (message) => {
    wss.clients.forEach((client) => {
      client.send(`${message}`);
    });
  });

  ws.on('close', () => {
    wss.clients.forEach((client) => {
      client.send(`유저 한명이 떠났습니다. 현재 유저 ${wss.clients.size} 명`);
    });
  });
});

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

module.exports = router;
```

전송

1. 새로운 유저가 접속 했습니다. 현재 유저 2
2. 유저 한명이 떠났습니다. 현재 유저 1 명
3. 새로운 유저가 접속 했습니다. 현재 유저 2
4. 유저 한명이 떠났습니다. 현재 유저 1 명
5. 새로운 유저가 접속 했습니다. 현재 유저 2
6. 121
7. 33333

NEW TECHNOLOGY OF THE MONTH

이달의 신기술

REVOLUTION
혁신, 진화 그리고 혁명

REVOLUTION 1
인류의 이동을 중심으로 살펴본
1, 2차 산업혁명

REVOLUTION 2
세계 경제 질서의
근본적 변화를 야기한
1, 2차 산업혁명

REVOLUTION 3
3, 4차 산업혁명을 이끄는
기술 제품

이달의 신기술 100호 특집

삶을 변화시킨 대표 기술 및 상품

01
Vol. 100
JANUARY 2022

이달의 산업기술상
산업 재해로부터 작업자 안전을 지킨다
㈜오토닉스

트렌드
2022년 주목할 기술
AI, Space Tech, 그리고 NFT

CLUB
날아라 푸른 하늘로
중앙대학교 항공기 제작 동아리 마하

9 772288 490002
ISSN 2288-4904 ₩6,000

koa

koa

next generation web framework for node.js

koa?

- Express 팀에서 만든 프레임 워크 입니다
- Express 를 IBM 이 사버려서, 유지 보수 및 새로운 기능 추가가 예전만 못하다는 평을 듣고 있는 상황에서 Express 팀이 기존의 Express 의 단점을 보완해서 만든 것이라서 최근에 떠오르는 프레임 워크 입니다
- Express 대비 더 가볍고, 빠른 것이 특징입니다
- 미들웨어 레벨에서도 Async/Await 을 제공하여 비동기 프로그래밍을 더 편리하게 사용이 가능합니다

*Express

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;
const server = app.listen(PORT, () => {
        console.log(`Express is listening to <http://localhost>:${PORT});
});
```

*Koa

```
const koa = require('koa');
const app = express();
const PORT = process.env.PORT || 3000;
const server = app.listen(PORT, () => {
        console.log(`Koa is listening to <http://localhost>:${PORT});
});
```

*Express

```
app.use((req, res, next) => {
        console.log(`Time : ${Date.now()}`);
    next();
});
```

*Koa

```
app.use(async (ctx, next) => {
        console.log(`Time: ${Date.now()}`);
    await next();
});
```

*Express → 38510 req/sec

```
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => res.send('Hello world!'));
app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

*Koa → 50933 req/sec

```
const Koa = require('koa');
const app = new Koa();
app.use(async ctx => ctx.body = 'Hello world' });
app.listen(3000);
```

프로젝트

세팅하기

오랜만에 초기 세팅을 해봅시다!

- 연습도 할 겸 한번 세팅을 해보죠!
- Prettier 설치!
- npm install -D prettier
- Prettier 설정 파일인 .prettierrc 파일과 .vscode setting.json 파일은 따로 치기 귀찮으니 가져 옵시다! → 이전 프로젝트 폴더에서 카피하기!

오랜만에 초기 세팅을 해봅시다!

- ES-lint 설치
- npm install -D eslint
- npm install -D eslint-config-airbnb-base eslint-plugin-import
- Eslint 설정 파일인 .eslintrc.js 파일도 가져 옵시다!

오랜만에 초기 세팅을 해봅시다!

- Typescript 설치
- npm install -D typescript
- npm install -D @types/node

Koa 설치

Koa 프레임워크 설치 및 실행

- npm install koa

```
const Koa = require('koa');

const app = new Koa();
const PORT = 4500;

app.use(async (ctx, next) => {
  console.log(ctx.request);
  console.log(ctx.response);
  ctx.body = 'Hello, koa world!';
});

app.listen(PORT);
```

```
Hello, koa world!
```

Koa

- Koa 는 req, res 가 하나로 합쳐 캡슐화 시킨 context 인 ctx 를 사용

request

```
{
  method: 'GET',
  url: '/',
  header: {
    host: 'localhost:4500',
    connection: 'keep-alive',
    pragma: 'no-cache',
    'cache-control': 'no-cache',
    'sec-ch-ua': '"Google Chrome";v="105", "Not)A;Brand";v="8", "Chromium";v="105"',
    'sec-ch-ua-mobile': '?0',
    'sec-ch-ua-platform': '"Windows"',
    'upgrade-insecure-requests': '1',
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36',
    accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9',
    'sec-fetch-site': 'none',
    'sec-fetch-mode': 'navigate',
    'sec-fetch-user': '?1',
    'sec-fetch-dest': 'document',
    'accept-encoding': 'gzip, deflate, br',
    'accept-language': 'ko-KR,ko;q=0.9,en-GB;q=0.8,en;q=0.7,en-US;q=0.6,pt;q=0.5',
    cookie: 'popup=hide'
  }
}
```

Koa

```
{
  status: 404,
  message: 'Not Found',
  header: [Object: null prototype] {},
  body: undefined
}
```

response

Pug

pug

pug

- Node.js 용 View engine 입니다!
- html 대비 간단하게 태그를 사용할 수 있습니다
- 템플릿 기능을 제공합니다

pug 설치 및 적용

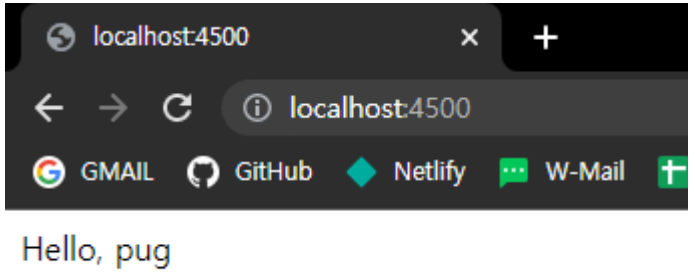
- npm install koa-pug -s
- Pug 미들웨어 적용하기

```
const Pug = require('koa-pug');
const path = require('path');

const pug = new Pug({
  viewPath: path.resolve(__dirname, './views'),
  app,
});

app.use(async (ctx) => {
  await ctx.render('chat');
});
```

pug 코드 작성

- Views 폴더를 만들고 chat.pug 파일을 생성
- Pug 를 작성해 봅시다

```
html
    head
    body
        div Hello, Pug!
```



pug 에 부트 스트랩 적용

- Pug 에 부트스트랩을 적용시켜 봅시다

```
html
    head
        link(href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min
.css" rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous")
```

pug + 부트스트랩

- 부트스트랩을 이용해서 간단한 채팅 UI 디자인

```
html
    head
        link(href="")
    body.d-flex.text-center.flex-column.align-items-center
        h1.w-100.p-3.bg-primary.text-white.font-bold Tetz 채팅
        div.w-50.h-75.p-5.bg-secondary.text-bottom.overflow-auto
            p.p-2.bg-primary.fw-bold.text-white 채팅 예시
        form.w-50
            input.w-75.m-3.p-3
            a.p-3.btn.btn-primary 보내기
```

WEBSOCKETS

Koa

Web socket

Koa-web socket 을 설치

- Npm i koa-websocket
- https://www.npmjs.com/package/koa-websocket
- 모듈 추가하기

```
const Koa = require('koa');
const websockify = require('koa-websocket');
const route = require('koa-route’);

const app = websockify(new Koa());
```

Koa-web socket 사용해 보기

- 웹 소켓 사용을 위한 코드 모듈 추가

```
app.ws.use(
  route.all('/chat', (ctx) => {
    ctx.websocket.send('아아~ 들리십니까 여긴 서버입니다!');
    ctx.websocket.on('message', (message) => {
      console.log(message.toString());
    });
  })
);
```

- /chat 이라는 경로로 들어오면 웹 소켓 통신을 통해 클라이언트로 Hello, world 라는 문구를 전송
- 클라이언트에서 message 가 오면 message 를 받아서 서버에 출력

클라이언트

코드 준비

클라이언트 코드 준비

- 백엔드에는 메시지를 받을 소켓 서버 코드를 준비 완료!
- 이번에는 클라이언트(프론트)에서 서버로 소켓 통신을 통해 데이터를 전달할 수 있도록 준비!
- 제일 외부에 public 폴더 작성하기!
- Public 폴더에 chat.js 라고 하는 파일을 작성하고 해당 파일에서 클라이언트 채팅 관련 기능을 구현!

클라이언트 코드 준비

- Chat.pug 파일에서 chat.js 파일 불러오기!

```
html
    head
        link(href="")
    body.d-flex.text-center.flex-column.align-items-center
        h1.w-100.p-3.bg-primary.text-white.font-bold Tetz 채팅
        div.w-50.h-75.p-5.bg-secondary.text-bottom.overflow-auto
            p.p-2.bg-primary.fw-bold.text-white 채팅 예시
        form.w-50
            input.w-75.m-3.p-3
            a.p-3.btn.btn-primary 보내기
    script(src="public/chat.js")
```

클라이언트 코드 준비

- Chat.js 가 잘 불러져 왔는지 확인하기 위해 chat.js 에 alert 코드 추가!

```
// @ts-check
alert('!!');
```

- 하지만 동작을 안하네요?
- 왜냐하면 Static 설정이 안되어 있기 때문입니다!

Koa static

- Koa 는 기본적으로 Express 대비 가벼운 프레임 워크를 제공하기 위해 express 에서는 기본적으로 들어 있던 기능들도 모듈로 분리하여 사용
- Express 에서는 Static 자체적으로 주소에 따른 라우팅 기능을 제공 하였지만 Koa 는 해당 기능을 쓰려면 koa-mount 라는 모듈 설치가 필요함

Express vs Koa

```
app.use('/uploads', express.static('uploads'));
```

- Express 내장 모듈로도 특정 주소에 대한 Static 폴더 설정이 가능

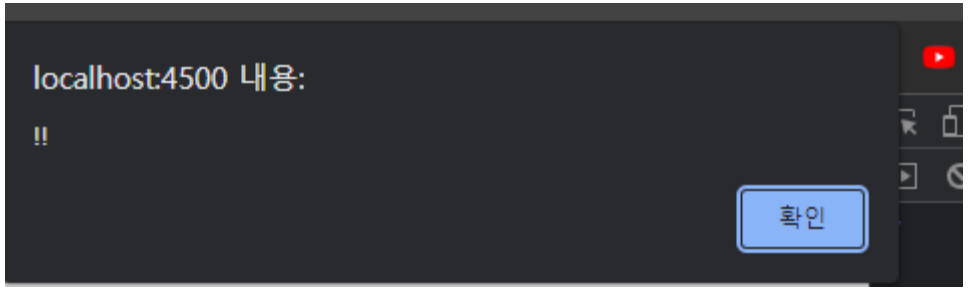
```
const serve = require('koa-static');
const mount = require('koa-mount');
app.use(mount('/public', serve('public')));
```

- Koa 는 koa-static 과 koa-mount 모듈을 둘 다 설치 해야만 가능

Koa static

- Koa-static 과 koa-mount 를 사용하여 public/ 주소로 public 폴더에 접근이 가능하도록 설정

```
const serve = require('koa-static');
const mount = require('koa-mount’);

// 다른 코드들
app.use(mount('/public', serve('public')));
```



클라이언트에서

소켓 통신 보내기

클라이언트에서 통신 보내기

- 클라이언트에서 Websocket 서버를 연결하고 서버에 addEventListener 를 등록하여 이전에 준비 해두었던 서버로 통신을 보내 봅시다!
- Open 이라는 AddEventListener 를 통해 접속 되면 바로 메시지를 전송

```
// @ts-check
const socket = new
WebSocket(`ws://${window.location.host}/chat`);

socket.addEventListener('open', () => {
  socket.send('안녕하세욧, 저는 클라이언트 에요!');
});
```

```
[nodemon] starting `node app.js`
서버는 4500 번 포트에서 실행 중입니다!
안녕하세욧, 저는 클라이언트 에요!
```

클라이언트에서 통신 받기

- 이번에는 서버에서 보낸 통신을 받아 봅시다!
- Message 라는 addEventListener 로 받습니다!
- 서버에서 보낸 메시지는 event 매개 변수의 data 키에 담겨서 들어옵니다

```
// @ts-check
const socket = new
WebSocket(`ws://${window.location.host}/chat`);

socket.addEventListener('message', (event) => {
  console.log(event.data);
});
```

```
아아~ 들리십니까 여긴 서버입니다!
```

IIFE

(Immediately Invoked Function Expression)

IIFE(Immediately Invoked Function Expression)

- 함수가 정의되지 마자 사용되는 즉시 실행 함수라고 합니다
- 보통 함수의 정의 부분을 외부로 부터 감추고 싶을 때 사용하는 방법입니다
- 현재 저희 클라이언트에서 console 창을 띄우고 socket 을 입력해 보세요

```
> socket
< WebSocket {url: 'ws://localhost:4500/chat', readyState: 3, bufferedAmount: 0, onopen: null, onerror: null…
```

- 저희가 정의한 소켓을 바로 확인할 수 있습니다!

IIFE(Immediately Invoked Function Expression)

- 이런 부분을 감추고 싶을 때, IIFE 를 사용합니다

```
// IIFE
(function () {
  const socket = new WebSocket(`ws://${window.location.host}/chat`);

  socket.addEventListener('open', () => {
    socket.send('안녕하세욧, 저는 클라이언트 에요!');
  });

  socket.addEventListener('message', (event) => {
    console.log(event.data);
  });
})();
```

```
  모든 클라이언트에게 데이터를 보낸다 실시!
> socket
```

Broadcast

Broadcast

- 웹 소켓 서버는 app.ws 에 할당 되어 있습니다.
- 웹 소켓 서버를 server 라는 변수에 로 받아 봅시다!
- 다만 웹 소켓 서버는 server 정보 이외에 많은 데이터를 담고 있으므로 해당 오브젝트 중에서 server 라는 키에 할당 된 값만 받아와야 합니다
- 구조 분해 할당 문법으로 받아 오면 됩니다!

```
const { server } = app.ws;
```

Server 변수를 사용하여 전체 전달

- Server 의 clients 키에는 접속한 클라이언트가 배열로 저장 됩니다!
- 저번에 사용한 forEach 를 통해 모든 클라이언트에 통신을 보내 봅시다!

```
app.ws.use(
  route.all('/chat', (ctx) => {
    const { server } = app.ws;

    server.clients.forEach((client) => {
      client.send('모든 클라이언트에게 데이터를 보낸다 실시!');
    });
  })
);
```

Server 변수를 사용하여 전체 전달

- 이제 다른 탭에서 접속을 하면 접속한 클라이언트의 수 만큼 console.log 가 찍힙니다!

```
2 모든 클라이언트에게 데이터를 보낸다 실시!
```

실제 채팅 구현

실제 채팅의 흐름

- 클라이언트의 form 에 내용을 입력하고 보내기를 누르면 서버로 데이터를 전달
- 서버는 해당 정보를 다시 모든 클라이언트에게 보내는 형태로 구조를 만들어 봅시다!
- Pug 에 구현한 form 의 input 에 id 를 부여하고 해당 form 에서 입력 받은 데이터를 서버로 전달해 봅시다!

Form 데이터 전달

- Pug 에서 id 는 # 으로 부여합니다!

```
html
    head
        link(href="")
    body.d-flex.text-center.flex-column.align-items-center
        h1.w-100.p-3.bg-primary.text-white.font-bold Tetz 채팅
        div#chat.w-50.h-75.p-5.bg-secondary.text-bottom.overflow-auto
        form(action="get" onsubmit="return false;")#form.w-50
            input(type="text" placeholder="채팅을 입력하세요!")#input.w-75.m-3.p-3
            a#btn.p-3.btn.btn-primary 보내기
    script(src="public/chat.js")
```

프론트 Chat.js 수정

- Input 과 보내기 버튼을 변수에 담아서 처리해 줍시다!
- 그리고 채팅이 표현 될, chat 도 미리 저장해 둡시다!

```
const socket = new WebSocket(`ws://${window.location.host}/chat`);
const btn = document.getElementById('btn');
const inputEl = document.getElementById('input’);
const chatEl = document.getElementById('chat');
```

프론트 Chat.js 수정

- 보내기 버튼을 클릭하면 inputEl 의 값을 읽어서 서버로 보내고 input 의 값은 초기화를 시켜봅시다!
- 그리고 이제는 유저의 이름도 같이 보내 봅시다!

```
btn.addEventListener('click', () => {
  const msg = inputEl.value;
  const data = {
    name: 'tetz',
    msg,
  };
  socket.send(JSON.stringify(data));
  inputEl.value = '';
});
```

서버 사이드 수정

- 서버에는 아직 클라이언트의 통신을 받는 코드가 없습니다!

```
ctx.websocket.on('message', (message) => {
   console.log(message.toString());
});
```

- On('message', (message) => {}) 로 클라이언트 통신을 받아 봅시다!

```
서버는 4500 번 포트에서 실행 중입니다!
안녕하세욧, 저는 클라이언트 에요!
12121
```

서버 사이드 수정

- 메시지 받기에는 성공 했으니 받은 메시지를 다시 클라이언트로 전달

```
app.ws.use(
  route.all('/chat', (ctx) => {
    const { server } = app.ws;

    ctx.websocket.on('message', (message) => {
      server.clients.forEach((client) => {
        client.send(message);
      });
    });
  })
);
```

프론트 Chat.js 수정

- 서버에서 전달 받은 채팅 내용을 chat div 에 추가해 봅시다!
- p 태그를 만들고, 부트스트랩 클래스를 붙인 뒤에 chat div 에 추가

```
socket.addEventListener('message', (event) => {
    const { name, msg } = JSON.parse(event.data);

    const msgEl = document.createElement('p');
    msgEl.classList.add('p-2');
    msgEl.classList.add('bg-warning');
    msgEl.classList.add('text-black');
    msgEl.classList.add('fw-bold');
    msgEl.innerText = `${name} : ${msg}`;
    chatEl.appendChild(msgEl);
    chatEl.scrollTop = chatEl.scrollHeight - chatEl.clientHeight;
});
```

서버 메시지

추가

서버의 안내 문구 코드 작업

- 서버는 새로운 유저가 접속하면 접속 했다는 메시지와 현 채팅방의 인원을 안내해 줘야 합니다
- 서버는 유저가 나가면 나갔다는 메시지와 현 채팅방의 인원을 안내해 줘야 합니다
- 해당 코들르 작업해 봅시다!

서버의 안내 문구 코드 작업

- 서버는 새로운 유저가 접속하거 나가면 알림 메시지와 현 채팅방의 인원을 안내해 줘야 합니다
- 이러한 부분은 백엔드 서버에서만 가능합니다! → 클라이언트 수 파악 등의 문제

접속 안내

- 웹 소켓 서버 상단에 위치 시켜 주면 됩니다
- 미들 웨어는 할 일을 마치면 뒤로 넘기기 때문에 웹 소켓 서버 상단에 위치

```
server.clients.forEach((client) => {
      client.send(
        JSON.stringify({
            name: '서버',
            msg: `새로운 유저가 참여 했습니다. 현재 유저 수 ${server.clients.size}`,
          },
        })
      );
    });
```

접속 종료 안내

- On('close', () => {}) 로 사용하면 됩니다!

```
server.clients.forEach((client) => {
      client.send(
        JSON.stringify({
            name: '서버',
            msg: `새로운 유저가 참여 했습니다. 현재 유저 수 ${server.clients.size}`,
          },
        })
      );
    });
```

실습, 랜덤 닉네임 정하기 구현!

- 아래의 데이터를 사용해서 랜덤한 닉네임을 정해서 전달하는 부분을 구현해 주세요!
- 각각의 배열에서 랜덤한 값을 골라서 리턴하는 함수를 만들어 봅시다!
- 그리고 각각의 배열에서 뽑힌 값을 합쳐서 닉네임을 구현해 보아요!
- 결과만 랜덤 하다면 다양한 방법으로 구현하셔도 좋습니다!

```
const adj = [
  '멋진',
  '잘생긴',
  '예쁜',
  '졸린',
  '우아한',
  '힙한',
  '배고픈',
  '집에 가기 싫은',
  '집에 가고 싶은',
  '귀여운',
  '중후한',
  '똑똑한',
  '이게 뭔가 싶은',
  '까리한',
  '프론트가 하고 싶은',
  '백엔드가 재미 있는',
  '몽고 디비 날려 먹은',
  '열심히하는',
  '피곤한',
  '눈빛이 초롱초롱한',
  '치킨이 땡기는',
  '술이 땡기는',
];
```

```
const member = [
  '유림님',
  '지훈님',
  '한솔님',
  '윤비님',
  '승환님',
  '영은님',
  '수지님',
  '종익님',
  '혜영님',
  '준우님',
  '진형님',
  '민정님',
  '소민님',
  '지현님',
  '다영님',
  '세영님',
  '의진님',
  '승수님',
  '해성님',
  '허원님',
];
```

유저별 색상 변경하기도 만들어 봅시다!

- 유저별로 구별하기 편하게 하기 위해 유저별로 색상을 지정해서 해당 유저가 채팅을 올리면 해당 색상으로 올라가도록 해봅시다!

	글자 타입 (text-)	배경 타입 (bg-)	경고 타입 (alert-)
primary	.text-primary	.bg-primary	alert-primary
secondary	.text-secondary	.bg-secondary	alert-secondary
success	.text-success	.bg-success	alert-success
danger	.text-danger	.bg-danger	alert-danger
warning	.text-warning	.bg-warning	alert-warning
info	.text-info	.bg-info	alert-info
light	.text-light	.bg-light	alert-light
dark	.text-dark	.bg-dark	alert-dark
white	.text-white	.bg-white	

유저별 색상 변경하기도 만들어 봅시다!

- 이미 랜덤 함수는 구현되어 있으므로 부트 스트랩의 색상만 넣어 줍시다!
- 단, 서버 메시지는 유저 접속 시 빨간색 / 접속 종료 시 검정색으로 줍시다!

```
const bootColor = [
  { bg: 'bg-primary', text: 'text-white' },
  { bg: 'bg-success', text: 'text-white' },
  { bg: 'bg-warning', text: 'text-black' },
  { bg: 'bg-info', text: 'text-white' },
  { bg: 'alert-primary', text: 'text-black' },
  { bg: 'alert-secondary', text: 'text-black' },
  { bg: 'alert-success', text: 'text-black' },
  { bg: 'alert-danger', text: 'text-black' },
  { bg: 'alert-warning', text: 'text-black' },
  { bg: 'alert-info', text: 'text-black' },
];
```

유저별 색상 변경하기도 만들어 봅시다!

- 이미 랜덤 함수는 구현되어 있으므로 부트 스트랩의 색상만 넣어 줍시다!
- 단, 서버 메시지는 유저 접속 시 빨간색 / 접속 종료 시 검정색으로 줍시다!

```
const bootColor = [
  { bg: 'bg-primary', text: 'text-white' },
  { bg: 'bg-success', text: 'text-white' },
  { bg: 'bg-warning', text: 'text-black' },
  { bg: 'bg-info', text: 'text-white' },
  { bg: 'alert-primary', text: 'text-black' },
  { bg: 'alert-secondary', text: 'text-black' },
  { bg: 'alert-success', text: 'text-black' },
  { bg: 'alert-danger', text: 'text-black' },
  { bg: 'alert-warning', text: 'text-black' },
  { bg: 'alert-info', text: 'text-black' },
];
```

유저별 색상 변경하기도 만들어 봅시다!

- 색상 조합도 랜덤 함수를 이용해 thema 변수에 넣고 서버로 전달하는 데이터에도 bg, text 부분을 만들어서 부트스트랩의 컬러 정보를 같이 전달

```
const thema = pickRandomArr(bootColor);
btn.addEventListener('click', () => {
    const msg = inputEl.value;
    const data = {
      name: nickName,
      msg,
      bg: thema.bg,
      text: thema.text,
    };
    socket.send(JSON.stringify(data));
    inputEl.value = '';
  });
```

유저별 색상 변경하기도 만들어 봅시다!

- 프론트에서 실제로 그려 줄 때, 백그라운드와 텍스트 컬러 부분에 대한 클래스를 추가

```
socket.addEventListener('message', (event) => {
  const { name, msg, bg, text } = JSON.parse(event.data);

  const msgEl = document.createElement('p');
  msgEl.classList.add('p-2');
  msgEl.classList.add('fw-bold');
  msgEl.classList.add(bg);
  msgEl.classList.add(text);
  msgEl.innerText = `${name} : ${msg}`;
  chatEl.appendChild(msgEl);
  chatEl.scrollTop = chatEl.scrollHeight - chatEl.clientHeight;
  console.log(chatEl?.scrollTop);
});
```

MongoDB

연결

몽고 디비 설치!

- Npm i mongodb
- 이전 프로젝트에서 mongo.js 코드 가져오기
- 몽고 클라이언트 모듈 가져오기

```
const { MongoClient, ServerApiVersion } = require('mongodb');

const uri =
  'mongodb+srv://tetz:qwer1234@cluster0.sdiakr0.mongodb.net/?retryWrites=true&w=majority';

const client = new MongoClient(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverApi: ServerApiVersion.v1,
});

module.exports = client;
```

mongo.js

```
const mongoClient = require('./public/mongo’);

// eslint-disable-next-line no-underscore-dangle
const _client = mongoClient.connect();
```

app.js

채팅 내역

DB 저장

서버로 채팅 메시지가 전달 시 DB에 저장!

- 클라이언트에서 전송 된 메시지가 들어오면 다시 클라이언트로 보내기 전에 DB에 저장하여 줍시다!
- 먼저 컨트롤을 편하게 하기 위해서 전달 받은 메시지를 JSON.parse 하여 JS 오브젝트로 변경한 다음 chat 이라는 변수에 넣기
- Chat 에는 name, msg, bg, text 등의 다양한 값이 들어 있으므로 이를 전개 연산자로 풀어서 DB에 넣기

```
ctx.websocket.on('message', async (message) => {
      const chat = JSON.parse(message);

      const insertClient = await _client;
      const chatCursor = insertClient.db('kdt1').collection('chats');
      await chatCursor.insertOne({
        ...chat,
      });

      server.clients.forEach((client) => {
        client.send(message.toString());
      });
    });
```

```
_id: ObjectId('632d0e8c6752783aa2a7cfae')
name: "멋진 영은님"
msg: "12121"
bg: "bg-primary"
text: "text-white"
```

채팅 내역

불러오기

채팅 내역 불러오기

- 이제 새로운 클라이언트가 접속을 하면 이전 채팅 내역을 전달 필요DB에 접속하여 모든 채팅 내역을 받아서 클라이언트로 전달
- 단, 이 액션은 클라이언트가 서버 접속 시 최초에 발생해야 합니다!
- 그리고, 사용자가 보낸 채팅과 구분될 필요가 있습니다! → Type 키 추가
- 또한, 이 내역은 전체 클라이언트에게 보내면 안되고 접속한 클라이언트에게만 보내면 됩니다!

```
app.ws.use(
  route.all('/chat', async (ctx) => {
    const { server } = app.ws;

    const client = await _client;
    const cursor = client.db('kdt1').collection('chats');
    const chats = cursor.find({});
    const chatsData = await chats.toArray();

    // type 을 sync 로 전달하여 이전 채팅 내역임을 알려기
    ctx.websocket.send(
      JSON.stringify({
        type: 'sync',
        data: {
          chatsData,
        },
      })
    );

    // 다른 코드들
```

채팅 구분하기

- 이전 채팅 내역과 실제로 사용자가 보내는 채팅은 구분이 되어야 합니다!
- DB가 아닌 클라이언트에서 받은 채팅은 type 에 ‘chat’ 을 추가해서 구분해 줍니다!

```
ctx.websocket.on('message', async (message) => {
      const chat = JSON.parse(message);

      const insertClient = await _client;
      const chatCursor = insertClient.db('kdt1').collection('chats');
      await chatCursor.insertOne({
        ...chat,
      });

      // 사용자로 부터 입력 받은 채팅에는 type: 'chat' 추가
      server.clients.forEach((client) => {
        client.send(
          JSON.stringify({
            type: 'chat',
            data: {
              ...chat,
            },
          })
        );
      });
    });
```

클라이언트

코드 수정

채팅 내역과 실제 채팅을 구분하여 처리

- 이제 채팅 내역과 실제 채팅은 type 이라는 키의 값을 통해 구분이 가능하므로 해당 분기에 따른 처리
- 기존의 구조 분해 할당 문법으로는 못 받으므로 하나의 객체로 받기

```
socket.addEventListener('message', (event) => {
    const msgData = JSON.parse(event.data);
```

채팅 내역과 실제 채팅을 구분하여 처리

- 이제 채팅 내역과 실제 채팅은 type 이라는 키의 값을 통해 구분이 가능하므로 해당 분기에 따른 처리
- 기존의 구조 분해 할당 문법으로는 못 받으므로 하나의 객체로 받기

```
const msgData = JSON.parse(event.data);
```

- 채팅 내역은 방대한 배열 데이터 이므로 해당 배열을 저장할 chats 선언

```
  const chats = [];
```

채팅 내역과 실제 채팅을 구분하여 처리

- msgData 의 type 이 sync 이면 이전 채팅 내역이므로 해당 데이터를 전부 chats 배열에 푸쉬 → 이때 전개 연산자를 사용하여 처리
- msgData 의 type 이 chat 이면 클라이언트로 부터 전달 된 실제 채팅이므로 chats 배열에 푸쉬

```
socket.addEventListener('message', (event) => {
    const msgData = JSON.parse(event.data);
    const { type, data } = msgData;

    if (msgData.type === 'sync') {
      const oldChats = data.chatsData;
      chats.push(...oldChats);
    } else if (msgData.type === 'chat') {
      chats.push(data);
    }
  });
```

채팅 내역

그려주기

채팅 내역을 클라이언트에서 그려주기

- 이제 분기 별로 채팅 내역을 그려 주면 됩니다!
- 먼저 이전 채팅 내역을 그려줄 때에는 chats 에 담긴 모든 배열의 값을 추가해 주면 됩니다
- 실제 채팅의 경우에는 들어온 메시지만 추가해 주면 됩니다!
- 해당 기능을 drawChats 라는 함수로 만들어 구현해 봅시다!

```
function drawChats(type, data) {
    if (type === 'sync') {
      chatEl.innerHTML = '';
      chats.forEach(({ name, msg, bg, text }) => {
        const msgEl = document.createElement('p');
        msgEl.classList.add('p-2');
        msgEl.classList.add(bg);
        msgEl.classList.add(text);
        msgEl.classList.add('fw-bold');
        msgEl.innerText = `${name} : ${msg}`;
        chatEl.appendChild(msgEl);
        chatEl.scrollTop = chatEl.scrollHeight - chatEl.clientHeight;
      });
    } else if (type === 'chat') {
      const msgEl = document.createElement('p');
      msgEl.classList.add('p-2');
      msgEl.classList.add(data.bg);
      msgEl.classList.add(data.text);
      msgEl.classList.add('fw-bold');
      msgEl.innerText = `${data.name} : ${data.msg}`;
      chatEl.appendChild(msgEl);
      chatEl.scrollTop = chatEl.scrollHeight - chatEl.clientHeight;
    }
  }
```

채팅 내역을 클라이언트에서 그려주기

- drawChats 을 상황에 맞게 적용

```
socket.addEventListener('message', (event) => {
    const msgData = JSON.parse(event.data);
    const { type, data } = msgData;

    if (msgData.type === 'sync') {
      const oldChats = data.chatsData;
      chats.push(...oldChats);
      drawChats(msgData.type, data);
    } else if (msgData.type === 'chat') {
      chats.push(data);
      drawChats(msgData.type, data);
    }
  });
```

서버 메시지

수정

서버 메시지 수정

- 서버 메시지는 type 도 없고, 색상 값도 없으므로 현재 표시가 안되고 있습니다.
- 해당 부분을 수정

```
server.clients.forEach((client) => {
      client.send(
        JSON.stringify({
          type: 'chat',
          data: {
            name: '서버',
            msg: `새로운 유저가 참여 했습니다. 현재 유저 수 ${server.clients.size}`,
            bg: 'bg-danger',
            text: 'text-white',
          },
        })
      );
    });
```

```
ctx.websocket.on('close', () => {
      server.clients.forEach((client) => {
        client.send(
          JSON.stringify({
            type: 'chat',
            data: {
              name: '서버',
              msg: `유저 한명이 나갔습니다. 현재 유저 수 ${server.clients.size}`,
              bg: 'bg-black',
              text: 'text-white',
            },
          })
        );
      });
    });
```

채팅 내역

시간 적용

채팅 내역 시간 적용

- 채팅은 시간에 따라 순서대로 적용이 되어야 하므로 채팅 데이터에 시간 항목을 추가하고, 채팅 내역을 불러 올 때 시간 순서대로 불러와서 해당 문제를 해결 합시다!
- 채팅 내용 삽입 시에 시간 항목 추가

```
await chatCursor.insertOne({
        ...chat,
        createdAt: new Date(),
});
```

채팅 내역 시간 적용

- 채팅 내용 불러 올 때, 시간 항목을 오름 차순으로 받아와서 시간 순서대로 정렬 되도록 수정

```
const client = await _client;
    const cursor = client.db('kdt1').collection('chats');
    const chats = cursor.find(
      {},
      {
        sort: {
          createdAt: 1,
        },
      }
    );
    const chatsData = await chats.toArray();
```

엔터키 적용

엔터키를 쳤을 때도 채팅이 전송 되게 수정

- 엔터키를 쳤을 때, 채팅이 전달 되도록 설정
- 엔터키를 치면 form 데이터가 자동 전송이 되고 페이지가 새로 고침이 되므로 해당 부분 수정

```
form(action="get" onsubmit="return false;")#form.w-50
```

엔터키를 쳤을 때도 채팅이 전송 되게 수정

- 인풋 태그에 입력되는 키의 값을 보다가 엔터키가 나오면 버튼이 클릭 된 것 처럼 처리하여 엔터키로 메시지 전달

```
inputEl.addEventListener('keyup', (event) => {
    if (event.keyCode === 13) {
      btn.click();
    }
  });
```


YAY
WE DID IT!
makeameme.org

React JS

React JS


수고하셨습니다!