

첫번째로 서버는 현재 Node.js(Express)기반으로 동작하고 있습니다. Node.js는 v8자바스크립트 엔진을 웹 브라우저가 아닌 서버로 사용할 수 있도록 만든 것이 Node.js입니다. Node.js의 특징은 첫번째로 단일 쓰레드(Non-blocking I/O)방식을 사용합니다. 두번째로는 V8 Engine입니다. 세번째로는 Event-driven방식을 사용합니다. 잠깐 설명을 하자면 대부분의 웹 서버는 사용자가 이벤트를 발생하기까지를 기다리면서 자원을 계속 소비하지만 Event-driven은 입력장치로 데이터를 전송했을 때에만 작동하는 방식입니다. 발생한 이벤트에 대해서만 웹서버가 연결해주기 때문에 자원을 최소화 할 수 있습니다. 네번째로는 Npm을 사용할 수 있습니다. 이러한 특징으로 Node.js를 선택하였습니다. 사용하는 라이브러리는 socket.io 모듈 body-parser모듈 그리고 mongodb모듈을 사용합니다. Socket.io모듈은 인공지능 서버와 애플리케이션과 실시간으로 소켓통신을 하기위해 사용합니다. body-parser은 현재 모든 데이터를 post전송으로 받고 있습니다. 그래서 body-parser라는 모듈을 사용하여 값들을 받습니다. mongodb모듈을 사용하여 몽고디비와 연결을 하고 컬렉션 추가와 업데이트를 사용합니다.



두번째로 데이터베이스는 몽고디비를 사용하고 있습니다. 몽고디비는 NoSQL의 대표적인 데이터베이스입니다. 몽고디비는 컬렉션의 집합이며 각자 컬렉션은 여러 개의 문서 객체를 가질 수 있습니다. 몽고디비를 사용한 이유는 스키마가 없고 진입장벽이 낮기 때문에 사용 했습니다. 스키마가 없없다는 것 필요할 때 마다 필드를 추가하거나 제거하는 것이 매우 쉬워지고 개발 과정이 매우 단순해지고 빠르게 개발이 가능하기 때문에 선택하였습니다.



세번째로 IDE는 Webstorm을 사용하고 있습니다. Webstorm은 Node.js나 최신 웹 트렌드를 지원하며 플랫폼에 유동적이고 훌륭한 자바스크립트 에디터를 제공하고 있어 선택하였습니다.

제작 과정

Node.js는 <https://nodejs.org/ko/> 공식 홈페이지에서 설치합니다.

몽고디비는<https://www.mongodb.com/> 공식 홈페이지에서 설치합니다.

웹스톰은 <https://www.jetbrains.com/webstorm/> 공식 홈페이지에서 설치합니다.

**var** app = ***require***(**'express'**)();  
**var** server = ***require***(**'http'**).*createServer*(app);  
**var** io = ***require***(**'socket.io'**)(server)

server.listen(9928,**function** () {  
 ***console***.log(**'Server Running! Good Luck!'**);  
 ***console***.log(**'[210.119.107.159:9928]'**);  
});

첫번째로 서버를 생성합니다. http서버를 생성하고 소켓서버로 업그레이드 합니다.

Post로 넘어오는 값을 사용하기 위해서

app.use(**bodyParser**.**urlencoded**({  
 **extended** : **false**}));

body-parser모듈 설정을 합니다.

io.on(**'connection'**, **function** (socket) {  
 ***console***.log(**'접속'** + socket.**id**);

socket.on(**'disconnect'**, **function** (socket) {  
 ***console***.log(**'접속종료'** + socket.**id**);  
 check = 0;  
 });

});

소켓서버에 접속을 했을 때 동작을 합니다.

소케서버에 접속을 하면 커넥션 이벤트가 발생하고 그 이벤트 안에서 새로운 이벤트를 잡아서 작동합니다.

app.post(**'/set\_user'**, **function** (req,res) {  
 check = 0;  
 **var** inputData;  
  
 req.on(**'data'**, (data) => {  
 inputData = ***JSON***.parse(data);  
 });  
  
 req.on(**'end'**, () => {  
 MongoClient.connect(url,{ **useNewUrlParser**: **true** }, **function**(err, db) {  
 **if** (err) **throw** err;  
 **var** dbo = db.db(**"app\_test"**);  
 user\_name = inputData.USER\_NAME;  
 train\_name\_test = inputData.TRAIN\_NAME;  
 **var** myobj = { **user\_name**: inputData.USER\_NAME, **train\_names**: inputData.TRAIN\_NAME, **train\_result**: inputData.TRAINSELECT};  
 **var** myquery = { **user\_name**: inputData.USER\_NAME };  
 dbo.collection(**"train"**).deleteOne(myquery, **function**(err, obj) {  
 **if** (err) **throw** err;  
 ***console***.log(**"1 document deleted"**);  
 db.close();  
 });  
 dbo.collection(**"train"**).insertOne(myobj, **function**(err, res) {  
 **if** (err) **throw** err  
 ***console***.log(**"디비 저장완료."**);  
 db.close();  
 });  
 });  
 });  
 check = 1;  
 res.write(**"저장완료"**);  
 res.end();  
});

애플리케이션에서 사용자가 지하철역 이름을 지정된 url에 post형식으로 전달하면 서버는 요청된 url에 맞게 이벤트를 실행합니다. 안드로이드에서 서버로 JSON 타입으로 값을 전달하기 때문에 서버에서도 JSON Parse를 이용하여 서버에서 전달받은 값들을 저장합니다. 여기서 사용자가 다시한번 값을 전달하면 업데이트를 하는 것이 아니라 정보를 삭제하고 새로 저장합니다. 정상적으로 작동을 하면 애플리케이션에 완료 메시지를 전달합니다.

socket.on(**'chat'**, **function**(data) {  
 ***console***.log(**'check: '** + check);  
 **if**(check == 1) {  
 ***console***.log(**'확인:'** + user\_name + **","** + train\_name\_test);  
 check = 0;  
 socket.emit(**'train'**, user\_name + **","** + train\_name\_test);  
 }  
});

app.post(**'/set\_page'**, **function** (req,res) {  
 ***console***.log(**"test"**);  
 ***console***.log(req.**body**.**user\_name**);  
 ***console***.log(req.**body**.**train\_names**);  
 ***console***.log(req.**body**.**train\_result**);  
  
 ***console***.log(**"사용자 이름:"** + req.**body**.**user\_name** + **" 열차 이름:"** + req.**body**.**train\_names** + **" 열차선택:"** + req.**body**.**train\_result**);  
  
 MongoClient.connect(url,{ **useNewUrlParser**: **true** }, **function**(err, db) {  
 **if** (err) **throw** err;  
 **var** dbo = db.db(**"app\_test"**);  
 **var** myquery = { **user\_name**: req.**body**.**user\_name** };  
 **var** newvalues = { **$set**: {**train\_result**: req.**body**.**train\_result** } };  
 dbo.collection(**"train"**).updateOne(myquery, newvalues, **function**(err, res) {  
 **if** (err) **throw** err;  
 ***console***.log(**"저장완료.1111"**);  
 db.close();  
 });  
 });  
 res.send(**"ok"**);  
 socket.broadcast.emit(**'android\_call'**, **'1'**);  
});

사용자가 정보를 등록하면 서버에서 인공지능 서버에 실시간으로 정보를 전달하여 메시지를 받은 인공지능 서버에서 전달 받은 정보를 가지고 학습을 하고 그 결과를 특정 url에 post형식으로 값을 전달하여 서버에서 데이터베이스에 업데이트합니다 데이터베이스 업데이트 후 서버에서 애플리케이션에게 메시지를 전송합니다.

app.post(**'/get\_page'**, **function** (req,res) {  
 MongoClient.connect(url,{ **useNewUrlParser**: **true** }, **function**(err, db) {  
 **if** (err) **throw** err;  
 **var** dbo = db.db(**"app\_test"**);  
 **var** query = { **user\_name** : user\_name };  
 dbo.collection(**"train"**).**find**(query).toArray(**function**(err, result) {  
 ***console***.log(result[0].**train\_result**);  
 res.write(result[0].**train\_result**);  
 res.end();  
 db.close();  
 });  
 });  
});

애플리케이션이 서버에서 보낸 메시지를 받고 애플리케이션에서 특정 url에 요청을 하면 서버에서 결과값을 애플리케이션에게 전송합니다.