

MQTT Programming 4

2020.05.20

Sang-woo Lee

glutton.leesw@gmail.com



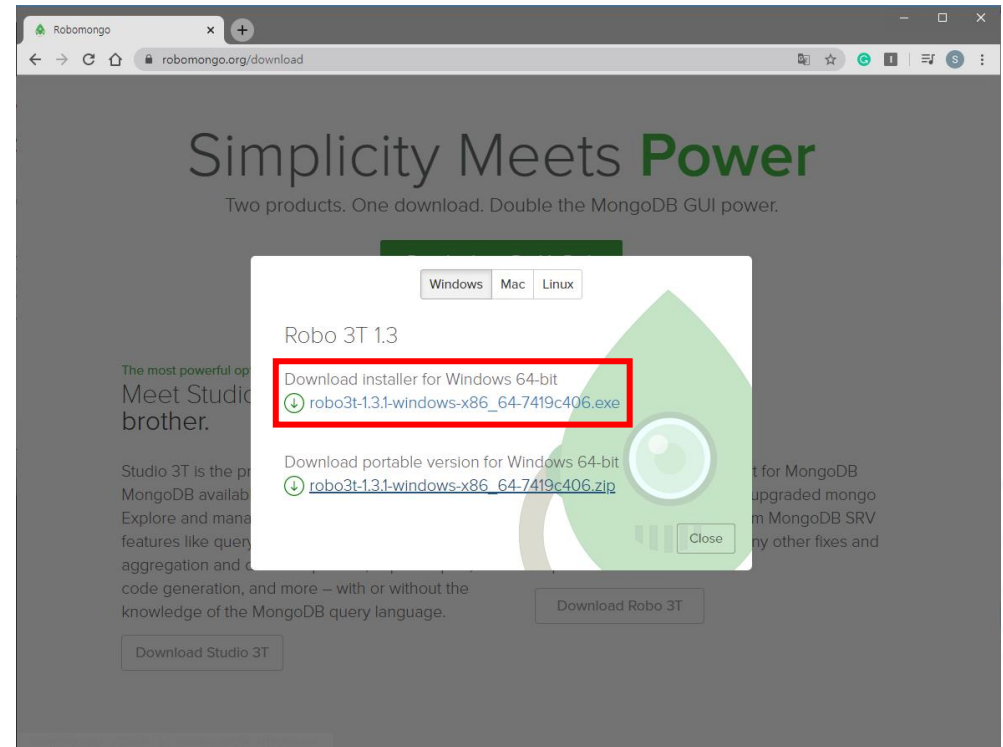
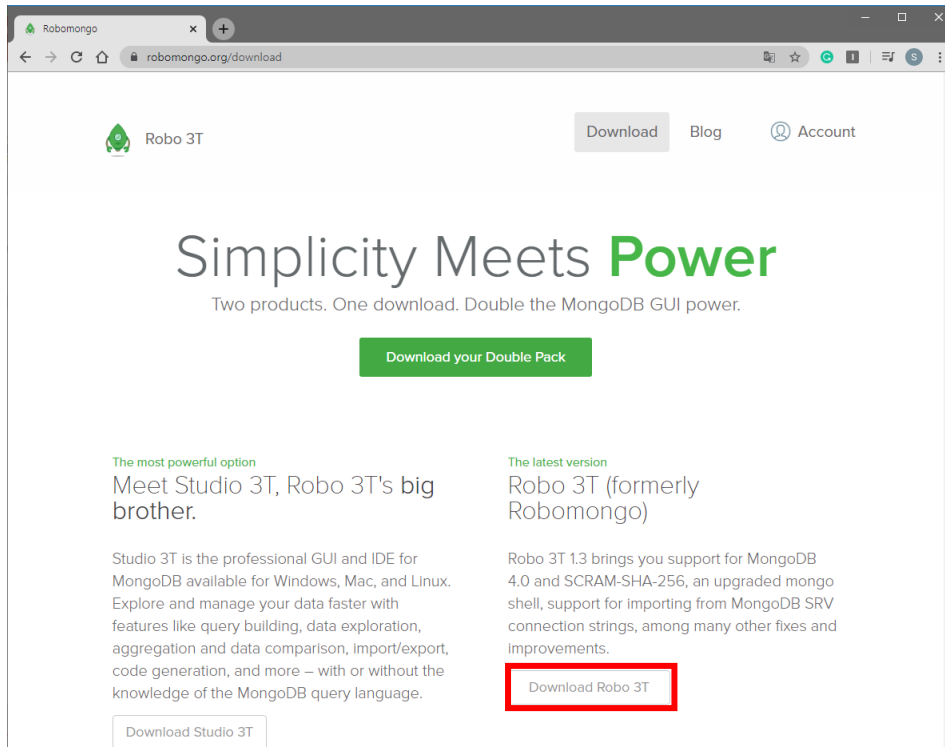
Contents

- Robomongo (Robo 3T) 설치 및 구동
- 센서 데이터를 MongoDB에 저장하기
- 소켓 통신을 이용한 센서 데이터 처리 (Web UI)



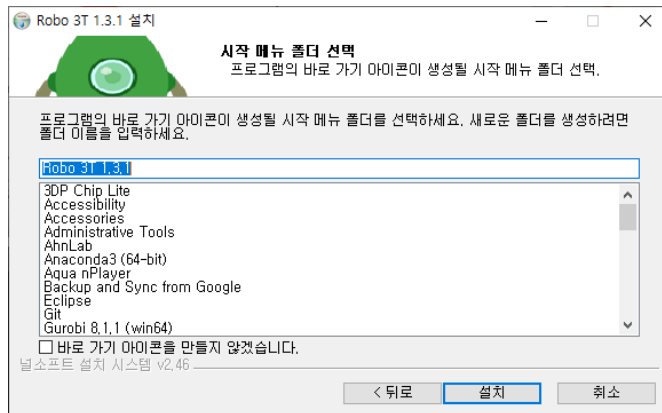
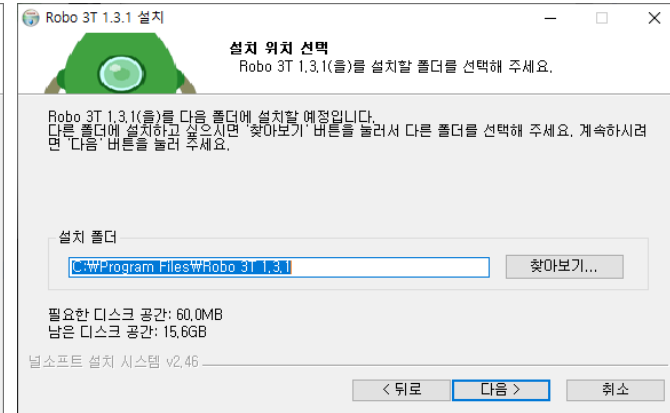
Robomongo (Robo 3T) 설치 및 구동

- Robo 3T 설치
 - MongoDB를 효율적으로 사용하기 위해 GUI 기반의 접속 프로그램인 Robomongo (Robo 3T) 설치



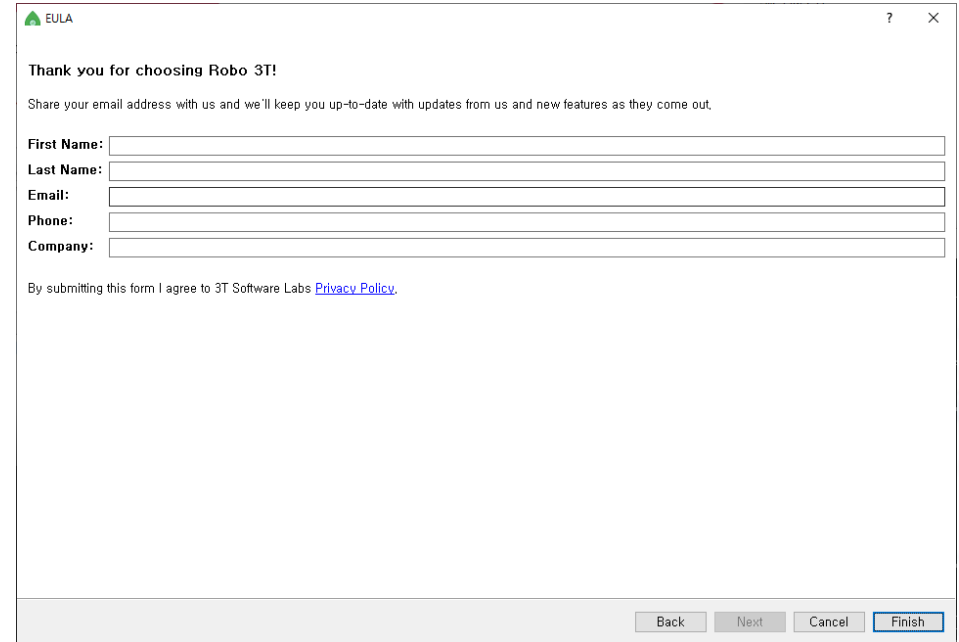
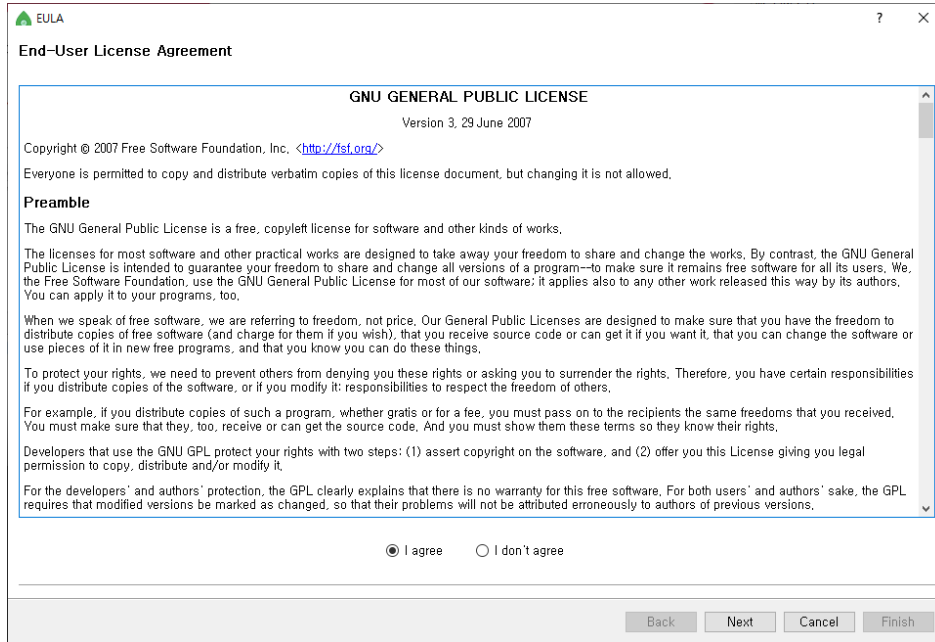
Robomongo (Robo 3T) 설치 및 구동

- Robo 3T 설치



Robomongo (Robo 3T) 설치 및 구동

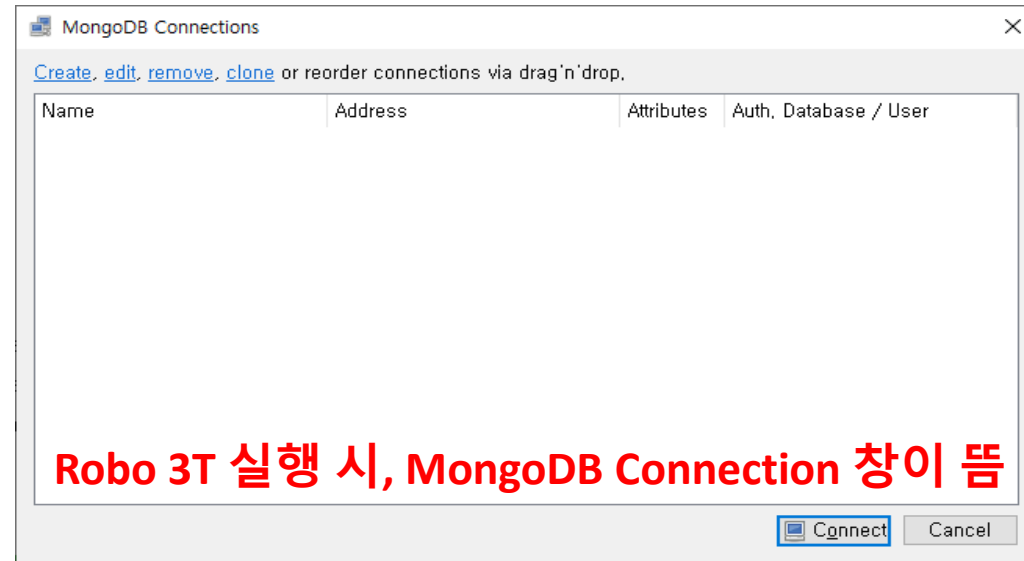
- Robo 3T 설치



Robomongo (Robo 3T) 설치 및 구동

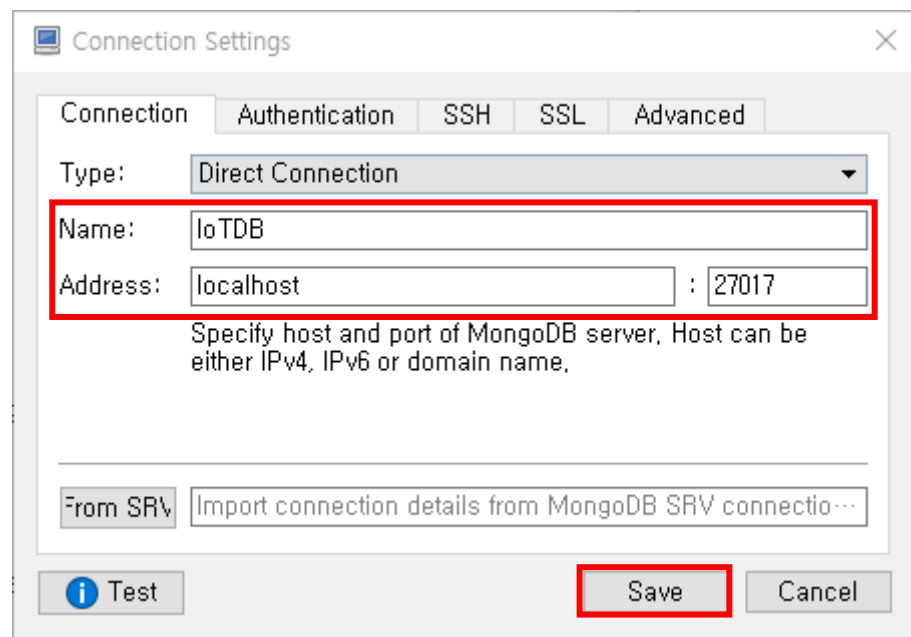
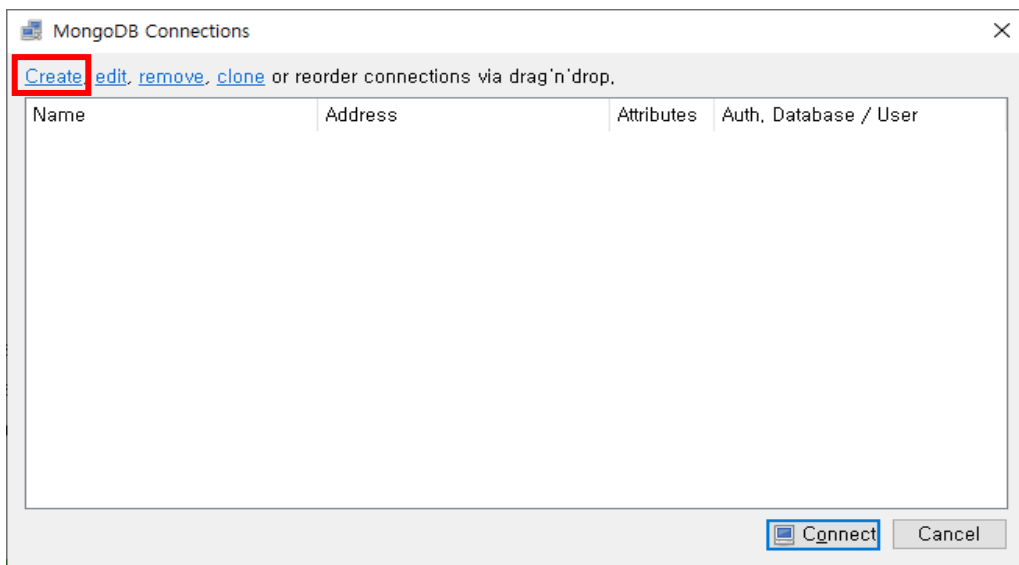
- Robo 3T 구동 및 DataBase 생성
 - 먼저, MongoDB 서버를 구동시킴
 - MongoDB 서버 구동이 완료된 후, Robo 3T를 구동시킴

```
명령 프롬프트 - mongod --dbpath C:\MQTTProject\mongodb\var
C:\MQTTProject\mongodb\bin> mongod --dbpath C:\MQTTProject\mongodb\var
2020-05-25T16:41:10.257+0900 | CONTROL | [initandlisten] Automatically disabling TLS 1.0, to force-enable TLS
1.0 specify --sslDisabledProtocols 'none'
2020-05-25T16:41:10.261+0900 | CONTROL | [initandlisten] MongoDB starting : pid=17588 port=27017 dbp
ath=C:\MQTTProject\mongodb\var 64-bit host=DESKTOP-G17TLUM
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] targetMinOS: Windows 7/Windows Server 2008
R2
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] db version v4.0.18
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] git version: 6883bdfb8b8cff32176b1fd176df04
da9165fd67
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] allocator: tcmalloc
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] modules: none
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] build environment:
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] distmod: 2008plus-ssl
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] distarch: x86_64
2020-05-25T16:41:10.263+0900 | CONTROL | [initandlisten] target_arch: x86_64
2020-05-25T16:41:10.263+0900 | CONTROL | [initandlisten] options: { storage: { dbPath: "C:\MQTTProje
ct\mongodb\var" } }
2020-05-25T16:41:10.270+0900 | STORAGE | [initandlisten] Detected data files in C:\MQTTProject\mongod
b\var created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTi
ger'
2020-05-25T16:41:10.272+0900 | STORAGE | [initandlisten] wiredtiger_open config: create,cache_size=7
649M,cache_overflow=(file_max=0M),session_max=20000,eviction=(threads_min=4,threads_max=4),config_b
ase=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_mana
ger=(close_idle_time=100000),statistics_log=(wait=0),verbose=(recovery_progress)
2020-05-25T16:41:10.638+0900 | STORAGE | [initandlisten] WiredTiger message [1590392470:637745][1758
8:140726690665056], txn-recover: Main recovery loop: starting at 4/4736 to 5/256
2020-05-25T16:41:10.809+0900 | STORAGE | [initandlisten] WiredTiger message [1590392470:802650][1758
8:140726690665056], txn-recover: Recovering log 4 through 5
```



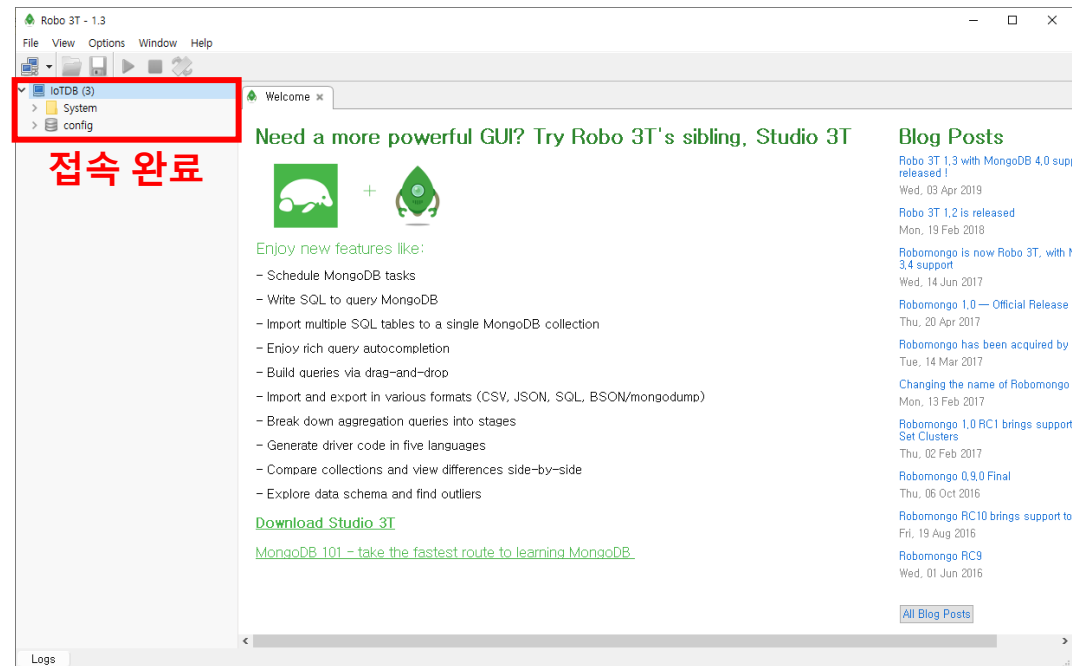
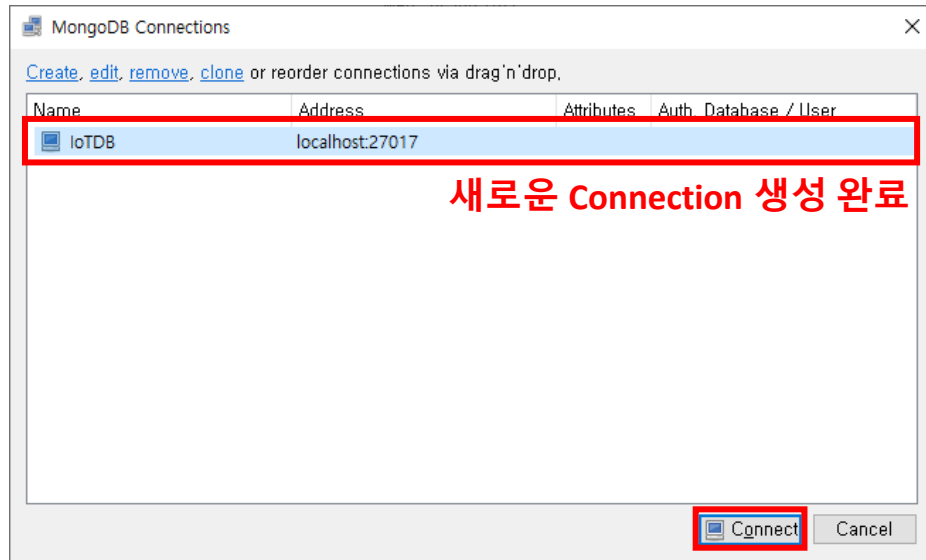
Robomongo (Robo 3T) 설치 및 구동

- Robo 3T 구동 및 DataBase 생성
 - Create를 눌러 Connection을 생성함
 - Name을 IoTDB로 입력함
 - Address는 localhost로 설정하고, 포트번호는 기본값인 27017 를 그대로 사용함
 - 이후, Save 버튼을 눌러 저장함



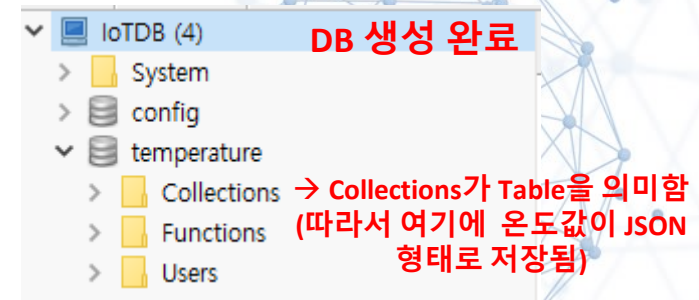
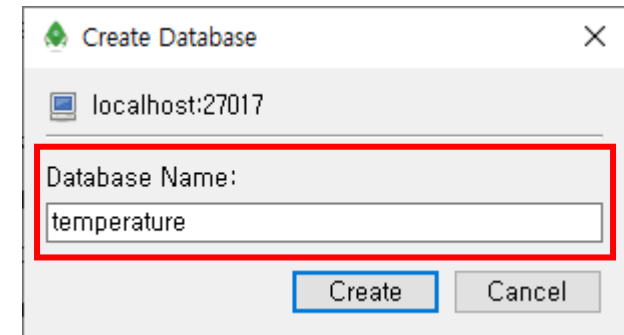
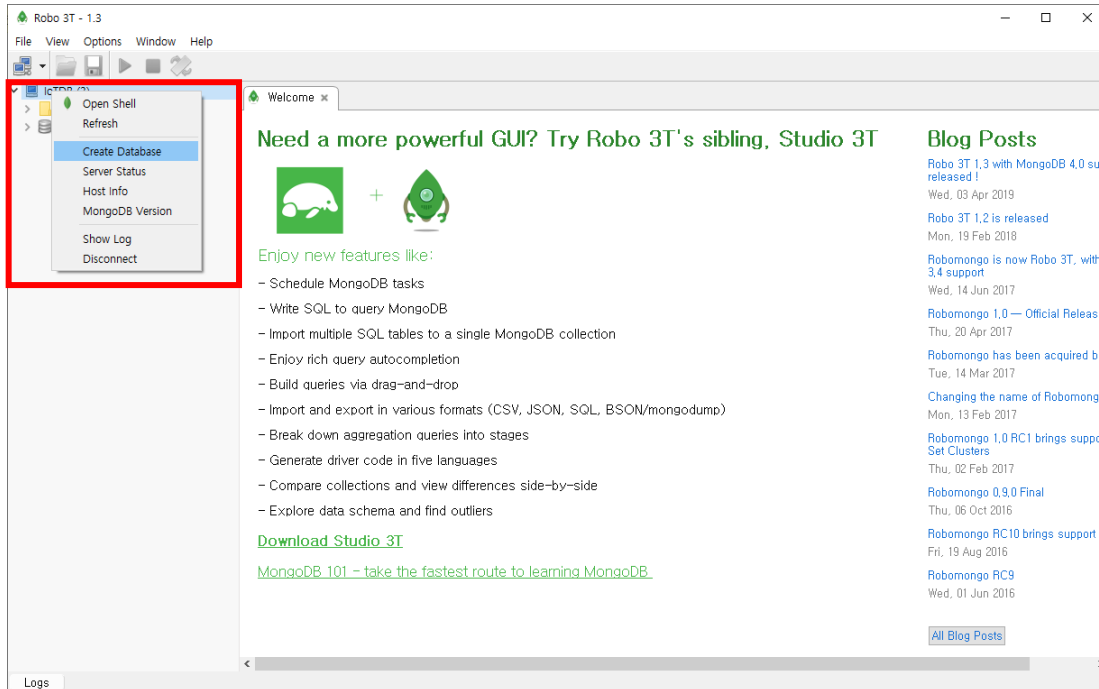
Robomongo (Robo 3T) 설치 및 구동

- Robo 3T 구동 및 DataBase 생성
 - IoTDB connection 생성이 완료되면, Connect를 통해 MongoDB에 접속함
 - 접속이 완료되면 Robo 3T 왼쪽 상단에 IoTDB의 구조 확인 가능



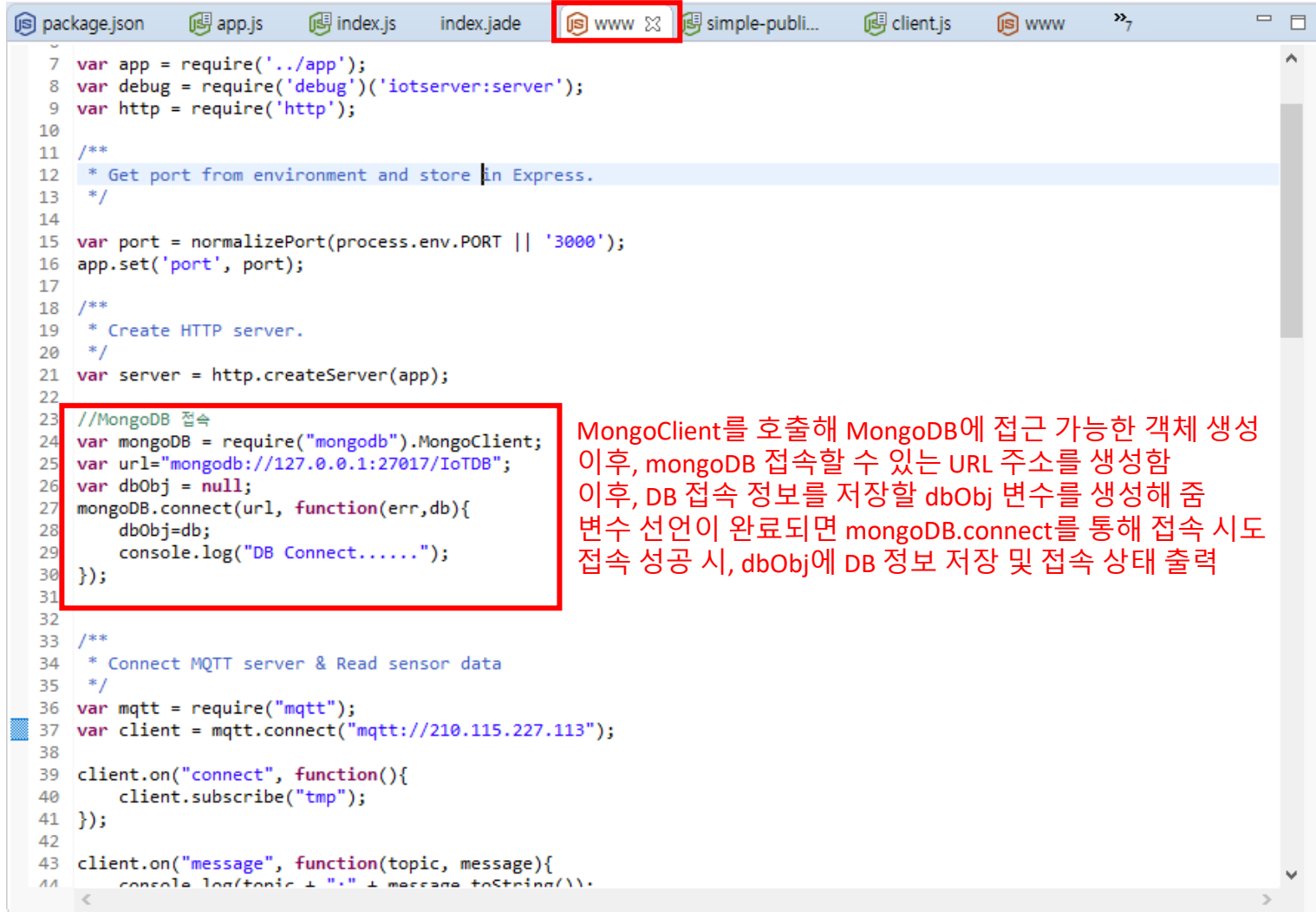
Robomongo (Robo 3T) 설치 및 구동

- Robo 3T 구동 및 DataBase 생성
 - IoTDB 우클릭 → Create Database 를 클릭하여 Database를 생성함
 - Database Name을 temperature로 입력해 DB를 생성함



센서 데이터를 MongoDB에 저장하기

- 센서 데이터를 MongoDB에 저장하기
 - Eclipse의 IoTServer 프로젝트의 www 파일에 MongoDB 접속 코드를 추가함



```
package.json app.js index.js index.jade www simple-publi... client.js www
7 var app = require('../app');
8 var debug = require('debug')('iotserver:server');
9 var http = require('http');
10
11 /**
12  * Get port from environment and store in Express.
13  */
14
15 var port = normalizePort(process.env.PORT || '3000');
16 app.set('port', port);
17
18 /**
19  * Create HTTP server.
20  */
21 var server = http.createServer(app);
22
23 //MongoDB 접속
24 var mongoDB = require("mongodb").MongoClient;
25 var url="mongodb://127.0.0.1:27017/IoTDB";
26 var dbObj = null;
27 mongoDB.connect(url, function(err,db){
28     dbObj=db;
29     console.log("DB Connect.....");
30 });
31
32
33 /**
34  * Connect MQTT server & Read sensor data
35  */
36 var mqtt = require("mqtt");
37 var client = mqtt.connect("mqtt://210.115.227.113");
38
39 client.on("connect", function(){
40     client.subscribe("tmp");
41 });
42
43 client.on("message", function(topic, message){
44     console.log(topic + " " + message.toString());
45 });
```

MongoClient를 호출해 MongoDB에 접근 가능한 객체 생성
이후, mongoDB 접속할 수 있는 URL 주소를 생성함
이후, DB 접속 정보를 저장할 dbObj 변수를 생성해 줌
변수 선언이 완료되면 mongoDB.connect를 통해 접속 시도
접속 성공 시, dbObj에 DB 정보 저장 및 접속 상태 출력

센서 데이터를 MongoDB에 저장하기

- 센서 데이터를 MongoDB에 저장하기
 - MongoDB에 센서 데이터 저장을 위한 코드를 추가함



```
22
23 //MongoDB 접속
24 var mongoDB = require("mongodb").MongoClient;
25 var url="mongodb://127.0.0.1:27017/IoTDB";
26 var dbObj = null;
27 mongoDB.connect(url, function(err,db){
28   dbObj=db;
29   console.log("DB Connect.....");
30 });
31
32
33 /**
34  * Connect MQTT server & Read sensor data
35  */
36 var mqtt = require("mqtt");
37 var client = mqtt.connect("mqtt://210.115.227.113");
38
39 client.on("connect", function(){
40   client.subscribe("tmp");
41 });
42
43 client.on("message", function(topic, message){
44   console.log(topic + ":" + message.toString());
45
46   var obj=JSON.parse(message);
47   obj.create_at = new Date();
48   console.log(obj);
49
50   //MongoDB에 센서 데이터 저장
51   var temperature = dbObj.collection("temperature");
52   temperature.save(obj, function(err, result){
53     if (err)
54       console.log(err);
55     else
56       console.log(JSON.stringify(result)); //{"n":1, "ok":1}
57   });
58
59 });
60
61 /**
```

테이블을 가져오기 위해 테이블 객체 생성
이후, 센서로 부터 읽어온 JSON 형태의 데이터 저장
접속 성공 및 에러에 대한 구문을 출력 조건 추가

센서 데이터를 MongoDB에 저장하기

- MongoDB에 센서 데이터 값이 저장되는지 확인하기
 - MongoDB 구동 및 Mosquitto 서버 구동

```
명령 프롬프트 - mongod --dbpath C:\MQTTProject\mongodb\var
C:\MQTTProject\mongodb\bin>mongod --dbpath C:\MQTTProject\mongodb\var
2020-05-25T16:41:10.257+0900 | CONTROL | [initandlisten] Automatically disabling TLS 1.0, to force-enable TLS
1.0 specify --sslDisabledProtocols 'none'
2020-05-25T16:41:10.261+0900 | CONTROL | [initandlisten] MongoDB starting : pid=17588 port=27017 dbp
ath=C:\MQTTProject\mongodb\var 64-bit host=DESKTOP-G17TUM
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] MongoDB Windows Server 2008
R2
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] db version v4.0.18
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] git version: 6883bdfb8b8cfff32176b1fd176df04
da9185fd67
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] allocator: tcmalloc
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] modules: none
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten] build environment:
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten]   distmod: 2008plus-
2020-05-25T16:41:10.262+0900 | CONTROL | [initandlisten]   distarch: x86_64
2020-05-25T16:41:10.263+0900 | CONTROL | [initandlisten]   target_arch: x86_64
2020-05-25T16:41:10.263+0900 | CONTROL | [initandlisten] options: { storage: {
  c:\mongodb\var" } }
2020-05-25T16:41:10.270+0900 | STORAGE | [initandlisten] Detected data files in
db\var created by the 'wiredTiger' storage engine, so setting the active stora
ger.
2020-05-25T16:41:10.272+0900 | STORAGE | [initandlisten] wiredtiger_open config
649M, cache_overflow=(file_max=0M), session_max=20000, eviction=(threads_min=4, th
ase=false, statistics=(fast), log=(enabled=true, archive=true, path=journal, compre
ager=(close_idle_time=100000), statistics_log=(wait=0), verbose=(recovery_progre
2020-05-25T16:41:10.638+0900 | STORAGE | [initandlisten] WiredTiger message [15
8:140726690665056], txn-recover: Main recovery loop: starting at 4/4736 to 5/2
2020-05-25T16:41:10.803+0900 | STORAGE | [initandlisten] WiredTiger message [15
8:140726690665056], txn-recover: Recovering log 4 through 5
```

MongoDB 서버 구동

```
명령 프롬프트 - mosquitto -v
C:\MQTTProject\sw\mosquitto>mosquitto -v
1590396778: mosquitto version 1.6.3 starting
1590396778: Using default config.
1590396778: Opening ipv6 listen socket on port 1883.
1590396778: Opening ipv4 listen socket on port 1883.
1590396779: New connection from 210.115.227.113 on port 1883.
1590396779: New client connected from 210.115.227.113 as mattjs_81f65b70 (p2, c1, k60).
1590396779: No will message specified.
1590396779: Sending CONNACK to mattjs_81f65b70 (0, 0)
1590396779: Received SUBSCRIBE from mattjs_81f65b70
1590396779:   tmp (QoS 0)
1590396779: mattjs_81f65b70 0 tmp
1590396779: Sending SUBACK to mattjs_81f65b70
```

Mosquitto 서버 구동

센서 데이터를 MongoDB에 저장하기

- MongoDB에 센서 데이터 값이 저장되는지 확인하기
 - Mosquitto Subscriber 실행 및 IoTServer 실행

```
명령 프롬프트 - mosquitto_sub -t tmp -p 1883
Microsoft Windows [Version 10.0.17763.1217]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\SolBee Lee>cd C:\

Mosquitto Subscriber 실행

C:\>cd MQTTProject

C:\MQTTProject>cd sw

C:\MQTTProject\sw>cd mosquitto

C:\MQTTProject\sw\mosquitto>mosquitto_sub -t tmp -p 1883
```

```
eclipse-workspace - IoTServer/bin/www - Eclipse IDE
File Edit Navigate Search Project Run Emmet Window Help

Project Explorer
  applications
  IoTServer
    JavaScript Resources
    bin
      www
    node_modules
    public
    routes
    views
    app.js
    package.json
    README.md
  IoTServer2
    JavaScript Resources
    bin
    www
    node_modules
    public
    routes
    views
    app.js
    package.json
    README.md
  LEGO_User
  SmartCampus
  swing-example
  ws4d-jcoap
  ws4d-jcoap-applications

package.json
app.js
index.js
index.jade
www
simple-publi...
client.js
www

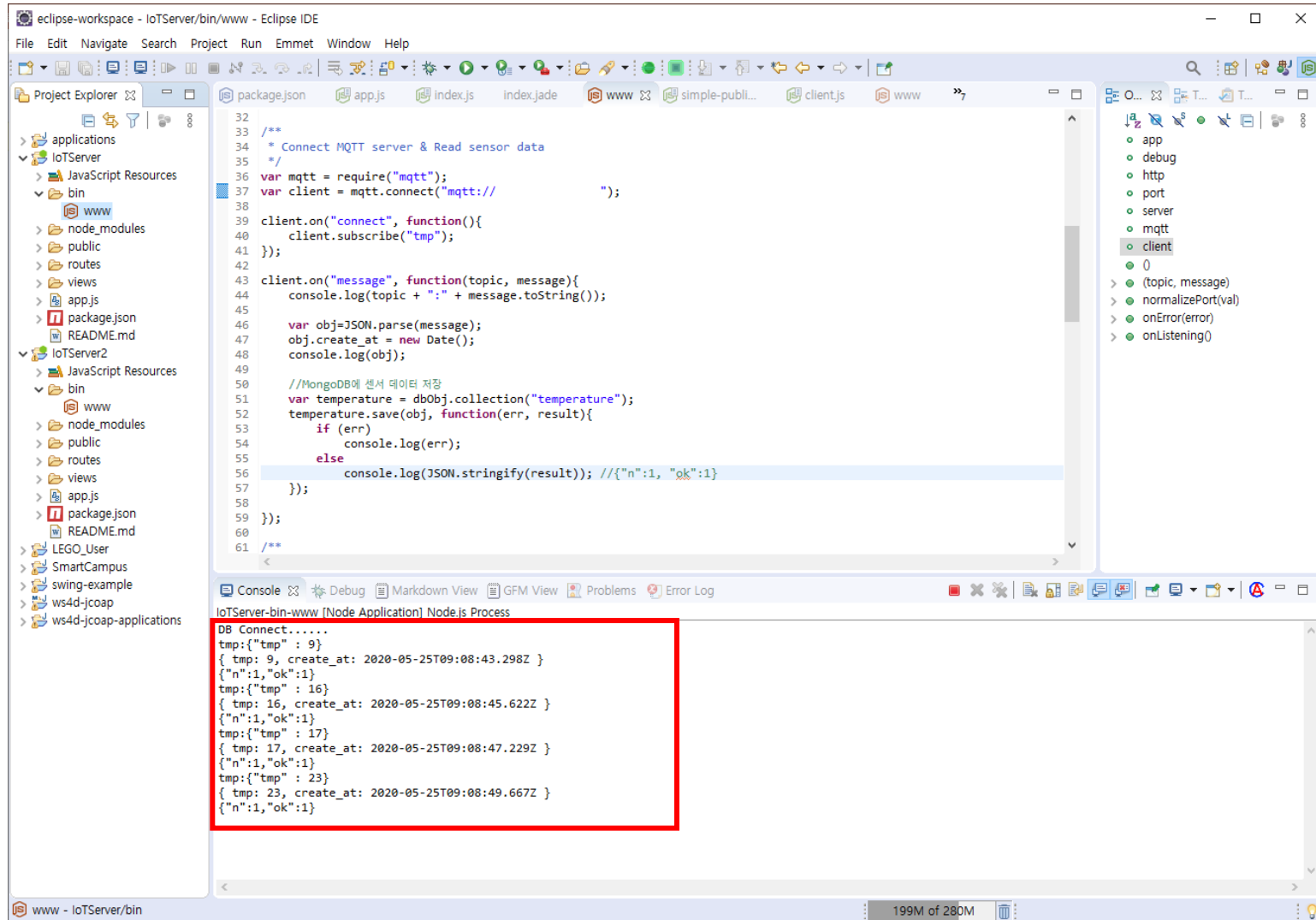
22 //MongoDB
23 var mongoDB = require("mongodb").MongoClient;
24 var url="mongodb://127.0.0.1:27017/IoTDB";
25 var dbObj = null;
26 mongoDB.connect(url, function(err,db){
27   dbObj=db;
28   console.log("DB Connect.....");
29 });
30
31
32
33 /**
34  * Connect MQTT server & Read sensor data
35  */
36 var mqtt = require("mqtt");
37 var client = mqtt.connect("mqtt://210.115.227.113");
38
39 client.on("connect", function(){
40   client.subscribe("tmp");
41 });
42
43 client.on("message", function(topic, message){
44   console.log(topic + ":" + message.toString());
45
46   var obj=JSON.parse(message);
47   obj.create_at = new Date();
48   console.log(obj);
49
50   //MongoDB에 센서 데이터 저장
51   var temperature = dbObj.collection("temperature");

Console
IoTServer-bin-www [Node Application] Node is Process
DB Connect.....

IoTServer - Run As로 실행
→ DB Connect..... 출력됨 (DB 연결 성공)
```

센서 데이터를 MongoDB에 저장하기

- MongoDB에 센서 데이터 값이 저장되는지 확인하기
 - Eclipse의 콘솔 창에도 온도 값이 출력되어 해당 값이 성공적으로 DB에 저장되는 것을 확인



The screenshot shows the Eclipse IDE interface. The left sidebar displays the Project Explorer with a tree view of the project structure, including folders like 'applications', 'IoTServer', and 'IoTServer2'. The main editor window shows the source code for 'www.js', which is a JavaScript file for connecting to an MQTT server and saving sensor data to MongoDB. The code includes comments in Korean and uses the 'mqtt' library for communication. The console window at the bottom shows the output of the application, which includes a 'DB Connect.....' message followed by several JSON objects representing sensor data. These objects contain 'tmp' (temperature) and 'create_at' (timestamp) fields. The console output is highlighted with a red rectangle.

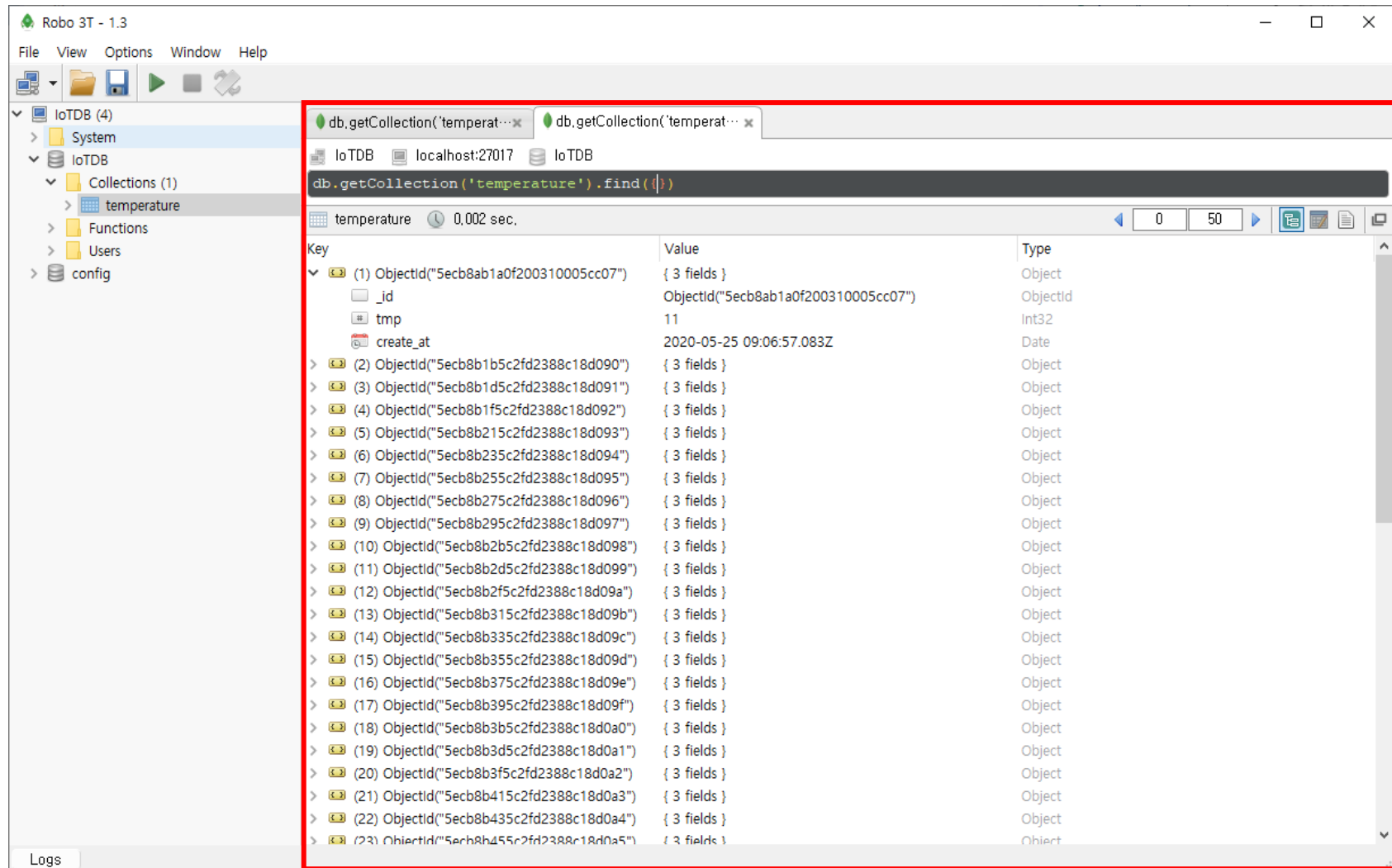
```
32
33
34 /**
35  * Connect MQTT server & Read sensor data
36  */
37 var mqtt = require("mqtt");
38 var client = mqtt.connect("mqtt://");
39
40 client.on("connect", function(){
41   client.subscribe("tmp");
42 });
43
44 client.on("message", function(topic, message){
45   console.log(topic + ":" + message.toString());
46
47   var obj=JSON.parse(message);
48   obj.create_at = new Date();
49   console.log(obj);
50
51   //MongoDB에 센서 데이터 저장
52   var temperature = dbObj.collection("temperature");
53   temperature.save(obj, function(err, result){
54     if (err)
55       console.log(err);
56     else
57       console.log(JSON.stringify(result)); //{"n":1, "ok":1}
58   });
59 });
60
61 /**
```

Console Output:

```
DB Connect.....
tmp:{"tmp" : 9}
{ tmp: 9, create_at: 2020-05-25T09:08:43.298Z }
{"n":1,"ok":1}
tmp:{"tmp" : 16}
{ tmp: 16, create_at: 2020-05-25T09:08:45.622Z }
{"n":1,"ok":1}
tmp:{"tmp" : 17}
{ tmp: 17, create_at: 2020-05-25T09:08:47.229Z }
{"n":1,"ok":1}
tmp:{"tmp" : 23}
{ tmp: 23, create_at: 2020-05-25T09:08:49.667Z }
{"n":1,"ok":1}
```

센서 데이터를 MongoDB에 저장하기

- MongoDB에 센서 데이터 값이 저장되는지 확인하기
 - Robo 3T를 이용해 DB 확인하기

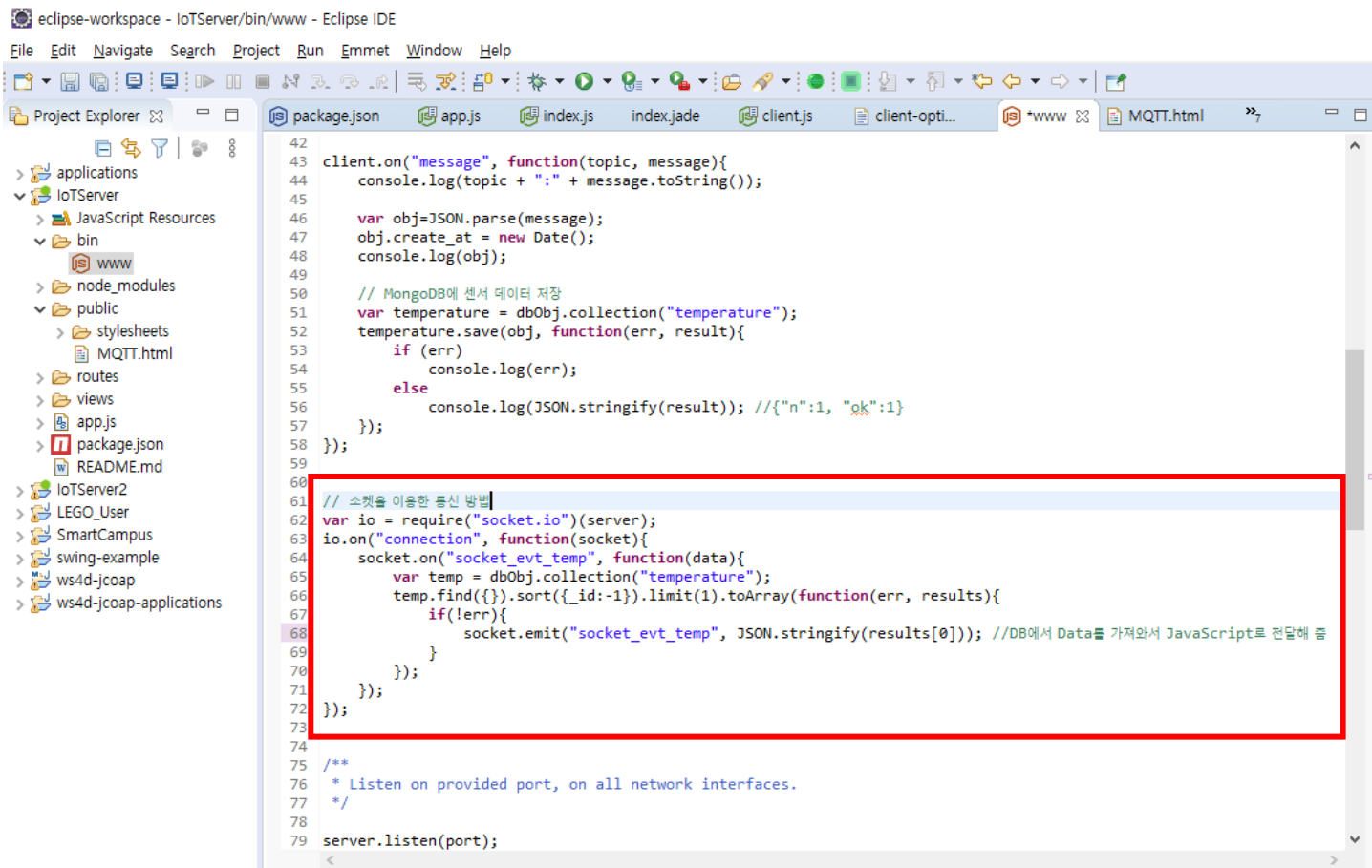


The screenshot shows the Robo 3T - 1.3 interface. On the left, the 'IoTDB (4)' database is expanded, showing the 'temperature' collection. The main window displays the results of a query: `db.getCollection('temperature').find({})`. The results are shown in a table with columns 'Key', 'Value', and 'Type'. The first document is expanded, showing fields: `_id` (ObjectId), `tmp` (Int32), and `create_at` (Date). The table lists 23 documents, each with a unique ObjectId and a timestamp.

Key	Value	Type
(1) ObjectId("5ecb8ab1a0f200310005cc07")	{ 3 fields }	Object
_id	ObjectId("5ecb8ab1a0f200310005cc07")	ObjectId
tmp	11	Int32
create_at	2020-05-25 09:06:57.083Z	Date
(2) ObjectId("5ecb8b1b5c2fd2388c18d090")	{ 3 fields }	Object
(3) ObjectId("5ecb8b1d5c2fd2388c18d091")	{ 3 fields }	Object
(4) ObjectId("5ecb8b1f5c2fd2388c18d092")	{ 3 fields }	Object
(5) ObjectId("5ecb8b215c2fd2388c18d093")	{ 3 fields }	Object
(6) ObjectId("5ecb8b235c2fd2388c18d094")	{ 3 fields }	Object
(7) ObjectId("5ecb8b255c2fd2388c18d095")	{ 3 fields }	Object
(8) ObjectId("5ecb8b275c2fd2388c18d096")	{ 3 fields }	Object
(9) ObjectId("5ecb8b295c2fd2388c18d097")	{ 3 fields }	Object
(10) ObjectId("5ecb8b2b5c2fd2388c18d098")	{ 3 fields }	Object
(11) ObjectId("5ecb8b2d5c2fd2388c18d099")	{ 3 fields }	Object
(12) ObjectId("5ecb8b2f5c2fd2388c18d09a")	{ 3 fields }	Object
(13) ObjectId("5ecb8b315c2fd2388c18d09b")	{ 3 fields }	Object
(14) ObjectId("5ecb8b335c2fd2388c18d09c")	{ 3 fields }	Object
(15) ObjectId("5ecb8b355c2fd2388c18d09d")	{ 3 fields }	Object
(16) ObjectId("5ecb8b375c2fd2388c18d09e")	{ 3 fields }	Object
(17) ObjectId("5ecb8b395c2fd2388c18d09f")	{ 3 fields }	Object
(18) ObjectId("5ecb8b3b5c2fd2388c18d0a0")	{ 3 fields }	Object
(19) ObjectId("5ecb8b3d5c2fd2388c18d0a1")	{ 3 fields }	Object
(20) ObjectId("5ecb8b3f5c2fd2388c18d0a2")	{ 3 fields }	Object
(21) ObjectId("5ecb8b415c2fd2388c18d0a3")	{ 3 fields }	Object
(22) ObjectId("5ecb8b435c2fd2388c18d0a4")	{ 3 fields }	Object
(23) ObjectId("5ecb8b455c2fd2388c18d0a5")	{ 3 fields }	Object

소켓 통신을 이용한 센서 데이터 처리 (Web UI)

- 소켓 (Socket)을 이용한 JavaScript와의 통신
 - Node.js로 구현된 웹 서버와 웹 페이지 간의 소켓 통신을 위해 기존 웹 서버가 사용하고 있는 포트 번호 3000번을 사용하여 구현함
 - 이를 위해 IoTServer 프로젝트의 www 파일을 수정함



```
eclipse-workspace - IoTServer/bin/www - Eclipse IDE
File Edit Navigate Search Project Run Emmet Window Help

Project Explorer
  applications
  IoTServer
    JavaScript Resources
    bin
      www
    node_modules
    public
      stylesheets
      MQTT.html
    routes
    views
    app.js
    package.json
    README.md
  IoTServer2
  LEGO_User
  SmartCampus
  swing-example
  ws4d-jcoap
  ws4d-jcoap-applications

package.json app.js index.js index.jade client.js client-opti... www MQTT.html

42
43 client.on("message", function(topic, message){
44   console.log(topic + ":" + message.toString());
45
46   var obj=JSON.parse(message);
47   obj.create_at = new Date();
48   console.log(obj);
49
50   // MongoDB에 센서 데이터 저장
51   var temperature = dbObj.collection("temperature");
52   temperature.save(obj, function(err, result){
53     if (err)
54       console.log(err);
55     else
56       console.log(JSON.stringify(result)); //{"n":1, "ok":1}
57   });
58 });
59
60
61 // 소켓을 이용한 통신 방법
62 var io = require("socket.io")(server);
63 io.on("connection", function(socket){
64   socket.on("socket_evt_temp", function(data){
65     var temp = dbObj.collection("temperature");
66     temp.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){
67       if(!err){
68         socket.emit("socket_evt_temp", JSON.stringify(results[0])); //DB에서 Data를 가져와서 JavaScript로 전달해 줄
69       }
70     });
71   });
72 });
73
74
75 /**
76  * Listen on provided port, on all network interfaces.
77  */
78
79 server.listen(port);
```


소켓 통신을 이용한 센서 데이터 처리 (Web UI)

- 소켓 (Socket)을 이용한 JavaScript와의 통신
 - io는 웹 서버의 3000번 포트와 연결된 소켓을 의미함
 - 소켓 통신은 이벤트 방식으로 동작함
 - 소켓 생성 후, connection 이벤트 발생 시, 이 때 수행되는 함수는 socket 객체를 매개변수로 받음
 - ✓ 이 때의 socket도 3000번 포트를 가리킴
 - Connection 시, 함수 내에서 socket.on()이라는 이벤트 등록 함수를 사용하여 socket_evt_temp 이벤트를 등록함
 - 해당 이벤트가 socket을 통해 넘어오면 data를 매개변수로 받는 함수가 수행됨
 - Data를 매개변수로 받는 함수에서는 DB에 연결하여 table 정보인 collection을 가져옴
 - 이후, table의 모든 데이터를 id 기준 내림차순으로 정렬하여 배열 형식으로 값을 results에 담음
 - ✓ 맨 아래의 값이 최신의 데이터이기 때문
 - 에러가 발생하지 않으면, socket.emit() 를 통해 해당 이벤트에 데이터를 담아 3000번 포트에 전송

// 소켓을 이용한 통신 방법

```
var io = require("socket.io")(server);  
io.on("connection", function(socket){  
  socket.on("socket_evt_temp", function(data){  
    var temp = dbObj.collection("temperature");  
    temp.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){  
      if(!err){  
        socket.emit("socket_evt_temp", JSON.stringify(results[0]));  
      }  
    });  
  });  
});
```

io는 3000번 포트와 연결된 소켓

Connection 이벤트 발생 시 socket을 매개변수로 받는 함수 수행

socket_evt_temp 이벤트 등록

DB 연동 및 테이블 가져오기

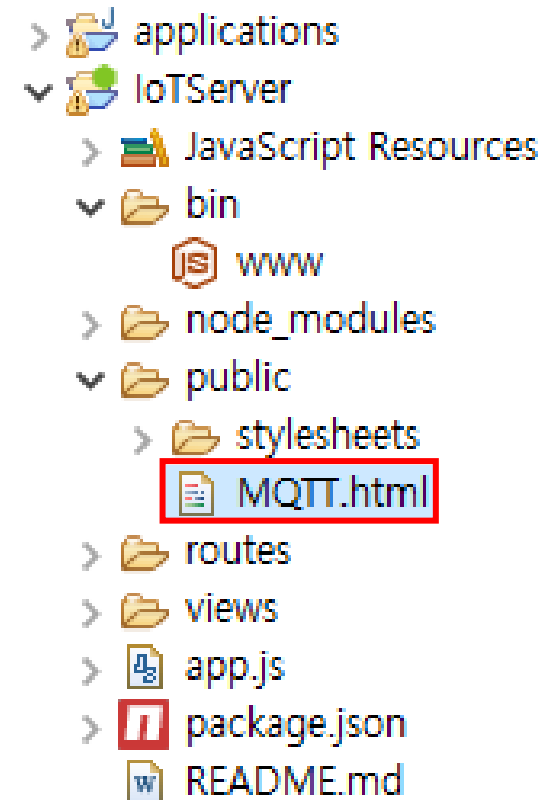
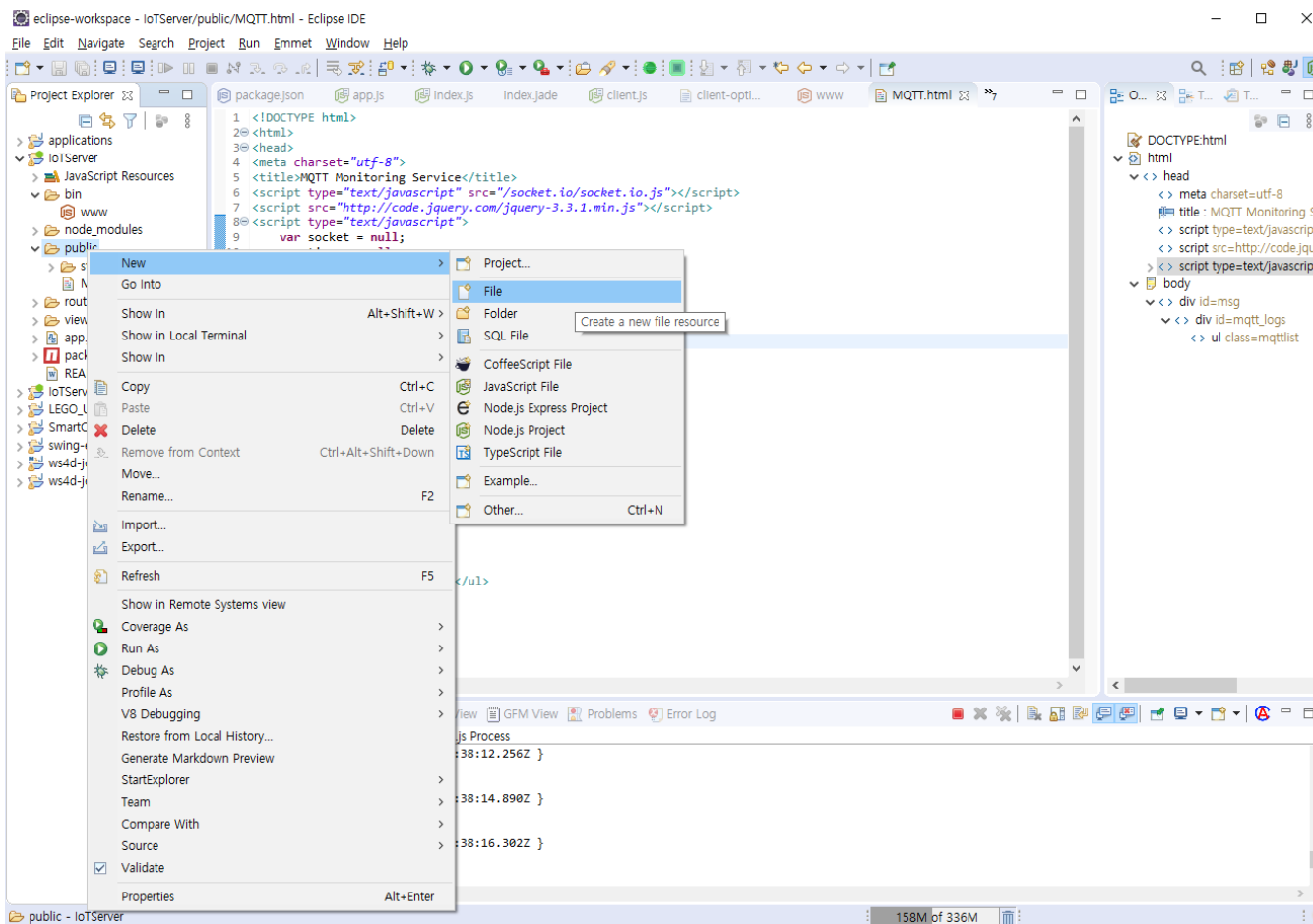
테이블 값 정렬 후, 하나의 데이터를
배열 형식으로 가져옴

//DB에서 Data를 가져와서 JavaScript로 전달해 줌

에러가 발생하지 않았을 경우, 해당 데이터를
JSON 형식의 문자열로 바꿔 3000번 포트에 전송함

소켓 통신을 이용한 센서 데이터 처리 (Web UI)

- 소켓 (Socket)을 이용한 JavaScript와의 통신
 - 웹 페이지 (html)에서 센서 데이터를 받아 처리하는 부분을 구현함
 - 이를 위해 IoTServer 프로젝트의 public 폴더 아래 MQTT.html 파일을 생성함
 - ✓ Public 폴더 우클릭 – New – File 선택 후, 파일 이름을 MQTT.html로 입력해 생성함



소켓 통신을 이용한 센서 데이터 처리 (Web UI)

- 소켓 (Socket)을 이용한 JavaScript와의 통신
 - 우선적으로 Node.js에서 설치한 socket.io.js라는 JavaScript가 필요함
 - ✓ 해당 모듈 내에 소켓 생성을 위해 필요한 함수들이 존재하기 때문
 - 또한, 구현에 있어 jQuery를 사용하기 위해 jQuery 라이브러리를 불러와야 함
 - ✓ 사용자와 직접적인 상호작용을 하는 프론트엔드 (Front-end)에서 주로 jQuery가 사용됨

```
<!DOCTYPE html>
<html lang="ko">
<meta charset="UTF-8">
<title>MQTT Monitoring Service</title>
<script src="/socket.io/socket.io.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script>
var socket = io.connect();
var timer = null;
$(document).ready(function(){
socket.on("socket_evt_temp", function(data){
data = JSON.parse(data);
$(".mqttlist").html('<li>'+data.tmp+'C'+</li>');
});
if(timer==null){
timer = window.setInterval("timer1()", 1000);
}
});
function timer1(){
socket.emit("socket_evt_temp", JSON.stringify({}));
}
</script>
</head>
<body>
MQTT Mornitoring Service
<div id="msg">
<div id="mqtt_logs">
<ul class="mqttlist"></ul>
</div>
</div>
</body>
</html>
```



소켓 통신을 이용한 센서 데이터 처리 (Web UI)

- 소켓 (Socket)을 이용한 JavaScript와의 통신
 - 먼저, 사용할 socket 변수 (3000번 포트 사용)와 timer 변수를 선언함
 - 이후, jQuery 구문을 통해 document 객체를 불러오고 ready() 함수를 사용하여, html이 준비가 되면 웹 서버와의 connection을 시도함
 - 함수 내에 데이터 수신을 위한 구문을 추가함

```
8<script type="text/javascript">
9  var socket = null;
10 var timer = null;
11 $(document).ready(function(){
12     socket=io.connect(); // Port number: 3000
13     //데이터 수신
14     socket.on("socket_evt_temp", function(data){
15         data=JSON.parse(data);
16         $(".mqttlist").html('<li>'+data.tmp+'</li>');
17     });
18     if(timer==null){
19         timer = window.setInterval("timer1()", 1000);
20     }
21 });
22 function timer1(){
23     socket.emit("socket_evt_temp", JSON.stringify({}));
24     console.log("-----");
25 }
26 </script>
27 </head>
28 <body>
29 MQTT 모니터링 서비스
30 <div id="msg">
31     <div id="mqtt_logs">
32         <ul class="mqttlist"></ul>
33     </div>
34 </div>
35 </body>
36 </html>
```

← 웹 서버와의 connection 시도

← 마찬가지로 socket_evt_temp 이벤트 등록 및 함수 구현

← 문자열로 들어온 data를 JSON 형식의 객체로 변환

← html의 .mqttlist 클래스에 매개변수로 받은 데이터의 온도 값을 넣어 줌

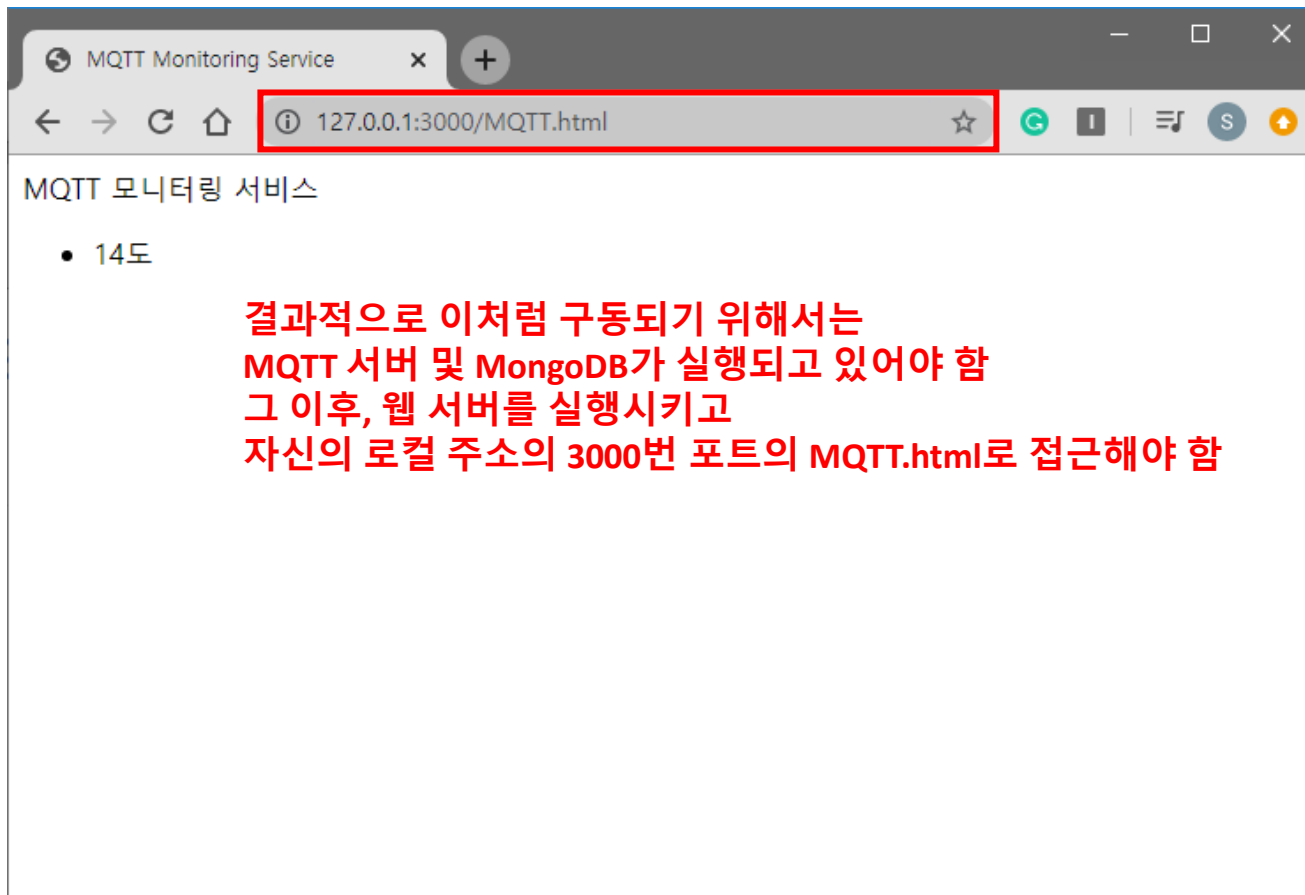
← 만약 timer가 null이라면 setInterval 함수를 사용해 timer1()의 주기를 1초 (1000)로 설정함

timer1() 함수는 1초마다 socket.emit()을 통해 데이터 없이 socket_evt_tmp 이벤트를 웹 서버로 전송함

← 실제 웹 페이지 화면을 구성하는 html 구조를 나타냄 mqttlist 클래스에 센서 데이터 값이 표시되는 구조

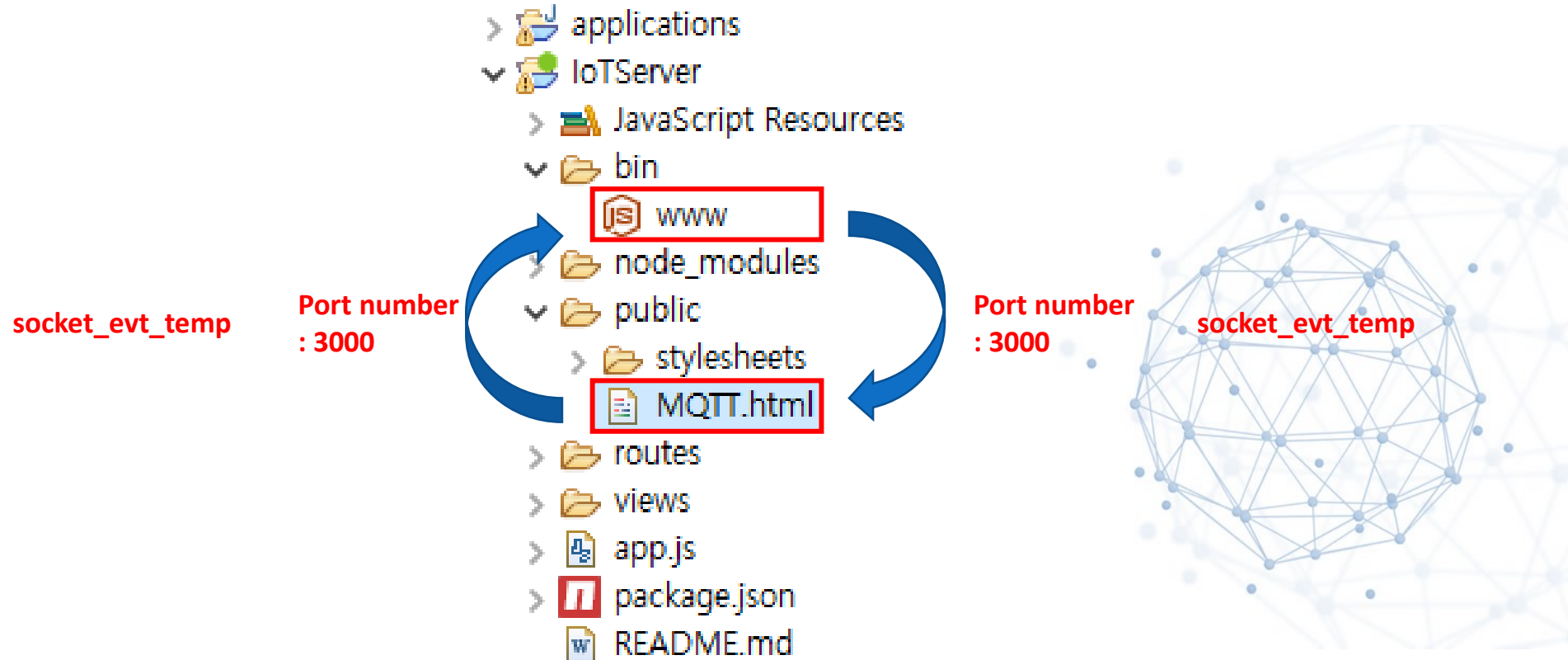
소켓 통신을 이용한 센서 데이터 처리 (Web UI)

- 소켓 (Socket)을 이용한 JavaScript와의 통신
 - 따라서 웹서버는 1초마다 요청되는 socket_evt_temp 이벤트를 수신하여, 이에 해당되는 센서 데이터 값을 웹 페이지 (JavaScript)로 넘겨줌
 - 이를 통해 웹 페이지에 표시되는 센서 데이터 값이 1초마다 업데이트되어 모니터링 서비스를 제공할 수 있게 됨



소켓 통신을 이용한 센서 데이터 처리 (Web UI)

- 소켓 (Socket)을 이용한 JavaScript와의 통신
 - 소켓 통신을 통한 www 파일과 MQTT.html, 즉, 웹 서버와 웹 페이지 간의 센서 데이터 처리 과정
 - socket_evt_temp라는 이벤트를 통해 이벤트에 의해 동작하는 센서 데이터 처리 방식



Thank you

