IoT-network

MQTT-project Report

학과: 빅데이터
학번: 20195275
이름: 한 대 현
담당교수: 김의직
담당조교: 이상우
실험날짜: 2021년 5월 30일
제출날짜: 2021년 5월 30일

한림대학교

목차

01 실험목적 2

02 추진배경 3

03 추진절차 4

04 결론도출 17

05 참고문헌 21

01
실험목적

MQTT는 Publish/Subscribe 방식, 메시지를 전후방으로 전송함으로써 간결함과 직관성을 보장한다.
시스템의 확장이 쉽고 모든 클라이언트의 상태를 관리하고 추적할 수 있다. 또한 네트워크 부하를 낮추며 IoT 기반 통신 네트워크에 적합한 조건이다.
그렇기에 실제로 코드를 짜보면서 mosquitto를 이용해 MQTT 서비스를 이용한 후 적합한 데이터를 사용, 사용자가 원하는 데이터를 출력하게끔 서버를 구축한다.

02 추진배경

늘고있는 확진자 수

위 사진은 국내 코로나 확진 환자 및 격리 해제 인원수를 날짜별로 만든 그래프이다. 사진처럼 국내 코로나 확진 환자는 꾸준히 증가하는 추세이다. 이는 사람들의 경각심 부족, 정보의 접근성 등 여러 문제가 복합적으로 만들어낸 상황이라고 생각, 코로나 정보를 보다 빠르게 확인할 수 있는 서비스를 제공하기 위한 목적으로 시스템을 설계하게 되었다.

03 추진절차

MQTT는 크게 Broker,publisher,subscribe 로 구성된다.
먼저 원하는 데이터베이스를 선택 및 확인하고 그에 알맞는 MQTT Client 서버를 구축한다. 구축된 MQTT Client를 이용해 데이터를 Broker Server에게 원할하게 전달하고 전달된 데이터를 Web Browser에게 다시 전달해 사용자가 데이터를 확인할 수 있게 한다.
기본적으로 mosquitto, mongoDB는 실행이 되는 상태이다.

Java Code

1. Eclipse를 이용해 공공 데이터 포털의 데이터를 불러오는 java code를 만든다.

2. Actuator 작동 code를 설계한다

Node.js Code

1. 기존 Node.js 코드를 VS code 개발환경에서 실행한다.

2. java code에서 불러오는 데이터를 원할한 접속이 가능하도록 Node.js 코드를 수정한다.

3. 데이터를 MongoDB랑 수신하도록 한다.

Html Code

1. Vs code에서 Node.js 파일에 있는 데이터를 읽는다.

2. 읽은 데이터를 사용자가 직접 확인하기 쉽도록 Web page를 만들어 코드를 짠다. (Html)

Java Code

기본적인 자바코드는 다음과 같다.
1. MqttPublisher_API_project 클래스에 콜백 함수를 상속 시킨다.
2. 상속시킨 클래스 내부에 브로커 서버 및 데이터를 발행하는 public void run() 코드를 실행한다.
3. 접속할 브로커 서버의 상세주소 및 클라이언트 id 옵션등이 포함되어 있는 public void connectBroker() 코드를 실행한다.
4. 서울시 사망자,확진자,격리중인 사람 데이터를 포함하고 있는 get_covid19_data함수와 코로나검사 실시기관인 get_covid19hospital_data함수를 실행한다.
5. 라즈베리파이, 아두이노 제어소스코드를 포함하는 Acuator 코드를 실행한다.

아래 코드는 배열로 초기화, split함수를 이용해 문자를 자르면서 출력하게 만들었다.

```
public class MqttPublisher_API_project implements MqttCallback{ //implement callback(상속) 추가& 필요한 메소드 정의
    static MqttClient sampleClient; //Mqtt Client 객체 선언

    public static void main(String[] args) {
        MqttPublisher_API_project obj = new MqttPublisher_API_project();
        obj.run();
    }
    public void run() {
        connectBroker(); // 브로커 서버에 접속
        try {
            sampleClient.subscribe("btn"); //btn topic subscribe
        } catch (MqttException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        // 계속 반복하기 위한 while문
        while(true) {
            try {
                String[] covid19_data  = get_covid19_data(); //보건복지부 API
                String[] covid19hospital_data = get_covid19hospital_data(); // 건강보험심사평가원 API
                // temp
                String[] sutime = covid19_data[0].split(",");
                String[] Death = covid19_data[1].split(",");
                String[] Def = covid19_data[2].split(",");
                String[] Isol = covid19_data[3].split(",");

                for (int i = 0; i < sutime.length; i++) {
                    System.out.println("-------------------------------------------");
                    publish_data("sutime", "{\"DateAndTime\": "+sutime[i]+"}"); // 데이터를 확인한 시간 데이터 발행
                    publish_data("Death", "{\"deceasedPatient\": "+Death[i]+"}"); // 사망자 수 데이터  발행
                    publish_data("Def", "{\"confirmedPatient\": "+Def[i]+"}"); //확진자 수 데이터 발행
                    publish_data("Isol", "{\"QuarantinePatient\": "+Isol[i]+"}"); //격리중 환자수 데이터 발행
                    publish_data("Sido1", "{\"동작구 코로나19 검사기관\": "+covid19hospital_data[0]+" "+covid19hospital_data[1]+"}"); //동작구 코로나19 검사기관 데이터 발행
                    publish_data("Sido2", "{\"강릉구 코로나19 검사기관\": "+covid19hospital_data[2]+" "+covid19hospital_data[3]+"}"); //강릉구 코로나19 검사기관 데이터 발행
                    publish_data("Sido3", "{\"강릉구 코로나19 검사기관\": "+covid19hospital_data[4]+" "+covid19hospital_data[5]+"}"); //관악구 코로나19 검사기관 데이터 발행
                    Thread.sleep(10000);
                }

            }catch (Exception e) {
                // TODO: handle exception
                try {
                    sampleClient.disconnect();
                } catch (MqttException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                e.printStackTrace();
                System.out.println("Disconnected");
                System.exit(0);
            }
        }
```

Java Code

브로커 서버 및 클라이언트 서버의 옵션을 지정해준다.
기본적으로 MQTT를 이용하기 위해 mosquitto 의 포트번호인 1883과 브로커 서버의 주소 127.0.0.1로 지정해준다. catch문을 통해 오류가 나면 메세지를 뜨게 한다.
Qos 레벨은 0으로 지정한다 -> 한번에 전송이 성공하지 않으면 전송은 실패한 상태로 끝이 나게 된다. 즉 QoS를 보장하지 않는 상태.

```
public void connectBroker() {
    String broker = "tcp://127.0.0.1:1883"; //접속 할 브로커 서버의 주소
    String clientId = "practice"; // 클라이언트 ID
    MemoryPersistence persistence = new MemoryPersistence();
    try {
        sampleClient = new MqttClient(broker, clientId, persistence);//Mqtt Client 객체 초기화
        MqttConnectOptions connOpts = new MqttConnectOptions(); //접속시 접속의 옵션을 정의하는 객체 생성
        connOpts.setCleanSession(true);
        System.out.println("Connecting to broker: "+broker);
        sampleClient.connect(connOpts); //브로커 서버에 접속
        sampleClient.setCallback(this);//Call back Option 추가
        System.out.println("Connected");
    } catch(MqttException me) {
        System.out.println("reason "+me.getReasonCode());
        System.out.println("msg "+me.getMessage());
        System.out.println("loc "+me.getLocalizedMessage());
        System.out.println("cause "+me.getCause());
        System.out.println("excep "+me);
        me.printStackTrace();
    }
}

public void publish_data(String topic_input, String data) {
    String topic = topic_input; //토픽
    int qos = 0; //Qos level (0,2,3)
    try {
        System.out.println("Publishing message: "+data);
        sampleClient.publish(topic, data.getBytes(), qos, false);
        System.out.println("Message published");
    } catch(MqttException me) {
        System.out.println("reason "+me.getReasonCode());
        System.out.println("msg "+me.getMessage());
        System.out.println("loc "+me.getLocalizedMessage());
        System.out.println("cause "+me.getCause());
        System.out.println("excep "+me);
        me.printStackTrace();
    }
}
```

Java Code

get_covid19_data 함수는 공공데이터기관의 서울시 사망자, 확진자, 격리중인 사람수의 데이터를 불러온다.
기본적으로 인증키는 필수 이고 인증키 아래는 해당 페이지의 옵션 값이다. 만약 5월 데이터가 아닌 4월 데이터를 전송하고 싶으면 startCreateDt=202104xx 로 바꿔주면 된다.
기본적으로 XML파일로 저장된 데이터들을 불러오기에 select문을 이용해 데이터를 불러온다.

```
//서울시 5월 한달 간 사망자, 확진자, 격리중인 사람 수
public String[] get_covid19_data() {
    Date current = new Date(System.currentTimeMillis());
    SimpleDateFormat d_format = new SimpleDateFormat("yyyyMMddHHmmss");
    System.out.println(d_format.format(current));
    String date = d_format.format(current).substring(0,8); //실시간 날짜 저장
    String time = d_format.format(current).substring(8,10); //실시간 시간 저장

    String url = "http://openapi.data.go.kr/openapi/service/rest/Covid19/getCovid19SidoInfStateJson"
            + "?serviceKey=6MAuV1Ni9N7pH8GR1D2%2FzleRYEVE9SQDA%2Bx8F5Uzo0Yl%2FqBdCjkYNtbVkxG6mkw%2BXIxD0ipjASwWvZVh1YM0LA%3D%3D" // 인증키
            + "&pageNo=1"
            + "&numOfRows=10"
            + "&startCreateDt=20210501"
            + "&endCreateDt=20210531";

    //데이터를 저장할 변수 초기화

    Document doc = null;
    String sutime = ""; //데이터 시간
    String Death = ""; //사망자 수
    String Def = "";  //확진자 수
    String Isol = ""; //격리중 환자수

    //Jsoup으로 API 데이터 가져오기
    try {
        doc = Jsoup.connect(url).get();
    } catch (IOException e) {
        e.printStackTrace();
    }


    Elements elements = doc.select("item");
    for (Element e : elements) { //각각 ITEM을 조사하기 위해
        if (e.select("gubunEn").text().equals("Seoul")) { //서울시 covid19 데이터
            sutime += e.select("createDt").text()+","; //시간 데이터
            Death += e.select("deathCnt").text()+","; //사망자 데이터
            Def += e.select("defCnt").text()+","; //확진자 데이터
            Isol += e.select("isolIngCnt").text()+","; //격리 환자 데이터
        }
    }
    System.out.println("확인용" +sutime);
    String[] out1 = {sutime,Death,Def,Isol};
    return out1;
}
```

Java Code

get_covid19hospital_data 함수도 get_covid19_data 함수와 마찬가지로 사용된다.
한가지 덧 붙이자면 위의 run() 코드에서 실행될때 각각 원하는 데이터를 배열형으로 나열하기에 string[] out2 ={}를 이용해 데이터를 배열로 나열해준다.
아래는 서울시 동작구,강동구,관악구의 병원을 알려주는 코드이다.

```
//서울시 코로나검사 실시기관
public String[] get_covid19hospital_data() {
    String url = "http://apis.data.go.kr/B551182/pubReliefHospService/getpubReliefHospList"
            + "?serviceKey=6MAuV1Ni9N7pH8GR1D2%2FzleRYEVE9SQDA%2Bx8F5Uzo0Yl%2FqBdCjkYNtbVkxG6mkw%2BXIxD0ipjASwWvZVh1YM0LA%3D%3D"
            + "&pageNo=1"
            + "&numOfRows=10"
            + "&spclAdmTyCd=97"; //수정

    //데이터를 저장할 변수 초기화
    //서울시 코로나검사 실시기관
    String Sido1 = "";
    String Sido2 = "";
    String Sido3 = "";

    //서울시 병원 번호
    String tel1 = "";
    String tel2 = "";
    String tel3 = "";
    Document doc = null;

    try {
        doc = Jsoup.connect(url).get();
    } catch (IOException e) {
        e.printStackTrace();
    }

    //서울시 코로나검사 실시기관
    Elements elements = doc.select("item");
    for (Element e : elements) {
        if (e.select("sgguNm").text().equals("동작구")) {
            Sido1 = e.select("yadmNm").text();
            tel1 = e.select("telno").text();
        }
        if (e.select("sgguNm").text().equals("강동구")) {
            Sido2 = e.select("yadmNm").text();
            tel2 = e.select("telno").text();
        }
        if (e.select("sgguNm").text().equals("관악구")) {
            Sido3 = e.select("yadmNm").text();
            tel3 = e.select("telno").text();
        }
    }
    String[] out2 = {Sido1,tel1, Sido2,tel2, Sido3,tel3};
    return out2;
```

Java Code

Acuator 코드이다. 이 부분은 실제로 라즈베리파이, 아두이노가 있어야 실행이 된다.
지금은 버튼만 눌리면 버튼이 눌린 메시지만 나오도록 구현해 놓았다.

```
    ///@@@@@@@@@@@@@@@@@@
    @Override
    public void connectionLost(Throwable arg0) { //연결이 끊어지면 ...
        // TODO Auto-generated method stub
        System.out.println("연결이 끊김");
    }

    @Override
    public void deliveryComplete(IMqttDeliveryToken arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void messageArrived(String topic, MqttMessage msg) throws Exception { //topic에 따라 동작이 달라져야 되기에 topic을 확인
        // TODO Auto-generated method stub
        if (topic.equals("btn")){ //topic이 led이면 아래 소스 수행
            System.out.println("------------------Actuator Function------------------");
            System.out.println("버튼 눌림");
            System.out.println("btn: " + msg.toString());
            // 라즈베리파이, 아두이노 제어소스코드 들어가면 실제로 제어 가능
            System.out.println("-----------------------------------------------------");
        }
    }
}
```

Node.js Code

기본적인 Node.js 코드는 다음과 같다.
1. 서버를 초기화 한다
2. MongoDB를 사용하기 위한 연결 객체를 만들어 준다.
3. MQTT에 접속이 성공하면 3가지의 토픽을 구독한다
ㄴ 1. 격리환자 토픽 구독(Isol)
   2. 확진자 토픽 구독(Def)
   3. 사망자 토픽 구독(Death)
4. MQTT 응답 메세지 수신시 동작
5. Mongo DB에서 최근 데이터 불러와서 HTML 페이지에 업데이트 한다
6. HTML 페이지 버튼 제어

```
 */
//서버 초기화 과정
var app = require('../app');
var debug = require('debug')('iotserver:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
//서버 초기화 과정 끝


// Connect Mongo DB
var mongoDB = require("mongodb").MongoClient;
var url = "mongodb://127.0.0.1:27017/IoTDB"; //Robo 3T
var dbObj = null;
mongoDB.connect(url, function(err, db){
  dbObj = db; //dbOdbj == mongoDB객체
  console.log("DB connect"); //성공적인 연결
});

```

Node.js Code

Robo 3T와 mongoDB를 연결하기 위한 서버를 만들어 준다.
url은 mongoDB 주소(27017) 및 Robo 3T 주소가 들어가 있다.
ㄴ 127.0.0.01:27017
ㄴ IoTDB
연결이 성공적으로 되면 DB connect 메시지 출력

```
 */
//서버 초기화 과정
var app = require('../app');
var debug = require('debug')('iotserver:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
//서버 초기화 과정 끝


// Connect Mongo DB
var mongoDB = require("mongodb").MongoClient;
var url = "mongodb://127.0.0.1:27017/IoTDB"; //Robo 3T
var dbObj = null;
mongoDB.connect(url, function(err, db){
  dbObj = db; //dbOdbj == mongoDB객체
  console.log("DB connect"); //성공적인 연결
});

```

Node.js Code

서버에 접속이 성공적으로 되면 java code에서 불러온 데이터의 토픽을 구독한다.
ㄴ Isol == 격리환자 , Def == 확진자, Death == 사망자
알아야 할 것은 서울시를 기준으로 한 것이다. 만약 지역까지 보게 하려면 java code에 새로운 객체를 만들어 지역 데이터를 추가해줘야 한다.

```
 */
var mqtt = require("mqtt");
var client = mqtt.connect("mqtt://127.0.0.1") //자기 자신을 얘기하는 IP 주소

// 접속에 성공하면, 3가지 토픽을 구독.
client.on("connect", function(){
  client.subscribe("Isol"); //격리환자 구독
  console.log("Subscribing Isol");
  client.subscribe("Def"); //확진자 구독
  console.log("Subscribing def");
  client.subscribe("Death"); // 사망자 구독
  console.log("Subscribing death");
})

// MQTT 응답 메세지 수신시 동작
client.on("message", function(topic, message){
  console.log(topic+ ": " + message.toString()); // 수신한 메세지 Topic 출력
  var obj = JSON.parse(message); // 수신한 메세지의 데이터를 obj 저장
  obj.create_at = new Date(); // 현재 날짜 데이터를 obj에 추가함.
  console.log(obj);
```

Node.js Code

html 페이지에 데이터를 업데이트 하기 위한 코드이다.
"socket_evt_update" 이름을 가진 이벤트가 수신되면 데이터가 배열로 들어온다.
배열로 된 데이터들은 Html 페이지에 업데이트가 된다.

```
// Mongo DB에서 최근 데이터 불러와서, HTML 페이지에 업데이트
var io = require("socket.io")(server); //package.json에서  "socket.io": "^2.0.4" 실행
io.on("connection", function(socket){
  socket.on("socket_evt_update", function(data){ //"socket_evt_update"이란 이름을 가진 이벤트가 수신이 되면 아래코드 실행
    var Isol = dbObj.collection("Isol"); // Isol 라는 이름의 collection 선택
    var Def = dbObj.collection("Def"); // Def 라는 이름의 collection 선택
    var Death = dbObj.collection("Death"); // Death 라는 이름의 collection 선택

    // 격리 데이터
    Isol.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){
      // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴 -> 배열로 만듬
      if(!err){
        console.log(results[0]);
        socket.emit("socket_up_Isol", JSON.stringify(results[0]));
      }
    });
    // 확진자 데이터
    Def.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){
      // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴 -> 배열로 만듬
      if(!err){
        console.log(results[0]);
        socket.emit("socket_up_Def", JSON.stringify(results[0]));
      }
    });
    // 사망자 데이터
    Death.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){
      // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴 -> 배열로 만듬
      if(!err){
        console.log(results[0]);
        socket.emit("socket_up_Death", JSON.stringify(results[0]));
      }
    });
  });
```

Node.js Code

java code 와 html 페이지에 있는 버튼을 제어하기 위한 코드

```
//HTML 페이지에서 버튼이 눌리면 LCD 제어를 위해 MQTT publish
socket.on("socket_evt_bnt",function(data){
  //버튼에 따라서 data(on/off) publish
  console.log("publishing btn");
  console.log(data);
  //publish
  client.publish("btn",data); //topic
});
});


/**
```

Html Code

Html은 크게 데이터를 받는 코드와 받은 데이터를 시각화 하는 코드가 존재한다.
키 값은 총 3개로 이루어져 있고 10초마다 데이터가 출력되게 하였다.
키 값을 받아 데이터를 출력 및 읽어온다.
체크박스의 체크여부에 따라 해당하는 데이터 값만 출력되도록
checkbox 함수를 만들어 if문을 설정해주었다.

1. QuarantinePatient == 격리환자
2. confirmedPatient== 확진환자
3. deceasedPatient== 사망환자

만약 키 값이 문자열 이면 동적으로 할당해줘야 한다.

```
var timer = null;
function checkbox(type){
        if( type === 'isol'){
            socket.on("socket_up_Isol", function(data){
                data = JSON.parse(data);
                $(".mqttlist_Isol").html('<li>'+'격리환자: '+data.QuarantinePatient+'명.'+'</li>');
            });
        } else if(type === 'def'){
            socket.on("socket_up_Def", function(data){
                data = JSON.parse(data);
                $(".mqttlist_Def").html('<li>'+'확진환자: '+data.confirmedPatient+'명.'+'</li>');
            });
        } else if( type === 'death'){
            socket.on("socket_up_Death", function(data){
                data = JSON.parse(data);
                $(".mqttlist_Death").html('<li>'+'사망환자: '+data.deceasedPatient+'명.'+'</li>');
            });
        }
    if(timer==null){
        timer = window.setInterval("timer_1()", 10000); //10000 시간 단위로 데이터를 읽는다는 뜻
    }
};
```

Html Code

Check 함수를 설정해 체크박스가 체크 되면 해당 메시지를 출력하도록 설정하였다.
아래의 코드는 시각화를 위한 코드이다.
checkbox를 줌으로써 사망자,확진자,격리환자에 대한 체크박스가 생겼고
웹의 제목을 다는 h1부분과 버튼을 클릭하면
해당 링크에 접속이 되는 하이퍼 링크 기능을 추가했다.

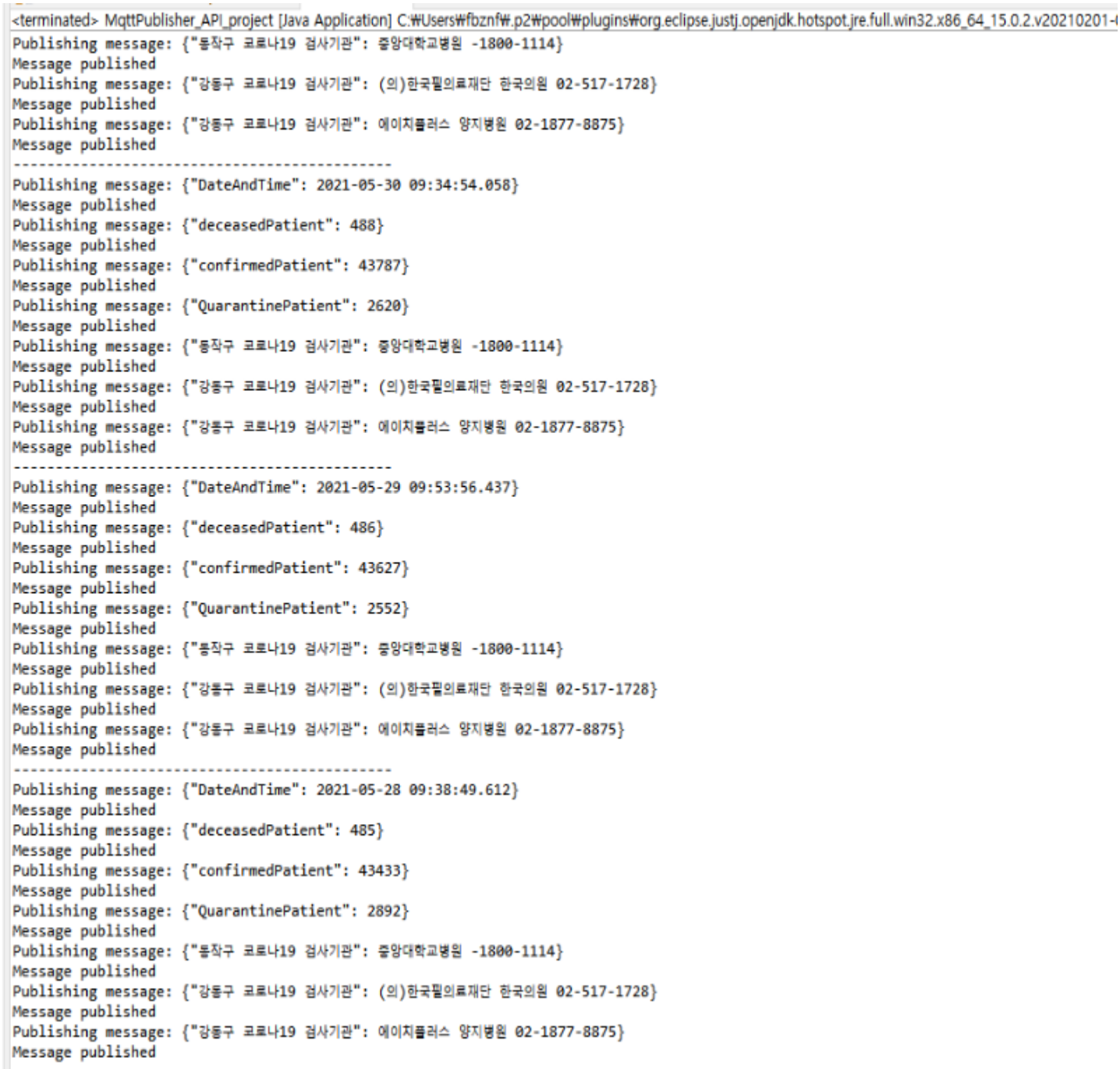
```
    function timer_1(){
        socket.emit("socket_evt_update", JSON.stringify({}));
    }

</script>
</head>
<body  style="text-align:center;" >
    <h2>COVID19</h2>
    2019년 12월 중국 후베이성 우한시에서 발생하여 세계적으로 확산된, 새로 발견된 신종 코로나바이러스에 의한 호흡기 감염질환
    <br><br><div class="search">
        <br><br><button onclick="Check()">원하는 데이터 선택</button><br>
        <label><input type="checkbox"  onclick="checkbox('death')"  value="사망자" > 사망자</label>
        <label><input type="checkbox"  onclick="checkbox('def')" value="확진자" > 확진자</label>
        <label><input type="checkbox"  onclick="checkbox('isol')"  value="격리환자" > 격리환자</label>
    </div>
    <h1>서울시 COVID19 정보</h1>
    <div id="msg">
        <div id="mqtt_logs">
            <ul class="mqttlist_Isol"></ul>
            <ul class="mqttlist_Def"></ul>
            <ul class="mqttlist_Death"></ul>
            <button id="button1" onclick="window.open('http://ncov.mohw.go.kr/')"><b>한국 COVID19</b></button>
            <button id="button2" onclick="window.open('https://www.seoul.go.kr/coronaV/coronaStatus.do')"><b>COVID19 서울시 발생 현황</b></button
        </div><br>

    </div>
```

결론 도출

Java Code의 출력 결과이다.
Thread.sleep(10000) 으로 설정, 10초간격을 유지
서울시 5월 1일~ 31일까지의 데이터이다.
사진처럼 31일부터 출력된다.



결론 도출

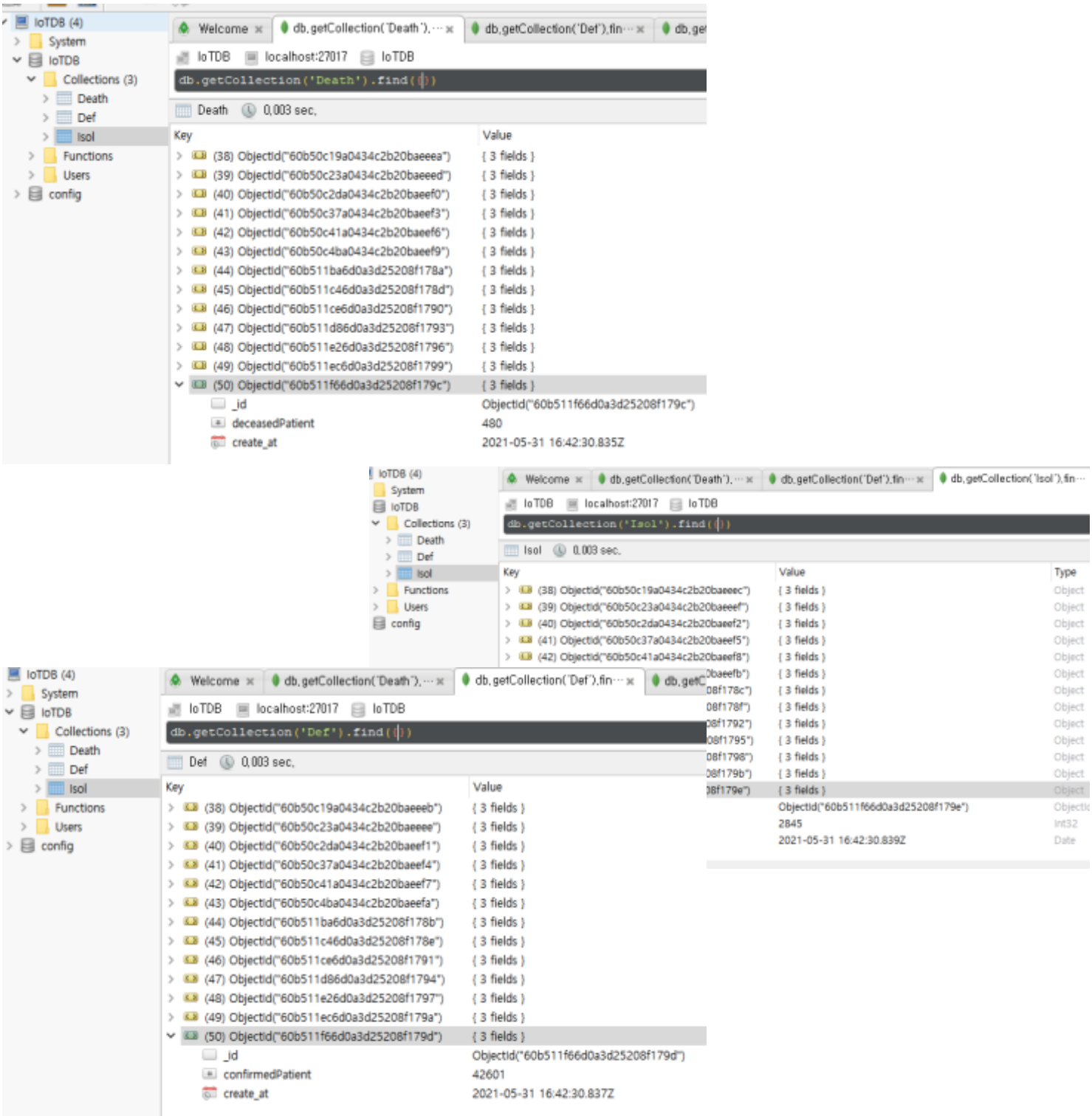
Node.JS Code의 출력 결과이다.
Java Code랑 다르게 여기서는 확진환자, 사망자, 격리환자, 현재 시간만 출력하게 하였다.

```
Death: {"deceasedPatient": 480}
Def: {"confirmedPatient": 42601}
Isol: {"QuarantinePatient": 2845}
{"n":1,"ok":1}
{"n":1,"ok":1}
{"n":1,"ok":1}
----------------------
{
  _id: 60b511f66d0a3d25208f179e,
  QuarantinePatient: 2845,
  create_at: 2021-05-31T16:42:30.839Z
}
{
  _id: 60b511f66d0a3d25208f179d,
  confirmedPatient: 42601,
  create_at: 2021-05-31T16:42:
}
{
  _id: 60b511f66d0a3d25208f17
  deceasedPatient: 480,
  create_at: 2021-05-31T16:42:
}
```

```
DB connect
Subscribing Isol
Subscribing def
Subscribing death
Death: {"deceasedPatient": 488}
Def: {"confirmedPatient": 43787}
Isol: {"QuarantinePatient": 2620}
{"n":1,"ok":1}
{"n":1,"ok":1}
{"n":1,"ok":1}
----------------------
{
  _id: 60b511ba6d0a3d25208f178c,
  QuarantinePatient: 2620,
  create_at: 2021-05-31T16:41:30.756Z
}
{
  _id: 60b511ba6d0a3d25208f178b,
  confirmedPatient: 43787,
  create_at: 2021-05-31T16:41:30.754Z
}
{
  _id: 60b511ba6d0a3d25208f178a,
  deceasedPatient: 488,
  create_at: 2021-05-31T16:41:30.751Z
}
----------------------
```

결론 도출

Node.JS 에서 받은 DB를 Robo 3T가 받아낸 결과이다.
데이터가 잘 들어오는 것이 확인 가능하다.



결론 도출

마지막으로 Html Code에서 실행되는 웹 페이지 이다.
버튼을 클릭하면 해당 링크로 접속이 되고 체크박스도 원활히 실행이 된다.
데이터의 정보는 수시로 바뀌며 10초간격으로 바뀐다.

참고문헌

1. 이상우 조교님
2. Mosquitto 사이트
3. 네이버 백과사전
4. 공공데이터포털

2. https://mosquitto.org/
3. https://terms.naver.com/entry.naver?docId=3386832&cid=58369&categoryId=58369
4. https://www.data.go.kr/index.do