프로젝트 제목

김준영, 20175210, 스마트IoT

<제목 차례>

I. 서론

1. 프로젝트의 필요성

최근 IoT 서비스는 계속 확대되고 있는 경향을 보인다. 4차 산업혁명과 함께 수많은 장치들이 인터넷에 연결되고 있고, 우리의 삶을 더 편하게 만들어 주고 있다. 하지만 센서를 가진 작은 디바이스・기기들은 기존의 인터넷이 사용하는 프로토콜을 사용하기에 너무나도 헤더사이즈가 크기 때문에, IoT 기기들을 위한 경량형 프로토콜을 사용해야 통신을 잘 할 수 있다. 그 중 대표적으로 CoAP과 MQTT 프로토콜이 있는데, 이번 프로젝트는 MQTT 프로토콜로 진행을 한다. MQTT는 Publish / Subscribe 형태를 취하여 3가지 QoS(Quality of Service)레벨을 제공한다. Publisher 와 Subscriber 사이에 Broker가 존재하여 데이터를 관리하고 뿌려주는 역할을 하는 것도 존재한다.

2. 프로젝트의 목적

IoT 기기들이 MQTT 프로토콜을 가지고 어떻게 통신하고 구현되는지 프로젝트를 통하여 경험해보고 구현해 봄으로서, 작동 원리를 파악 하는 것이 프로젝트의 주목적이다.

한국도로공사 공공데이터포탈의 '노선별 영업소간 구간내 소요시간 일괄조회' API를 통하여, 여름 휴가철에 상습 정체구간인 남양주 – 남춘천 사이의 평균 시간들의 데이터를 얻어, 양방향 소요시간이 얼마나 되는지 측정하고 예상 도착시간도 같이 알려준다. 또한 가변차로 시행여부 정보도 얻을 수 있다.

이를 통해 집에서 출발하기 전, 소요시간을 파악하여, 고속도로를 우회할지 말지 결정하여 휴가철에 유용하게 사용할 수 있다.

Ⅱ. 본론

1. 시스템 구조

그림 1은 입구 영업소 통과시간과 출구 영업소 통과시간 시스템의 구조를 보여준다. 본 시스템은 센서, MQTT Broker, Node.js, Resource DB, Web UI로 구성된다. 센서는 입구 영업소 통과시간과 출구 영업소 통과시간을 한국도로공사 API를 통해 10분 주기로 수집한 뒤, MQTT Broker 서버에 Publish 하는 역할을 수행한다. MQTT Broker는 Publish 된 데이터를 MQTT subscriber에게 전달한다. Node.js는 센서 데이터를 수신하여 Resource DB에 저장하는 MQTT subscriber의 역할과 Web UI를 제공하는 HTTP server의 역할을 수행한다. Resource DB는 수신되는 센서 데이터를 저장하고, Node.js의 요청에 따라 센서 데이터를 제공한다.

시스템의 전반적인 동작 과정은 다음과 같다. 먼저, Node.js의 MQTT subscriber가 MQTT Broker에게 남춘천(입구) -> 남양주(출구)토픽과, 남양주(입구) -> 남춘천(출구) 토픽의 구독을 요청하고, 구독요청을 받은 MQTT Broker는 MQTT subscriber를 토픽의 구독자 리스트에 추가한다. 남춘천(입구) -> 남양주(출구) 소요시간 데이터를 Publish하고, 남양주(입구) -> 남춘천(출구) 소요시간 데이터를 Publish하면 MQTT Broker가 해당 데이터를 구독하고있는 Subscriber에게 전송한다.

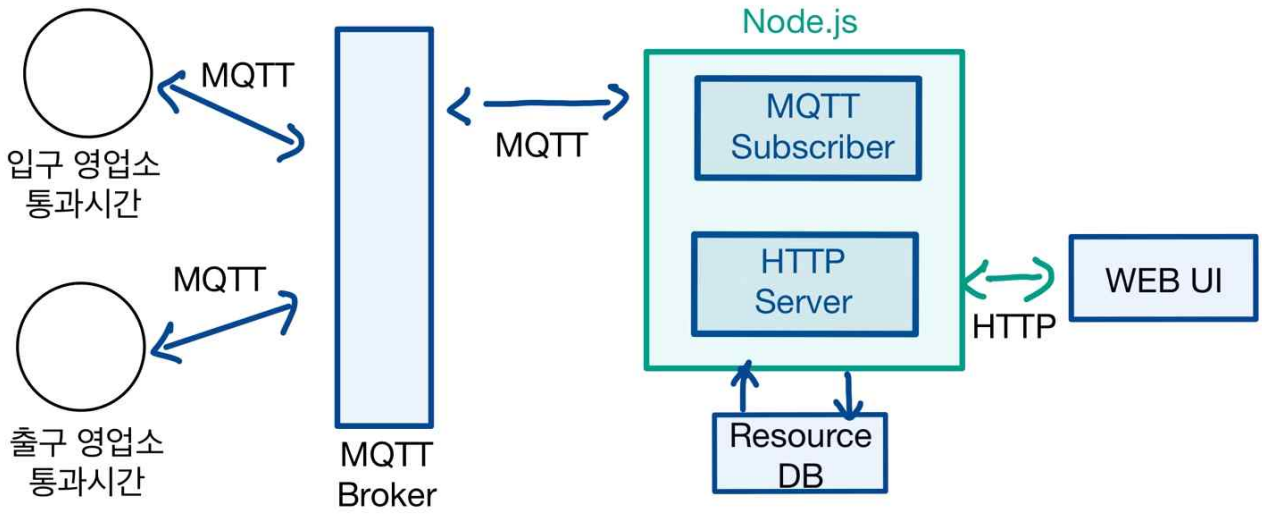

그림1. 입구영업소 통과시간 측정, 출구영업소 통과시간 측정 시스템 구조 예시

2. 소스코드 설명

□ MQTT Publisher - 자바 소스코드 설명

① run()메소드 • 객체 동작 메소드

```
20
21⊖   public static void main(String[] args) {
22        MqttPublisher_API obj = new MqttPublisher_API();
23        obj.run();
24    }
25⊖   public void run() {
26        connectBroker(); // 브로커 서버에 접속
27        try { // 여기 추가
28            sampleClient.subscribe("Arrive Time");
29        } catch (MqttException e1) {
30            // TODO Auto-generated catch block
31            e1.printStackTrace();
32        }
33        while(true) {
34            try {
35                int Toward_Chuncheon = get_traffic_time_toward_Chuncheon(); //남양주 -> 남춘천 토픽
36                int Toward_Namyangju = get_traffic_time_toward_Namyangju(); //남춘천 -> 남양주 토픽
37
38                publish_data("Toward_Chuncheon", "{\"Toward_Chuncheon\": "+ Toward_Chuncheon+"}"); //데이터 publish
39                publish_data("Toward_Namyangju", "{\"Toward_Namyangju\": "+ Toward_Namyangju+"}"); //데이터 publish
40
41                Thread.sleep(600000); // 시간설정만큼 발행 (10분마다 발행으로 기본설정)
42                                      //왜냐하면 기본 api가 15분마다 새로운 정보로 업데이트 되기 때문에
43            }catch (Exception e) {
44                // TODO: handle exception
45                try {
46                    sampleClient.disconnect();
47                } catch (MqttException e1) {
48                    // TODO Auto-generated catch block
49                    e1.printStackTrace();
50                }
51                e.printStackTrace();
52                System.out.println("Disconnected");
53                System.exit(0);
54            }
55        }
56    }
```

[설명]
main문에서 객체를 생성하고, 생성한 객체가 run메소드를 통해 Topic을 발행하는 메소드이다. Toward_Chuncheon에는 남양주IC에서 남춘천IC까지 소요되는 시간의 평균 계산한 메소드의 결과를 저장해주고, Toward_Namyangju에는 남춘천IC에서 남양주IC까지 소요되는 시간의 평균을 계산한 메소드의 결과를 저장한다. 저장한 후에는 publish_data로 토픽을 각각 발행해주고 Broker에게 보내는데, Thread.sleep(600000)으로 설정해, 10분마다 발행하는 것으로 해준다. 한국도로공사 API가 15분마다 갱신되기 때문에, 10분마다 발행해야 데이터 손실을 방지할 수 있기 때문이다. try-catch문으로 에러를 검출하고, 예외가 발생할 시 처리한다.

② connectBroker() 메소드 ·Broker 연결 메소드

```
public void connectBroker() {
    String broker = "tcp://127.0.0.1:1883"; // 브로커 서버의 주소
    String clientId = "practice"; // 클라이언트의 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); // 브로커서버에 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();
    }
}
```

[설명]
Broker서버에 접속하는 connectBroker() 메소드다. 변수 broker에 Broker서버의 주소를 입력하고, 변수 clientId 에 Client의 ID를 넣어준다. Broker에 접속하고 callback옵션도 같이 넣어 준다.

③ publish_data 메소드 · Topic 발행 메소드

```
public void publish_data(String topic_input, String data) {
    String topic = topic_input; // 토픽
    int qos = 0; // QoS level
    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();
    }
}
```

[설명]
발행할 Topic이름(topic_input)과, 발행할 데이터(data)를 매개변수로 가지는 publish_data(String topic_input, String data) 메소드이다. MQTT에는 QoS레벨(Quality of Service)이 존재하는데, 발행하고 응답메세지를 받지 않는 QoS레벨을 0으로 설정한다.

④ get_traffic_time_toward_Namyangju 메소드 ・남춘천IC->남양주IC소요시간평균시간

```
public int get_traffic_time_toward_Namyangju() { // 남춘천 -> 남양주 소요시간 메소드

    String url = "http://data.ex.co.kr/openapi/odhour/trafficTimeByRoute" //한국도로공사 api 목록 중 노선 별 영업소간 구간내 소요시간 조회 주소
            + "?key=8238184168" // api 키값
            + "&type=xml"
            + "&startUnitCode=656" //남춘천 영업소 코드 = 입구 영업소
            + "&endUnitCode=651"  //남양주 영업소 코드 = 출구 영업소
            + "&carType=1"; //1종
                        //1종을 선택한 이유 : 가장 빠르게 다니고 교통량이 가장 많이 때문에

    //데이터를 저장할 변수 초기화
    String timeAvg_toward_Namyangju = "0"; //남춘천 -> 남양주 소요시간 평균

    Document doc = null;

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

    Elements elements = doc.select("list"); //list항목을 선택하고
    for (Element e : elements) {
        if (e.select("startUnitCode").text().equals("656")) { //시작영업소가 남춘천(656)이면
            timeAvg_toward_Namyangju = e.select("timeAvg").text(); //평균소요시간을 변수에 저장한다
        }
    }
    int toward_Namyangju_Sec = Integer.parseInt(timeAvg_toward_Namyangju); //timeAvg_toward_Namyangju가 초 단위 문자열 형태이므로
                                                                           // Integer.parseInt로 int형으로 바꿔준 다음에
    int timeAvg_toward_Namyangju_Min = toward_Namyangju_Sec / 60; //보기 쉽게 분 단위로 바꿔준다.
    return timeAvg_toward_Namyangju_Min;
}
```

```
▼<data>
   <code>SUCCESS</code>
   <message>인증키가 유효합니다.</message>
   <count>5</count>
  ▼<list>
     <carType>1</carType>
     <endUnitCode>651 </endUnitCode>
     <endUnitName>남양주</endUnitName>
     <startUnitCode>656 </startUnitCode>
     <startUnitName>남춘천</startUnitName>
     <sumTm>1145</sumTm>
     <timeAvg>1879</timeAvg>
     <timeMax>3435</timeMax>
     <timeMin>1614</timeMin>
     <tmName>15분</tmName>
     <tmType>2</tmType>
   </list>
```

[설명]

남춘천IC -> 남양주IC까지 1종 차량의 소요시간 평균을 구하는 메소드이다. 먼저 1종차량만 선택한 이유는, 교통량이 가장많고 다른 차종에 비해 속도가 가장 빠르기 때문에, 교통량의 척도를 잘 파악할 수 있기 때문에 1종차량 데이터만 수집했다. url변수에 오픈API주소를 넣어주고, 밑에 그림에서 보이듯이 list안의 변수 중 시작점변수(startUnitCode)를 기준으로 소요시간 평균을 구한다. get_traffic_time_toward_Namyangju() 메소드에서는 시작지점이 남춘천IC(startUnitCode = 656)가 해당된다.

⑤ get_traffic_time_toward_Chuncheon 메소드 • 남양주IC->남춘천IC소요시간평균시간

```
public int get_traffic_time_toward_Chuncheon() { // 남양주 -> 남춘천 소요시간 메소드

    String url = "http://data.ex.co.kr/openapi/odhour/trafficTimeByRoute" //한국도로공사 api 목록 중 노선 별 영업소간 구간내 소요시간 조회 주소
            + "?key=8238184168" // api 키값
            + "&type=xml"
            + "&tmType=1"
            + "&startUnitCode=651" //남양주 영업소 코드 = 입구 영업소
            + "&endUnitCode=656" //남춘천 영업소 코드 = 출구 영업소
            + "&carType=1"; //1종

    //데이터를 저장할 변수 초기화
    String timeAvg_toward_Chuncheon = "0"; //남양주 -> 남춘천 소요시간 평균

    Document doc = null;

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

    Elements elements = doc.select("list");
    for (Element e : elements) {
        if (e.select("startUnitCode").text().equals("651")) { // 시작영업소가 남양주(651)이면
            timeAvg_toward_Chuncheon = e.select("timeAvg").text(); //평균소요시간을 변수에 저장한다
        }
    }
    int toward_Chuncheon_Sec = Integer.parseInt(timeAvg_toward_Chuncheon); //timeAvg_toward_Chuncheon가 초 단위 문자열 형태이므로
                                                                           // Integer.parseInt로 int형으로 바꿔준 다음에
    int timeAvg_toward_Chuncheon_Min = toward_Chuncheon_Sec / 60; //보기 쉽게 분 단위로 바꿔준다
    return timeAvg_toward_Chuncheon_Min;
}
```

```
▼<list>
    <carType>1</carType>
    <endUnitCode>656 </endUnitCode>
    <endUnitName>남춘천</endUnitName>
    <startUnitCode>651 </startUnitCode>
    <startUnitName>남양주</startUnitName>
    <sumTm>1145</sumTm>
    <timeAvg>2680</timeAvg>
    <timeMax>3908</timeMax>
    <timeMin>2625</timeMin>
    <tmName>15분</tmName>
    <tmType>2</tmType>
</list>
```

[설명]
남춘천IC -> 남양주IC까지 1종 차량의 소요시간 평균을 구하는 메소드이다. 동일하게 1종 차량의 데이터만 추출한다. url변수에 오픈API주소를 넣어주고, 밑에 그림에서 보이듯이 list안의 변수 중 시작점변수(startUnitCode)를 기준으로 소요시간 평균을 구한다. get_traffic_time_toward_Chuncheon() 메소드에서는 시작지점이 남양주 IC(startUnitCode = 656)가 해당된다.

⑥ messageArrived메소드

```
    @Override
    public void messageArrived(String topic, MqttMessage msg) throws Exception {
        // TODO Auto-generated method stub
        if (topic.equals("Arrive Time")){
            System.out.println("--------------------예상도착시간측정--------------------");
            System.out.println(msg.toString());
            System.out.println("시간을 측정해보자~!");
            System.out.println("--------------------------------------------------------");
        }
    }
}
```

[설명]

html 페이지에서 버튼을 클릭했을 때, 발생되는 messageArrived 메소드이다. vs Code에서 버튼클릭 이벤트가 발생한 걸 알았을 때, Topic(Arrive Time)을 Publish • 발행하고, 이를 이클립스에서 Topic을 구독하여 예상도착시간측정을 출력하는 메소드이다.

□ MQTT Subscriber · VSCode www파일 소스코드 설명

① MongoDB 접속

```
// Connect Mongo DB
 var mongoDB = require("mongodb").MongoClient;
 var url = "mongodb://127.0.0.1:27017/IoTDB";
 var dbObj = null;
 mongoDB.connect(url, function(err, db){
   dbObj = db;
   console.log("DB connect");
 });

 /**
  * MQTT subscriber (MQTT Server connection & Read resource data)
  */
 var mqtt = require("mqtt");
 var client = mqtt.connect("mqtt://127.0.0.1") //local host를 의미 , 내 컴퓨터 자
신
```

[설명]
MongoDB에 접속하는 코드이다. 포트번호 27017이 생성되고 ROBO 3T에 데이터베이스 IoTDB에 접속에 성공하면, 콘솔에 "DB connect"를 출력해준다.

② Topic 구독

```
// 접속에 성공하면, 2가지 토픽을 구독.
 client.on("connect", function(){
   client.subscribe("Toward_Chuncheon"); //남양주 -> 남춘천 평균 소요시간 토픽 구독
   console.log("Subscribing Toward_Chuncheon"); //console에 출력
   client.subscribe("Toward_Namyangju");  //남춘천 -> 남양주 평균 소요시간 토픽 구독
   console.log("Subscribing Toward_Namyangju"); //console에 출력
 })
```

[설명]
Broker 서버에 접속하면, Publisher가 발행하는 Topic을 구독하고(Subscirbe) 콘솔에 어떤 Topic을 구독했는지 출력해준다. 자바 이클립스에서 발행한 Topic (Toward_Chuncheon, Toward_Namyangju)를 각각 구독하고, 콘솔에 각 데이터를 구독하고 있다고(Subscribing) 출력해준다.

③ MQTT 메시지 수신시 동작

```
// 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(JSON파일) 저장
   obj.create_at = new Date(); // 현재 날짜 데이터를 obj에 추가함. 날짜라는 키를
 json파일에 저장
   console.log(obj);
```

[설명]
MQTT Type의 Topic이 수신되었을 때 동작되는 메소드이다. Publisher가 발행한 Topic 데이터를 console.log(topic + ": " + message.toString()으로 출력해주고, 변수 obj에 토픽 메시지를 JSON형식으로 저장해준다. 객체 obj가 언제 생성되었는지 Date클래스를 통해 날짜와 시간도 콘솔에 출력해준다.

④ Topic을 Database에 저장

```
   // 수신한 메세지를 Mongo DB에 저장
   if (topic == "Toward_Chuncheon"){
      // 남양주 -> 남춘천 평균 소요시간 토픽이면,
     var Toward_Chuncheon  = dbObj.collection("Toward_Chuncheon"); //
      Toward_Chuncheon 라는 이름을 가진 collection 선택
     Toward_Chuncheon.save(obj, function(err, result){
// collection에 남양주 -> 남춘천 평균 소요시간 저장
       if (err){
         console.log(err);
       }else{
         console.log(JSON.stringify(result));
       }
     });
   }else if (topic == "Toward_Namyangju"){
      // 남춘천 -> 남양주 평균 소요시간 토픽이면
     var Toward_Namyangju  = dbObj.collection("Toward_Namyangju");
     // Toward_Namyangju라는 이름을 가진 collection선택
     Toward_Namyangju.save(obj, function(err, result){
     // collection에 남춘천 -> 남양주 평균 소요시간 저장
       if (err){
         console.log(err);
       }else{
         console.log(JSON.stringify(result));
       }
     });
   }
 });
```

[설명]
Topic을 MongoDB Robo3T에 저장하는 코드이다. 각 Topic "Toward_Chuncheon" 은 Database Collection "Toward_Chuncheon"에 저장해주고, Topic "Toward_Namyangju"은 Database Collection "Toward_Namyangju"에 저장해준다.

⑤ Socket 통신

```
// Mongo DB에서 최근 데이터 불러와서, HTML 페이지에 업데이트
var io = require("socket.io")(server);
io.on("connection", function(socket){
  socket.on("socket_evt_update", function(data){
  //socket_evt_updated 이벤트가 발생되면, 실행
    var Toward_Chuncheon = dbObj.collection("Toward_Chuncheon");
  //Toward_Chuncheon 콜렉션을 선택한다
    var Toward_Namyangju = dbObj.collection("Toward_Namyangju");
  //Toward_Namyangju 콜렉션을 선택한다
    Toward_Chuncheon.find({}).sort({_id:-1}).limit(1).toArray(function(err, resul
ts){
      //-1은 마지막부터 접근을 의미, 마지막db에 들어온 데이터부터 정렬
      // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴
         -> 배열로 만듬(result에 저장)
      if(!err){
        console.log(results[0]); //콘솔로 찍어줌
        socket.emit("socket_up_Toward_Chuncheon", JSON.stringify(results[0]));
      //소켓으로 전달, 이벤트는 socket_up_Toward_Chuncheon
      }
    });

    Toward_Namyangju.find({}).sort({_id:-1}).limit(1).toArray(function(err, resul
ts){
      // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴 -> 배열로 만듬
      if(!err){
        console.log(results[0]); //콘솔로 찍어준다
        socket.emit("socket_up_Toward_Namyangju", JSON.stringify(results[0]));
      //이벤트는 socket_up_Toward_Namyangju
      }
    });
  });
```

[설명]

MongoDB의 최신 데이터를 불러와서 정렬한 후, Socket이벤트로 html페이지에 업데이트 해주는 코드이다. MongoDB Collection "Toward_Chuncheon"에 저장된 데이터와 MongoDB Collection "Toward_Namyangju"에 저장된 데이터를 각각 변수에 저장하고 정렬한 후, socket.emit메소드로 html페이지로 보내준다. socket은 www파일과 html페이지를 연결해주는 연결다리 역할이라고 생각하면 편하게 이해할 수 있었다.

⑥ 버튼이벤트 Socket 메소드

```
//html 페이지에서 예상도착시간 버튼이 눌리면 실행되는 메소드
socket.on("socket_evt_bnt", function(data){
 var now = new Date();
 console.log(data);
 client.publish("Arrive Time",data);
});
```

[설명]
html페이지에서 예상도착시간 버튼이 눌리면 실행되는 메소드다. 버튼이 눌리면 html페이지에서 socket.emit("socket_evt_bnt")를 발생하고 이를 socket.on으로 수신한 다음, "Arrive Time" Topic을 발행해준다.

□ VS code · html페이지 구현 소스코드 설명

① Socket 이벤트 수신시 실행되는 메소드

```
    $(document).ready(function(){
        socket.on("socket_up_Toward_Chuncheon", function(data){
            data = JSON.parse(data);
            if(data.Toward_Chuncheon < 40) //평균 소요시간이 40분 미만이면
                $(".mqttlist_Toward_Chuncheon").html('<li>' + '남양주IC -> 남춘천IC
까지 소요시간 : ' +data.Toward_Chuncheon+ ' 분' + '</li>'); //소요시간만 출력
            else //평균 소요시간이 40분이상이면
                // 가변차로가 시행중이라는 것을 같이 출력해준다.
                $(".mqttlist_Toward_Chuncheon").html('<li>' +
'남양주IC -> 남춘천IC까지 소요시간 : ' +data.Toward_Chuncheon+ ' 분'
+ ', (가변차로가 시행중입니다)' + '</li>');
            Arrive_Chuncheon_min = parseInt(timenow.getMinutes()) + parseInt(data.
Toward_Chuncheon); //현재시간 + 소요시간
            Arrive_Chuncheon_hour = parseInt(timenow.getHours()); //현재시간
            if(Arrive_Chuncheon_min >= 60){ // 분을 시간으로 바꿔주는 조건문
                Arrive_Chuncheon_hour += 1;
                if(Arrive_Chuncheon_hour >= 24){
                    Arrive_Chuncheon_hour = 0;
                }
                Arrive_Chuncheon_min -= 60;
            }
        });
        socket.on("socket_up_Toward_Namyangju", function(data){
            data = JSON.parse(data);
            if(data.Toward_Namyangju < 40) //평균 소요시간이 40분 미만이면
                $(".mqttlist_Toward_Namyangju").html('<li>' + '남춘천IC -> 남양주IC
까지 소요시간 : ' +data.Toward_Namyangju+' 분' + '</li>'); //소요시간만 출력
            else //평균 소요시간이 40분 이상이면
                //위와 같이 가변차로가 시행중이라는 것을 같이 출력해준다.
                $(".mqttlist_Toward_Namyangju").html('<li>' + '남춘천IC -> 남양주IC
까지 소요시간 : ' +data.Toward_Namyangju+' 분' + ', (가변차로가 시행중입니
다)' + '</li>');
            Arrive_Namyangju_min = parseInt(timenow.getMinutes()) + parseInt(data.
Toward_Namyangju); //현재시간 + 소요시간
            Arrive_Namyangju_hour = parseInt(timenow.getHours()); //현재시간
            if(Arrive_Namyangju_min >= 60){
                Arrive_Namyangju_hour += 1;
                if(Arrive_Namyangju_hour >= 24){
                    Arrive_Namyangju_hour = 0;
                }
                Arrive_Namyangju_min -= 60;
            }
        });
        if(timer==null){
            timer = window.setInterval("timer_1()", 3000); //3초에 한번씩 전송
        }
    });
```

[설명]
www파일에서 socket.emit으로 내보낸 파일들을 socket.on을 통해서 수신하여 html페이지를 구현하기 위한 코드이다. socket.emit메소드와 socket.on의 메시지가 동일하다면, data 파일을 JSON 형식으로 바꿔준다. Toward_Chuncheon의 데이터값이 40미만이면 소요시간만 출력해주고, 만약 40이상(소요시간이 40분이상이면) 소요시간과 가변차로가 실행중인 것을 출력해준다. Toward_Namyangju도 동일하게 적용해준다. Date클래스로 현재시간단위와 분단위를 추출하여 소요시간을 구하고, 60분이 넘어가면 1시간을 더해주고, 24시간제에서 24가 넘어가면 0시로 바꿔주는 조건문도 추가하여 시간을 구한다.

② 예상도착시간 버튼이 클릭되었을 때 실행되는 메소드

```
    function predict_Arrivetime(){ //버튼이 클릭되면 실행되는 이벤트
        socket.emit("socket_evt_bnt", "When We Arrive that place?");
        document.getElementById("btnclick_print_current_time").innerText
     = "현재시간 : " + timenow.getHours() + "시 " + timenow.getMinutes() + "분 ";
        document.getElementById("btnclick_print_toward_Chuncheon").innerText
     = " 남양주IC -> 남춘천IC까지 예상 도착 시간 : "+Arrive_Chuncheon_hour + "시 "
                                                  + Arrive_Chuncheon_min + "분";
        document.getElementById("btnclick_print_toward_Namyangju").innerText
     = " 남춘천IC -> 남양주IC까지 예상 도착 시간 : " + Arrive_Namyangju_hour+ "시
      + Arrive_Namyangju_min + "분";
}
```

[설명]
예상도착시간 버튼이 클릭되었을 때 실행되는 함수이다. socket.emit으로 버튼이 눌렸다는 것을 알려주고, "When we arrive that place?" data를 보낸다. 버튼이 눌리면 출력되어야 하기 때문에, innertext로 현재시간과, 남춘천ic도착시간, 남양주ic도착시간을 html페이지에 보여준다.

③ HTML 페이지 구현 코드

```
</script>
</head>
<body>
    여름 휴가철 상습 정체구간 서울양양고속도로 (남양주IC - 남춘천IC 구간) 소요시간
    <div id="msg">
        <div id="mqtt_logs">
            <p>
                <img src="ex.jpg"></p>
                <img src="ChunHighway.jpg"></p>
            </p>
            <ul class="mqttlist_Toward_Chuncheon"></ul>
            <ul class="mqttlist_Toward_Namyangju"></ul>
            <button id = "predict_Arrivetime" onclick = "predict_Arrivetime()"><b>
예상도착시간</b></button>
        </div>
    </div>
    <div id = "btnclick_print_current_time"></div>
    <div id = "btnclick_print_toward_Chuncheon"></div>
    <div id = "btnclick_print_toward_Namyangju"></div>
</body>
</html>
```

[설명]
html 페이지를 구현한 코드이다.

3. 실행화면

① Broker서버 접속 및 Topic 발행시 자바이클립스 실행 화면

```
Connecting to broker: tcp://127.0.0.1:1883
Connected
Publishing message: {"Toward_Chuncheon": 35}
Message published
Publishing message: {"Toward_Namyangju": 35}
Message published
```

```
Publishing message: {"Toward_Chuncheon": 32}
Message published
Publishing message: {"Toward_Namyangju": 80}
```

② Database접속 성공 및 Topic 구독시 실행화면

```
C:\MQTT_Project\IoT_server>npm start

> iot-server@0.0.0 start C:\MQTT_Project\IoT_server
> node ./bin/www

DB connect
Subscribing Toward_Chuncheon
Subscribing Toward_Namyangju
```

③ Topic 구독시 VS Code 실행화면

```
{
  _id: 60bb97c56499cf31c86d2f38,
  Toward_Chuncheon: 35,
  create_at: 2021-06-05T15:27:01.947Z
}
{
  _id: 60bb97c56499cf31c86d2f39,
  Toward_Namyangju: 35,
  create_at: 2021-06-05T15:27:01.950Z
}
Toward_Chuncheon: {"Toward_Chuncheon": 32}
{ Toward_Chuncheon: 32, create_at: 2021-06-06T11:46:14.034Z }
Toward_Namyangju: {"Toward_Namyangju": 80}
{ Toward_Namyangju: 80, create_at: 2021-06-06T11:46:14.038Z }
```

④ HTML 페이지 접속 및 버튼 클릭시 VS Code 실행화면

```
GET /MQTT.html 304 6.219 ms - -
GET /ex.jpg 304 3.946 ms - -
GET /ChunHighway.jpg 304 1.408 ms - -
When We Arrive that place?
When We Arrive that place?
When We Arrive that place?
```

⑤ HTML 페이지 버튼 클릭시 자바이클립스 실행화면

```
Message published
Publishing message: {"Toward_Namyangju": 35}
Message published
--------------------예상도착시간측정--------------------
When We Arrive that place?
시간을 측정해보자~!
------------------------------------------------------
--------------------예상도착시간측정--------------------
When We Arrive that place?
시간을 측정해보자~!
```

⑥ ROBO 3T 데이터가 넣어진 화면

Toward_Muncheon 0.005 sec.

	_id	Toward_Munche	create_at
1	ObjectId("6...	2012	2021-06-02 17:13:10.435Z
2	ObjectId("6...	2012	2021-06-02 17:13:15.828Z
3	ObjectId("6...	2012	2021-06-02 17:13:21.238Z
4	ObjectId("6...	2012	2021-06-02 17:13:26.762Z
5	ObjectId("6...	2012	2021-06-02 17:13:32.155Z
6	ObjectId("6...	2012	2021-06-02 17:13:37.538Z
7	ObjectId("6...	2012	2021-06-02 17:13:42.922Z
8	ObjectId("6...	2012	2021-06-02 17:13:48.356Z
9	ObjectId("6...	1747	2021-06-02 17:42:18.547Z
10	ObjectId("6...	1747	2021-06-02 17:42:24.257Z
11	ObjectId("6...	1747	2021-06-02 17:42:29.880Z
12	ObjectId("6...	1747	2021-06-02 17:42:35.320Z
13	ObjectId("6...	1747	2021-06-02 17:42:40.707Z
14	ObjectId("6...	1747	2021-06-02 17:42:46.097Z
15	ObjectId("6...	1747	2021-06-02 17:54:18.671Z
16	ObjectId("6...	1747	2021-06-02 17:54:24.061Z
17	ObjectId("6...	1747	2021-06-02 17:54:29.526Z
18	ObjectId("6...	1747	2021-06-02 17:54:34.860Z
19	ObjectId("6...	1747	2021-06-02 17:54:40.231Z
20	ObjectId("6...	1747	2021-06-02 18:00:10.229Z
21	ObjectId("6...	1747	2021-06-02 18:00:15.615Z
22	ObjectId("6...	1747	2021-06-02 18:00:21.008Z
23	ObjectId("6...	1747	2021-06-02 18:00:26.422Z
24	ObjectId("6...	1747	2021-06-02 18:00:31.811Z

⑦ HTML 페이지 캡쳐 화면

여름 휴가철 상습 정체구간 서울양양고속도로 (남양주IC - 남춘천IC 구간) 소요시간

- 남양주IC -> 남춘천IC까지 소요시간 : 35 분
- 남춘천IC -> 남양주IC까지 소요시간 : 33 분

예상도착시간
현재시간 : 0시 52분
남양주IC -> 남춘천IC까지 예상 도착 시간 : 1시 27분
남춘천IC -> 남양주IC까지 예상 도착 시간 : 1시 25분

여름 휴가철 상습 정체구간 서울양양고속도로 (남양주IC - 남춘천IC 구간) 소요시간

- 남양주IC -> 남춘천IC까지 소요시간 : 33 분
- 남춘천IC -> 남양주IC까지 소요시간 : 85 분, (가변차로가 시행중입니다)

예상도착시간
현재시간 : 19시 33분
남양주IC -> 남춘천IC까지 예상 도착 시간 : 20시 6분
남춘천IC -> 남양주IC까지 예상 도착 시간 : 20시 58분

III. 결론

이번 프로젝트의 목적은 MQTT Protocol이 전반적으로 어떻게 작동하고, 어떤 구조를 가지고 동작하는지 파악하는 것이 주된 목적인 것 같다. CoAP은 Server / Client구조를 가지고 있는 것과는 다르게, MQTT는 Client가 Publisher / Subscriber 이 동시에 될 수 있다는 점과 Topic들을 관리하는 Broker서버가 존재하는 구조를 가진다는 것이다. 공공데이터 API를 활용하는 것. 실습에서 꾸준히 Node.js를 활용하는 것과, JSON파일을 사용해보고 자바스크립트와 자바를 통해 큰 틀을 구현하고, 직접 Publish와 Subscribe모델을 구현하고, 마지막으로 html페이지를 만드는 것 까지 실습과 프로젝트를 통해서 많은 경험을 할 수 있었다. 전반적으로 MQTT가 어떻게 작동하는 것도 파악하는 것도 중요하지만, 추후에 개발자가 되어서 내가 어떤 일을 하게 될지 간접적으로 느낄 수 있어서 굉장히 의미가 있었던 프로젝트였다. 구현하고 싶었던 것들이 훨씬 많았으나, 아직 코딩실력과 자바스크립트 활용 부분이 많이 부족하여 구현하지 못한 요소들이 많아서 정말 아쉽다. 코딩실력과 MQTT / CoAP Protocol과 구현 방법, 구조를 잘 파악하고 공부하면, 비단 좋은 개발자가 될 수 있을뿐더러, 우리의 삶을 더 나아지게 만들 수 있을 것 같다.

<참고자료>

1. *http://data.ex.co.kr/openapi/basicinfo/openApiInfoM?apiId=0117&pn=-1*

: 한국도로공사 공공 API주소

2. *http://bananawabanana.blogspot.com/2017/08/mqtt.html, MQTT 기반 Web 채팅 예제*

3. https://woowabros.github.io/experience/2017/08/11/ost_mqtt_broker.html,