프로젝트 제목

안원영, 20155137, 컴퓨터공학과

목차

I. 서론

1. 프로젝트의 필요성

프로젝트와 관련된 기존 IoT 서비스의 동향 및 문제점을 제시하고, 프로젝트의 필요성에 대해 작성한다.

현대인들은 많은 시간을 온라인상에서 보내고 있다. 온라인이 가지는 장점도 많지만 단점도 존재한다. 내가 생각하는 단점 중 가장 문제가 되는 부분을 생각해 보았다.

온라인에서 가장 쉽고 빠르게 접할 수 있는 컨텐츠는 유흥거리와 자극적인 정보라고 생각한다. 이런 컨텐츠들은 우리에게 재미를 가져다 주지만 쉽고 얕은 내용을 가지고 있다. 따라서 사람들은 이런 컨텐츠에 많이 노출될수록 '생각' 이라는 것을 점차 하지 않게 된다. 적은 시간동안이라면 괜찮겠지만 지금처럼 많은 시간을 온라인에서 보내는 현대인들에게 있어서 생각할 시간이 없어진다는 것은 큰 문제가 될 것이다.

따라서 이러한 문제를 해결하기 위해 이 프로젝트를 구성하였다.

2. 프로젝트의 목적

앞서 언급된 필요성과 관련하여 프로젝트의 구체적인 목표를 제시한다. 그리고, 전체적인 프로젝트 진행 과정을 간략하게 작성한다.

프로젝트를 통해 앞서 설명한 문제가 심각함을 인지하고 개선해 보고자 한다.

프로젝트의 중점적인 목표는 사람이 생각할 시간을 많이 가지게 하기 위한 방법들 중 하나를 소개하는 것이다.

프로젝트에서 소개할 방법은 바로 '책' 이다. 책은 좋은 정보와 함께 생각할 시간을 제공 한다. 또한 온라인보다 정확하고 신뢰할 수 있는 정보를 제공함으로써 개인의 성장에 도움이 될 수 있을 것이다.

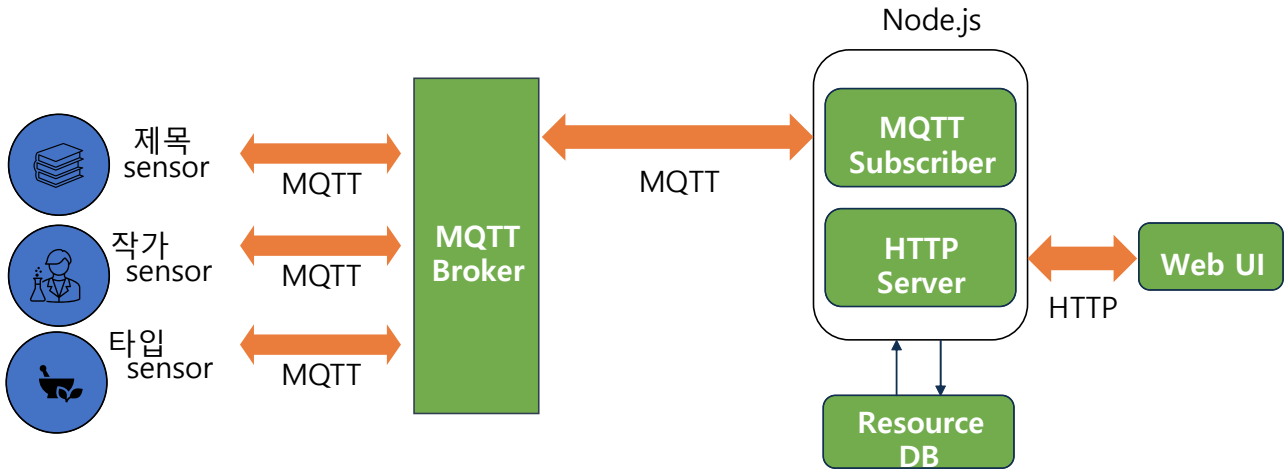
책을 추천하기 위해 사용할 API는 국립중앙도서관에서 재공하는 서사추천도서 이다. 이 책은 크게 4가지의 종류로 나눠진다. 문학,인문과학,사회과학,자연과학으로 나눠서 책을 추천한다. 이 프로젝트에서는 랜덤으로 4가지 종류중 하나를 선택해 2021.01.01 ~ 2021.05.31 까지의 추천도서 중 10가지의 정보를 추려 보여준다.

II. 본론

1. 시스템 구조

프로젝트의 시스템 구조 그림을 제시한다. 그리고, 시스템을 구성하는 각 구성요소의 기능 및 특징을 소개하고, 시스템의 동작과정 과정을 서술한다. 아래는 예시이다.

그림 1은 타이틀, 작가, 종류 모니터링 시스템의 시스템 구조를 보여준다. 본 시스템은 센서, MQTT Broker, Node.js, Resource DB, Web UI로 구성된다. 센서는 타이틀, 작가, 종류 데이터를 국립중앙도서관 API를 통해 정보를 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 타이틀, 작가, 종류 토픽의 구독을 요청하고, 구독요청을 받은 MQTT Broker는 MQTT subscriber를 토픽의 구독자 리스트에 추가한다. 그 다음, 타이틀, 작가, 종류 센서가 정보를 Publish하면, MQTT Broker는 해당 데이터를 MQTT Subscriber에게

전송한다. 또한 버튼을 만들어 버튼을 누를때마다 버튼색이 바뀌도록 설정했다.

그림1. 기온, 강수량 모니터링 시스템 예시

2. 소스코드 설명

이클립스 코드

3. 이클립스에 있는 코드의 역할은 publisher 로 정보를 보내는 용도로 사용됩니다. MQTTClient 객체를 만들어 구독을 할 수도 있습니다.

4. 먼저 받아올 데이터를 저장할 전역변수 배열을 3개 설정했습니다. 책의 제목, 작가, 책의 타입을 받아와서 저장하는 역할을 합니다.

5. 주기적으로 publish를 발생하기위해 while문 안에서 데이터를 요청했습니다.

6. 데이터는 get_book_data() 메소드에서 가져오며, 이 메소드는 국립중앙도서관의 사서가 달마다 추천해주는 책 정보들을 가져오게 됩니다. 저는 20210101 의 날짜부터 현재날짜까지 책의 종류를 바꿔가며 받아오기 위해 현재 날짜 정보를 가진 date메소드와 책 종류가 들어간 배열을 이용해 인덱스를 랜덤으로 가져와 목록을 요청하도록 하였습니다.

7. 정리하자면, 이번년도 1월부터 현재까지 추천한 책들 중 랜덤으로 선택된 책의 종류를 선택해 책 정보를 publish 해주게 됩니다.

8. 가져온 데이터는 publish_data() 메소드를 통해 publish 하게 됩니다. 만들어둔 MQTTClient 객체를 이용해 publish를 하게 되면 www.js에서 구독한 topic의 정보를 받습니다.

*MqttPublisher_API.java × MqttPublisher_API.java

```java
package mqtt_practice;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;

public class MqttPublisher_API implements MqttCallback{ // implement callback 추가 & 필요한 메소드 정의
    static MqttClient sampleClient;// Mqtt Client 객체 선언

    public static void main(String[] args) {
        MqttPublisher_API obj = new MqttPublisher_API();//클래스 객체를 만듬
        obj.run();//현재 클래스의 run 메소드 실행 -> run메소드에 여러가지 메소드를 포함
    }
    //책 데이터를 저장할 배열
    String[] author = new String[10];
    String[] title = new String[10];
    String[] type = new String[10];

    public void run() {
        connectBroker(); // //브로커 서버 접속 메소드 실행
        try { // 여기 추가
            sampleClient.subscribe("led");//클라이언트 객체가 led를 구독한다.
        } catch (MqttException e1) {

            e1.printStackTrace();
        }
        int i = 0;
        while(i < 10) {//주기적으로 발행(publish)하고 싶으면
            try {
                get_book_data();//책 데이터 생성
//              System.out.println("test : " + author[i]);

                publish_data("author", "{\"author\": "+ "\"" + author[i] + "\"" +"}"); // 작가 데이터 발행
                publish_data("title", "{\"title\": "+ "\"" + title[i] + "\"" +"}");; // 제목 데이터 발행
                publish_data("type", "{\"type\": "+ "\"" + type[i] + "\"" +"}");; // 책타입 데이터 발행


                Thread.sleep(3000); // 5초마다 publish 를 발생시킴
            }catch (Exception e) {

                try {
                    sampleClient.disconnect();
                } catch (MqttException e1) {
                    e1.printStackTrace();
                }
                e.printStackTrace();
                System.out.println("Disconnected");
                System.exit(0);
            }
        }
    }

    public void connectBroker() {//브로커와 연결
        String broker = "tcp://127.0.0.1:1883"; //브로커 서버의 아이피 주소 ,1883:브로커 서버 포트번호
        String clientId = "practice"; // 클라이언트의 ID
        MemoryPersistence persistence = new MemoryPersistence();
        try {
            //브로커가 누구와 연결할지 client ID로 결정된다.
            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
            //메시지가 오면 callback이 일어난다.
            sampleClient.setCallback(this);// Call back option 추가 -> callback 기능을 활성화 시킴
            System.out.println("Connected");
        } catch(MqttException me) {//브로커 서버 접속에 오류가 났을때
            System.out.println("reason "+me.getReasonCode());
            System.out.println("msg "+me.getMessage());
            System.out.println("loc "+me.getLocalizedMessage());
            System.out.println("cause "+me.getCause());
```

```
87    }
88
89    public void publish_data(String topic_input, String data) { //
90        String topic = topic_input; //토픽을 설정
91        int qos = 0; // QoS level
92        try {
93            System.out.println("Publishing message: "+data);
94            //토픽, 전달할 데이터, Qos, false
95            sampleClient.publish(topic, data.getBytes(), qos, false);//연결된 client에 맞게 topic 설정또는 보냄
96            System.out.println("Message published");
97
98        } catch(MqttException me) {
99            System.out.println("reason "+me.getReasonCode());
100           System.out.println("msg "+me.getMessage());
101           System.out.println("loc "+me.getLocalizedMessage());
102           System.out.println("cause "+me.getCause());
103           System.out.println("excep "+me);
104           me.printStackTrace();
105       }
106   }
107
108   public void get_book_data() { //API : 국립중앙도서관 추천도서
109       // 현재 시간 확인해서 날짜, 시간 저장
110       String[] book_type = {"11","6","5","4"};
111       Random r = new Random();
112       String bt = book_type[r.nextInt(4)];//책의 타입을 랜덤하게 설정해서 가져온다.
113
114       Date current = new Date(System.currentTimeMillis());
115       SimpleDateFormat d_format = new SimpleDateFormat("yyyyMMdd");
116       String date = d_format.format(current).substring(0,8); // 오늘 날짜 저장
117
118       //데이터는 html 파일 형식으로 url이 가지고있다.
119       String url = "https://nl.go.kr/NL/search/openApi/saseoApi.do"
120               + "?key=da37d369eb8c3047f55b17eddc77094d12ee1535f30712678b25b2bf8143935d" // api 키값
121               + "&startRowNumApi=1"
122               + "&endRowNumApi=10"
123               + "&start_date=20210101"
124               + "&nd_date="+date//오늘날짜까지
125               + "&drCode=" + bt;
126
127       Document doc = null;
128
129       // Jsoup으로 API 데이터 가져오기파싱
130       try {
131           doc = Jsoup.connect(url).get();
132       } catch (IOException e) {
133           e.printStackTrace();
134       }
135       Elements elements = doc.select("item");//item에 각각의 데이터가 저장되어있음
136       int i = 0;
137       for (Element e : elements) {
138           author[i] = e.select("recomauthor").text();
139           title[i] = e.select("recomtitle").text();
140           type[i] = e.select("drCodeName").text();
141           i++;
142       }
143   }
```

www.js 코드

① www.js는 토픽을 구독하고 DB와 연동되어 정보를 저장하고 socket에 정보를 저장해 html 파일로 넘겨주는 역할을 한다.

② MongoClient를 호출해 mongodb에 접근할 수 있는 객체를 이용해서 mongodb와 연결한다.

③ io객체는 웹서버의 3000번 포트와 연결된 소켓을 가리킨다. 여기서 3000번 포트는 현재 로컬호스트에 열려있는 웹서버의 포트번호이다. 이 3000번 포트를 사용해 소켓으로 통신 할 것이다.

④ 자기 자신의 로컬 호스트와 연결된 client 객체를 만들어 원하는 topic을 구독합니다. 여기서 author, title, type 를 구독합니다.

⑤ Client.on은 이클립스의 publisher로부터 메시지를 받으면 동작합니다. 메시지를 받으면 해당 topic 을 확인해서 DB의 만들어진 테이블에 정보를 저장 합니다.

⑥ 저는 DB의 Book_author, Book_title, Book_type 테이블을 만들어 저장하였습니다.

⑦ Io.in은 socket을 이용해 데이터를 전달받고 전달해주는 역할을 합니다. socket에는 해당 이벤트와 함께 보낼 데이터를 입력하게 됩니다.

⑧ Socket에 Socket_evt_update 이벤트가 오게되면 DB의 각각의 테이블에 있는 가장 나중에 저장된 정보를 1개 가져옵니다. 가져온 정보를 socket.emit을 이용해 socket_up_author과 같이 해당 topic의 이벤트를 발생시켜 데이터를 socket에 담아 전송합니다. 이때 socket은 열려있는 3000번 포트를 가리킵니다.

⑨ 마지막으로 저는 버튼을 하나 추가시켜 버튼을 누르면 버튼의 색이 변경되도록 만들었습니다. 따라서 socket.on에 socket_evn_bnt 이벤트가 오면 콘솔에 변경될 색상을 찍어주고 emit으로 socket_up_color 이벤트를 가진 곳으로 변경될 색을 전달하게 됩니다.

```
 * Create HTTP server.
 */

var server = http.createServer(app);

// Connect Mongo DB -> mongodb에 접속
var mongoDB = require("mongodb").MongoClient;
//브로커 서버 아이피(해당 컴퓨터 아이피)
var url = "mongodb://127.0.0.1:27017/IoTDB";//mongodb의 테이블 이름과 같아야하고, 27017은 고유 포트번호이다.
var dbObj = null;
mongoDB.connect(url, function(err, db){//mongoDB에 연결되면
  dbObj = db;//
  console.log("DB connect");
});

/**
 * MQTT subscriber (MQTT Server connection & Read resource data)
 */
var mqtt = require("mqtt");//브로커를 설정함
var client = mqtt.connect("mqtt://127.0.0.1")//자기 자신 로컬 호스트로 작성해야 함

// 접속에 성공하면, 3가지 토픽을 구독.
//client.on("발생 이벤트","이벤트 핸들러");//connection 하고나면 구독해야함
client.on("connect", function(){
  client.subscribe("author");//작가 구독
  console.log("Subscribing author");//콘솔창에 잘 구독했다고 출력해주는 역할
  client.subscribe("title");//타이틀 구독
  console.log("Subscribing title");
  client.subscribe("type");//책 종류 구독
  console.log("Subscribing type");

})

// MQTT 응답 메세지 수신시 동작 -> 클라이언트가 메세지 수신시
//구독한 데이터가 응답이 왔을경우
client.on("message", function(topic, message){//topic과 value값이 전달됨
  console.log(topic+ ": " + message.toString()); // 수신한 메세지 Topic 출력
  var obj = JSON.parse(message); // 수신한 메세지의 데이터를 obj 저장, Json파일을 의워줌 -> 몽고db에 필요함
  obj.create_at = new Date(); // 현재 날짜 데이터를 obj에 추가함.obj의 형식은 JSON 파일이다.
  //JSON 파일은 키와 데이터로 이루어진 hashmap 이다. -> 키값과 데이터가 한쌍이다. -> 키를 가져오면 데이터 알 수 있음
  console.log(obj);//이것을 몽고db에 저장하면 됨

  // send the received data to MongoDB
  // 수신한 메세지를 Mongo DB에 저장 -> topic
  if (topic == "author"){ // 만약 토픽이 작가라면,
    //MongoDB에 센서 데이터 저장
    var Book_author  = dbObj.collection("Book_author"); // Book_author 이름을 가진 테이블 collection 선택
    Book_author.save(obj, function(err, result){//result에는 작가의 정보가 저장되어있다.
      //작가 테이블 collection에 obj(작가 데이터) 저장
      if (err){//에러 났을 경우
        console.log(err);
      }else{//에러가 안났을경우 저장된 데이터(매개변수로 받은 result)를 콘솔에 출력
        console.log(JSON.stringify(result));
      }
    });
  }else if (topic == "title"){ // 만약 토픽이 책제목 이라면,
    var Book_title  = dbObj.collection("Book_title"); // Book_title 이름을 갖은 collection 선택
    Book_title.save(obj, function(err, result){ // collection에 책이름 데이터 저장
      if (err){
        console.log(err);
      }else{
        console.log(JSON.stringify(result));
      }
    });
  }else if (topic == "type"){ // 만약 토픽이 책타입 이라면,
    var Book_type  = dbObj.collection("Book_type"); // Book_type 이름을 갖은 collection 선택
    Book_type.save(obj, function(err, result){ // collection에 책타입 데이터 저장
      if (err){
        console.log(err);
      }else{
        console.log(JSON.stringify(result));
      }
    });
  }
});
```

```
96
97  // get data from MongDB and then send it to HTML page using socket
98  // Mongo DB에서 최근 데이터 불러와서, HTML 페이지에 업데이트
99  // 소켓 : DB의 데이터를 HTML 페이지에 업로드 하기위해 사용
100 //소켓을 사용한 데이터 전달 --> html page
101 var io = require("socket.io")(server);//소켓을 가져와서 서버와 연결
102 //소켓을 열어둬서 html 파일이 접근할 수 있도록 함
103 io.on("connection", function(socket){ //연결이 되면 function 수행
104  //HTML 페이지에서 "socket_evt_update" 이벤트가 발생하면 소켓을 통해서 www.js 에 알린다.
105  // HTML페이지로 전송할때 이벤트 이름(온도,습도,미세먼지) 을 달리해서 데이터를 보낸다.
106   socket.on("socket_evt_update", function(data){//연결된 소켓에서 socket_evt_update 이벤트가 발생하면 function 실행
107    //db에 접속하여(dbobj) 저장되어 있는 테이블을 선택
108     var author = dbObj.collection("Book_author"); // temperature 라는 이름의 collection 선택
109     var title = dbObj.collection("Book_title"); // humidity 라는 이름의 collection 선택
110     var type = dbObj.collection("Book_type"); // pm_value 라는 이름의 collection 선택
111
112     // 책 작가 데이터
113     // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴 -> 배열로 만듬
114     // id로 정렬하는데 거꾸로 정렬해서 가장(최근)마지막으로 들어온 데이터 1개의 정보를 배열형식으로 가져온다.
115     author.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){//배열이 result에 저장됨
116       if(!err){//에러가 아니면 소켓에 데이터를 담아서 다시 MQTT.html로 보낸다
117         console.log(results[0]);//DB에서 찾은 temp 데이터중 가장최근의 데이터(result)에서 key값만 콘솔에 출력
118         socket.emit("socket_up_author", JSON.stringify(results[0]));//JSON 으로 상대편 html로 emit:전달한다.
119       }
120     });
121
122     // 책 제목 데이터
123     title.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){
124       // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴 -> 배열로 만듬
125       if(!err){
126         console.log(results[0]);
127         socket.emit("socket_up_title", JSON.stringify(results[0]));
128       }
129     });
130
131     // 책타입 데이터
132     type.find({}).sort({_id:-1}).limit(1).toArray(function(err, results){
133       // collection에서 가장 최근 데이터 정렬-> 하나의 데이터만 불러옴 -> 배열로 만듬
134       if(!err){
135         console.log(results[0]);
136         socket.emit("socket_up_type", JSON.stringify(results[0]));
137       }
138     });
139   });
140
```

```
141
142   //MQTT.HTML 페이지에서 버튼이 눌리면, MQTT.HTML에 있는 함수에서 해당 socket.on() 의 "socket_evt_bnt" 이벤트를 발생시킴
143   socket.on("socket_evt_bnt", function(data){//socket_evt_bnt 이벤트(버튼눌리면) 발생시 실행됨
144
145     console.log("Change Color : ", data);//콘솔에 변경될 색상 출력
146     socket.emit("socket_up_color", data);//socket에 변경될 색상 전달
147
148   });
149 });
150
```

html 코드

① 먼저 io객체를 이용해 웹 서버와 connection을 할 (3000번 포트)socket을 생성합니다.

② html에서도 socket을 이용해 데이터를 주고받습니다. 해당 이벤트가 발생했을 때 html의 태그를 이용해 태그 안에 내용을 추가시켜줍니다.

③ 저는 여기서 좀더 정보를 깔끔하게 보기 위해서 table을 만들어 정리하였습니다. Css를 이용해서 테이블의 테두리와 안에 들어가는 리스트의 형태를 보기좋게 바꿔줬습니다.

④ 또한 버튼을 만들어 클릭이 일어나면 button_on() 함수를 호출하게 되는데, 이 함수는 color 를 가지고 있는 배열을 만들어 랜덤으로 배열의 색을 socket에 담에 보내게 됩니다. 보낸 색은 www.js 에서 socket_evt_bnt 이벤트를 가진곳으로 전달됩니다.

JS www MQTT.html ×

public > MQTT.html > html > script

```
<!DOCTYPE html>
<html lang="ko">
<meta charset="UTF-8">
<title>MQTT Monitoring Service</title>
<style>
    li{ list-style: none;
        padding-left: 0;
    }
</style>
<script src="/socket.io/socket.io.js"></script><!-- 소켓 기능을 가져오는 곳  -->
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script><!-- Jquery를 가져오는 곳 -->
<script>
    var socket = io.connect();/* io로 소켓을 가져옴  */
    var timer = null;/* 3초마다 받아올수 있도록 타이머 생성  */
    $(document).ready(function(){
        socket.on("socket_up_author", function(data){//socket_up_temp 이벤트가 수신되면 실행된다.
            data = JSON.parse(data);//JSON 형식으로 데이터 받아옴
            $(".mqttlist_author").html('<li>' + data.author +'</li>');/*html파일에 id가 "mqttlist_temp"인 곳에 데이터 추가해줌  */
        });
        socket.on("socket_up_title", function(data){
            data = JSON.parse(data);
            $(".mqttlist_title").html('<li>'+data.title +'</li>');
        });

        socket.on("socket_up_type", function(data){
            data = JSON.parse(data);
            $(".mqttlist_type").html('<li>'+ data.type +'</li>');
        });
        socket.on("socket_up_color", function(data){//버튼 색상 변경
            $("#color").css("color",data);
        });

        //타이머 해놓고 일정 간격으로 함수 호출 -> 함수에서 www.js의 socket.on 를 실행
        //-> socket.on 에서 DB의 테이블에서 가장 최근 정보를 가져와, MQTT.html의 해당이벤트 발생 socket으로 전송
        if(timer==null){//3초간격으로 timer_1 함수를 실행시킴
            timer = window.setInterval("timer_1()", 5000);
        }
    });
```

```
    function timer_1(){
        // socket_evt_temp 라는 이벤트를 발생시키는 곳
        socket.emit("socket_evt_update", JSON.stringify({}));/* www.js 소켓으로 socket_evt_temp 이벤트가 발생했다고 알림 */
    }
    function button_on(){//버튼 색상을 변경하기 위한 함수
        var color = ["red","blue","yellow","white"];
        var r = Math.floor(Math.random() * 4);//0~3 의 난수를 발생

        socket.emit("socket_evt_bnt", color[r]);//color 배열의 r 인덱스를 리턴
    }

</script>

</head>
<body>
MQTT Mornitoring Service
    <div id="msg">
        <div id="mqtt_logs">
            <table border="1">
                <tr><h3>책 정보를 출력합니다.</h3></tr>
                <tr><th>작가</th><td><ul class="mqttlist_title"></ul></td></ul></tr>
                <tr><th>제목</th><td><ul class="mqttlist_author"></ul></td></ul></tr>
                <tr><th>분류타입</td><td><ul class="mqttlist_type"></ul></td></ul></tr>
                <tr><td><button id="color" onclick="button_on()"><b>Color Change Start</b></button></td></tr>
            </table>
        </div>
    </div>
</body>
</html>
```

9. 실행화면

최종적으로 html 파일로 작가, 제목, 분류타입이 표시됩니다. 깔끔하게 보기 위해 테이블을 사용하였습니다.

III. 결론

프로젝트의 목적, 진행과정, 결과를 요약하여 서술한다. 또한, 프로젝트의 의의, 활용방안, 앞으로 더 보완해야 할 부분에 대해서 서술한다.

Mqtt의 동작방식은 client와 broker가 정보를 주고받는 형식으로 진행된다. 여기서 client는 publisher가 될수도 있고 subscriber가 될 수도 있다. 이 프로젝트에서 publisher 역할을 이클립스의 MQTTpubliser가 해주었고 정보를 보내 브로커인 mosquitto가 www.js 로 정보를 보내주면 www.js 에서 DB에 데이터를 저장해주고 socket에 가공된 데이터를 전달해 html 페이지에서 이 내용들을 받아볼 수 있게 되었다. MQTT의 동작방식을 구체적으로 알 수 있게 되었고 배운 것을 토대로 센서와 연동시킬수만 있다면, 현재 우리가 쓰고있는 iot 기기들을 구현할 수 있을 것이다. 아직은 많이 부족해서 실제 센서와의 연결이 어떻게 되는지와 다양한 센서 연동 예시들을 통해 공부를 해 나가야 할 것같다.