

SMART GREEN CAMPUS

<정보통신공학과>

20181643 서동권
20181745 윤진원
20181676 이동엽
20181680 이진욱
20181692 최민석
20201849 임세나



Index.

01. MQTT Broker

02. Back-End

03. Front-End

MQTT Broker

01



이전 발표 내용

수신한 데이터를 원하는 포맷으로 변경

```
subclient.on( event: 'message' , cb: function (topic : string , message : Buffer )  
  //let value = message.toString('utf-8');  
  let [sName,loc,tmp]=message.toString().split( separator: ",")  
  .
```

```
{"sensor_name":"temperature","location":"N5","value":"31.30"}  
{"sensor_name":"temperature","location":"N5","value":"31.30"}  
{"sensor_name":"temperature","location":"N5","value":"31.50"}  
{"sensor_name":"temperature","location":"N5","value":"31.60"}  
{"sensor_name":"temperature","location":"N5","value":"31.40"}  
{"sensor_name":"temperature","location":"N5","value":"31.40"}  
{"sensor_name":"temperature","location":"N5","value":"31.30"}  
{"sensor_name":"temperature","location":"N5","value":"31.30"}  
{"sensor_name":"temperature","location":"N5","value":"31.10"}
```

진행 상황

```
import { Controller, Inject } from '@nestjs/common';
import {
  ClientProxy,
  MessagePattern,
  Payload as pd,
} from '@nestjs/microservices';
import { take } from 'rxjs';
```

```
//Subscriber 구현 코드
@MessagePattern('smartgreen') //구독하는 주제 1
sumData(@pd() message) {
  const [sName, loc, tmp] = message.toString().split(',');
  const data = {
    sensor_name: sName,
    location: loc,
    value: tmp,
  };
  const sensors = JSON.stringify(data);
  console.log(sensors);
}
```

Nest.js 프레임워크를 사용하여 코드를 변경합니다. 또한, @MessagePattern 패키지에서 가져온 데코레이터를 사용합니다. Nest.js를 이용하여 mqtt의 broker를 구현하려면 이 데코레이터를 사용해야 합니다.

@MessagePattern을 사용하고, 값을 JSON.stringify를 통해 JSON 형식으로 변경하여, 원하는 포맷에 맞추어 코드를 변경합니다.

진행 상황

```
~/Development/mqtt main ?1
› npm run start dev

> mqtt@0.0.1 start
> nest start dev

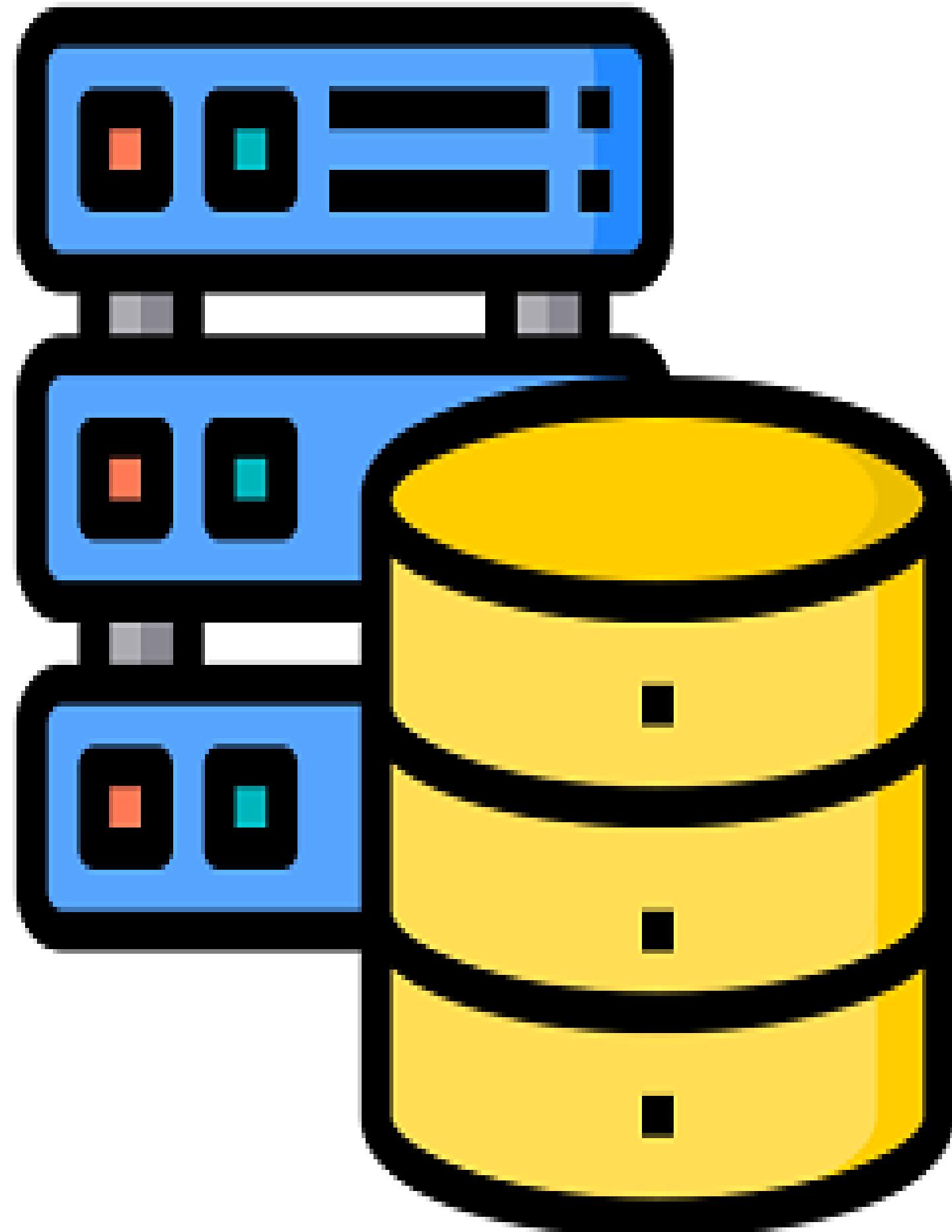
[Nest] 2004 - 11/08/2022, 12:48:12 AM      LOG [NestFactory] Starting Nest application...
[Nest] 2004 - 11/08/2022, 12:48:12 AM      LOG [InstanceLoader] ClientsModule dependencies initialized +58ms
[Nest] 2004 - 11/08/2022, 12:48:12 AM      LOG [InstanceLoader] AppModule dependencies initialized +1ms
[Nest] 2004 - 11/08/2022, 12:48:15 AM      LOG [NestMicroservice] Nest microservice successfully started +3131ms
{"sensor_name":"temperature","location":"N5","value":"22.80"}
{"sensor_name":"temperature","location":"N5","value":"22.80"}
{"sensor_name":"temperature","location":"N5","value":"22.80"}
```

서버를 열고, 데이터를 수신했을 때 원하는 포맷에 맞게, 정상적으로 수신되는 과정을 볼 수 있습니다.

향후 계획

1. DB파트 쪽에 데이터를 송신(@Post)하는 과정을 추가적으로 구현해야 합니다.
2. 현재는 컴퓨터 공학과 쪽에서 실제로 센싱한 데이터를 받은게 아니기 때문에 센싱한 데이터를 받아서 DB쪽에 송신해야 합니다.

Back-End



02

이전 발표 내용

nodemailer 모듈을 이용한 알림 구현

```
16 //온도가 35도가 넘어가면 경고 발생 -> 콘솔창이 아닌 페이지에서 볼 수 있도록
17 if (req.body.sensor_name == "temperature" && req.body.value >= 35)
18
19     const temp = req.body.value;
20
21     const emailParam = {
22
23         toEmail: '20181676@edu.hanbat.ac.kr',
24
25         subject: 'temperature problem occur',
26
27         text: `경고!! 온도가 ${temp}도로 너무 높습니다.`
28     };
29
30     mailer.sendMail(emailParam);
31 }
```

nodemailer 의 sendMail 를 사용하여 메일 전송에 주소와 내용을 설정

The screenshot shows a developer's environment with three main windows:

- Code Editor:** Displays a Node.js script for a temperature monitoring application. It includes logic to check if a sensor's value is above 35, send an email notification to '20181676@edu.hanbat.ac.kr' with the subject 'temperature problem occur', and a body containing '경고!! 온도가 \${item}도로 너무 높습니다.'.
- Browser:** Shows an open Gmail inbox. A new message from 'ldy_1204@naver.com' is highlighted with an orange box, containing the text '경고!! 온도가 80도로 너무 높습니다.' (Warning!! The temperature is too high at 80 degrees).
- Terminal:** A MySQL command-line interface running on localhost. It lists several rows of sensor data, with the last two rows highlighted by an orange box: '2022-08-04 06:24:00 | 22 | temperature | N8'. The command used was 'mysql -h localhost -u root -p'.

클라이언트로 부터 전송된 값을 DB에 저장하고, 일정 값 이상일때 메일 전송

진행 상황

NESTJS를 이용한 User DTO로 정보 받기

```
import { IsString, Matches, MaxLength, MinLength } from "class-validator";

export class AuthCredentialsDto {
    @IsString()
    @MinLength(4)
    @MaxLength(20)
    username: string;

    @IsString()
    @MinLength(4)
    @MaxLength(20)
    //영어랑 숫자만 가능한 유효성 체크
    @Matches(/^[a-zA-Z0-9]*$/ , {
        message: 'password only accepts english and number',
    })
    password: string;
}
```

회원가입시 User 정보를 DTO로 만들어서 저장하게 합니다.
DTO는 계층간 데이터 교환이 이루어 질 수 있도록 하는 객체를 말합니다.

진행 상황

NESTJS를 이용한 회원가입 구현

The screenshot shows a database table on the left and a Postman API request on the right. The database table has columns: id, username, and password. The rows show data: 1 dlehdduq1, 2 dlehdduq2, 3 dlehdduq3, 4 dlehdduq5, and 5 smart-green-campus. The Postman request is to the endpoint localhost:3000/auth/signup. The Body tab shows a JSON payload: { "username": "smart-green-campus", "password": "1234"}. A red arrow points from the 'auth' in the URL to the @Controller('auth') annotation in the code. Another blue arrow points from the 'signup' in the URL to the signUp() method in the code.

```
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @Post('/signup')
  signUp(
    @Body(ValidationPipe) authCredentialsDto: AuthCredentialsDto,
  ): Promise<void> {
    return this.authService.signUp(authCredentialsDto);
  }

  @Post('/signin')
  signIn(
    @Body(ValidationPipe) authSignInDto: AuthSignInDto,
  ): Promise<{ accessToken: string }> {
    return this.authService.signIn(authSignInDto);
  }

  @Post('/test')
  @UseGuards(AuthGuard())
  test(@GetUser() user: User) {
    console.log('user', user);
  }
}
```

auth를 통해 AuthController에 접근하고, 이후 POST 요청을 통해 회원가입을 진행함을 알 수 있습니다.
회원가입 요청이 처리가 되면, 데이터 베이스에 저장하는 방식으로 구현하였습니다.

진행 상황

```
async signUp(authCredentialsDto: AuthCredentialsDto): Promise<void> {
    const { username, password } = authCredentialsDto;

    const salt = await bcrypt.genSalt();
    const hashedPassword = await bcrypt.hash(password, salt);
    const user = this.userRepository.create({
        username,
        password: hashedPassword,
    });

    try {
        await this.userRepository.save(user);
    } catch (error) {
        if (error.code === '23505') {
            throw new ConflictException('Existing username');
        } else {
            throw new InternalServerErrorException();
        }
    }
}
```

Controller에서 요청을 받으면 Service에서 처리합니다.

해당 코드는 요청받은 json 형태의 정보를 하나의 변수에 저장해서 중복된 이름을 검사합니다.

진행 상황

NESTJS를 이용한 로그인 구현

```
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @Post('/signup')
  signUp(
    @Body(ValidationPipe) authCredentialsDto: AuthCredentialsDto,
  ): Promise<void> {
    return this.authService.signUp(authCredentialsDto);
  }

  @Post('/signin')
  signIn(
    @Body(ValidationPipe) authSignInDto: AuthSignInDto,
  ): Promise<{ accessToken: string }> {
    return this.authService.signIn(authSignInDto);
  }

  @Post('/test')
  @UseGuards(AuthGuard())
  test(@GetUser() user: User) {
    console.log('user', user);
  }
}
```

auth를 통해 AuthController에 접근하고, 이후 POST 요청을 통해 로그인을 진행함을 알 수 있습니다.
로그인 요청이 처리가 되면, Promise 객체를 통해 토큰을 발급하는 방식으로 구현하였습니다.

진행 상황

```
async signIn(authSignInDto: AuthSignInDto): Promise<{ accessToken: string }> {
  const { username, password } = authSignInDto;
  const user = await this.userRepository.findOne({
    where: { username: username },
  });

  if (user && (await bcrypt.compare(password, user.password))) {
    // 유저 토큰 생성 (Secret + Payload)
    const payload = { username };
    const accessToken = await this.jwtService.sign(payload);

    return { accessToken };
  } else {
    throw new BadRequestException('logIn failed');
  }
}
```

Controller에서 요청을 받으면 Service에서 처리합니다.

해당 코드는 요청받은 json 형태의 정보를 username과 password가 일치하는지 일치하는지 확인하고, 토큰을 발급합니다. 토큰은 1시간의 유효시간을 갖습니다.

향후 계획

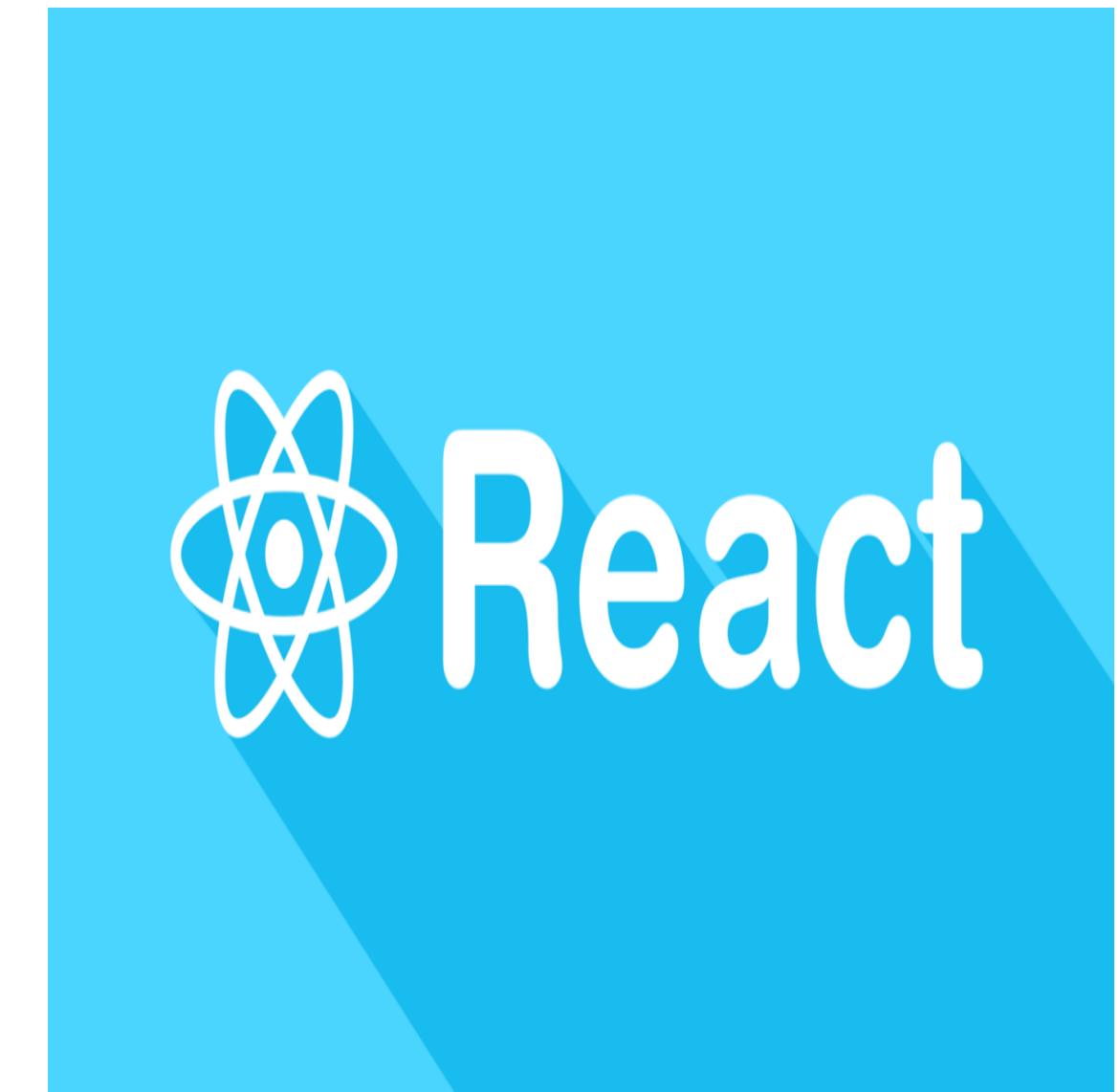
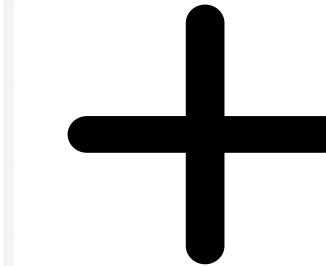
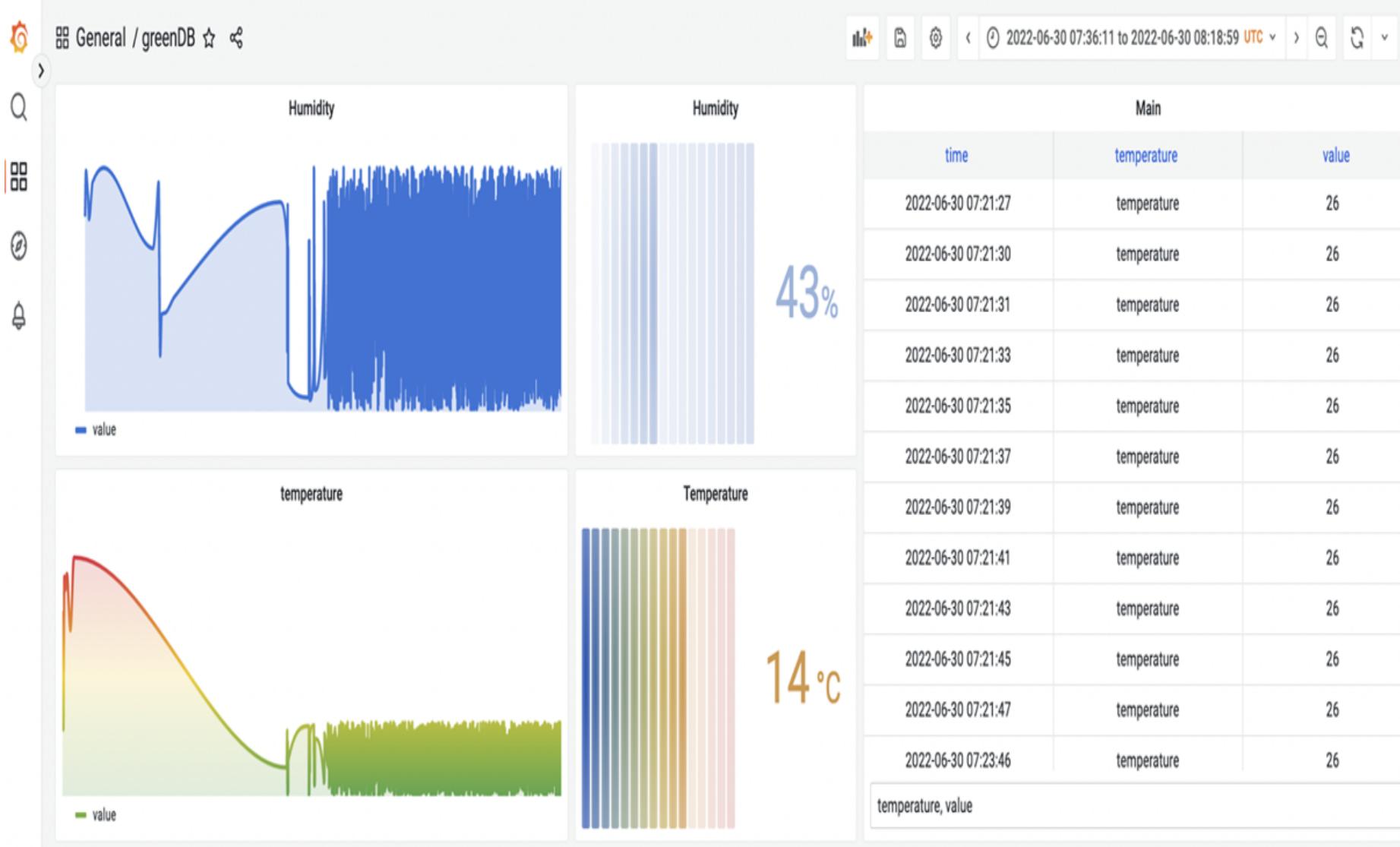
1. 기존의 센서 값을 받아오는 서버가 Node 기반의 코드이기 때문에 Nest 기반의 코드로 변경해야 합니다.
2. 기존의 센서 값에 대한 CRUD 중 수정하는 로직에서 동적쿼리를 적용해야 합니다.
3. 회원가입 및 로그인에 대한 비즈니스 로직에서 정보를 수정하고, 삭제하는 코드를 구현해야 합니다.

Front-End



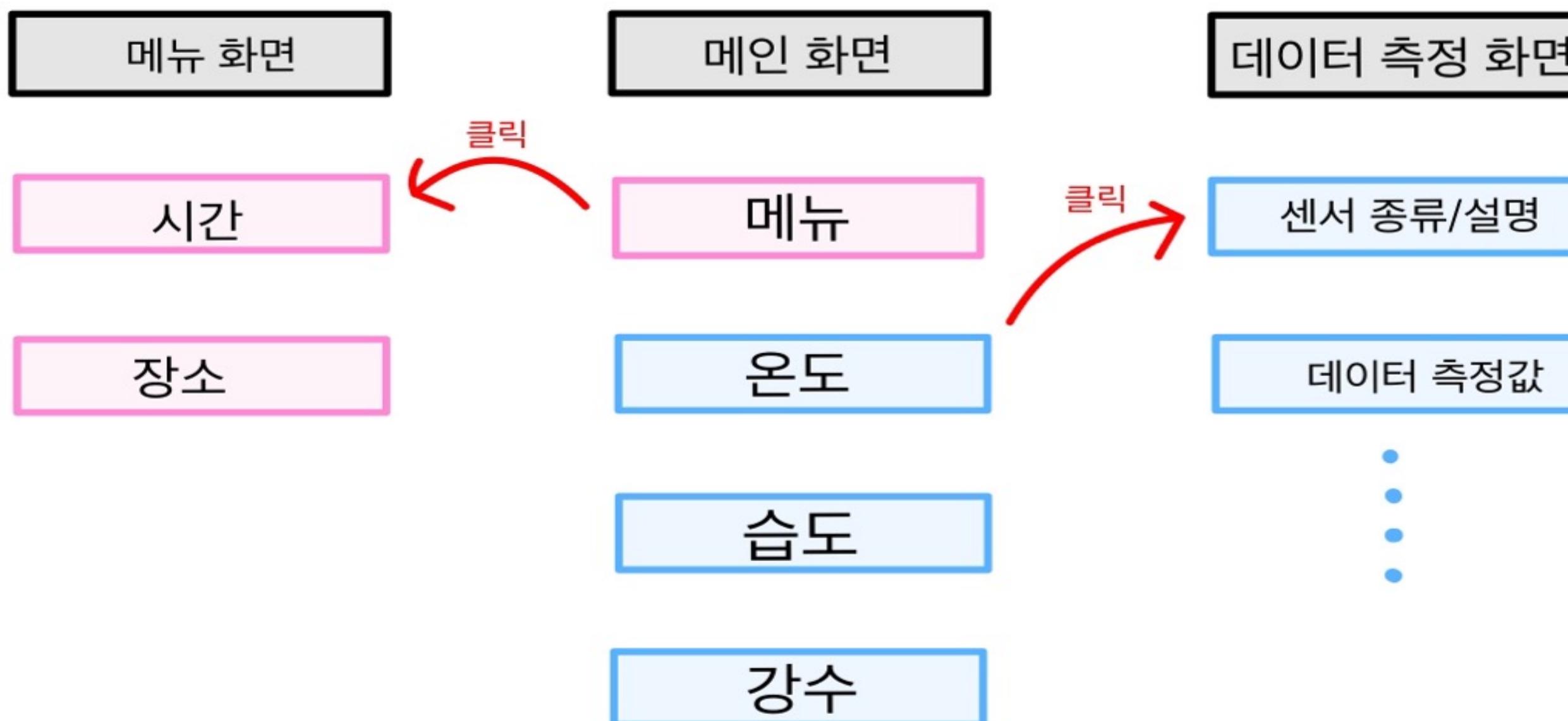
03

이전 발표 내용

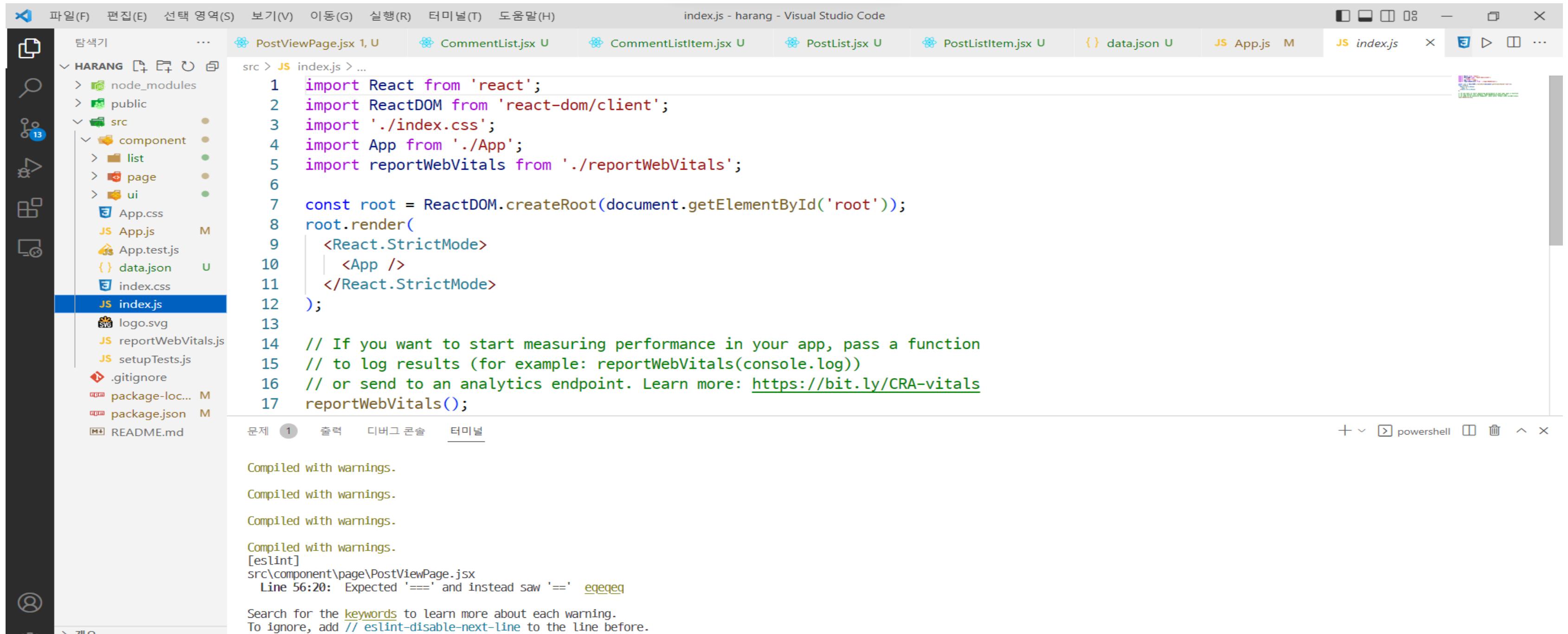


Grafana를 활용하여 데이터 시각화 및 알림 기능을 구현하였으며
React를 학습하여 Grafana의 내용을 웹 페이지로 구현할 예정

진행 상황 - React를 활용하여 웹 페이지 구현 (1) 구조



진행 상황 - React를 활용하여 웹 페이지 구현 (2) 과정



The screenshot shows a Visual Studio Code window with the title "index.js - harang - Visual Studio Code". The left sidebar displays a file tree for a project named "HARANG". The "src" directory contains "component", "list", "page", and "ui" sub-directories, along with "App.css", "App.js", "App.test.js", "data.json", "index.css", and "index.js". The "index.js" file is currently selected and open in the editor. The code in the editor is as follows:

```
src > JS index.js > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6
7 const root = ReactDOM.createRoot(document.getElementById('root'));
8 root.render(
9   <React.StrictMode>
10   |   <App />
11   </React.StrictMode>
12 );
13
14 // If you want to start measuring performance in your app, pass a function
15 // to log results (for example: reportWebVitals(console.log))
16 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
17 reportWebVitals();
```

Below the editor, the terminal tab is active, showing the output of the compilation process:

```
Compiled with warnings.
Compiled with warnings.
Compiled with warnings.
Compiled with warnings.
[eslint]
src\component\page\PostViewPage.jsx
Line 56:20: Expected '===' and instead saw '==' eqeqeq
Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.
```

진행 상황 - React를 활용하여 웹 페이지 구현 (3) 결과

The image displays two side-by-side browser windows showing the results of a React application development process.

Left Window (Main Page):

- Page Title: Smart Green Campus
- Header: 메뉴 (Menu)
- Content Boxes:
 - 온도 (Temperature)
 - 습도 (Humidity)
 - 강수량 (Precipitation)

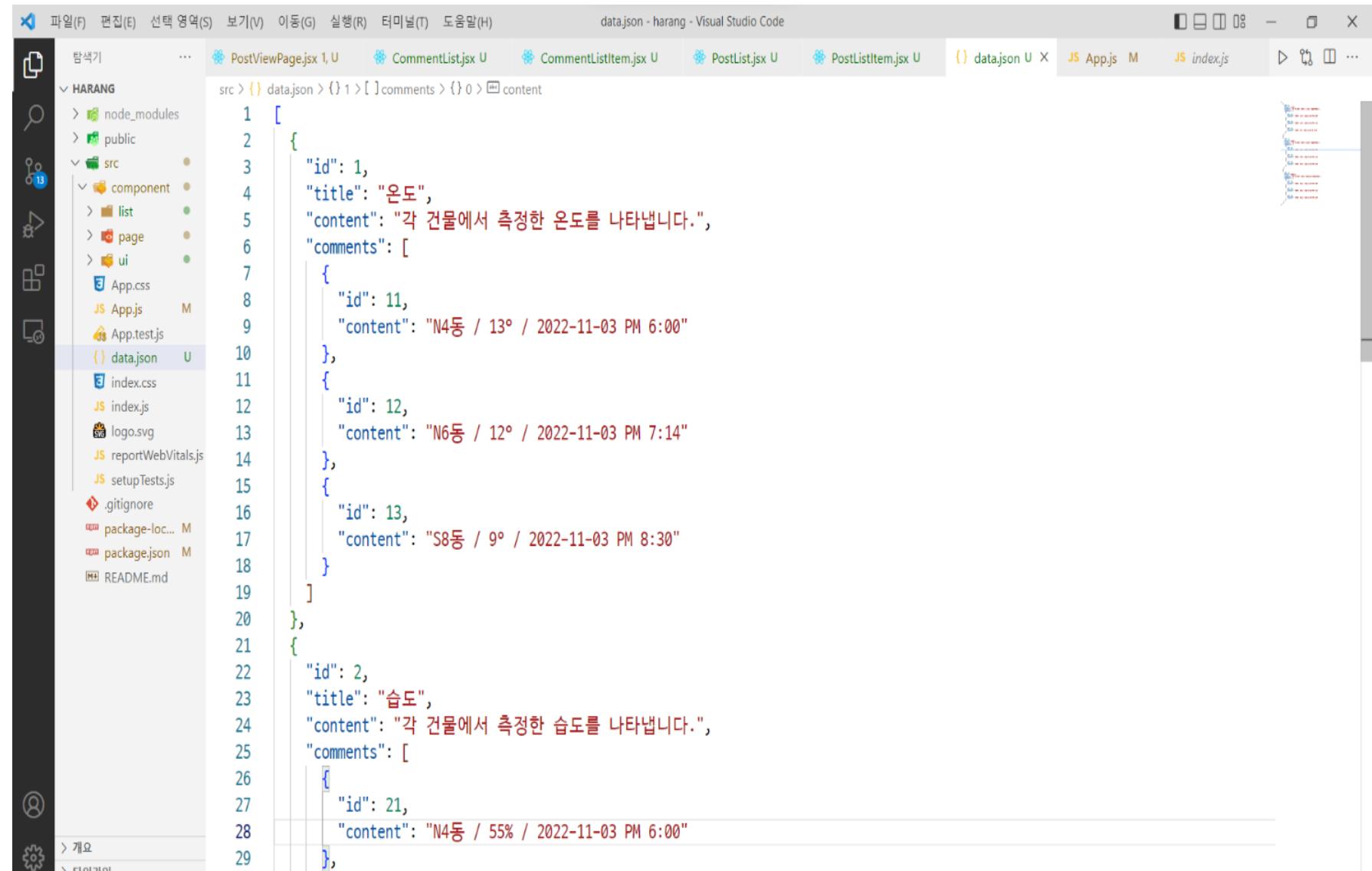
Right Window (Data Measurement Page):

- Page Title: Smart Green Campus
- Header: 뒤로 가기 (Back)
- Content Boxes:
 - 온도 (Temperature)
 - 각 건물에서 측정한 온도를 나타냅니다.
 - 데이터 측정 결과 (Data Measurement Result)
 - N4동 / 13° / 2022-11-03 PM 6:00
 - N6동 / 12° / 2022-11-03 PM 7:14
 - S8동 / 9° / 2022-11-03 PM 8:30

Labels:

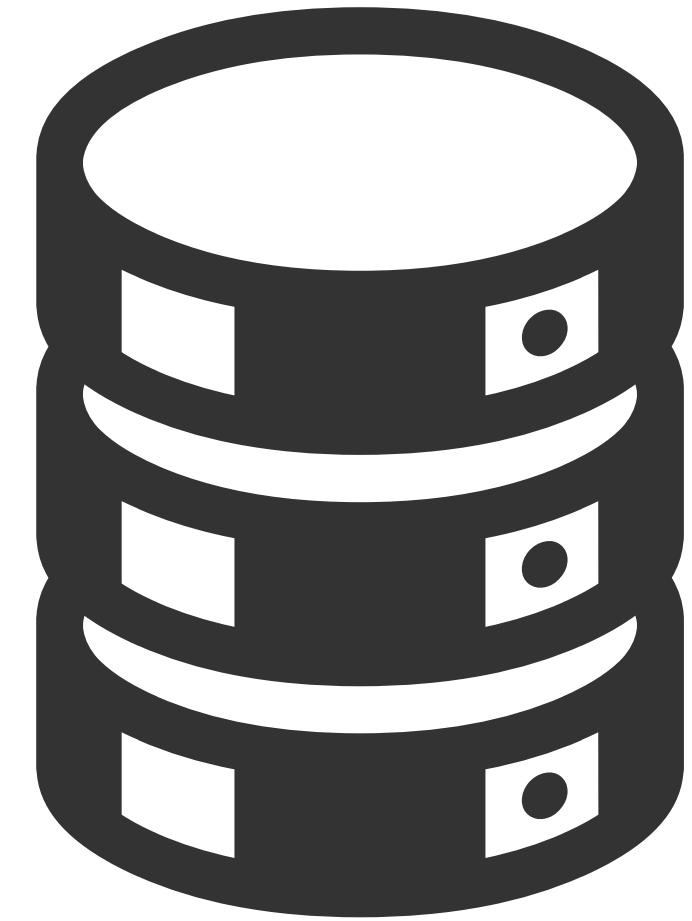
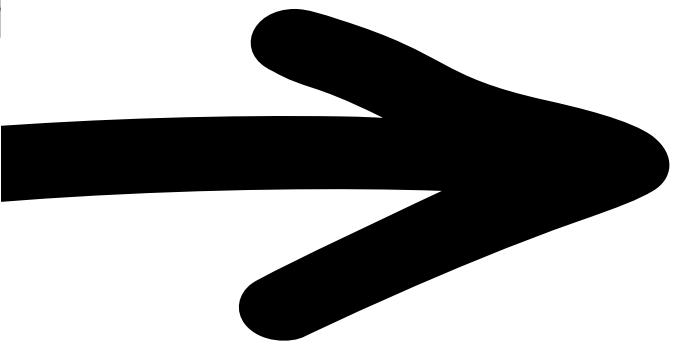
- 메인 화면 (Main Page) - Points to the left window.
- 데이터 측정 화면 (Data Measurement Page) - Points to the right window.

향후 계획



The screenshot shows a Visual Studio Code interface with a JSON file named 'data.json' open. The file contains data for posts and comments. The code is as follows:

```
src > {} data.json > {} 1 > [ ] comments > {} 0 > content
1 [
2   {
3     "id": 1,
4     "title": "온도",
5     "content": "각 건물에서 측정한 온도를 나타냅니다.",
6     "comments": [
7       {
8         "id": 11,
9         "content": "N4동 / 13° / 2022-11-03 PM 6:00"
10      },
11      {
12        "id": 12,
13        "content": "N6동 / 12° / 2022-11-03 PM 7:14"
14      },
15      {
16        "id": 13,
17        "content": "S8동 / 9° / 2022-11-03 PM 8:30"
18      }
19    ],
20  },
21  {
22    "id": 2,
23    "title": "습도",
24    "content": "각 건물에서 측정한 습도를 나타냅니다.",
25    "comments": [
26      {
27        "id": 21,
28        "content": "N4동 / 55% / 2022-11-03 PM 6:00"
29      },
30    ]
31  }
]
```



1. 임의로 만든 데이터 대신 데이터베이스 및 API 서버와 연동하여 실제 값 얻기
2. 웹 페이지 구조 구체화

**THANK
YOU**