

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

04. 결산 및 계획

MQTT Broker

01



이전 발표 내용

NestJS를 이용해 아두이노 센서 값 수신

```
~/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"}
```

@MessagePattern을 사용하고, 원하는 포맷으로 변경하여 서버를 통해서 아두이노의 센서 값을 수신 받았습니다.

진행 상황

NestJS를 이용하여 DB로 데이터 전송

```
@Module({
  metadata: {
    imports: [
      ClientsModule.register([
        {
          name: 'TEST_CLIENT',
          transport: Transport.MQTT,
          options: {
            hostname: 'broker.hivemq.com',
            port: 1883,
            protocol: 'mqtt',
          },
        },
      ]),
      HttpModule,
    ],
  }
})
```

axios 패키지를 사용하기 위한 HttpModule을 추가하였습니다.

```
const axios = require('axios');

axios
  .post('http://192.168.10.171:3000/sensors', { sensor })
  .then((res: AxiosResponse<any>) => {
    console.log(res);
  })
  .catch((error) => {
    console.log(error);
  });
}
```

DB파트에 데이터를 전송을 위해 axios의 post 기능을 활용했습니다.

진행 상황

NestJS를 이용하여 DB로 데이터 전송

```
[Nest] 49506 - 2023. 01. 09. 오후 7:34:31    LOG [InstanceLoader] AppModule dependencies initialized +1ms
[Nest] 49506 - 2023. 01. 09. 오후 7:34:31    LOG [NestMicroservice] Nest microservice successfully started
{ sensor_name: 'Temperature', location: 'N5', value: 3 }
{ sensor_name: 'Temperature', location: 'N5', value: 12 }
{ sensor_name: 'Temperature', location: 'N5', value: 8 }
{ sensor_name: 'Temperature', location: 'N5', value: 30 }
{ sensor_name: 'Temperature', location: 'N5', value: 26 }
{ sensor_name: 'Temperature', location: 'N5', value: 0 }
{ sensor_name: 'Temperature', location: 'N5', value: 0 }
{ sensor_name: 'Temperature', location: 'N5', value: 33 }
{ sensor_name: 'Temperature', location: 'N5', value: 14 }
{ sensor_name: 'Temperature', location: 'N5', value: 4 }
{ sensor_name: 'Temperature', location: 'N5', value: 19 }
{ sensor_name: 'Temperature', location: 'N5', value: 2 }
{ sensor_name: 'Temperature', location: 'N5', value: 21 }
{ sensor_name: 'Temperature', location: 'N5', value: 12 }
```

	id	sensor_name	location	value	user_id
12	15	TEMPERATURE	N5	25	10
13	16	TEMPERATURE	N5	22	10
14	17	TEMPERATURE	N5	22	10
15	18	TEMPERATURE	N5	29	10
16	19	TEMPERATURE	N5	29	10
17	20	TEMPERATURE	N5	27	10
18	21	TEMPERATURE	N5	18	10
19	22	TEMPERATURE	N5	12	10
20	23	TEMPERATURE	N5	12	10
21	24	TEMPERATURE	N5	12	10

실시간으로 DB파트에 데이터가 전송되는 것을 확인할 수 있습니다.

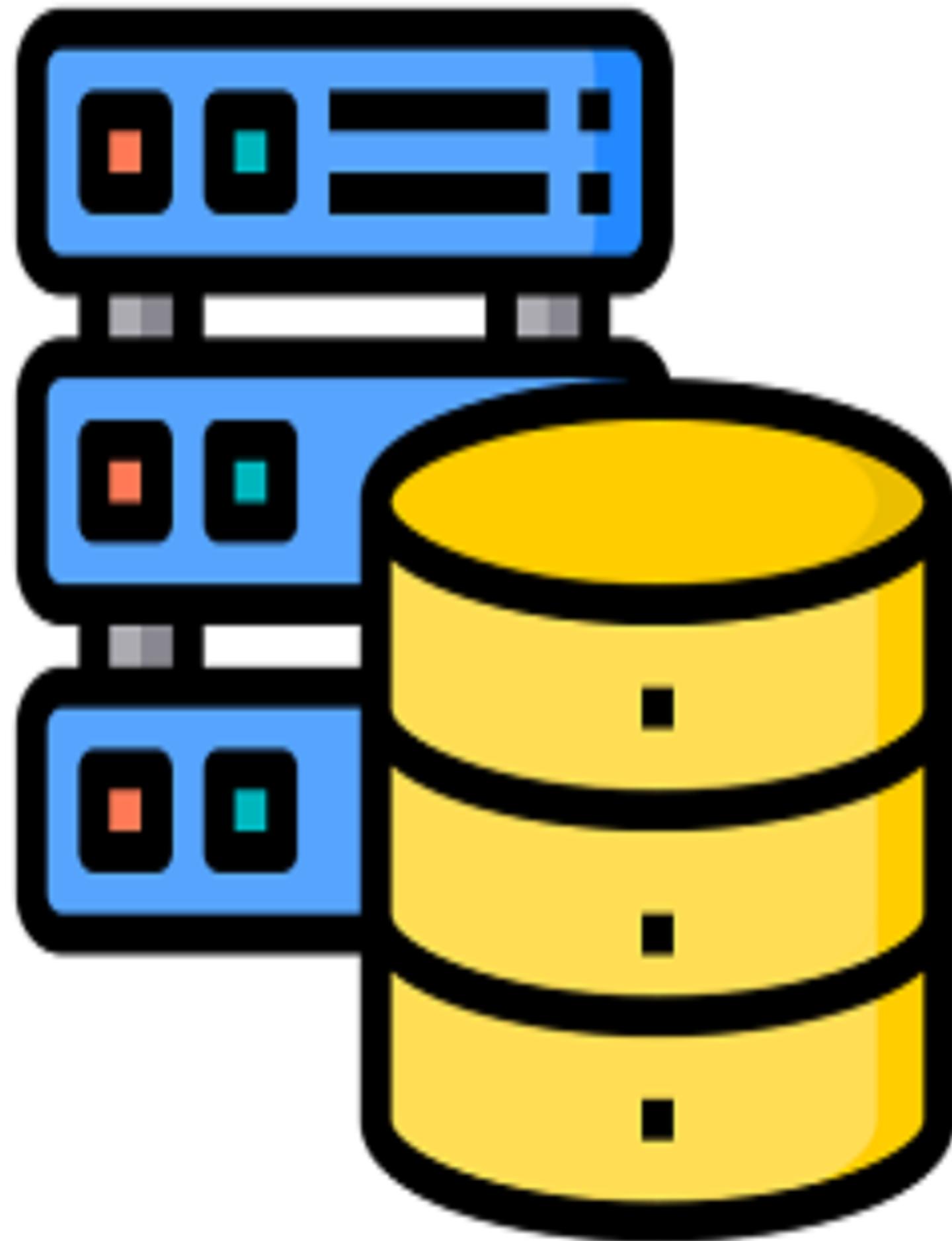
실제로 DB에 저장된 데이터를 확인할 수 있습니다.

향후 계획

1. DB팀에서 요청에 대한 사용자 인증 과정을 추가하였습니다.
-> 그에 맞는 회원가입, 로그인 로직을 구현할 예정입니다.

2. DB(Back-End)파트에서 구현한 사용자 인증에 대한 토큰을 발급받아, 데이터와 함께 전송하는 로직을 구현할 계획입니다.

Back-End



02

이전 발표 내용

NestJS로 구현한 회원 가입 및 로그인

The screenshot shows a database table on the left and a Postman API request on the right.

Database Table:

	id	username	password
1		dlehdduq1	
2		dlehdduq2	
3		dlehdduq3	
4		dlehdduq5	
5		smart-green-campus	

Postman Request:

- Method: POST
- URL: localhost:3000/auth/signup
- Body (JSON):

```
1 {  
2   "username": "smart-green-campus",  
3   "password": "1234"  
4 }
```

Auth Controller Code:

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

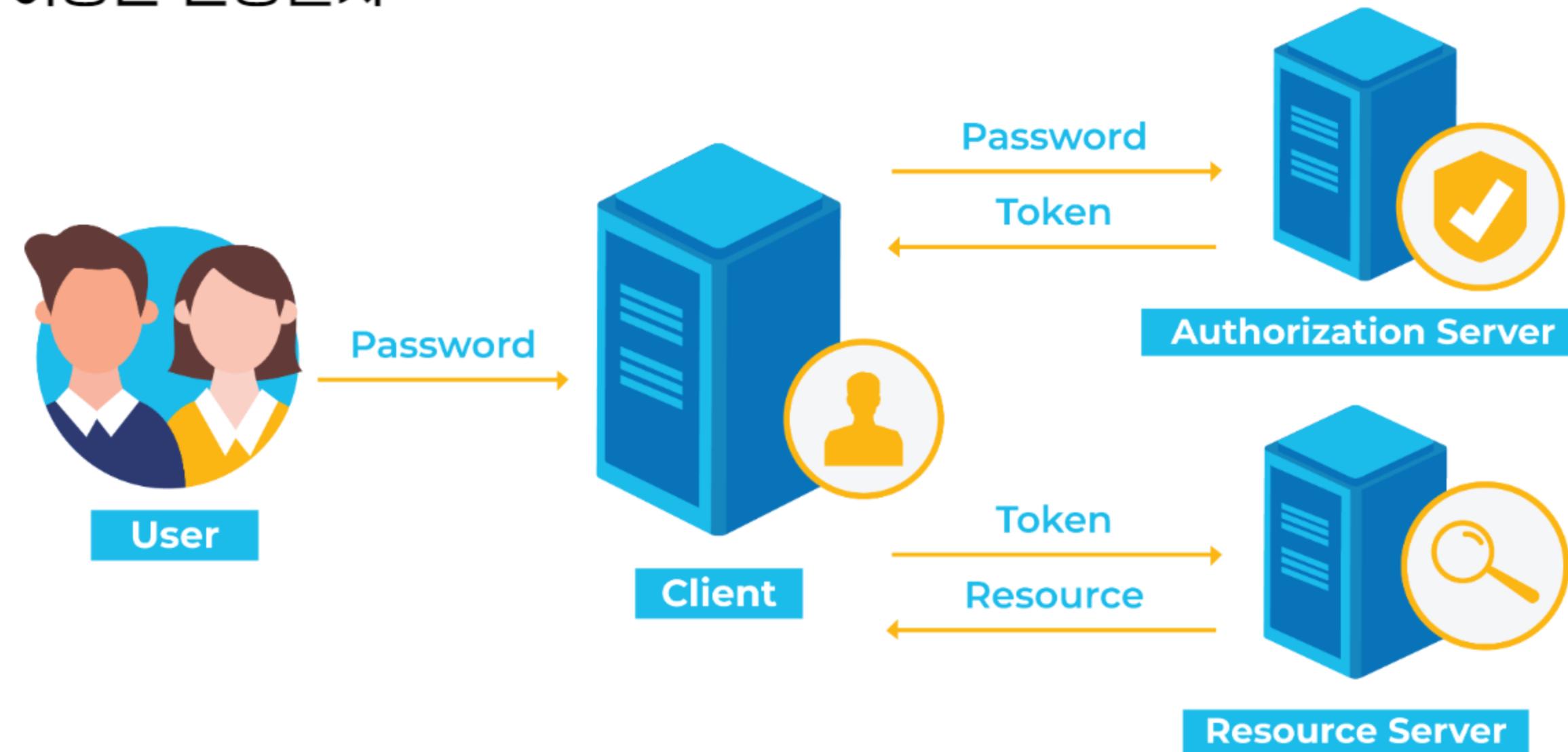
  @Post('/signup')
  signUp(
    @Body(ValidationPipe) authCredentialsDto: AuthCredentialsDto,
  ): Promise<void> {
    return this.authService.signUp(authCredentialsDto);
  }

  @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);
  }
}
```

진행 상황

토큰을 이용한 인증절차

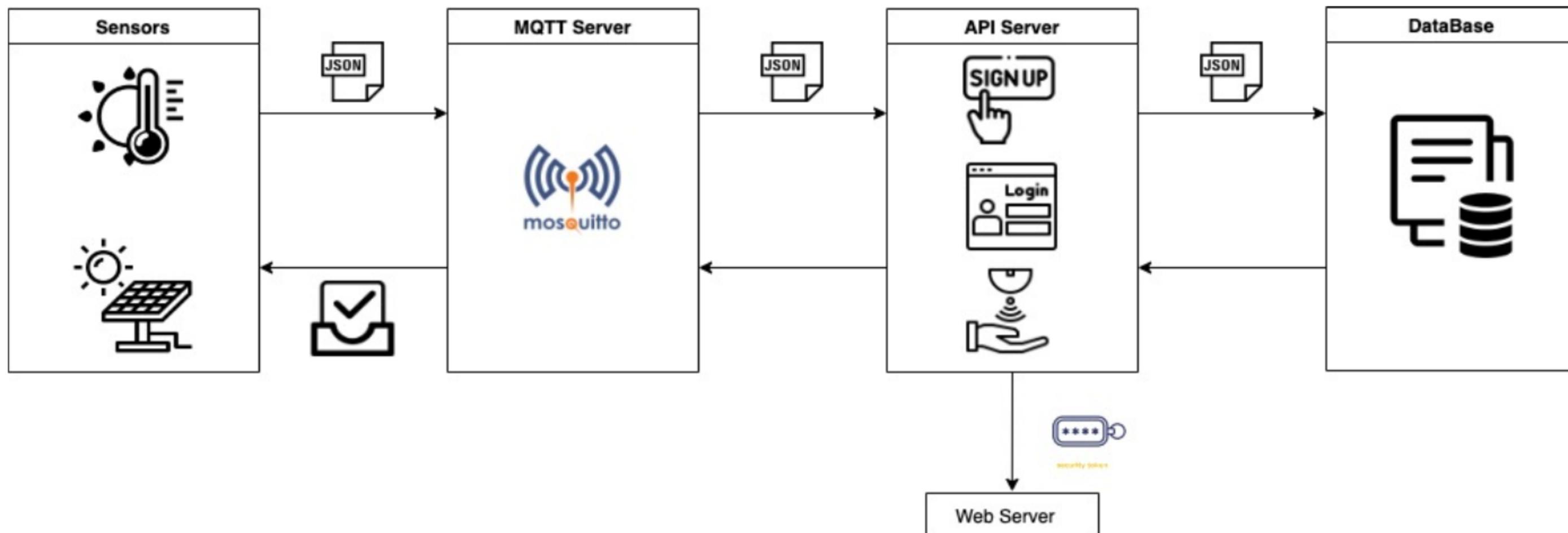


okta

токен은 발급 받으면 사용자의 브라우저에 저장되며, 관리자는 토큰에 대한 사용 제한을 설정하여 로그아웃하거나, 지정된 시간이 지나면 토큰을 자동으로 파기되도록 설정할 수 있습니다.

진행 상황

센서 CRUD



진행 상황

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

- Code Editor:** Displays the `sensor.controller.ts` file from a Nest.js application. The code implements a `SensorController` with methods for getting all sensors, getting a sensor by ID, creating a new sensor, updating a sensor value, and deleting a sensor.
- Database Explorer:** Shows the `nest-db` database connected to the `smart-green-campus` application. It lists tables such as `sensor`, `user`, and `sequences`. A table view shows sensor data with columns: `id`, `sensor_name`, `location`, `value`, and `user_id`.
- Terminal:** Displays the command `npm run start:dev` and the resulting log output from the Nest.js application. The logs show the application starting, routes being mapped, and various controller methods being triggered with their respective timestamps and details.

Controller에서 요청을 받으면 Service에서 처리합니다.
해당 코드는 인증 절차를 거쳐서 발급 받은 토큰으로 센서들의 데이터를 처리하는 코드입니다.

향후 계획

1. 요구사항 분석 및 DB 설계에 놓친 부분이 없는지 검토하고, 수정할 부분을 보완해야 합니다.
2. FE팀과 같이 연동해서 구현한 비즈니스 로직들을 검증해야 합니다.
3. MQTT 팀과 연동해서 토큰 발급 이후 비즈니스 로직들을 검증해야 합니다.

Front-End



03

이전 발표 내용

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "HARANG". The "index.js" file is selected.
- Code Editor:** Displays the content of "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();
```
- Terminal:** Shows the output of the compilation process, indicating warnings and ESLint errors.

Smart Green Campus

뒤로 가기

온도

각 건물에서 측정한 온도를 나타냅니다.

데이터 측정 결과

N4동 / 13° / 2022-11-03 PM 6:00

N6동 / 12° / 2022-11-03 PM 7:14

S8동 / 9° / 2022-11-03 PM 8:30

React.js를 활용하여 임의의 데이터를 보여줄 수 있는
간단한 웹 페이지를 구현하였습니다.

진행 상황 - 그래프 추가

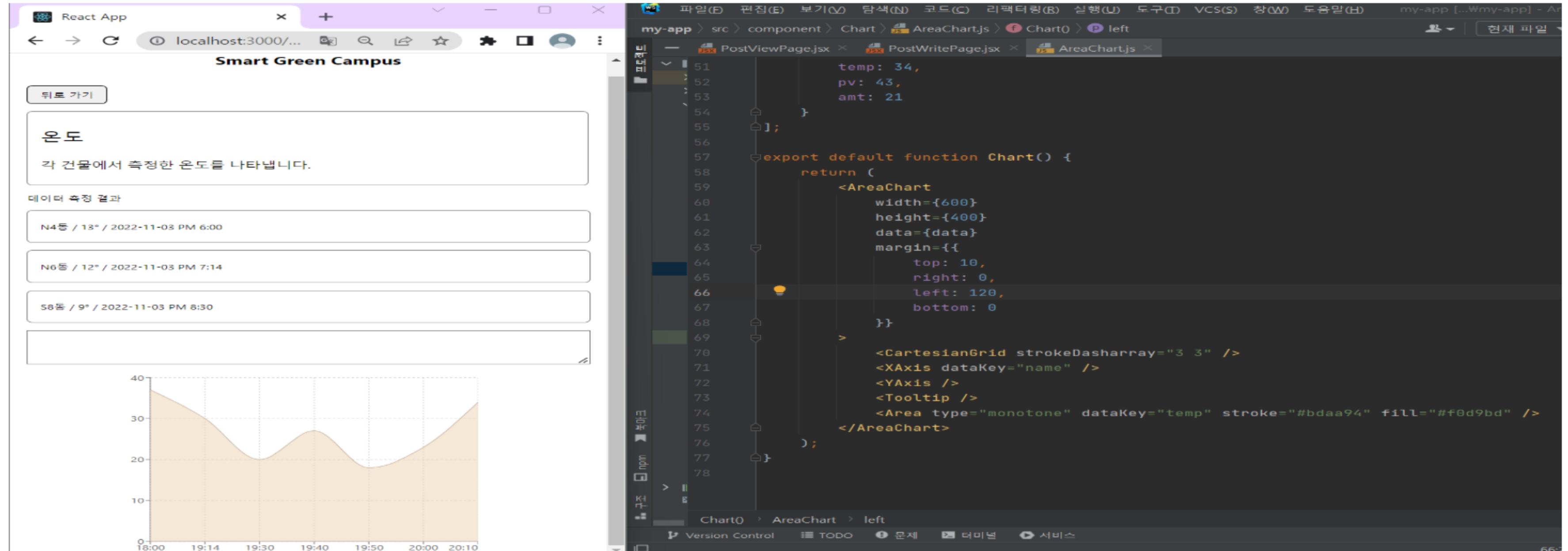
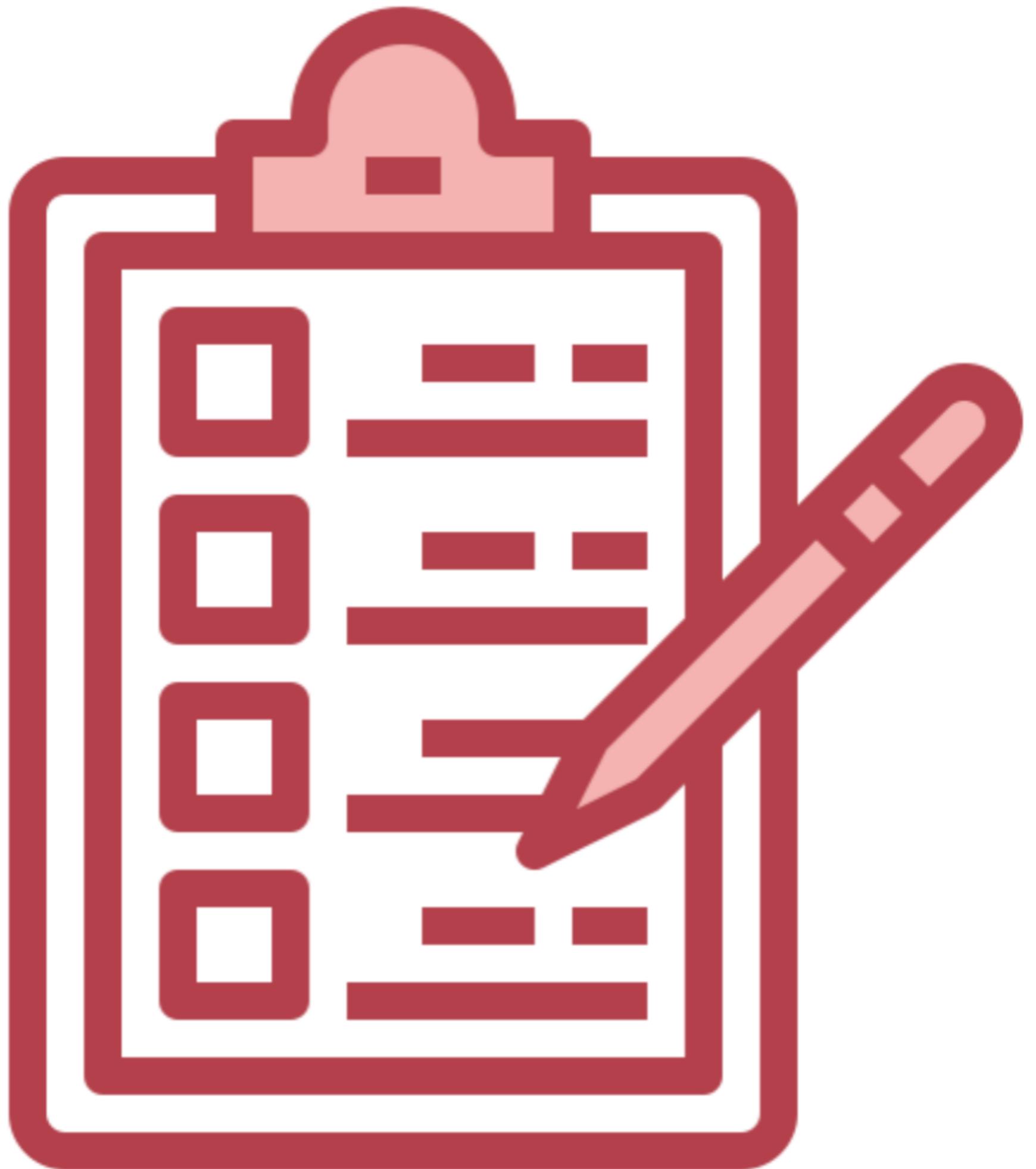


Chart Component를 생성하여 그래프를 추가해 초기 구상했던 디자인(Grafana)과 유사하게 구현하였습니다.

향후 계획

1. 그래프를 데이터 구체화
2. 웹 페이지 디자인 구체화
3. 임의의 값이 아닌 실제로 측정된 값을 표현하는 방법 조사



결산 및 계획

04

2022년 결산



Grafana를 이용한 DB 연동
및 데이터 시각화

2. 메인 페이지 UI 구성



Node.js를 이용한 WAS 구현

2. 기존 서버를 Nest.js로 구현
3. 회원가입 및 로그인 기능 추가



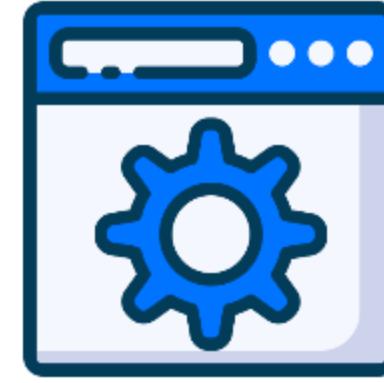
MQTT 통신을 위한 TOPIC 조
율

2. 받아온 토픽을 서버로 전달

2023년 계획



1. React를 이용한 프론트 서버 구현
2. 웹 페이지 UI 구성 및 백엔드와 연동



1. 프론트엔드와의 서버연동
2. 웹에서 구현하고자 하는 기능에 대한 API 구현



1. 토큰을 이용한 사용자 인증 절차 추가 -> 보안

**THANK
YOU**