NGK Assignment 3
Gruppe #17

Tue Alexander Jørgensen - 201810594

Asger Holm Brændgaard - 201810596

Jacob Nyvang Hansen – 201811364

Indhold

Introduktion

Gruppen har i denne opgave valgt at arbejde med opgave A, som går ud på at udvikle et webAPI, som skal kunne håndtere temperaturmålinger, som uploades fra indlejrede systemer. Det skal så være muligt at tilgå disse målinger via WebAPI'et.

Opgaven har følgende krav

- Tidspunkt (dato og klokkeslæt)
- Sted, som består af
    - Navn: string (Navn på lokaliteten)
    - Lat: double (gps-koordinatens latitude)
    - Lon: double (gps-koordiantens longitude)
- Temperatur – I grader celsius med 1 decimals nøjagtighed
- Luftfugtighed – et heltal som angiver luftfugtigheden i procent
- Lufttryk – i milibar med 1 deciamls nøjagtighed.

Yderligere skal der være en funktion som muliggøre at få en notifikation på klientsiden, når der modtages en ny måling på serveren.

Før der kan uploades en vejrobservation, skal en bruger være logget ind, dette gøres ved brug af JWT-token. Der vil være at finde 3 forskellige slags bruger: almen bruger, vejrstation og admin.

Løsningen bliver dels testet gennem unittest, men der bliver også lavet løbende test ved brug af Postman.

Projektet indeholder selve API'et samt en enkelt page, som kan præsentere liveopdatering af vejrobservationer.

Design og implementering

Database

Databasen er bestående af tre tabels en for placering, en for brugere, og en for vejrmålinger. Her er bruger uafhængig af de to andre, da denne kun anvendes til logindelen af systemet.

En bruger (*User*) har 6 kolonner: UserID, FirstName, LastName, Email, PwHash og Role. UserID er for at databasen kan skelne mellem dem, firstname, lastname og email giver sig selv. PwHash er Password og Role definere hvilken rolle brugeren har for systemet. Der er tre forskellige roller: Admin, User og WeatherStation.

De to andre tabeller af afhængige på den måde, at en vejrmåling skal have en placering for at kunne oprettes, men en placering kan sagtens have flere vejrmålinger. Det vil sige at det er en 1 til n relation mellem vejrmåling (*WeatherRepport*) og placering (*place).*

Ydermere, er et krav om liveopdatering blevet opstillet. Til at kunne håndtere klienter der får liveopdateringer, når en vejrstation poster en rapport, er SignalR blevet brugt. Denne opretter en hub, som klienter kan tilgå. Derved bliver en metode kaldt i hubben, som sender informationerne videre til klienterne, som dernæst selv kan afgøre hvordan disse skal repræsenteres. I denne sammenhæng er en enkelt page blevet oprettet, som udskriver rapporten. Dette kan ses i test og resultater samt præsentationsvideoen.

AccountController

For at opfylde kravet om sikkerhed til løsningen, er der udviklet en controller til brugere, som står for at håndtere oprettelsen af nye brugere og login af allerede eksisterende brugere.

Gruppen har implementeret account på den måde at en bruger logger ind med sin Email og et password, som brugeren selv har registreret, da brugeren oprettede sig. Her valideres der så på om en bruger allerede er oprettet, og hvis brugeren er det, bliver han tildelt en JWT-token, som bruges til at navigere resten af systemet.

```
[HttpPost(template: "login"), AllowAnonymous]
0 references | Tueaj, 1 day ago | 1 author, 1 change
public async Task<ActionResult<TokenDto>> Login(UserDto login)
{
    login.Email = login.Email.ToLower();
    var user = await _context.User.Where(u :User => u.Email == login.Email) // IQueryable<User>
        .FirstOrDefaultAsync(); // Task<User>
    if (user != null)
    {
        var validPwd :bool = Verify(text: login.Password, hash: user.PwHash);
        if (validPwd)
        {
            var token = new TokenDto();
            token.JWT = GenerateToken(user);
            return token;
        }
    }
    ModelState.AddModelError(key: string.Empty, errorMessage: "Forkert brugernavn eller password");
    return BadRequest(ModelState);
}
```

Figur 1: AccountController login funktion

Her ses det i den grønne boks at der bliver søgt i databasen efter en Email. Der valideres så på om det medgivende password er rigtigt, det kan ses i den blå boks. Til sidst bliver der så genereret en ny JWT-token, som tilknyttes brugerens login session, dette fremgår i den røde boks.

WeatherReportController

WeatherReportController er den controller der står for at de vejr opdateringer der bliver sendt til APIet bliver lagt ind i databasen. Samtidigt er det også denne som står for de forskellige get metoder: Få de tre nyeste vejrmålinger, få vejrmålinger mellem to datoer og at kunne hente alle vejrmålinger for en bestemt dag.

```
[HttpPost( template: "postReport")]
[Authorize(Roles = "WeatherStation")]
0 references
{
    if (report == null)
    {
        return BadRequest( error: new {errormessage = "Bad report"});
    }
    var place = await _dbController.FindPlaceById(report.PlaceId);

    if (place.Name == null )
    {
        return BadRequest( error: new {errormessage = "Place doesnt exist"});
    }
    WeatherRepport newReport = new WeatherRepport();
    {
        newReport.Place = await _dbController.FindPlaceById(report.PlaceId);
        newReport.PlaceId = report.PlaceId;
        newReport.AirPressure = report.AirPressure;
        newReport.Humidity = report.Humidity;
        newReport.Temp = report.Temp;

        if(report.Time == default)
        {
            newReport.Time = DateTime.Now;
        }
        else
        {
            newReport.Time = report.Time;
        }
    }
    report.Time = newReport.Time;
    report.PlaceName = newReport.Place.Name;

    await _hub.Clients.All.SendAsync( method: "SendReport", report);

     _dbController.AddAndSaveWeatherReport(newReport);

     return Created( uri: newReport.ToString(), newReport);
}
```

Figur 2: post funktion i WeatherControl

Overstående figur viser post funktionen, som opretter en vejrmåling i databasen, her ses først i den blå boks bliver placeringen valideret, hvis den findes bindes den til den nye rapport, hvis placeringen ikke findes i databasen, giver den et badRequest retur, som også kan ses under Test og Resultater.

I den røde boks bliver resten af vejrmålingen oprettet, her bliver der også valideret om time er sat, og hvis den ikke er det, sætter den en default værdi for time.

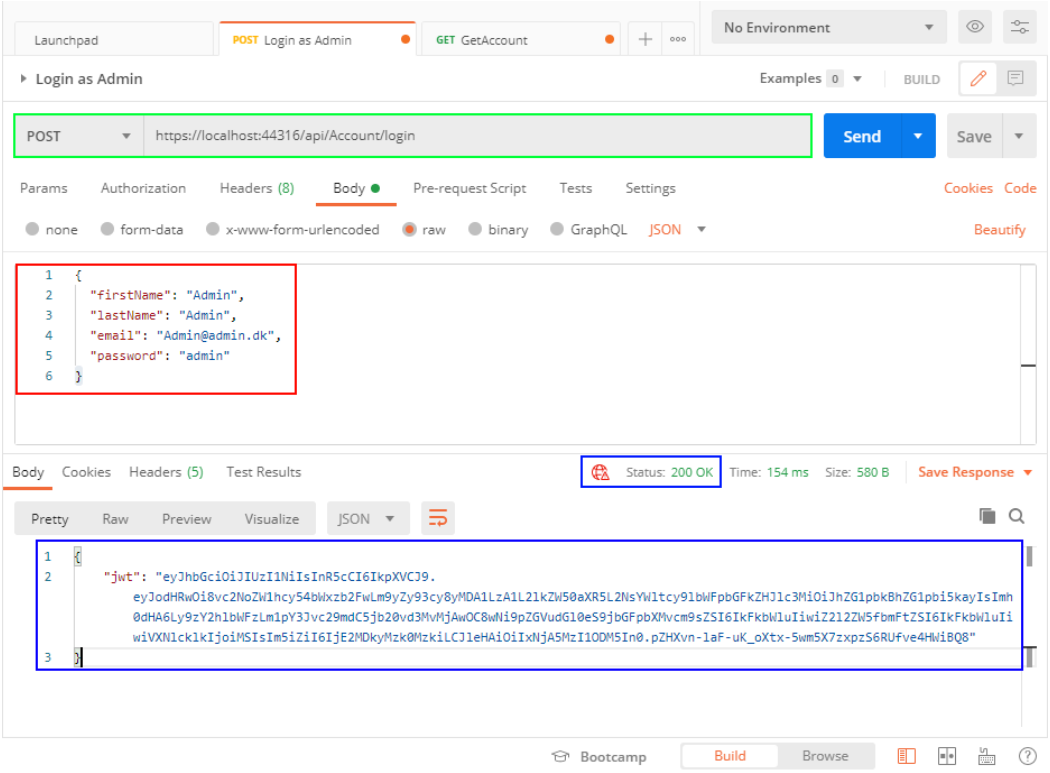
I den grønne boks, sender den så til Hubben, som gør at det bliver udskrevet på Client Side, som også vises i præsentationsvideoen.

Test og resultater

Postman test

Til at teste api'et er der anvendt Postman[1]. På den måde har det været muligt at lave et kald til APIet og se hvad der bliver returneret, og om apiet håndtere ens request rigtigt. Det vil sige at Postman fungerer som en black box test af ens api. Da man giver et input og kigget på outputtet.

Først har gruppen kigget på Authentication, som ses på nedenstående figur

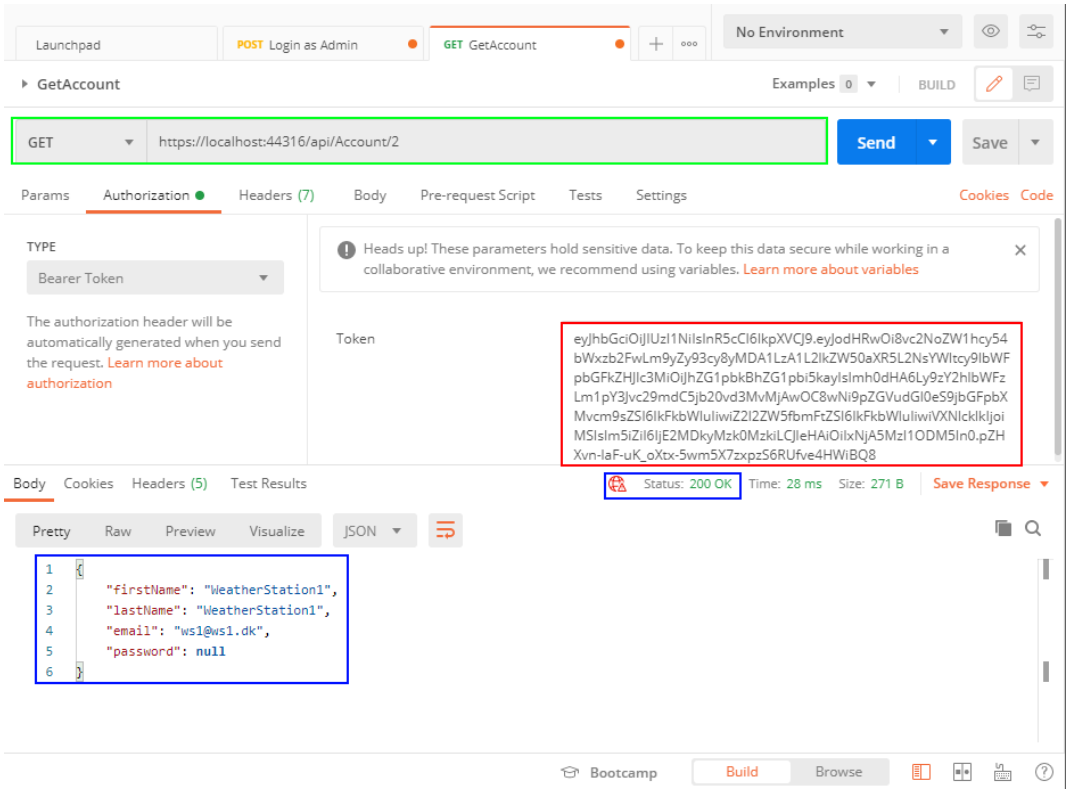


Figur 3:Postman login request, modtaget JWT token

Her ses det at når login bliver kaldt i den røde figur, får man en JWT-token retur i den blå boks, denne bruges i resten af programmet til at validere om brugeren har rettigheder til at udføre en givende kommando.

Understående figur viser Get Account, som man skal være admin for at kunne udføre, og som det ses på overstående figur er brugeren logget ind som admin. Derved bliver den bruger man søger efter returneret.

[1] https://www.postman.com/



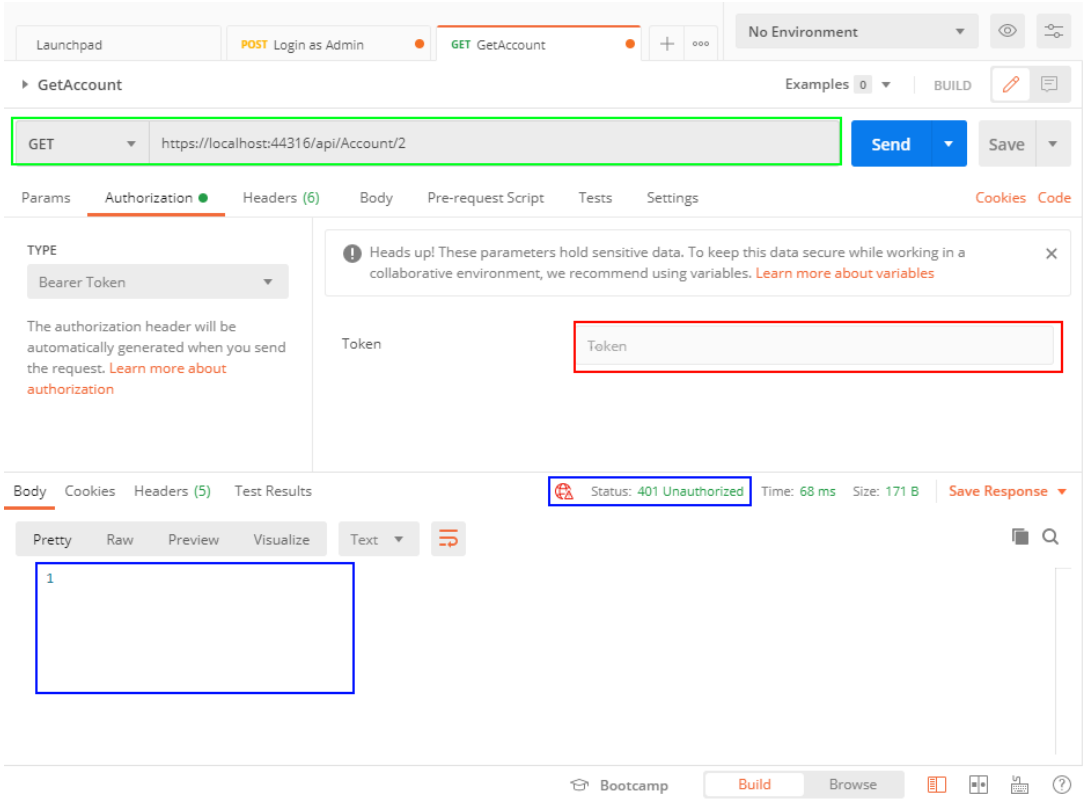
Figur 4: postman Get Account med JWT token

	UserId	FirstName	LastName	Email	PwHash	Role
	1	Admin	Admin	admin@admin....	$2a$10$C5PIAjl...	Admin
▷	2	WeatherStation1	WeatherStation1	ws1@ws1.dk	$2a$10$n686S7...	WeatherStation
	3	WeatherStation2	WeatherStation2	ws2@ws2.dk	$2a$10$io0n0F...	WeatherStation
	4	WeatherStation3	WeatherStation3	ws3@ws3.dk	$2a$10$SBZ6gk...	WeatherStation
	NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Figur 5: Udklip fra database view

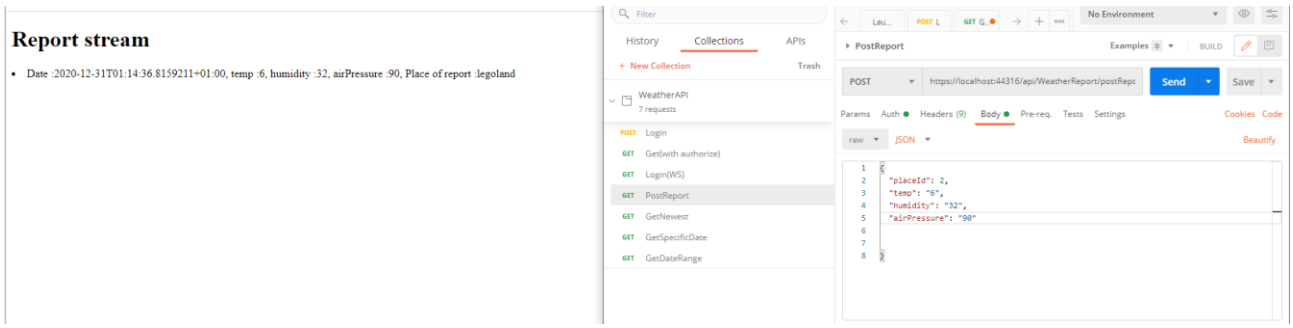
Overstående figur viser den data der opbevares i databasens tabel for users, her ses det at UserId 2, som det ses i den grønne boks i figur 3, er det ID 2 vi søger efter. Det ses også i den grønne boks at dette stemmer overens med den entitet der er i databasetabellen.

Understående figur her er JWT token blevet fjernet, for at eftervise at man skal have en ”rigtig” token før man får adgang, her ses det blandt andet at der bliver returneret en 401, som er markeret i den blå boks, som er acces denied, og det ses at der ikke gives et svar med.



Figur 6: Postman get Account uden JWT token

Nedenstående figure viser hvorledes en WeatherReport bliver sendt til hubben, og derved præsenteret på en klient, som i dette tilfælde er en page.



Figur 7. Post af WeatherReport præsenteret på page

Unit test af controller klasser

For at validere funktionaliteten for controllerklasserne er der oprettet et Unittest projekt, som undersøger funktionaliteten af de individuelle controllers, på den måde er gruppen sikre på at controlleren fungerer som ønsket. Der er blevet testet udvalgt funktionalitet. Én af disse test, vil blive præsenteret her. Resten vil være at finde i projektet.

```
[Test]
0 references
public void GetReportsFromDateReportstInDB_SpecificDate_NoErrorDetected()
{
    //Arrange
    var reportList = new List<WeatherReport>();
    reportList.Add( item: new WeatherReport { AirPressure = 5, Humidity = 3, Id = 1, Place = new Place(), PlaceId = 1, Temp = 10, Time = DateTime.Now });
    reportList.Add( item: new WeatherReport { AirPressure = 6, Humidity = 4, Id = 2, Place = new Place(), PlaceId = 2, Temp = 11, Time = DateTime.Now });
    reportList.Add( item: new WeatherReport { AirPressure = 7, Humidity = 5, Id = 3, Place = new Place(), PlaceId = 3, Temp = 12, Time = DateTime.Now });

    DateDto dt = new DateDto();
    dt.StartDate = "Dec 29, 2020";
    dt.EndDate = "Dec 31, 2020";


    databaseController.GetReportsBetweenTwoDates( dateStart: DateTime.Parse(dt.StartDate),  dateEnd: DateTime.Parse(dt.EndDate)).Returns(reportList);


    //Act
    var result = (_uut.getReportsBetweenTwoDates(dt).Result.Result as CreatedResult);

    //Assert

    Assert.That(result.StatusCode, Is.EqualTo(201));
}
```

For at teste WebAPI-kaldende, vil man asserte på StatusCode, da disse fortæller brugeren hvilke eventuelle fejl der er opstået. I denne test sørges der for, at databaseControlleren returnerer en list af reports. Dernæst tjekkes der, om hvor vidt StatusCoden for controller-kaldet returnerer den korrekte værdi. I dette tilfælde er det en 201 StatusCode.

Konklusion

Det er blevet gjort klart under denne opgave, hvor brugbar autorisering af brugeren med JWT-tokens er. Udover denne, kunne gruppen også se fordelene ved at indlejre funktionalitet i et WebAPI, som dernæst kan tilgås ”udefra”.

Undervejs i processen blev der tilføjet og fjernet modeller, da visse problematikker opstod. Dette viser vigtigheden af gode modeller, og vigtigheden af, at lægge en klar plan for, hvad systemet skal kunne håndtere. Dette blev især gjort klart, da oprettelse af WeatherReports samt signalR websocket fandt sted. Det er afgørende af der bliver oprettet Dto’er, således databasen delen kan abstraheres væk fra controllers. Derved sikrer man sig imod Json cycles, som kan være svære at håndtere.

Rent testmæssigt, er der blevet opnået erfaring inden for Postman, samt brug af Nunit, til test af et WebApi. Postman muliggør hurtige test undervejs i udviklingen. Ydermere muliggører Postman også god dokumentation ved end udvikling. Nunit’s funktionalitet gør det derfor også muligt at lave unittests på de enkelte controllers, og derfor kan man sikre sig korrekt funktionalitet af Api’et.