

Produkttrapport

SmartWeight

Daniel Simonsen
21-11-2022

Titelblad

TECHCOLLEGE

Techcollege Aalborg,
Struervej 70,
9220 Aalborg

Elev:

Daniel Simonsen

Firma:

Skolepraktik

Projekt:

SmartWeight

Uddannelse:

Datateknikker m. speciale i
programmering

Projektperiode:

19/09/2022 – 23/11/2022

Afleveringsdato:

23/11/2022

Fremlægelsesdato:

24/11/2022

Indledning

Rapporten indeholder produktets opsætning, brugervejledning og teknisk produktdokumentation. Her kan du læse, hvordan produktet fungerer, hvad der kræves som kunde for at kunne bruge produktet, og andre former for opsætning.

Indholdsfortegnelse

| | |
|-----------------------------------|----|
| Titelblad..... | i |
| Indledning..... | ii |
| Om produktet..... | I |
| Flowchart..... | I |
| Model Library | I |
| Backend API | 2 |
| Frontend Cross-platform App..... | 3 |
| Embedded Hardware | 5 |
| Brugervejledning | 7 |
| Problemformulering..... | 7 |
| Anvendelse | 7 |
| Login..... | 7 |
| Connections | I |
| Vægtmåling..... | 0 |
| Overview..... | 0 |
| Teknisk produktdokumentation..... | I |
| ASP.NET API..... | I |
| Connections | I |
| Measurements | I |
| Partial Measurements | 2 |
| Users | 3 |
| Weight | 3 |
| SQL Database | 5 |
| Users | 5 |
| Weights | 5 |
| Measurements | 5 |
| Connections | 6 |
| MAUI Cross-platform App..... | 6 |
| Login..... | 6 |
| Overview..... | I |
| Connections | I |
| ESP8266 Embedded Enhed | I |

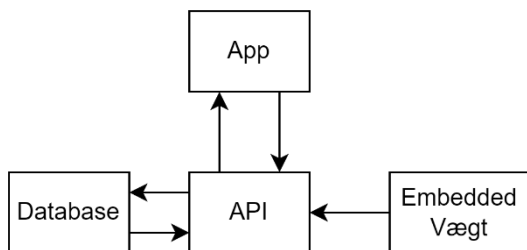
Om produktet

Flowchart

Den store del af produktet bruges via appen. Det er i appen, du logger ind på din konto, forbinder din konto til din vægt, og læser dine tal. Appen snakker derfor med API'et ofte, da API'et håndterer det meste logik i produktet.

Når en vægt sender en måling, laver den en POST request til API'et med dens oplysninger om målingen.

Alt der kunne gemmes, bruger-, vægt- og forbindelsesoplysninger, gemmes i SQL databasen, og hentes igen af API'et.



Model Library

I dette Visual Studio projekt, har jeg alle mine modeller, klasser og enums, jeg kunne få brug for i backends og frontends. Det er her, jeg definerer min klassestruktur og også det projekt, der håndterer EntityFramework og dens migrations, altså code-first til databasehistorikken.

[ApiClient.cs](#)

ApiClient er en klasse jeg bruger i både API og app. Den fungerer som en wrapper til HttpClient, hvor jeg let kan sende requests til mine endpoints via enums fra biblioteket.

```
#region Request => SimpleResponse
3 references
public async Task<SimpleResponse> Post<Content>(Endpoints endpoint, string url = "", Content? content = default)
    => await Request(HttpMethod.Post, endpoint, url, content);
3 references
public async Task<SimpleResponse> Post<Content>(Endpoints endpoint, Content? content = default)
    => await Request(HttpMethod.Post, endpoint, "", content);

4 references
public async Task<SimpleResponse> Get<Content>(Endpoints endpoint, string url = "")
    => await Request<object>(HttpMethod.Get, endpoint, url);
0 references
public async Task<SimpleResponse> Put<Content>(Endpoints endpoint, string url = "", Content? content = default)
    => await Request(HttpMethod.Put, endpoint, url, content);
3 references
public async Task<SimpleResponse> Delete<Content>(Endpoints endpoint, string url = "")
    => await Request<object>(HttpMethod.Delete, endpoint, url);

5 references
private async Task<SimpleResponse> Request<Content>(HttpMethod method, Endpoints endpoint, string url = "", Content? content = default) =>
    new SimpleResponse($"{ApiUrl}/{_endpoints[endpoint]}/{url}",
        await _client.SendAsync(
            new HttpRequestMessage(method, $"{ApiUrl}/{_endpoints[endpoint]}/{url}")
            {
                Content = content is not null ? JsonConvert.Create(content) : null
            }
        )
    );
#endregion
```

PRODUKTRAPPORT
Code-snippet af ApiClient.cs

Billedet ovenover viser request delen af min ApiClient. ApiClient har 5 public metoder – 4 af dem er de klassiske http crud operationer, post, get, put & delete, og den sidste er en overload af post, til hvis der en situation, hvor jeg skal bruge mere i URL'en end det, min Endpoints enum tilbyder. Post og Put metoderne tager begge imod en Content generic i stedet for object, da jeg synes, at det giver mere mening.

Alle de offentlige metoder peger hen imod den private Request metode. Request laver en ny SimpleResponse, som er en wrapper til HttpResponseMessage. SimpleResponse tager imod request URL'en og HttpResponseMessage. Her bruger jeg bl.a. ApiUrl, _client og _endpoints til konstruering af den fulde URL. Alle properties laves i ApiClient's constructor.

```
2 references
public ApiClient()
{
    ApiUrl = "https://localhost:7065/api";
    _client = new HttpClient();
    _endpoints = new Dictionary<Endpoints, string>()
    {
        {Endpoints.CONNECTIONS, "connections"},
        {Endpoints.MEASUREMENTS, "measurements"},
        {Endpoints.MEASUREMENTS_OVERVIEW, "measurements/overview"},
        {Endpoints.MEASUREMENTS_PARTIALS, "measurements/partials"},
        {Endpoints.USERS, "users"},
        {Endpoints.USERS_LOGIN, "users/login"},
        {Endpoints.WEIGHTS, "weights"},
    };
}
```

ApiUrl giver selvfølgelig mening, at det er production url på det hostede API, men da projektet kun køres lokalt, bruger jeg localhost.

Backend API

Min backend API består af et ASP.NET projekt. Projektet er bindelaget til alt der foregår i systemet, da den både skal snakke sammen med database og app, og modtage requests fra den embeddede vægt.

Med ASP.NET får jeg Swagger til rådighed, som kan læses og forstås i Teknisk Produktdokumentation.

BaseController.HandleMeasurement

Da vægten ikke har en reference til brugeren, der bruger vægten, skal jeg kunne forbinde de 2 objekter med hinanden, for at få lavet en korrekt vægtmåling. Her introducerede jeg Connections, hvor en bruger kan oprette forbindelse til en vægt og dermed være forbundet til vægten, men jeg skal stadigvæk kunne sætte dem sammen i en vægtmåling.

PRODUKTRAPPORT

Her bruger jeg min HandleMeasurement metode, i min BaseController, som alle mine controllere arver fra. Hver gang der kommer ny forbindelse til en vægt og hver gang der kommer en ny partial measurement fra min vægt, kalder controllerne HandleMeasurement.

```
/// <summary> Checks if there's a connection between any ids, and if so, add the ...
2 references
protected async Task<IActionResult> HandleMeasurement(int id, MeasurementPartialTypes type, IActionResult result)
{
    // Is weight connected to a user, if not, then maybe user connects after weight was used
    Connection? conn =
        type == MeasurementPartialTypes.USER ? _context.Connections.FirstOrDefault(c => c.UserId == id) :
        type == MeasurementPartialTypes.PARTIAL_MEASUREMENT ? _context.Connections.FirstOrDefault(c => c.WeightId == id && c.IsConnected) :
        null;
    if (conn is null) return result;

    List<Measurement> measurements = _context.Measurements.ToList();
    // Get partial measurements, if any
    List<PartialMeasurement> partialMeasurements = measurements
        .Where(m => m.UserId == null
            && m.WeightId == conn.WeightId)
        .ToList<PartialMeasurement>();
    if (!partialMeasurements.Any()) return result;

    // Delete connection regardless of what happens next
    await DeleteAsync(Endpoints.CONNECTIONS, $"{conn.UserId}?fromApp=true");

    // Get user from connection
    if (_context.Users.Find(conn.UserId) is null) return StatusCode(
        StatusCodes.Status500InternalServerError,
        "User not found - connection is corrupted.");

    foreach (Measurement measurement in partialMeasurements.Cast<Measurement>())
    {
        measurement.UserId = conn.UserId;
        _context.Entry(_context.Measurements.Find(measurement.Id)).CurrentValues.SetValues(measurement);
    }

    await _context.SaveChangesAsync();

    return Ok("Measurement saved.");
}
```

Code-snippet af BaseController.HandleMeasurement

HandleMeasurement skal bruges i både connection og partial measurement. Derfor kan jeg ikke være sikker på, hvilket id jeg får i min parameter. Jeg bruger det id til at finde en connection alt efter værdien af min type parameter. MeasurementPartialTypes er en enum, der kun bruges i HandleMeasurement, til at finde ud af, om id'et er fra en bruger eller en partial measurement. Hvis jeg ikke finder en connection eller nogle partial measurements, returnerer jeg det originale resultat, fra endpointet HandleMeasurement blev kaldt fra.

Hvis der er en connection og en partial measurement forbundet til connections vægt id, kan jeg håndtere resten af logikken.

Forbindelsen mellem bruger og vægt bliver slettet, da det er ment, at du kun skal vejes én gang per forbindelse.

Derefter tjekker jeg, om brugeren stadig findes i databasen for at sikkert tilknytte brugerrelation til mine målinger. Brugeren burde altid eksistere, og gør den ikke, er det en intern fejl.

Alle målingernes UserId bliver nu sat til Connections brugerid for at fuldføre processen. Til sidst skal alt selvfølgelig gemmes og returnerer en 200 statuskode.

Frontend Cross-platform App

Min app er lavet i .NET MAUI og kompileres til Android, iOS og Windows. Det er en simpel 3-siders app, som er brugt til forbindelse mellem bruger og vægt, og til at se brugerens vægtoversigt.

ConnctionsViewModel.OnNewConnection

I min forbindelsesside har jeg et brugerinput felt, hvor brugeren kan indtaste id'et på vægten, brugeren gerne vil forbindes til. Da mine forbindelser er databinded, skal jeg indsætte en ny forbindelse via inputfeltet, som kræver lidt mere arbejde. Når brugeren trykker på **Add Connction**, kalder den NewConnectionCommand, som har OnNewConnection som callback.

```
1 reference
private async void OnNewConnection()
{
    try
    {
        if (User is null) throw new AlertException("Invalid login state", "You must be logged in to add a connection!");
        if (!int.TryParse(WeightIdInput, out int weightId)) throw new AlertException("Invalid id", "Weight ids must be integers");
        else if (Connections.Any(connVm => connVm.Connection.WeightId == weightId))
            throw new AlertException("Already added", $"Connection to weight {weightId} is already added.");

        SimpleResponse res = await Client.Post(Endpoints.CONNECTIONS, $"{User.Id}/{weightId}", new {});
        if (!res.IsSuccess) throw new AlertException("API Error", res.Message);

        Connection? conn = res.GetContent<Connection>();
        if (conn is null) throw new AlertException("Unable to get connection", "Connection received from API is null");

        // Add new connection and reset input
        Connections = new(Connections)
        {
            new ConnectionViewModel(conn, true)
        };
        WeightIdInput = string.Empty;
    }
    catch (AlertException ex)
    {
        await Alert(ex.Title, ex.Message);
    }
}
```

Code-snippet af ConnectionsViewModel.OnNewConnection

Jeg har valgt at pakke hele metoden ind i en try-catch, så det er nemmere for mig at give brugeren en alert, når der er noget galt. Her bruger jeg bl.a. min egen AlertException, som ses på billedet nedenunder.

```
7 references
internal class AlertException : Exception
{
    2 references
    public string Title { get; set; }

    5 references
    public AlertException(string title, string message) : base(message)
    {
        Title = title;
    }
}
```


PRODUKTRAPPORT

For at køre koden fejlfrit, skal der være en bruger logget ind. Uden bruger kan der ikke læses forbindelser. Det næste skridt er at se, om brugeren har indtastet et validt id. Til sidst tjekkes der, om vægten med det indtastede id allerede findes i listen.

Når alt er validt, skal jeg have oprettet forbindelsen i backenden. Her bruger jeg bl.a. min ApiClient Client, nævnt tidligere under mit library, for at sende requests til mit API. Hvis noget går galt, enten i mit kald eller parsing af respons værdi, thrower jeg mit AlertException.

Når jeg endelig har fået min nye connection, erstatter jeg den nuværende Connections property (der er databinded til appens frontend) med en ny værdi, der indeholder de tidligere items samt den nye connection. Jeg erstatter Connection property værdien i stedet for at bruge Connections.Add, for at trigger OnPropertyChanged, som opdaterer min frontend.

Til sidst rydder jeg inputfeltet, så der kan tilføjes en ny vægt.

Embedded Hardware

Min embedded hardware er selve vægten i projektet. Det er den, der sørger for at vægtmålinger bliver sendt til API'et. Her er et screenshot af main loopet, hvor der måles vægt og sendes til API'et. Jeg bruger HX711 biblioteket til at lave en ny HX711 scale.

```
void loop() {  
    // Waiting for scale to be ready and for User to initiate default weight  
    if (!shouldInitialize) return;  
  
    scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);  
    scale.set_scale(CALIBRATION_FACTOR);  
    scale.tare();  
  
    printToDisplay("Place weight");  
    delay(WaitTimeMS);  
    printToDisplay("Measuring");  
  
    // User is using weight  
    float units = scale.get_units(10);  
    double value = scale.get_value(10);  
    long read = scale.read();  
    long read_average = scale.read_average(10);  
  
    Serial.println(  
        "Units: " + String(units)  
        + "\nValue: " + String(value)  
        + "\nRead: " + String(read)  
        + "\nAverage: " + String(read_average)  
    );  
    printToDisplay(String(units));  
  
    if (WiFi.status() == WL_CONNECTED) PostWeight(units);  
    else Serial.println("Error in WiFi connection. Status is " + String(WiFi.status()));  
  
    reset();  
}
```

Code-snippet af main loopet i ESP'en.

PRODUKTRAPPORT

Loopet venter konstant på at `shouldInitialize` bliver sat til `true`. Det bliver den, når der kommer et interrupt fra den fysiske knap. Efter interruptet starter vægtmodulet og gør sig klar til brug. Det fysiske display skriver det, der er skrevet i `printToDisplay` argumentet.

Efter 5 sekunder (`WaitTimeMS`, der er defineret længere oppe i filen) måler den vægten. Tallet bliver printet ud på displayet og sendt til API'et, hvis der er wifi forbindelse.

Efter requesten er blevet sendt, sætter den `shouldInitialize` tilbage til `false`, så den er klar til senere brug.

Brugervejledning

Problemformulering

Noget af det sværeste ved at tabe sig, er at holde sig motiveret til at fortsætte, og at se at det man gør, faktisk gør en forskel for ens krop og helbred. Selvom projektet ikke kan vise dine procenter og fortælle dig, hvordan din levetid har forbedret sig, kan den give dig en oversigt over din vægt, uden at du selv skal holde styr på det...

Så, hvordan kan det gøres nemmere at holde styr på ens vægt, uden at bruge for meget tid på, at skrive det ned i en bog, og hvordan kan produktet hjælpe med motivationsboost?

Anvendelse

Login

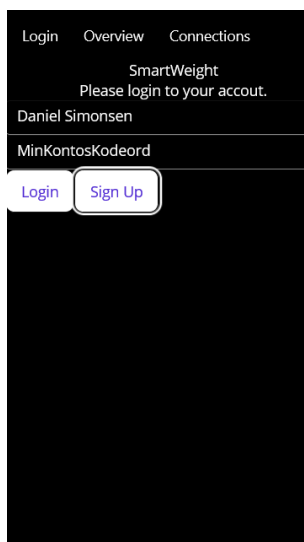
For at bruge produktet skal du først have en bruger i systemet. Uden bruger er der ikke nogen steder at gemme dine vægtoplysninger...

Start med at åbne appen og besøg Login siden. Har du ikke åbnet appen før, er det den første side. Indtast nu dit ønskede brugernavn og kodeord i felterne.

Har du allerede en bruger, kan du selvfølgelig bare logge ind via **Login** knappen.

Har du ikke en bruger, skal du trykke på **Sign Up** knappen.

Efter nogle sekunder bliver du viderestillet til **Connections** siden, som er dér, du opretter forbindelse til din vægt.

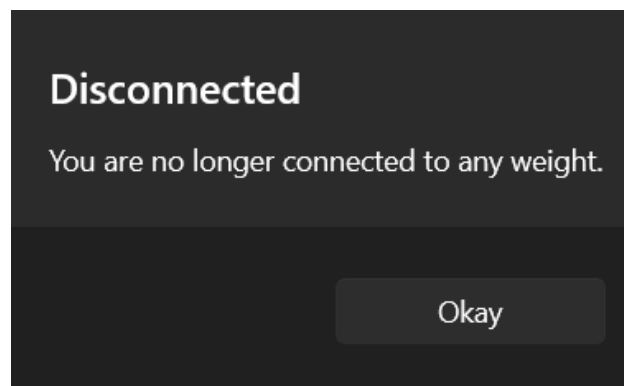
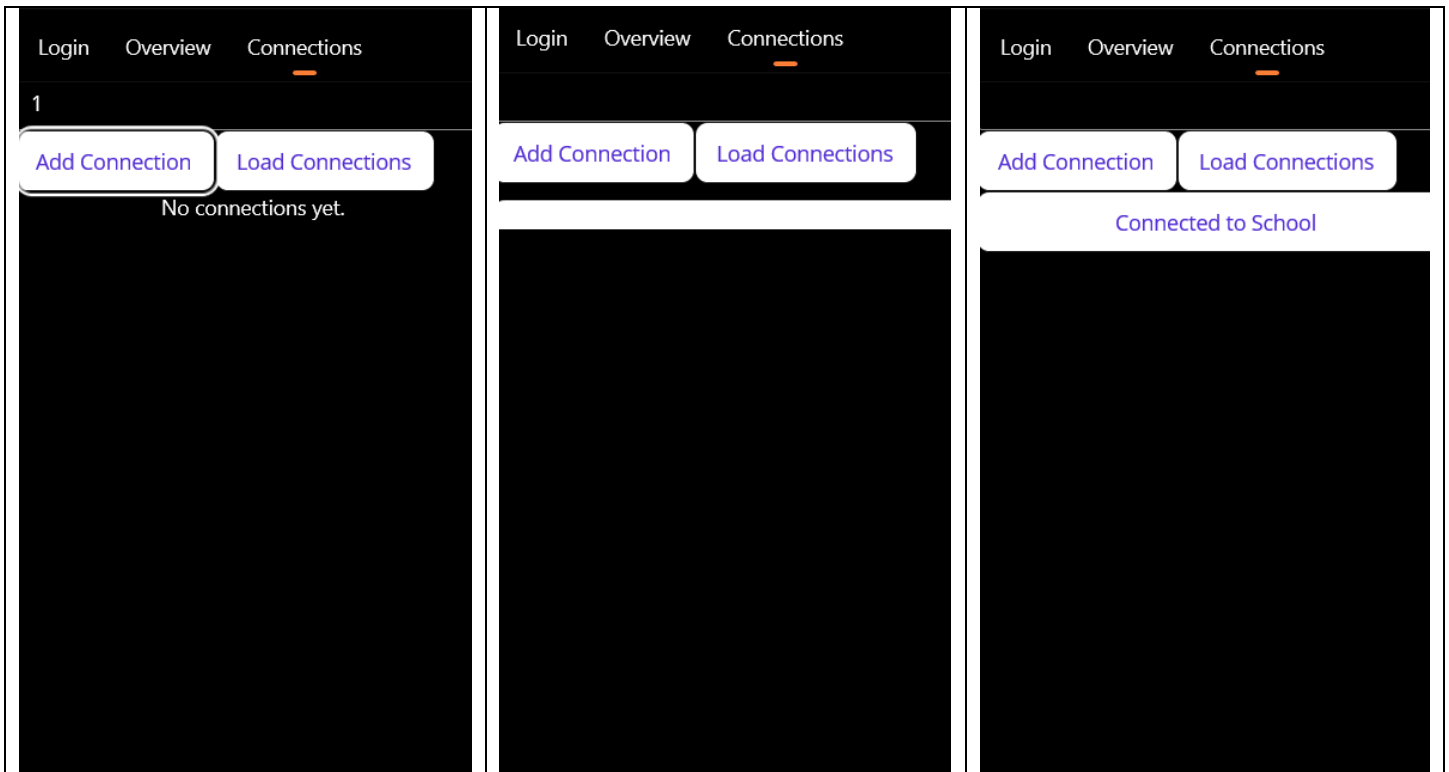


Her er et screenshot af, hvordan det skal se ud, når du opretter din konto.

Connections

Efter login skal du oprette forbindelse til din vægt. I det tomme felt over knapperne, skal du skrive id'et på din vægt. Derefter tryk på **Add Connection**.

Efter tryk skulle din skærm gerne ligne billede 2. Dette er en fejl i MAUI (Frameworket, som appen er udviklet i), og skal bare genindlæses igen. Tryk derfor på en anden side (evt. login) for at få siden til at ligne billede 3. Vælger du at afbryde forbindelsen til vægten, du er forbundet til, skal du bare trykke på **Connected to ...** knappen. Den vil derefter give dig en alert, om at du nu ikke er forbundet.



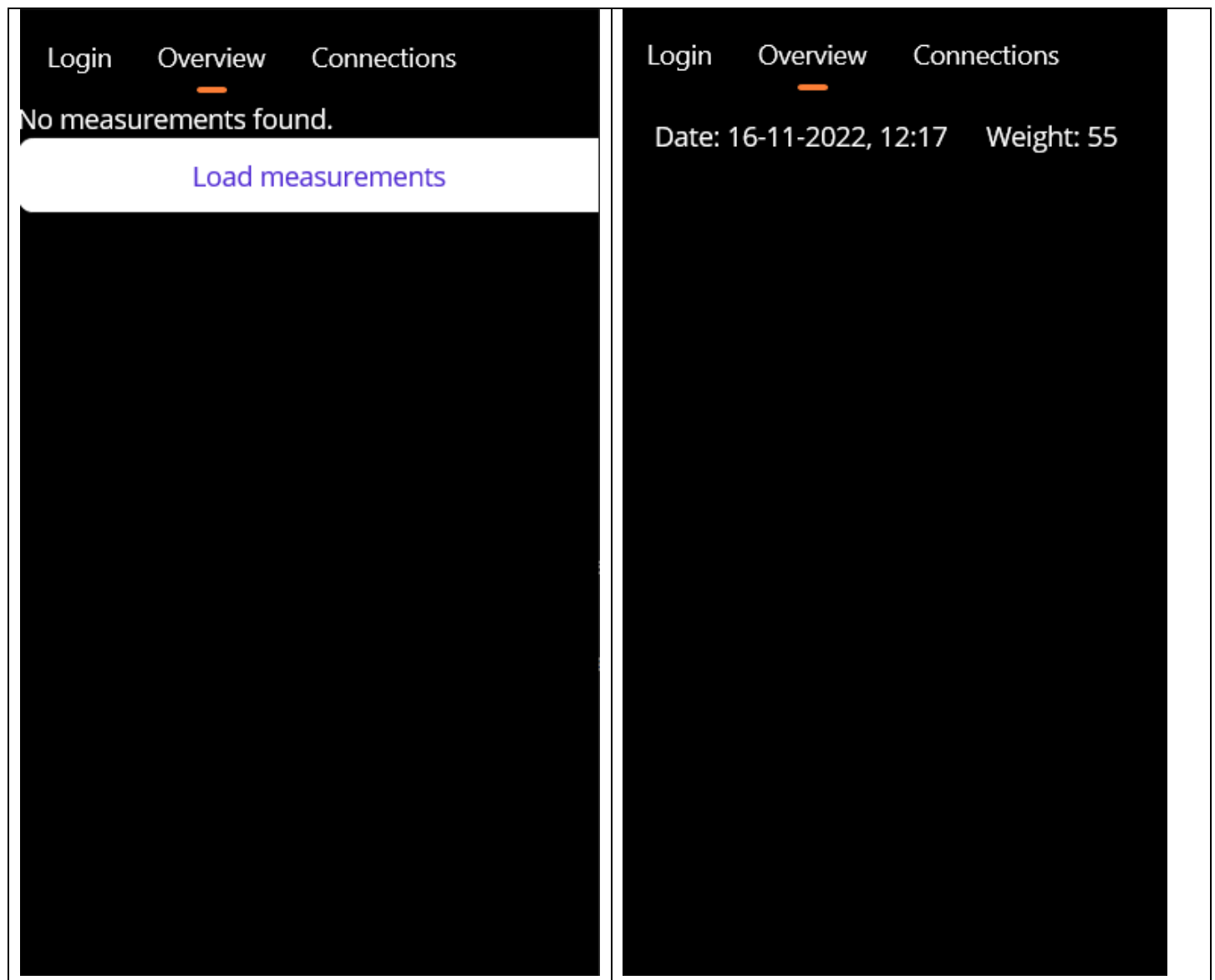
Vægtmåling

Tilslut den embeddede vægt til noget strøm, og vent på at den er forbundet til internettet. Dette kan læses på dens display. Når den er forbundet, skal du trykke på knappen til højre fra displayet. Dette er fordi, vægten skal måle, hvad standard vægten er.

Efter tryk slukker den røde led og tænder den grønne. Dette indikerer at vægten er nu klar til brug. Sæt noget på vægten og tryk på knappen igen. Den embeddede enhed måler nu vægten på objektet, og sender oplysningerne til systemet. Du kan nu besøge **Overview** siden på din app.

Overview

Overview (eller oversigtsiden) er siden, hvor alle dine vægtmål kan læses. Når du først besøger siden, ligner den billedet nedenunder. Giv den nogle sekunder til at indlæse vægtmålingerne fra projektets API eller tryk selv på **Load measurements**.



Teknisk produktdokumentation

ASP.NET API

Jeg bruger ASP.NET's API projekttype. Her har jeg bl.a. adgang til Swagger, som kan hjælpe med tests af endpoints og også give en god visuel præsentation af mine controllers og deres endpoints.

Connections

Forbindelser/Relationer mellem bruger og vægt. Dette bliver brugt, når vægten sender en værdi op til API'et. For at tilknytte en bruger til vægtmålingen, skal der oprettes forbindelse til vægten. Hver gang der bliver oprettet forbindelse til en vægt, tjekker den om der er en vægtmåling, der ikke har fået en brugerreference. Hvis der er, bliver brugerreferencen i connectionen tilføjet til vægtmålingen.

| Connections | |
|-------------|--------------------------------------|
| POST | /api/connections/{userId}/{weightId} |
| GET | /api/connections/{userId}/all |
| GET | /api/connections/{userId} |
| DELETE | /api/connections/{userId} |

POST /{weightId}: Brugeren kan tilføje en forbindelse mellem sig selv og vægt.

GET /all: Få alle brugerens tidligere forbindelser, så der let kan genoprettes forbindelse til eksisterende vægt.

GET: Få brugerens aktive forbindelse

DELETE: Afbryd eller slet brugerens nuværende forbindelse. Fuld slet kræver query delete=true

Measurements

Vægtmåling fra vægten med relation til brugeren, som ejer vægtmålingen.

PRODUKTRAPPORT

| Measurements | |
|--------------|-------------------------------------|
| POST | /api/measurements/collection |
| GET | /api/measurements/overview/{userId} |
| GET | /api/measurements/all |
| DELETE | /api/measurements/all |
| POST | /api/measurements |
| GET | /api/measurements |
| GET | /api/measurements/{id} |
| PUT | /api/measurements/{id} |
| DELETE | /api/measurements/{id} |

POST /collection: Send en liste af målinger, hvis den embeddede enhed har mistet internetforbindelse, og har gemt flere vægtmålinger, der skal sendes.

GET /overview/{userId}: Alle målinger relateret til brugeren

GET /all: Alle målinger fra alle brugere. Kan filtreres via filter=0,1,2 efter internt enum. Endpointet er kun til testing.

DELETE /all: Slet alle målinger. Ligesom dens GET, kan målingerne filtreres og er kun ment til tests.

POST: Tilføj fuld vægtmåling til databasen. Bruges kun internt i API'et.

GET: Få alle vægtmålinger gemt. Her er der ingen filtrering.

GET /{id}: Få bestemt vægtmåling efter id.

PUT /{id}: Opdatér en vægtmåling efter id.

DELETE /{id}: Slet en vægtmåling efter id.

Partial Measurements

En *Partial Measurement* er en vægtmåling uden brugerrelation. En vægt kan kun vide, hvem den selv er, og hvilket tal den har målt. Derfor sender den en partial measurement til API'et, så API'et selv kan lave den om til en fuld Measurement.

| PartialMeasurements | |
|---------------------|---|
| DELETE | /api/measurements/partials/{aggregated} |
| POST | /api/measurements/partials |
| GET | /api/measurements/partials |
| GET | /api/measurements/partials/{id} |
| PUT | /api/measurements/partials/{id} |
| DELETE | /api/measurements/partials/{id} |

DELETE /{aggregated}: Slet alle partial measurements via aggregated string, der er sat sammen med alle id-er, separeret med ",".

POST: Den embeddede vægt, der sender sine oplysninger f.eks. WeightId, Value & Timestamp.

PRODUKTRAPPORT

GET: Få alle partial measurements. Mest brugt til testing.

GET /{id}: Standard få en partial measurements via id.

PUT /{id}: Standard opdatér en partial measurement via id.

DELETE /{id}: Standard slet en partial measurement via id.

Users

Brugere i systemet. En bruger kan bruge vægten, hvor vægten enten kan være privat eller offentlig. For at sikre at en bruger får de rigtige tal gemt til dem selv, skal de forbinde til vægten.

| Users | |
|--------|---------------------------|
| POST | /api/users/login |
| DELETE | /api/users/login/{userId} |
| POST | /api/users |
| GET | /api/users |
| GET | /api/users/{id} |
| PUT | /api/users/{id} |
| DELETE | /api/users/{id} |

POST /login: Få en bruger logget ind på appen.

DELETE /login/{userId}: Få en bruger logget ud af appen. Her afbrydes alle vægt forbindelser, der måtte være aktive.

POST: Opret bruger i systemet.

GET: Få alle brugere i systemet.

GET /{id}: Få bestemt bruger efter id.

PUT /{id}: Opdatér bestemt bruger efter id.

DELETE /{id}: Slet bruger fra systemet.

Weight

Objektreferencen til den embeddede vægt. Den har et id, der kan bruges til at spore tilbage til, hvilken vægt der tog en bestemt måling. Vægten kan også gøres mere personlig via eget givet navn.

| Weight | |
|--------|-------------------|
| POST | /api/weights |
| GET | /api/weights |
| GET | /api/weights/{id} |
| PUT | /api/weights/{id} |
| DELETE | /api/weights/{id} |

POST: Opret ny vægt i systemet

GET: Få alle vægte i systemet.

PRODUKTRAPPORT

GET /{id}: Få bestemt vægt efter id.

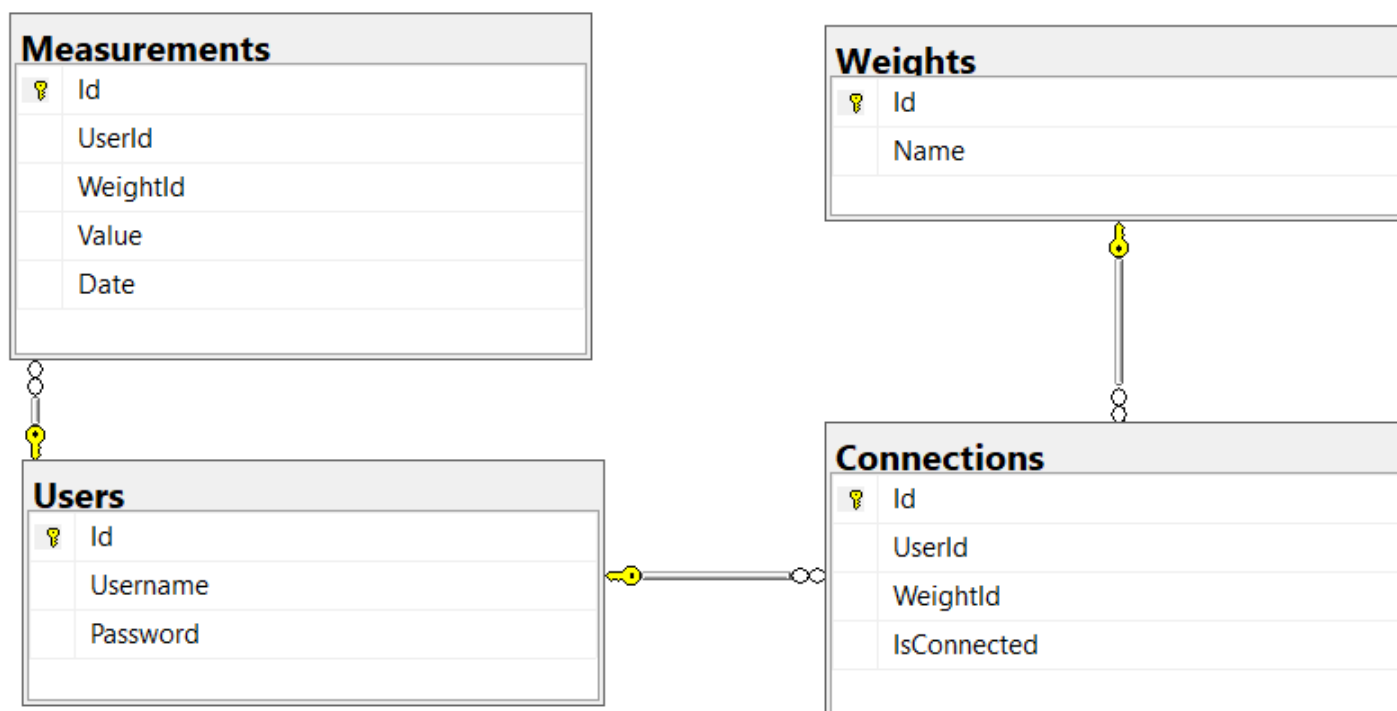
PUT /{id}: Opdatér bestemt bruger efter id.

DELETE /{id}: Slet vægt fra systemet.

SQL Database

Til database valg valgte jeg at gå med en klassisk Microsoft SQL Database. EntityFramework var en stor hjælp til databasehåndteringen. Da jeg bruger code-first princippet, skulle jeg bare lave mine modeller og lave en smule opsætningskode for at få databasen til at spille sammen med mine modeller og hermed mit API og app.

I min database har jeg 4 tabeller...



Her er en oversigt over mine tabeller i SQL.

Users

Min user tabel gemmer på typiske brugerinformationer. Username og Password til login brug, og selvfølgelig et Id til at holde styr på mit data.

Weights

Min weight tabel gemmer på let informationer om min embeddede vægte. Det eneste den gemmer, uderover id, er et valgfrit navn.

Measurements

Min measurement tabel gemmer på både Measurements og Partial Measurements. Da den eneste forskel på de to modeller er at en Partial Measurement ikke har et bruger id, gav det mening at gemme dem i samme tabel. Når en partial measurement får et bruger id senere, opdateres dataen bare i databasen i stedet for at slette fra den ene tabel og tilføje til den anden.

UserId og WeightId er begge foreign keys til User og Weight tabellerne. Value er vægtmålingens værdi, og Date er tidspunktet målingen blev taget på.

Connections

Min connection tabel gemmer på nuværende forbindelse imellem bruger og vægt. Når en bruger tilknytter sig vægten via appen, bliver der gemt en forbindelse mellem brugeren og vægten der blev skannet. Forbindelserne slettes sjældent, og bruges mest til at tjekke dens IsConnected property.

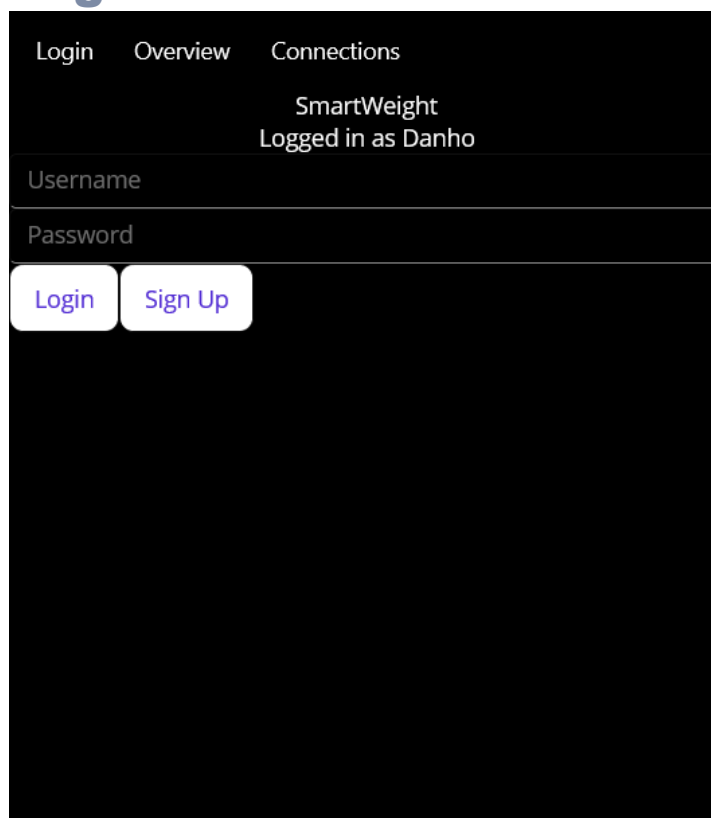
MAUI Cross-platform App

Som app udviklingsmiljø har jeg valgt at prøve Microsofts nye MAUI, som bygger videre på deres Xamarin.

Appen skulle være meget simpel. Du logger ind på appen via din bruger i systemet. Derefter kan du enten se din egen oversigt, altså se alle dine vægtmålingsposter, der tidligere er gemt, eller oprette forbindelse til den vægt du skal til at bruge.

Appen består derfor af 3 sider...

Login



Login siden er *meget* simpel. Du indtaster brugernavn og kodeord, og vælger enten login eller sign up. Findes loginet ikke, spørges der om du vil lave en ny konto i systemet. Har du været logget ind før, gemmes dine oplysninger i appens eget lager, og du kan derfor let ignorere login siden.

Overview

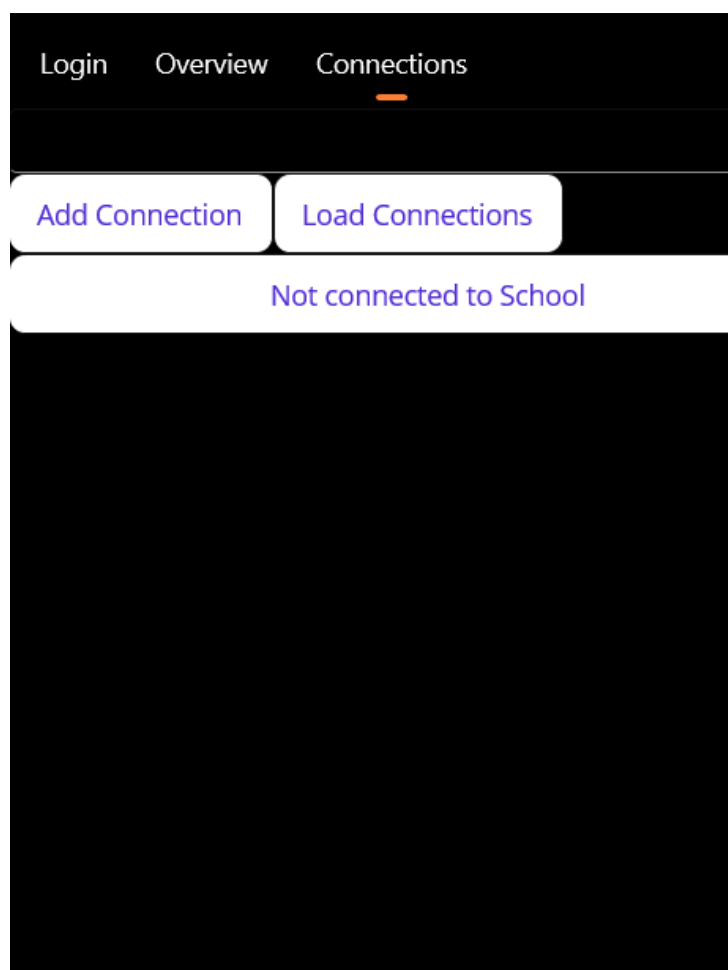


Overview giver dig en oversigt over dine tidligere vægtmålinger. Efter nogle sekunder bliver alle dine vægtmål sat på skærmen, så du let kan se, hvordan tallene har udviklet sig.

Det første billede illustrerer at der ikke er nogle målinger læst endnu. Enten fordi den skal hente dataen eller fordi, der ikke er indsat data til brugeren endnu.

Det andet billede illustrerer at der er hentet data.

Connections



Connections er lidt ligesom en Bluetooth side i din telefonindstillinger. Alle dine tidligere forbindelser bliver sat på siden, så du let kan til- eller frakoble dine forbindelser til vægtene, du tidligere har været forbundet til.

Billedet viser hvordan siden ser ud. Over alle knapperne er der et tekstinput, hvor brugeren kan indtaste id-et på vægten, brugeren gerne vil forbindes til, hvis vægten ikke allerede er i oversigten.

"School" er én vægt der allerede er på brugeren her, der er logget ind. Brugeren kan let trykke på "Not connected to School" knappen for at oprette forbindelse.

ESP8266 Embedded Enhed

ESP enheden bruger jeg til vægtmåling. ESPen er selve vægten, og er den, der sender data op til mit API. Desværre fungerer vægtkomponentet en smule anderledes end en normal vægt.

Før vægtkomponentet kan måle en vægt, skal den først måle standardvægten – altså hvor meget intet vægt vejer. For at kunne kommunikere med vægten, bruger jeg HX711 biblioteket.

Udover vægten og selve ESPen, bruger jeg også et display, to led lys og en knap.

Mit display viser mine tilstande på tekst og viser den endelige vægt.

De 2 led lys indikerer om vægten er klar til brug. Ved rødt lys skal den initialiseres ved tryk på knap.

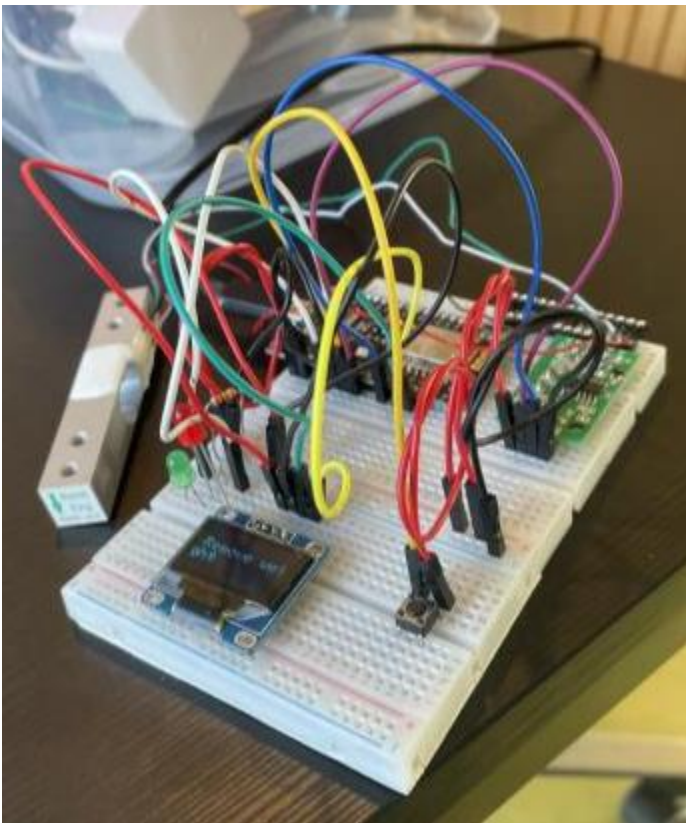
Ved grønt lys er den klar til brug, og begynder først måling ved tryk på knap.

Knappen er styrer derfor den interne tilstand.

For at kommunikere med projektets API bruger jeg ES8266HTTPClient, ESP8266WiFi & WifiClient.

Til timer interrupt og tidspunkt på måling bruger jeg Ticker & chrono.

Til at bruge mit display bruger jeg AdaFruit_GFX, Adafruit_SSD1366 & Wire.



Her er et billede af min fysiske løsning. Rød er power, sort er ground, gul og hvid er output til LED og display, grøn er output til display, blå og lilla er clock/data til HX komponentet til vægten.