Processrapport

SmartWeight

Daniel Simonsen
21-11-2022

Titelblad

Techcollege,
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

Fremlæggelsesdato:
24/11/2022

Læsevejledning

Produktrapporten anbefales at læses først, da det er nemmest at forstå processen, hvis man kender produktet på forhånd.

Forord

Undervejs kommer der ord/navne, som kan lyde ubekendte…

API: Application Programmable Interface, en server/"tjener" til produktet. Et API forstås som en tjener, hvis man sidder i restaurant. Du giver tjeneren en ordre, tjeneren går til køkkenet og beder om ordren, og får et svar tilbage. Projektets API fungerer på samme måde, hvor man f.eks. på appen beder om data, som API'en henter fra databasen.
Endpoints: Et API har flere endpoints i en controller. En controller ville være en afdeling, der f.eks. styrer brugerne eller vægtmålingerne i systemet. Ved at spørge det rigtige sted, kan du få det data, du leder efter.

EntityFramework & DbContext: EntityFramework er et framework, der gør det nemmere at snakke sammen med databasen. Til kommunikation skal EntityFramework bruge en DbContext, som er hovedobjektet, der snakker med databasen.

.NET MAUI & React-Native: Begge frameworks er brugt til app-udvikling. .NET MAUI er Microsofts nye cross-platform miljø, hvor React-Native er Facebooks cross-platform miljø. MAUI er skrevet i C# og React-Native i TypeScript eller JavaScript.

JavaScript, C#, C, Python: Forskellige programmeringssprog.

Indholdsfortegnelse

Case beskrivelse

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 dit helbred 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?

Afgrænsning

Da jeg ikke har en vægt, der kan tage imod vægten af et menneske, bruger jeg små objekter som stadig giver et vægttal.
Jeg ville også gerne bruge en Raspberry Pi til vægtmåling, da Raspberry Pi brug er ét a kravene i Linux/Embedded faget. Desværre har jeg ikke tid til at sætte mig ind i Raspberry Pi og få den til at spille i mit projekt.

Projektplanlægning

Tidsplan

Min tidsplan er både estimeret og realiseret. Grå dage er weekender og sorte dage er ikke afsat til tværfagligt projekt ifølge lærerne.

Se **Error! Reference source not found.**.

Arbejdsfordeling

Som klassisk programmør bruger man mere tid på programmering end rapportskrivning. Alt hvad der foregik af dokumentation, både tidsplan, proces- og produktrapport, blev pænt lagt til side, så der var mere programmering. Det er nemmere at skrive om produkterne før og efter påbegyndelse, og så løbende opdatere tidsplanen.
Tidsplanen i sig selv var svær at arbejde med. Når du laver en proces for et produkt, vil du gerne starte i dele og gør de dele færdige, før du stater på nye dele, end at have flere kørende på samme tid. Desværre har vi fag kørende i løbet af ugen med forskellige lærere, og man vil selvfølgelig gerne arbejde på sin app, når der står App III på skemaet, så man får det bedste ud af de ressourcer man har til rådighed…
Selve processen gik fint. Jeg ramte selvfølgelig ikke plet ved mange af mine punkter – nogle under og nogle over. Jeg havde ikke forventet at bruge super lang tid på selve appopsætningen, eller have store problemer med kamera og QR-kode, men det fik jeg.
Da .NET MAUI har problemer med deres kamera/billede funktion internt, kunne jeg ikke bruge min originale ide med kamera til at skanne en QR-kode for at oprette forbindelse. Da der ikke er andet der giver mening i mit projekt, har jeg ikke noget telefon hardware inkluderet i mit projekt.

Metode- og teknologivalg

App

Der er mange forskellige teknologier til at lave apps. Ved app-design vælger man som regel en teknologi, der understøtter cross-platform. Xamarin & den nye .NET MAUI er nogle gode bud på C# appudvikling, og evt. Flutter & React Native som klassiske bud på JavaScript-like syntaks.

Af de 4 nævne frameworks har jeg allerede prøvet at udvikle i både MAUI og React Native. Jeg har selv valgt at arbejde med den nye MAUI frem for React Native eller at lære de andre frameworks at kende. Jeg føler, at det er nemmere at sætte et MAUI projekt op, og få appen emuleret frem for et React Native projekt. Jeg løber en lille risiko ved at bruge det nye MAUI, da der er mangel på hjælpemidler, og meget af frameworket stadig kan være fyldt med fejl eller ikke udviklet færdigt.

API

I forhold til API, er der primært C#'s ASP.NET Web API, JavaScripts Node ved brug af express.js, Pythons Django og Cs NodeMCU via ESP8266WebServer. Eftersom jeg har arbejdet i alle 3, undtagen Django, har det været lidt svært at beslutte, hvilken teknologi der giver mest mening. Både for projektet, men også for struktur og opsætning – hvad kræver mest, og hvor giver det mening at bruge de forskellige teknologier?

Jeg har valgt at gå med C#'s ASP.NET Web API. Jeg har nærmest ingen erfaring med ESP8266WebServer eller Django, så de kunne jeg nemt krydse af listen. Selvom NodeMCU er relativ simpel, mangler jeg en bedre struktur for mit projekt. Derfor ville det næste blive JavaScripts Node. Jeg har meget erfaring med JavaScript, og har enligt ikke noget imod opsætningen, syntaks eller struktur. Dog har jeg valgt C#'s ASP.NET Web API over JavaScripts Node, fordi meget af det kode der er i Node er allerede i ASP.NET Web API. Derudover er det ikke lang tid siden, jeg sidst arbejdede med en ASP.NET Web API ift. en Node API.

Database

Databaser er for det meste delt op i relationel/non-relationel. I princippet kunne jeg have brugt begge dele:

- Relationel: Fordi mine modeller følger en bestemt struktur, som ikke har nogle valgfrie properties udover mine vægtmålinger. Brugerne har relation til deres målinger i form af 1..*. En bruger kan have mange målinger, men 1 måling har kun 1 bruger.
- Non-relationel: Fordi mine målinger gemmes i masse. I en non-relationel database, ville det betyde at jeg kan gruppere mit data via bruger id, og få json objekter tilbage, fremfor at gemme *alle* målinger i samme tabel.

Embedded

Af embeddede enheder kender jeg til ESP8266, ATMega168 & Raspberry Pi. Min embeddede enhed skal kunne læse vægten på et objekt og sende dataen til mit API. Derfor skal den have et komponent til at læse vægt, og kunne være i stand til at sende dataen videre via internetforbindelse.

Til projektet bruger jeg en ESP2866, fordi den har minimumskravene på produktet. Har jeg tid til det, vil jeg også bruge en Raspberry Pi, da vi skal bruge en Pi i Linux/Embedded faget.

Væsentlige elementer fra produktrapporten

I processen under API, nævner jeg mine endpoints som er bedst forstået i produktrapporten.

Proces

Mindmap/Flowchart

For at få de 4 store dele sat sammen, har jeg brug for at lave et mindmap/flowchart til mig selv, så jeg har ordentligt overblik over mit projekt. Her kan jeg dykke ned i hvordan jeg vil sætte mit projekt op og få det hele til at spille sammen ordentligt og optimalt. Det er også her, jeg finder ud af hvilke modeller/klasser, der skal bruges til projektet – både backend og frontend.
Se mit Original Flowchart

API

Som API teknologi besluttede jeg mig for at bruge ASP.NET Core. Jeg har tidligere lavet 3 andre API'er i ASP.NET Core, så derfor var det relativt nemt at gå til. API'et er det vigtigste led i et fullstack projekt, for uden API'et, er der ingen kommunikation mellem frontend, database og embeddede maskiner. Derfor synes jeg at det gav mening for mig at starte der.
Efter at have lavet modellerne i et library, tak til min flowchart proces, havde jeg også en god idé om, hvilke former for controllers, der skulle tilføjes til API'et.
Da jeg var færdig med de basale endpoints, kom jeg til redis-cache delen. Jeg skulle have en cache, der gemmer på den nuværende bruger, der bruger en bestemt vægt, og også de midlertidige målinger fra vægte, der ikke har en bruger tilknyttet. Jeg konkluderede så, at det var for meget arbejde ift. hvad det hjalp. Derfor bliver alt nu gemt i min SQL-database – både forbindelser mellem bruger og vægt og også vægt målinger, der ikke har en bruger tilknyttet. Dog hvis en vægt måling har været gemt i systemet i 10 minutter uden bruger, bliver den fortolket som ugyldig måling og automatisk slettet.
Tak til Swagger, som kommer med til test af endpoints i ASP.NET projekter, var det hurtigt og nemt at teste og fiske fejl, der var opstået under konstruktionen af API'et.
Der kan læses om API'ets endpoints i produktrapporten.

Database

Som database bruger jeg Microsoft SQL. Jeg gemmer 4 tabeller, som er vist i bilaget her. Det hele er styret af EntityFramework fra mit library projekt. Hver kommunikation mellem backend og database foregår via min DbContext klasse, som er en ORM for mine modeller/klasser.
2 af mine tabeller, **Users** og **Weights**, er generel data omkring brugerne og vægtene. En bruger har flere målinger på evt. forskellige vægte, hvilke er der min **Measurements** tabel kommer i spil. Dog ved vægten ikke hvilken bruger, der bruger vægten, når der måles vægt. Uden bruger relation til min måling, er målingen ikke fuldendt. Derfor har jeg brug for en **Connections** tabel, som er et mellemled mellem bruger og vægt.

Når der er en valid forbindelse mellem bruger og vægt, kæder API'et brugeren sammen med vægten, der er kommet ind fra ESP2866'eren.
Se

Database tabellerne her

Realiseret tidsplan

Tidsplanen for mig har været svær at komme igennem. Da jeg har minimal erfaring med at sætte tidspunkt på mine udviklingsevner, har jeg fuldstændig misset mine skud og oveni lagt tidsplanen på hylden. Nogle få gange har jeg fået den opdateret med mine realiserede tidspunkter.
Se min Tidsplan.

Konklusion

SmartWeight kan nu hjælpe dig med at holde styr på din vægt. I stedet for at skrive dit vægttal ned i en bog, og holde styr på bogen, kan du bare veje dig selv på SmartWeight. Dine tal bliver automatisk gemt og sendt til din app. Når du ser dine tal, kan de hjælpe dig med at få motivationsboost.

Diskussion

Produktet virker *næsten*. Jeg havde heller ikke forventet et færdigt produkt, men jeg havde selvfølgelig forventet mere, end hvad jeg har afleveret. Da meget af udviklingen er nyt for mig, var det ret svært at komme i gang. Jeg havde ikke den største erfaring med EntityFramework & MAUI, og jeg har aldrig haft god forståelse i embedded programmering.

Jeg synes, at jeg kom godt igennem mit projekt, trods meget af det var nyt og udfordrende. Jeg kom helt sikkert i mange klemmer, men det er bare et spørgsmål om erfaring med teknologierne.
Det har været spændende at arbejde med et projekt og dokumentere det, frem for at direkte gå i gang med at kode, og håbe at man har alle sine egne krav med.

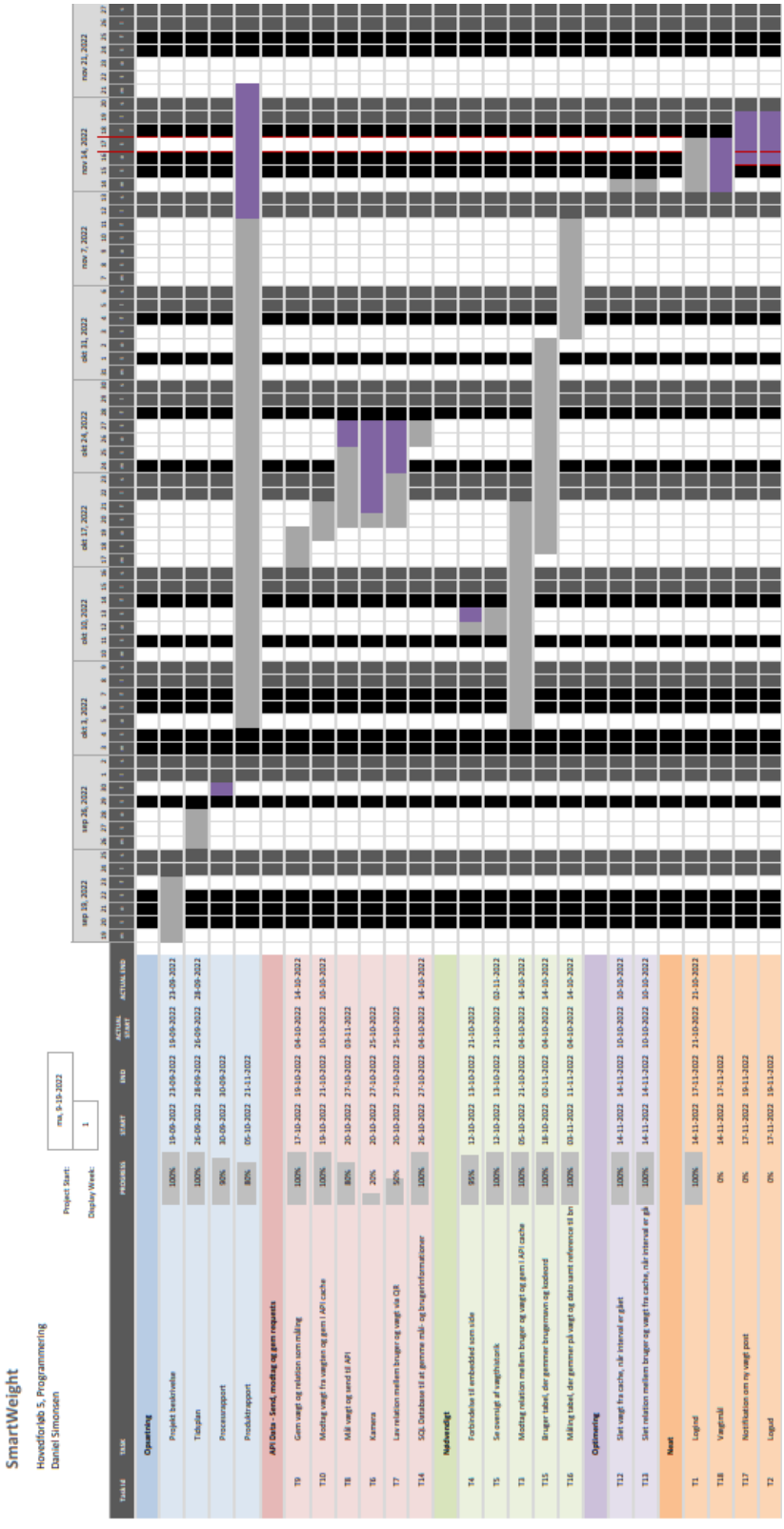
Jeg er dog ked af, at jeg ikke kunne få kameraforbindelse til min app, og generelt er jeg meget skuffet over min app. Desværre er MAUI ikke helt færdigudviklet, hvilket er grunden til, at jeg ikke kunne oprette forbindelse til telefonens kamera. Desuden er min app ikke særlig køn, og halvdelen af den fungerer optimalt. Havde jeg haft mere tid, ville jeg enten arbejde mere på min app og få den til at virke, eller også havde jeg fuldstændig skiftet teknologi og lavet det i react-native, som jeg havde overvejet tidligere.

Havde jeg haft mere tid, ville jeg også have haft inkluderet en graf på Overview siden, i stedet for at vise en tabel med rå tal. Det giver større forståelse på en oversigt med grafer frem for rå tal, men da jeg allerede havde en masse nyt at skulle prøve, kunne jeg ikke få graferne med i produktet.

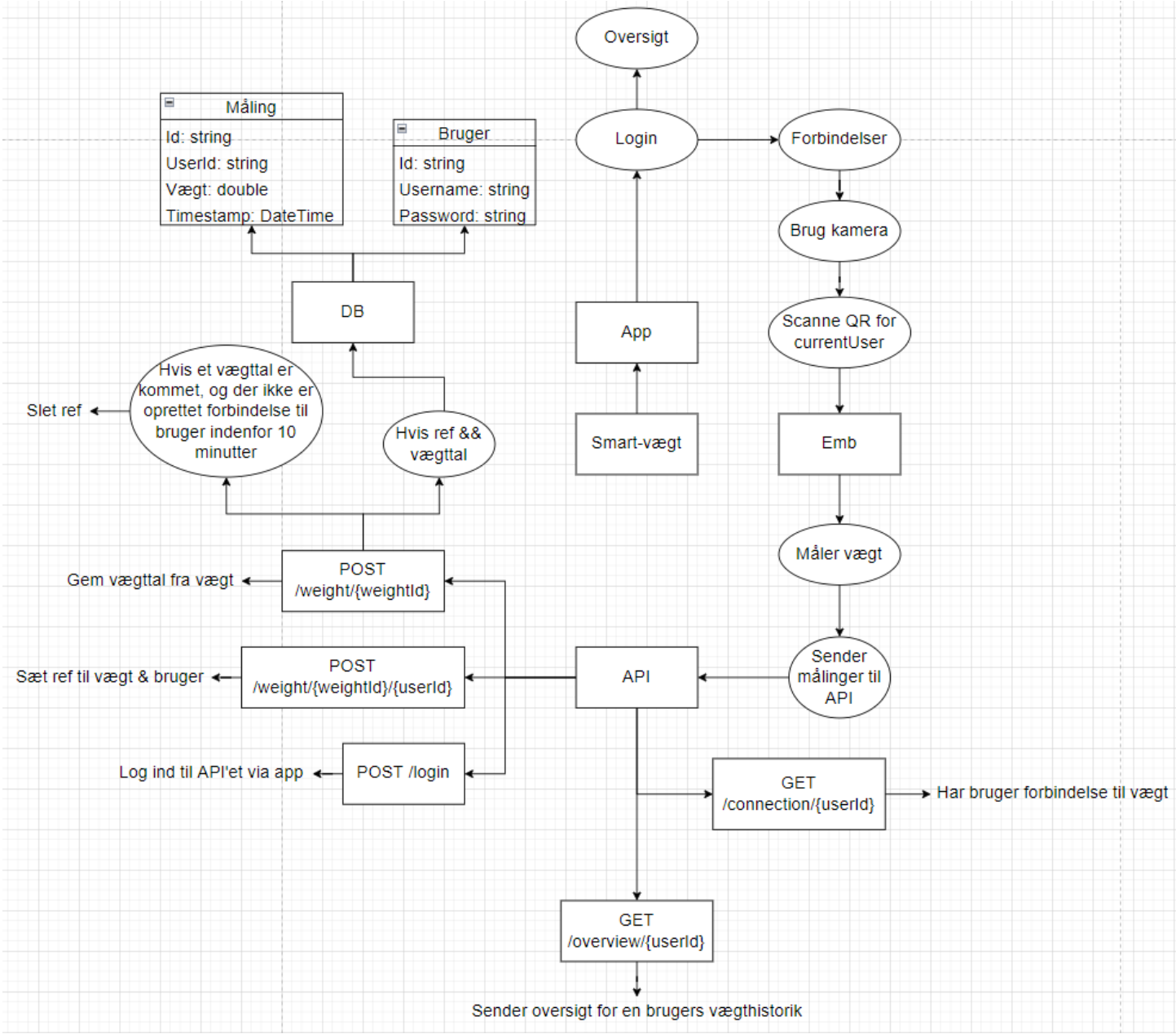
Til næste gang kunne jeg godt forestille mig at bruge ASP.NET til Web API igen. EntityFramework og Swagger gjorde min backend udvikling meget nemmere at debugge og sætte op. Det eneste jeg er bange for, er min håndtering fra backend til frontend, da jeg højst sandsynligt vælger at udvikle en frontend i Typescript. Jeg skulle gerne have mine C# modeller til at give mening i Typescript, og skal nok lave et model library til begge sprog…

Bilag

A. Tidsplan



B. Original Flowchart



C. Database tabeller