Procesrapport

CitizenTaxi

Daniel Simonsen
01-12-2023

Titelblad

Techcollege Aalborg,
Struervej 70,
9220 Aalborg

Elev:
Daniel Simonsen

Firma:
Skoleoplæringscenter

Projekt:
CitizenTaxi

Uddannelse:
Datatekniker m. speciale i programmering

Projektperiode:
06/11/2023 – 07/12/2023

Afleveringsdato:
01/12/2023

Fremlæggelsesdato:
07/12/2023

Vejledere:
Frank Rosbak
Lars Thise Pedersen
Per Madsen

Læsevejledning

Til projektet er der lavet en proces- og en produktrapport. Jeg anbefaler at læse denne rapport først, da det er vigtigt at forstå problemet samt tidsbegrænsning for at få mest ud af produktrapporten.

Indholdsfortegnelse

Indledning

Borgerne i Region Nordjylland klager over den lange ventetid, når de ringer til Flexsygehustaxa. De vælger derfor at ringe til lægesekretærerne, og bede dem om at bestille en taxa for dem.
Lægesekretærerne kan derefter ringe via internt nummer og få taxaen bestilt til borgeren hurtigere, end hvis borgeren selv havde ringet, dog tilføjer det endnu en opgave hos lægesekreterne, som kunne have været undgået, hvis borgeren selv havde bestilt taxaen.

Case beskrivelse

Problemformulering

Hvordan kan man få borgerne til selv at bestille deres Flexsygehustaxa uden hjælp fra lægesekretærerne og uden den unødvendige lange ventetid, og kan man ellers formindske lægesekretærernes ekstra arbejde?

Afgrænsning

For at gennemføre projektet som en prototypeløsning til Flexsygehustaxas lange ventetid afgrænser jeg bl.a. sygehuspersonale adgang (f.eks. bruge deres eksisterende loginoplysninger) og rigtig taxabestilling (hvis de har et offentligt API til rådighed). Produktet har egne brugere og simulation af taxabestilling af denne årsag.

Projektplanlægning

Estimeret tidsplan

Billedet nedenunder viser min estimerede tidsplan i form af Gantt.
Jeg forestiller mig konstant at opdatere min procesrapport, indtil jeg vurderer, den er færdig.

Daniel Simonsen estimeret tidsplan		06-Nov	07-Nov	08-Nov	09-Nov	10-Nov	13-Nov	14-Nov	15-Nov	16-Nov	17-Nov	20-Nov	21-Nov	22-Nov	23-Nov	24-Nov	27-Nov	28-Nov	29-Nov	30-Nov	01-Dec
Opgave	Estimerede datoer	man	tir	ons	tor	fre	man	tir	ons	tor	fre	man	tir	ons	tor	fre	man	tir	ons	tor	fre
Forberedelse	06/11 til 09/11																				
Procesrapport	09/11 løbende til 27/11																				
Produktrapport	09/11 løbende til 27/11																				
Backend	10/11 og 13/11																				
Backend test	13/11 til 16/11																				
Frontend	16/11 til 20/11																				
Host & user test	23/11 til 27/11																				
Kodedokumentation	28/11 til 01/12																				

Realiserede tidsplan

Min realiserede tidsplan viser, hvor meget jeg har undervurderet frontend/hjemmeside-delen. Jeg måtte endda tilføje en lør/søn ind i mit skema, så jeg kunne komme tilbage på sporet igen.

Jeg konkluderede også, da jeg startede på programmeringsdelen, at det var smartest at dokumentere under udviklingsprocessen, imens koden stadig var frisk i hovedet. Dog mistede jeg lidt sporet, da jeg blev en smule stresset med at falde bagud i tidsplanen, hvilket betød, at jeg glemte at dokumentere min kode.

Daniel Simonsen estimeret tidsplan		06-Nov	07-Nov	08-Nov	09-Nov	10-Nov	13-Nov	14-Nov	15-Nov	16-Nov	17-Nov	20-Nov	21-Nov	22-Nov	23-Nov	24-Nov	25-Nov	26-Nov	27-Nov	28-Nov	29-Nov	30-Nov	01-Dec
Opgave	Estimerede datoer	man	tir	ons	tor	fre	man	tir	ons	tor	fre	man	tir	ons	tor	fre	lør	søn	man	tir	ons	tor	fre
Forberedelse	06/11 til 09/11																						
Procesrapport	09/11 løbende til 27/11																						
Produktrapport	09/11 løbende til 27/11																						
Backend	10/11 og 13/11																						
Backend test	13/11 til 16/11																						
Frontend	16/11 til 20/11																						
Host & user test	23/11 til 27/11																						
Kodedokumentation	28/11 til 01/12																						

Arbejdsfordeling

Den første uge var planlagt til ren forberedelse af projektet. Her indgår brainstorm, kravspecifikation, design, opgaver og estimeret tidsplan. Jeg havde derefter forestillet mig, at backend og frontend ville tage minimal tid, da jeg er stærk indenfor begge stakke, og så bruge en masse tid på backend og frontend tests. Til sidst ville jeg have fuld fokus på dokumentationsdelen.

Realiteten blev, at min backend og backend test gik hurtigere end forventet. Hvad jeg så senere fandt ud af var, at jeg havde brug for den ekstra tid, da jeg stødte på mange forskellige udfordringer i min frontend og kom derfor næsten 4 dage over tidsplanens estimering. Efter en hård weekend kunne jeg heldigvis komme ca. 2 dage frem igen, hvilket betød, at jeg havde mindre tid tilbage til rapport og dokumentation. Heldigvis havde jeg i forvejen været god til at løbende dokumentere min kode, så jeg ikke skulle bruge nær så meget tid på kodedokumentation som planlagt.

Metode- og teknologivalg

Brugerflade

For at vise mit projekt skal jeg have en brugerflade. Det er vigtigt, at borgerne og lægesekretærerne har en nem og overskuelig brugerflade til at bestille taxa til borgerne. Herunder er der bl.a. spørgsmål, om jeg skal have udviklet en hjemmeside, telefonapp eller computerapp.

Da de fleste borgere nok har en form for tablet, og lægesekretærerne sidder ved deres computer, ville det være oplagt at lave en hybrid af både mobil- og computerapp. Da jeg ikke har tid til at udvikle begge, må en hjemmeside være mest oplagt at lave, da man kan åbne en hjemmeside på både tablet og computer.

Der er selvfølgelig mange måder at lave hjemmesider på. Man kan både lave en klassisk html/css/JavaScript hjemmeside eller bruge et eksternt framework/library til at gøre udviklingsprocessen nemmere og hurtigere.
Jeg kan vælge at lave en Blazor eller MVC (Model-View-Controller) hjemmeside i programmeringssproget C#, eller vælge at lave en hjemmeside via et JavaScript framework/library som Vue, React eller Angular.

Da hjemmesiden er rimelig stor i forhold til, at der skal være et loginsystem, borgerside med bestillinger samt administrationsside med håndtering af borgere, notater og bestillinger, ville det være en besværlig oplevelse at arbejde med MVC, da MVC ikke tilbyder klientintegrationer uden JavaScript. Det ville nok give mest mening at arbejde i Blazor, da Blazor tilbyder klientintegration i C#, men da jeg aldrig har brugt Blazor før, ville det være for stor en risiko at løbe indenfor min tidsgrænse.

Mine sidste muligheder ville være et JavaScript framework/library. Herunder har jeg aldrig prøvet Angular, så der er kun Vue og React tilbage af sikre muligheder. Vue har vi lært på hovedforløb 4, men jeg har ikke brugt frameworket siden, hvorimod jeg har god erfaring med React, og tænker at det ville være hurtigst at gå til. Ved brug af React, gør jeg det også nemmere for mig selv med min udvikling af step-by-step formular til borgersidens bestillingsproces, samt de mange modals/popups jeg skal bruge til oprettelse, opdatering og sletning af borgere, notater og bestillinger på adminsiden.

Backend

For at borgere kan se deres egen side, og lægesekretærer kan se administrationssiden, skal de have en rolle, der styrer, hvad de har adgang til at se. Derudover kan man ikke direkte gemme data (som logins og brugere) via en hjemmeside, hvilket betyder, at jeg skal have en backend til at håndtere data sendt fra hjemmesiden, og ligeledes sende data tilbage til hjemmesiden igen. Det betyder, at jeg skal have en backend, der kan gemme data et sted og lytte til hjemmesidens anmodninger.

Når man lytter til en hjemmesides anmodninger, tales der oftest om et API (Application Programmable Interface). Mit API skal være bindeled mellem mine gemte data og min hjemmeside for at få data sendt. De fleste populære programmeringssprog understøtter deres egen implementering af et API, såsom Python Django/Flask, JavaScript Express eller C# ASP.NET.
De har alle deres egne styrker, som f.eks. er Pythons frameworks hurtige at sætte op, men man skal bygge næsten alt fra bunden, som f.eks. sikkerhed, hvilket er en stor faktor indenfor API'er.

JavaScript deler lidt den samme historie med Pythons frameworks i, at det er relativt nemt at sætte op, men der mangler noget sikkerhed og en masse eksterne NPM (Node Package Manager) pakker for at sætte alt korrekt op. Det ville fungere fint sammen med min React hjemmeside, da man både kan lave et Express API i JavaScript og TypeScript, som React også understøtter. Det vil sige, at mit udviklingsmiljø ville foregå indenfor JavaScript/TypeScript, hvor jeg formindsker sprogforvirring og ikke mindst kan genbruge egne klasser, funktioner, og programmeringsbiblioteker gennem backend og frontend.
C#'s ASP.NET har en rimelig fast standard. ASP.NET vil have, at alle dine modeller har deres egen controller, som er med til at styre ruteanmodninger. Derudover har ASP.NET allerede indbygget sikkerhed og giver mig et nemt og overskueligt udviklingsmiljø.

Da jeg er stor fan af controllermønsteret og ikke har stor erfaring indenfor sikkerhed, tænker jeg, at ASP.NET ville være en stor hjælp, selvom jeg sagtens kunne bruge Express og de 2 Python frameworks til at lave mit eget controllersystem. Jeg har god erfaring med C# API'er i ASP.NET og tænker, at jeg ville være mest effektiv med den platform, men ville ikke have noget imod at arbejde i JavaScript/TypeScript. Det kommer dog også an på hvilke ORM'e, jeg har til rådighed.

ORM

Da mit API også skal sørge for at gemme data et sted, ville det være smartest at kikke på en ORM (Object Relation Mapping). En ORM's formål er at forbinde til en database og hente samt konvertere (mappe) det gemte data til prædefinerede modeller/klasser. På den måde slipper jeg for at have SQL-sætninger i min kode og dermed undgår alle former for SQL-injections, som kan være med til at gøre min backend usikker. Ved brug af en ORM, fortæller jeg ORM'en, hvilke modeller der hører til hvilke tabeller, således at ORM'en er i fuld kontrol over forbindelsen mellem backend og database.

Jeg har i tidligere projekter prøvet at gemme relationel data i en Express API men fik aldrig noget funktionelt. Til gengæld har jeg en smule erfaring med at gemme relationel data med EntityFrameworkCore i C#, og selvom jeg ikke er rutineret i EntityFramework, tænker jeg, at det er en mindre risiko at løbe, end hvis jeg kæmper med at prøve at gemme med en JavaScript ORM.

Derfor vælger jeg at lave min backend i ASP.NET med en EntityFrameworkCore ORM, der kan gemme mit data i en MSSQL relationel database.

Database

Jeg skal selvfølgelig gemme mine logins, borgere, administratorer, og til demoformål også gemme egne borgernotater og taxabestillinger. Da EntityFrameworkCore allerede kommer med en MSSQL (Microsoft Structured Query Language) relationel database, kan jeg lige så godt benytte mig af den. Det har ikke stor betydning, om det er en MSSQL eller en MySQL database – så længe databasen kan gemme på mine relationer imellem mine modeller.

Vælger jeg en nonrelationel database som MongoDB eller PouchDB, ville jeg selv skulle holde styr på mine relationer og hente dem selv, hver gang jeg har brug for dem. Det kan EntityFrameworkCore allerede gøre for mig, da en MSSQL database er relationel, kan EntityFrameworkCore selv finde ud af at hente det data, jeg har brug for via Explicit loading (jeg fortæller min orm, hvilke data jeg vil have, og så henter den det for mig).

IDE

For at kunne lave programmer skal jeg bruge en IDE (Integrated Development Environment), der kan kompilere min kode til "computersprog". De fleste IDE'er har også gerne flere features indbygget som f.eks. konsol, filsystem, debugging og kodehjælp som syntaks og autocomplete. Når jeg kikker efter en god IDE, skal den hjælpe mig med at finde fejl samt gøre det letlæseligt for mig og evt. andre udviklere, der kunne kikke på min kode.

Her er jeg dog splittet af 2 IDE'er: Visual Studio og Visual Studio Code. Jeg har god erfaring med begge IDE'er men bruger dem til forskellige formål. Som regel bruger jeg Visual Studio til backend programmering, da det oftest er i C#, jeg udvikler backend, og Visual Studio er den primære IDE, man bruger, når man udvikler i C#. Dog, når jeg arbejder i JavaScript/TypeScript, arbejder jeg gerne i Visual Studio Code, fordi Visual Studio Code ikke er lige så tung som Visual Studio og har gerne nogle bedre udvidelser og syntaks, når det gælder frontend udvikling.

Jeg vælger derfor at bruge Visual Studio til at holde styr på min C# backend og Visual Studio Code til at holde styr på min TypeScript React frontend. På den måde får jeg det bedste af begge verdener uden at miste hverken syntaks highlighting eller eksterne konfigurationer for at finde ud af, hvordan jeg benytter ASP.NET's Swagger værktøj ved brug af Visual Studio Code.

Design

Før jeg går i gang med at udvikle en hjemmeside, vil jeg først vide, hvordan den ser ud. Hvis jeg designer min hjemmeside først, kan det spare mig en masse tid med at finde ud af, hvilke elementer der skal være på min side samt hjælpe mig på vej til definition af mine opgaver.

Her vil jeg helst bruge et reelt designværktøj som Adobe XD eller Figma. Begge applikationer tilbyder prototyping, hvilket betyder, at man kan integrere med designet, som var det en reel hjemmeside/app. Jeg ved, at man på webudvikleruddannelsen bruger Adobe XD, da jeg har venner, der har gået på uddannelsen og har fået flere designs i Adobe XD. Dog, da jeg kikkede efter Adobe XD, fandt jeg ud af, at XD er med i en betalt Adobe-pakke, som hvis jeg skulle bruge Excel, som er bag en betalt Office-pakke. Jeg har derfor valgt at bruge Figma som gratis alternativ.

Ved hjælp af Figma kan jeg designe i komponenter, hvilket ligger tæt op ad Reacts struktur, da React er et komponentbaseret værktøj. Som tidligere nævnt, er der også prototyping i Figma, så jeg kan integrere med mit design og få en bedre idé om, hvilke elementer, komponenter og sider, jeg skal have med i min hjemmeside.

Version- og opgavestyring

For at holde styr på projektets udvikling, ville det være klogt at version- og opgavestyre projektet. Når jeg har opgavestyring, er det nemt for mig at finde ud af, hvilke opgaver jeg skal have lavet før aflevering. Når jeg har versionsstyring, hjælper jeg mig selv med at finde ud af, hvornår jeg har ændret eller slettet noget i min kode, som evt. kunne have været med til at skabe fejl i applikationen. Derudover giver versionsstyring mig frihed til at omfaktorere min kode og ikke miste tid på at vende tilbage til det, jeg havde tidligere.

Jeg bruger GIT til versionsstyring, da GIT er standarden indenfor programmering, når det gælder versionsstyring. Herunder behøver jeg blot at vælge en service, der kan tilbyde mig nem adgang til GIT, så jeg ikke nødvendigvis behøver at skrive kommandoer for at committe, pushe og mege mine ændringer til min versionsstyringsrepository.
Heldigvis kender jeg allerede 2 værktøjer, som både tilbyder GIT men også opgavestyring.

Azure DevOps er en service, administreret af Microsoft, som er rimelig populært indenfor projektstyring ved programmeringsprojekter. På Skoleoplæringscenteret har vi fået meget erfaring med at benytte DevOps' SCRUM feature til at oprette Epics, Features, Backlog Items og Tasks.

Github har også for nyligt tilføjet en feature på deres service, der tilbyder TODO boards og Roadmaps. Github's "Project" tab er gerne forbundet til et Github repository og repository'ets "issues". Projektstyringssiden bruger repository'ets issues som opgaver, hvor man via TODO boardet kan tilføje sine egne kolonner, og man via roadmappet nemt kan flytte opgaverne rundt i en kalender.

Jeg er glad for begge services, men har valgt at bruge Github, da jeg synes, deres projektside er nemmere at overskue end Azure DevOps' sprintsside. Derudover har jeg også de fleste af mine projekter på min Github konto i forvejen, hvor min Azure DevOps konto er tilknyttet min skolekonto, som sandsynligvis bliver slettet, efter jeg er færdiguddannet.

Hosting

For at hjemmesiden og tilsvarende API og database kan tilgås udenfor min lokale maskine, skal jeg have fundet services, der tilbyder hosting af de 3 led. Når man leder efter en host, vil man selvfølgelig gerne have en billig og tæt host samt en pålidelig oppetid, så brugerne af produktet får en god oplevelse.

Som studerende på TECHCOLLEGE, tilbyder IT-Center Nord gratis hosting af hjemmeside samt database til 5 projekter. Tilsvarende, som TECHCOLLEGE programmeringsstuderende, er vi tilbudt et bestemt antal kredit hos Microsoft Azure, som vi kan benytte årligt. Ellers er der en masse andre services udenfor vores studenterbonus, der kan tilbyde gratis hosting som Netlify og evt. Heroku eller andre betalende services.

Alle mine lag er spredt over 3 forskellige services. Min database er hosted gennem ITCN, så jeg har gratis adgang til databasen. API'et er hosted via Azure, da ITCN ikke understøtter hosting af ASP.NET API, hvorimod der er en masse ressourcer på hosting af et C# API i Microsoft Azure. Til sidst hostes hjemmesiden via Netlify, da ITCN ikke understøtter Node.js hjemmesider (som React, Vue og Angular) og Azure ikke tilbyder gratis hosting af samme type service, hvorimod Netlify er 100% gratis, dog med fast hosting region sat i øst-USA.

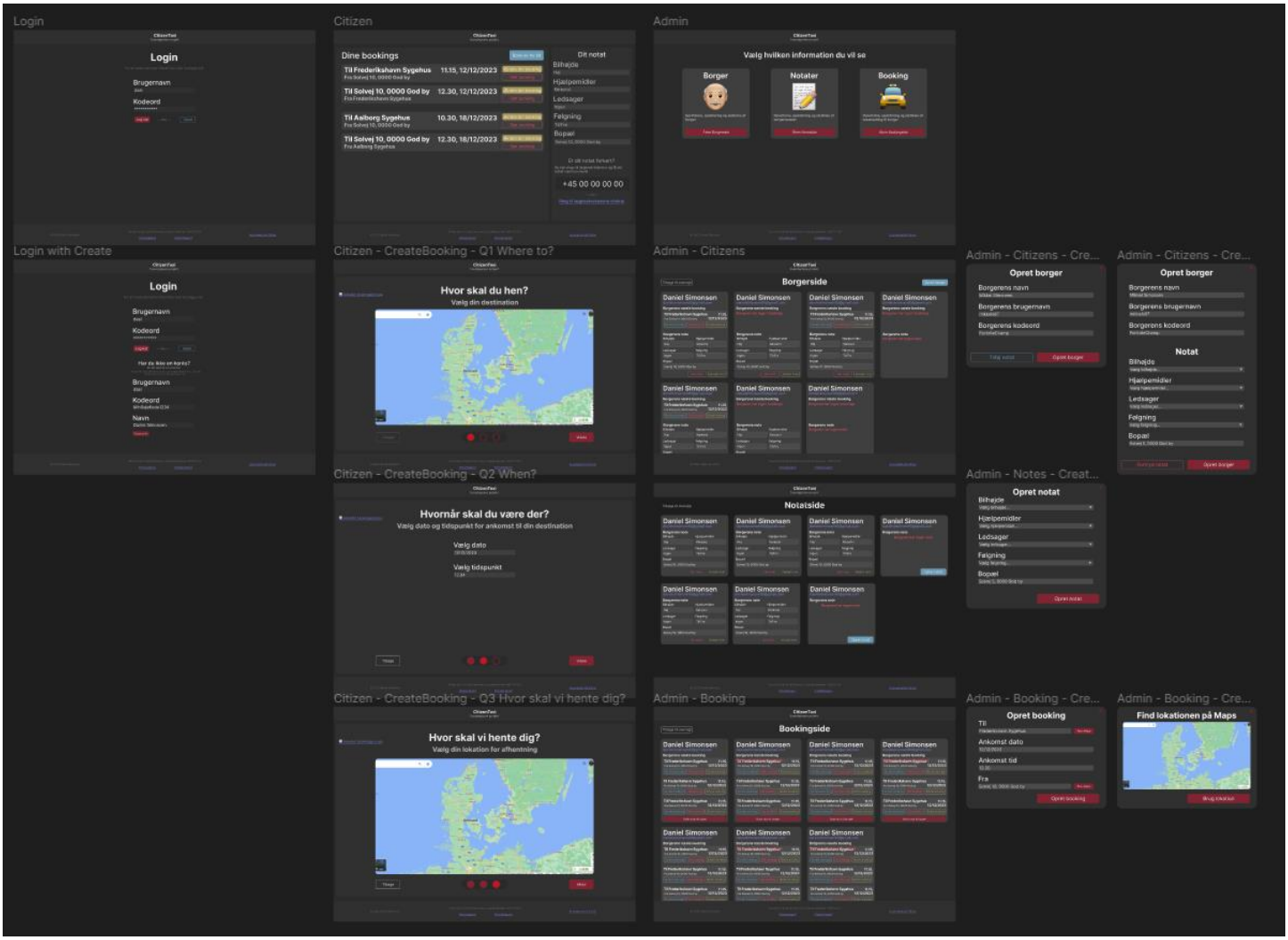
Grundet hjemmesidens placering har jeg derfor valgt også at hoste mit API i øst-USA, således at forbindelsen mellem hjemmesiden og API'et er hurtig, men forbindelsen mellem hjemmesiden og brugeren er langsom.

Design

Som tidligere nævnt bruger jeg Figma til at designe min hjemmeside. Figma er komponentbaseret ligesom React frameworket, som jeg bruger til udvikling af den reelle hjemmeside.

Her har jeg delt mit Figma projekt op i 2 dele: Selve hjemmesidedesignet og alle dens gentagende komponenter samt variationer.

Her vises det originale design til hjemmesiden, da den er mere relevant af de 2 dele.



Alle design-relaterede komponenter og elementer kan findes under mine [ressourcer mappe på GitHub].

Konklusion

Jeg er stolt over det færdige produkt og føler at produktet, med noget mere tid og udvikling, kunne fungere godt til løsning af lægesekretærerne, borgerne og især flexsygehustaxa-servicens problem med borgernes lange telefonkøer.

Mine testpersoner synes alle, at produktet fungerede godt som prototype og ville elske at have produktet som reel løsning.

Diskussion

Jeg synes, produktet er blevet rigtigt godt på baggrund af min viden indenfor programmering. Havde jeg haft noget mere tid, ville jeg have kikket mere på sikkerhed og evt. tilbyde Google Maps, så borgerne bl.a. kan finde deres adresse via kort. Derudover ville selve oprettelsesformularen på loginsiden have være fjernet, da den kun eksisterer grundet prototype/demo situation. Der mangler også en side til at administrere alle administratorer, hvor man evt. kunne overveje en ny rolle specielt til lægesekretærerne i stedet for at give dem fuld administrator adgang.

I forhold til teknologivalg kunne jeg have overvejet at blive i samme sprog – enten lave et projekt i fuld C# og bl.a. have en Blazor hjemmeside eller gå fuld TypeScript og have en Express backend. Det blev lidt rodet at arbejde med 2 forskellige sprog – især fordi jeg skulle skrive hele mit Common lag om i TypeScript for at kunne fortolke mit JSON data fra API'et korrekt.

Projektet har hjulpet mig i både tidsplanlægning af mit arbejde og arbejde med testpersoner. Tidsplanlægning vil selvfølgelig altid være et skud i tågen, men jeg føler, at jeg er blevet bedre til at planlægge mine projekter og mine arbejdsopgaver. I forhold til samarbejde med testpersoner ville jeg have ønsket, at jeg kunne overvåge deres testproces fysisk, som man normalt ville gøre. Dog grundet geografi, offentligt transporttid og begrænset projekttid, havde jeg ikke mulighed for dette, hvilket jeg synes er en skam, for det kunne helt sikkert have været en bedre oplevelse for alle, hvis jeg var til stede og overvågede testprocessen.

Logbog

6. november
Jeg har været i tvivl om valg af svendeprøveprojekt, og er splittet imellem online pakkeleg og online bestilling af patienttaxa. Jeg har brugt dagen på at brainstorme begge idéer, og har bl.a. ringet til min mor, som er lægesekretær, for at få svar på nogle spørgsmål angående patienttaxaidéen, da problemet er noget hun oplever i sin hverdag.

7. november
Fik brainstormet patienttaxa idéen og efter vejledning med Frank, konkluderede jeg, at det var det mest fyldegørende projekt af de to. Jeg lavede derefter mine kravspecs.

8. november
Jeg brugte hele dagen på frontend design i Figma, så jeg har et bedre overblik over projektet, og hvad jeg skal have lavet og tænkt over. Her kom mange nye idéer til properties men også bl.a. integration med Google Maps. Efter vejledning med Lars besluttede jeg at omdøbe ”patienttaxa” til ”borgertaxa”, da borgerne ikke nødvendigvis er patienter ”endnu”.

9. november
Efter jeg blev færdig med mit design i går, har jeg fået et godt overblik over, hvad jeg skal have udviklet og implementeret. Jeg brugte derfor dagen på at oprette opgaver og sætte dem i tidsplan. Derudover fik jeg startet på procesrapporten.

10. november
Fik oprettet mine backend projekter (Common, DataAccess, Business & API) samt relevante modeller og funktionalitet – dog uden SignalR og authenticate funktionalitet.
Havde nogle problemer med mine migrations og mine GET operationer, som jeg kæmpede med i noget tid.
Sluttede dagen af med at dokumentere det nye kode.

13. november
Fik lavet DataAccessTest projekt i NUnit frameworket. Jeg havde lidt svært ved at oprette en in-memory database, da jeg ikke kunne kalde metoden fra EntityFrameworkCore. Jeg fandt ud af, at UseInMemoryDatabase metoden findes i EntityFrameworkCore.InMemory NuGet pakken, hvilket var lidt irriterende.
Derudover opdagede jeg, at DbSet.ToList() kalder databasen hvorimod DbSet.Find() bruger cache, hvilket var ret forvirrende, da jeg skulle teste min Exists og Add metoder i mine repositories.

14. november
Har arbejdet hele dagen på API test. Jeg var ret forvirret over, hvordan jeg overhovedet testede mit API-projekt og havde også nogle EntityFramework tracking-problemer, når jeg testede min Update metode. Men jeg nåede endelig i mål, efter at have kæmpet med det i næsten 12 timer.

15. november
Fik startet på frontenden og fuld implementerede authentication og password hashing. Mangler kun dokumentation, men nu har jeg igen brugt 12 timer af dagen med fuld fokus på mine opgaver, så dokumentationen udskydes til i morgen.

16. november
Jeg fik lavet mit dokumentation og arbejdet på login systemet i frontend. Mangler noget styling og evt. dokumentation.

17. november
Jeg fik stylet min login og implementeret min ”CitizensProvider”, så jeg har adgang til borgerens oplysninger via en service/provider i min frontend.

18. november
Implementeret det meste af borgersiden. Brugte max en time.

19. november
Blev færdig med hoveddelen af borgersiden og startede på create delen af bestillingerne.

20. november
Blev færdig med create og update af bookings. Sad med en pagination bug i nogle gode timer men fik det endelig løst. Startede derefter på SingalR for live notifikationer fra taxaen.

21. november
Efter at have opsat alt mit SignalR i backend og frontend, kunne jeg ikke få en timer til at køre i min backend, som skulle sende min frontend notifikationer. Endte med at simulere det i frontenden i stedet.

22. november
Jeg fik startet på admindelen og lavede de fleste komponenter generiske, således at jeg kan bruge dem på alle 3 sub-sider.

23. november
Mere arbejde på komponenterne til adminsiden.

24. november
Refactor api requests til en redux-like action dispatch system, så alle requests burde håndteres ens og ikke sprede tilfældige requests over alle komponenter. Fik en fejl med mine modals, der ikke gad at lukke ordentligt, som jeg ikke fik fikset.

25. november
Fik refactoreret og fikset adskillige fejl efter næsten 11 timers frustration. Derudover fik jeg implementeret de resterende undersider, som admin-noter og admin-bestillinger, så nu burde hjemmesiden være funktionel og mangler nogle komponenttests, dokumentation og hosting.

26. november
Fik fjernet de sidste TODO's og legede med nogle farver med lightmode brugere i bagtanke. Derudover har jeg brugt hele dagen på usabilitytestopgaver og fikse de fejl, der kom, da jeg løste opgaverne selv.

27. november
Brugte 6 timer på at få API og Frontend til at snakke sammen. Stødte ind på en masse publish og CORS fejl, men efter stor frustration og stor koncentration, fik jeg løst det. Jeg venter nu på, at min mor tester mit projekt sammen med hendes kollegaer, da projektet er inspireret af en reel situation, de kæmper med på arbejdet.

Resten af aftenen blev brugt på at omformulere min metode- og teknologivalg sektion her i procesrapporten, da jeg havde struktureret mine valg uden flow.

28. november
Fik startet på produktrapporten og blev ringet op af min mor, fordi hun fandt en fejl og derfor sad fast i test-processen. Fejlen var React, der ikke registrerede værdien i inputfeltet, og påstod at der ikke var en værdi. Jeg fik den fikset indenfor 10 minutter, men da det var sent på dagen, ville mor og hendes kollegaer teste produktet af igen i morgen.

29. november
Jeg har fået arbejdet en del på begge rapporter, og mangler kun nogle få emner samt vejledning, før jeg kan sende det videre til gennemlæsning før aflevering.

30. november
Fik noget vejledning til rapporter og fremlæggelse og begyndte at skrive brugervejledning.

1. december
Fik de sidste usabilitytest resultater og fik rettet på mine rapporter.