|  |
| --- |
| [Forfatterens navn]  01-12-2023 |

|  |
| --- |
|  |
| Procesrapport |
| CitizenTaxi |



|  |
| --- |
| **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)  (Thomas Vrangbæk Thomsen) |
|  |

# Titelblad



Techcollege Aalborg,

Struervej 70,

9220 Aalborg

# Læsevejledning

Jeg anbefaler at læse denne rapport først.

// TODO: Ord-/fortkortelsesliste

Indholds fortegnelse

[Titelblad i](#_Toc150432571)

[Læsevejledning 1](#_Toc150432572)

[Indledning 3](#_Toc150432573)

[Case beskrivelse 3](#_Toc150432574)

[Problemformulering 3](#_Toc150432575)

[(Afgrænsning) 3](#_Toc150432576)

[Projektplanlægning 4](#_Toc150432577)

[Estimeret tidsplan 4](#_Toc150432578)

[Arbejdsfordeling 4](#_Toc150432579)

[Metode- og teknologivalg 4](#_Toc150432580)

[Realiseret tidsplan 5](#_Toc150432581)

[Konklusion 5](#_Toc150432582)

[Diskussion 5](#_Toc150432583)

[Bilag 6](#_Toc150432584)

[Logbog 7](#_Toc150432585)

# 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 borgeren.

Lægesekretærerne kan derefter ringe via internt nummer og få taxaen booket 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 prototype løsning til Flexsygehustaxas lange ventetid, afgrænser jeg bl.a. sygehus adgang (f.eks. bruge deres eksisterende login oplysninger) og rigtig taxa bestilling (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.

A screenshot of a computer

Description automatically generated

## 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 gå 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 kom i mange forskellige problemer 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 betyder 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. Dog 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.

Dog er der selvføligelig 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 C#, eller vælge at lave en hjemmeside via et JavaScript framework/library som Vue, React eller Angular.

Da hjemmesiden er rimelig stor ift. at der skal være et loginsystem, borgerside med bestillinger samt administrationsside med håndtering af borgere, notater og bestillinger, ville det være en træls 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 så være et JavaScript framework/library. Herunder har jeg aldrig prøvet Angular, så der er kun Vue og React tilbage. Vue har vi lært på hovedforløb 4, men jeg har ikke brugt frameworket siden. Til gengæld, har jeg siden da lært og udviklet nogle sider i 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 min borgersides bestillingsproces, samt de mange modals/popups jeg skal bruge til oprettelse, opdatering og slettelse 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 ligesådan 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 så være mellemled mellem mit gemte data og min hjemmesides anmodninger til at få det 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 spille fint sammen med min React hjemmeside, da man både kan lave et Express API i Java- og TypeScript, som React også understøtter. Det vil sige, at mit udviklingsmiljø ville foregå indenfor Java-/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 rute anmodninger. Derudover har ASP.NET allerede indbygget sikkerhed og giver mig et nemt og overskueligt udviklingsmiljø.

Da jeg er stor fan af controller mønsteret 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 selv lave mit eget controller system. 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 med Java-/TypeScript. Det kommer dog også an på hvilke ORMe, 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 mit data usikkert. Ved brug af en ORM, fortæller jeg ORM’en, hvilke modeller hører til hvilke tabeller, sådan 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 helt stærk 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, administratore, 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 gør ikke det helt store, 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 skrive noget kode, 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 spillet 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 er oftest i C# jeg udvikler backend, og Visual Studio er den primære IDE man bruger, når man udvikler C#. Dog når jeg arbejder i Java-/TypeScript arbejder jeg gerne i Visual Studio Code, fordi Visual Studio Code er ikke lige så tung som Visual Studio, og har gerne nogle bedre udvidelser, 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, skal jeg lige 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 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 webudvikler uddannelsen 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. Så som gratis alternativ, har jeg valgt at bruge Figma.

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 selvfølgelig 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 versionsstyrings repository.

Heldigvis kender jeg allerede 2 værktøjer, som både tilbyder GIT men også opgavesstyring.

Azure DevOps er en service administreret af Microsoft, og 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 ny 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 kalenender.

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, og jeg vil helst gerne beholde projektet efter uddannelsen.

## Realiseret tidsplan // TODO

# Konklusion // TODO

Skal hænge sammen med case beskrivelsen og problemformuleringen.

## Diskussion

Diskutere forskellige side af løsningen, fordele/ulemper.

Hvis der var mere tid, hvor kunne produktet udvides eller forbedres.

Reflekter over projektet

Hvad har du lært?

Valgte du de rigtige teknologier?

# Bilag // TODO

[Figur 1 Placeholder billede 4](#_Toc150432394)

[Figur 2 Biag A: Estimeret Tidsplan 7](file:///C:\Users\Ejer\Downloads\Processraport%20template.docx#_Toc150432395)

# Logbog

**6. november**

Ringede til mor ang. Patienttaxa ide og fik nogle svar besvaret. Brugte dog dagen på at brainstorme min pakkeleg ide.

**7. november**

Fik brainstormet patienttaxa idéen, og efter vejledning med Frank, konkluderede 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 mig 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 har derfor brugt 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 instanserne en in-memory database, da jeg ikke kunne kalde metoden fra EntityFrameworkCore. Jeg fandt så 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 min dag væk på API test. Jeg var ret forvirret over, hvordan jeg overhovedet testede mit api projekt og havde også noget 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 brugt 12 timer igen, så det må vente 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 hoved-delen af borgersiden og startede på create delen af bookings.

**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 mig 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 admin delen, og lavede de fleste komponenter generiske, sådan 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 sub sider, som admin-noter og admin-bestillinger, så nu burde hjemmesiden være funktionel og mangler nogle komponent tests, dokumentation og hosting.

**26. november**

Fik fjernet de sidste // TODO’s og kikket legede med nogle farver ift. lightmode brugere. Derudover har jeg brugt hele dagen på usabilitytest opgaver 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 tungen i munden, 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.