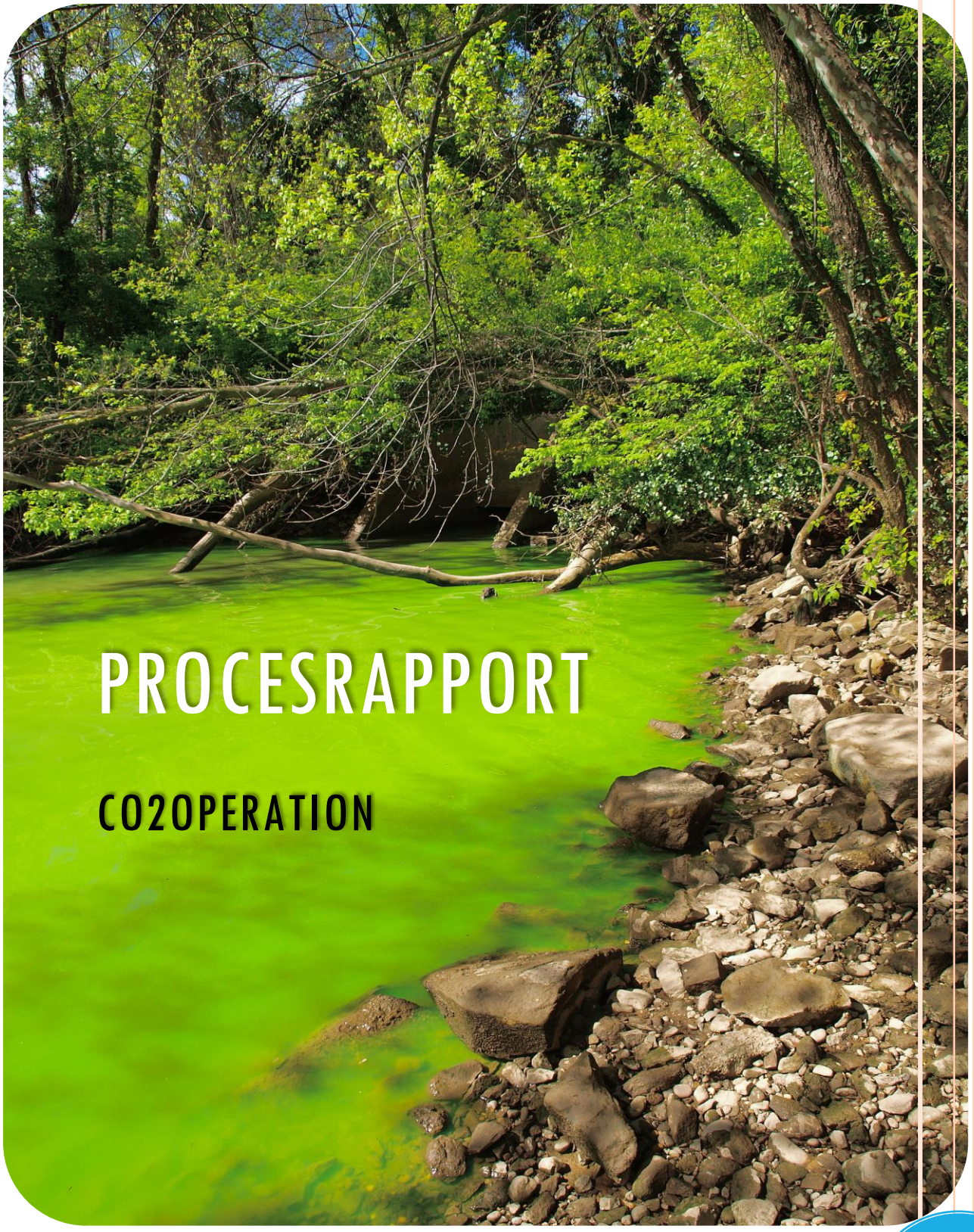


PROCESRAPPORT

CO2OPERATION

Jonas, Benjamin, Casper, Andi

Dato:	**06/10-2020**
Skole:	**ZBC Ringsted**
Uddannelse:	**Datatekniker /m. speciale i programmering**
Klasse:	**4008h3dakp**

Projektdeltagere:

Andi

Benjamin

Casper

Jonas

Vejledere:

Camilla Mai Ryskjær

Mikkel Andreas Krøll Christiansen

Indhold

1. Læsevejledning & Introduktion

Procesrapporten er opdelt i forskellige dele, alt efter hvor i processen rapporten omhandler. Hvert større afsnit vil have en kortere forklaring tilknyttet, hvori vi mere specifikt forklarer hvad læseren kan forvente af den følgende del af processen.

Vi starter med en indledning til projektet, hvor der bliver forklaret hvilke kriterier, baggrund og tilgang vi har taget til projektet. Dette bliver efterfuldt af en længere problemstilling, hvori vi beskriver den faktuelle baggrund for hvorfor vi har valgt projektets problemformulering. Dette vil fungere som en baggrund til projektet, samt en indledning til resten af rapporten.

Herefter starter vi med en udpensling af foranalysen. Dette inkluderer analyse-diagrammer, som vi ikke følte hørte til i den tilhørende produktrapport, da det rettere beskrev hvordan vi var nået frem til de beslutninger vi har defineret deri.

Næste del omhandler analyse delen, der er blevet designet før projektet blev opdelt i de mindre subsystemer. Dette inkluderer de større diagrammer, som strækker sig over hele systemet. Vi har valgt at inkludere dette i procesrapporten, da vi føler at det giver en bedre baggrund til de design beslutninger vi begrunder senere i rapporten. De diagrammer der kun omhandler det færdiggjorte projekt, kan findes i produktrapporten.

I design delen, begrunder vi vores valg af udviklings arkitektur og designmønstrer. Dette inkluderer længere og mere dybdegående forklaringer der ligger bag vores implementeringer. Vi har valgt at tage udgangspunkt i at læseren allerede ville have et kendskab til principperne, og har derfor ikke forklaret de mere basale ting bag. Men i stedet hvorfor vi har valgt at bruge den specifikke teknik i dette projekt.

Teknologivalg omhandler hvilke beslutninger der ligger bag vores valg af diverse frameworks og teknologier. Dette inkluderer en kort beskrivelse af teknologien, samt hvordan teknologien er blevet brugt projekten. Det skal dog siges, at valg af teknologi, kan være meget præget af personlig præference og familiaritet. Hvilket selvfølgelig også er tilfældet i dette projekt.

Kapitel 9 omhandler specielle uddrag fra produktet, som vi mener fortjener ekstra fremhævning. Her giver vi en dybere forklaring på tankegangen bag den specifikke funktionalitet eller arkitektur, som ellers nemt kunne gå tabt i mængden i et projekt af denne størrelse.

I den næste del finder man dokumentationen for de arbejdsdage der har været til rådighed i projektperioden. Dette er dokumenteret i form af noter fra de daglige morgenmøder. Der kan desuden findes referater af opstartsmødet og midtvejs-mødet, hvori man kan finde hvilke overvejelser vi havde gjort os på de forskellige tidspunkter i forløbet. Dette bliver efterfuldt af den realiserede tidsplan, hvor man kan finde en visualisering af hvordan processen reelt endte med at forløbe tidsmæssigt.

Som det sidste finder man konklusion vi er kommet frem til, på baggrund af vores analysearbejde.

Happy reading!

2. Indledning & Opgavebeskrivelse

Vi skal udvikle et system inspireret på et af FN's 17 verdensmål for bæredygtig udvikling.

Vi har valgt at fokusere på mål 13: Klimaindsats, samt det mere specifikke delmål: 13.3, som lyder:

Klimaindsats: 13.3 Forbedre undervisning, viden, og den menneskelige og institutionelle kapacitet til at modvirke, tilpasse, begrænse skaderne og tidlig varsling af klimaændringer.

For at hjælpe med at opnå dette verdensmål mål, har vi valgt at udvikle en løsning der hjælper brugeren med at holde styr på deres eget CO2 aftryk, specifikt i sammenhæng med transport. Ved hjælp af systemet vil brugeren kunne få et større overblik over hvad effekt deres hverdag kan have på klimaet.

Vores løsning vil tage form af en android Xamarin applikation hvori vi viser en Angular SPA webapp. Androidapplikationen læser løbende på brugernes bevægelse og sender rejse informationen videre via API til vores logik lag. Bruger interaktion i android applikationen, sker via Angular applikation indlejret heri.

Angular applikationen giver mulighed for oprettelse af bruger profil, samt loginfunktionalitet, håndteret over API kald til logik-laget. Angular applikationen modtager rejse statistik information for den bruger der er logget ind via API.

Data i større mængder fra API'en til Angular applikation bliver håndteret via en GraphQL forbindelse. GraphQL giver mulighed for at udvælge hvilke data der skal returneres via API kaldet, baseret på domænemodellen. I tilfælde af at der kun er behov for at returnere enkelte dele af diverse modeller, vil en GraphQL løsning resultere i langt hurtigere svar tider, samt simplere udvidelsesmuligheder.

2.1 Link til kode biblioteker:

Angular SPA / Frontend:

https://github.com/bentikki/CO2OPERATION_WebApp

Xamarin:

https://github.com/Casper-Nielsen/cooperationApp1

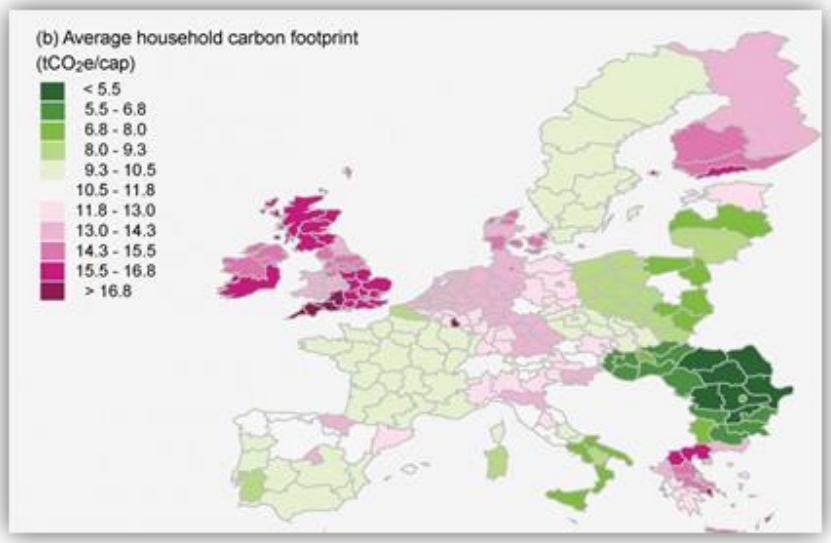
Samlet repository for MVC .NET API, Backend og GraphQL:

https://dev.azure.com/Co2Operation/Co2operation/_git/Co2operation

3. Problemstilling

3.1 Indledning

Ifølge en rapport udarbejdet i 2017, er danske husstande blandt de fem værste, hvad angår udledning af C02, med hele 14,5 tons CO2 udledt årligt. Det er 3 tons over det gennemsnitlige 11,5 tons man finder i EU (Ivanova et al. 2017, s. 3-9). Dette er højere end vores nærmeste naboer i Tyskland med 11,8, samt markant højere end Norge med 9.3.



3.2 Klima aspekter

Co2 udledningen bliver delt op i 3 kategorier: mad, tøj og mobilitet, hvor det fremgår at de forskellige regioner ligger forskelligt på de tre områder. F.eks. Ligger region Hovedstadens CO2 aftryk hvad angår mad på 3.894Kg pr. indbygger årligt, mens man i Nordjylland har et gennemsnit på 3.380Kg. Omvendt ligger man i Hovedstaden mobilitetsmæssigt meget lavere end Nordjylland, med et gennemsnitligt CO2 forbrug på henholdsvis 4.200kg/årligt i Hovedstaden, mod 5.000kg/årligt i region Nordjylland (Ivanova D, et al 2017, Mapping the carbon footprint of EU regions, IOP Publishing. Lokaliseret d. 27.08.20:, u.d.).

3.3 Co2 Typer

Endvidere bliver CO2 udledningen delt op i 2 typer, den direkte- og indirekte udledning. Den direkte udledning dækker eksempelvis opvarmning af huse og udstødning fra biler. Den indirekte dækker 'usynlige' kilder, dette kunne være hvor meget CO2 der er blevet brugt ved fremstillingen af et stykke tøj samt, hvis man medberegner transporten til Danmark, før den bliver solgt til forbrugeren (Ivanova D, et al 2017, Mapping the carbon footprint of EU regions, IOP Publishing. Lokaliseret d. 27.08.20:, u.d.).

Det gennemsnitlige CO2-aftryk for en dansk husstand

3.4 Danske Husstande

Den største CO2 synder i danske husstande, findes i forhold til madvarer (Ivanova D, et al 2017, Mapping the carbon footprint of EU regions, IOP Publishing. Lokaliseret d. 27.08.20:, u.d.). Den danske husstand har et årligt gennemsnits forbrug af madvarer imellem 3.531 og 3.894 kg, afhængigt af regionen.

3.5 Information tilgængelighed

Desuden fremgår det af rapporten at omkring halvdelen af europæere føler sig godt eller meget godt informeret omkring de forskellige aspekter af klimaforandring samt hvordan man kan gøre noget ved det. Dog føler én ud af ti at de overhovedet ikke er informeret omkring nogle aspekter vedrørende dette ((European Commission 2008, s 18), u.d.).

4. Problemformulering

Hvordan laver vi en teknisk løsning der kan hjælpe med at reducere brugerens CO2 aftryk?

5. Foranalyse

I denne del af procesrapporten, dokumenterer vi hvilke overvejelser og design redskaber vi har brugt, før den mere "kode"-mæssige analyse del kunne begynde. Dette inkluderer diskussion om metodevalg, ide-afstemning i form af rigt billede, Use Case diagram, samt FURPS.

5.1 Metodevalg

Vi har valgt at arbejde ud fra den agile projektmodel, primært i form at daglige scrum møder og iterations møder med 2 dages sprint imellem. Vi har valgt denne fremgangsmåde da vi ikke kunne være sikre på hvor meget udviklings tid der reelt ville være til projektet. Ved at vælge den agile udviklings strategi, sikrer vi os så meget som muligt imod uforudsete udfordringer, samt ændringer i tidsplanen. Dette har også været en fordel, da dette projekt har været i mange forskellige dele. De daglige møder har gjort det lettere at holde styr på hvor langt de forskellige personer er kommet med deres ansvarsområder. Derved har vi kunne danne os et bedre overblik over hvor langt projektet er i sin helhed.

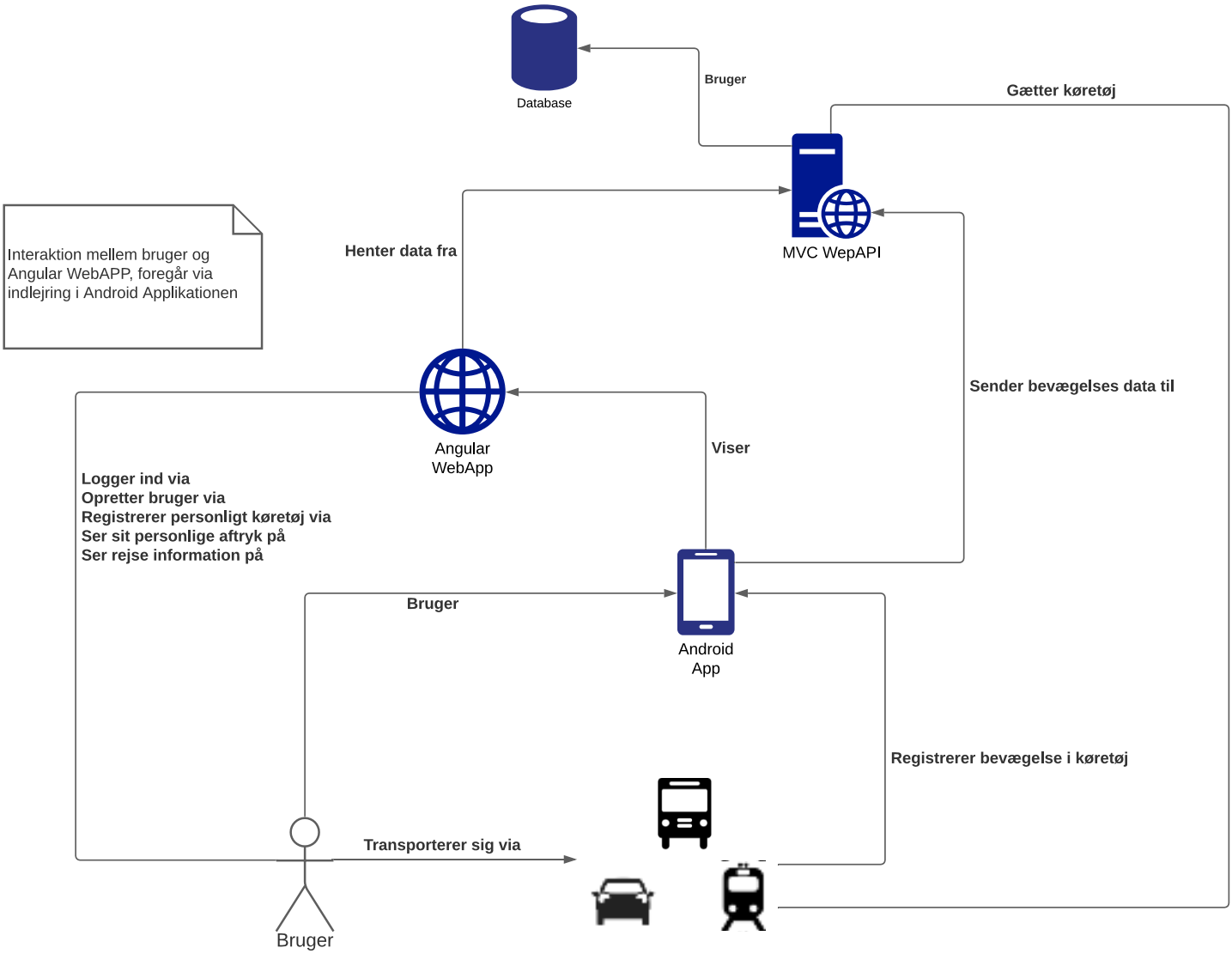
Et minus ved vores fremgangsmåde har været, når vi skulle sammenkoble de mange enkelte dele. Kommunikationen imellem de forskellige dele af systemet, samt de mere abstrakte ideer bag sammenspillet, har været svært at danne sig et overblik over. Set i bagspejlet, havde det været en god idé at designe et Sequence Diagram før vi startede udviklingsdelen af opgaven. Dette må vi tage med til næste projekt.

Vi har valgt at bruge den første halvanden uge af projekttiden på at lave foranalyse, i form af idegenerering, FURPS, Use Cases, Wireframes, samt et Rigt Billede. Vi gik til projektet med tankegangen om at fremgangsmåden var mere vigtig end slutproduktet, derfor har vi valgt kun at gå efter at have en prototype klar, frem for et fuldendt produkt.

5.2 Rigt Billede

Projektstart gik på at udarbejde et rigt billede, dette gjorde vi for hurtigt at afstemme hvad projektet skulle udarbejdes efter, i grove træk. Vores første iteration var dog væsentlig mindre visuel, men ideen bag var den samme.

Vi har her valgt at visualisere hvilket sammenhæng vi planlægger der vil være igennem de forskellige subsystemer og handlinger. Mere abstrakt har vi valgt at visualisere at aktøren **Bruger** transporterer sig via et køretøj, hvilket bliver opfanget af **Android App**'en. Selve udregningen af hvilken form for transportmiddel der er blevet brugt på en rute, bliver dog udregnet i vores Back-end, baseret på faktorer som hastighed og rute. Se Figur.



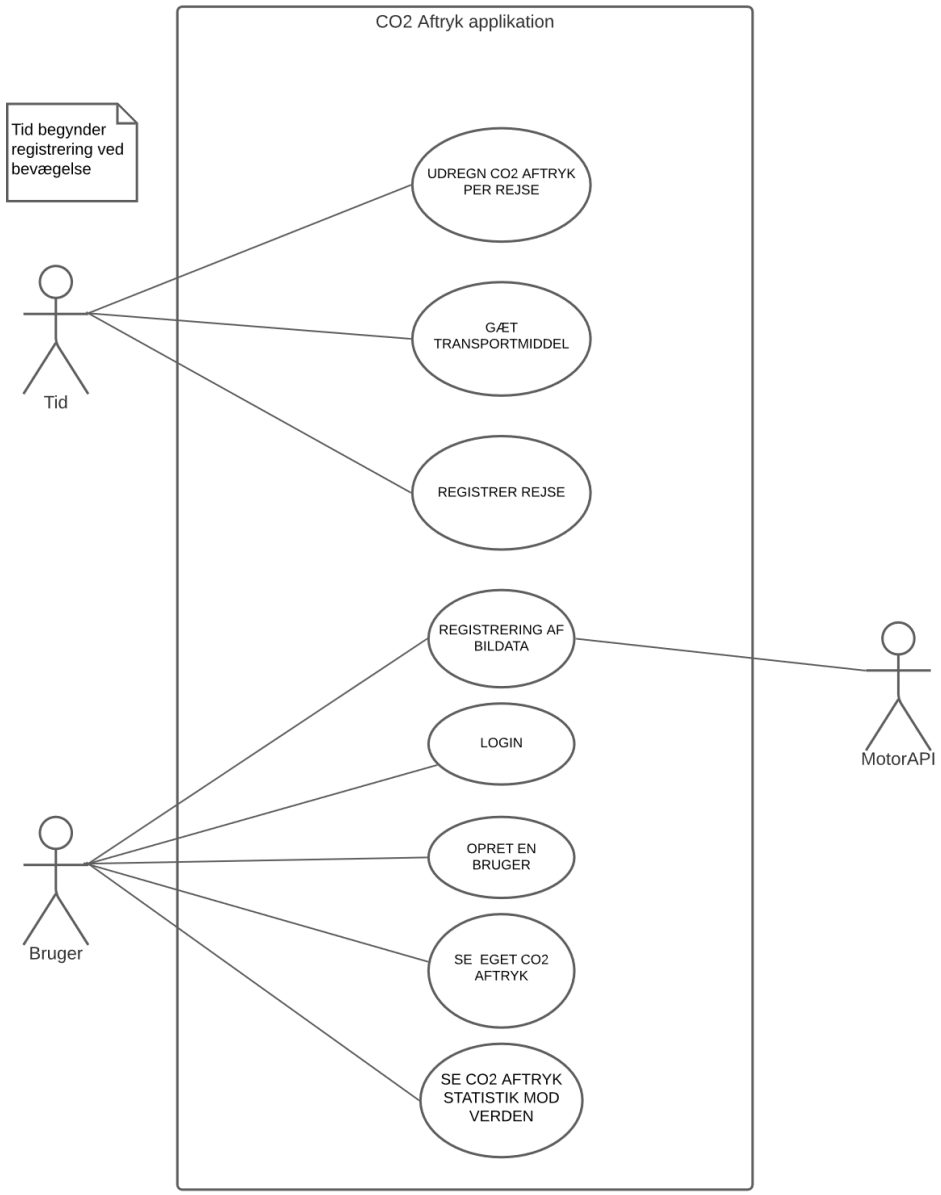
Figur 1: Rigt Billiede over systemet

5.3 Use Case diagram

Vi har på baggrund af vores projekt, fundet følgende Use Cases. Et antal aktører findes desuden i Use Case diagrammet, her visualiseres både aktøren **Tid** og aktøren **Bruger.** Vi har valgt at vise alle Use Cases, der lige nu er funktionelle i vores system. Use Cases som ikke endte med at blive realiseret, kan findes i diverse iterationsmøde noter som bilag.

Aktør beskrivelser:

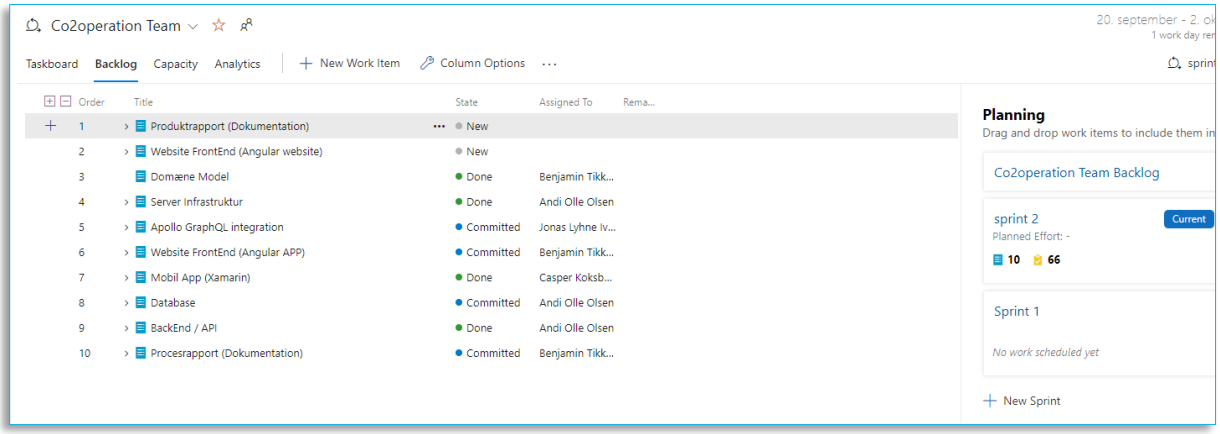
Bruger	Her refereres til en bruger der har installeret android-applikationen. Denne aktør viser kun de aktive funktioner brugeren kan udføre.
Tid	Tid er en mere abstrakt aktør. Her refereres til de passive funktionaliteter der køres af systemet, på baggrund af brugerens handlinger, men uden brugerens input. Disse bliver primært aktiveret efter specifikke tidsperioder, eller ved registreret bevægelse.

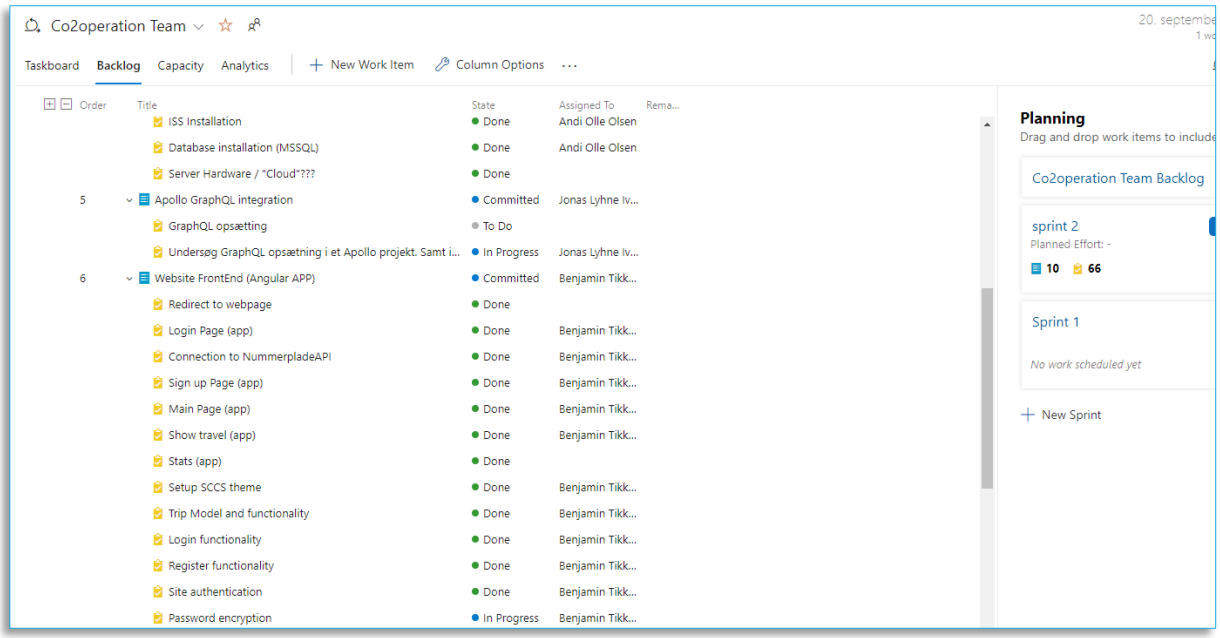


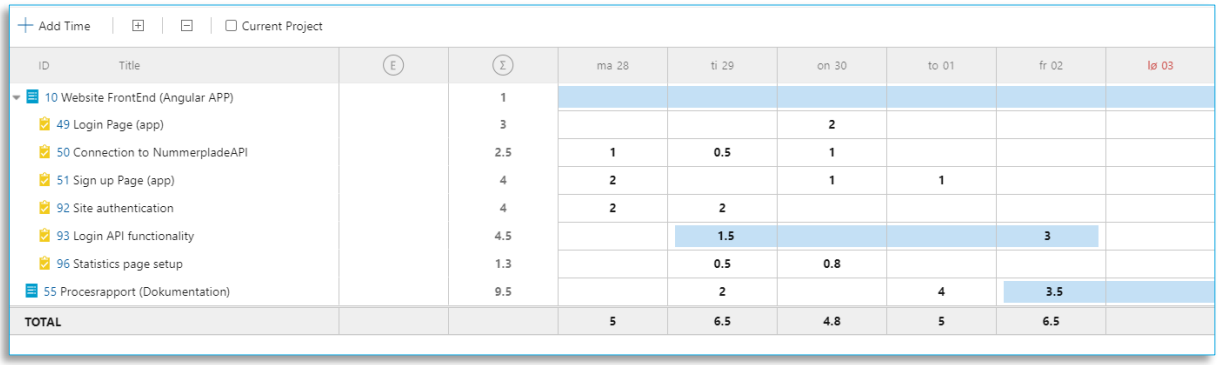
Figur 2: Use Case – diagram

5.4 Estimeret tidsplan

Da vi har valgt en Agil projektmodel, har vi ikke lavet en estimeret tidsplan. I stedet har vi kørt efter Use Cases og tasks, opdelt i diverse Epics. Vi har brugt Azure DevOps, til at holde styr på den igangværende proces (Se figur). Ved hver task har vi registreret hvor meget tid der reelt er blevet brugt, samt hvilken kategori (design, udvikling, test mm.) den specifikke task ligger under. Den realiserede tidsplan vil blive gennemgået i et senere afsnit.







ID	Title	E	Σ	ma 28	ti 29	on 30	to 01	fr 02	lø 03
10	Website FrontEnd (Angular APP)		1						
49	Login Page (app)		3			2			
50	Connection to NummerpladeAPI		2.5	1	0.5	1			
51	Sign up Page (app)		4	2		1	1		
92	Site authentication		4	2	2				
93	Login API functionality		4.5		1.5			3	
96	Statistics page setup		1.3		0.5	0.8			
55	Procesrapport (Dokumentation)		9.5		2		4	3.5	
TOTAL				5	6.5	4.8	5	6.5	

5.5 FURPS

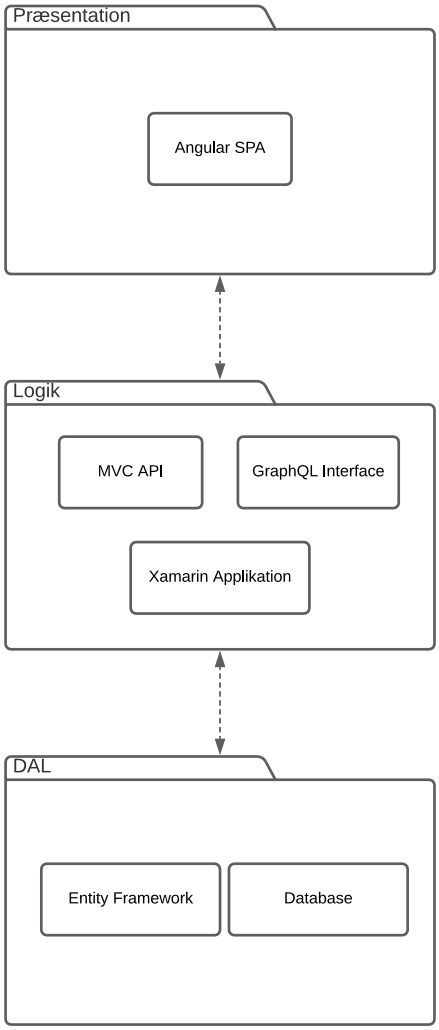
Vores FURPS-dokumentation er vedlagt i det eksterne bilag: ***Co2operation–FURPS.PDF*** (Bilag 1)

6. Analyse

Denne del af procesrapporten vil omhandle de mere "kode" relaterede analyse diagrammer. Dette inkluderer den logiske arkitektur, Domænemodellen, System Sekvens Diagram over hele systemet, samt Wireframes over brugerfladen.

6.1 Logisk Arkitektur

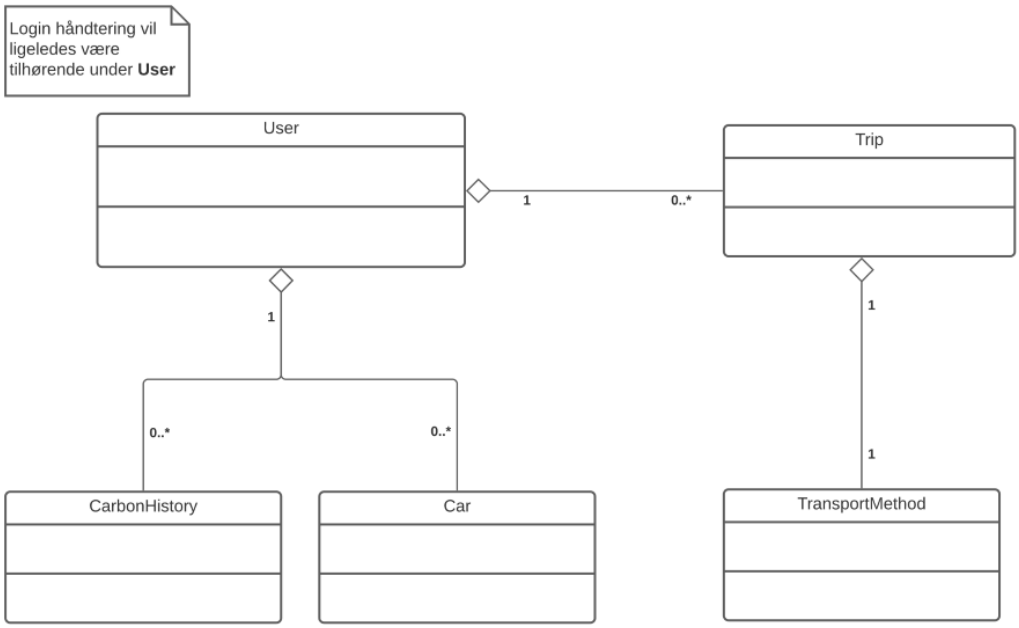
Vi har delt vores logiske arkitektur ud fra 3-lags modellen. Mange af de enkelte dele er yderligere delt op i lag, efter SOLID principperne. Derfor vil denne model vise en mere abstrakt idé om vores opdeling af sub-systemer.



Figur 2: Logisk arkitektur model

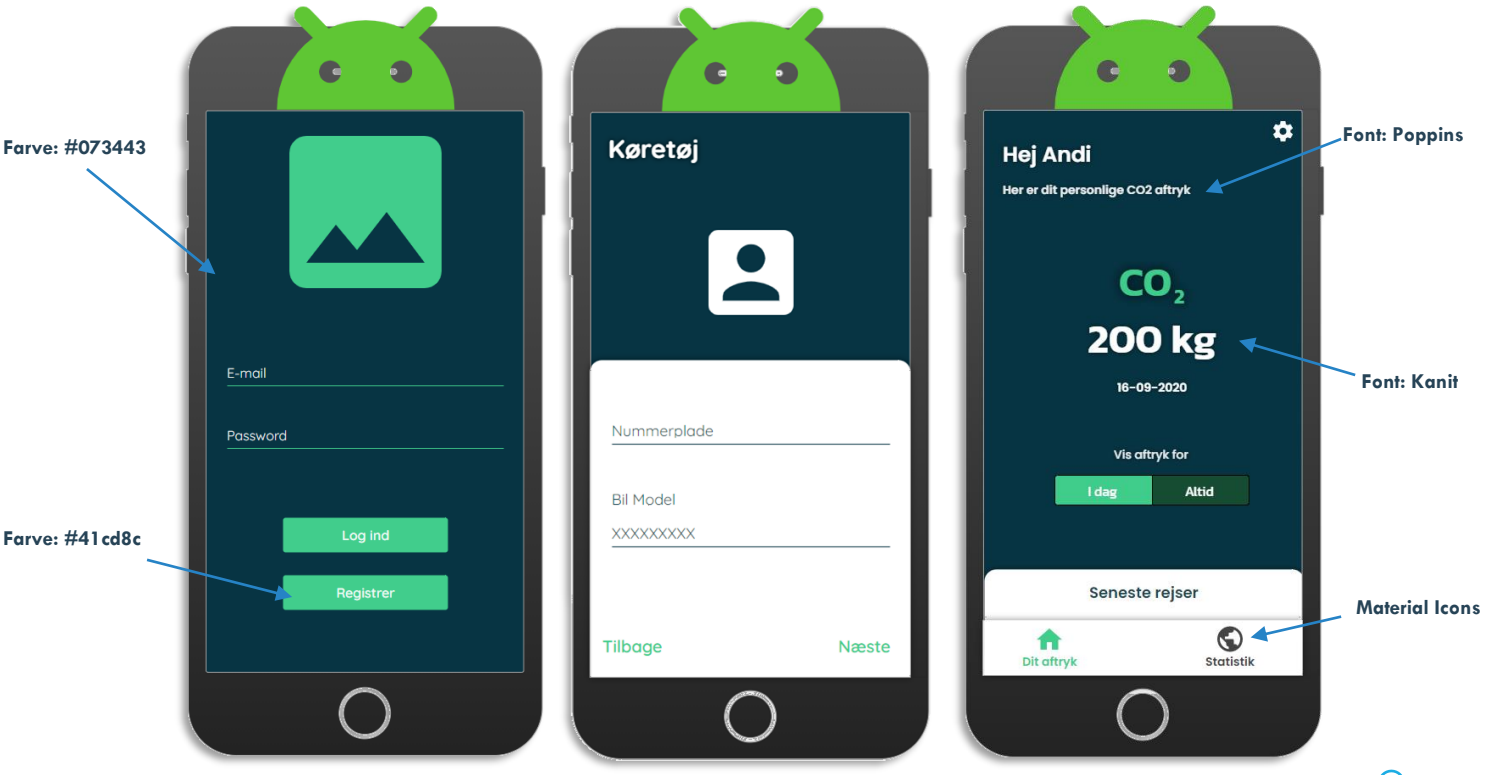
6.2 Domænemodel

Domænemodellen er udviklet ud fra vores første iterations planlægning. Den har ikke været nogle markante ændringer til modellen igennem udviklingsprocessen, og domæne-modellen samt kardinaliteter har holdt hele vejen igennem. De specifikke klassediagrammer bliver defineret nærmere senere i rapporten.



6.3 Wireframes

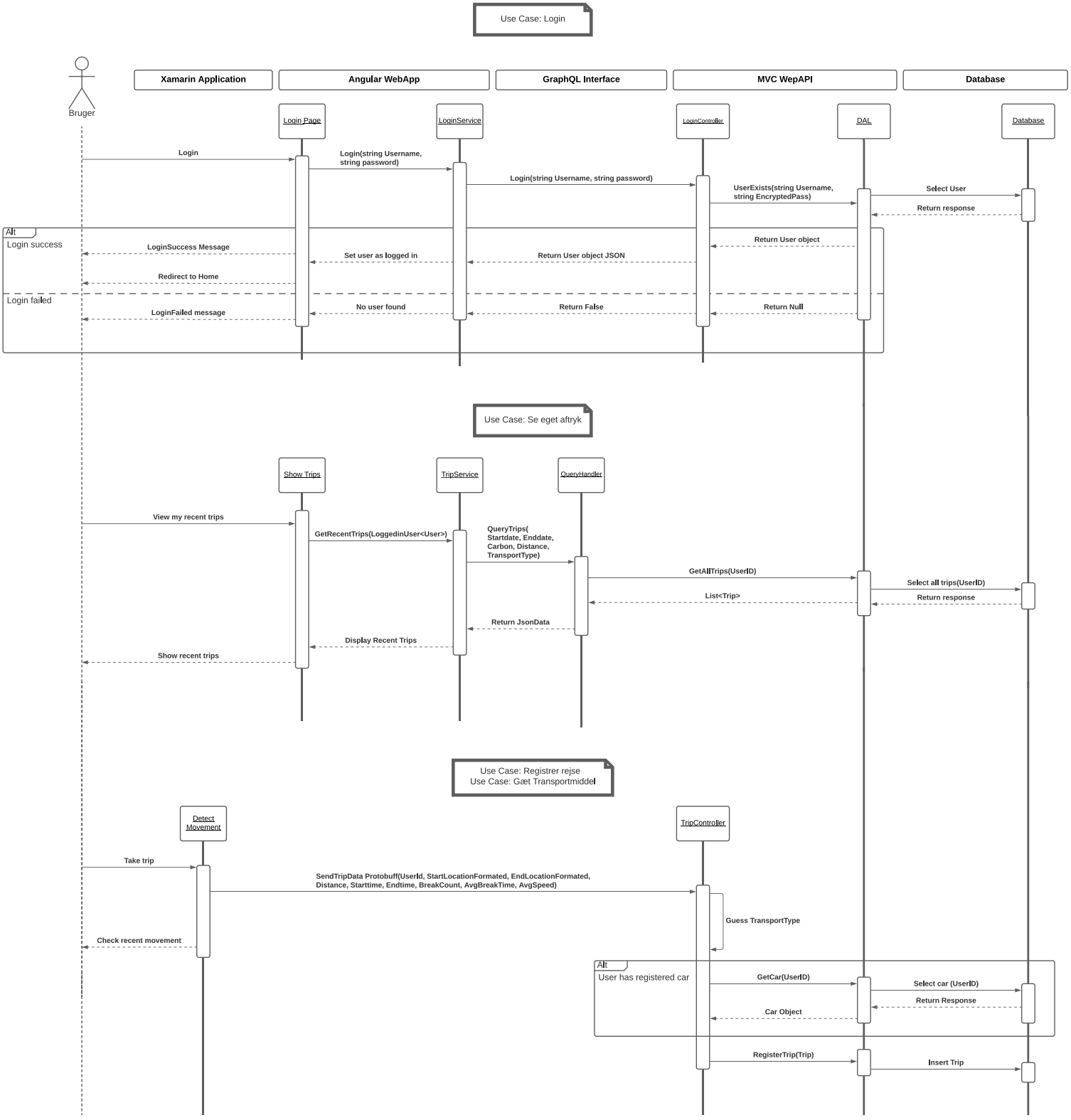
Som en del af analyseprocessen, udarbejdede vi diverse Wireframes for GUI delen. Dette blev gjort så vi kunne visualisere hvordan vi forventede slutproduktet skulle se ud, samt udtænke funktionalitet. Wireframes blev lavet som en mockup, med midlertidige visualiseringer som ville blive skiftet ud i det endelige produkt. Farvevalg, fontvalg, og dimensioner er repræsentative af det endelige design ønske.



6.4 System Sekvens Diagram

Vi har designet et System sekvens diagram, som skal bruges til at danne overblik over de forskellige kommunikationer imellem sub-systemer. For bedre at kunne visualisere de forskellige sub-systemer, er kommunikationen blevet indelt i faste arealer imellem. Nogle Use Cases gør dog ikke brug af alle sub-systemer. I disse tilfælde vil der ikke figurere et activation "stop" ved det ikke-berørte sub-system.

Da et SSD-diagram hurtigt kan blive meget stort, har vi valgt at indsnævre dette til de mest interessante Use Cases. De valgte Use Cases vil også være dem der har mest kommunikation på tværs af sub-systemer.



Figur 3: SSD-diagram over systemets samarbejde.

7. Design

Vi har valgt at beskrive design processen i to dele. I den første del begrunder vi for vores arkitektur valg. Herefter beskriver vi vores implementerede designmønstre, samt hvilke tanker der danner baggrund for valget heraf.

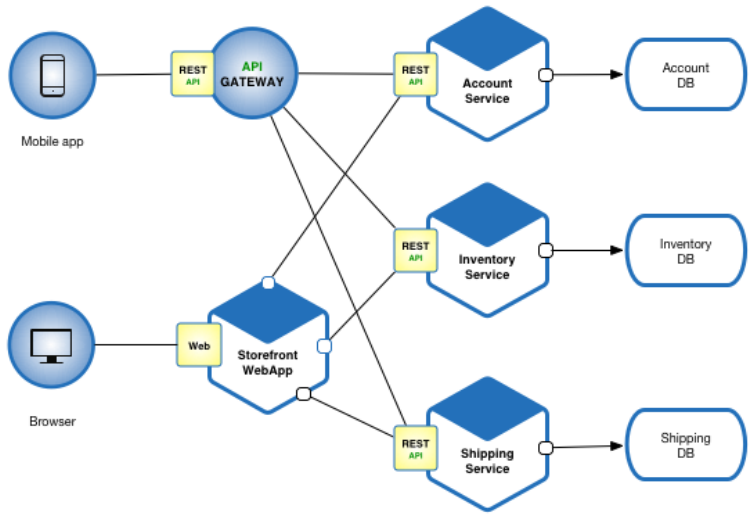
7.1 Arkitektur

7.1.1 Arkitektur valg

Vi har valgt at gøre brug af en Micro services arkitektur som kort fortalt er i stedet for et stort program er nærmer en masse små programmer som kan snakke samme via en Api og i vores tilfælde via GrahQL,

Micro services struktur er en masse små services som udgør selve applikation.

Som kan ses i (nedenstående) figur, kan man skifte "Account services" ud, uden at ændre/have en indflydelse på "Shipping services" eller "Inventory service"



Hvorfor Micro services arkitektur frem for monolit?

Hvorfor har vi valgt at lave det som Micro services i stedet for den mere konventionelle en stor applikation form, det ville nok have været nemmer, men det giver bedre mening at gøre brug af Micro services i projekt co2oparaiton da den struktur både hjælpe os fokusere på de individuelle "Applikationer" samt vi mener at det også giver meningen at kunne udskifte kun en lille del af systemet i stedet for at ænder dele af en monolit applikation.

7.2 Designmønstre

7.2.1 MVC

Et af de designmønstre vi har valgt at gøre brug af i dette projekt, har været MVC-arkitekturen. Model-View-Controller arkitekturen er tæt på en personificering af 3-lags-modellen, hvor Modellen tager rollen som Data-lag, Controller som Logik-lag og View som Præsentations-lag.

I dette projekt bruges MVC designet primært i C# backend og API programmet. Denne beslutning er taget, efter programmet fungerer som det primære bindeled mellem databasen og de resterende sub-systemer. Dette betyder at diverse Views, i form af API endpoints, skal stilles til rådighed. Samtidig med at data skal hentes fra databasen til en Model, hvorefter der skal påføres logik i Controlleren før data bliver sendt videre til Viewet. MVC-mønstret betyder at vi har kunne holde en striks lag-opdeling, seperations-of-concerns, og har haft nemmere ved at lave eventuelle ændringer til funktionaliteten, uden at dette har haft effekt på andre dele af sub-systemet.

Dette har f.eks. kunne være en ændring af hvilke værdier der skulle stilles til rådighed i et API endpoint. Ved at bruge et MVC-mønster, har vi i visse tilfælde kunne opnå dette ved blot at ændre i Viewet, uden at behøve at tilføje ændringer til Model eller Controller.

7.2.2 Dependency Injection

Designmønstret Dependency Injection bliver brugt for at undgå høje koblinger imellem klasser. Dette betyder at en høj niveau klasse bliver injected som et parameter, direkte ind i konstruktøren på en lavere niveau klasse. Dette gøres primært ved hjælp af et interface bindeled, frem for klassen direkte. Derved overholdes SOLID reglen for Dependency Inversion, hvormed man nemmere vil kunne udskifte de relevante høj niveau klasser, så længe at de implementerer det samme interface.

Dependency Injection mønsteret bliver anvendt meget i Angular applikation, hvor alle services bliver injected, eller får dependencies injected i deres konstruktør. Mange af de mere brugte services bliver desuden kørt som singletons.

Set i bagspejlet, burde der have været flere interfaces pålagt, end der er i nuværende iterations. Meget af de større gevinster ved at bruge dependency injectons, bliver desværre ikke opnået når et interface bindeled ikke bliver brugt.

7.2.3 Service design pattern

Repository pattern er et design mønster til at mediere data mellem domæne laget og data access laget (DAL). Herved fungerer et repository som en 'in-memory' data collection, som har enkapsuleret Entity modellerne, med de tilhørende funktioner som bruges til at håndtering af objekterne.

Dette design mønster var et godt valg til at lave et abstraktions lag mellem væres Entity Framework ORM, og graphQL projektet for at holde en lav kobling mellem lagene.

Vi benyttede en struktur med et generisk IRepository<T> som indeholder metode struktur til de basale ORM CRUD funktioner, herved kunne man også i fremtiden udskifte den underliggende ORM, uden at skulle lave ændringer flere steder.

Selve implementationen af funktionerne fra IRepository sker gennem en abstrakt EFRepository class som arver fra interfacet, derved skal koden til funktionerne kun skrives den ene gang.

8. Teknologivalg

Vi vil i dette afsnit beskrive hvorfor vi har valgt diverse frameworks, brugt til projektet. Dette vil primært være en overordnet beskrivelse, samt være fyldt med personlig mening.

8.1 Angular

Vi har valgt at gøre brug af Angular til vores SPA. Dette er primært blevet gjort, på baggrund af at vi allerede havde et kendskab til dette framework.

Med det sagt, så opfylder Angular også mange af de krav vi selv havde lagt til et udviklingsframework. Angular egner sig godt til reelle designmønstre som Dependency Injection og Service Design pattern, samtidig med at frameworket giver adgang til komponent funktionalitet og frontend modeller. Dette giver mulighed for hurtig design af en brugervenlige UI, samtidig med at der er god mulighed for kode arkitektur og genbrugelighed.

8.2 .Net MVC WebAPI

En af de primære grunde til at vi valgte at bruge en .Net MVC WebAPI til vores projekt, er at vi alle har vores primære hjemmebane i C# sproget. Et MVC projekt giver desuden mulighed for hurtig opbygning, samt nem videreudvikling.

8.3 GraphQL

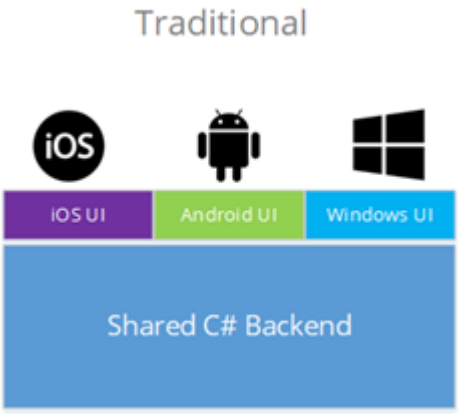
GraphQL er et query language som bruges til at foretage kald mod en API som er opsat efter GraphQL's metode, hvorpå alle den data som skal være queriable, bliver sat op i et graphQL schema, hvorfra man kan foretage queries på lige hvad man har brug for. Denne betyder også at man kun skal opsætte et endpoint som bliver queriet imod.

Men denne fremgangsmåde er meget kode tung i backenden, men når det først er sat op, er der god mulighed for nemt at udvide schemaet med flere datatyper hvis der skulle være behov for dette i fremtiden.

En anden bonus ved GraphQL tilgangen er at det drager nytte af dependency injection og service desigen pattern, hvilket betyder at det fungerer godt sammen med repository pattern som vi har benyttet.

8.4 Xamarin

Xamarin er et open-source framework, som bruger C# som kodebase med Microsoft .NET core.



Xamarin er et udviklingsværktøj for at skrive android, iOS og Windows apps med deres "natilve user interface", og kan dele kode over flere forskellige platforme, som inkluderer Windows, macOS og Linux.

Xamarin.Android: Tidligere kendt som Mono for Android, bruger Mono runtime, androids UI designer, biblioteker både som .NET core og biblioteker som binder til native Andorid/Javas API'er.

Xamarin.iOS: Tidligere blev kaldt MonoTouch, tillader udviklere at lave c# og .net baseret apps som kan køre på apple produkter, I forhold til Mono(Xamarin.Android) bliver dette oversat ned til maskine kode for det målrettet mod iPhone og iPad, dette bliver gjort fordi iOS' kerne tillader ikke just-in-time compilers.

Xamarin.Forms : Gør mere for at lave det crossplatform, så i stedet for at kun dele backend, får de også en fælles frontend kode, ved at gøre brug af XAML, men giver også mulighed for nemt at integrere native platform features så som iOS safe Area og android elevation. Den giver også mulighed for at kode med Model-View-ViewModel(MVVM) design pattern.

8.5 Protobuf

Vi har valgt at bruge Protobuf til kommunikationen mellem Xamarin applikationen og MVC API'en, da det gjorde det nemmere og var hurtigere at implementere end at lave et nyt JSON-objekt med alle de individuelle properties der skulle bruges, samt da vi ikke skulle ændre i datastrukturen.

Da Protobuf også laver en form for kontrakt, som er med til at sikre dataintegriteten, og da det hele bliver konverteret til binære data, hvilken både gør datapakken mindre og som en bonus bidrager det også til sikkerheden, da man skal kende datastrukturen til at kunne læse daten.

8.6 Entity Framework

Vi har valgt at bruge Entity Framework, da dette var den ORM vi alle havde mest kendskab til. Entity framework giver dog også adgang til hurtig udvikling, da frameworket tilbyder standard CRUD-operationer "ud a boksen". Dette viste sig at være en god fordel i dette projekt, da der ikke er blevet gjort brug af komplicerede SQL statements, hvor Entity Framework til viser svagheder.

Vi har dog valgt at gå med en Database-first strategi, hvilket betyder at vi designede databasen uden et større hensyn til Entity Framework. Dette har betydet at vi har kunne arbejde med en højere normaliseret database, end man normalt har mulighed for i en Code-first EF-løsning.

9. Beskrivelse af elementer fra produktrapport

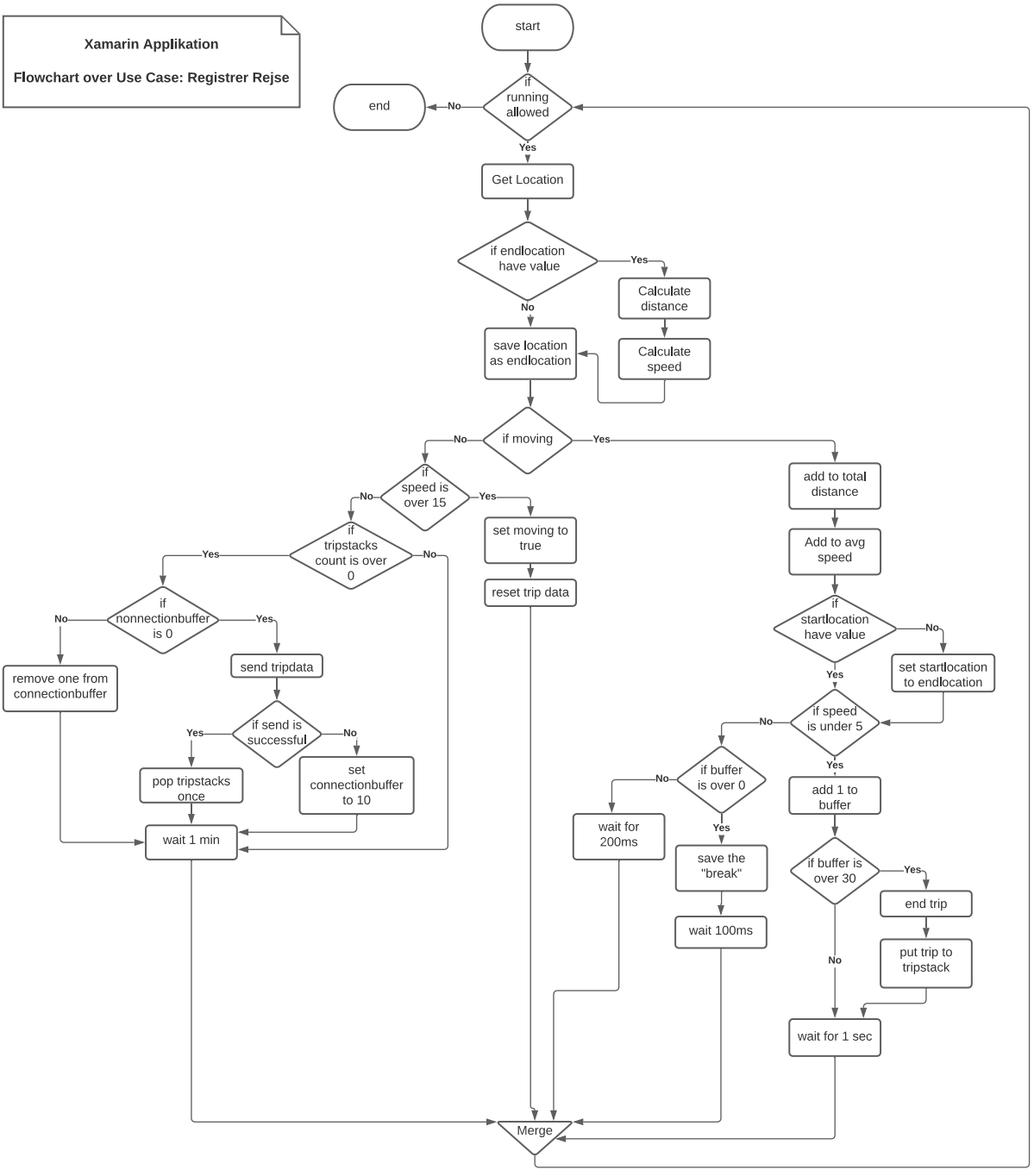
Her har vi valgt at beskrive dele at projektet, som ellers normalt ville tilhøre produktrapporten. Dette er gjort for at sætte fokus på dele af produktet, vi mener kunne kræve ekstra forklaringer.

Specifikt har vi valgt at sætte fokus på et flowchart fra Xamarin applikationen, over Use Case: Registrer rejse. Hvor vi dokumenterer det proces-flow der bliver kørt igennem for at brugerens rejse bliver registreret.

Desuden sætter vi fokus på GraphQL arkitekturen, hvori vi forklarer hvordan vi har implementeret de enkelte dele. Samt hvorfor dette har været en optimal rute for vores projekt. Dette vil være en mere kodebaseret forklaring ud fra klassediagrammet.

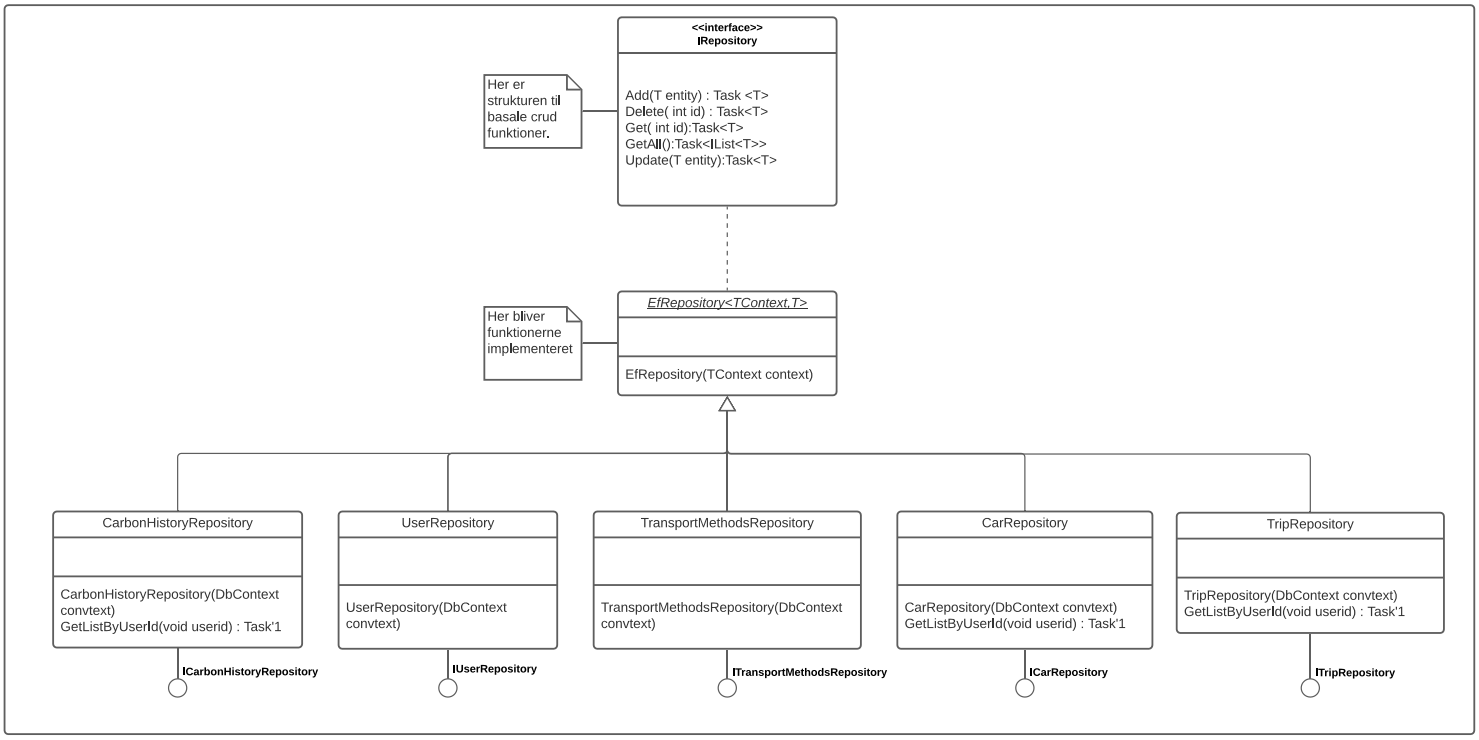
9.1 Xamarin Applikation & Flowchart

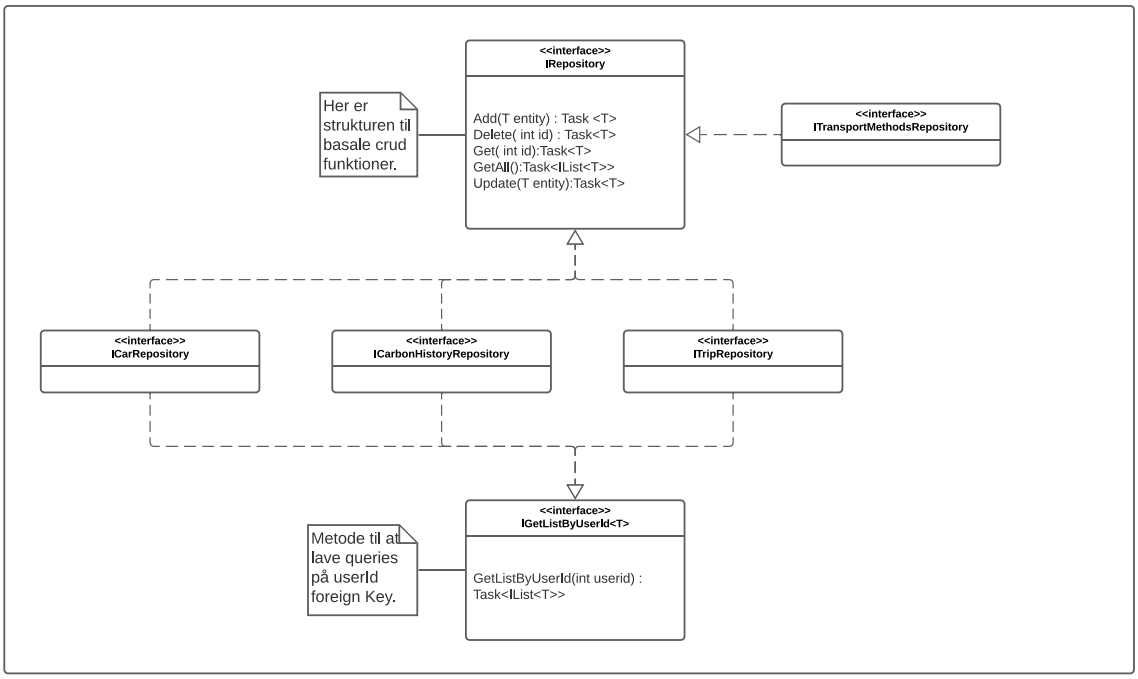
Vi har valgt at lave et flowchart over hvad der sker når Xamarin applikation skal måle ruter. For at starte målingen skal man som bruger være logget ind i appen og tillade "Enable Tracking". Derefter vil den køre i baggrunden selv om applikationen ikke er aktiv.



9.2 GraphQL Interface

Vi visualiserer her opbygningen af vores GraphQL backend projekt. I dette system har vi implementeret Service Design Pattern i forbindelse med Repository Pattern, samt gør brug af Generics. Vi har opnået en høj skalærbarhed, som vi gerne vil fremvise.





10. Logbog

10.1 Fælles daglige morgenmøder

Vi har valgt at føre samlet logbog ved hvert morgenmøde. Her beskriver vi hvad vi forventer at lave i løbet af dagen, samt om der er noget vi skal være opmærksomme på. Det er også her vi ville gennemgå eventuelt kodefremvisning, så alle deltagere ville være up-to-date.

Daglige standup møder varer normalt kun 5-10 minutter og vi har formødet at holde det de fleste dage. I nogle tilfælde har vi dog været nødsaget til at skubbe mødet til senere på dagen, af diverse årsager. I disse tilfælde har vi stadig dokumenteret hvad arbejdsdagen er blev brugt på indtil da.

Perioden hvor vi har holdt morgenmøder, har løbet gennem hele projektperioden. Dog med undtagelse af de første 2 dages idégenererings tid, som blev holdt et par uge før selve projektperioden.

Der blev desuden holdt et projektstart møde i begyndelsen af forløbet, her dokumenterede vi hvad vi forventede af projektet, samt hvad vores ideer var for gennemførslen.

Dette fulgte vi op på ved andet iterations møde, omkring midtvejs i projektet, hvor vi holdt et længere møde omkring den nuværende status for diverse sub-systemer og Epics. Her dokumenterede vi også den nuværende Domæne model og ER-diagram, i tilfælde af at der kunne opstå ændringer undervejs.

Projektstarts-møde er blevet afholdt d. 14/09-2020.

Midtvejs-møde er blevet afholdt d. 28/09-2020

14/09-2020

Vi har brugt dagen på projektstartsmøde. Vi har desuden lavet udkast til FURPS, samt påbegyndt beskrivelse test/krav scenarier.

14/09-2020 – Projektstarts-møde

Intro:

Dette møde blev afholdt som noget af det første i projektforløbet. Her snakkede vi om hvordan vi forventede at skulle udvikle projektet, diskuterede hvilke teknologier og systemer vi ville bruge, samt den praktiske udførelse af projektet.

Vi diskuterede også hvordan vi ville planlægge og dokumentere projektet løbende.

Beslutninger der blev dokumenteret i dette møde, vil ikke nødvendigvis holde hele vejen igennem, og kan til en hver tid ændres efter behov.

Møde noter:

WEB API:
- MVC .NET
  - REST API
- IIS til hosting
- MSSQL Database (spatiel data) - Køres på IIS server
  - Bruger
  - Statistik
- Google Protobuff (afhængig af komplikationer under implementering)

Angular:
- Data:
  - Hastighed
    - Bruges til at vurdere køretøj.
  - Placering
  - "rutine route"
  - Intelligent løsning der skal opdage adfærdsmønstre.
    - OpenStreetMap API/local opstillet OpenStreetMap database
    - Alternativ løsning med bruger input.
- Brugerens oplevelse:
  - Brugeren vil skulle oprette en profil for at bruge appen.
  - Brugeren vil skulle udfylde informationer.
  - Nummerplade skal kunne finde bilmodel. Til udregning af CO2 Aftryk.
  - Evt. Nummerplade API (https://www.nummerpladeapi.dk/)
  - Sammenligning med andres aftryk
  - Skal vise brugerens totale CO2 aftryk.
  - Eventuel med statistik versus andre brugere (Landet, regionen).
- Samlet statistik over alle brugere / Opdelt i mindre dele (f.eks. Regioner).
- Forslag til at nedsætte co2/betale noget af brugerens co2 aftryk.
- Integrering med GraphQL (ApolloGRAPH/ gRPC and Graphql)
- Google Protobuf (afhængig af komplikationer under implementering)

Android app:
- Xamarin(evt. Hardcode til en browser).
- Potentielt vise push-beskeder ved x adfærd.
- Kun skal vise Angular side i browser.

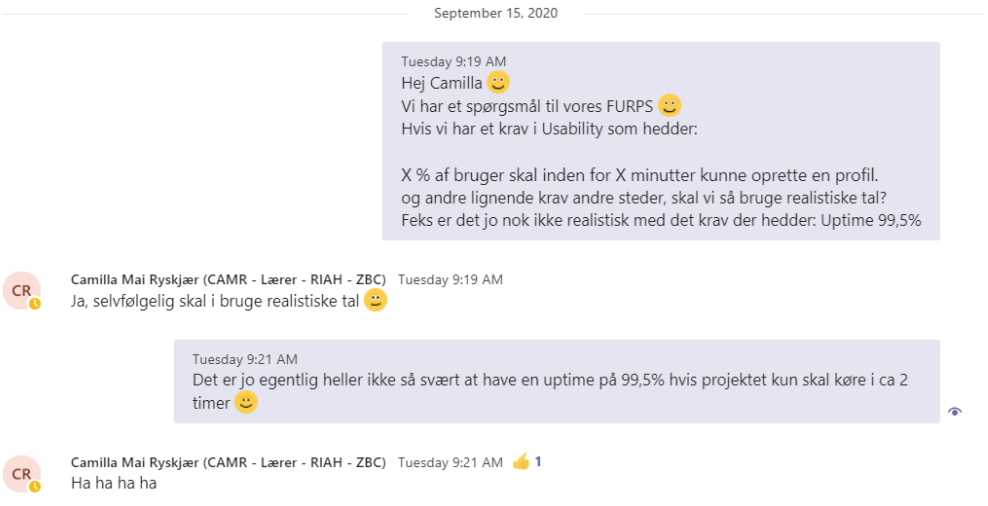
Hardware:
- Panorama til fremvisning (evt. Bare modelbiler til at vise hvordan vi måler dataene. Og evt. fremtidige muligheder. Eg. Der er mulighed for at tilføje denne feature/del her for at kunne gøre x)

Process:
- Daglig Logbog (Personlig).
- Møde hver morgen. Samlet logbog (Referat).
- Midtvejsmøde - Inkluder domænemodel.

15/09-2020

Der skal laves FURPS+ krav i dag. Vi sætter os og arbejder samlet med at finde krav og test scenarier. Forventningen er at have et rough draft klar ved enden af dagen.

Vejledning fra Camilla:



16/09-2020

VI fik i går lavet et draft til FURPS+. Funktionelle og ikke funktionelle krav er blevet fastsat og opdelt.

Vi skal i dag have lavet FURPS+, Use Case, samt Wireframes. Vi satser på at have alle tre dele i et semi-færdig stadie, i slutningen af dagen.

17/09-2020

Status fra i går:

- Use Cases er lavet for de fleste funktionelle krav. De sidste Use Cases skal findes og skrives ned i løbet af dagen.
- FURPS+ skal der skrives læse vejledning, indledning, samt opsætning. Største delen af krav er blevet fastlagt, samt test scenarier hvor relevant. Vi regner med at have FURPS+ dokumentet klar når dagen er omme.
- Wireframes er blevet opsat i Moqframes programmet med interaktion. Designet er opsat med placeholders for grafikken.

18/09-2020

Vi har fået færdiggjort FURPS dokumentet i en endelig form, denne er blevet kigget igennem af Sune. Vi har herefter fået diskuteret hvilke task vi skal have lavet, samt i hvilken rækkefølge de skal færdiggøres. I samme omgang blev det først udkast af et Azure DevOps workbord sat op, med de "task" og Use Cases vi havde defineret.

21/09-2020

Dagen starter med at vi bliver advaret ang., covid-19 samt Camila valgte at kaste varm kaffe på et ømt sted, samt derefter fortæller en historie om alderdom. Alt i alt, en god start på dagen.

Planen for i dag er som følger, vi skal blive færdig med opsætning af tasks i Azure DevOps, samt aflevere FURPS til Camilla, hvor efter vi håber at få noget feedback. En estimeret tidsplan skal udearbejdes samt muligvis en risikoanalyse hvis tiden er til det. Første udkast af Wireframes til PC-versionen af hjemmesiden er blevet færdiggjort af Casper & Andi.

22/09-2020

VI regner med at arbejde på Domæne modellen hele dagen i dag. Forventningen er at have diverse entities og tilhørende properties færdigt i slutningen af dagen. En domænemodel vil skulle planlægges færdig før vi kan gå videre med udviklingen. Hvis tiden er til det, vil vi også gerne udvikle et udkast til ER-diagram.

23/09-2020

Vi fik i går fuldendt første iteration af domæne model og ER-diagram. Herefter fordelte vi individuelle opgaver.

Fordelingen af overordnet ansvar blev som følger:

- **MVC-Backend**: Andi
- **Xamarin applikation**: Casper
- **Angular SPA**: Benjamin
- **GraphQL**: Jonas

Dagen i dag kommer til at starte med undervisning i deployment-diagram fra Camilla. Efterfuldt af design af vores egen version. Herefter vil vi fortsætte med de individuelt fordelte opgaver, som defineret på workboardet.

24/09-2020

Der blev i dag arbejdet videre med de igangværende tasks. Xamarin applikation er klar til del-test, Server infrastrukturen giver problemer med opsætningen af databasen, API'en er blevet videreudviklet, GraphQL er der blevet arbejdet på test projekt, Angular projektet har fået færdiggjort theme og frontend styling.

25/09-2020

I dag forventer vi at arbejde videre på de individuelle opgaver.

- Xamarin applikation: skal testes. Her er det primært hastighedsberegningen der er klar til test.
- Angular webapp: skal videreudvikles, funktionalitet og UX skal videreudvikles. Domæne model skal udvikles hertil.
- Server: Webdeploy giver stadig problemer. Dette skal der findes en løsning til.
- GrahpQL: Arbejde videre på test projekt, muligvis direkte overførsel.

28/09-2020

I weekenden er der blevet arbejdet på GraphQL interface, samt arbejdet videre på services til angular appen opsat med test-api, da vores egen API ikke er testklar. I dag forventer vi at holde iterationsmøde/midtvejs-møde, hvilket vil inkludere fremvisning for holdet og kode gennemgang.

Funktionalitet skal udvikles til at gætte et færdigt rute object properties, inklusive køretøj og CO2 aftryk. Data til udregning bliver sendt fra Xamarin appen, men skal udregnes i backenden.

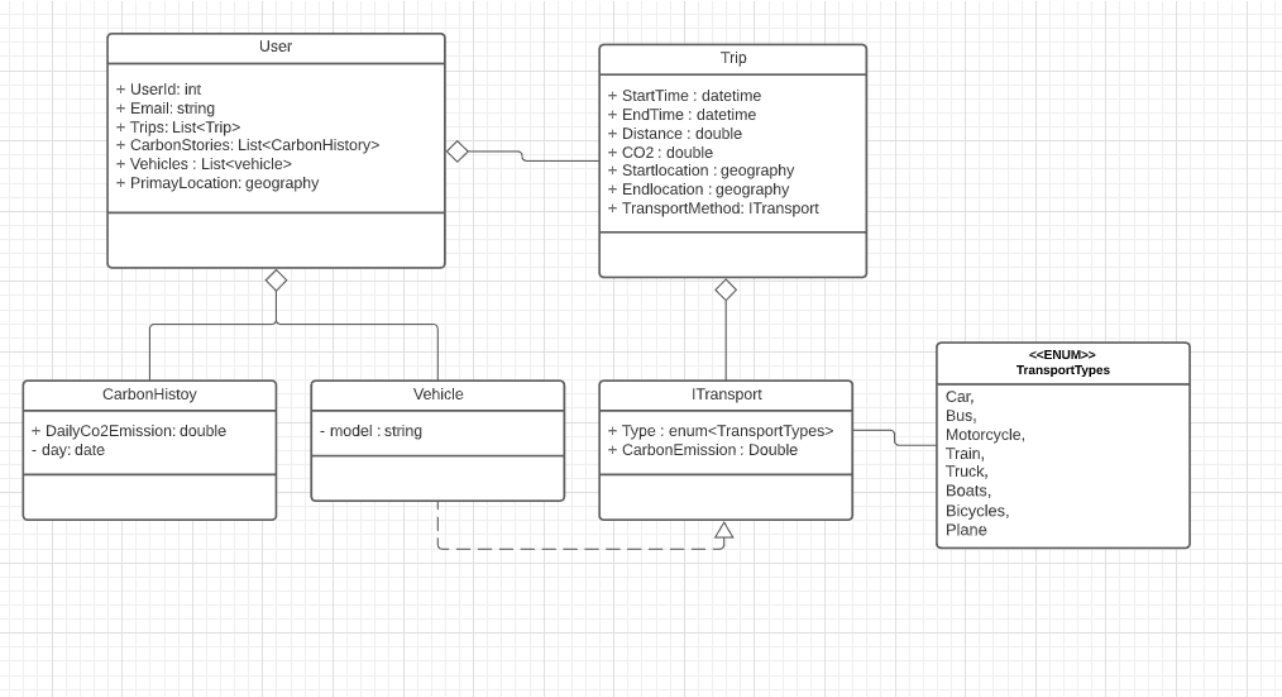
28/09-2020 – Midtvejs-møde

Intro:

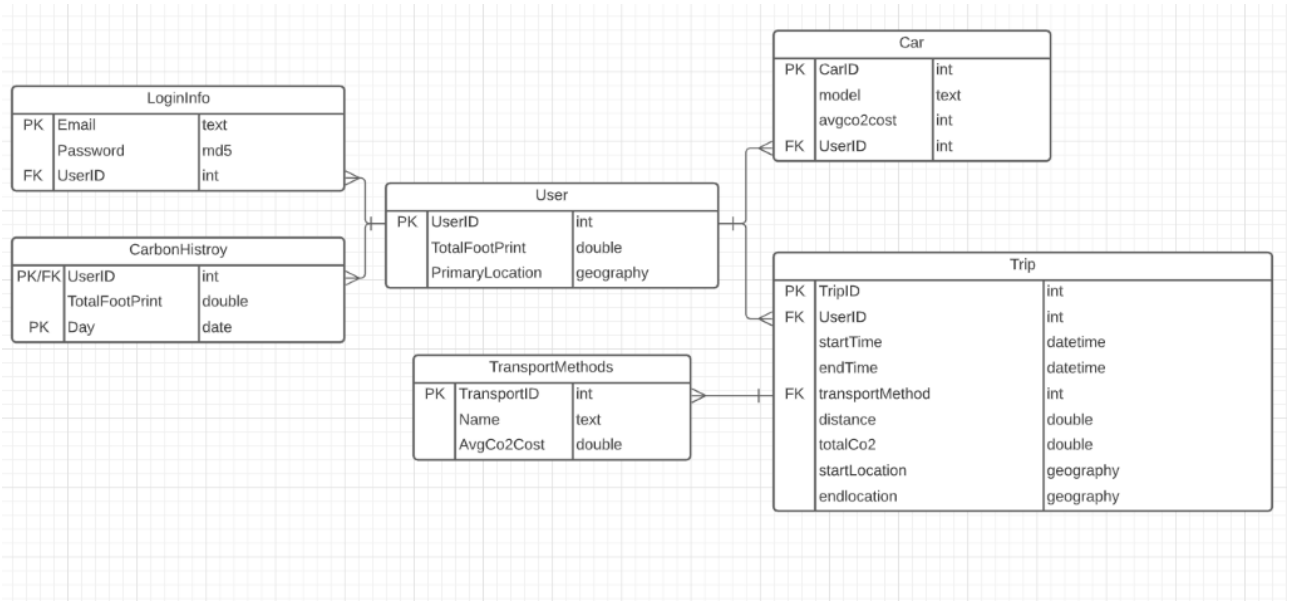
Dette møde blev afholdt omkring midtvejs igennem projektet. Her fremviste vi de nuværende fremskridt i de enkelte sub-systemer. Dette inkluderede bl.a. kode fremvisning for de resterende projektmedlemmer. Vi dokumenterede desuden hvor langt vi var kommet på nuværende tidspunkt, samt om der havde været eventuelle ændringer siden start, som var for komplekse til at diskutere på de daglige morgenmøder.

Møde noter:

Domcæne model:



ER-diagram



Backend/database:

Blevet opsat med entity framework, med CRUD. Test af password hash er lavet på backend, dette vil skulle laves i front-end angular applikationen i det endelige produkt.

Api kald er [/co2/["Insert action here"]]

(note) backend som det står nu hasher ind kommene password strings.

Her skal der laves endpoint til login. Dette er en ændring fra den originale plan, hvor login skulle laves igennem GraphQL.

Forbindelse imellem databasen og Backend MVC API'en er nu opsat og fungerer efter hensigten.

Nye endpoints skal laves i backenden. Dette skal laves til modtagelse af trip data fra Xamarin appen, login fra Xamarin/Web-app, samt registrering af bruger fra web-app.

Xamarin app:

Funktionaliteten med at optage bevægelse fungerer, samt videresendelsen af data er klar. Der mangler stadig endpoints i backenden, samt håndtering deraf.

Applikationen viser angular web-appen. Denne funktionalitet fungerer korrekt nu.

Background service fungerer nu som den skal.

Her mangler noget funktionalitet til at huske brugeren i applikationen. Dette kan resulterer i at brugen vil skulle logge ind ved brug, via appen, frem for web-appen.

Der opfanges bevægelse i baggrunden, som bliver sendt til api'en ved fuldendt rejse.

Der skal muligvis sendes data med i api kaldet, som omhandler antallet og varigheden af stop på rejsen. Dette skal resultere i bedre præcision i tilfælde af offentlig trafik.

loop pause:
Når turmålingen ikke er startede er det 1 min.
Når turmålingen køre er det 200ms
Når turmålingen er startede men den holder stille (+-1km/h) 1sek, første gang den måler en bevægelse 100ms derefter tilbage til 200ms

GraphQL:

Dette giver desværre flere problemer end først beregnet. Vi bliver derfor nødsaget til at begrænse brugen af teknologien, til mere relevante funktioner.

Dette betyder at vi ikke kan bruge GrapQL til login og registrer funktionerne, men godt vil kunne køre med GrapQL på statistik og trip-visnings siderne.

Angular applikation:

Angular applikation er tæt på færdiggjort. Der mangler dog stadig at sætte kommunikation op imellem angular appen og backend Web Api'en, dette betyder at login og bruger oprettelses funktionalitet ikke fungerer i praksis, men kun i teori med en test api.

Desuden er der stadig et hængende problem med NummerpladeAPI, som ikke har svaret tilbage på vores forespørgsel om adgang. Dette betyder at vi muligvis vil være nødsaget til at bruge en falsk nummerplade opslags api, eventuelt en som vi selv udvikler. Dette vil give os mulighed for at "fake" et nummerpladeopslag, som svarer tilbage med de relevante informationer.

Statistik siden er ikke blevet udviklet endnu. Da der skal være mulighed for at andre også kan lave frontend udvikling.

29/09-2020

Vi regner med at skulle arbejde videre GraphQL implementering i angular projekt, samt få GraphQL interfacet til at snakke med Backend.

Vi skal have lavet API endpoints til bruger kontrol, login og bruger oprettelse. Dette skal laves i backend samt gives adgang via API endpoints. Herefter skal endpoint findes via Angular appen.

I Xamarin appen skal der laves fejl info ved manglende internetforbindelse. Login til at hente userID, denne funktionalitet skal bruges til at måle pauser i en Trip.

30/09-2020

GraphQL burde være klar til integration i dag. Vi forventer at integrationen af GraphQL interfacet vil være klar inden dagen er omme. I Xamarin mangler vi stadig user ID, vi forventer at sætte brugerID på som url parameter fra angular appen, som så vil kunne hentes i Xamarin appen.

Vi har stadig problemer med CORS blokering mellem Angular, Xamarin og backend MVC api projektet. Vi vælger derfor at ændre login og registrer funktionerne på MVC WbApi projektet, til at tage imod GET frem for POST. Dette vil vi eventuelt ændre til en mere korrekt opsætning senere, efter at vi er kommet videre med andre dele af projektet. Vi vælger at prioritere at få prototypen i en funktionel tilstand, frem for sikkerheden. Dette retfærdiggøre ved at der er tale om en kontrolleret prototype.

01/10-2020

I dag blev der arbejdet videre på GraphQL. Integreringen giver stadig problemer, men der er lys forude. Angular applikationen fik integreret nummerplade lookup, hvor igennem vi kan finde CO2 aftryks og bil informationer. Denne funktionalitet har ellers ventet på sig, grundet manglende adgang til NummerpladeAPI. Dette blev fikset ved at finde en alternativ provider, og lave funktionaliteten igennem den.

Endpoints i backenden er klar til login, signup og trip registrerings Use Cases. Xamarin applikation kan ligeså skrive trips til backend API'en, som så registrerer disse. Forbindelsen mellem Xamarin applikationen og backend MVC apien gøres over protobuff, frem for json.

Der er desuden blevet arbejdet videre på procesrapport, hvori der er blevet skrevet indledning, problemstilling, samt metodevalg.

02/10-2020

Dagen blev brugt på at arbejde videre med GraphQL. Integration med Backend projektet er næsten klart, hvilket betyder at der kun mangler sammenkobling med Angular applikationen nu.

Der opstod et problem med Xamarin applikation, hvilket syntes at stamme fra CORS opsætningen i MVC API'en. CORS blev opsat korrekt på MVC API, hvorefter at Xamarin applikation fungerede korrekt igen. Hvorfor problemet opstod ud af det blå, efter at have fungeret de sidste par dage, vides ikke.

Der blev arbejdet videre med dokumentation i form af procesrapport og produktrapport. Det virker til at vi er nødsaget til at arbejde i weekenden, hvilket har betydet at vi brugte den sidste del af fredagen på at fordele opgaver.

05/10-2020 – Sidste arbejdsdag i projektperioden

I weekenden er der blevet arbejdet på følgende:

GraphQL: Flere test projekter blev lavet, desværre uden held. Den endelige fik vi heldigvis gevinst med. Dokumentation vedrørende GraphQL, blev desværre ikke færdiggjort. Den endelige løsning endte med at inkludere en sekundær backend, skrevet på .Net Core.

Xamarin: Der er blevet skrevet dokumentation på Xamarin. Desuden er der blevet skrevet hvorfor Xamarin blev valgt som teknologi, til procesrapporten.

Desuden er der blevet skrevet den resterende dokumentation i weekenden. Dette inkluderer Rich Picture, SSD, design mønstre - dependency injection, skrevet info om værktøjer, lavet klassediagramer, skrevet inro til diverse emner. Desuden er der blevet lavet bilag til log og møder. Alt dette er herefter blevet tilføjet til en samlet Procesrapport

Produktrapport, 80% færdig. Denne vil skulle sammenscættes i dag.

Vi regner med at arbejde med følgende i dag:

Vi skal have lavet proces- og produkt- rapporter færdige.

Regner med at skrive om Xamarin. Skrive ekstra tekst til flowchart.

GraphQL skal integreres i både Backend og Angular.

Der er problemer med at integrere GraphQL i Angular projektet, dette betyder at vi skal bruge en del af dagen i dag på at få fikset denne integrering.

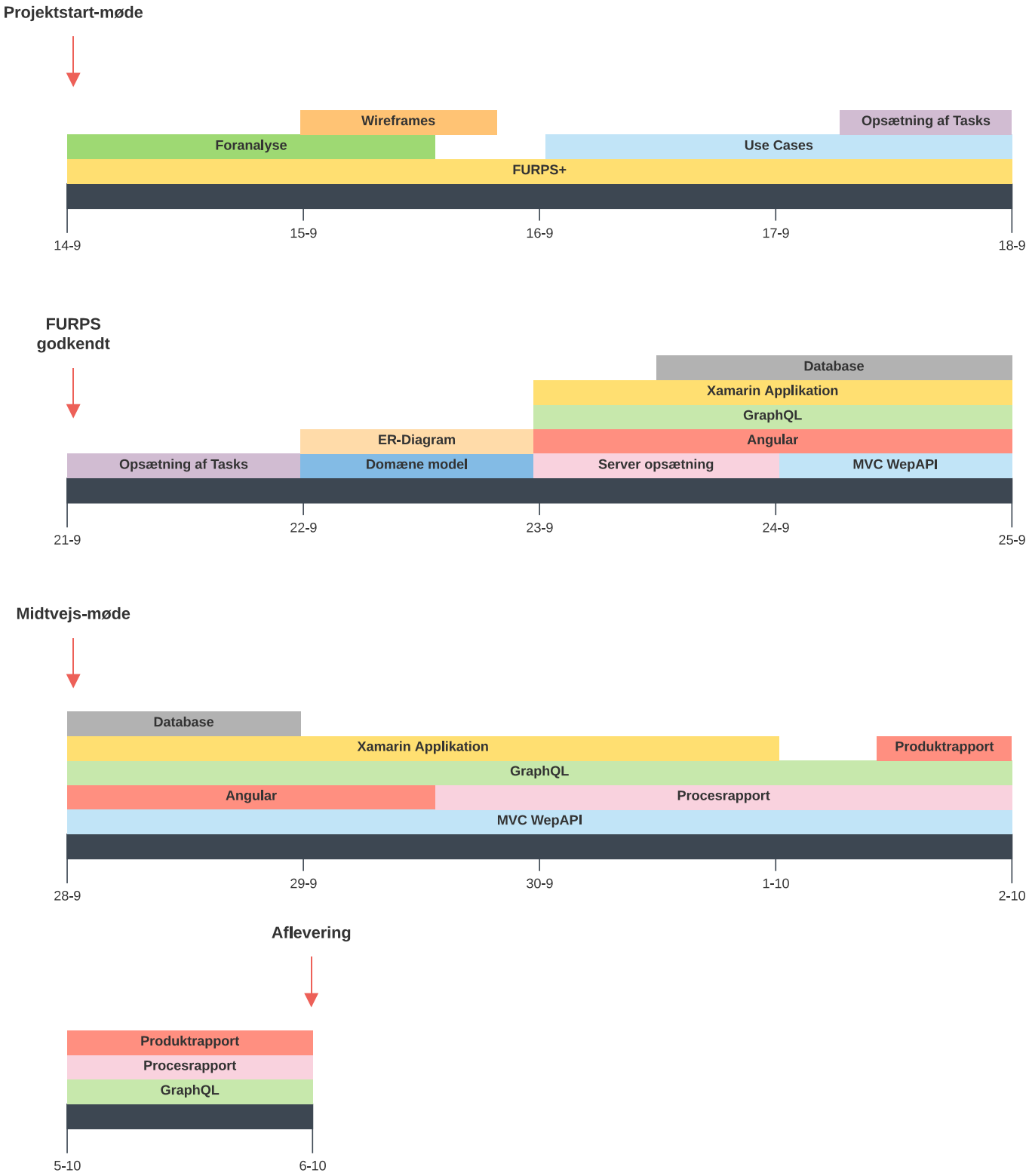
10.2 Daglig logbog - Individuel

Desuden har vi kørt en personlig logbog for hver enkelt deltager. Denne bliver lavet når arbejdsdagen var over. Heri bliver der beskrevet hvad den enkelte har arbejdet med i løbet af dagen.

De personlige logbøger kan findes i Bilag 2: **Logbog – Individuelt.PDF**

11. Realiseret tidsplan

Den endelige arbejdstid der reelt blev brugt på de forskellige opgaver vises nedenfor. Start og slutdatoer for opgaver, er selvfølgelig altid flydende. Specielt når der er tale om mange sammen bundende systemer. Vi har derfor valgt at bestemme slutdato ud fra hvornår udvikleren ikke længere brugte sin fulde tid på opgaven. Selvfølgelig har mange af opgaverne involveret mere end en udvikler, hvilket er forklaringen på at der ikke alle steder er 4 igangværende opgaver.



12. Konklusion

Hvordan laver vi en teknisk løsning der kan hjælpe med at reducere brugerens CO2 aftryk?

Vi mener at have fremvist hvordan vi med en teknisk løsning kan informere en bruger om hvilken effekt de personligt har på miljøet. Vi har valgt at lægge vægt på den tekniske del af problemformulering og derved fremvist hvilke arkitekturer, samt diverse tekniske løsninger, der kan benyttes ved målet om at lave et system så skalerbart og udvidelsesvenligt, som muligt. Da projektet i sin helhed er set igennem tekniske briller, konkluderer vi at dette har været den bedste fremgangsmåde.

Informationsformidling efter brugerens eget behag, mener vi stadig er en bedre løsning end at tvinge informationen ned over dem. Dette vil man kunne opnå med en løsning, der i størstedelen af tiden samler information i baggrunden. For derved at forstyrre en bruger mindst muligt. Valget bag at fokusere på den skjulte pris bag transport brug, er blevet taget på denne baggrund.

I mange situationer, bliver en normal borgers beslutninger taget ud fra uvidenhed, samt en frygt for forandring i hverdagen. Ideen bag at fokusere på de små forandringer, man som normal person kan lave i sin hverdag, mener vi derfor har den største chance for rent faktisk at have en effekt.

Teorien bag, at det er nemmere at bede en person om at tage bus, cykel, eller tog. Fremfor at ændre grundsten i deres hverdag. Er derfor vores konklusion, samt hele ideen bag vores forslåede løsning.

13. Litteraturliste

(European Commission 2008, s 18). (u.d.). Hentet fra European Commission 2008, Special Eurobarometer 300, Europeans' attitudes towards climate change. Lokaliseret d. 27.08.20:

Ivanova D, et al 2017, Mapping the carbon footprint of EU regions, IOP Publishing. Lokaliseret d. 27.08.20:. (u.d.). Hentet fra Ivanova D, et al 2017, Mapping the carbon footprint of EU regions, IOP Publishing. Lokaliseret d. 27.08.20: