Aarhus Universitet
Semesterprojekt 4
Gruppe 5

Rapport

CarnGo

Gruppemedlemmer:
Edward Hestnes Brunton
(201705579)
Marcus Gasberg
(201709164)
Martin Gildberg Jespersen
(201706221)
Mathias Magnild Hansen
(201404884)
Tristan Moeller
(201706862)
Erik Mowinckel
(20107667)
Hamza Ben Abdallah
(201609060)

Antal tegn:
73.370 (med mellemrum)
Microsoft Word konvertering fra PDF

Vejleder:
Jesper Michael Kristensen, Lektor

29. maj 2019

1 Versionshistorik

Ver.	Dato	Initialer	Beskrivelse
1.0	19-04-2019	TM	Opsætning af rapportdokument
1.1	28-04-2019	TM	Tilføjelse af afgrænsning og analyse
1.2	11-05-2019	TM	Tilføjelse af metode- og proces afsnit
1.2	19-05-2019	TM	Tilføjelse af arkitekturafsnit
1.3	22-05-2019	TM	Tilføjelse af softwaredesign afsnit
1.4	23-05-2019	TM, MH	Tilføjelse af accepttest afsnit
1.5	23-05-2019	MG	Opdatering af arkitektur views
1.6	23-05-2019	MGJ, HBA, MG	Opdatering af applikationsmodeller
1.7	25-05-2019	TM, MGJ, HBA, MG, EB, MH	Gennemretning af rapport
1.8	26-05-2019	TM	Korrekturlæsning
1.9	27-05-2019	TM, EB	Tilføjelse af referencer

2 Ansvarsområder

Ansvarsområde	**HBA**	**TM**	**MGJ**	**MH**	**EB**	**MG**	**EM**
Resume / abstract				P			
Forord				P			
Indledning				P			
Projektformule-ring		P		P		P	
Funktionelle krav	P	P	P	P	P	P	P
Ikke-funktionelle krav	P	S	S	P	P	S	
Afgrænsning		P					
Metode og Proces		P					
Analyse		P					
Software Arkitektur	P	P	P	P	P	P	
Login	P					P	
Sign Up						P	
Car Profile					P		
Messages		P				P	
Notification		P	P			P	
Headerbar						P	
User Profile					P	P	
Database		P	P	P	P	P	P
Send Request		P	P			P	
Search				P			
Unittest	P	P	P	P	P	P	
Integrationstest						P	
Accepttest	P	P		P	P		
Diskussion		S	P				
Konklusion				P			
Fremtidigt arbejde					P		

Tabel 2.1: Ansvarsområder for medlemmer af PRJ3 Gruppe 7, hvor **P** indikerer, at medlemmet har været primæraktør, og **S** indikerer, at medlemmet har været sekundæraktør.

3 Resume

I dette projekt anvendes en Scrum baseret udviklingsproces til at implementere og teste en biludlejningsapplikation. Der er fokuseret på at udarbejde en grafisk brugergrænseflade samt en database, som understøtter udlejningsprocessen. Systemet giver privatpersoner mulighed for at oprette en brugerprofil som lejer, og disse kan registrere sig som udlejere. Udlejere kan registrere biler i systemet, mens lejere kan søge efter tilgængelige biler ud fra udvalgte kriterier. Når man har oprettet en profil som lejer eller udlejer, kan man logge ind i applikationen. Her er det muligt at redigere eller fjerne sin brugerprofil og bilprofiler. Når lejer har valgt en bil, kan der sendes en anmodning om leje til den pågældende udlejer. Udlejer får en notifikation og kan herefter vælge at godkende eller afvise anmodningen. Ved godkendelse er handelen gennemført, og lejer notificeres herom.

Informationer vedrørende brugere og biler gemmes i en relationel database, som hostes af Microsoft Azure. Det er valgt at anvende frameworket WPF til udvikling af applikationen, hvilket begrænser den til at fungere på en PC med Windows OS som den eneste platform. Desuden er det fravalgt at gøre brug af lokalitetstjenester, datakryptering og forsikringer. Store dele af systemet er implementeret og unit testet. Systemet er herefter integrationstestet med udgangspunkt i en Bottom up strategi. Endelig er der udført automatiserede såvel som manuelle accepttests. Næsten alle krav er opfyldt, og prototypen er derfor tæt på at være fuldt funktionsdygtig.

4 Abstract

In this project a Scrum based development process is used to implement and test a car rental application. The focus is on making a graphical user interface and a database that support the car rental process. The system allows individuals to sign up as renters, and these can register as car owners. Car owners can register cars in the system, while renters are able to search for available cars based on some set of chosen criteria. When the user profile is created, the renter or car owner can log into the application. This allows them to edit or remove the user profile or car profiles. Once the renter has decided on a car, it is possible to send a request to the car owner in question. The car owner receives a notification and has to decide whether to accept or reject the request. If the request is accepted the transaction is completed and the renter is notified.

Information regarding users and cars is stored in a relational database, which is hosted by Microsoft Azure. The application is developed in the framework WPF, and since it only supports a single platform, the application is limited to work on a PC with Windows OS. Furthermore, it was decided that no location services, data encryption or ensurance policies were to be used in the project. Large parts of the system have been implemented and unit tested. This was followed by integration tests in which a Bottom up strategy was utilized. Finally, automated as well as manual acceptance tests were performed. Almost all of the requirements are met, so the application is close to being fully functional.

Indhold

5 Forord

Denne rapport er udarbejdet af Projektgruppe 5 på fjerde semester, Aarhus Universitet og omhandler projektet CarnGo. Projektgruppe 5 består af syv studerende fra studieretningen IKT (Informations- og Kommunikationsteknologi). Et af gruppemedlemmerne er fra Norge, hvorfor visse dele af rapporten er skrevet på norsk.

Semesterprojektet er udført i foråret 2019, og denne rapport har til formål at give et indblik i baggrunden for projektet, udviklingsprocessen samt hvordan produktet er designet og fungerer. Hensigten er desuden, at rapporten skal tydeliggøre arbejdet, som er udført i forbindelse med projektet for vejleder og censor, samt udgøre grundlaget for den mundtlige eksamen i semesterprojektet.

Rapporten suppleres med følgende separate bilagsdokumenter: Accepttestspecifikation, Afgrænsning, Analyse, Arkitektur, Implementering, Kravspecifikation, Møder, Ordliste, Proces, Projektformulering, Referencer, Softwaredesign, Testdokument og Wireframe. Desuden er alle elektroniske kilder, der refereres til, vedlagt rapporten som bilag.

Projektgruppen er vejledt af Jesper Kristensen og projektrapporten er afleveret den 27/05/2019. Der skal lyde en tak til Lykke Møller for korrekturlæsning.

6 Indledning

6.1 Motivation og kontekst

Mange mennesker har ikke egen bil af økonomiske eller praktiske årsager og er derfor nødsaget til at anvende offentlige transportmidler eller samkørsel. Der opstår dog ofte situationer, hvor det er fordelagtigt at have en bil til rådighed. Derfor har flere private aktører lanceret applikationer, hvor man som privatperson kan udleje sin bil, eller lease en andens bil i en periode. Et eksempel på et sådant firma er GoMore, der har udviklet en platform, som skal forsimple udlejningsprocessen og formidle kontakt mellem lejer og udlejer. Brugernes reviews indikerer imidlertid, at der er væsentlige problemer forbundet med brugen af platformen.

Der skrives bl.a. på Trustpilot, at GoMore afkræver for høje gebyrer i forbindelse med udleje og forsikring, og at afgifterne stiger med tiden[1][2]. Mange vil derfor ikke længere anvende tjenesten af økonomiske årsager. Visse brugere nævner, at de ikke er trygge ved tjenesten på grund af manglende gennemsigtighed fra GoMore, mens andre giver udtryk for, at der mangler tillid mellem lejer og udlejer.

Det fremgår, at mange af brugerne er utilfredse og leder efter alternativer. Der er således behov for et produkt, som simplificerer processen og giver brugerne mulighed for at leje og udleje biler, uden at de bliver tynget af høje gebyrer og afgifter. Tjenesten skal udgøre en tryg ramme for udlejningsprocessen, så der opstår tillid mellem lejer og udlejer.

6.2 Projektformulering

Dette projektarbejde tager udgangspunkt i projektoplægget til 4. semester IKT på Aarhus Universitet[3]. Der arbejdes derfor efter en iterativ udviklingsproces, hvor semesterets kurser integreres. Formålet er at implementere og teste et software udviklingsprojekt. Der anvendes objektorienteret analyse og design i udviklingen af en applikation, som har en grafisk brugergrænseflade og en database.

Målet for dette projekt er at udvikle en funktionel prototype af en biludlejningsapplikation. Systemet skal give privatpersoner mulighed for at udleje og leje biler, mens udlejningsprocessen simplificeres mest muligt. Både lejere og udlejere skal kunne registrere sig som brugere gennem brugergrænsefladen, og systemet skal gemme informationer vedrørende brugere og biler. Det skal være nemt for lejer at søge efter biler og leje en ønsket bil ud fra nogle kriterier. Det kunne f.eks. være en maksimumspris eller en maksimal afstand til bilen. Applikationen skal kunne tilgås på flere platforme, herunder Web, PC og Smartphone.

6.3 CarnGo

CarnGo skal løse nogle af de problemer, som er forbundet med etablerede biludlejningstjenester såsom GoMore. Applikationen skal give privatpersoner mulighed for at udleje biler i et tidsrum og til en pris, som de selv specificerer. Dermed får lejere mulighed for at vælge mellem tilgængelige biler og leje den bil, som opfylder netop deres behov.

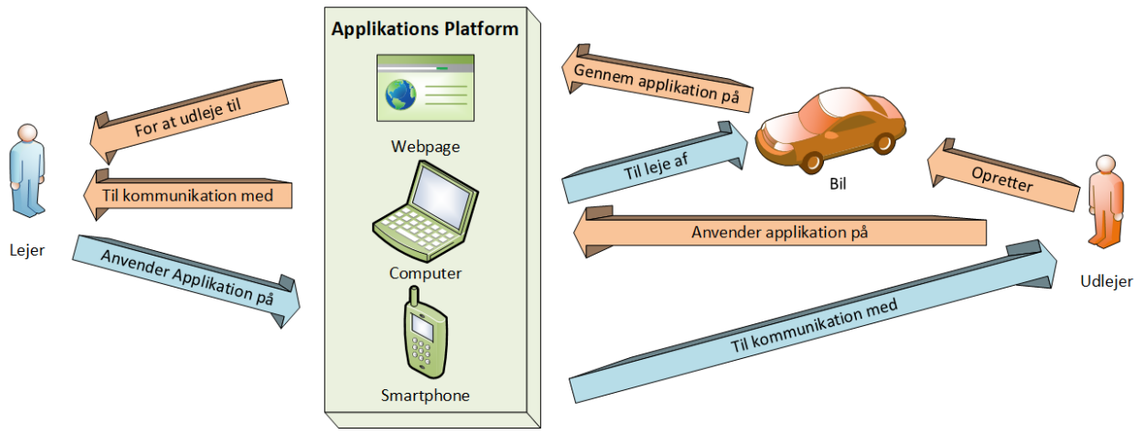
For at CarnGo bliver en brugervenlig applikation, som kan konkurrere med lignende tjenester, er der visse funktionaliteter, som den nødvendigvis må indeholde. Det skal være muligt at oprette en brugerprofil som henholdsvis lejer eller udlejer, alt efter om man ønsker at lease en andens bil eller stille sin egen bil til rådighed. Som udlejer skal man kunne oprette en bilprofil, der indeholder relevante informationer vedrørende bilen. Efter oprettelsen kan man logge ind på sin profil på applikationen og redigere/slette sin brugerprofil eller

bilprofiler.

Lejere skal have mulighed for at søge mellem de oprettede bilprofiler, til de finder en bil, som opfylder deres krav. Når der skal indgås en aftale om leje af en bil, er der behov for et besked- eller notifikationssystem mellem lejer og udlejer. Det skal således være muligt for lejer at anmode om leje af bil, og udlejer skal have mulighed for at reagere på anmodningen med en godkendelse eller afvisning.
Funktionaliteten skal integreres med en brugergrænseflade, som er lettilgængelig og intuitiv at anvende for at sikre den bedst mulige brugeroplevelse.

6.4 Skitse af systemet

Ud fra ovenstående beskrivelse er der udarbejdet en skitse, som viser, hvordan der integreres med applikationen CarnGo. Denne er vist i figur 6.1.



Figur 6.1: Skitse af CarnGo

7 Kravspecifikation

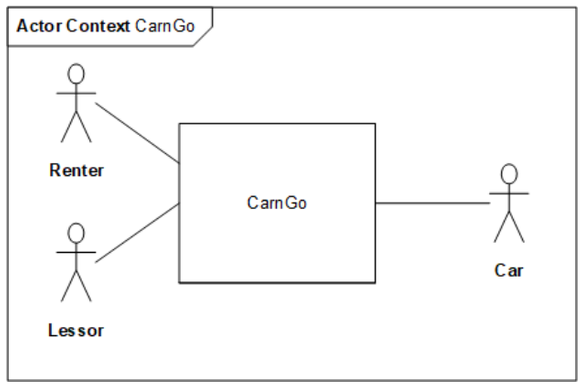
I dette afsnit beskrives både de funktionelle og de ikke-funktionelle krav for CarnGo applikationen,

7.1 Funktionelle krav

Til at starte med beskrives de funktionelle krav, hvilke aktører der er involverede, og hvordan CarnGo applikationen fungerer og ses fra brugerens synsvinkel.

7.1.1 Aktør beskrivelse

For at afklare hvilke aktører der anvender og interagerer med CarnGo, er der udarbejdet et aktør-kontekst diagram. Dette kan ses i figur 7.1.



Figur 7.1: Aktør-kontekst diagram for CarnGo

I figur 7.1 ses de to primære aktører *Lessor*, og *Renter*. Begge primære aktører er brugere af CarnGo applikationen, dog har de to aktører forskellig funktionalitet og autorisering. *Renter* kan leje biler til eget brug, men *Lessor* har derudover mulighed for at sætte biler til leje. Diagrammet viser også en *Car* som sekundær aktør. Bilen anses som sekundær aktør idet, at en fysisk bil skal kunne udlejes og lejes i applikationen, og derfor har indflydelse på systemet. Nu hvor aktørerne til systemet er kendt, kan der i de næste afsnit beskrives personas og kravene for systemet.

7.1.2 Personas

Personas er eksempler på typer af brugere af et system. Personabeskrivelserne tager udgangspunkt i den målgruppe, produktet ønsker at ramme - det segment af mennesker, som ikke har egen bil til rådighed.
Til inspiration for personas er der blevet undersøgt anmeldelser af firmaet GoMore, som har stillet en platform til rådighed for brugere, som ønsker at udleje privatbiler. Figur 7.2 viser tre reviews af GoMore fra Trustpilot. De er udvalgt, fordi de repræsenterer majoriteten af reviews af GoMore. Brugerne af GoMore er generelt utilfredse med de afgifter, som er blevet påtvunget udlejningsprocessen. De mener ikke længere, det er værd at bruge udlejningsportalen og leder derfor efter andre tjenester. CarnGo er et oplagt alternativ, da der er fokuseret på at holde applikationen fri for gebyrer og afgifter.

Figur 7.2: Udvalgte TrustPilot reviews for GoMore[4]

Personas tager udgangspunkt i to fiktive personer, som skal repræsentere det segment af mennesker, som applikationen prøver at ramme. Det er disse brugere og deres ønsker, som kravene baseres på, og som CarnGo skal imødekomme. De to fiktive personer ses nedenfor.

7.1.3 Lejer:

Navn og billede
Kirsten
Detaljer
26-årig studerende fra Aarhus, der er udeboende og tager offentlig transport, da hun ikke har råd til en bil. Har familie i København.
Mål
Vil gerne have mulighed for at leje en bil for at besøge sin familie et par gange om året.

7.1.4 Udlejer:

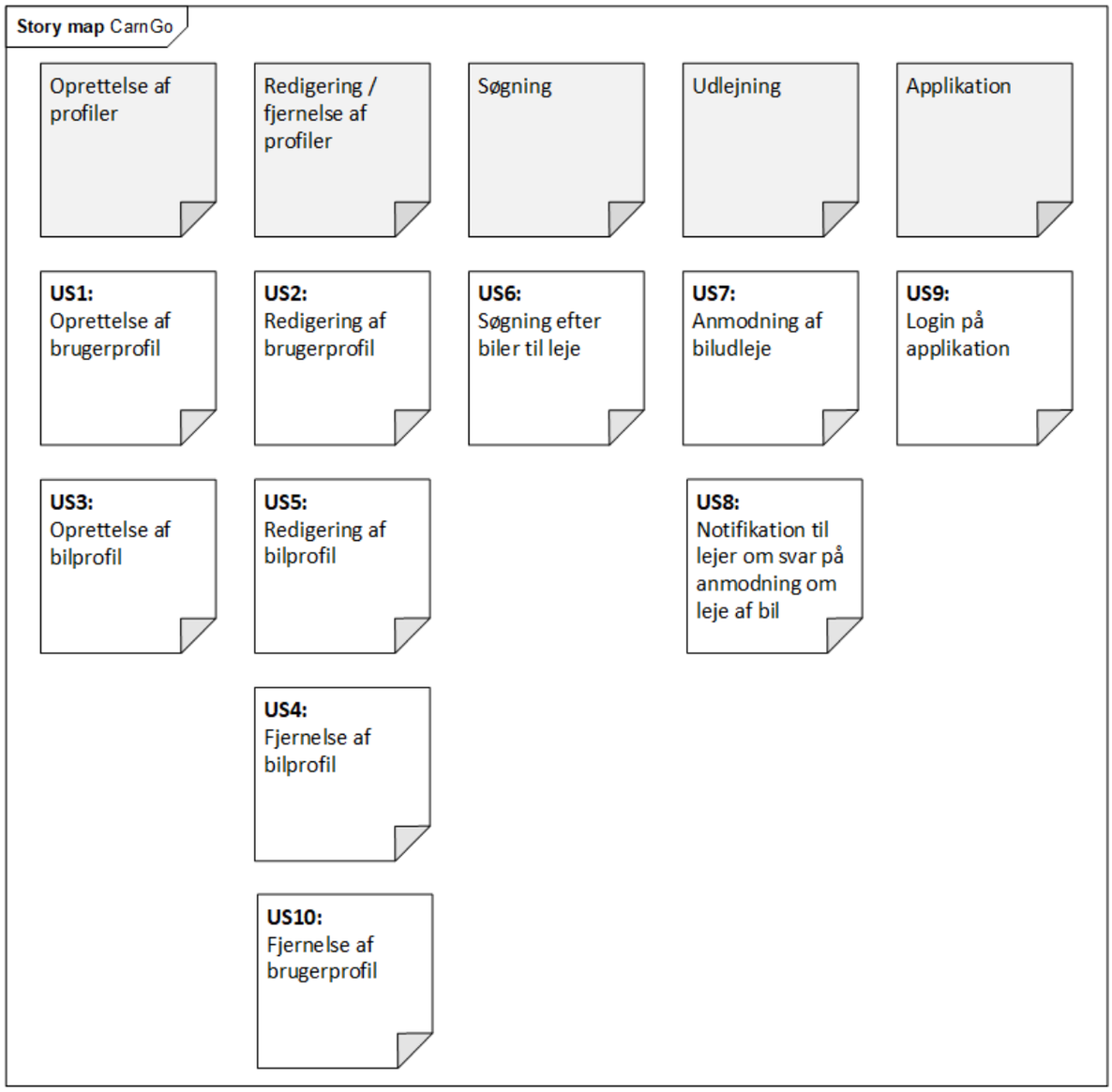
Navn
Svend
Detaljer
36-årig ingeniør fra Aarhus. Har taget lån til et hus og en bil, men cykler til og fra arbejde. Bilen bruges kun til møder et par gange om måneden.
Mål
Vil gerne leje sin bil ud, de dage han ikke bruger den for at supplere sin indkomst. Han gider ikke bruge penge på høje afgifter gennem en udlejningsplatform.

Med udgangspunkt i disse personas kan der udarbejdes krav for systemets funktionalitet. Der er foretaget en vurdering af, hvilke funktioner applikationen skal have for, at der skabes mest mulig værdi for førnævnte personas. Det har givet anledning til en række User Stories, som beskrives nærmere i det følgende.

7.1.5 User Stories

Til at forklare funktionaliteten af systemet anvendes der User Stories[5]. User Stories er kortfattede, ikke-tekniske beskrivelser af brugerens interaktion med systemet. Det vælges at lave User Stories i stedet for Use Cases, fordi Use Cases er meget omfattende. Use Cases beskriver brugsscenarier og undtagelser detaljeret og sekventielt. Frem for at fokusere på detaljerne for de enkelte scenarier, vælges det at anvende User Stories til at danne et overblik over hvilke features, der skal udvikles, og hvordan brugeren grundlæggende skal interagere med applikationen. Dette valg sparer tid for teamet og gør, at man hurtigt kan komme videre i udviklingsprocessen. Derudover vil User Stories kunne præsenteres for kunden inden for kort tid, hvilket også skaber værdi for dem. Generelt bør kunden være mere involveret, når der anvendes User Stories, men det skal siges, at der er både fordele og ulemper forbundet med dette. I dette tilfælde agerer gruppen selv kunder, og derfor kan User Stories give en vis grad af frihed i udviklingsarbejdet. For en mere uddybende forklaring se afsnit **6 User Stories** i bilag **Kravspecifikation**

User Stories for CarnGo er opsummeret i et Story Map i figur 7.3 [6]. Første række angiver epics, dvs. de kategorier som systemets User Stories falder ind under. For hver epic er der således en til flere relaterede User Stories, som er listet vertikalt. Det er intentionen, at de bliver mere specialiserede, som man bevæger sig nedad. Placeringen af den enkelte User Story har dog ingen betydning for dens prioritet.



Figur 7.3: User Story Map

Som det kan ses i figur 7.3, er der fem epics og 10 User Stories. Disse User Stories er udvalgt, da de er med til at skabe mest mulig værdi for personas i systemet. Tilsammen udgør de nemlig en prototype, der gør det muligt for personas at oprette sig som bruger og udleje/leje en bil. Samtidig opfyldes det økonomiske perspektiv fra begge personas, da der ikke indgår betaling gennem applikationen og derfor ingen skjulte eller høje gebyrer.
De vigtigste User Stories bliver beskrevet fuldt ud i dette afsnit, mens de resterendes fulde beskrivelser kan findes i afsnit **6 User Stories** i bilag **Kravspecifikation**.

7.1.6 Oprettelse af profiler

User Stories inden for dette epic beskriver, hvordan en bruger kan interagere med sin profil, og hvordan oprettelsen af den foregår. Den første User Story i denne epic beskriver brugerens første interaktion med applikationen.

US1: Oprettelse af brugerprofil

Egenskaber:
Som bruger
Ønsker jeg at kunne registrere mig som lejer eller udlejer
Fordi jeg vil kunne anvende systemet, enten som lejer eller udlejer
Baggrund: Brugeren har ikke registreret en profil endnu

Scenarie: Registrer lejer
Givet brugeren ønsker at oprette en profil
Når brugeren navigerer til siden for brugeroprettelse,
Og der indtastes e-mail og ønsket kodeord
Og brugeren registrerer sig.
Så navigeres brugeren til Login siden

Scenarie: Registrer udlejer
Givet brugeren ønsker at registrere sig som udlejer
Når brugeren logger ind med en registreret bruger
Og navigerer til brugerindstillinger
Og registrerer sig som udlejer
Så vises en menu for oprettelse af bilprofiler

US3: Oprettelse af bilprofil

Egenskaber:
Som Udlejer
Ønsker jeg at oprette en bilprofil for den bil, jeg vil udleje
For at kunne få kontakt til mulige lejere af bilen.
Baggrund: Brugeren har registreret sig som udlejer

Scenarie: Oprettelse af bilprofil
Givet at udlejer er logget ind
Når udlejer navigerer til siden for bilprofiler
Og udfylder en skabelon med information om bilen
Og sætter bilen til leje
Så vises bilprofilen for mulige lejere

7.1.7 Redigering/ fjernelse af profiler

Denne epic omhandler brugerprofiler og bilprofiler. Både lejer og udlejer skal have mulighed for at ændre personlige informationer såsom e-mail eller navn. Der skal også være mulighed for at slette sin bruger, hvis man ikke længere ønsker at benytte tjenesten. Udlejer skal desuden have mulighed for at redigere vigtige bilattributter for sine biler, og slette en bilprofil, hvis den pågældende bil ikke længere skal lejes ud.

7.1.8 Søgning

US6: Søgning efter biler til leje

Egenskaber:
Som Lejer
Ønsker jeg at kunne søge efter udlejede biler, der opfylder mine behov
For at se biler, som er til leje og eventuelt anmode om at leje en.
Baggrund: Lejer har registreret en profil

Scenarie: Søgning efter biler til leje
Givet at lejer er logget ind
Når lejer navigerer til siden for søgning
Og foretager en søgning baseret på nogle bilattributter
Så vises de biler, der opfylder kriterierne.

7.1.9 Udlejning

User Stories inden for dette epic beskriver, hvordan udlejningsprocessen fungerer mellem udlejer og lejer.

US7: Biludlejer besvarer anmodning om leje

Egenskaber:
Som Udlejer
Ønsker jeg at kunne se anmodninger om leje af min bil
For at kunne godkende/afvise eventuelle anmodninger
Baggrund: Udlejer har registreret sig som udlejer og oprettet en bilprofil

Scenarie: Udlejer godkender anmodning om leje af bil
Givet udlejer er logget ind på applikationen
Og udlejer vises alle anmodninger for leje af sin bil
Når udlejer godkender en anmodning
Så meddeles den anmodede om at kontakte udlejer

Scenarie: Udlejer afviser anmodning om leje af bil
Givet udlejer er logget ind på applikationen
Og udlejer vises alle anmodninger for leje af sin bil
Når udlejer afviser en anmodning
Så meddeles den anmodede om afvisningen

US8: Lejer anmoder om billeje

Egenskaber
Som Lejer
Ønsker jeg at få notifikationer når min anmodning om leje af en bil besvares

For at kunne begynde kontakt med udlejer om betaling og afhentning
Baggrund: Lejer har oprettet en profil og har anmodet om leje af en eller flere biler.

Scenarie: Lejer får godkendt en anmodning om leje af bil
Givet at lejer er logget ind på applikationen
Og en udlejer har godkendt lejers anmodning
Når lejer trykker på notifikationer
Så vises notifikation om, at lejers anmodning er godkendt
Og oplysninger der gør lejer i stand til at kontakte udlejer

Scenarie: Lejer får afvist en anmodning om leje af bil
Givet at lejer er logget ind på applikationen
Og en udlejer har afvist lejers anmodning
Når lejer trykker på notifikationen
Så ser lejer, at anmodningen er afvist

7.1.10 Applikation

Denne epic beskriver, hvordan en bruger kan logge ind på applikationen med enten sin udlejer- eller lejerprofil.

7.2 Ikke-Funktionelle krav

De vigtigste ikke-funktionelle krav bliver beskrevet i dette afsnit. Disse krav har fulgt FURPS-modellen og MoSCoW-prioritering, og i det efterfølgende vises udvalgte MUST krav for applikationen[7][8]. For en fuld beskrivelse se afsnit **7 Ikke-funktionellekrav** i bilag **Kravspecifikation**.

Functionality

- CarnGo applikationen skal aldrig fryse, dvs. ophøre med at respondere på brugerens input, i mere end 3 sekunder af gangen.
- CarnGos knapper skal reagere, når der bliver sluppet på knappen.

Usability

- CarnGos typografi skal ha et kontrastforhold over 4.5:1 for at følge WCAG AA standarden[9].

Reliability

- CarnGo skal håndtere 10 aktive brugere af applikationen på samme tid.
- CarnGo skal håndtere 20 inaktive brugere på applikationen på samme tid.

Performance

- CarnGo skal være klar til brug inden for 10 sekund efter, at applikationen åbnes.

Maintainability

- CarnGo's GUI kode skal følge et GUI design pattern, der gør det nemt at overføre til andre platforme.
- CarnGo skal kunne tilgås på en PC med internet forbindelse og Windows 10 OS.

Security

- CarnGo skal censurere brugerens password, når der logges ind på applikationen.

8 Afgrænsning

Indledning

I dette projekt er der løbende blevet lavet overvejelser vedrørende, hvad systemet skulle indeholde, hvilke features og operationer som skulle opfyldes, samt overholdelse af love og tekniske krav. For at se alle afgrænsninger i projektet se bilag **Afgrænsning**. I dette afsnit beskrives derfor udvalgte afgrænsningerne for projektet.

8.1 Webapplikation eller platformsuafhængig applikation

I første udkast af systemet var der valgt at lave både en webapplikation, som kunne tilgås gennem en browser, samt en applikation, som var platformsuafhængig. Men efter evaluering af tidsresourser blev applikationen begrænset til én platform. Hvis der skal konkurreres med GoMore, er det dog en vigtig faktor at lave en platformsuafhængig applikation.

8.2 Pengetransaktioner

Det økonomiske element er en stor faktor i dette projekt. Det er kritisk at give brugeren en afgiftsfri betalingsplatform, som er pålidelig. Her kunne man have implementeret en payment gateway til en autoriseret e-handelstjeneste som fx PayPal[10][11]. Det er dog fravalgt for prototypen i projektet, da det er ubekendt, hvordan der oprettes en payment gateway i en Windows applikation, samt der er lovmæssige reglementer, som er ukendt for gruppen. Betalingen vil derfor ligge internt mellem brugerne. Platformen forsyner dog udlejeren med mulighed for at angive den daglige udlejningspris.

8.3 Forsikring og Lov

Hele forsikringsproceduren undlades i dette projekt, da gruppen ikke har nok forudsætninger til at definere og udforme en forsikringskontrakt med et eksternt firma eller etablere et internt. Hvis produktet skulle markedsføres, er dette en afgørende faktor. En anden ting som skulle tilføjes er validering af ens profil. Dette kunne fx gøres med NEM-Id[12] eller andre identifikationsmidler. Biler har ofte en stor markedsværdi, og det er derfor essentielt at garantere en sikker og tryg udlejningsproces - både for lejeren af bilen og udlejeren.

I forlængelse med opbevaring af personoplysninger er der også flere GDPR-regler som skal overholdes[13]. Dette indebærer føring af en fortegnelse, dokumentation af at god databehandling efterleves mm.

8.4 Personaanalyse og Markedsføring

Projektets kravspecifikation tager udgangspunkt i personas (User Stories), så derfor ville det have været ideelt at have foretaget en segmeneteringsanalyse, så heterogene medlemsbaser kunne opdeles ud fra forskellige karakteristika. Ud fra denne analyse kunne gruppen have fundet og etableret en målgruppe, og derved specificere de essentielle elementer for systemet; hvordan rammes kunderne, og hvordan adskilles der fra konkurrenten GoMore. Ud fra undersøgelser baseret på TrustPilot reviews udledes, at brugerne er utilfreds med to punkter: Afgifter og mangel på tryghed/sikkerhed gennem deres udlejningsportal. Selvom der bliver tilføjet disse to elementer til produktet, kan GoMore let sænke deres udgifter for at konkurrere med CarnGo, samt holde dominans på markedet. Hvis CarnGo skal blive kommercialiseret, er det nødvendigt at tilføje noget nyt og innovativt for at tiltrække kunder,

da GoMore allerede har en stor brugerbase.

Systemets afgrænsninger er defineret, og det er nu vigtigt at kigge på udviklingsprocessen af projektet. Dette beskrives i næste afsnit.

9 Metode og Proces

I dette afsnit beskrives processen bag CarnGo, et 4. semesterprojekt udarbejdet i samarbejdet med Aarhus Universitet. I forlængelse med dette vil der være en beskrivelse af de redskaber og udviklingsmetoder, som er anvendt under arbejdsprocessen. Projektet har visse krav og læringsmål angivet af Aarhus Universitet. Det skal:

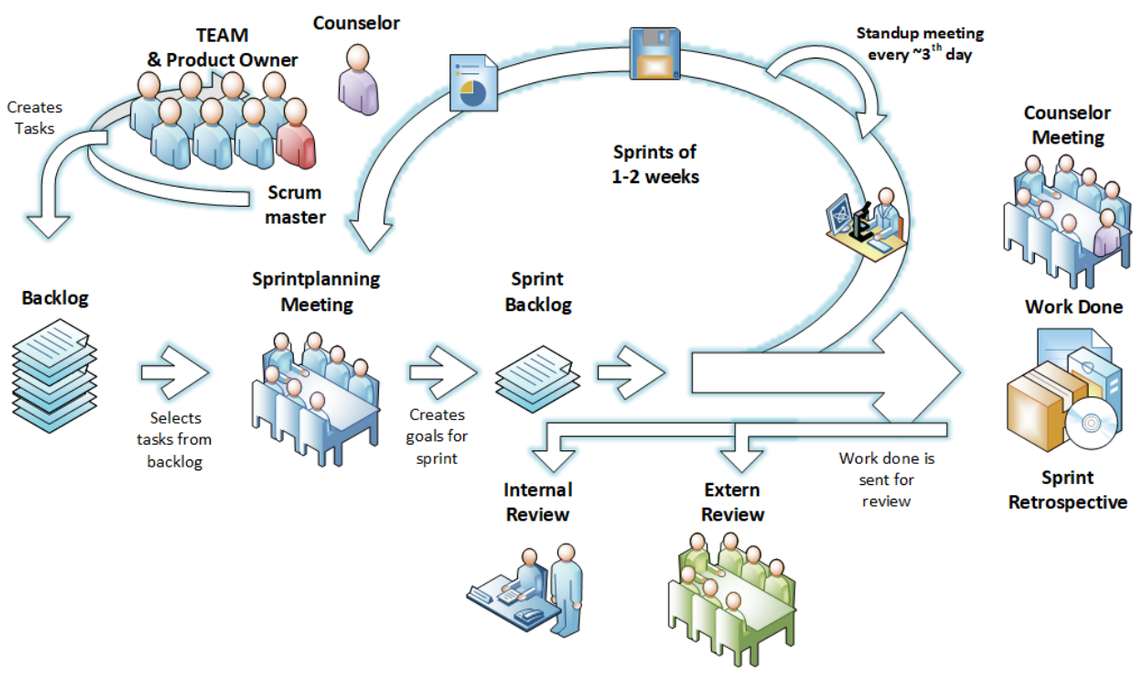
- Anvende en iterativ udviklingsproces
- Anvende projekt- og versionsstyringsværktøjer
- Kombinere viden fra flere af semestrets kurser og anvende denne i projektet.

Et vitalt element for udviklingsprocessen er, at den er iterativ. Det er såvel et krav for projektet, men også en central del af softwareudvikling. Hvis planlægningen skal være statisk, kræver det stabile krav og en forudsigende arbejdsproces. Dette har vist sig at være en farlig antagelse for softwareudvikling[14]. Projektgruppen har derfor valgt at anvende SCRUM som udviklingsmetode[15]. Hvis der ønskes den fulde procesdokumentation, henvises læseren til bilag **Proces**.

9.1 Anvendelse af Scrum i PRJ4, Gruppe 5

Til forskel fra de tidligere semestre har der ikke været noget krav om udviklingsmetoder til projektet med den undtagelse, at der skulle anvendes en iterativ udviklingsproces. Det blev besluttet at anvende Scrum, da flere medlemmer af gruppen havde god erfaring med dets fleksible og adaptive tilgang til arbejdsprocessen. De korte iterationer giver en effektiv feedback mekanisme. Gruppen havde desuden en tilpasset version af Scrum, udarbejdet i et af de tidligere semesterprojekter[16].

Alle iterationer af projektets udarbejdelse har tilføjet elementer til gruppens anvendelse af Scrum. Gennem stadierne er der blev gjort erfaringer og reflekteret over, hvordan gruppen optimerer arbejdsprocessen, samt stadigvæk sikrer kvalitet. Figur 9.1 illustrerer den endelige udviklingsmetode for projektgruppen.



Figur 9.1: Anvendelse af Scrum i projektgruppe 5. Dette er en udvidet version af den, som blev udarbejdet i semesterprojekt 3, gruppe 7 [17]

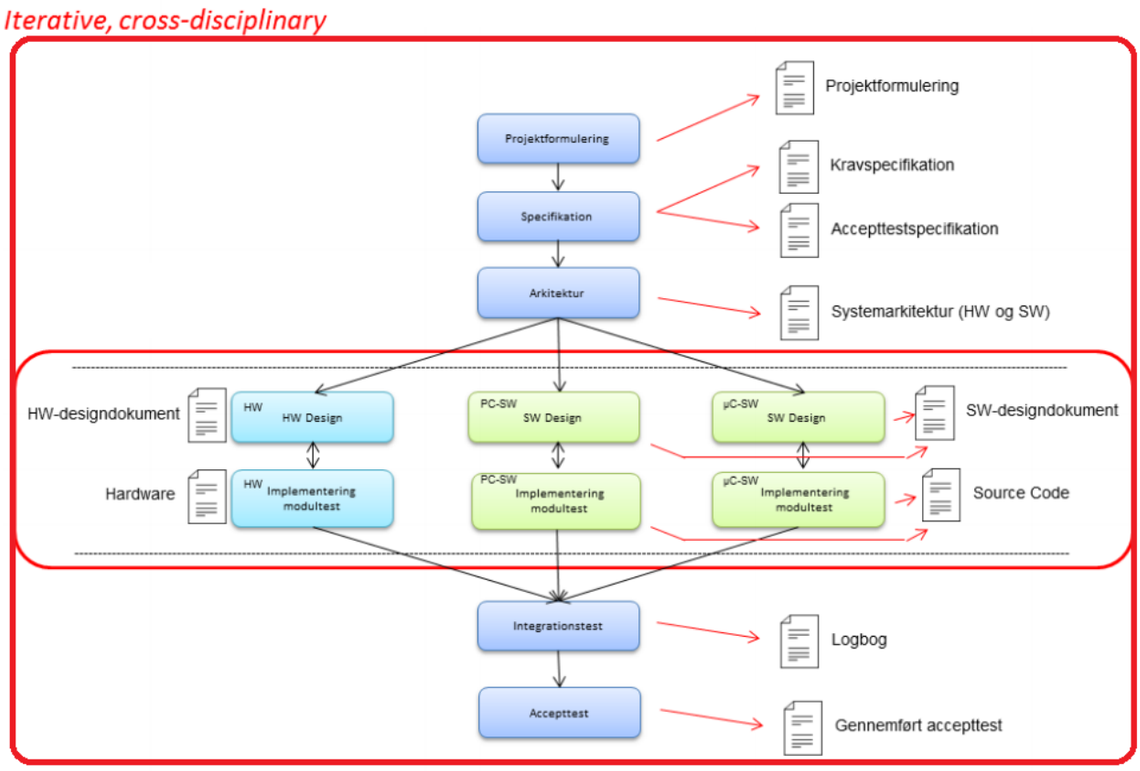
Gruppen har en Scrum Master, dog varetages rollen som Product Owner af hele gruppen: Projektets krav er specificeret af gruppen. Gruppen vælger hvilke funktioner, egenskaber og arbejdsopgaver, som skal prioriteres. Rollen som Scrum Master er tildelt Tristan Møller. Han har til ansvar at facilitere Scrum, fjerne interne og eksterne forhindringer og forbedre gruppens arbejdsproces.

Hvert sprint har en varighed af 1-2 uger, hvor der er Standup møder ca. hver 3. dag. Relevante opgaver tages fra Backlog'en og medtages i sprintplanlægningen. Hvert gruppemedlem skal gerne have arbejdsopgaver svarende til 6-8 timers arbejde per uge. Den sidste mandag i sprintet reviewes alle substantielle eller kritiske tasks. Desuden sendes materiale til review hos vejleder, hvis gruppen ønsker en vurdering udefra.

Når et sprint er slut, tager gruppen et retrospektiv af sprintet. Her vurderes, hvad der er gået godt, og hvilke processer der skal optimeres. I de senere sprint blev der også foretaget en risikovurdering på projektets status. Dette har været en af de mest essentielle feedback mekanismer i projektforløbet og har gjort det muligt at forbedre gruppens arbejdsprocesser.

9.2 Iterativ ASE-udviklingsmodel og SYSML

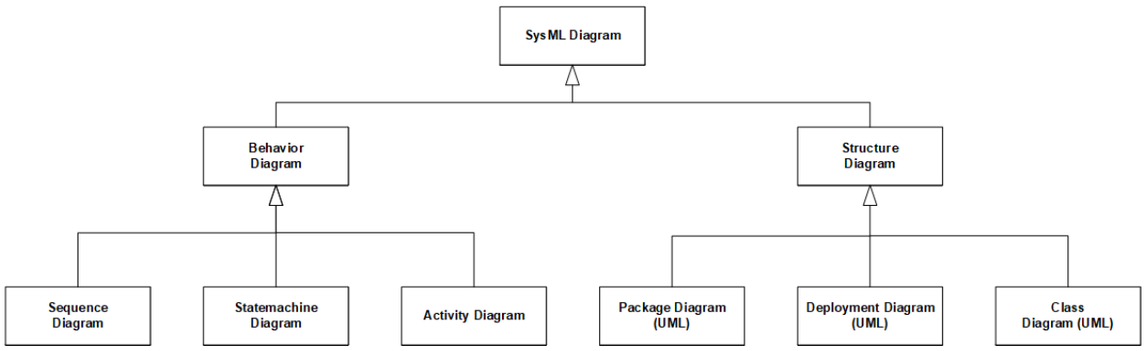
I de forrige semestre har det været et krav at anvende ASA-modellen. Modellen er velegnet til software projekter, hvor der er et godt domænekendskab. Den har dog flere elementer fra vandfaldsmodellen, idet processerne er sekventielle[18]. Scrum er en iterativ arbejdsmetode, og der anvendes en iterativ tilgang ved udarbejdelse af alle processer i ASE-modellen (se figur 9.2). Processerne beskrevet i ASE-modellen anvendes i flere iterationer (Sprints), hvor hver iteration ikke nødvendigvis er bundet til én proces. I et sprint kan der både arbejdes på kravspecifikation og arkitekturen.



Figur 9.2: Iterativ ASE-model[19]

Der anvendes SysML som analyse- og designredskab. Det bruges til at modellere og beskrive systemet. Af figur 9.3 ses det at Sekvens-, Aktivitet- og Statemachine diagrammer bruges til

at beskrive opførelsen af systemet. Class diagram fra UML er også inkluderet under struktur, da den beskriver strukturen af software samt relationen mellem de forskellige klasser.



Figur 9.3: Anvendelse af SysML

Projektets udviklingsproces er nu defineret, samt modeller og redskaber som anvendes under udviklingen af systemet. Før systemet endeligt kan designes og implementeres, er det nødvendigt at foretage en analyse af systemet ud fra de givne krav. For mere information om proces ses bilaget **Proces**.

10 Analyse

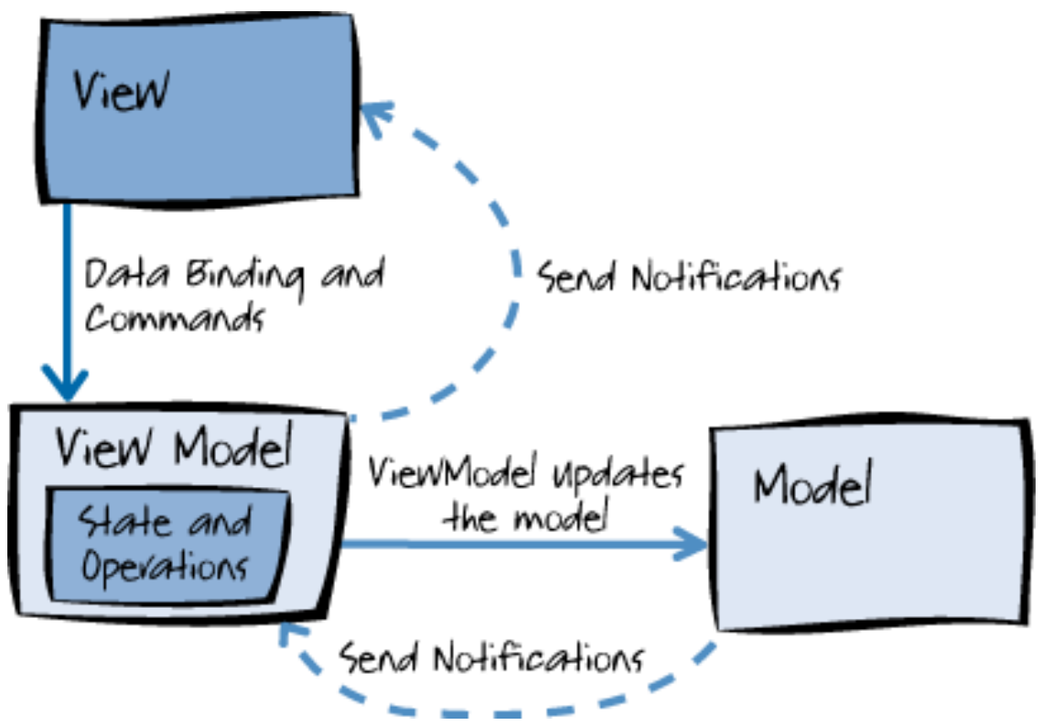
I dette afsnit beskrives resultater af analysen. De største overvejelser for projektet har omhandlet den grafiske brugergrænseflade og databasen. De endelige valg begrundes kort for applikationen og databasen. For den fulde analyse henvises til bilaget **Analyse**.

10.1 Applikationstype

Det var et krav for systemet, at det skulle være en applikation, der kunne interagere med brugere gennem en grafisk brugergrænseflade. Der var tale om flere typer applikationer, samt hvorvidt den skulle være platformsuafhængig. WPF-systemet blev valgt efter dets MVVM-strukturerede design passede til gruppens behov, samt det gav et system med lav kobling mellem de forskellige lag (Presentation, Business og Data layer)[20][21]. Desuden var frameworket en del af pensum, hvilke betød at der var rig mulighed for at få hjælp af faglærere.

10.1.1 MVVM

MVVM-arkitekturen gør det muligt at holde applikationens forretnings- og præsentationslogik adskilt. Arkitekturen består af tre kernekomponenter: View, ViewModel og Model (se figur 10.1).



Figur 10.1: MVVM-arkitektur[22]

Views indkapsler den grafiskebrugergrænseflade og den logik der er stærkt koblet til de visuelle elementer, som præsenteres for brugeren.

ViewModels indholder præsentationslogikken. Det har ingen direkte reference til View'et. ViewModellen indeholder Properties og Commands, som databindes til View'et. View'et bestemmer, hvordan funktionalitetet skal gengives, hvor ViewModeller har til ansvar at koordinere View'ets interaktion med systemets data og modeller.

Modeller indeholder applikationens forretnings-logik og data. Dette kan være nedhentning og administration af applikationsdata og for at sikre at forretningsregler, der sikrer data-konsistens og gyldighed, bliver overholdt.

10.2 Database

For at kunne opbevare brugerdata, er det nødvendigt at have en database. Hertil blev der overvejet to forskellige løsninger: En lokal database eller administreret clouddatabase. Den lokale database krævede, at der konstant skulle være en computer/enhed, som holdt databasen aktiv (Det er ikke let at distribuere en relationel database ud på flere maskiner). Microsoft Azure SQL database udbyder et administreret clouddatabase system og tilbyder studerende datahukommelse i et givet tidsinterval[23]. Desuden er Azure også indbygget i Visual Studio, hvilket er det IDE, som de fleste i gruppen bruger[24].
Det er valgt at lave en relationel database, da NoSQL er optimeret mere mod store mængder data af varierende type og mindre mod garantier fremfor, at data er fejlfrit[25][26].

Analysen af systemet har reduceret designvalgene for projektet ud fra de ressourcer og funktionalitet, som er blevet prioriteret. Som det første planlægges arkitekturen for systemet.

11 Arkitektur

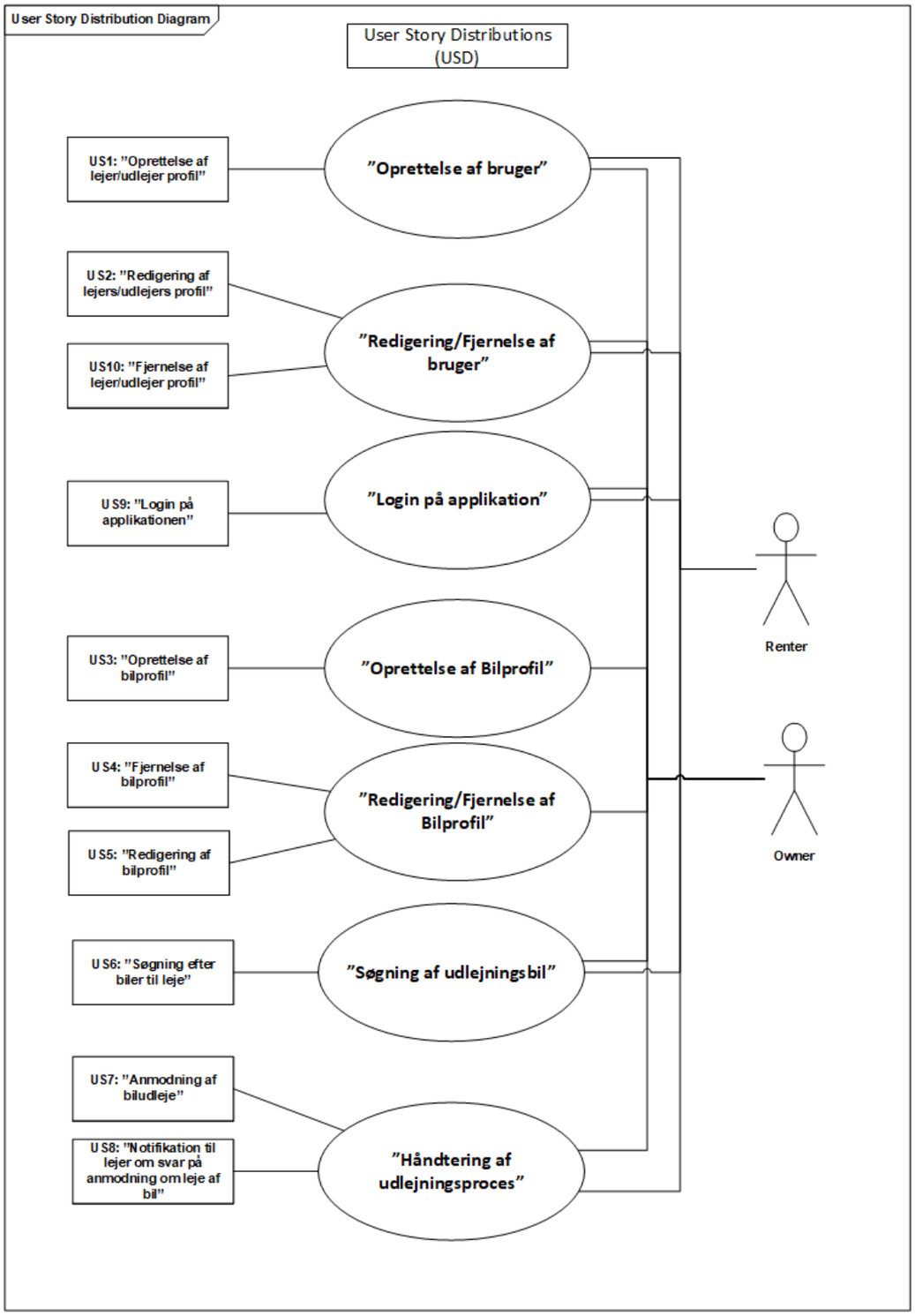
I de følgende afsnit beskrives arkitekturen for CarnGo, som er baseret på bilag **Kravspecifikation** og **Analyse**. Afsnittet kan dog læses uden kendskab til den fulde dokumentation af analyse og krav. Læseren henvises til bilag **Arkitektur** for den komplette dokumentation.

For at beskrive arkitekturen af det softwareintensive system bruges 4+1 modellen[27]. Her bruges de udvalgte User Stories til at illustrere arkitekturen, og 4+1 modellen bruges til at beskrive systemet ud fra forskellige perspektiver.

- **Logical View:** Beskriver systemets funktionalitet, som leveret til slutbrugerene. Dette illustreres vha. applikationsmodeller. Da der laves en WPF-applikation er Model-view-viewmodel(MVVM) brugt til at beskrive software arkitekturen.
- **Process View:** Beskriver systemetprocesserne, og hvordan de kommunikerer, samt systemets runtimeadfærd.
- **Development View:** For at beskrive systemkomponenterne anvendes et package diagram.
- **Physical View:** Beskriver de fysiske lag såvel som de fysiske forbindelser mellem komponenter. Dette illustreres med et deployment diagram.
- **Scenarier:** Her bruges User Stories til at illustrere arkitekturen. USDD diagrammet, viser hvordan scenerierne fra User Stories bliver samlet til en helhed, for at beskrive systemets forskellige elementer og funktionalitet (se figur 11.1). **User Story Distribution diagrammet** er ikke et foruddefineret værktøj, men et der bliver brugt til at give overblik over, hvordan User Stories bliver fordelt til systemfunktionalitet i dette projekt. Det er redskab til at samle kravene for en eller flere user stories således, at der kan benyttes værktøjer som SysML.
- **Security View:** Der tilføjes også et sikkerhedsview, som fokuserer på, hvordan systemet implementeres ud fra sikkerhedsmæssige elementer, og hvordan sikkerhed påvirker systemegenskaberne.

11.1 General Arkitektur

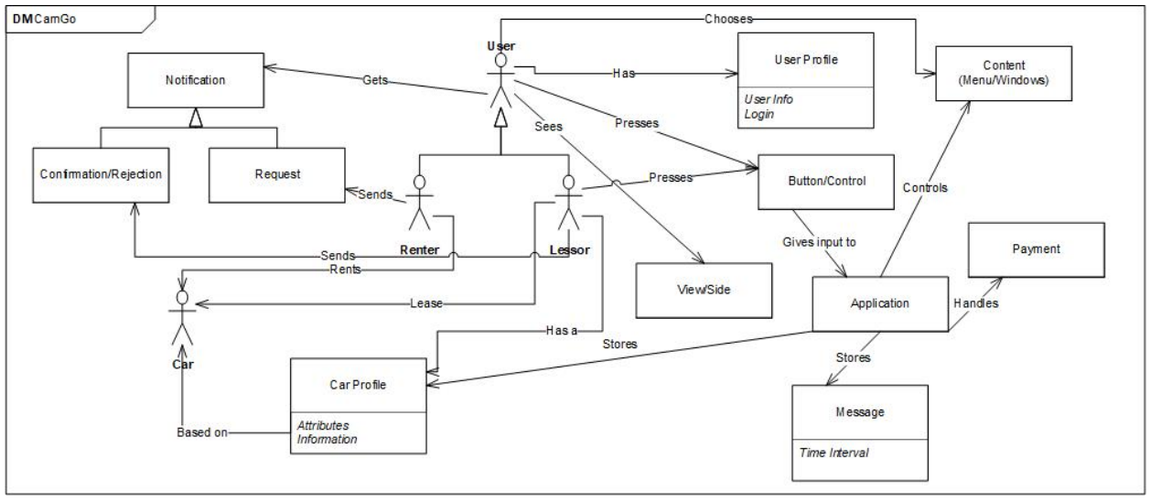
Før den egentlige software arkitektur påbegyndes, er der nogle generelle ting, der være på plads. User Story Distribution Diagrammet viser User Stories sammenhæng med arkitekturen. Det gør den, da den viser distributionerne af User Stories i et format, der er kompatibel med 4+1 modellen.



Figur 11.1: User Story Distribution Diagram (USDD)

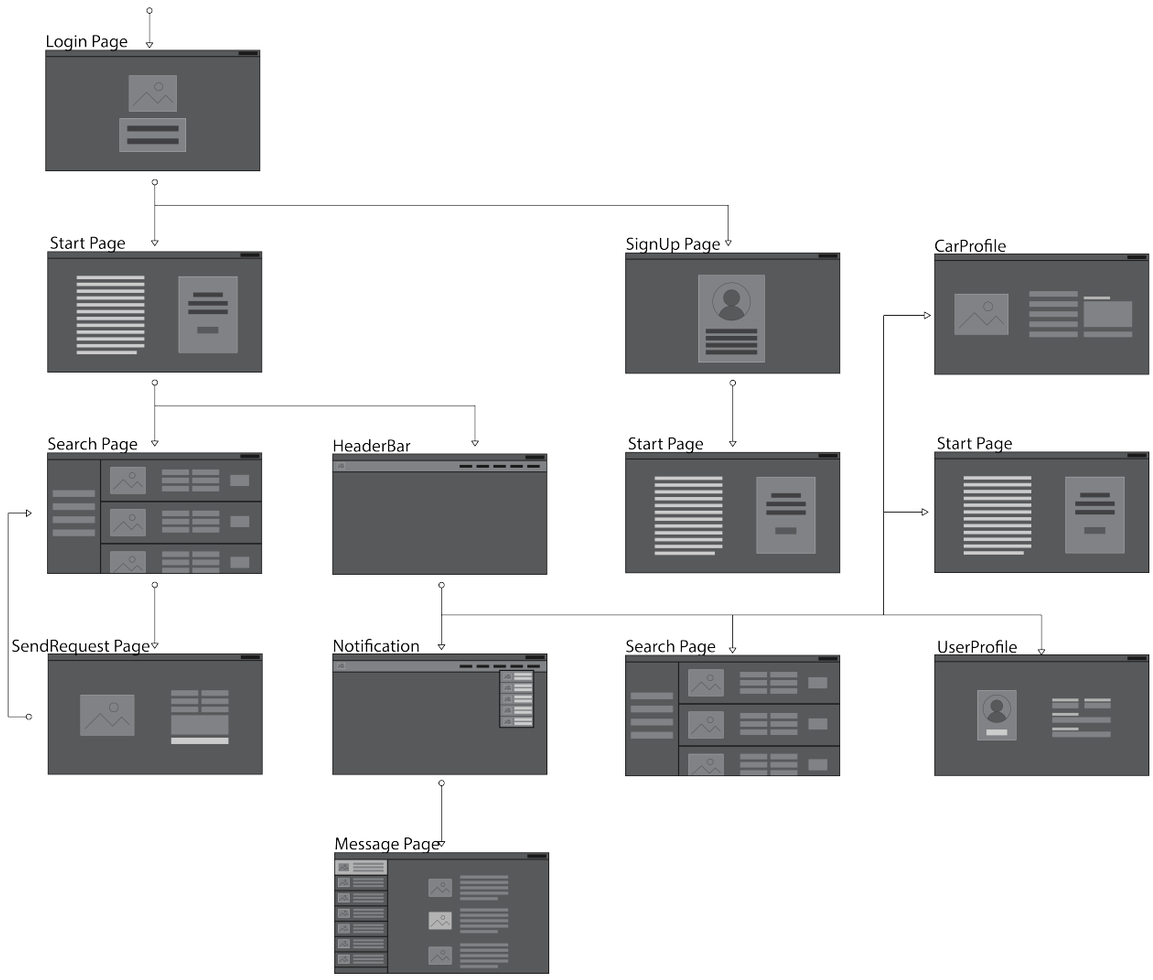
User Stories er strukturet ud fra det domæne angivet i **Problemformuleringen** og **Kravspecifikationen**. Figur 11.2 konceptualiserer domænet og beskriver dets adfærd og data. Domænemodellen angiver systemets eksterne og interne aktører/roller, som interagerer med softwarekomponenterne. Her introduceres rollerne i systemet, og hvilke rettigheder og muligheder de har i forhold til interaktionen med systemet og dets data. Kasserne skal ses som softwarekomponenter eller koncepter for systemet. Aktørerne er de roller, brugeren kan

have i systemet - de er en 'User', indtil de registeres i systemet; enten som 'Renter' eller 'Lessor'.



Figur 11.2: Domænemodel af CarnGo-systemet.

Wireframet nedenfor illustrerer programmets flow, og de vinduer som er konceptualiseret ud fra domænet og kravene angivet i bilag **Kravspecifikation**. Navigationen mellem de forskellige frames og deres karaktistik afspejles i applikationsmodellerne for det logiske view.



Figur 11.3: System Flow Wireframe[28]. På grund af den høje opløsning vises billedet ikke optimalt, læseren henvises til bilag **Wireframe** for den ideelle version.

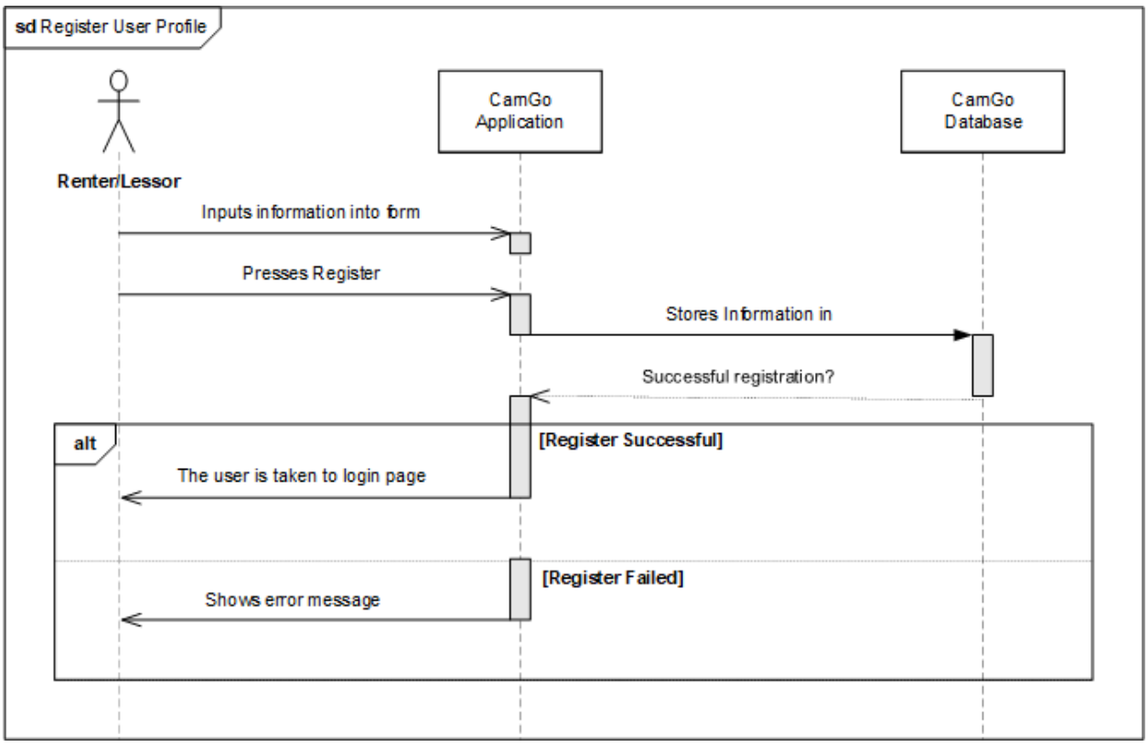
De næste afsnit vil gennemgå systemets arkitektur, grænseflade og adfærd.

11.2 Scenarier

Scenarierne beskriver systemets funktionalitet og adfærd, og tages udgangspunkt i User Story Distributionerne. Kun udvalgte distributioner vil blive gennemgået, de resterende henvises læseren til bilag **Arkitektur**. For at illustrere systemets adfærd bruges sekvensdiagrammer, som viser sammenspillet mellem aktør, applikation og database. Hver sektion består af et sekvensdiagram og tilhørende beskrivelse.

11.2.1 Oprettelse af bruger

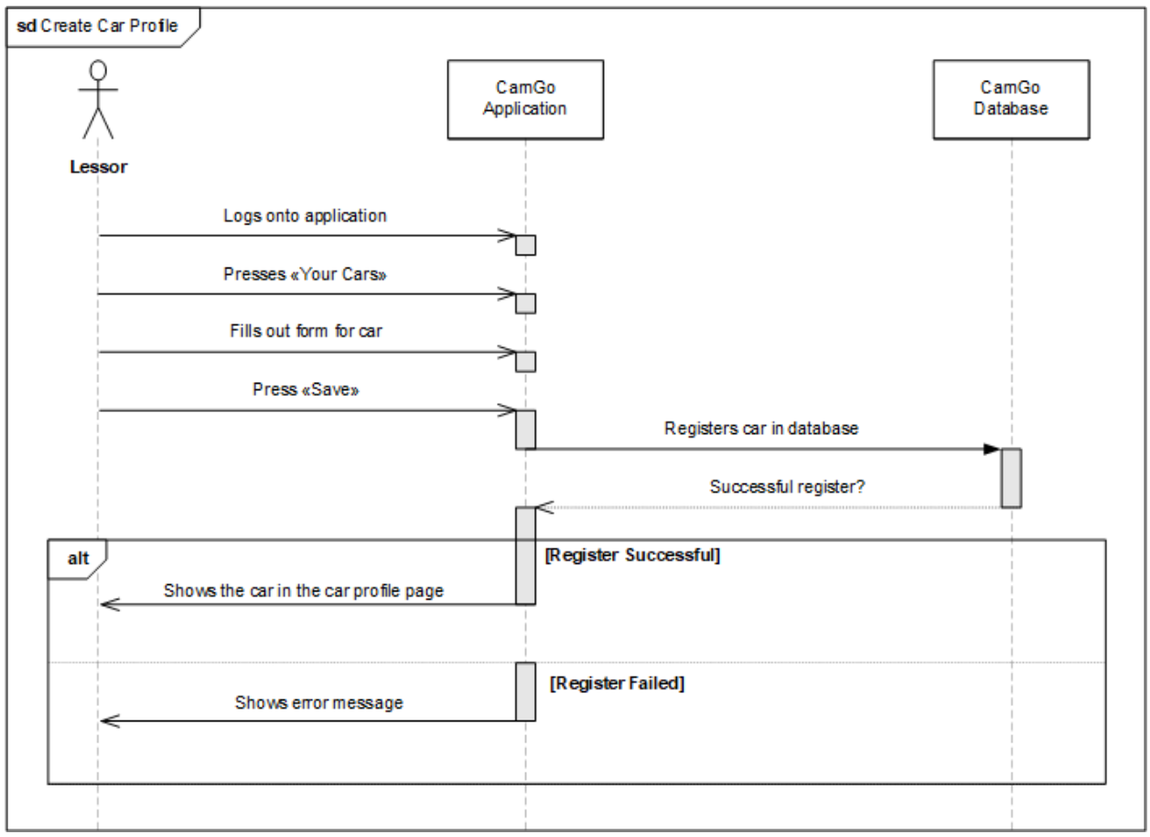
For at kunne anvende CarnGo applikationen må brugeren oprette en profil. Brugerens information gemmes i databasen ved oprettelse.



Figur 11.4: Her ses et sekvens diagram for samspillet mellem lejer/udlejer, CarnGo applikationen samt databasen.

11.2.2 Oprettelse af bilprofil

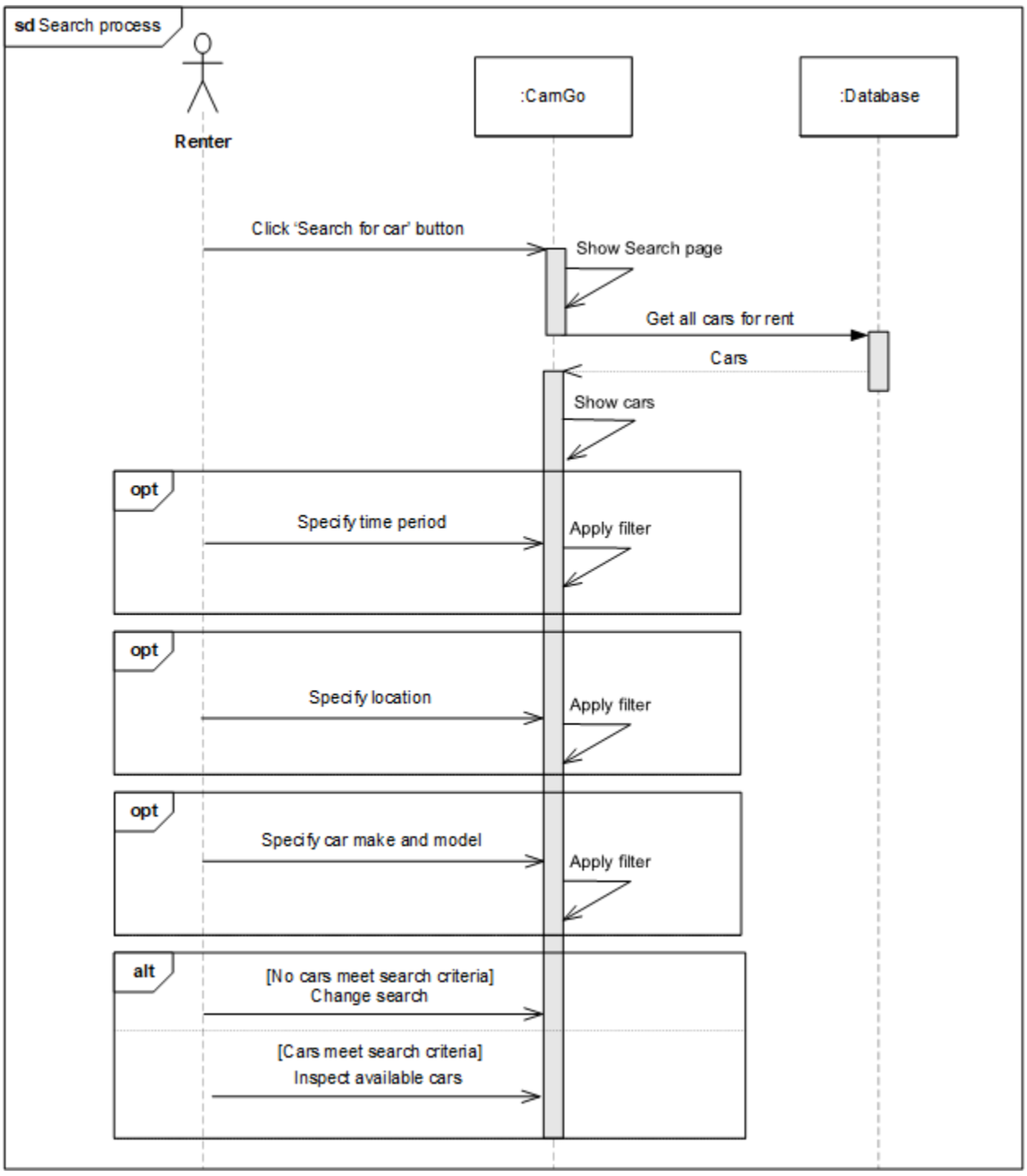
En bilprofil repræsenterer alt information omkring den bil, som udlejes. En bruger med udlejerettigheder skal have mulighed for at oprette en bilprofil. En bruger med lejerettigheder kan ikke oprette bilprofiler.



Figur 11.5: Her ses et sekvensdiagram for samspillet mellem udlejer, applikation og database.

11.2.3 Søgning

Når en lejer ønsker at leje en bil, så skal han/hun søge efter en i applikationens database. I kataloget er lagret alle bilprofiler, som er sat til leje.

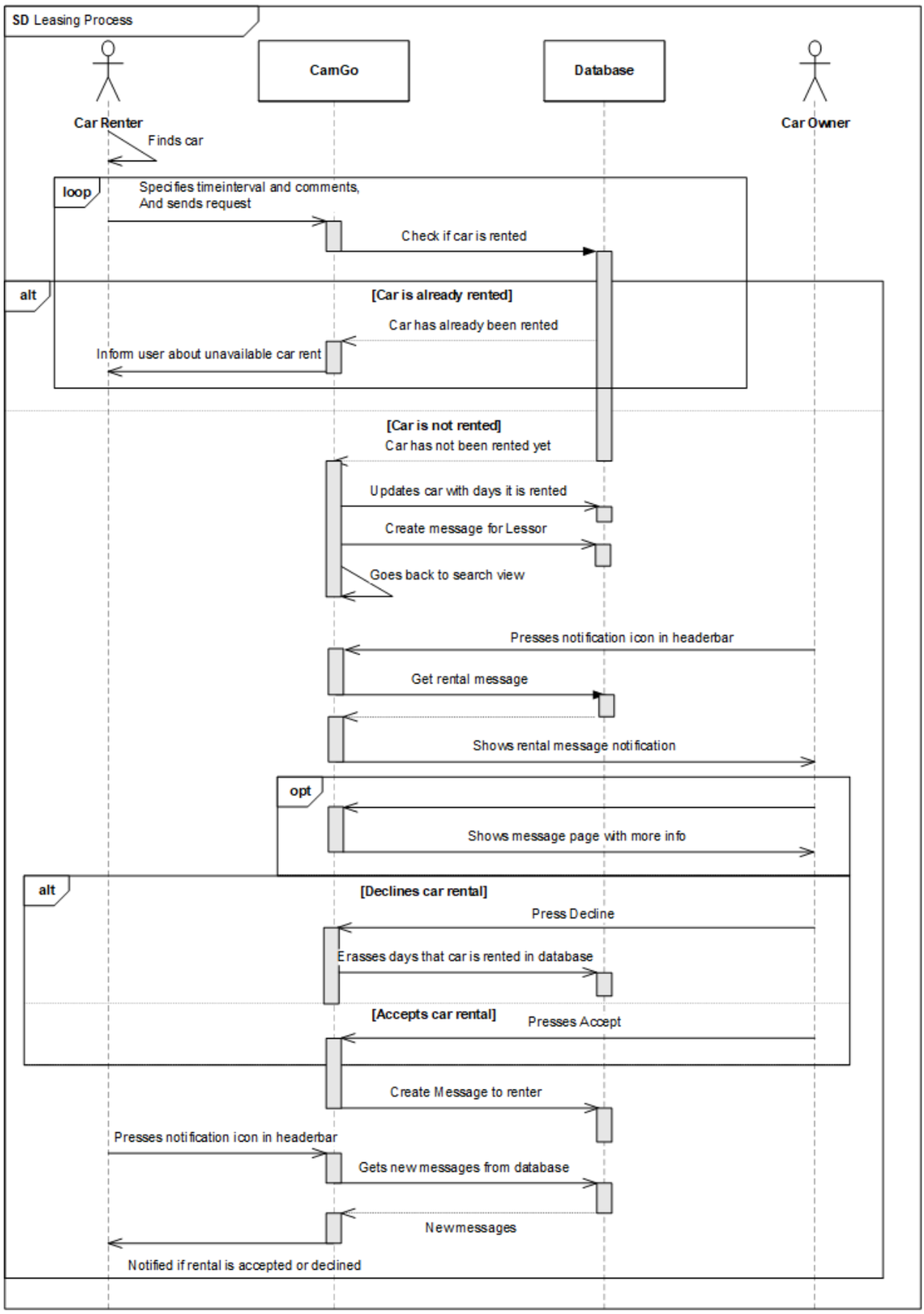


Figur 11.6: Sekvensdiagram for søgning efter biler til leje.

11.2.4 Håndtering af udlejningsproces

Efter at lejer har søgt i applikationens database og har fundet en bil, som lejer ønsker at leje, så starter udlejningsprocessen. Dette realiseres med følgende trin:

1. Lejer anmoder udlejer om billeje
2. Udlejer godkender/forkaster anmodning om billeje
   (a) Hvis Udlejer godkender så tager lejer kontakt til udlejer i forhold til udveksling af bil og betaling.



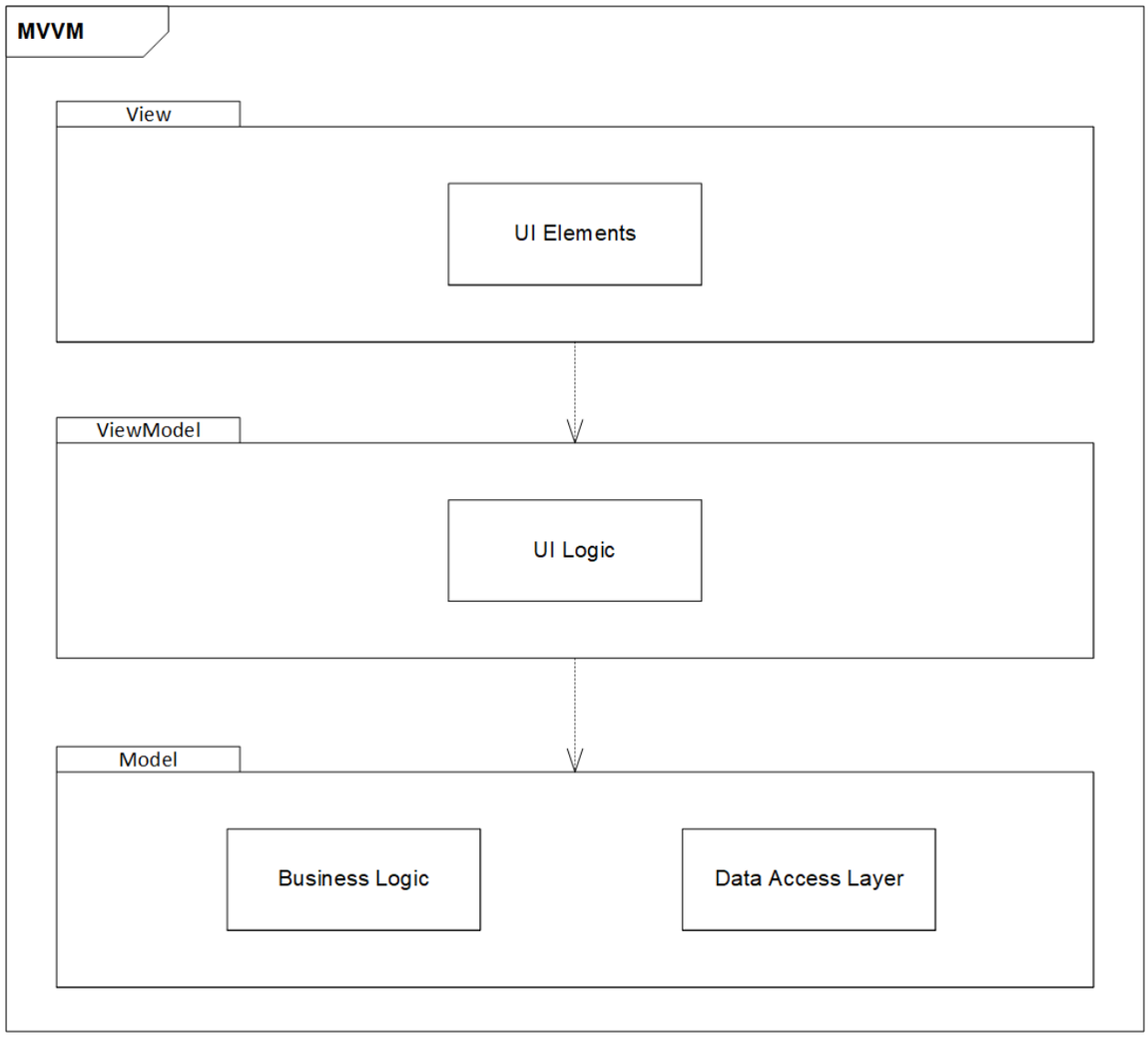
Figur 11.7: Sekvensdiagram for håndtering af udlejningsprocessen.

Se afsnit **2.2 Scenarier** i bilag **Arkitektur** for mere information om scenarier.

11.3 Logical View

Det logiske view omhandler funktionaliteten af systemet, som leveres til brugeren. Systemets funktionalitet, struktur og adfærd beskrives ved brug af applikationsmodeller, herunder klasse- og tilstandsdiagrammer. Der gennemgås de samme distributioner som i det forrige afsnit.

Klassediagrammerne er struktureret efter MVVM-arkitekturen, som illustreres i figur 11.8. Tilstandsmaskinerne viser, hvordan komponenterne interagerer med hinanden og deres relationer.

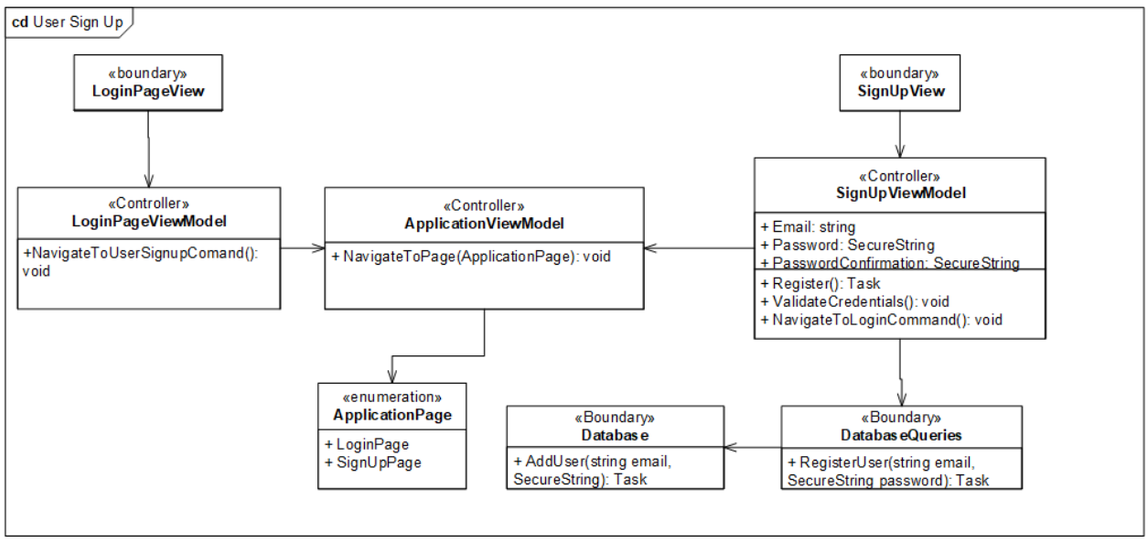


Figur 11.8: MVVM-arkitektur

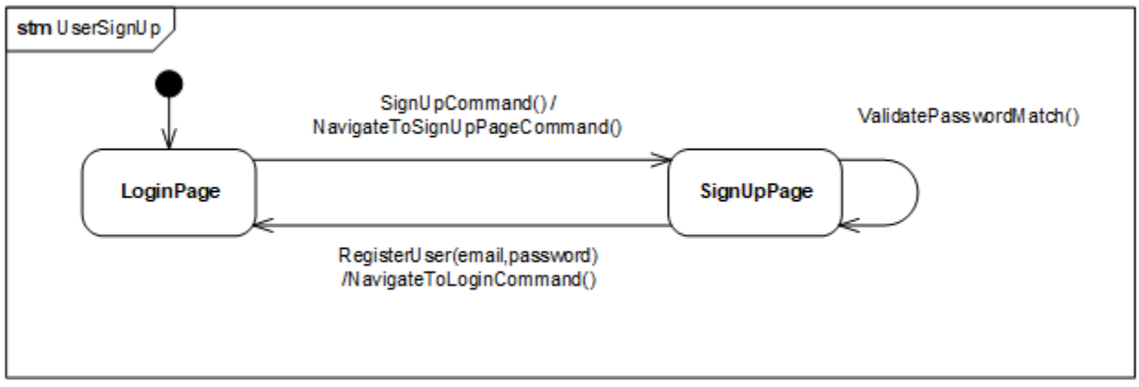
Arkitekturen for MVVM definerer, at ViewModel ikke skal have nogen relation til View, som indeholder de grafiske elementer for systemet og er framework bestemt med hensyn til WPF. (Hermed kan man også genbruge ens ViewModeller, hvis man skifter framework). Det skal dog stadigvæk være muligt at skifte View, samtidigt med at der kun er en relation fra View til ViewModel. Til dette oprettes klassen ApplicationViewModel. Alle ViewModeller anvender denne klasse til at kunne skifte View.

11.3.1 Oprettelse samt redigering af brugerprofil

Systemet giver brugeren to muligheder ved programstart: Login eller oprettelse. Når et af de to krav er opfyldt, navigeres brugeren til selve programmet. HeaderbarView er brugerens mulighed for at navigere til andre Views - den indeholder en navigationssti til næsten alle Views i systemet. Brugerinformationerne gemmes i databasen og kan fremover bruges til at logge ind med.



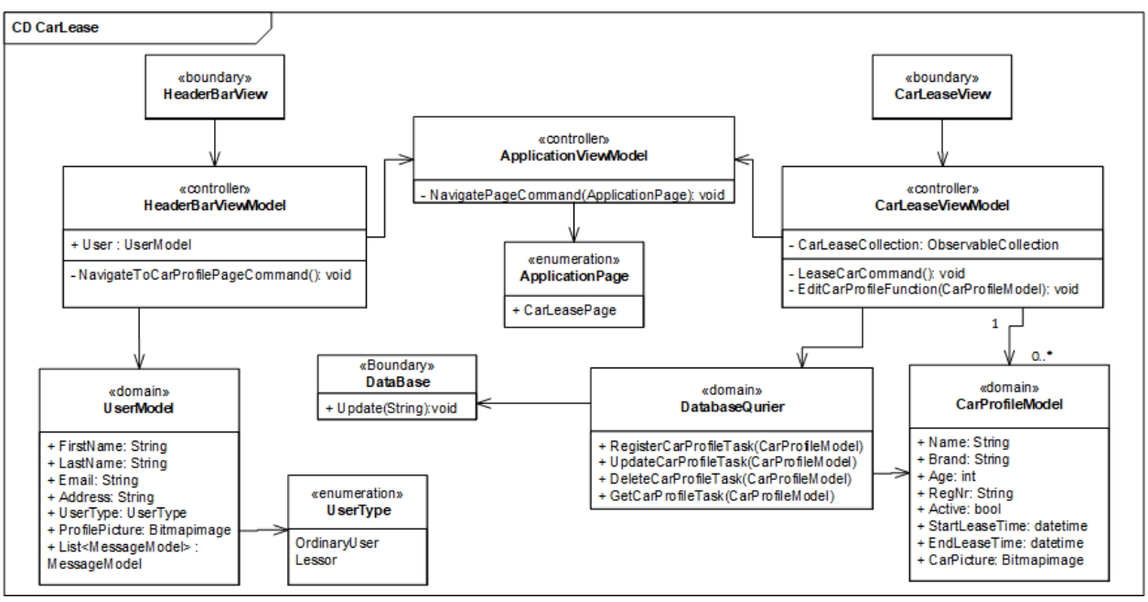
Figur 11.9: Klassediagram for oprettelse af brugerprofil.



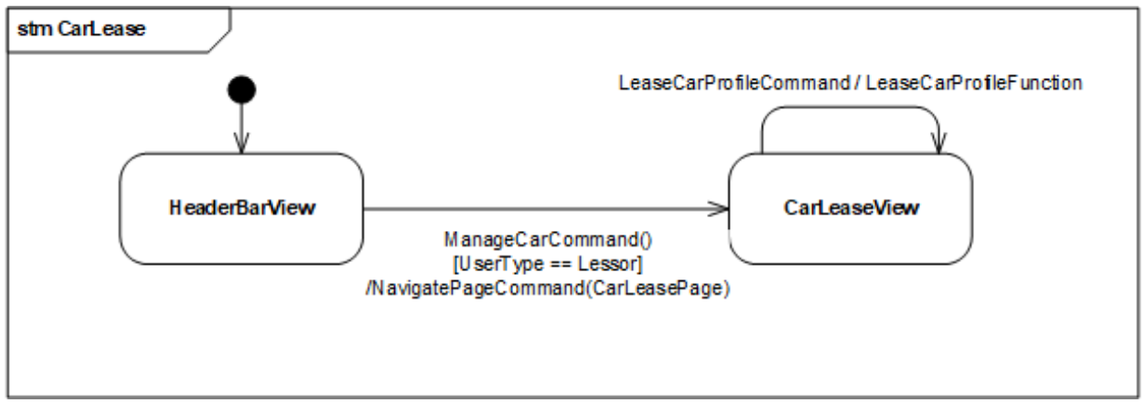
Figur 11.10: Tilstandsdiagram for oprettelse af brugerprofil.

11.3.2 Oprettelse af bilprofil

Hvis en bruger er registreret som udlejer, skal systemet give brugeren rettigheder til at oprette en bilprofil. Brugeren kan navigere til vinduet gennem HeaderBarView. Bilprofilen gemmes i databasen, og vil herefter vises i applikations søgeresultater.



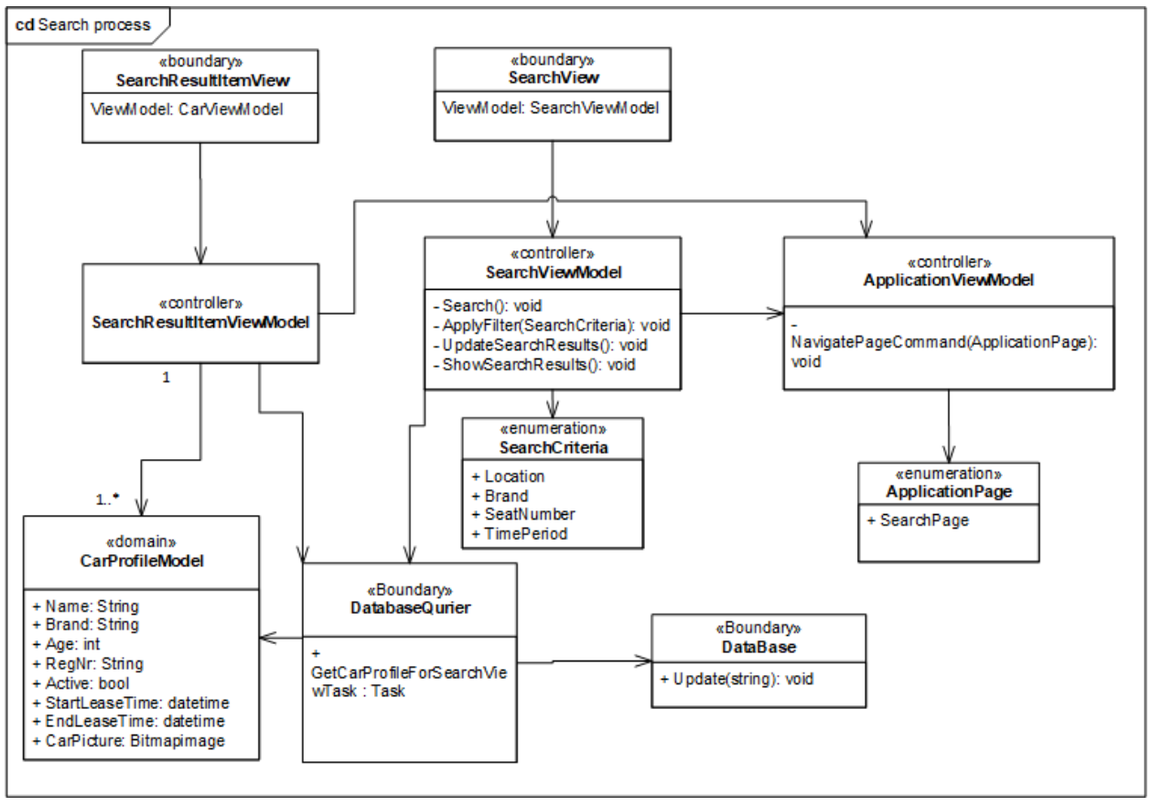
Figur 11.11: Her ses klassediagrammet for tilføjelse af bil til en brugerprofil samt redigering af en bilprofil.



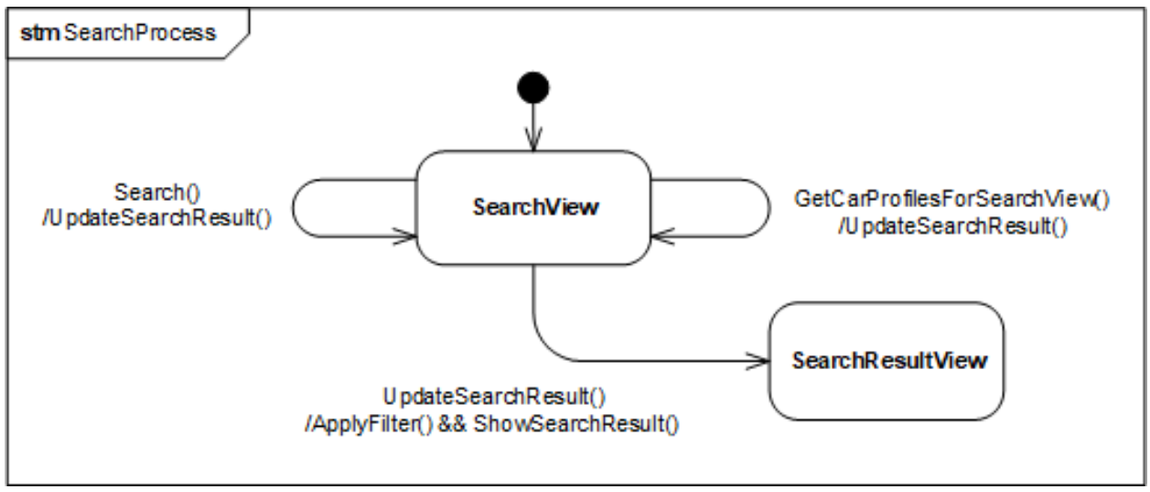
Figur 11.12: Her ses tilstandsdiagram for tilføjelse samt redigering af bil til en brugerprofil.

11.3.3 Søgning

Brugeren navigerer til søgningssiden og angiver kriterier for biludlejningen (tidsinterval, bilmærke, lokation mv.). Systemet vil løbende hente bilprofiler fra databasen, der passer til brugerens kriterier.



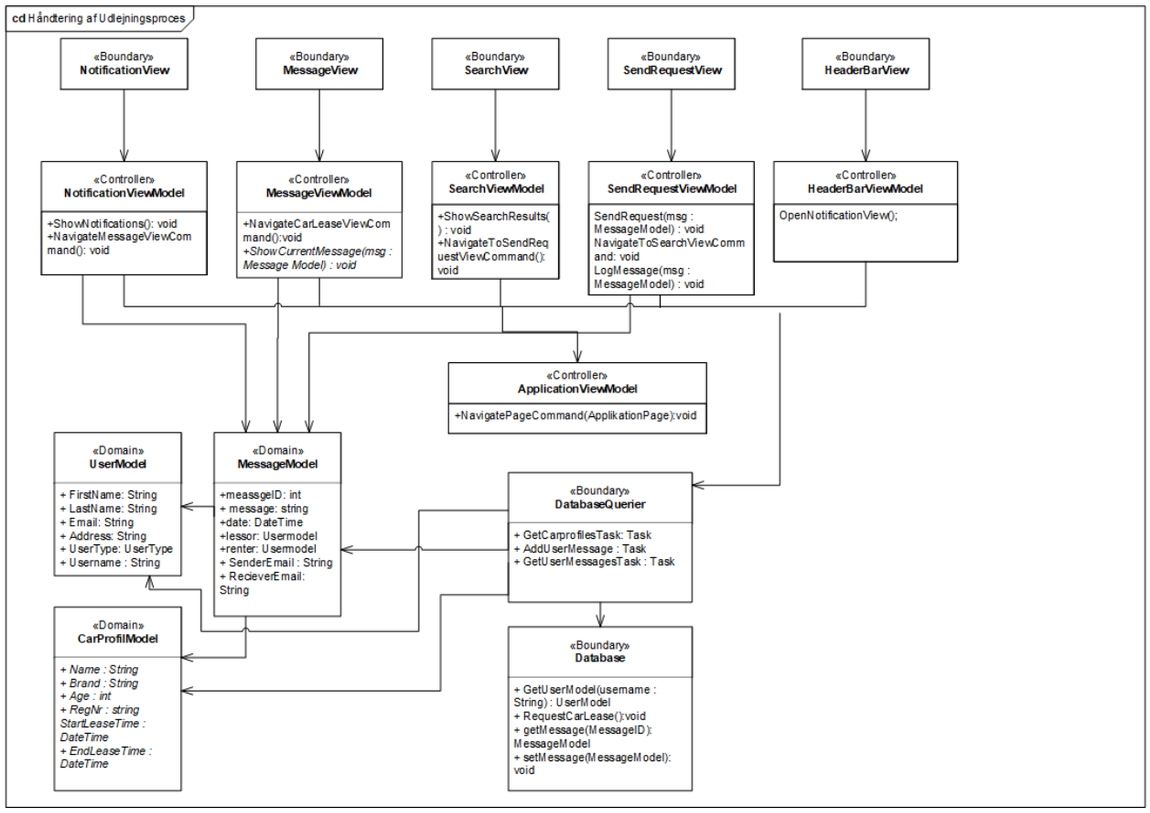
Figur 11.13: Klassediagram for søgning efter biler til leje.



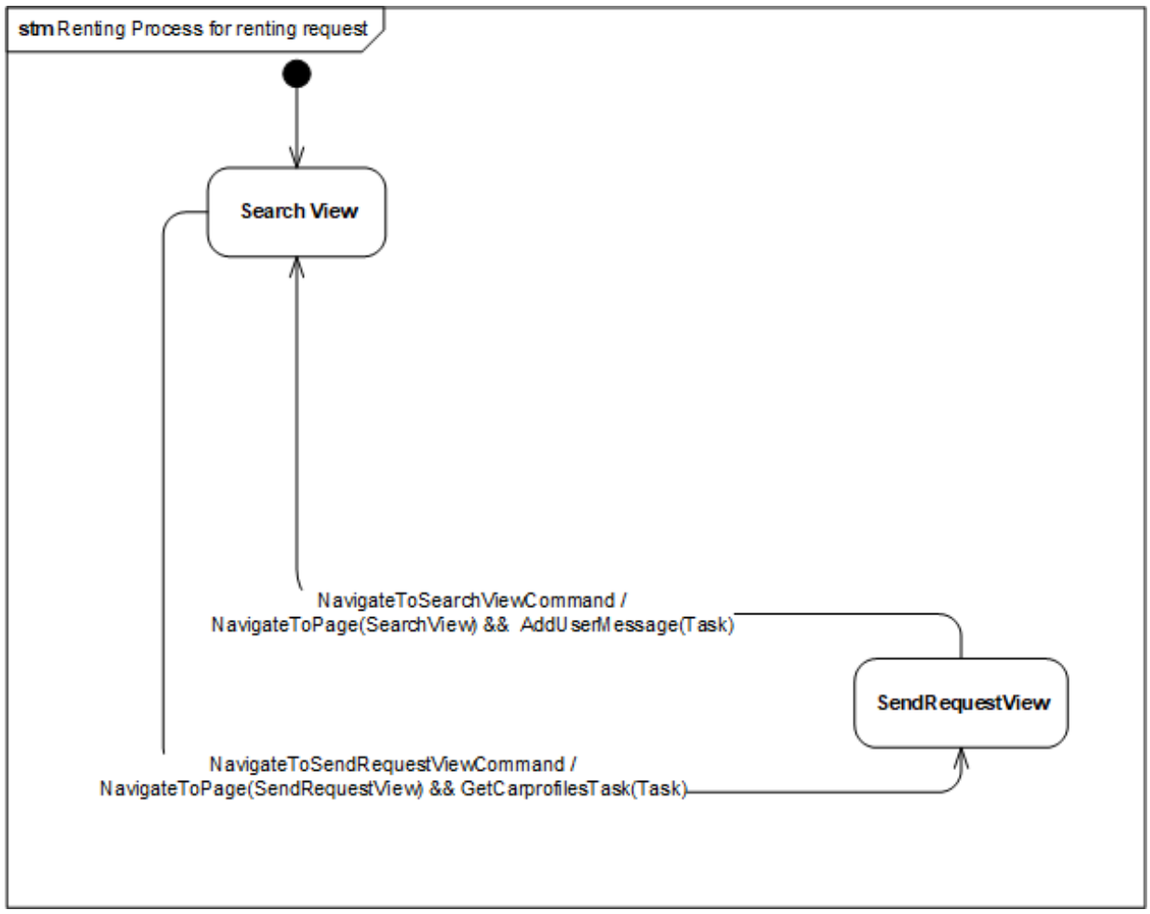
Figur 11.14: Tilstandsdiagram for søgning efter biler til leje.

11.3.4 Håndtering af udlejningsproces

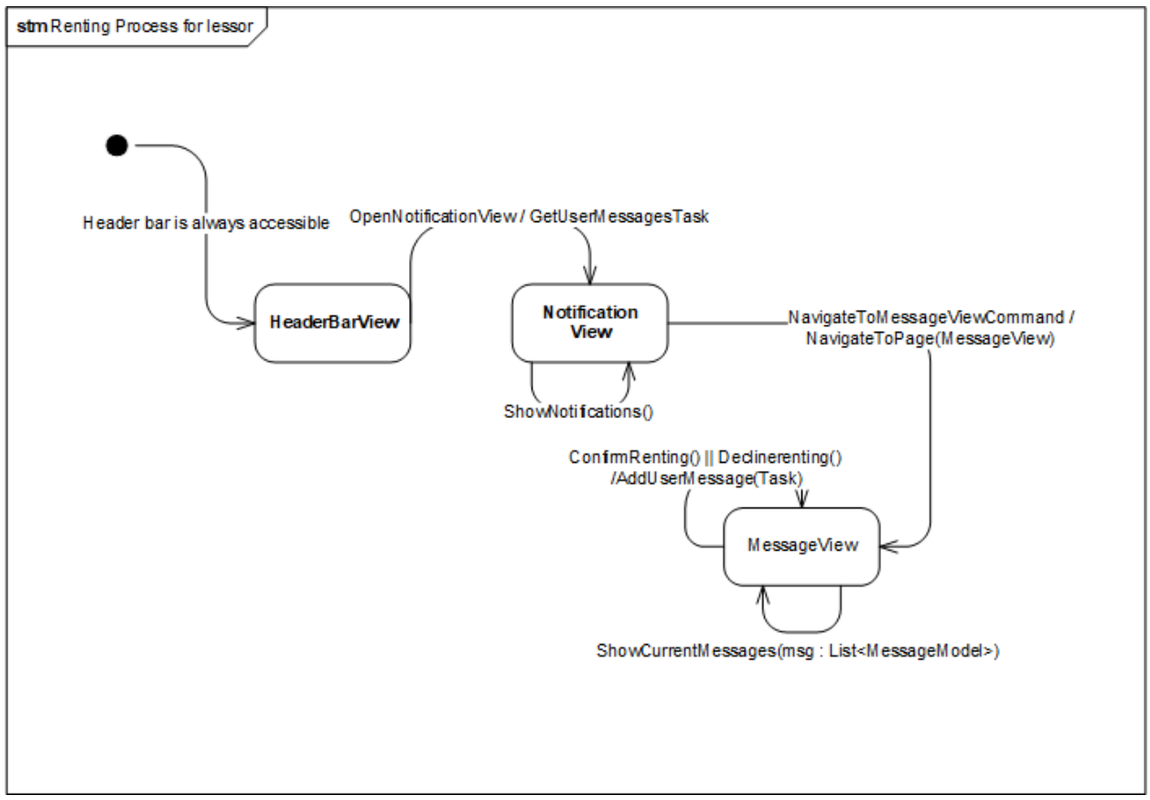
Brugeren har nu fundet den bilprofil, som ønskes at lejes. Der navigeres til et vindue, hvor brugeren kan sende en besked til udlejeren. Udlejeren vil nu modtage en besked, som kan findes i NotifikationView'et, som findes i Headerbaren. Hvis brugeren vælger en besked, navigeres der til et MessageView, som viser det fulde indhold af beskeden. Udlejeren kan enten godkende eller afvise anmodningen. Hvis anmodningen godkendes, udleveres udlejeres oplysninger til brugeren.



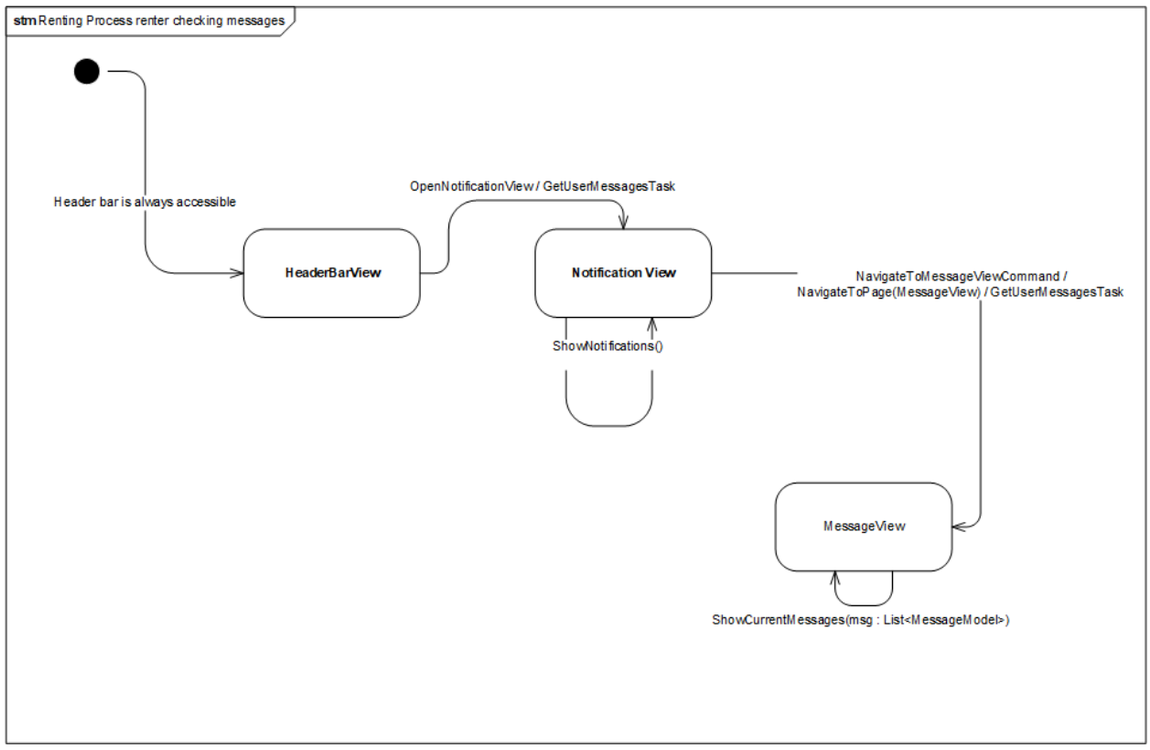
Figur 11.15: Klassediagram for håndtering af udlejningsprocessen.



Figur 11.16: Tilstandsdiagram for håndtering af udlejningsprocessen, når lejeren ønsker at leje en bil.



Figur 11.17: Tilstandsdiagram for håndtering af udlejningsprocessen, når udlejeren tjekker sine notifikationer eller beskeder og giver svar til lejeren.



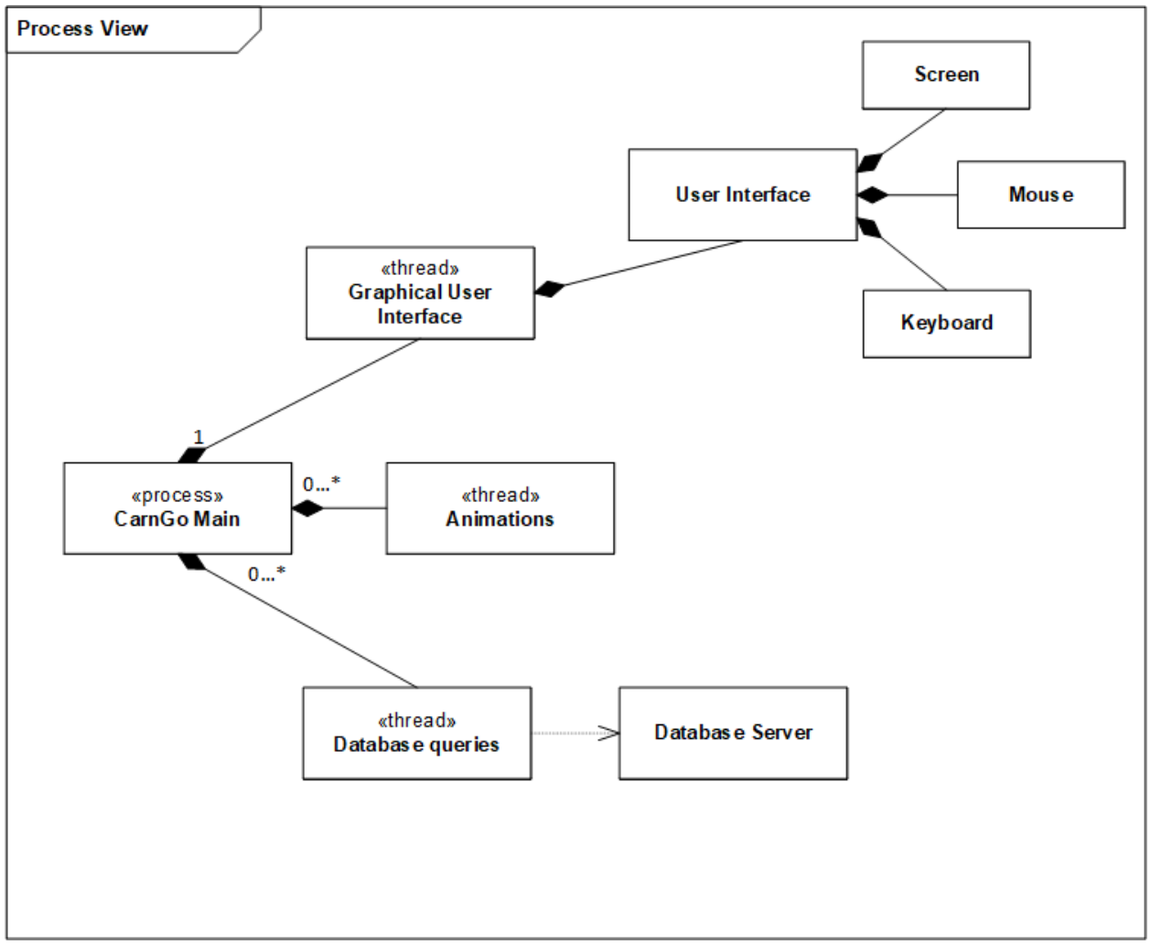
Figur 11.18: Tilstandsdiagram for håndtering af udlejningsprocessen, når lejeren får respons tilbage fra udlejeren.

Se afsnit **2.3 Logical View** i bilag **Arkitektur** for mere information om logical view.

11.4 Process View

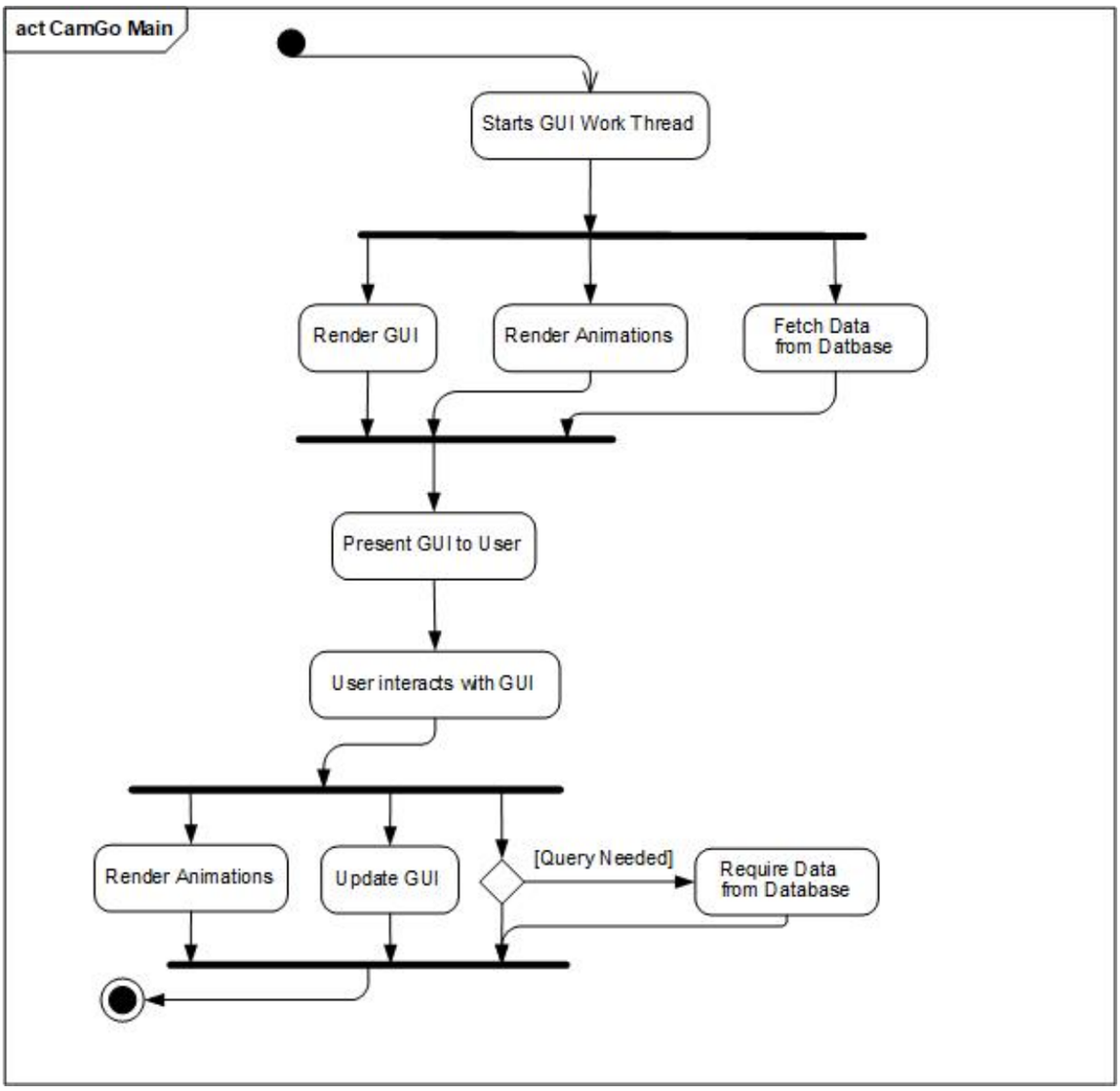
Dette view beskriver systemets dynamiske aspekter. Her angives systemprocesserne, og hvordan de kommunikerer med hinanden (med fokus på runtimeadfærd). Figur 11.19 illustrerer procesnedbrydningen af systemet. Her angives aspekter for programmet, som er trådkontrolleret. Trådtyperne kan inddeles i tre kategorier:

- **Graphical User Interface**: Dette er GUI-tråden, som render og præsenterer alle de grafiske elementer for brugeren, samt modtager og behandler inputs. Det er vigtigt, at GUI-tråden ikke forstyrres fra andre operationer, fx database queries, således brugeroplevelsen er optimal.
- **Animations**: Alle animationer til den grafiske brugeroverflade udregnes og behandles på en separat tråd. Det samles med GUI-tråden til sidst og præsenteres for brugeren.
- **Database queries**: Alle database queries er asynkrone og afvikles på en separat tråd. Queries bruges til at hente information, som skal præsenteres eller anvendes af GUI-tråden.



Figur 11.19: Dette diagram viser de processor, der bliver anvendt i systemet, samt med hvilke tråde der bliver anvendt til de forskellige processor

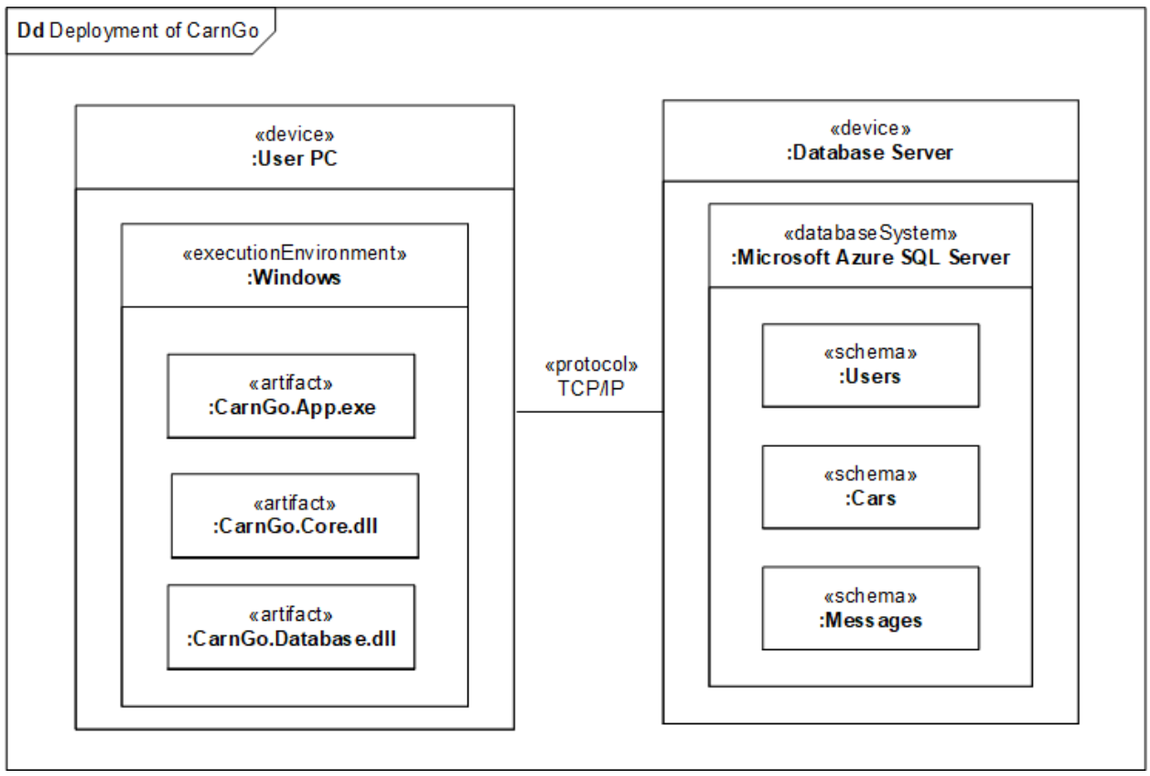
De abstrakte trådkontrollerede aspekter og hvordan de afvikles illustreres i et aktivitetsdiagram (Se figur 11.20). Her ses det, hvordan at de grafiske elementer og database queries sker samtidigt. Den grafiske brugeroverflade vil løbende blive indlæst og præsenteret til brugeren. Hvis der er data fra queries, som skal præsenteres til brugeren, vil det også blive indlæst på den grafiske brugeroverflade, men det vil være uafhængigt af GUI-tråden. Se afsnit **2.4 Process View** i bilag **Arkitektur** for mere information om process view.



Figur 11.20: Aktivitetsdiagram for hovedprocessen for CarnGo

11.5 Physical View

I dette afsnit beskrives det fysiske standpunkt af systemet. Her beskrives de software artifakter i det fysiske lag, såvel som de fysiske forbindelser mellem komponenter. Deployment diagrammet viser de to lag, applikationen opererer i, og hvordan hardware- og softwarekomponenterne interagerer med hinanden - her præsenteres hardwareprocesserne og placering af softwareelementer på pågældende hardware.

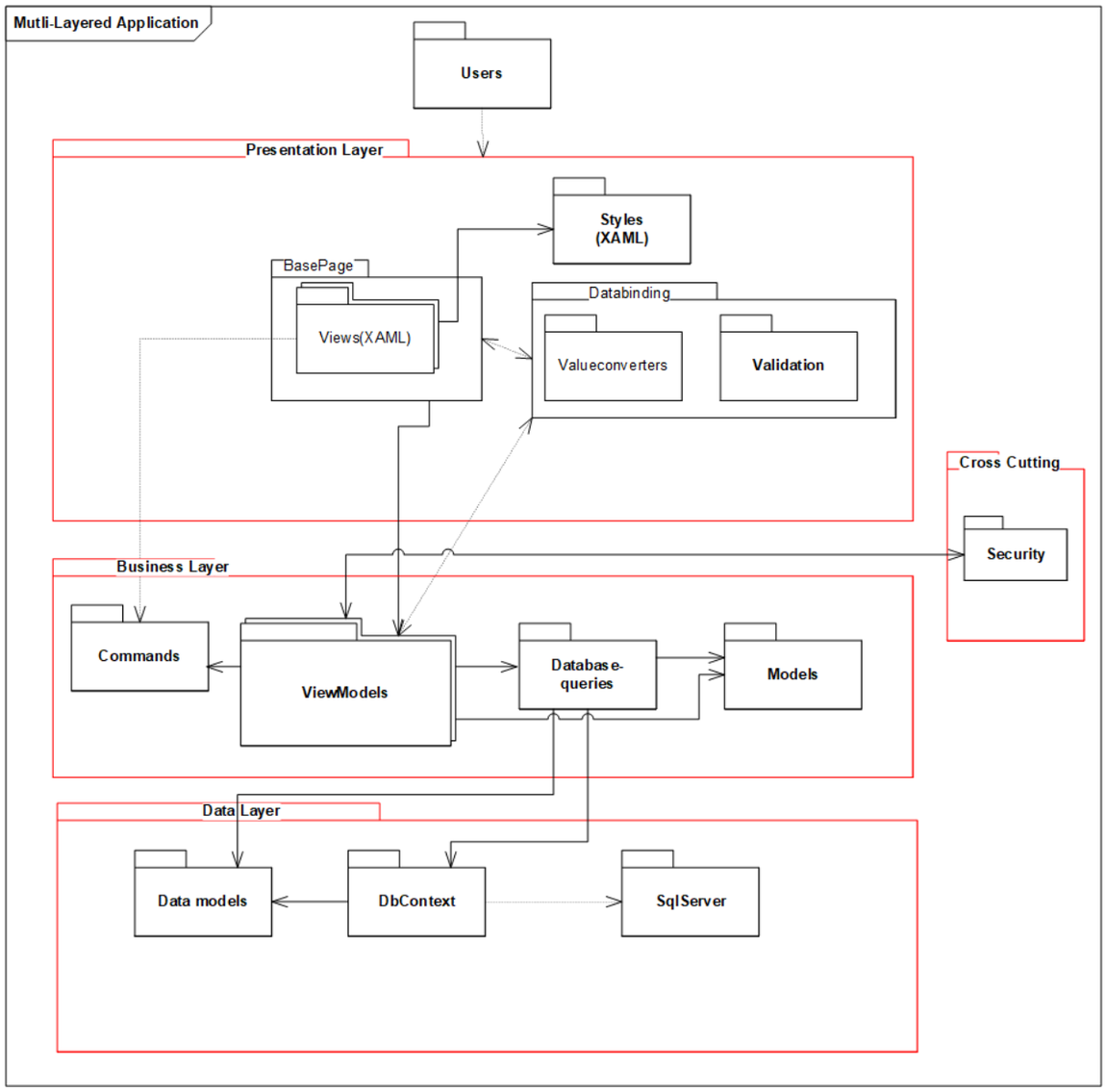


Figur 11.21: Deployment View

Af figur 11.21 ses det, at der er to fysiske komponenter i systemet, hvor det ene er brugerens PC, der kører med et Windows styresystem, og det andet er en database server. Den type af arkitektur er oftest betegnet som Client-Server-arkitektur eller en 2-tier arkitektur [29]. Dette er den simpleste form for arkitektur, men også den mest skrøbelige. Den er dog valgt, da den gør udviklingen hurtigere og mindre avanceret, hvilket passer godt til dette system.
Hos brugeren køres WPF-applikationen, CarnGo.App.exe, der bruger CarnGo.Core og CarnGo.Database bibliotekerne. Grunden til denne client arkitektur er valgt er som tidligere beskrevet på grund af lavere kobling mellem komponenterne, og det gør systemet nemmere at udvide, hvis det engang er nødvendigt.
Til host af database serveren er valgt en Microsoft Azure SQL Server, der giver en cloud løsning til host af ens SQL Server. På denne server skal der være en måde at opbevare information omkring brugere, biler og beskeder. Se afsnit **2.6 Physical View** i bilag **Arkitektur** for en detaljeret beskrivelse.

11.6 Development View

Formålet med dette view er at få overblik over hele applikation. Her illustreres de forskellige lag for applikation og tilhørende moduler og deres forbindelser. Figur 11.22 viser et Model Package diagram, som viser koblingerne mellem lagene og de interne komponenter. Se afsnit **2.5 Development View** i bilag **Arkitektur**.



Figur 11.22: Model diagram for Multi-Layered Applikation. En stiplet linje indikerer en forbindelse, som skabes gennem WPF frameworket - eksempelvis databinding.

11.7 Security View

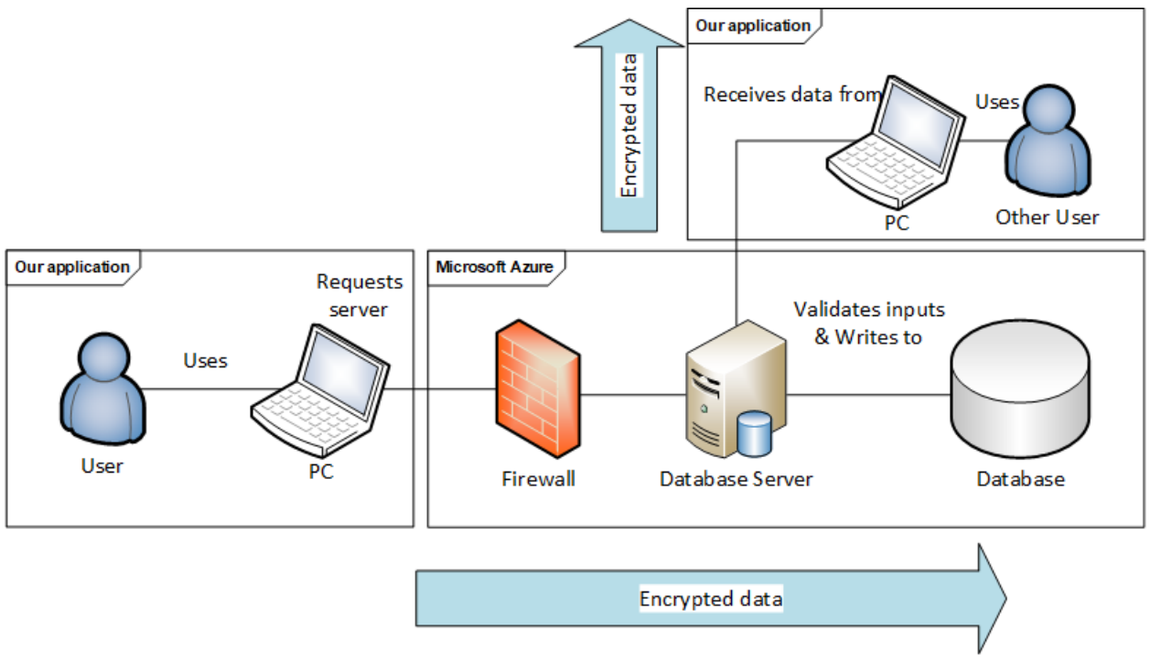
Systemet har en del fortrolige og personfølsomme oplysninger. Det er derfor essentielt, at det fastslås, hvilke oplysninger der opbevares, hvordan det skal behandles, samt hvilke trusler der findes, og hvordan det kan løses. I dette afsnit beskrives, hvordan systemet kun er tilgængeligt for dem, der har tilladelse, og hvad som gøres for at beskytte brugerens oplysninger mod uautoriseret manipulation.

11.7.1 Koncepter om brugersikkerhed

Denne applikation håndterer sensibel data om brugerne i form af navne, adresser, registreringsnummer på biler osv., så det er vigtigt at opretholde en form for sikkerhed omkring brugernes konti. Den nemmeste måde at få adgang til disse oplysninger er gennem brugerkontiens adgangskode, og derfor tages der nogle foranstaltninger omkring denne.

- Brugerens adgangskode holdes ikke i ren tekst i hukommelsen af programmet
- Brugeren er påduttet en adgangskodesikkerhed ved, at adgangskoden skal indeholde både tal og bogstaver
- Brugeren er påduttet en adgangskodesikkerhed ved, at adgangskoden skal være længere end 6 karakterer.

Der udvikles ikke selv et system til opbevaringen af data, men i stedet anvendes en Microsoft Azure SQL database. Derved fås en bedre sikkerhed mod både ondsindede angreb og for opbevaring af brugernes data[30].
Når en bruger skal ændre, tilføje eller slette information, så foregår det også over en sikker https-forbindelse, der sender krypteret data til databaseserveren[31]. I den tidlige arkitektur er der lagt op til det simpleste system, der giver værdi for brugeren. Dette afspejles i figur 11.23.

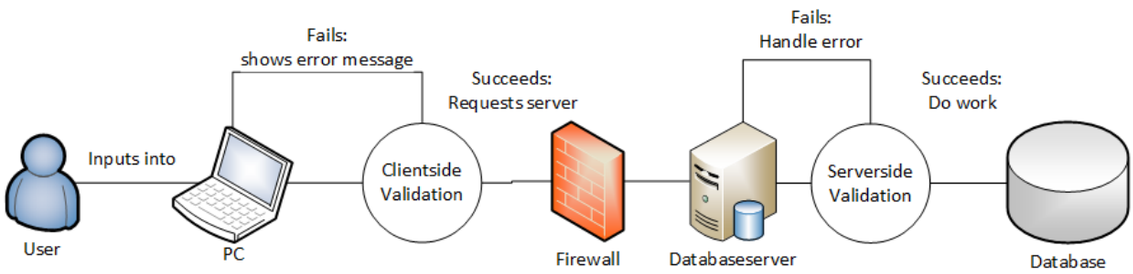


Figur 11.23: Security diagram for simpel bruger-til-server interaktion

Denne type arkitektur anvendes i de tidlige stadier, da den er simplest at udvikle på, men den er dårlig i et produktionsmiljø, da hvis en ting fejler, så fejler hele systemet.

11.7.2 Koncepter om validering

Når man har brugerinputs i sin applikation, skal disse valideres for at forhindre eventuelle trusler i form af ondsindede brugerinputs. Dette kan ske hos klienten (altså i applikationen), på serveren eller på begge. Validering hos klienten aflaster serveren og giver derved bedre serverperformance. Derfor valideres e-mail om, det er en valid e-mail og password om det opfylder kravene på applikationen (clientside). Når dette er gjort sendes bruger inputs til serveren, hvor de valideres inden videre behandling. Valideringerne kan ses visualiseret i figur 11.24.



Figur 11.24: Datavalideringsdiagram for clientside og serverside valideringer

Der anvendes en ekstern server til database og eventuel webserver, og derfor er det altså muligt at delegere flere valideringsopgaver til serveren, hvilket gør, at brugerens applikation får bedre performance. Se afsnit **2.7 Security View** i bilag **Arkitektur** for mere information om security view.

12 Software Design

I dette afsnit beskrives de overvejelser, valg og udviklingsprocesser, der er foretaget under design af applikationen CarnGo.

Applikationen består af to separate delsystemer: En SQL database og en WPF-applikation. Designet af grænsefladen for WPF-applikationen er baseret på Wireframet og prædefineret krav i afsnit **6 User Stories** og **7 Ikke-funktionellekrav** i bilag **Kravspecifikation**. Udviklingen af databasen tager udgangspunkt i de valg foretaget i analysen for valg af databehandlingsplatform og arkitekturen angivet i afsnit **3 Database beskrivelse** bilag **Arkitektur**. For den fulde dokumentation se bilag **Software Design**.

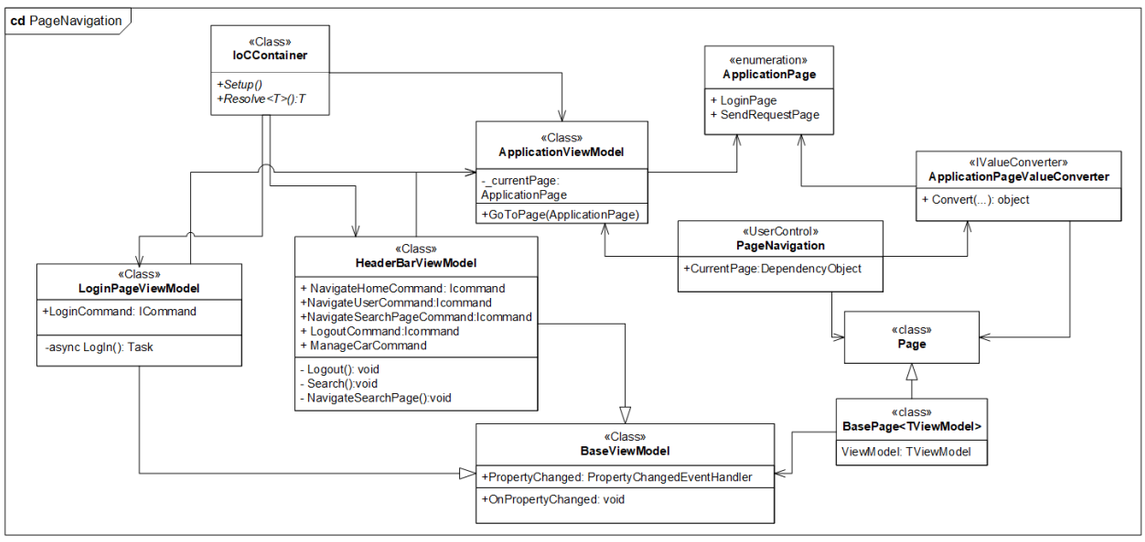
12.1 WPF-applikation

Applikation er baseret på arkitekturmønstret Model-View-ViewModel, som er mønsteret ofte anvendt ved brug af WPF. Fordelene og ulemperne ved brug af MVVM er allerede beskrevet i de forrige afsnit og vil ikke blive uddybet. Der er valgt en ViewFirst tilgang til designet, hvor Viewet laves først, og derudfra laves præsentationslogikken i ViewModel og herefter businesslogikken i modellaget. I de følgende afsnit beskrives mønsterets anvendelse i applikation, samt de generelle design overvejelser for Views og ViewModels.

12.1.1 Page Navigation

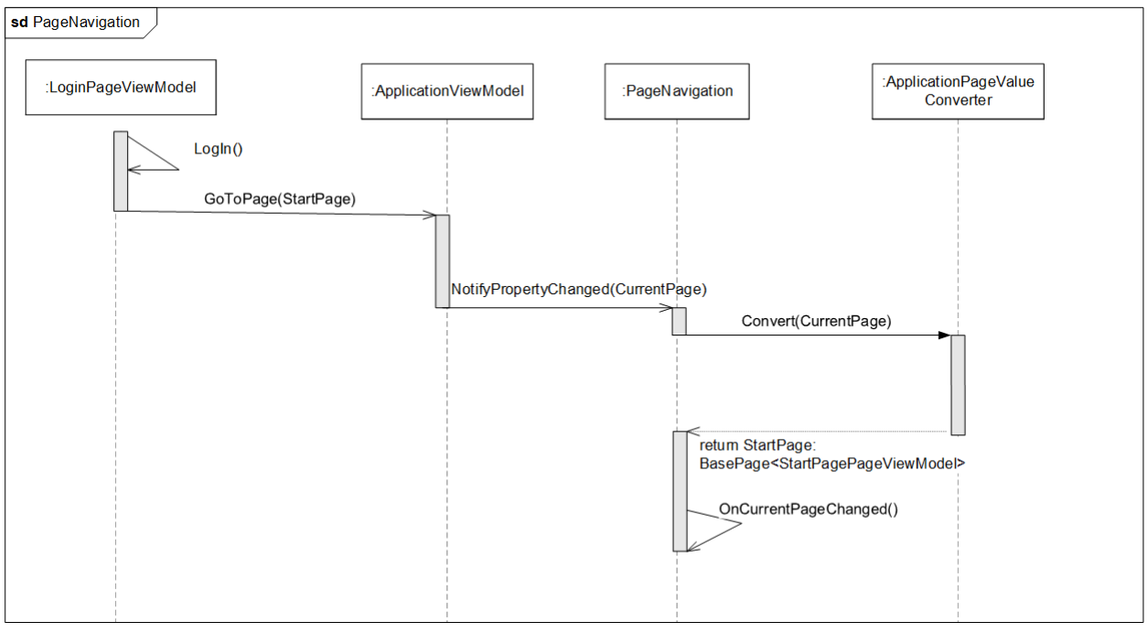
En afgørende faktor for systemet er at kunne navigere mellem Views og ViewModels runtime og stadig opretholde den lave kobling i mellem. Hvis hvert ViewModel selv havde til ansvar for at skifte mellem View, så skulle den have kendskab til hvert View, den kunne skifte til. Dette strider både imod MVVM-arkitekturen og skaber også høj kobling i ViewModellerne. Derfor udliciteres dette ansvar til klassen ApplicationViewModel.

ApplicationViewModel består af en Usermodel (instans af den bruger som er logget ind), og en CurrentPage property, som er defineret som en ApplicationPage enumerationen, der kan mappes en-til-en med et View. CurrentPage anvendes så til at konvertere til de konkrete View-instanser i ApplicationPageValueConverterklassen. Figur 12.1 viser et klassediagram for navigationsdesignet. ApplicationViewModellen er statisk og laves som en singleton, da alle sideskift er afhængige af hinanden (Man kan ikke vise to sider af gangen)[32].



Figur 12.1: Her ses klassediagrammet for PageNavigation, hvor man ser de forskellige klasser involveret, når en side skal skiftes ud med en anden. Kun to ViewModels er inkluderet i diagrammet, men alle ViewModeller har adgang til den statiske instans af ApplicationViewModel.

Nedenfor ses et sekvensdiagram, som beskriver skiftet fra LoginView til StartPage. Her anvendes ViewModellens singleton instans af ApplicationViewModel til at skifte side med 'GoToPage'-metoden.



Figur 12.2: Her ses sekvensdiagrammet for PageNavigation, hvor Login skifter til StartPage Viewet.

12.1.2 Headerbar

Design af View - Headerbar
Headerbaren er placeret i toppen af applikationen og kan findes på samtlige sider, hvor en bruger er logget ind. Det vil sige, at Login- og Oprettelses-siden er undtagelserne for dette. Det er også gennem den, at man som bruger navigerer gennem applikationen ved brug af navigationsredskaber (knapper m.v.). Grænsefladen for Viewet tager udgangspunkt i figur 12.3

Figur 12.3: Headerbar WireFrame

Der angives navigationsegenskaberne for Headerbaren i Wireframet, som blev introduceret i Arkitekturen (Se bilag **Wireframe**).

Design af ViewModel - Headerbar
Arkitekturen af Headerbaren er tæt tilknyttet til ApplicationPageViewModel. Hver knap tilknyttes en command, som anvender ApplicationPage til at skifte side ved aktivering. Søgebaren er eventbaseret og aktiveres, når brugeren indtaster en streng og trykker på søg (fx keyboard aktivering). Eventet sendes til SearchView, og der navigeres til selvsamme side.

Et andet vigtigt element i Headerbaren er notifikationsknappen, som ved aktivering vises som en popup-menu i figur 12.4. Menuens indhold er udliciteret til et separat View og ViewModel.

12.1.3 Beskeder & Notifikationer

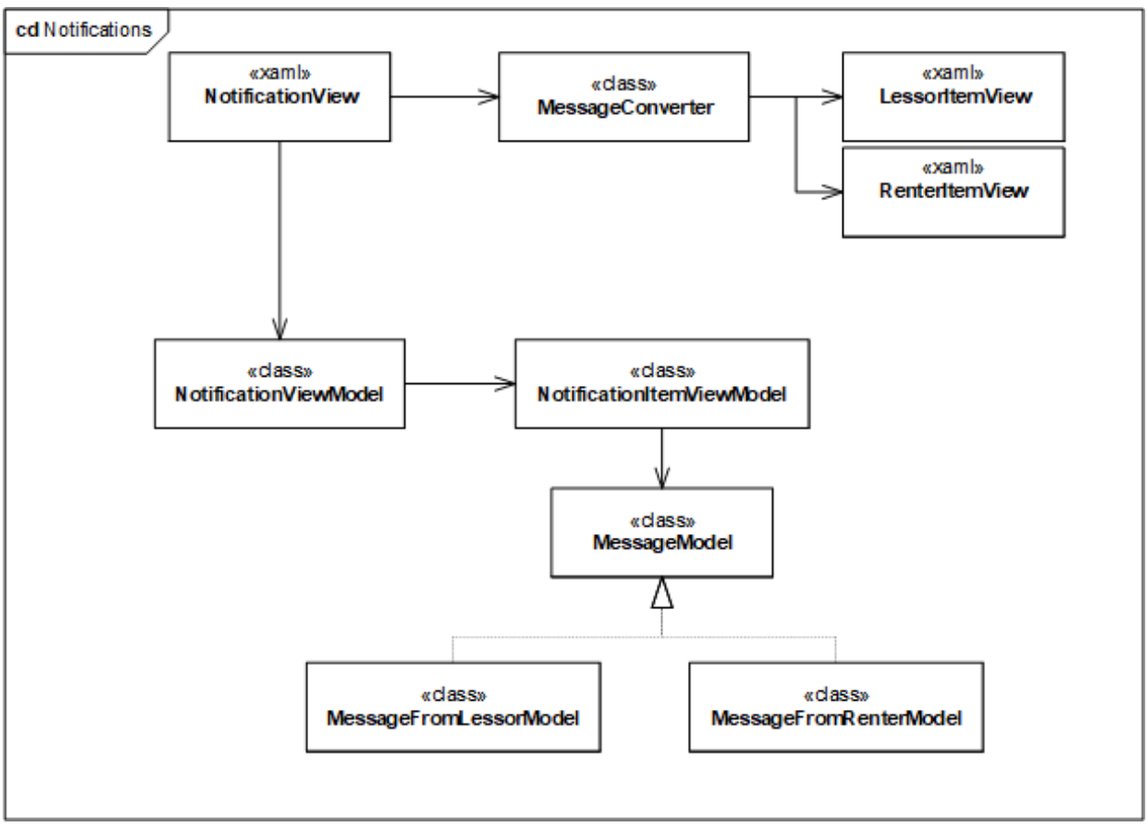
Design af View - Notification
NotificationView udfoldes ved aktivering i Headerbaren. Brugeren præsenteres for de 10 eller færre nyeste notifikationer. En notifikation indeholder kun de vigtigste informationer, da det ikke er muligt at vise alt indhold for en besked. Den visuelle repræsentation af vinduet kan ses i figur 12.4. NotificationView'et kan ses som en beskedbeholder, hvor der kan opstå forskellige typer notifikation beskeder. Det er derfor essentielt, at det er let at tilføje nye typer beskeder/notifikationer. Der er to typer beskeder defineret for systemet:

1. Lejer anmoder om bil
2. Udlejer respons til anmodning

Figur 12.4: WireFrame for Notifikationsvinduet

Design af ViewModel - Notification
En bruger vil modtage notifikationer, når der interageres med applikationen, specifikt når der enten lejes eller udlejes en bil. Hvert type besked vises forskelligt, og der skal derfor differentieres runtime mellem beskedtyperne. Hertil bruges en **value converter**, som konverterer indholdet af en besked til et 'WPF item', så det kan vises i Notifikationsvinduet (En notifikationsbesked er repræsenteret som en blok inde i vinduet, se figur 12.4). Hvert besked har dets eget View og Model, og der angives typen af notifikationen i Modellen,

herved kan converteren differentiere runtime mellem beskederne. Hvis systemet udvides, og der tilføjes flere typer beskeder, kan de let tilføjes ved at oprette et nyt View og tilhørende Model. Figur 12.5 illustrerer strukturen af designet.



Figur 12.5: Klassediagram for notifikationer

Design af View - Message
I forlængelse med Notifikationsvinduet skal brugeren have adgang til alle modtagne beskeder. Hvert element i Notifikationsvinduet navigerer til Messagevinduet, hvor brugeren præsenteres for den fulde besked og alle tidligere beskeder (designet af vinduet ses i figur 12.6):

1. Den venstre side af vinduet viser alle brugerens beskeder
2. Den højre side viser indholdet af den besked, som brugeren har valgt
    - Lejer anmodning: Her vises beskeden fra lejer. Udlejeren kan godkende eller afvise anmodningen i Notifikationsvinduet.
    - Udlejer respons: Her vises beskeden fra udlejeren, samt om denne har godkendt bilanmodningen.

Figur 12.6: WireFrame for Message

Design af ViewModel - Message
Brugerens beskeder vil være være et 1:1 forhold med Notifikationsvinduet, her vises de samme beskeder. Beskederne er dog ikke statiske, de loades igen, når brugeren skifter mellem Notifikationsvinduet og Messagevinduet. Idet brugeren vælger en besked/notifikation, sendes et event til Messagevinduet, hvor eventet indeholder den specifikke besked. Alt indhold for beskeden vises på den højre side af View'et (se figur 12.6).

12.1.4 Login

Design af View - Login
Loginsiden er det første brugeren ser, når applikationen eksekveres. Det overordnede WireFrame viser navigationsegenskaberne for vinduet (se bilag **Wireframe**). Fra Loginsiden skal der udelukkende kunne navigeres til Sign Up, da det ville kompromittere programmet (der kan være data gemt fra tidligere brug af applikationen, fx en tidligere bruger, og det fjernes ved login). Brugeren kan vælge at registrere sig i systemet, hvis personen ikke har en konto. Figur 12.7 viser designet for vinduet.

Figur 12.7: Wireframe af login

Design af ViewModel - Login
Der er en del sikkerhedsforanstaltninger, som skal overholdes for login og konti (tager udgangspunkt i kravene defineret i Security View). Brugerens kodeord og e-mail valideres. Koden er fortrolig information, som ikke skal vises i applikation. For at realisere dette omdannes koden til typen securestring[33].

12.1.5 Søgning

Design af View - Search
Betegnelsen 'Search' dækker over den del af applikationen, som giver lejere mulighed for at finde og leje biler. Der er dedikeret en side i applikationen til søgefunktionaliteten, og designet af denne tager udgangspunkt i en WireFrame, som kan ses i figur 12.8. Det fremgår, at siden er delt i to. Som bruger skal man have mulighed for at se hvilke biler, der er tilgængelige, dvs. de biler som tidligere er blevet registreret i systemet af en udlejer, og som nu kan lejes. Dette er vist til højre i figur 12.8, hvor der kan scrolles gennem udvalgte biler. Hver af de adskilte enheder repræsenterer én bil og indeholder således et billede, nogle relevante informationer vedrørende bilen og en knap, som giver brugeren mulighed for at leje den.

Til venstre i figur 12.8 er en række søgekriterier, som giver brugeren mulighed for at filtrere resultaterne. Dermed behøver brugeren ikke at gennemgå samtlige biler, men kun de relevante biler, der opfylder de indtastede kriterier. Der foretages ikke database forespørgsler hver gang, der foretages en søgning. Det vurderes, at det er en bedre løsning at hente et vist antal biler fra databasen ad gangen, for så efterfølgende at filtrere dem, når der foretages en søgning. Med denne fremgangsmåde mindskes interaktionen med databasen, og derved er der mindre overhead og risiko for fejl.

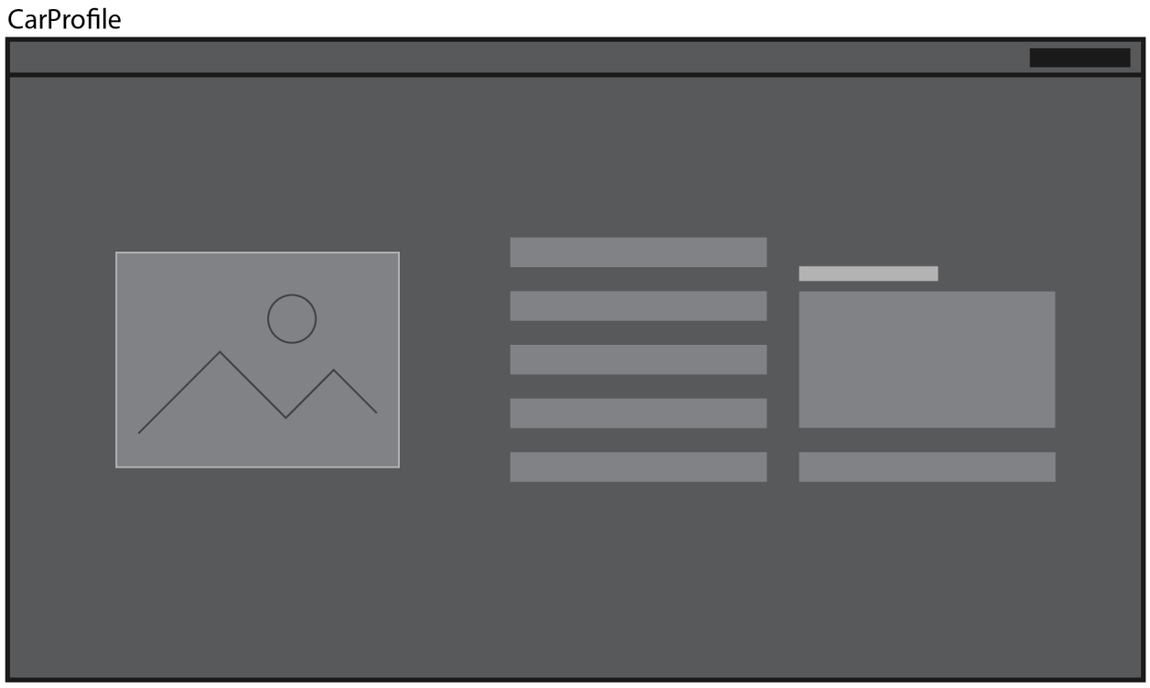
Figur 12.8: Search Page WireFrame

Design af ViewModel - Search
Der kan navigeres til Søgevinduet på to måder: Enten ved at trykke på 'Find Car' knappen eller søgebaren, som begge befinder sig i Headerbaren. Benyttelsen af søgebaren medfører at der navigeres til Search og foretages en filtrering efter lokation på baggrund af den indtastede streng.

Når brugeren har indtastet oplysninger i venstre side og foretager en søgning, bliver kriteriet tilføjet til en liste med den værdi, som der skal sammenlignes med. Hvis der ikke er udfyldt nogen kriterier, vises alle bilerne, og ellers skal alle kriterierne tjekkes og være opfyldt for en given bil, før den inkluderes i View'et. Denne fremgangsmåde har fordelen, at hvert enkelt søgekriterie kan holdes simpelt og testes separat.[34]

12.1.6 Bilprofil

Design af View - CarLease
I dette afsnit beskrives designet for siden CarLease, og der tages udgangspunkt i en Wireframe, som kan ses i figur 12.9. Siden giver brugere mulighed for at sætte en bil til leje. Forudsætningen for, at man kan tilgå siden er, at man er registreret som udlejer, og først da bliver det muligt for brugeren at navigere til siden. Her har man mulighed for at indtaste oplysninger om bilen (bilmærke, model, fabriksår m.v.). Derudover kan der uploades et billede af bilen, og brugeren skal angive det tidsinterval, hvor bilen er tilgængelig for lejere.

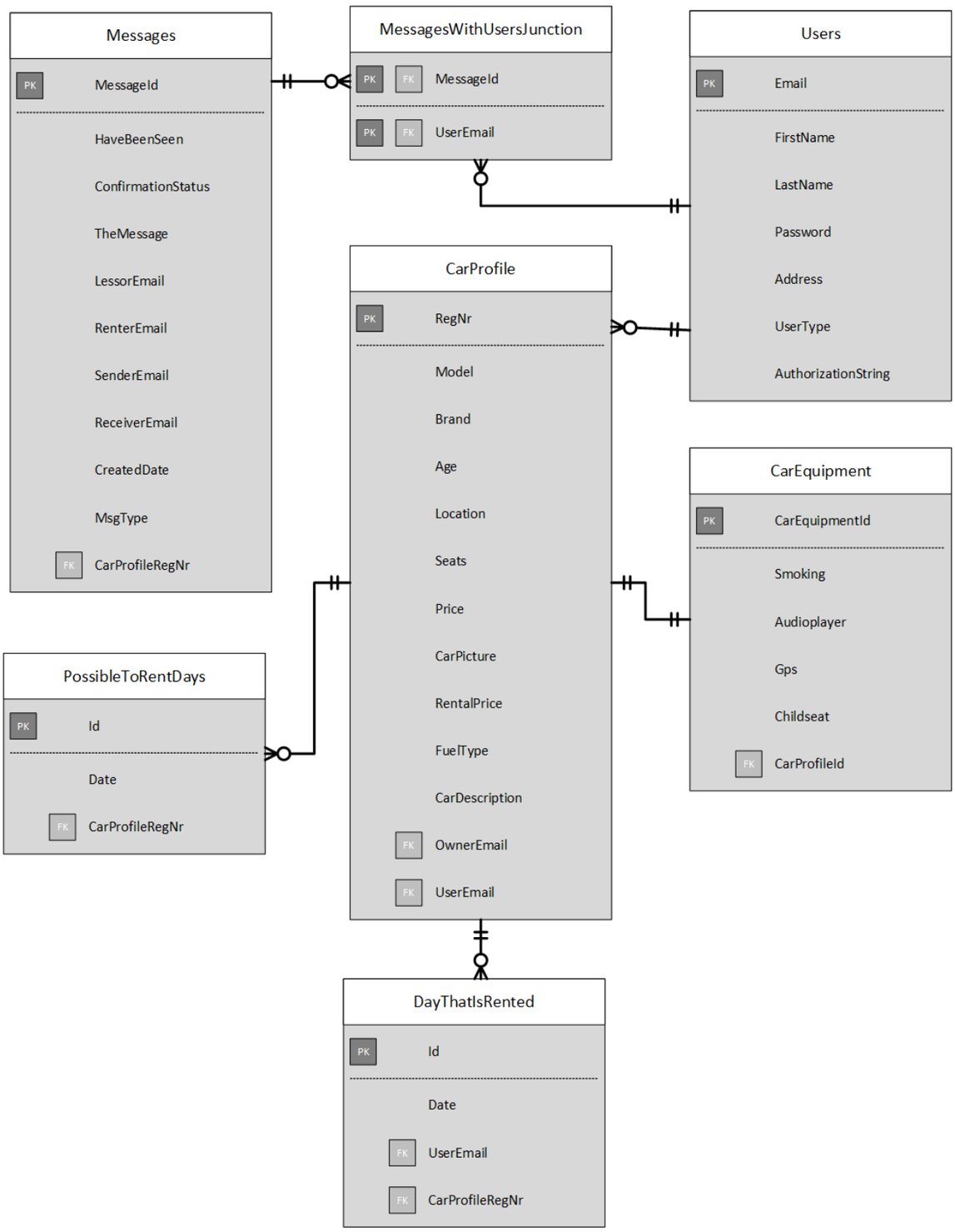


Figur 12.9: Car Profile WireFrame

Design af ViewModel - Car Profile
Sidens funktionalitet varierer alt efter om udlejer registrerer bilen for første gang, eller blot opdaterer oplysninger for en allerede registreret bil. Som udgangspunkt er der mulighed for at gemme en ny bil med nye oplysninger. Hvis der allerede er registreret en bil, giver siden mulighed for at vælge en bil fra en liste, redigere dennes oplysninger, oprette en ny bil profil samt slette en eksisterende profil. Hvis udlejer vil uploade et billede af bilen åbnes en ny dialogboks, hvor man kan vælge et billede fra sin PC. Billedet lagres i applikationen, men konverteres til en streng før det lagres i databasen.

12.2 Database

Databasen designes som en relationel database, der skal opbevare applikationens data og være fælles opbevaringstjeneste for alle brugere i systemet. Det skal være muligt at tilgå data og samtidigt opretholde datas integritet. Systemets data- og informationsstruktur defineres i E/R diagrammet set nedenfor med brug af Crows Foot notation[35]. For en nærmere beskrivelse af ER diagrammet se bilaget **Software Design** afsnit **13 Design af CarnGo Database**.

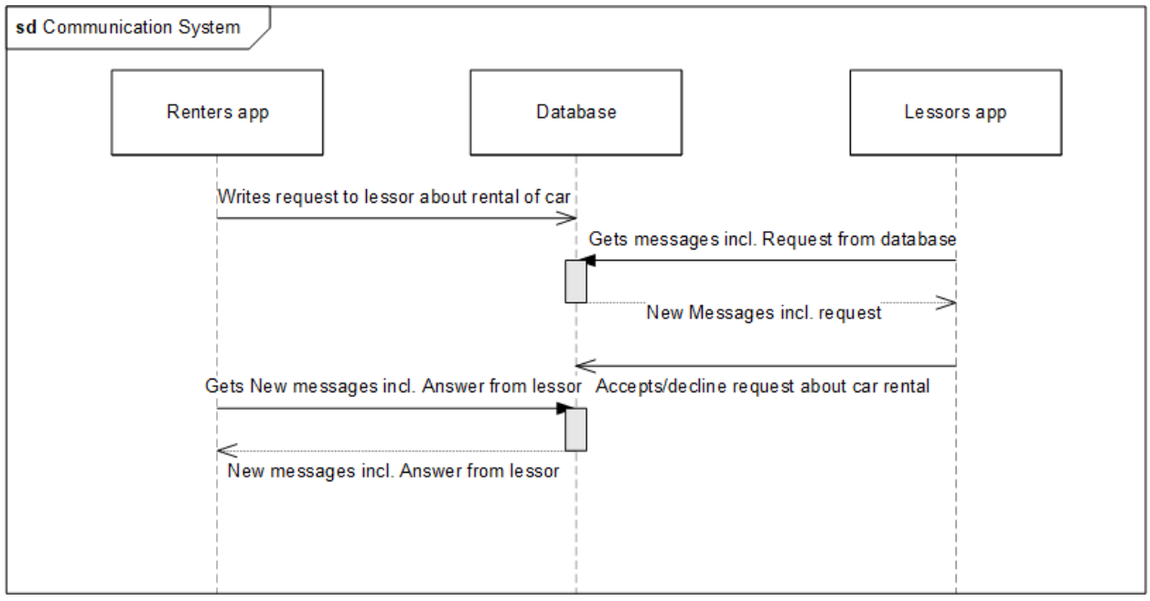


Figur 12.10: Entity-Relationship diagram for CarnGo databasen. E/R diagrammet er designet ud fra Crows Foot notation

12.2.1 Kommunikationssystem

En vigtig ting i CarnGo applikationen er, at der kan kommunikeres mellem en lejer og udlejer. Til dette formål designes et kommunikationssystem. Kommunikationen starter altid fra lejers applikation, hvor han fra SendrequestViewet sender en anmodning om leje af bil til udlejer. Denne anmodning/besked består af en fra og til dato, i det tidsinterval bilen ønskes udlejet samt en tekstbesked. Denne besked bliver sammen med de dage bilen ønskes udlejet gemt i databasen med en reference til udlejer. Udlejer kan ved hjælp af denne reference til beskeden gå ind og læse beskeden ved at trykke på notifikations ikonet i headerbaren. Han kan vælge enten at acceptere eller afvise anmodning, enten i drop-down boksen eller i messageview. Ved at acceptere bliver beskeden ændret i databasen til at være accepteret, og udlejer får en besked/notifikation om dette, når han trykker på notifikationsikonet i headerbar. I tilfældet

hvor udlejer afviser anmodningen vil dagene lejer har ønsket at leje bilen blive slettet fra databasen igen, så en anden lejer har chancen for at udleje bilen i tidsperioden. Når dette gøres, vil beskeden/anmodningen sendt til udlejer også ændres til at være afvist, og lejer vil blive notificeret om dette ved at hente beskeden i databasen, hvilket gøres automatisk med et tryk på notifikationsikonet i headerbar. Figur 12.11 viser et sekvensdiagram for kommunikationssystemet.



Figur 12.11: Her ses kommunikationssystemet, hvor kommunikation mellem lejer og udlejer gennem databasen vises.

Det var meningen, at notifikationer skulle opdateres automatisk skulle opdateres i headerbar, men dette blev ikke gjort af flere grunde. Der henvises til bilag **13.2.3 Problemer med notifikationer** for en beskrivelse af hvordan problemet blev forsøgt løst, og hvorfor det stadig ikke virkede.

13 Implementering

13.1 Introduktion

I følgende afsnit præsenteres de overvejelser, der er blevet gjort i sammenhæng med implementeringen af systemet. Udover dette beskrives de overvejelser, der er blevet gjort i forhold til patterns under implementeringen af systemet.

SOLID:
Under implementeringen af systemet har gruppen haft forsøgt at følge SOLID principperne så vidt som muligt[36]. Der er dog visse udfordringer forbundet med dette i sammenhæng med en MVVM-arkitektur. Et eksempel på dette er, at ViewModeller ofte indeholder meget logik, de er svære at opdele, og de er ofte offer for ændringer, da ændringer i Viewet ofte skal afspejles i ViewModel.

13.2 Frameworks

I dette afsnit beskrives de vigtigste frameworks, der er anvendt til implementeringen af systemet. Se afsnit **3 FrameWorks** i bilag **Implementering** for mere information om de anvendte frameworks.

Entity Framework Core
Entity Framework Core er en Object Relation Mapper, der også bliver brugt til projektets data access[37]. Her anvendes C# modeller til at skabe relationer og entities i Azure SQL Databasen og et context objekt til at kunne tilgå databasen. Til disse modeller laves CRUD-operationer, så der opnås persistering af data[38].

Prism
Fra frameworket Prism anvendes der primært en DelegateCommand-klasse, der gør det muligt at definere og binde Commands til XAML viewet[39].
EventAggregator anvendes også som den primære form for kommunikation mellem ViewModels. Den gør, at man kan sende beskeder mellem ViewModels, som f.eks. en søgning fra HeaderBarViewModel, hvor strengen sendes til SearchViewModel.

Unity
Unity er systemets IoC Container, der sørger for registrering, oprettelse og livstid for de forskellige klasser i systemet[40].
Den anvendes i sammenspil med EventAggregator for at sikre, at alle ViewModels får den samme instans.

13.3 Patterns

I dette afsnit beskrives de patterns, der er anvendt i projektet. Se afsnit **4 Patterns** i bilag **Arkitektur** for mere om patterns.

Factory Patterns:
Factory method pattern går ud på at adskille dannelsen af objekter fra bruget af dem gennem et metode kald[41]. Dette koncept bruges i sammenspil med Unity, hvor man f.eks. kan kalde IoCContainer.Resolve<HeaderBarViewModel>() og få lavet et objekt, hvor der er taget hånd om alle afhængigheder af objektet.
Derudover kan man som i Abstract Factories registrere objekter i forhold til et interface, hvilket gør, at det er nemt at udskifte en konkret implementering med en anden, så længe det samme interface er opfyldt[42].

Det bedste eksempel på dette er i BasePage klassen, der står for initialisering af den side, der hører til siden. Dette ses i linje 8 af listing 1.

Listing 1: FrameworkElementAnimation AnimateIn funktion

```
1  public class BasePage<TViewModel> : BasePage
2      where TViewModel : BaseViewModel
3  {
4      private TViewModel _pageViewModel;
5
6      public BasePage()
7      {
8          PageViewModel = IoCContainer.Resolve<TViewModel>();
9      }
10
11     public TViewModel PageViewModel
12     {
13         get => _pageViewModel;
14         set
15         {
16             if (_pageViewModel == value)
17                 return;
18             _pageViewModel = value;
19             DataContext = _pageViewModel;
20         }
21     }
22 }
```

Nu hvor de vigtigste aspekter af implementering er beskrevet, så er det værd at kigge på, hvordan implementeringen af systemet er blevet testet.

14 Software test

I det følgende afsnit beskrives hvordan CarnGo testes, herunder hvilke redskaber, frameworks og principper, som er anvendt i praksis. For at sikre at applikation opfører sig som forventet og opfylder kravene, er det essentielt at teste softwaren grundigt. Software test skal i denne sammenhæng forstås som, at man systematisk og objektivt verificerer en software enheds tilstand og opførsel baseret på nogle objektive og prædefinerede kriterier eller krav. Med systematisk og objektivt menes der, at testen skal være målbar, og at det skal være muligt for en anden at gentage testen med identiske resultater. At kriterierne er objektive betyder, at der ikke er plads til fortolkning af resultatet. For flere informationer vedrørende applikationens software tests henvises til bilag **TestDokument**.

14.1 Unit Testing

En unit test er et automatiseret stykke kode, som kalder den enhed, der skal testes[43]. Den grundlæggende idé er at påvirke enheden for herefter at tjekke nogle antagelser omkring slutresultatet. Unit tests anvendes i vidt omfang til at teste applikationen CarnGo - hver enkelt test case er struktureret på samme vis og består af tre trin:

- *Arrange* objekter, opret og opsæt dem efter behov
- *Act* på et objekt, udfør aktiviteten som skal testes
- *Assert* at resultatet er som forventet

Til at skrive testene er der anvendt frameworks kaldet NUnit og NSubstitute[44]. Dette gør test cases lettere at skrive og vedligeholde, og de kan hurtigt eksekveres ved tryk på en knap. For mere om unit tests se afsnit 3 i bilag **TestDokument**

14.2 Continuous Integration

For at undgå problemer med integration af softwaren er der i projektet anvendt en praksis kaldet Continous Integration[45]. Det indebærer, at teamets medlemmer integrerer ofte, og at hver integration verificeres af et automatisk build. Github er anvendt som remote repository, og Jenkins er anvendt som build server[46][47]. Det er konfigureret på en sådan vis, at Jenkins bliver triggered, når der pushes til Git serveren. Dermed startes et build job, hvor projektet bygges og testkøres, og først herefter kan der igen pulles fra serveren. Denne fremgangsmåde sikrer, at integrationsfejl opdages og korrigeres tidligt i processen. Samtidig er man aldrig langt fra et fungerende build, og som udvikler får man løbende feedback i forhold til kodens og testens kvalitet.

14.3 Coverage

For at sikre kvaliteten af testene er der løbende blevet set på, hvor meget af applikationens kode, der er dækket af tests. Dette betegnes Test Coverage, og det giver et billede af, hvilke områder af programmet, der bliver anvendt i applikationens test suite[48]. Coverage giver et kvantitativt mål for, hvor godt testene dækker koden, og denne viden kan anvendes til systematisk at udvide eller tilføje test cases, indtil alt applikationskoden er omfattet. Der skelnes mellem forskellige former for Coverage, og der er ulemper forbundet med dem alle. Generelt er fuld coverage heller ikke ensbetydende med ingen fejl, men det anvendes alligevel i projektet, da det er et simpelt og objektivt mål for kvaliteten af tests. Til at beregne Coverage for CarnGo er der anvendt værktøjet dotCover, og der sigtes efter 100% Coverage. Den er dog endt med at være væsentligt lavere med blot 38%[49].

14.4 ZOMBIE

ZOMBIE er et akronym, der fungerer som en guide for, hvad der skal testes i projektet, og de overvejelser der er gjort undervejs[50]. **ZOM**-delen af ordet indikerer, at man altid bevæger sig fra simple til mere komplekse testscenarier. **BIE**-delen af ordet står derimod for de overvejelser, man bør gøre sig undervejs, mens man sigter efter simplicitet i testscenarierne. **B** står for Boundaries og indikerer, at man bør teste grænseværdier. **I** står for Interfaces, da man bør teste alle metoder, events, kaldte interfaces og afhængigheder. Endelig står **E** for Exceptional Behaviour, da man bør tage højde for og teste uforventet opførsel, timeouts og fejlhåndtering.

Nedenfor ses en test case, som anvender NUnit og NSubstitute frameworks[51][52]. Den tester, at der navigeres til startsiden, når bruger logger ind på applikationen. Enheden der testes (uut) er en instans af LoginPageViewModel. Først sikres det, at brugerens e-mail og password består valideringen (Arrange), og herefter kaldes en Login Command (Act). Endelig tjekkes der på, at der modtages et funktionskald af GoToPage() med StartPage som argument (Assert). I forhold til ZOMBIE testes der på metoder/afhængigheder (**I**).

```
[Test]
public async Task Login_LoginNavigatesToLogin_GoToStartpage()
{
    // Arrange
    _fakeEmailValidator.Validate(Arg.Any<string>()).Returns(true);
    _fakePasswordValidator.Validate(Arg.Any<SecureString>())
    .Returns(true);

    // Act
    await _uut.Login();

    // Assert
    _fakeApplication.Received()
    .GoToPage(Arg.Is<ApplicationPage>
    (page => page == ApplicationPage.StartPage));
}
```

For mere om test kvalitet se afsnit 5 i bilag **TestDokument**

14.5 Integrationstest

De vigtigste dele af applikationen er integrationstestet. Det vil sige, at der testes interaktioner og afhængigheder mellem flere moduler[53]. Frem for at lave udtømmende tests for hele applikationen, er det valgt at fokusere på de dele af systemet, hvor der er størst sandsynlighed for, at fejl kan forekomme. Derfor er der fokuseret på at teste modellaget og data access laget sammen, herunder database, queries og konvertering af data. Der er taget udgangspunkt i en Bottom up integrationsstrategi, hvilket vil sige at de moduler eller klasser, som har færrest afhængigheder testes sammen først, og herefter udvides der med flere moduler, indtil alle er omfattet af tests[54]. Denne strategi anvendes fordi, at det vurderes, at fejl ofte vil opstå i de laveste niveauer af CarnGo applikationen, da der ofte skal interageres med databasen og konverteres data. Med denne fremgangsmåde bliver fejlene opdaget og korrigeret tidligt i processen.

15 Accepttestspecifikation

15.1 Introduktion

Accepttesten for både de funktionelle og ikke-funktionelle krav er udført i overensstemmelse med kravspecifikationen set i bilag **Kravspecifikation** og er foretaget d. 24/05/2019. Resultaterne er opsummeret i tabellen 15.1, som er vist nedenfor. I dette afsnit beskrives hvilke tests, som ikke blev godkendt. For den fulde accepttestspecifikation henvises til bilag **Acccepttestspecifikation**.

15.2 Microsoft UI Automation

Microsoft UI Automation (UIA) er et API, som gør det muligt at identificere og manipulerere applikationens brugergrænsefladeelementer og -objekter[55]. Dette værktøj anvendes til at automatisere tests for både funktionelle og ikke-funktionelle krav. Her angives specifikke instruktioner til, hvordan brugergrænsefladen skal manipuleres, baseret på accepttestspecifikationerne angivet for hver User Story og enkelte ikke-funktionelle krav. Her opstilles forsikringer om grænsefladens respons til brugeren. Hvis UIA ikke registrerer respons i overensstemmelse med forsikringerne, vil testene fejle. Denne form for GUI-testing tester hele systemet og interagerer med brugergrænsefladen svarende til, hvad en testperson ville gøre manuelt. Dette giver en effektiv form for feedback, idet hver gang der opdateres i applikationens kildekode, vil accepttestene blive kørt. Hvis de fejler, notificeres gruppemedlemmerne med det samme, som beskrevet i afsnit **4 Continuous Integration** i bilag **TestDokument**.

Strukturen af applikationen gør, at det ikke altid er muligt at hævde på de konkrete grafiske elementer. Her må der anvendes White Box testning, hvor der assertes på elementer i businesslogikken frem for brugergrænsefladen.

Nedenfor ses resultaterne for accepttesten for de funktionelle krav. Det fremgår, at de fleste af kravene er godkendt, hvilket kan bekræftes af CodedUI tests. Der er ni tests, som er godkendt ud af ti. Fjernelse af brugerprofil er ikke implementeret, hvorfor den ikke er godkendt. Det vurderes, at denne funktionalitet har mindst værdi for de personas, som er beskrevet tidligere. Derfor har den haft lavest prioritet.

Testgrundlag	**Beskrivelse**	**Status**
US1	Oprettelse af brugerprofil	Godkendt
US2	Redigering af brugerprofil	Godkendt
US3	Oprettelse af bilprofil	Godkendt
US4	Fjernelse af bilprofil	Godkendt
US5	Redigering af bilprofil	Godkendt
US6	Søgning efter biler til leje	Godkendt
US7	Anmodning om biludleje	Godkendt
US8	Notifikation til lejer om svar på anmodning om leje af bil	Godkendt
US9	Login på applikation	Godkendt
US10	Fjernelse af brugerprofil	Ikke godkendt

Tabel 15.1: Accepttest resultat for funktionelle krav

15.3 Ikke-funktionelle krav

Accepttesten for de ikke-funktionelle krav er blevet foretaget med en blanding af Coded UI tests og visuelle tests. Det er muligt at bruge værktøjet til at analysere applikationens grænseflade ifh. til font, farvekoder og visuel performancetid. Et eksempel er vist nedenfor, hvor CodedUI anvendes til at bekræfte, at den anvendte fonttype er af typen "Robotic". Resultaterne for accepttesten af de ikke-funktionelle krav kan ses i bilag **Accepttest**.

```
[TestMethod]
public void Robotic_Typografi()
{
    // Act
    this.UIMap.ClickLoginEmailBox();
    Keyboard.SendKeys("car@renter");
    this.UIMap.ClickLoginPasswordBox();
    Keyboard.SendKeys("123asd");
    this.UIMap.ClickLoginButton();
    this.UIMap.ClickOwnerButton();
    this.UIMap.ClickUserInformation();
    this.UIMap.ClickOnuserInfo();

    //Assert
    this.UIMap.AssertOnRoboticFont();
}
```

16 Diskussion

Under 4. semester er gruppen blevet introduceret til User Stories, der er en anden måde at lave funktionelle krav på. Gruppen har derfor måttet tage en beslutning om, hvorledes de funktionelle krav skulle skrives som User Stories eller Use Cases, som gruppen havde brugt de tre forrige semestre. Gruppen besluttede at prøve kræfter med User Stories af flere årsager. En af grundene var at User Stories lægger mere op til agil udvikling. Dette passer bedre med den måde gruppen arbejder på, da der bliver brugt SCRUM, men måske endnu vigtigere fordi User Stories er mere abstrakte end Use Cases, hvilke gjorde det meget lettere for gruppen at ændre ting undervejs i projektforløbet.

I projektet har kunderne heller ikke nogle specifikke krav til, hvordan applikationen skal fungere, da gruppen selv er kunden. Dette gjorde, at gruppen kom i gang med at implementere funktionalitet i applikationen hurtigere, og gruppen undgik hermed at lave alt for mange ændringer i kravspecifikationen i løbet af projektet. Det kan dog diskuteres, om dette er optimalt, da det stærkeste element ved User Stories er den stærke relation til målgruppen/brugeren. Dette kerneelement forsvinder, idet gruppen selv er kunden, og anvendelsen af User Stories kommer til at minde om Use Cases. De tekniske aspekter af Use Cases er dog abstraheret, hvilken gør det muligt at skabe en mere iterativ og agil arbejdesproces - det blev sjældent oplevet, at det var nødvendigt at ændre i kravene.

En af fordelene ved anvendelsen af User Stories er, at de er lette at specificere tests ud fra. Der er udviklet flere software værktøjer og sprog såsom Gherkins, der muliggør dette ved at mappe User Stories direkte til test cases[56]. Disse muligheder blev også undersøgt i forbindelse med CarnGo, men det viste sig at være svært at foretage Black Box testing for systemet, blandt andet på grund af brugergrænsefladen og den udbredte anvendelse af input validering[57]. Derfor blev der afsøgt andre muligheder, herunder GUI-testing med Coded UI. Ud fra et testperspektiv kunne der altså lige så godt have været anvendt Use Cases.

17 Konklusion

I dette afsnit konkluderes der på Projektformuleringen og kravene til projektet, og det beskrives kort, hvad der er lykkedes, og hvad der ikke er. Desuden resultaterne konkluderes der på udviklingsprocessen og de erfaringer, der er opnået i løbet af denne.

Der er implementeret en biludlejningsapplikation kaldet CarnGo. Den har en grafisk brugergrænseflade, som giver brugerne mulighed for at leje og udleje biler. Brugeren skal først logge ind med en e-mail og password på 'Login'-siden. Disse gemmes i en relationel database, og hvis man ikke er registreret i forvejen, har man mulighed for at navigere til en 'Register'-side og oprette sig. Når brugeren er logget ind, kan der tilgås en 'Edit User'-side med personlige informationer, hvor man kan indtaste navn og adresse eller uploade et billede. Der skelnes ikke mellem, om man er lejer eller udlejer ved oprettelsen, men denne oplysning kan angives her. Som udlejer kan der tilgås en yderligere side, hvor man kan oprette en bilprofil ved at indtaste relevante oplysninger og uploade et billede af bilen, der herefter gemmes i databasen. Ved at navigere til 'Find Car'-siden kan man se de biler, der er gemt i databasen, og som dermed kan lejes. Det er muligt at søge mellem bilerne ud fra lokation, bilmærke, antal sæder, afhentnings- og leveringstidspunkt eller en kombination af disse. Når brugeren har fundet en bil, som det ønskes at leje, bliver man ved tryk på en 'Rent'-knap ledt til en 'Rent Car'-side. Her er der angivet yderligere oplysninger om bilen såsom registreringsnummer, pris, brændstof og en kort beskrivelse af bilen. Der vælges nu udlejningsperiode, og der skrives en besked til udlejer. Brugere kan se deres beskeder ved at trykke på notifikationer i Headerbaren. Herfra kan der navigeres til en 'Message'-side, hvor alle ens indgående og udgående beskeder er vist, og her kan udlejer vælge at godkende eller afvise en anmodning fra lejer. Endelig kan brugeren logge ud af applikationen, hvilket fører tilbage til 'Login'-siden.

Det kan konkluderes, at store dele af applikationen er implementeret og virker efter hensigten. Databasen gemmer informationer om brugere og biler, det er muligt at leje biler, og brugere kan kommunikere indbyrdes gennem beskedsystemet. Accepttesten afslørede, at kun én ud af 10 User Stories mangler at blive implementeret. Der har netop været fokus på at få den grundlæggende funktionalitet implementeret, og det fremgår, at dette er lykkedes. Konsekvensen har dog været, at visse ting måtte nedprioriteres. F.eks. har der ikke været fokus på sikkerhed/kryptering, eller anvendelsen af lokalitetstjenester. Der mangler således en form for regulering, da forhold såsom betaling og afhentningsted må aftales indbyrdes af brugerne.

CarnGo er lavet som en Desktop Application i WPF, og den er ikke gjort tilgængelig på flere platforme. Arkitekturen er dog tilpasset det oprindelige krav om platformsuafhængighed, og anvendelsen af MVVM bevirker, at man relativt let vil kunne udskifte View laget. Eksempelvis ville langt størstedelen af applikationen kunne genbruges uden væsentlige ændringer, hvis man udskiftede WPF med Xamarin, og dermed ville CarnGo også være tilgængelig på Android og iOS.[58] Endelig kan det konkluderes, at der er anvendt en iterativ udviklingsproces. Den har taget udgangspunkt i Scrum, og dette har bl.a. medført, at der løbende er blevet ændret i kravspecifikation og arkitektur. Overordnet har tidsestimeringen i udviklingsprocessen været succesfuld, da den vigtigste funktionalitet er implementeret i CarnGo. Der er dog stadig plads til forbedring, og den lave Coverage indikerer, at der godt kunne bruges mere tid på Unit Tests og integrationstests.

18 Fremtidigt arbejde

I denne seksjonen beskrives det fremtidige arbeidet som var planlagt hvis vi skulle videre utviklet produktet. Dette inkluderer kravene i kravspesifikasjonen som ikke har blitt implementert ennå (Se bilag **Kravspecifikation**). Samt oppfølging på eventuelle problemene som oppstår under accepttest og integrasjonstest.

18.1 Udvidelser

En av de store utvidelsene som var planlagt var å gjøre CarnGo til en webapplikasjon for å gjøre den mer tilgjengelig for brukere. Det ville også være mulig å overføre applikasjonen til en mobil plattform siden den er implementert med bruk at MVVM modellen. Andre eksempler på ting som ikke har blitt implementert som er del av MoSCoW analysen er kart/GPS funksjonaliteten som ville la leiere søke på biler i sitt nærområde samt la utleiere velge hvilket område de vil leie bilen sin ut i.

En av de andre store planlagte utvidelsene var implementere er et fullt chat system mellom brukerne. Da ville CarnGo kunne fasilitere all kommunikasjon mellom brukerne isteden for bare en mulighet for dem til å utveksle kontakt informasjon.

Det er også viktige utvidelser i forhold til sikkerheten av brukerdata som må være implementert for et eventuelt release. I applikasjonens nåværende tilstand oppbevares alt av brukerdata på databasen inkludert passord i klar tekst. I tillegg er ingen av meldingene sendt mellom brukere kryptert på noen måte når de blir lagret.

19 Bilagsoversigt

Den vedhæftede 'Bilag.zip' fil har følgende struktur.

- Accepttestspecifikation.PDF
- Afgrænsning.PDF
- Analyse.PDF
- Arkitektur.PDF
- Implementering.PDF
- Kildekode
  - CarnGo
    - CarnGo
    - CarnGo.Accepttests
    - CarnGo.Integrationtest
    - CarnGo.Test.Unit
    - Database
- Kravspecifikation.PDF
- Møder
  - Gruppemøder.PDF
  - Sprintmøder.PDF
  - Vejledermøder.PDF
- Ordliste.PDF
- Proces.PDF
- Projektformulering.PDF
- Softwaredesign.PDF
- Referencer
- TestDokument.PDF
- Wireframe.PNG

Referencer

[1] D. b. a. GoMore, „TrustPilot Reviews, indsamlet d. 5-04-2019“, 2019. side: https://dk.trustpilot.com/review/www.gomore.dk?page=3.

[2] GoMore, „GoMore“, 2019. side: https://gomore.dk/.

[3] A. Universitet, „I4PRJ4 Præsentation af vejlederstab“,

[4] Unknown, *Reviews fra GoMore*.

[5] M. Cohn, *User Story*. side: https://www.mountaingoatsoftware.com/blog/stories-epics-and-themes.

[6] *Story Map*, 2019. side: https://www.agilealliance.org/glossary/storymap/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'story*20mapping))~searchTerm~'~sort~false~sortDirection.

[7] Wikipedia, „Moscow Method“, 2019. side: https://en.wikipedia.org/wiki/MoSCoW_method.

[8] Wikiepedia, „FURPS“, 2019. side: https://en.wikipedia.org/wiki/FURPS.

[9] W3C, „Web Content Accessibility Guidelines“, 2019. side: https://www.w3.org/WAI/standards-guidelines/wcag/.

[10] Olga Okhrimenko, „How to integrate a payment gateway into a website?“, s. 10, side: https://justcoded.com/blog/how-to-integrate-a-payment-gateway-into-a-website/.

[11] E. Musk, „Paypal“, 2019. side: https://www.paypal.com/dk/home.

[12] Digitaliseringsstyrelsen, „NemID“, 2019. side: https://www.nemid.nu/dk-da/.

[13] Gdpr.dk, „GDPR regler“, 2019. side: https://gdpr.dk/.

[14] M. Fowler, „The New Methodology“, s. 1–18, 2015.

[15] P. 5, „Projektgruppens Scrum Anvendelse“, s. 2, 2019.

[16] . Projektgruppe Semesterprojekt 3, „Scrum anvendelse I PRJ3 Gruppe 7“, s. 2,

[17] . Projektgruppe 3. Semesterprojekt, „Procesdokument, 3. Semesterprojekt“, 2018.

[18] Wikipedia, „Vandfaldsmodelln“, side: https://da.wikipedia.org/wiki/Vandfaldsmodellen.

[19] P. H. Mikkelsen, „Vejledning til udviklingsprocessen for projekt 3“, åg. 3, s. 1–9, 2014. side: http://studerende.au.dk/studier/fagportaler/diplomingenioer/undervisningdiplom/projektvejledning-katrinebjerg/.

[20] Microsoft, „Model-View-ViewModel (MVVM)“, 2017. side: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm.

[21] Wikipedia, „WPF“, 2019. side: https://en.wikipedia.org/wiki/Windows_Presentation_Foundation.

[22] Microsoft, *MVVM*, 2012. side: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10).

[23] ——, „Microsoft Azure SQL Database“, 2019. side: https://azure.microsoft.com/da-dk/services/sql-database/.

[24] Wikipedia, „Visual Studio“, side: https://da.wikipedia.org/wiki/Microsoft_Visual_Studio.

[25] ——, „Relational Database“, 2019. side: https://en.wikipedia.org/wiki/Relational_database.

[26] ——, „NOSQL“, 2019. side: https://en.wikipedia.org/wiki/NoSQL.

[27] ——, „4+1 Model“, side: https://en.wikipedia.org/wiki/4%2B1_architectural_view_model.

[28] RedWeb, *Wireframe*. side: https://redweb.dk/aktuelt/hvad-er-en-wireframe-og-hvorfor-skal-du-bruge-det.

[29] Wikipedia, „Client-Server arkitektur“, side: https://en.wikipedia.org/wiki/ClientâĂŞserver_model.

[30] Microsoft, *Microsoft Azure Security*, 2019. side: https://azure.microsoft.com/da-dk/overview/security/.

[31] ——, *Database Encryption*, 2019. side: https://docs.microsoft.com/da-dk/azure/sql-database/sql-database-always-encrypted-azure-key-vault.

[32] GeeksforGeeks, *Singleton pattern*. side: https://www.geeksforgeeks.org/singleton-design-pattern-introduction/.

[33] Microsoft, „SecureString“, s. 10, 2017. side: https://docs.microsoft.com/en-us/dotnet/api/system.security.securestring?view=netframework-4.8.

[34] A. ONeill, *WPF: CollectionView Tips*, 2019. side: https://social.technet.microsoft.com/wiki/contents/articles/26673.wpf-collectionview-tips.aspx#Complicated_Filtering.

[35] Wikipedia, „E/R diagram and Crow's foot notation“, side: https://en.wikipedia.org/wiki/EntityâĂŞrelationship_model.

[36] ——, „SOLID“, 2019. side: https://en.wikipedia.org/wiki/SOLID.

[37] R. Peres, *Entity Framework Core Succinctly*, 2018. side: https://docs.microsoft.com/en-us/ef/core/.

[38] Wikipedia, „Create, read, update and delete“, 2019. side: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete.

[39] Prism Library, *Prism*. side: https://prismlibrary.github.io/docs/.

[40] Unity, *Unity*. side: https://github.com/unitycontainer/unity/.

[41] Dofactory, *Factory Method*. side: https://www.dofactory.com/net/factory-method-design-pattern.

[42] ——, *Abstract Factory*. side: https://www.dofactory.com/net/abstract-factory-design-pattern.

[43] S. T. Fundamentals, „Unit Testing“, 2019. side: http://softwaretestingfundamentals.com/unit-testing/.

[44] R. C. Martin og M. Feathers, *The Art Of Unit Testing*, Second edi. Manning Publications Co., 2014, s. 1–123.

[45] F. Jakobsen og T. Jensen, *Continuous Integration*, 2019.

[46] Github, „GitHub“, side: https://github.com/.

[47] Jenkins, „Jenkins“, side: https://jenkins.io/.

[48] S. Cornett, „Code Coverage Analysis“, s. 1–9, 2006. side: http://www.bullseye.com/coverage.html.

[49] *dotCover*, 2019. side: https://www.jetbrains.com/dotcover/?fbclid=IwAR0qx3VnjZ7ivsi6EzOmwUvPJAEc-ToI1Dvzrx3CYL5Cj5yE3aHEz1-gf0m0.

[50] *ZombieTesting*, Aarhus.

[51] NUnit, *NUnit*. side: https://nunit.org/.

[52] NSubtitute, *NSubstitute*. side: https://nsubstitute.github.io/.

[53] F. Jakobsen og T. Jensen, *Integration Test Patterns*, 2019.

[54] *Bottom Up Integration Approach*, 2019. side: http://www.professionalqa.com/bottom-up-approach.

[55] „Coded UI Documentation“, tek. rap., 2019, s. 1–9. DOI: .1037//0033-2909.I26.1.78. side: https://docs.microsoft.com/en-us/aspnet/identity/overview/.

[56] Cucumber, *Cherkin*. side: https://cucumber.io/docs/guides/overview/.

[57] GeeksforGeeks, „Black and White Box Testing“, 2019. side: https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/.

[58] Wikipedia, „Xamarin“, 2019. side: https://en.wikipedia.org/wiki/Xamarin.