Aarhus Universitet
Semesterprojekt 4
Gruppe 5

Softwaredesign

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)

Vejleder:
Jesper Michael Kristensen, Lektor

29. maj 2019

1 Versionshistorik

Ver.	Dato	Initialer	Beskrivelse
1.0	23/03/2019	MG	Tilføjet design af Headerbaren
1.1	31/03/2019	MG	Tilføjet design af UserSignUp
1.2	09/04/2019	MH	Opsætning af dokument og tilføjet introduktion
1.3	15/04/2019	TM	Tilføjet introduktion til Database og Notification
1.4	11/04/2019	TM	Tilføjet design af MessageView
1.5	26/05/2019	TM	Hurtig gennemgang af dokumentet

Indhold

2 Introduktion

Model-View-ViewModel:
I dette afsnit beskrives softwaredesignet for applikationen CarnGo. Applikationen baserer sig på et arkitektur mønster kaldet Model-View-ViewModel (MVVM), som er et ofte anvendt mønster inden for WPF. Et View er ansvarlig for UI-delen af applikationen og håndterer bruger input. Det vil altid være en relation fra view til en ViewModel, som indeholder præsentationslogikken. Viewmodels kan også ses som en specialisering af modellen, og de muliggør databinding og brugen af commands. Modellen indeholder forretningslogikken og data for applikationen, når den kører. Den grundlæggende idé med MVVM er, at modellen ikke kender til ViewModel, samt at ViewModel ikke har kendskab til Viewet. Dermed får man en lavere kobling og 'separation of concerns', som er en af de største fordele ved brug af mønsteret. En anden fordel ved brugen af MVVM er, at man får et testbart design. Hvis ViewModel ikke er afhængig af Viewet, og der ikke er kode i Viewets code-behind fil, kan der nemmere udføres automatiserede unit tests. En yderligere grund til at CarnGo anvender MVVM er, at applikationen lettere vil kunne overføres til andre platforme. Model og Viewmodel kan bygges til at køre på alle platforme, hvorimod Viewet skal tilpasses den specifikke platform. Der er således mange fordele ved brug af MVVM, og i det følgende beskrives mønsterets anvendelse i applikationen, herunder de konkrete Views og ViewModels samt generelle design overvejelser.

Database:
For at kunne opbevare alle bruger interaktioner, er det nødvendigt at have et databasesystem. Hertil designes en relationel database, som skal gøre det muligt at opbevare data og give flere klienter adgang til disse data. Databasen skal være serverbaseret, og vil bruges som et kommunikationsystem mellem brugere.

3 Side Navigation

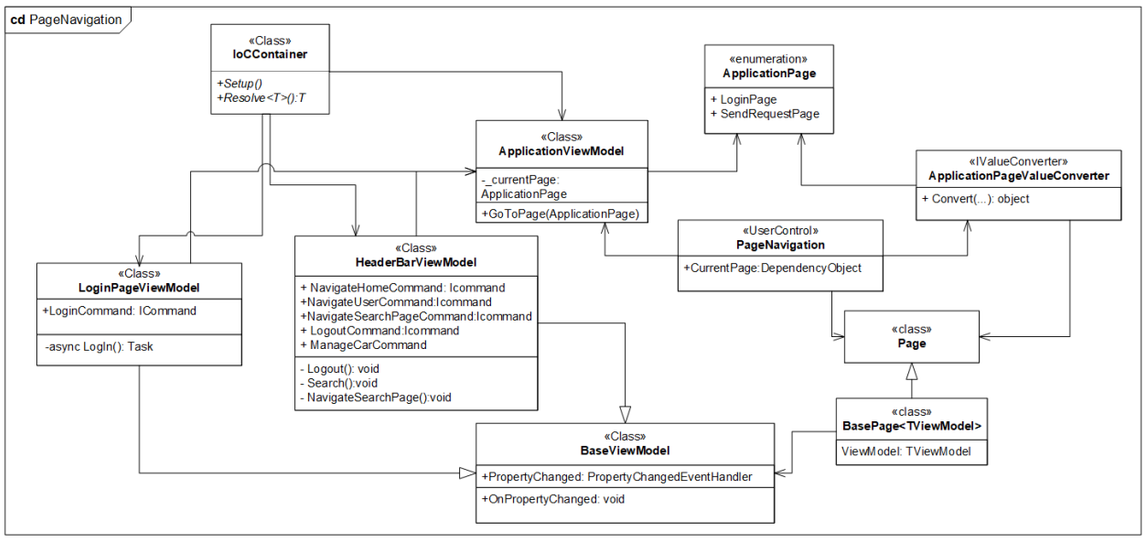
Før vi kommer mere ind på selve designet af de enkelte Views og Viewmodels, så er det en god ide og se på hvordan hele navigationssystemet er sat op. Det er nemlig vigtigt for CarnGo appen at der kan navigeres mellem de forskellige sider, når først brugeren er logget ind. Der vil efter login vises en startside, hvor brugeren herfra kan navigerer til forskellige sider ved hjælp af en headerbar, hvorefter det fra nogle af siderne også vil være muligt at navigere til andre sider igennem selve sidens design.
For at kunne implementere dette skulle en singleton bruges, der kunne tilgås fra alle siderne, så alle sider i teorien ville have mulighed for at skifte til de sider, den har brug for at skifte til. Det er essentielt, at det er samme instans de anvender. Hertil blev klassen ApplicationViewmodel lavet.

Det essentielle ved dette navigationssystem, er at opretholde lav kobling mellem ViewModllerne og stadigvæk opretholde MVVM-principperne i forhold ingen direkte relation mellem Views og ViewModels. Hvis hvert ViewModel selv havde til ansvar for at skifte mellem Views, skulle de havde direkte adgang til hvert ViewModel. Dette vil desuden også medfører, at ViewModellerne skulle kende til tilhørende View, da de selv skulle stå for at initierere det. Begge ansvar er udliciteret til ApplicationViewModel.

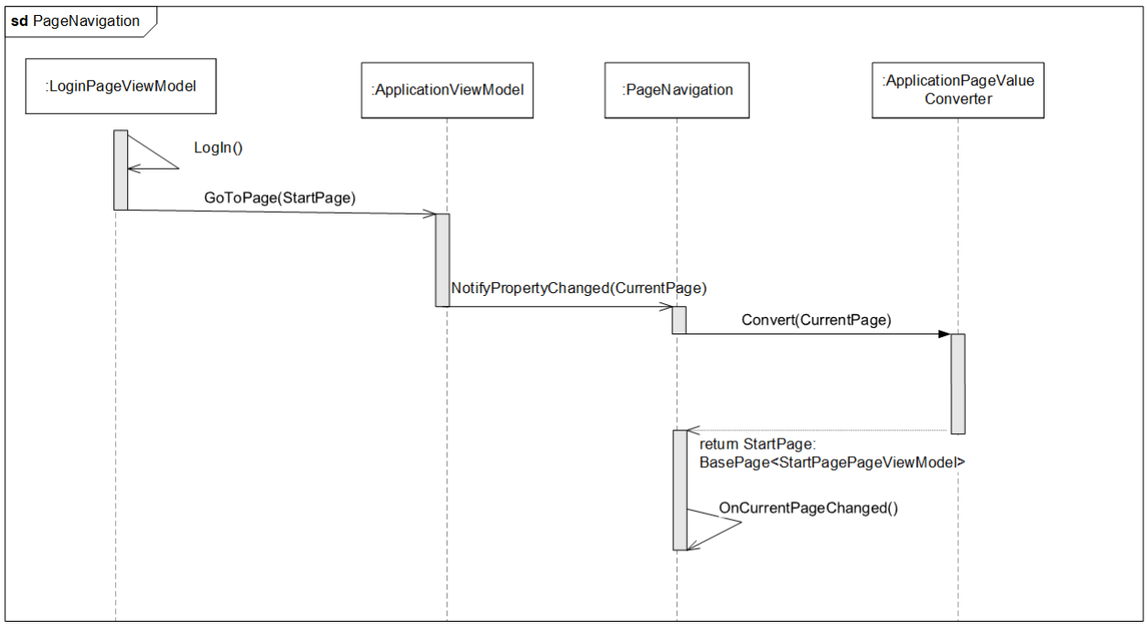
ApplicationViewmodel består af en usermodel(information om brugeren logget ind), funktioner til login af bruger, og en CurrentPage property, som holder en enum af typen ApplicationPage, der holder den nuværende side. Derudover er der en funktion(GoToPage) til at skifte CurrentPage Property.

ApplicationViewModel er registreret som en singleton i en IOC-container, der er brugt i applikationen, og igennem denne kan siderne enten bruge containerens resolve funktion til at få fat i applicationViewModel eller IOC-container kan injecte den ind i sidernes ViewModels.

Spørgsmålet er nu, hvordan der bliver skiftet imellem siderne. Det gøres ved at sætte CurrentPage Property i ApplicationViewModel til en af siderne specificeret i enumeration applicationpage. Dette er muligt, da der i MainWindowViewet er initialiseret en brugerdefineret usercontrol kaldet pagenavigation. Denne usercontrol har en dependencyproperty også kaldet CurrentPage af typen Page, som CurrentPage fra ApplicationViewModel databindes til. Da de to typer ikke er de samme, bruges en valueconverter til at konverterer fra typen ApplicationPage til Page. I det en ændring forekommer vil et event forekomme for CurrentPage dependencyproperty i PageNavigation og den nuværende frame(side) vil blive ændret. Nedenfor ses et klassediagram samt sekvensdiagram for dette. På klasse diagrammet i figur 3.1 ses alle klasserne involveret, Login skifter til startPage, hvilke er de samme klasser involveret, når alle sider skifter mellem hinanden(undtagen de to viewmodels loginpageViewmodel og HeaderbarViewModel). Her er kun 2 ViewModels vist, selvom alle viewmodels er med. Dette blev gjort for at holde diagrammet overskueligt. LoginViewModel er her vist som en af de viewmodels, hvor man gennem siden skifter til en specifik side. HeaderBarViewmodel er der hele tiden efter man er logget ind i applikationen og det er gennem den muligt at skifte mellem flere forskellige sider. På figur 3.2 ses et sekvensdiagram, som beskriver skiftet fra Login viewet til StartPage viewet. Dette gælder selvfølgelig også for de to viewmodel, der tilhører de to views.



Figur 3.1: Her ses klassediagram for PageNavigation, hvor man ser de forskellige klasser involveret, når en side skal skiftes ud med en anden.



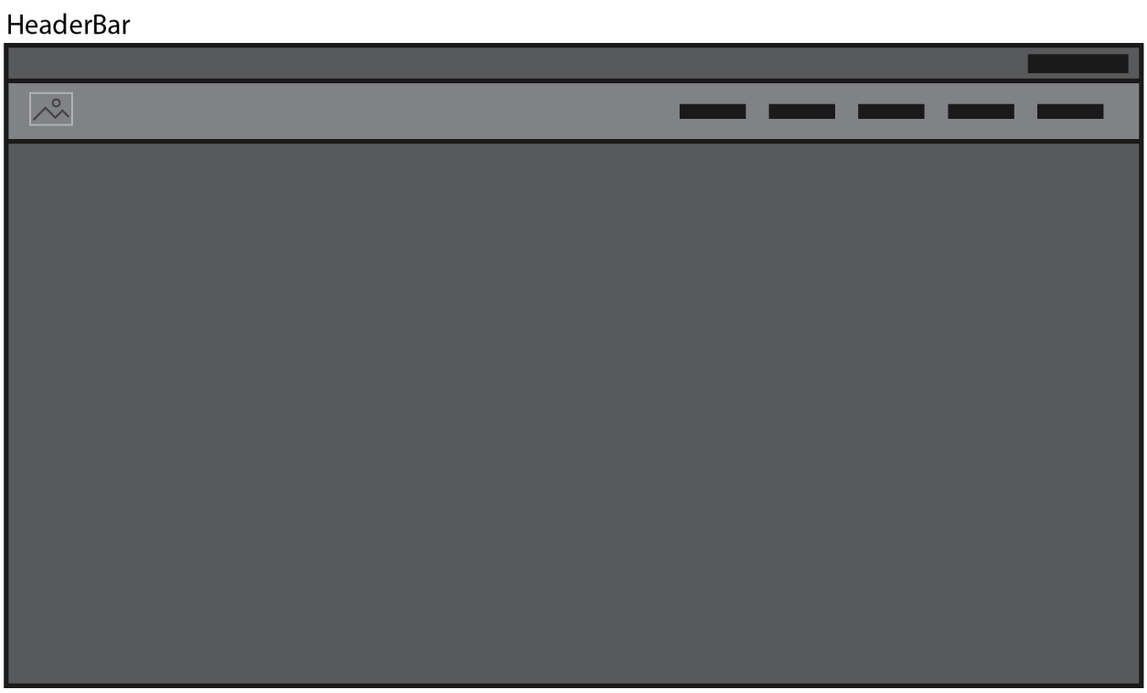
Figur 3.2: Her ses sekvensdiagrammet for PageNavigation, hvor Login skifter til StartPage Viewet

4 Design af Header Bar

Dette afsnit præsenterer designet af et View og en ViewModel til det, der er headerbaren. Der er valgt en tilgang til designet, der hedder ViewFirst, hvor Viewet laves først og derudfra laves funktionaliteten i ViewModel, så det stemmer overens med arkitekturen omkring HeaderBaren.

4.1 Design af View til Header baren

Til design af Headerbaren tages der udgangspunkt i de WireFrames lavet for applikationen, der ses i figur 4.1.



Figur 4.1: Header Bar WireFrame, der tages udgangspunkt i, i design af Viewet

Det der kendetegner HeaderBaren er, at det er en bar i toppen af Applikationen, der går på tværs af applikationen, og kan findes på samtlige sider. Det er også gennem denne, at man som en bruger navigerer gennem applikationen ved brug af navigationsredskaber (knapper, osv). Da man ikke skal være i stand til at navigere fra Login og Usersignup siden, inden man er godkendt som bruger, er dette dog den eneste undtagelse for, hvor headerbaren skal vises.

Nu hvor headerbarens funktionalitet er defineret meget overordnet, giver det mening at dykke lidt ned i de enkelte funktionaliteter. Tages der igen udgangspunkt i figur 4.1, så ses der i venstre side af baren et billede. Dette billede bør fungere, som navigation tilbage til startsiden, og samtidig vise logoet for applikationen. Det næste, der kan ses i headerbaren er en søgebar, der giver brugeren mulighed for lynhurtigt at søge efter en bil i deres nærområde. Efter søgebaren kommer flere navigationsknapper, der skal tage brugeren til henholdsvis Brugerprofil- og Bilsøgnings-siden. Så er der en notifikationsknap, der skal lave en dropdown-menu med alle notifikationer. Til sidst er der en knap, så brugeren kan logge ud.

4.2 Design af ViewModel til Header baren

Meget af funktionaliteten er givet ud fra Viewet og Arkitekturen, hvilket gør den nemt at designe. Der skal for alle knapperne, der kun har noget med navigation at gøre, være en kommando, der ved brug af ApplikationsViewModellen går til den rigtige side. Der er nogle

enkelte særtilfælde for navigationen, der er til Søgebaren og LogUd.

Startes der med Søgebaren, skal den indtastede streng sendes med ved navigationen. Dette kan umiddelbart gøres ved, at have en streng, man kan binde til i viewmodellen. Når så søgningen foretages (tryk på søg knappen eller keyboard-enter) så kan man gennem en EventAggregator smide et event til SearchViewModel om at der foretages en søgning på den ønskede streng.

Logud derimod skal først sørge for at fjerne brugeren fra applikationen og herefter navigere til LoginViewet.

Den næste funktionalitet, der skiller sig ud er notifikationsknappen. Selve det med at vise en Dropdown kan gøres rent i xaml. Da hver notifikationerne og hvert notifikationsitem har deres egen ViewModel, så er det ikke op til headerbaren at tage hånd om dette. HeaderbarViewModel skal dog abonnere på om, der er en ny ulæst notifikation, hvilket igen gøres igennem Event Aggregator.

5 Design af User Sign up

I dette afsnit præsenteres designet for den, hvor en bruger, der ønsker at registrere sig som bruger på applikationen, kan registrere sig. Dette afsnit beskriver både design af udseendet på siden, samt design af funktionaliteten af siden og databehandling af bruger inputs.

5.1 Design af View til user sign up

Som de andre views tager designet udgangspunkt i en udarbejdet WireFrame, der kan ses i figur 5.1.

Figur 5.1: WireFrame for Sign Up view

WireFramen i figur 5.1 beskriver det overordnede layout på siden, men også lidt funktionalitet i forhold navigation, hvilket dog kræver den fulde WireFrame. Dog kan det af denne ses, at der skal kunne navigeres mellem Login- og Signup-siderne. Da der på denne side kommer til at være meget bruger input, så bliver der også nødt til at være en form for indikation til brugeren, hvis de indtastede data ikke er valide. Dette kan dog umiddelbart løses ved at ændrer på stylingen af text-inputs til at vise dette på en pæn måde.

5.2 Design af ViewModel til user sign up

Denne ViewModel kommer i store træk til at bære præg af **Security View** fra arkitekturen, da der her håndteres brugerinformation og sensibel data, som brugerens koderord. Derfor bestræbes der efter ikke at holde nogle kodeord i clear-text i hukommelse (Variable eller lignende), men samtidig opnå den MVVM struktur, der ønskes i applikationen. Dette kræver dog et kompromis på sikkerhed, da der skal tilgås C# Passwordboxes via en monitor der hooker sig ind via en attached property. Dette er dog nødvendigt for at få fat i kodeordet i et SecureString format. Da der ikke kan sendes SecureString passwords til databasen, laves en SecureString hjælpe klasse, der kan konvertere SecureString til en string. Denne hjælpeklasse kan anvendes sammen med to PasswordBoxes for at lave en klassisk validering af brugerens kodeord er det brugeren egentlig ønsker. Til der udarbejdes et IValidator-interface, med en Validate funktion . Til interfacet tilføjes en PasswordValidator, der validerer om brugerens kodeord er mere end 6 karakterer langt og indeholder både tal og karakterer. Dette pådutter brugeren en form for sikkerhed omkring deres bruger, hvilket er i overensstemmelse med

Securit View. Desuden laves der også en validator, der validerer om den indtastede e-mail er en valid e-mail.

6 Design af Login

I dette afsnit præsenteres designet af Login, hvor en bruger, der ønsker at logge på sin konto på applikationen. Dette afsnit beskriver designet af vores View og dens udseende på siden, samt design af funktionaliteten.

6.1 Design af View til Login

Figur 6.1: Wireframe af login

Ud fra Wireframet i figur 6.1 kan der ses layoutet af Viewet. Det overordnede Wireframe viser funktionalitet angående navigation, mellem Views. LoginView skal kun navigere til startpage og til UsersignupView. View'et skal navigere videre til startpage, hvis brugeren logger ind med en oprettet konto. Hvis Brugeren ikke har en konto, trykker brugeren på knappen, som tillader brugeren at oprette en konto, og derved bliver navigeret til UserSignup page.

6.2 Design af Viewmodel til Login

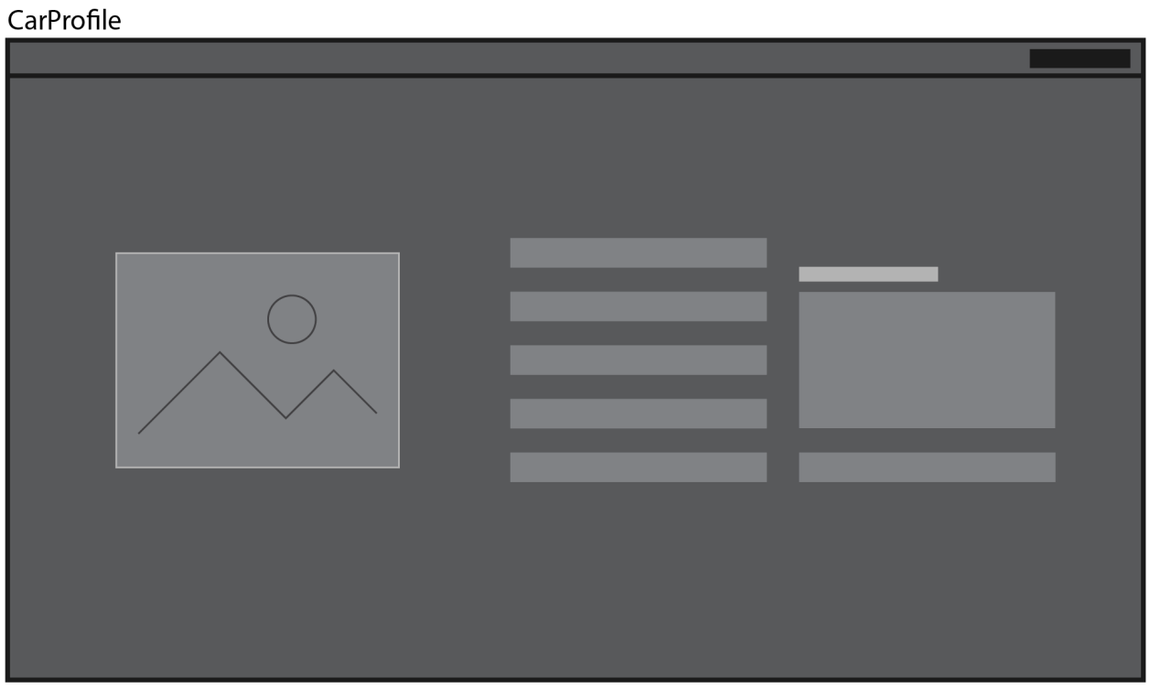
Når en bruger prøver at logge ind på sin konto, skal brugerens information overholde nogen regler, som f.eks at det skal være x antal langt kodeord, før det kan godkendes til at være et gyldigt kodeord. Der er et antal valideringer af brugerens kodeord og e-mail, hvor hvis de ikke bliver overholdt, bliver det udskrevet til brugeren hvilken af brugerens information ikke blev godekendt. Brugerens kodeord er fortrolig information, så det skal ikkee kunne ses i applikationen, så som løsning blev der benyttet securestring til koden, yderlig information af hvordan det fungerer kan ses på **UserSignUp**. Der bliver navigeret hen til Start page og Usersignup page via delegate commands.

7 Design av Car Profile Page

Dette avsnittet presenterer designet av Viewet og den tilhørende ViewModelen til CarProfilePage. Den er designet med ViewFirst prinsippet på samme måte som HeaderBar.

7.1 Design av View til Car Profile Page

Designet for CarProfilePage er basert på en WireFrame som ble laget for siden, denne kan sees på figur 7.1.



Figur 7.1: Car Profile WireFrame

CarProfile er siden man blir tatt til når man bestemmer seg for å leie ut bilen sin derfor skal den ha flere felter hvor brukeren kan inntaste informasjonen om bilen sin blant annet merke, modell og års nummer samt et sted hvor det kan lastes opp et bilde av bilen. Dette er også siden hvor brukeren kan bestemme hvilket tidsintervall bilen skal leies ut i. Denne siden blir også brukt som en side hvor utleieren kan se og redigere sin bil profil i ettertid.

7.2 Design av ViewModel til CarProfile

Funksjonaliteten til CarProfile vil variere etter om den skal brukes til å registrere en bil for første gang eller om utleieren skal redigere noe av informasjonen som er lagret om bilen.

CarProfilen blir lastet inn via ett event som kaldes når det navigeres til siden. Dette eventet søker etter hvilken bil som tilhører den innloggede brukeren i databasen. Denne dataen presenteres i en dropdown meny som lar brukeren velge hvilken bil som skal redigeres.

Hvis det er en ny bil som skal registreres vil siden starte i edit mode. Hvor det så er redigerbare felter, en knapp hvor brukeren kan laste opp et bilde og en save knapp. Hvis brukeren skal se/redigere en allerede eksisterende bil vil en read only versjon av siden åpnes hvor det viser en knapp hvor det står edit.

Edit knappen:
Edit knappen er den eneste kappen som blir presentert til brukeren da han åpner Car Profile

siden. Denne vil låse opp feltene og gjøre det mulig å redigere dem. I tilleg vil knappen endre seg til en save knapp.

Save knappen:
Save knappen sitt ansvar er å låse siden for videre redigering og samtidig oppdatere CarProfilen som ligger i databasen.

Upload photo:
Upload photo knappen åpner en ny dialog boks hvor man skal velge det nye bilde til bilen din. Dette bildet blir lagret som et BitmapImage i applikasonen og blir konvertert til en string når det overføres til databasen for lagring.

Delete Knappen:
Delete knappen sitt ansvar er å slette den valgte CarProfilen fra CarnGo applikationen og databasen og deretter navigere brukeren til HomePage hvis det var den siste bilen brukeren hadde registrert.

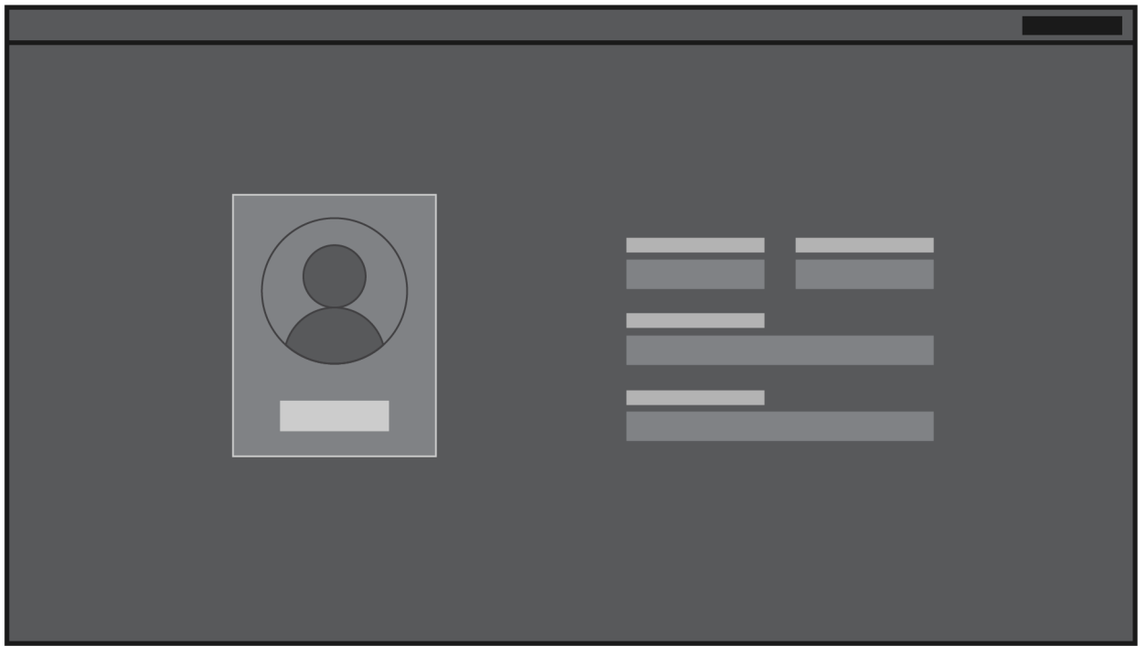
New Knappen:
New knappen brukes hvis brukeren vil registerer en ny bil i systemet selv om han allerede har en bruker profil lagret i systemet. Den vil da kalle en kommando som skaper en ny Carprofile som legges til brukerens liste av biler og lagrer den i databasen når det trykkes på save knappen.

8 Design av Edit User page

Dette avsnittet presenterer designet av Viewet og den tilhørende ViewModelen til EditUserPage. Den er designet med ViewFirst prinsippet på samme måte som HeaderBar.

8.1 Design av View til Edit User Page

Edit user page skal på mange måter fungere på samme måte som CarProfile. Den skal være siden som brukeren kan navigere til for å redigere bruker informasjonen som er lagret om dem i CarnGo.



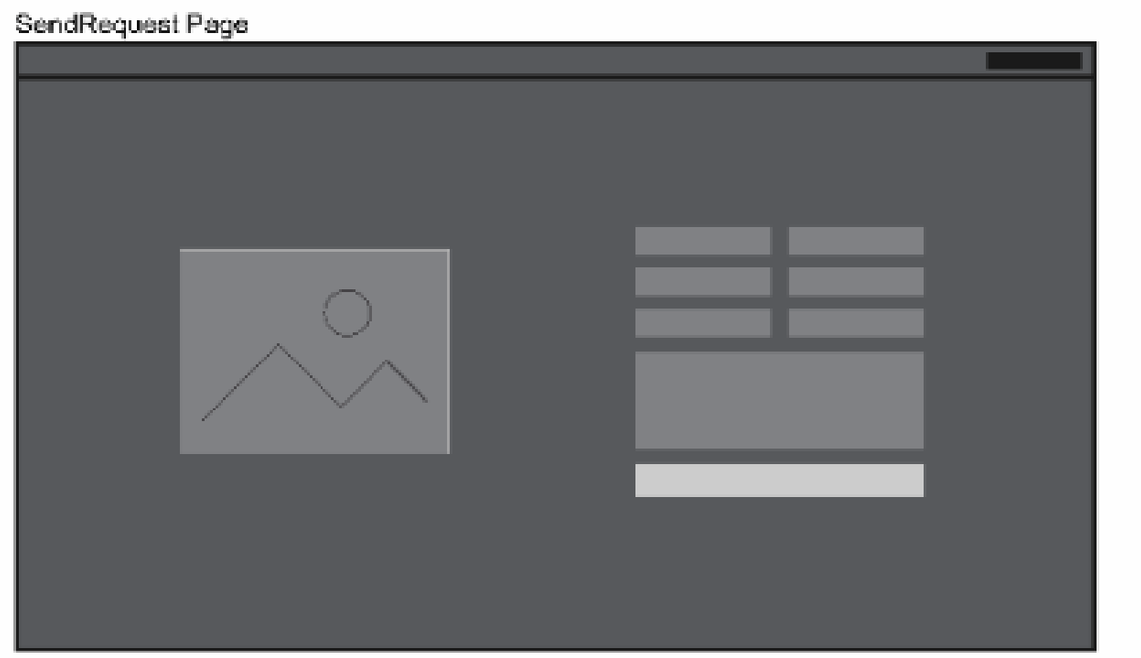
Figur 8.1: Edit User WireFrame

8.2 Design av ViewModel til EditUser

ViewModellen skal lagre en kopi av UserModelen som den skal utføre alle endringene på frem til brukeren trykke på save knappen. Når det trykkes på save skal den kopiere dataen over i brukerens UserModel og lagre endringene på databasen.

9 Design af Send Request Viewmodel

Når bruger/lejer har trykket på en bil inde fra SearchView bliver han dirigeret til SendRequest Viewet. Her har SendRequest Viewmodel til formål at hente information om bilen, der er blevet trykket på i SearchViewet. Registreringsnummer bliver sendt med fra SearchViewet, da SendRequest Viewmodel har abonneret på en Event Aggregator, som SearchViewmodel udsender information til, hvorfra information om bilen kan hentes i databasen. Informationen vil ved hjælp af databinding blive vist på venstre side af SendRequestViewet. På højre side vil information til udlejer indtastes og herefter sendes til udlejer, hvis informationen er indtastet korrekt. På figur 9.1 ses en wireframe, som designet er gået ud fra. Det endelige resultat varierer dog, da det på venstre side skulle vise lidt mere information end bare et billede.



Figur 9.1: SendRequest WireFrame, som designet tager udgangspunkt i.

9.1 UserControl med databinding

For at få udskrevet information og tilbehør til bilen er der blevet lavet en UserControl, der ved hjælp af en ContentControl i SendRequestViewet fået sin egen CarProfilModel, der er UserControllens Viewmodel. CarProfileModel er en property i SendRequestViewmodel, som bliver initialiseret med værdier, når bil information hentes fra databasen. For at vise noget informationen i CarProfileModel til brugeren gøres der brug af en valueconverter, for at vise en boolean værdi i CarProfileModel. Her er true et check ikon og false er et advarsel symbol.

9.2 Database Interaktion

Når brugeren navigerer til SendRequestviewet fra SearchViewet skal SendRequestViewmodel udtrække information fra databasen om bilen og udlejeren af bilen. Viewmodel bruger herefter denne information til at vise det på skærmen. Når lejeren herefter har indtastet information om ønsket udlejningsdato fra og til, og en besked, og trykket "Rent Car", så vil information gemmes i databasen, hvis der ikke er fejl i den indtastede information eller bilen allerede er udlejet i perioden.

9.3 Fejlhåndtering

En vigtig ting i applikationen er fejlhåndtering, så gruppen ikke gemmer forkert data i databasen eller information der ikke giver mening. Dette bliver håndteret ved at applikationens tekstbokse og DatePicker controllere bliver røde i kanten og fejlen nævnes i et tooltip. Det er stadig muligt at trykke på knappen "Rent Car". Her vil brugeren i stedet informeres med rød tekst under knappen, hvis information er forkert indtastet. Der bliver også undersøgt om bilen allerede er udlejet i perioden, hvis dette er tilfældet bliver brugeren også informeret med rød skrift.

9.4 RentCarFunction og commands

I SendRequestViewmodel bruges der en command, når knappen 'Rent Car' bliver trykket. Efter denne command(en prism delegatecommand) bliver triggered, så vil funktionen RentCarFunction kaldes. Denne funktions ansvar er for det første at tjekke for fejl i tekstbokse og datePickers, hvor funktionen her returnerer med en fejlmeddelelse. Efter dette gøres der brug af en hjælper funktion klasse, hvor funktionen confirmRentingDates bliver kaldt. Denne funktion tjekker på om bilen kan udlejes i den tidsperiode brugeren har indtastet i datepickers. Igen udskrives en fejlmeddelelse, hvis det ikke er muligt. Funktionen returnerer, hvis ConfirmRentingDates fejler. Hvis funktionen ikke har returneret på dette tidspunkt, så skal information lægges ind i databasen. Der bliver som det næste lavet en liste af de dage, hvor brugeren ønsker at udleje bilen. Herefter bliver informationen lagt ind i databasen for den valgte bil. Udlejer må herefter slette disse dage, hvis udlejeren ikke ønsker at udleje til brugeren(lejeren). Dette gøres for at man kan være sikker på at lejer tager stilling til om bilen skal udlejes i perioden. Til dette bliver en hjælper funktion i hjælperklassen igen brugt. Til sidst bliver beskeden lavet og sendt til udlejer(lagt ind i databasen).

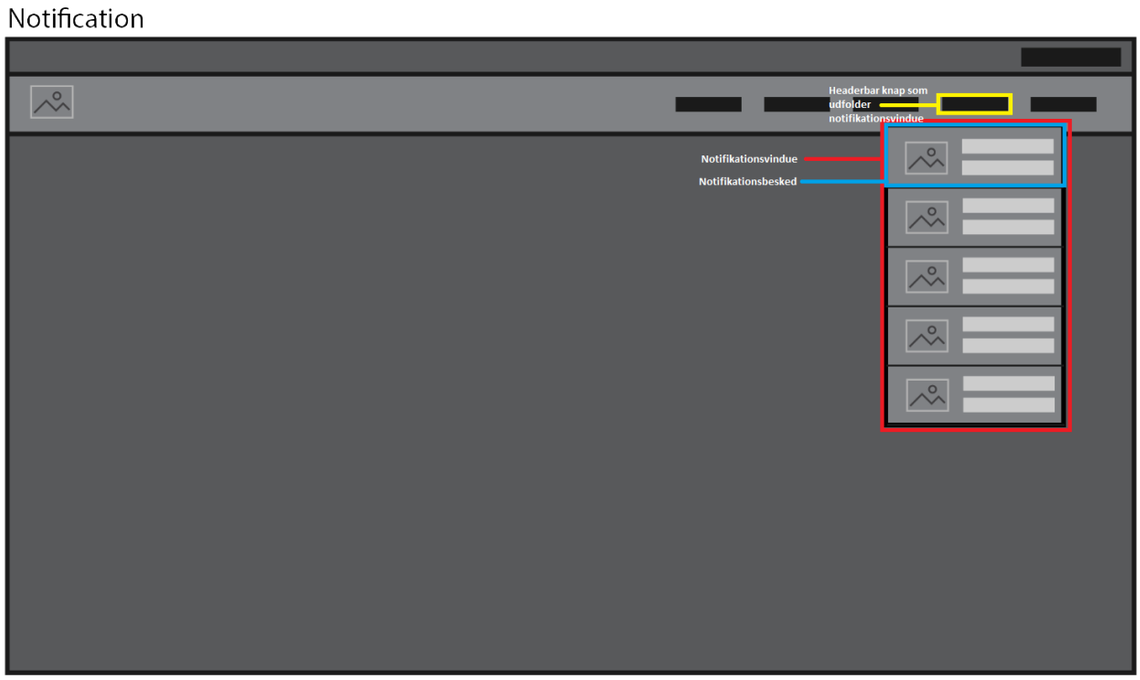
9.5 Ting der kunne gøres bedre

Lige nu bliver alle dage bilen kan udlejes vist med tekststrenge, da det er vigtigt at kunne se disse for brugeren. Dette kunne være vist med en til og fra dato i stedet eller på en lidt bedre måde end tekststrenge. En ide gruppen ikke nåede at implementere var at give en visuel repræsentation, noget lignende det man ser i en datepicker.
En anden ting er databasen, som laver flere queries, der er afhængige af hinanden, hvilket kunne være blevet gjort med transaktioner i stedet, hvilket ville hjælpe på data integritet.

10 Design af Notifikation vinduet

Inde i Headerbaren findes et ikon, som udfolder ved aktiveringen (Bruger interaktion). Her præsenteres brugeren for alle de notifikation (Beskeder), som er 'nye' for brugeren: beskeder som brugeren ikke tidligere har interageret med. Disse notifikation er 'nedkogte' beskeder, som kun indeholder de vigtigste informationer - det kunne fx være senderen og de første ord i den beskedstreng sendt. Den visuelle repræsentation af vinduet kan se i figur 10.1. Der kan være forskellige notifikationbeskeder, og det skal være let at tilføje nye beskedtyper. Der er to type beskeder defineret indtil videre:

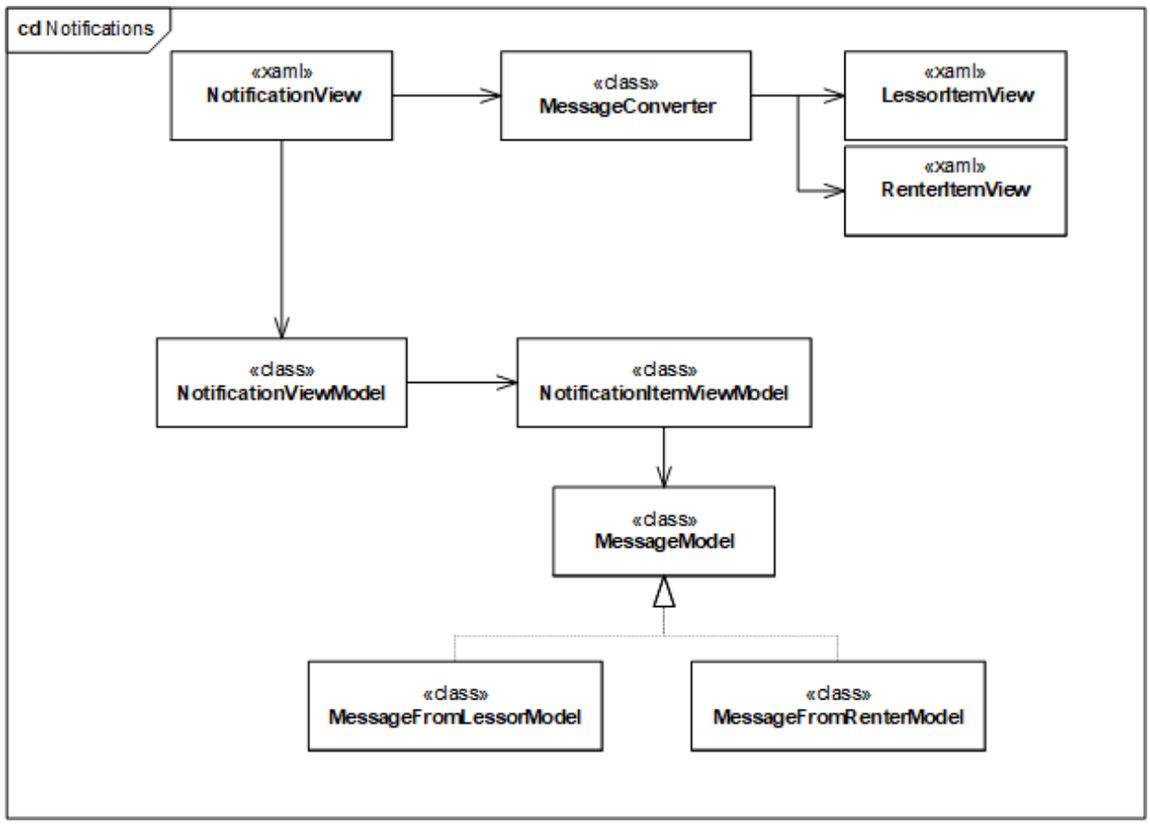
1. Udlejer respons til anmodning
2. Lejer anmodning om bil (Her kan udlejer godkende eller afslå anmodningen)



Figur 10.1: Frame for Notifikations vinduet

10.1 Besked opbygning

Notifikationsvinduet skal kunne indeholde forskellige typer beskeder, og det skal være let at tilføje nye beskeder. Hvis beskederne indeholder en besked på over 15 karakter, bliver de resterende karakter ikke vist i vinduet. Beskederne hentes fra databasen, hvor deres given typer definerer, hvordan de vises i notifikationsvinduet. I de følgende delafsnit forklares hvordan dette designes og figur 10.2 viser et klassediagram af operationen.



Figur 10.2: Klassediagram for notifikationer

10.2 MessageConverter

En bruger vil modtage notifikationer, når brugeren interagere med applikationen, fx når der enten lejes eller udlejes en bil. Disse notifikationer 'dropdown' og vises i headerbaren. Hver type besked vises forskelligt, der skal således runtime differenceres mellem besked typerne. Hertil bruges en 'valueconverter', som konverterer en besked til et 'wpf item' i notifikationsvinduet - en notifikationsbesked, repræsenteret som en blok inde i notifikationsvinduet (Se figur 10.1). Hvert besked har dets egen model, og for at illustrere informationer, skal der laves et tilhørende 'ItemView' for beskedmodellen - 'LessorItemView' er det visuelle vindue for informationen gemt i beskeden 'MessageFromLessorModel'. Hvis systemet udvides og der tilføjes flere typer beskeder, kan de let tilføjes ved at oprette et nyt ItemView, samt at beskederne nedarver fra MessageModel.

10.3 Interaktion med modeller og database

For hvert gang der registreres nye beskeder for den specifikke bruger fra databasen, opdateres notifikationsvinduet. Når notifikationsvinduet aktiveres, vil det altid vise alle ulæste beskeder - når brugeren trykker på en notifikationsbesked, navigeres brugeren til et andet vindue, som viser hele beskedens indhold, notifikationsbeskeden fjernes herefter fra notifikationsvinduet. Der loades de 10 nyeste beskeder fra databasen af gangen - brugeren har mulighed for at hente flere.

11 Design af MessageView

I forlængelse med Notifikationsvinduet, skal brugeren have mulighed for at navigere til hvert besked, således brugeren kan læse alt indholdet for beskeden. Når der bliver trykket på en notfikationsbesked, så navigerer systemet til MessageView'et. Vinduet skal bestå af to sektioner:

1. Den venstre side af vinduet skal vise alle brugerens beskeder
2. Den højre side skal vise indholdet af den besked, som brugeren har valgt.
    - Udlejer respons: Her vises beskeden fra udlejeren, samt om han har godkent bilanmodningen. Hvis dette er tilfældet, angives lejerens e-mail og et link til bilprofilen
    - Lejer anmodning: Her vises beskeden fra lejer - udlejeren kan godkende eller afvise anmodningen i notifkationsvinduet.

Nedenfor ses Wireframet for vinduet. Her ses hvordan de to sektioner er opdelt:

Figur 11.1: Frame for MessageView

11.1 Interaktion med modeller og database

Brugerens beskeder vil være et 1:1 forhold med notifikationsvinduet, her skal der vises de samme beskeder - beskederne er dog ikke statiske i den forstand, at de loades igen, når der skiftes mellem Notifikationsvinduet og Messagevinduet. Der vil være en søgebar til rådighed, hvor brugeren kan indtaste navnet på en bruger han/hun har kommunikeret med.

12 Design af Search

Dette afsnit beskriver designet for henholdsvis Search View og Search ViewModel. 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 søge funktionaliteten, og designet af denne tager udgangspunkt i en WireFrame, som kan ses i figur 12.1. 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 12.1, 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 12.1 er en række søgekriterier, som giver brugeren mulighed for at sortere i søgeresultaterne. Dermed behøver brugeren altså 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 af gangen, for så efterfølgende at filtrere dem, når der foretages en søgning. Selv om det betegnes 'Search' er det altså filtrering af søgeresultater, som er det essentielle, og dette er en del af præsentationslogikken. Med denne fremgangsmåde mindskes interaktionen med databasen, og derved er der mindre overhead og risiko for fejl.

Figur 12.1: Search Page WireFrame

12.1 Design af Search View

For at kunne vise bilerne på en hensigtsmæssig måde er der lavet en UserControl med et billede af bilen og informationer om bilmærke- og model, lokation, pris, antal sæder og udlejers navn. Disse er bundet til properties i den ViewModel, som er tilknyttet, og som repræsenterer det enkelte søgeresultat (SearchResultItem ViewModel). Informationerne trækkes ud af databasen og søgeresultaterne er initialiseret, når der navigeres til Search View. Dette indeholder desuden søgefelter for henholdsvis lokation, bilmærke, antal sæder, samt datoer for afhentning og tilbagelevering. Samtlige af disse søgekriterier er bundet til properties i den tilknyttede SearchViewModel.

12.2 Design af Search ViewModel

Der kan navigeres til søgevinduet ved at trykke på 'Find Car' knappen som befinder sig i Header Bar. Search ViewModel har abonneret på en EventAggregator, og derfor kan der navigeres fra Header Bar til Search uden at de har direkte kendskab til hinanden. Alternativt kan der foretages en 'hurtig-søgning' i søgefeltet i Header Bar. Dette medfører at der navigeres til Search og foretages en filtrering efter lokation på baggrund af den indtastede string. Når der trykkes på 'Rent'-knappen for en bil gøres der igen brug af Event Aggregator, og der navigeres til SendRequest View, mens bilens registreringsnummer sendes med.

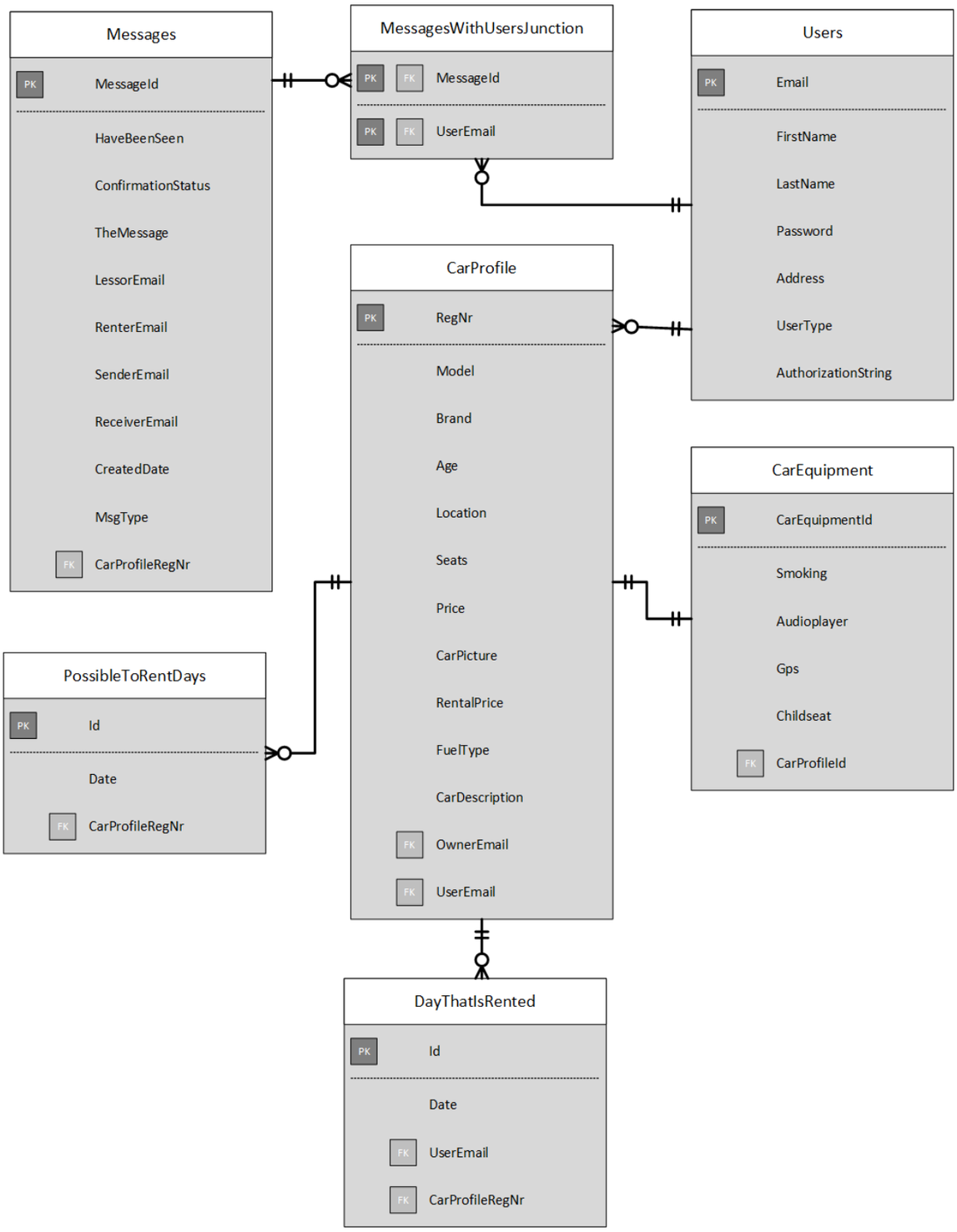
SearchViewModel har en ObservableCollection bestående af SearchResultItemViewModels. Det er denne, som udgør søgeresultatet, og som der bindes til i Viewet. Til selve filtreringen anvendes et CollectionView. Det giver mulighed for at manipulere den Collection, som Viewet er bundet til, mens abstraktionen mellem View og ViewModel bevares.[1] På denne vis anvendes ObservableCollection som en form for repository, og brugeren kan frit vælge at kombinere en eller flere af søgekriterierne, som der så bliver filtreret efter ved tryk på en søge-knap.

Når brugeren har indtastet en oplysning i venstre side bliver kriteriet tilføjet en liste af Predicates med den værdi, som der skal sammenlignes med. Hvis der ikke er udfyldt nogen kriterier vises alle bilerne, og ellers skal alle Predicates tjekkes og være opfyldt for en given bil før den inkluderes i Viewet. Denne fremgangsmåde har fordelen, at hvert enkelt søgekriterie kan holdes simpelt og testes separat.

SearchViewModel realiserer interfacet IDataErrorInfo med det formål at håndtere fejl. Data valideres bl.a. når der indtastes antal sæder eller vælges datoer i venstre side af Search View. Hvis ikke der er valgt en gyldig værdi, får tekstboksen en rød kant, og der bliver udstedt en passende fejlmeddelelse i tooltip. Det vil desuden heller ikke være muligt at foretage en søgning, hvis der er kriterier med ugyldig data.

13 Design af CarnGo Database

I dette afsnit beskrives designet af databasen. Databasens operationer og funktionalitet tager udgangspunkt i beskrivelsen i afsnit 3 i softwarearkitekturen. Databasen er designet ud som relationel model, hvor relationer kan sammenlignes med hinanden. Figur 13.1 er et E/R diagram, som er til værktøj til at modellere databasen - det viser alle entiteter og deres relationerne. E/R diagrammet repræsenteres ved Crows foot notation, da det fylder lidt mindre, selvom det ikke giver den bedste beskrivelse af relationerne mellem entities. Relationerne er dog rimelig overskuelige.



Figur 13.1: Entity–relationship diagram med Crows foot notation for CarnGo databasen

Der er i midten af diagrammet vist CarProfile, der er bilprofilen med alt information om den, som brugeren vælger at vise. Der er en optional en-til-mange relation mellem CarProfile og PossibleToRentDays samt CarProfile og DayThatIsRented. PossibleToRentDays er her de

dage, hvor udlejer ønsker at udleje sin bil og DayThatIsRented er de dage hvor bilen allerede er udlejet. Det blev lavet på denne måde, da det viste sig at være den bedste måde gruppen kunne komme på, da det at have en entitet, der holdte til og fra dage ville gøre det mere kompliceret og finde ud af hvilke dage en bil kunne lejes. Det fylder dog mere i databasen, hvilket er det negative i at gøre det på denne måde. CarProfile har også en en-til-en relation til CarEquipment, der bare er flere detaljer om bilen. Hver Bruger kan have en valgfri mængde biler brugeren ønsker at sætte til leje, hvilket beskriver relationen mellem CarProfile og Users. Hver Bruger har en mange-til-mange relation til Beskederne (Message entitetten), hvilket forklare junktiontabellen MessageWithUsersJunction mellem Users og Messages. Mere om denne relation blive forklaret i næste afsnit, hvor kommunikations system bliver forklaret.

Databasen ligger på en ekstern server og tilgås via SQL server query's, automatisk genereret af 'entity framework' ud fra funktioner i en klasse nedarvning fra typen DbContext. Disse funktioner kaldes via funktioner i model laget. Entity Framework core gør det at lave en relationel database meget lettere, når applikationen skrives i C#. Man kan her arbejde direkte på model objekter, der laves på helt samme måde som andre objekter i C# og entity framework kan ud fra disse objekter lave de entities sql databasen består af. Det gav dog nogle gange nogle udfordringer, at bruge ef core, da man ikke altid er klar over, hvordan frameworket fungerer og de fejl man får kan nogle gange svære at fikse.
En vigtig overvejelse er hvordan man kan tage mindst muligt af databasens kapacitet. Siden det tager mere tid at finde mange små elementer og hente dem ned, end at tage et større element, er det ønskeligt at lave større query's, frem for mange query's.

13.1 Overvejelser og hvad der kan gøres bedre

I CarnGo applikationen bliver databasen tilgået direkte, hvilket der senere hen i semestret viste sig ikke at være den bedste løsning af den grund, at gruppen på denne måde blev tvunget til at bygge sikkerheden omkring passwords op fra bunden. Dette undlod gruppen dog at gøre, da der ikke var tid til dette, og af denne grund bliver passwords ikke lagret sikkert i systemet. Dette kunne være undgået, hvis gruppen brugte et Web API, som der ikke var kendskab til i starten af semestret og dermed i planlægningen af, hvordan databasen skulle laves. På denne måde kunne gruppen have fået hjælp af fx Microsoft Identity platformen til at gemme passwords sikkert.

13.2 Kommunikation system

En vigtig ting i CarnGo applikationen er, at der kan kommunikeres mellem en lejer og udlejer. Til at få dette til at lykkedes skulle der laves et kommunikations system. Ideen var at kommunikation skulle foregå igennem en database, hvor en lejer kunne lægge en besked op i databasen til udlejer om leje af en af udlejers biler(tidsinterval samt tekstbesked). Udlejer ville igennem databasen notificeres om at en besked er modtaget, hvorved han læser beskeden og kan accepterer eller afvise. Når beskeden er accepteret eller afvist vil lejer få besked ved at beskeden i databasen er ændret, og lejer bliver notificeret om dette.

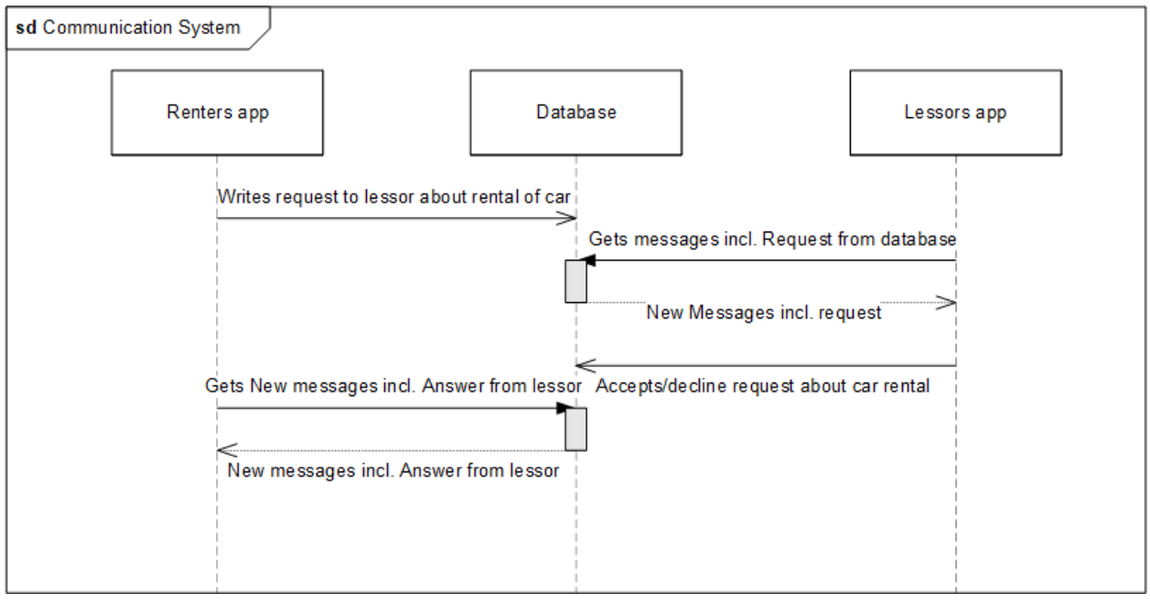
13.2.1 Entitetter i databasen

For selve kommunikationssystem er der nogle entities i databasen der er vigtige for at få det hele til at virke. Disse entities er Messages, som er selve beskeden i databasen med tekstbeskeden, samt email (primary key for user) for både lejer og udlejer samt sender og modtager. Der er en Confirmationstatus variabel for at lejer kan se om leje af bil er godkendt, afvist eller venter på bekræftelse. Der er også en besked type variabel for at vise om beskeden er fra lejer til udlejer eller omvendt. Der er også tilføjet en boolean for om en besked er set, som ville blive vigtig for at få notifikationer til at virke. Det er også vigtigt med en reference til bil entitetten, da både lejer og udlejer skal vide, hvilken bil korrespondancen omhandler. En besked har en reference til en junction tabel entitet mellem en bruger og en besked. Der kan nemlig være flere beskeder tilknyttet samme bruger(enten lejer eller udlejer) og for en besked kan der være flere brugere(en udlejer og en lejer. Det er vigtigt for en bruger at have en reference til sendte men også modtagede beskeder.
PossibleToRentDays og DaysThatIsRented er begge entitetter, der ikke er en del af selve kommunikationssystemet, men bliver påvirket af handlinger foretaget under korrespondancen. Hver bil i databasen har reference til en PossibleToRentDays og DaysThatIsRented, hvor PossibleToRentDays er de mulige dage bilen kan lejes og DaysThatIsRented er hvilke af disse dage bilen er udlejet. Når lejer sender besked til udlejer, så vil den valgte tidsperiode for leje af bil lægges ind i databasen(som DaysThatIsRented). Dette vil sige at andre lejere ikke kan leje bilen i denne tidsperiode før udlejer har afvist lejeren, hvorved DaysThatIsRented vil slettes fra databasen.

13.2.2 Håndtering i applikation

Hele kommunikationen starter altid hos lejeren, der starter korrespondancen i applikationens SendRequestView ved at sende en anmodning om leje af bil til udlejer. Her vil DaysThatIsRented blive lagt ind i databasen for bilen med tidsperiode for udlejning og en besked med junktion tabeller til brugere vil blive lagt ind i databasen.
I udlejers applikation var det meningen at udlejeren skulle notificeres om en ændring i applikationens Header bar. Dette blev dog ikke lavet, og vil forklares i afsnittet "Problemer med Notifikationer". I stedet vil notificeringen først dukke om i en pop op menu, når notifikations ikonet i header bar trykkes. I denne notifikation vil, der på udlejers side vises en knap til at accepterer lejers anmodning og ligeledes en knap til at afvise den. Notifikationen kan også åbnes ved at trykke på den(andre steder end de to knapper), hvorved hele tekstbeskeden vil ses. I tilfældet hvor udlejer afviser en anmodning vil DaysThatIsRented dage i frigives(slettes)på bilen i databasen og andre kan nu leje bilen i denne tidsperiode. I tilfældet hvor udlejer derimod bliver godkendt bliver der ikke ændret noget i databasen udover ConfirmationStatus, som ændres lige meget hvilket valg udlejer foretager.
I lejers applikation vil lejer nu kunne se om anmodningen er accepteret eller afvist ved at trykke på notifikationes ikonet i header baren. Igen vil han pga. problemer med notifikationer ikke kunne notificeres om ændringen automatisk uden at trykke på notifikations ikonet. Et

lille sekvensdiagram, som beskriver kommunikation mellem lejer og udlejer gennem databasen kan ses i figur 13.2.



Figur 13.2: Her ses kommunikation systemet, hvor kommunikation mellem lejer og udlejer gennem databasen vises.

13.2.3 Problemer med notifikationer

Det var meningen, at både lejer og udlejer ville blive notificeret om at en besked var sendt til dem uden at en aktiv handling(tryk på notifikations ikon i header bar) fra brugeren skulle opdaterer headerbar med nye beskeder. Det viste sig dog ikke at være muligt. Nedenfor ses de ideer, der var og hvorfor de viste sig ikke at løse vores problem.
Den første ide var at bruge Query notifikations, som var blevet nævnt under database undervisningen, hvor sql serveren vil være i stand til at udløse en trigger, når databasen bliver opdateret, hvorved applikationen vil notificeres om ændringer og eventuelt kan læse fra databasen og hermed UI. Det blev dog hurtigt klart at dette ikke var en mulighed, da den azure databases, som er den cloud database leverandør, der bruges i projektet ikke understøtter query notifications.
Den anden ide var at applikationen skulle polle databasen for ændringer i en funktion, der bliver udløst af en timer med et specifikt tidsinterval. Dette skulle foregå i en anden tråd, så UI tråden ikke blev påvirket. Det viste sig dog, at programmet crasher, når databasen bliver tilgået på flere tråde, hvor tråden notifikations polle tråden crasher og dermed programmet. Det var utroligt svært at debugge problemet og vi måtte til sidst give op på det. En mulig løsning på kunne være at bruge en unit-of-work design pattern. Der var dog ikke tid til at implementere denne design pattern.

Referencer

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