Aarhus Universitet
Semesterprojekt 4
Gruppe 5

Arkitektur

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	07/04/2019	MG	Tilføjet Security View
1.1	08/04/2019	TM	Tilføjet Process View
1.2	23/04/2019	TM	Gennemgang af dokument og rettelser
1.3	11/05/2019	TM	Tilføjelse af domænemodel og rettelser
1.4	24/05/2019	MG	Tilføjet/Fikset Development og Physical view
1.5	26/05/2019	MG	Redigeret i Scenarier og Logical View

Indhold

2 Systemarkitektur

I dette afsnit beskrives system- og softwarearkitekturen for applikationen CarnGo. Der vises, som det første, et User Story distribution diagram, der beskriver hvordan de User Stories beskrevet i kravspecifikation kommer til udtryk i system- og softwarearkitekturen. Her samles en eller flere User Stories til en distribution. Der findes et system sekvensdiagram til hvert distribution, samt en applikationsmodel i software arkitekturen. For at beskrive arkitekturen af det softwareintensive system bruges 4+1 modellen. Modellen bruges til at beskrive systemet ud fra forskellige perspektiver:

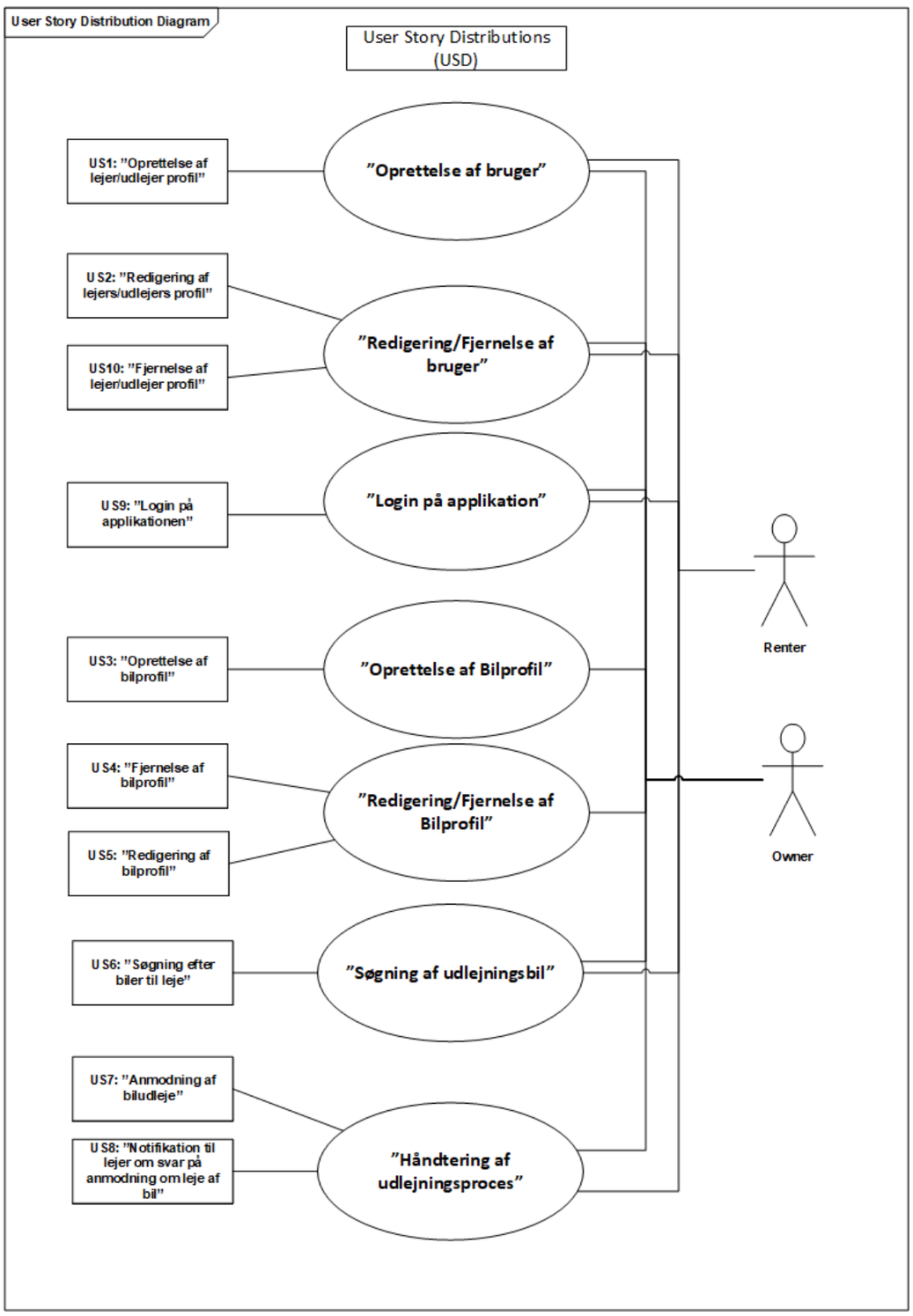
- **Logical View:** Beskriver systemets funktionalitet, som leveret til slutbrugerene. Dette illustreres vha. applikationsmodeller. Da der laves en WPF-applikation er Model-view-viewmodel(MVVM) brugt til at beskrive software arkitekturen.
- **Process View:** Beskriver systemetprocesserne og hvordan de kommunikerer, samt systemets adfærd.
- **Deployment 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 2.1)

  User Story Distribution Diagram: 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 har samme aspekter som et User Story Map, hvor User Stories bliver kategoriseret under et fælles 'Epic' (overordnet tema).
- **Security View:** Der tilføjes også et sikkerheds view, som fokuserer på, hvordan systemet implementeres ud fra sikkerhedsmæssige elementer, og hvordan sikkerhed påvirker systemegenskaberne.

2.1 User Story Distribution Diagram

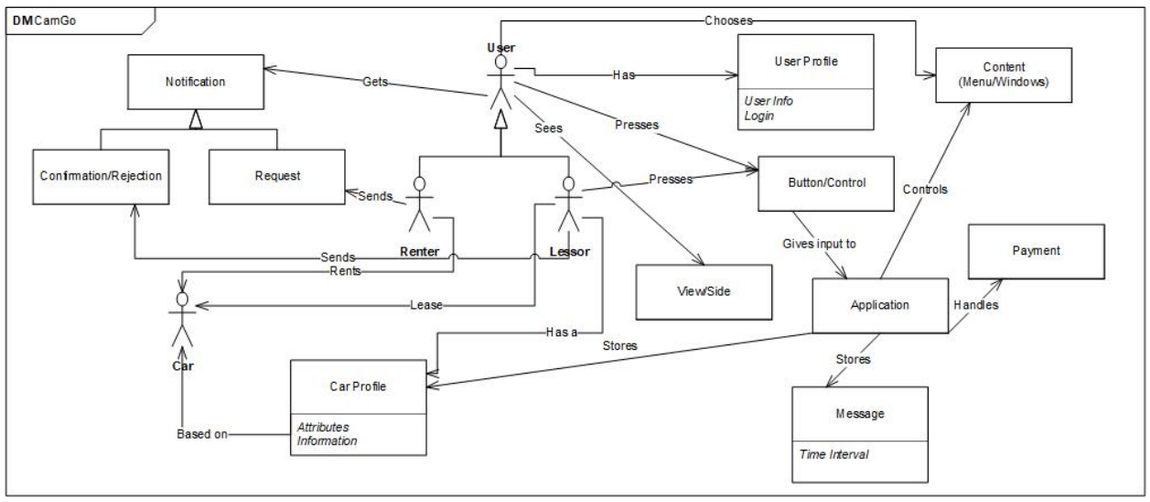
2.1.1 Fra User stories til arkitektur

User Story Distribution Diagrammet viser User Stories sammenhæng med arkitekturen. De andre views tager udgangspunkt i distributionerne.



Figur 2.1: User Story Distribution Diagram (USDD)

Disse User Stories er designet ud fra domænet, som implicit blev introduceret i Problemformulering og Kravspecifikationen. For at konceptualiserer domænet og beskrive dets adfærd og data, er der lavet en domænemodel. Domænemodellem angiver systemets eksterne og interne aktører/roller, som interagerer med softwarekomponenter og data. Domænemodellen ses nedenfor.



Figur 2.2: Domænemodel af CarnGo-systemet.

I domænemodellen introduceres alle 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 bruger, indtil de registreres i systemet; enten som lejer eller udlejer.

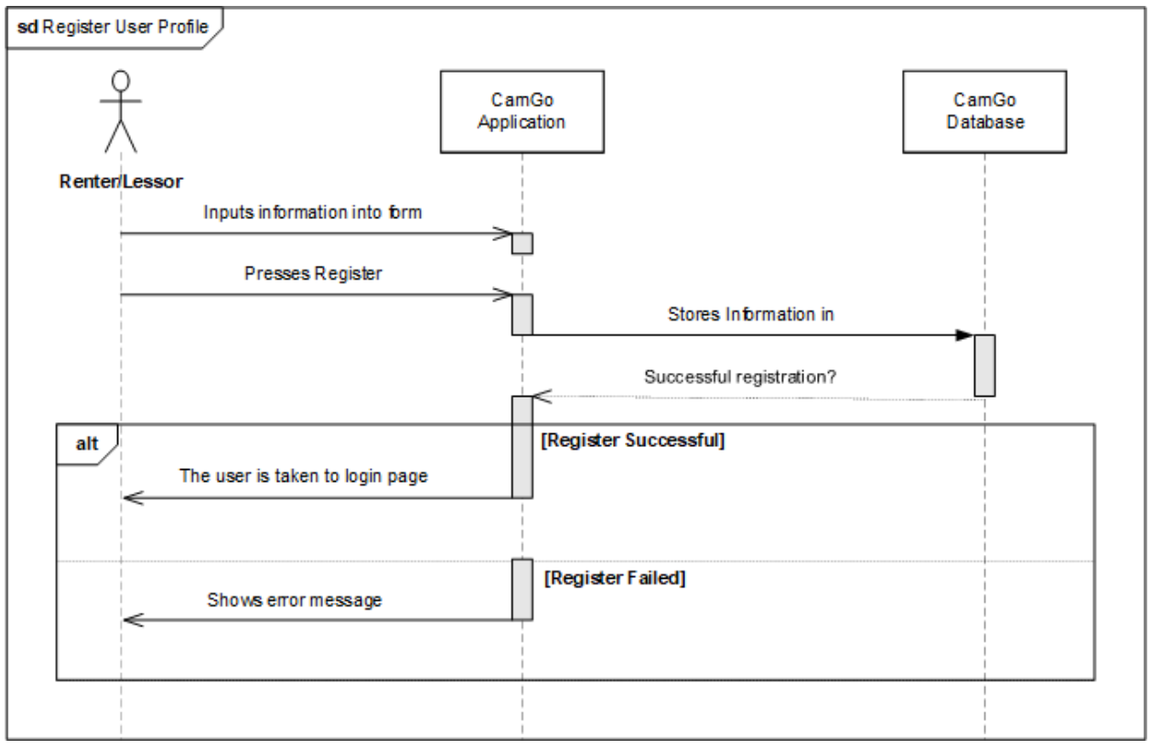
Nu hvor systemets koncepter er illustreret, er det muligt at beskrive dets arkitektur. De næste afsnit vil gennemgå systemets arkitektur, grænseflade og adfærd.

2.2 Scenarier

I dette afsnit beskrives systemets funktionalitet og adfærd, som tages udgangspunkt i distributionerne angivet tidligere. For at illustrere systemets adfærd bruges sekvensdiagrammer, som beskriver sammenspillet mellem aktører, applikation og database.

2.2.1 Oprettelse af bruger

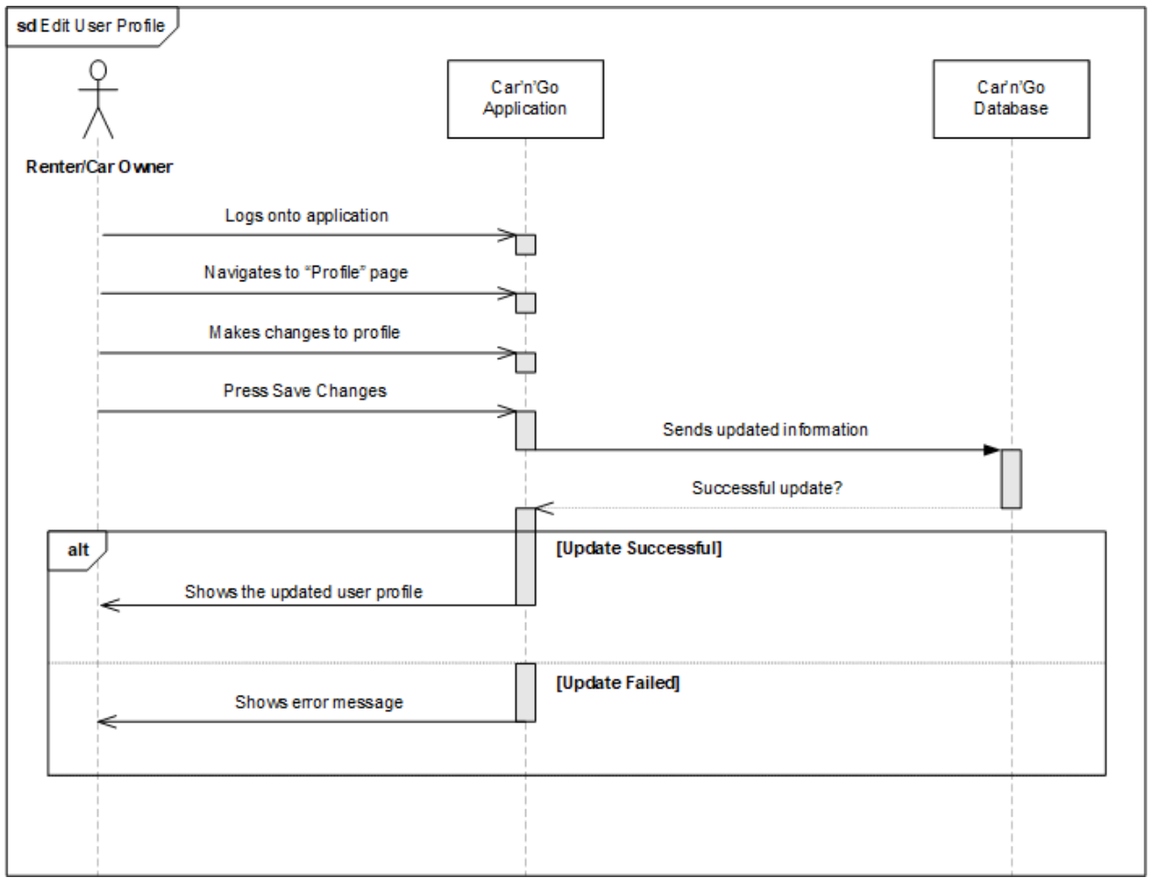
For at kunne anvende CarnGo applikation må brugeren oprette en profil. Brugerens information gemmes i databasen ved oprettelse.



Figur 2.3: Her ses et skevens diagram for samspillet mellem lejer/udlejer, CarnGo applikationen samt databasen.

2.2.2 Redigering/Fjernelse af bruger

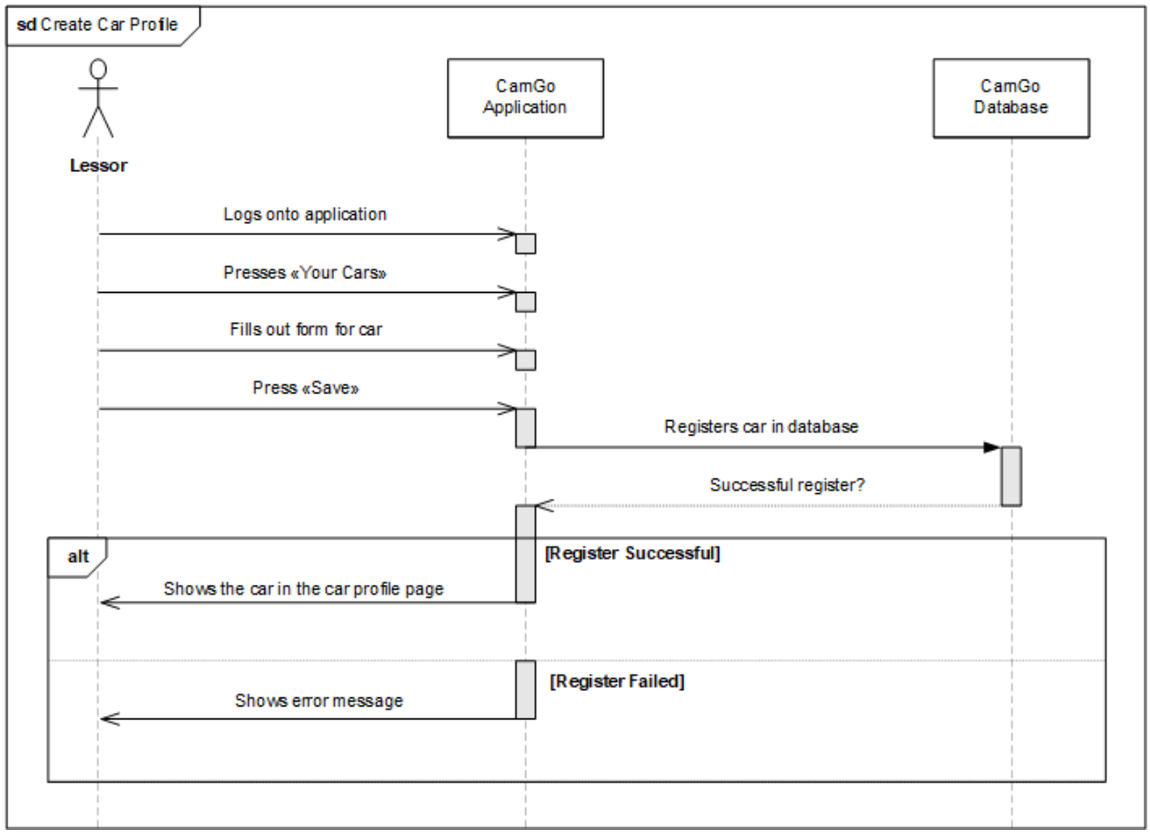
Alle registrerede brugere i systemet har tilknyttet visse informationer, som identificerer dem som person. Det skal være muligt at ændre brugerinformationer, samt slette brugeren, hvis den ikke ønskes længere - brugeren skal være logget ind i applikation for at kunne lave ændringer.



Figur 2.4: Sekvensdiagram for redigering og fjernelse af en brugerprofil.

2.2.3 Oprettelse af Bilprofil

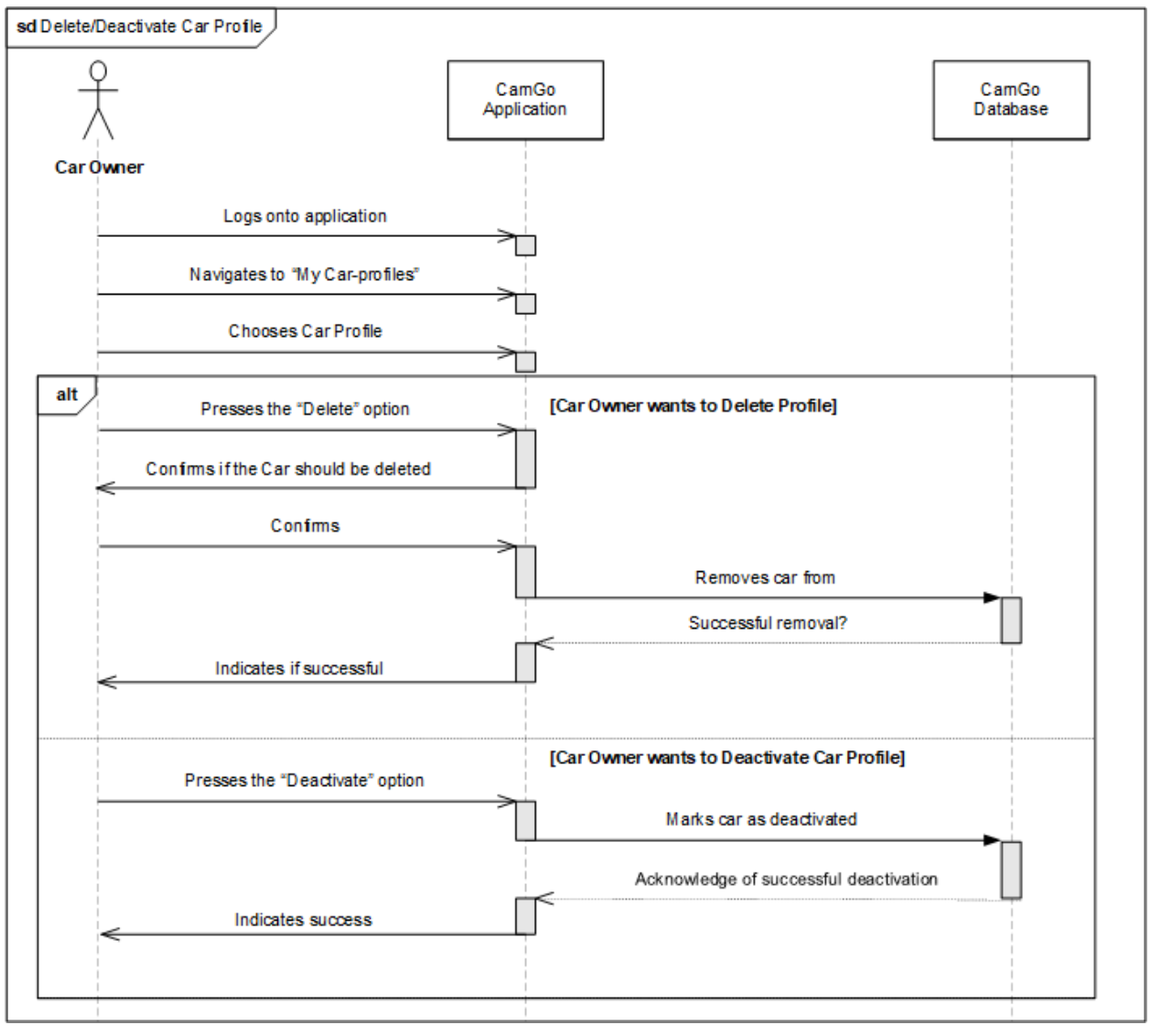
En bilprofil repræsentere alt information omkring den bil som udlejes. En bruger med udlejerrettigheder skal have mulighed for at oprette en bilprofil. En bruger med lejerrettigheder har ikke mulighed for at oprette bilprofiler.



Figur 2.5: Her ses et sekvensdiagram for samspillet mellem udlejer, applikation og database.

2.2.4 Redigering/Fjernelse af Bilprofil

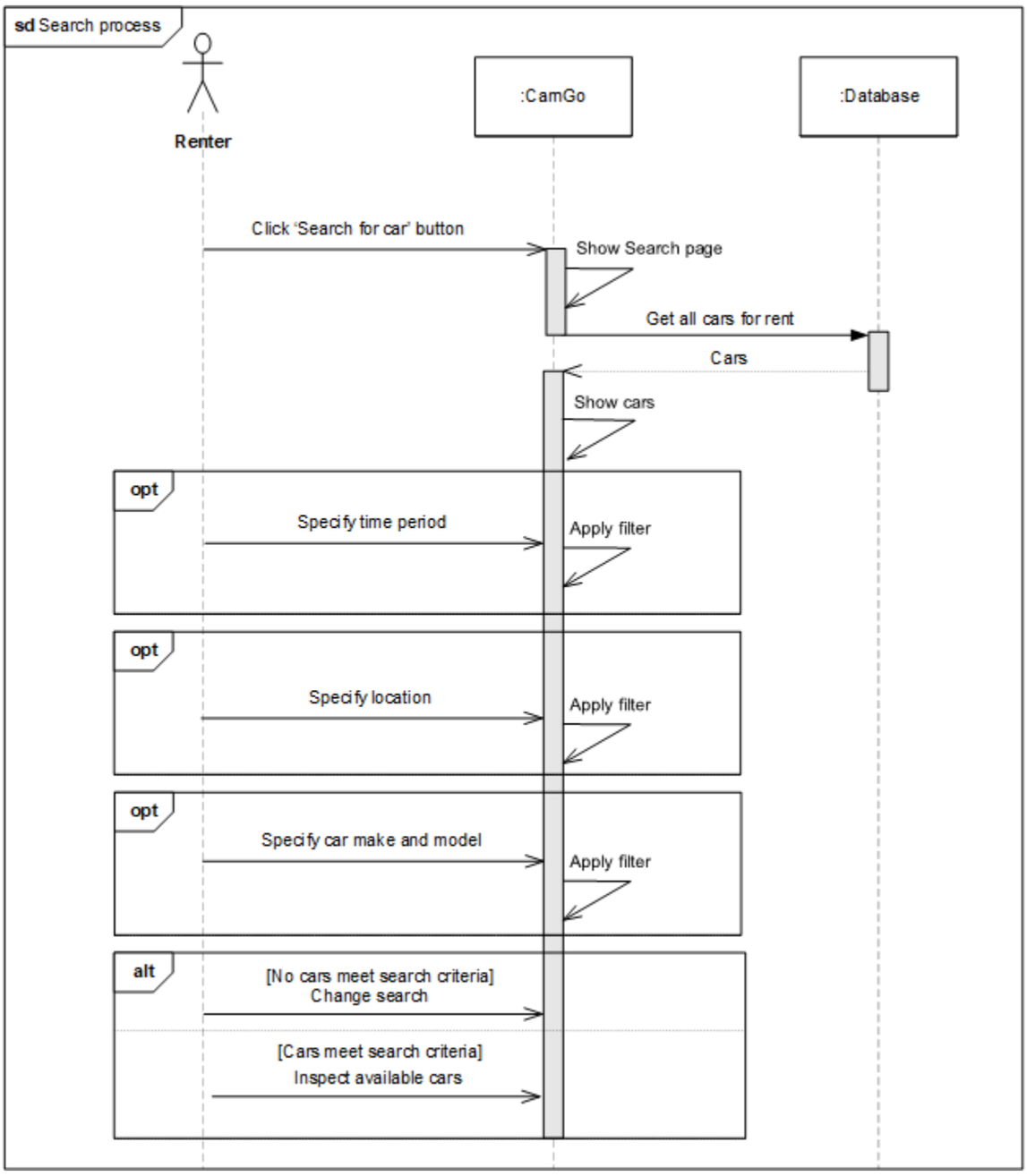
Det skal være muligt at ændre eller fjerne en allerede oprettet bilprofil - brugeren med udlejerrettigheder skal være logget ind i applikation for at kunne lave ændringer.



Figur 2.6: Her ses sekvensdiagrammet for fjernelsen af en bilprofil, som viser samspillet mellem udlejer, applikation og database.

2.2.5 Søgning af udlejningsbil

Når en lejer ønsker at leje en bil, så skal han søge efter en i applikationens katalog, hvor udlejere har sat dem til leje.



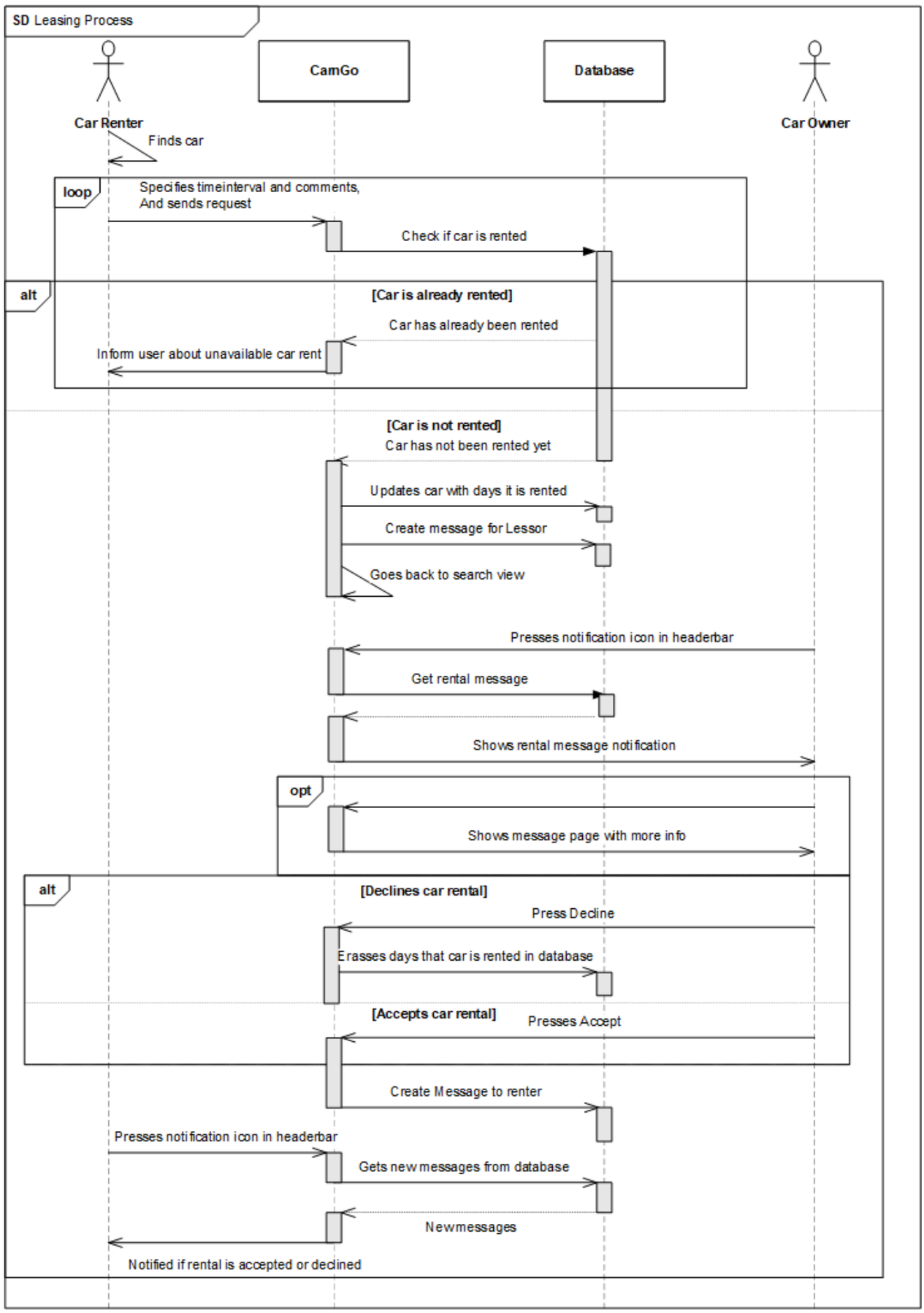
Figur 2.7: Sekvensdiagram for søgning efter biler til leje.

2.2.6 Håndtering af udlejningsproces

Efter at lejer har søgt i applikationens katalog, og har fundet en bil, som lejer ønsker at leje, så starter udlejningsprocessen. Dette realiseres med 3 trin:

1. Lejer anmoder udlejer om billeje
2. Udlejer godkender/forkaster anmodning om billeje

Valgfri Lejer tager kontakt til udlejer i forhold til udveksling af bil.



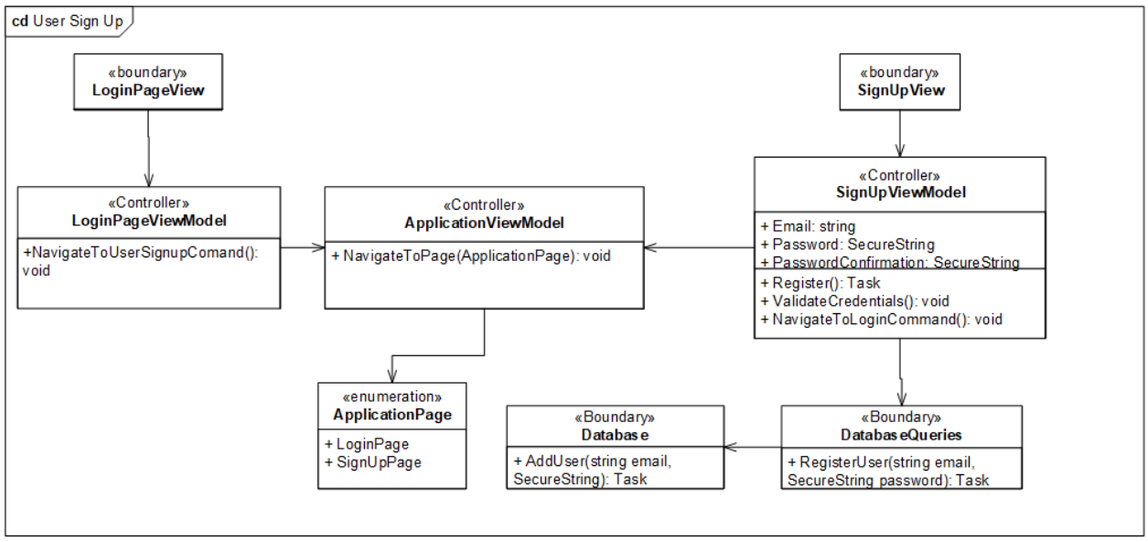
Figur 2.8: Sekvensdiagram for håndtering af udlejningsprocessen.

2.3 Logical View

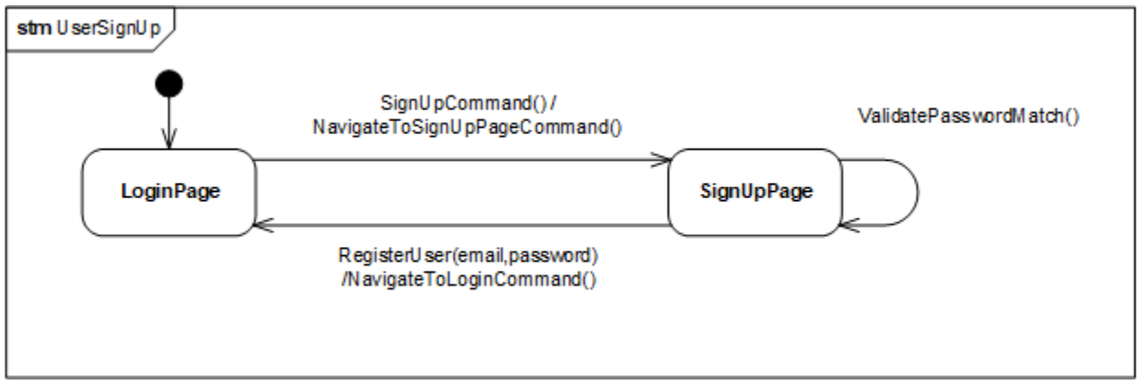
I dette afsnit præsenteres det logiske view i 4 + 1 modellen. Dette view omhandler funktionaliteten af systemet i forhold til brugeren og hvordan den er struktureret. Funktionaliteten beskrives i afsnittet ved brug af klassediagrammer og state-machines fra applikationsmodellen. I klassediagrammerne i de næste afsnit er der kun anvendt multiplicitet på relationerne, hvis der er en multiplicitet der er større end 1-til-1.

2.3.1 Oprettelse af brugerprofil

For User Sign Up er der lavet et klassediagram, samt et statemachine diagram. Klassediagrammet ses på figur 2.9 og statemchine diagrammet på figur **??**.



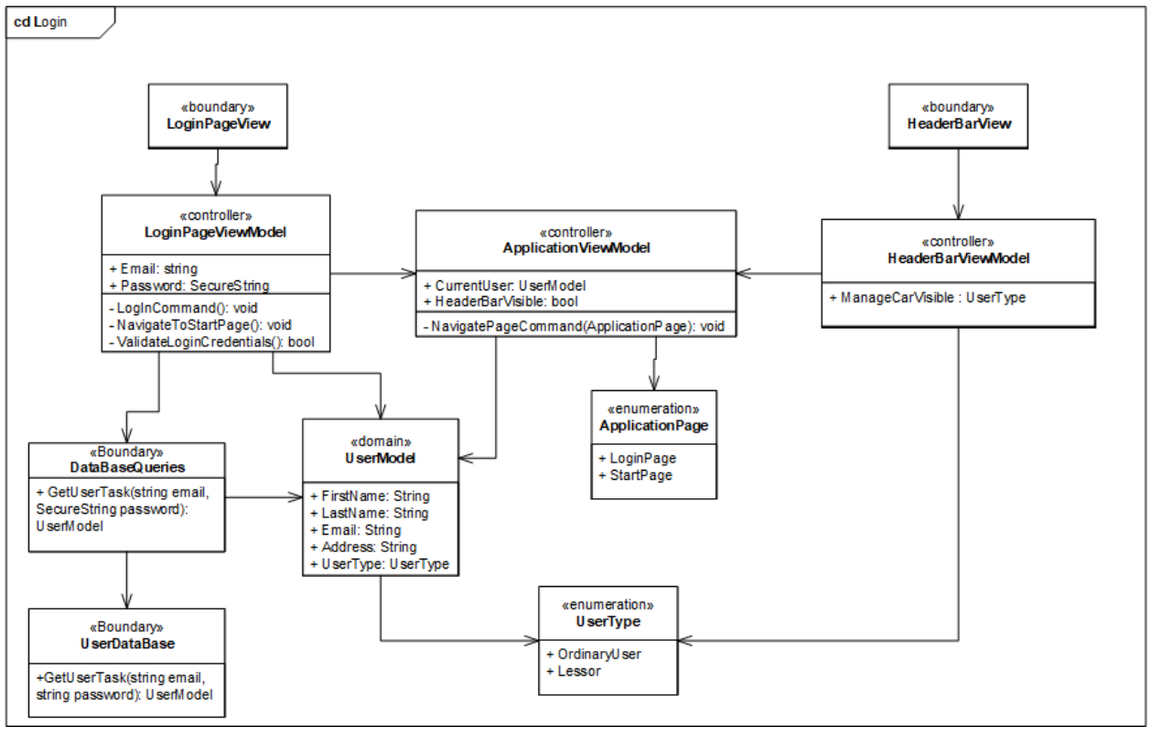
Figur 2.9: Klassediagram for oprettelse af bruger profil.



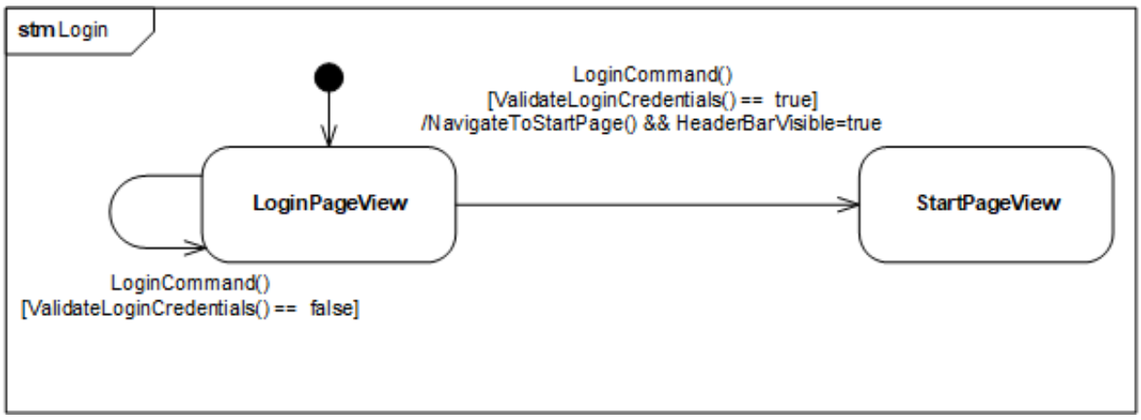
Figur 2.10: Statemachine for oprettelse af bruger profil.

2.3.2 Brugerlogin

Når en bruger har lavet en bruger profil skal han herefter logge ind for at kunne udleje sin bil gennem applikationen eller leje en bil fra en udlejer. Til at beskrive denne funktionalitet er der lavet et klassediagram og en statemachine, som kan ses nedenfor. På figur 2.11 ses klassediagrammet og på figur 2.12 ses statemachine.



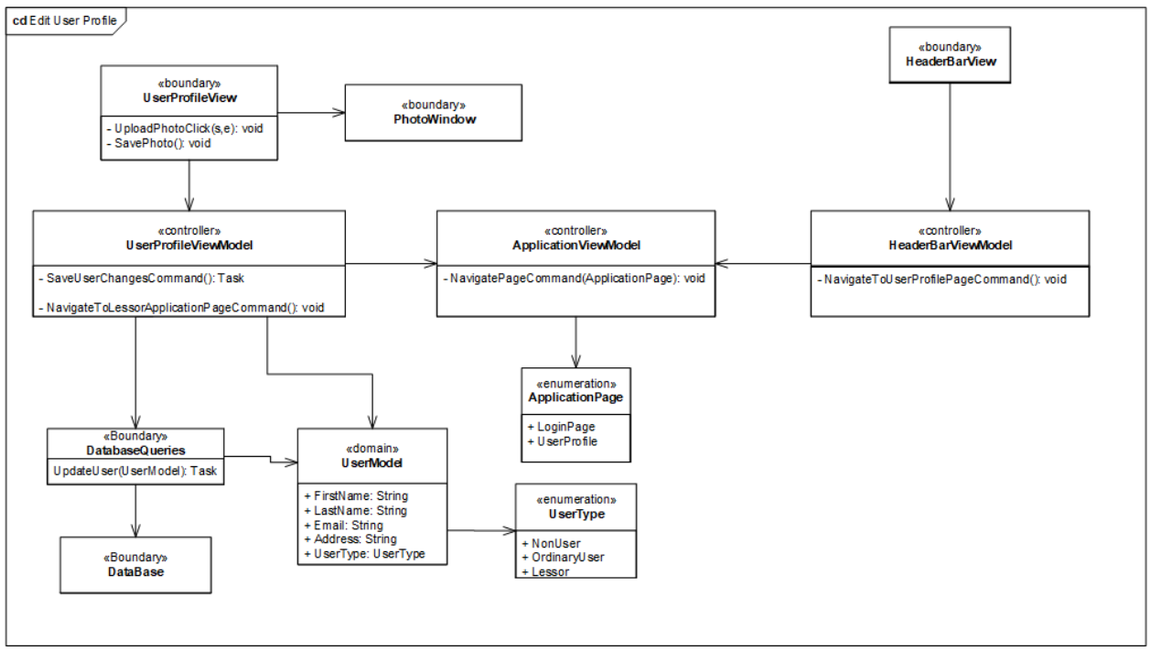
Figur 2.11: Klassediagram for bruger login.



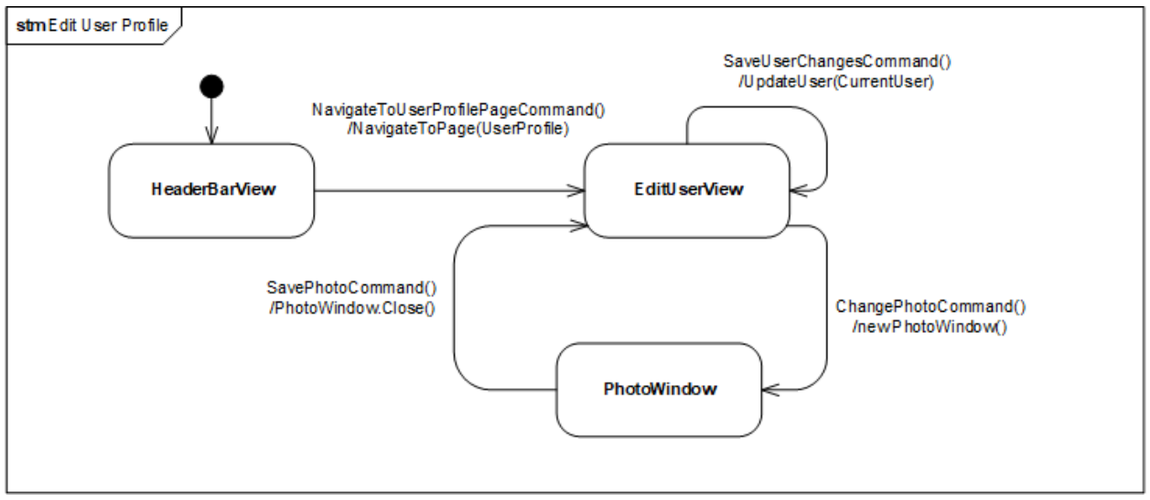
Figur 2.12: Statemachinediagram for bruger login.

2.3.3 Redigering af brugerprofil

Når man som bruger har oprettet en profil, kan det være at ens oplysninger ændrer sig eller man har lavet en fejl. Den logiske arkitektur for dette kan ses på figur 2.13 vises et klassediagram og på figur 2.14 vises et statemachine diagram.



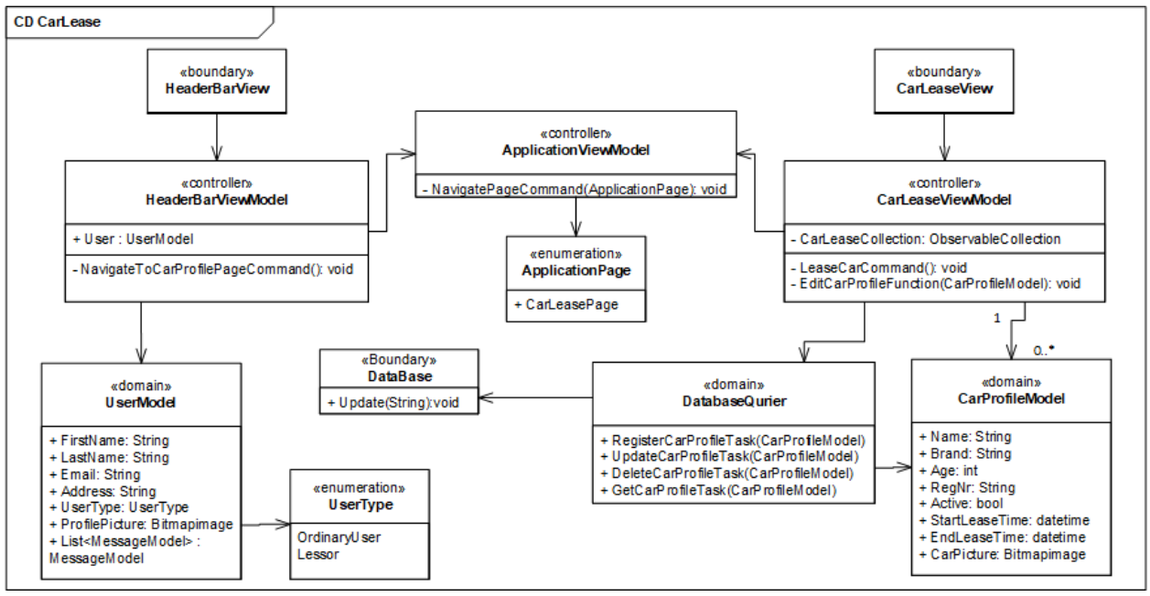
Figur 2.13: Klassediagram for redigering og fjernelse af en brugerprofil.



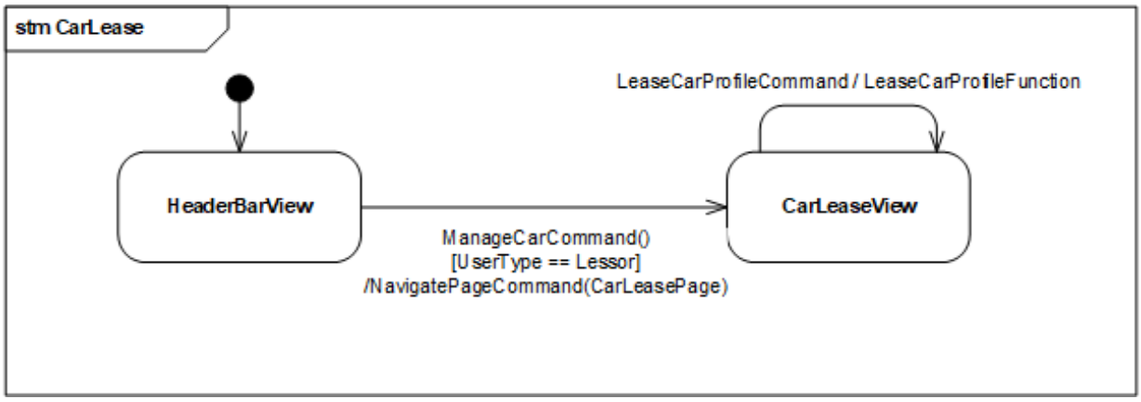
Figur 2.14: Statemachine for redigering og fjernelse af en brugerprofil.

2.3.4 Registrering af bilprofil

Der er også lavet en applikationsmodel, som består af et klassediagram, som ses på figur 2.15, og et statemachine diagram, som ses på figur 2.16.



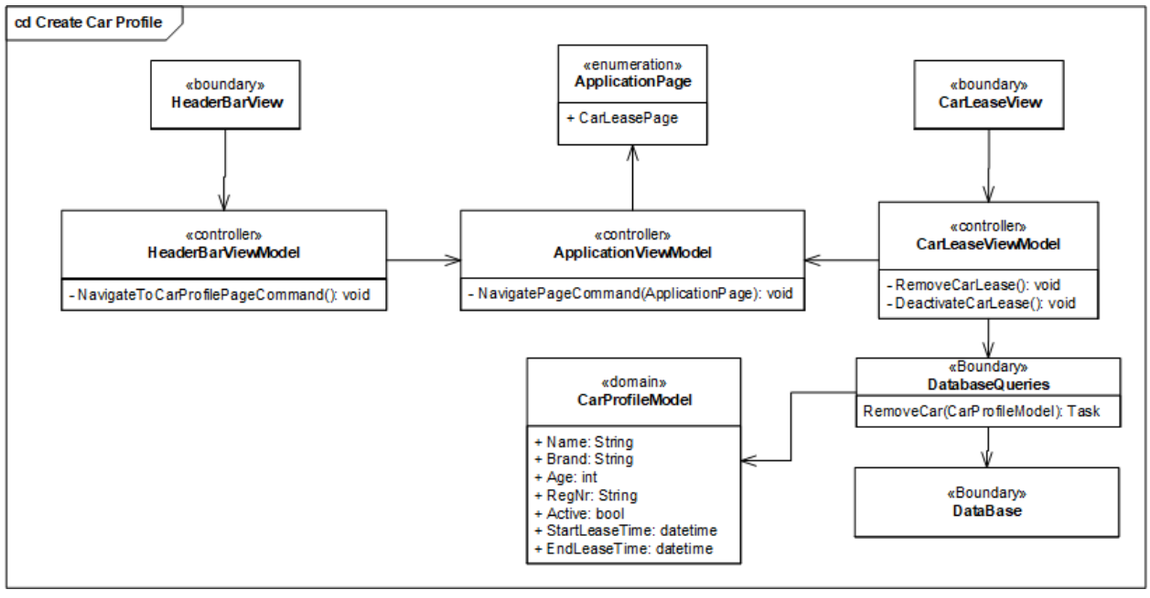
Figur 2.15: Her ses klassediagrammet for tilføjelse af bil til en brugerprofil.



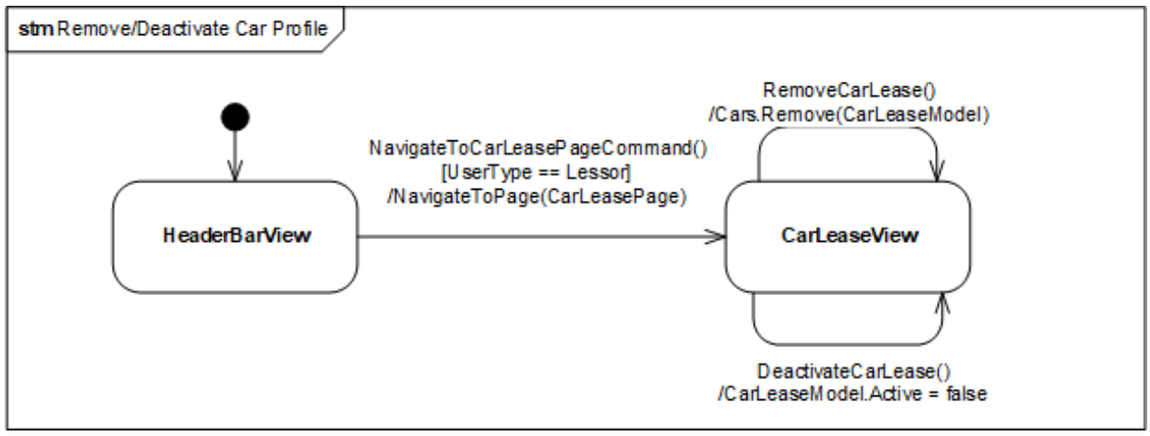
Figur 2.16: Her ses Statemachinediagram for tilføjelse af bil til en brugerprofil.

2.3.5 Fjernelse af bilprofil

Når en bruger ikke længere ønsker at udleje sin bil, så skal brugeren slette bilprofilen. Arkitekturen for denne funktionalitet er beskrevet med et klassediagram et klassediagram på figur 2.17 og et statemachine diagram, som ses på figur 2.18.



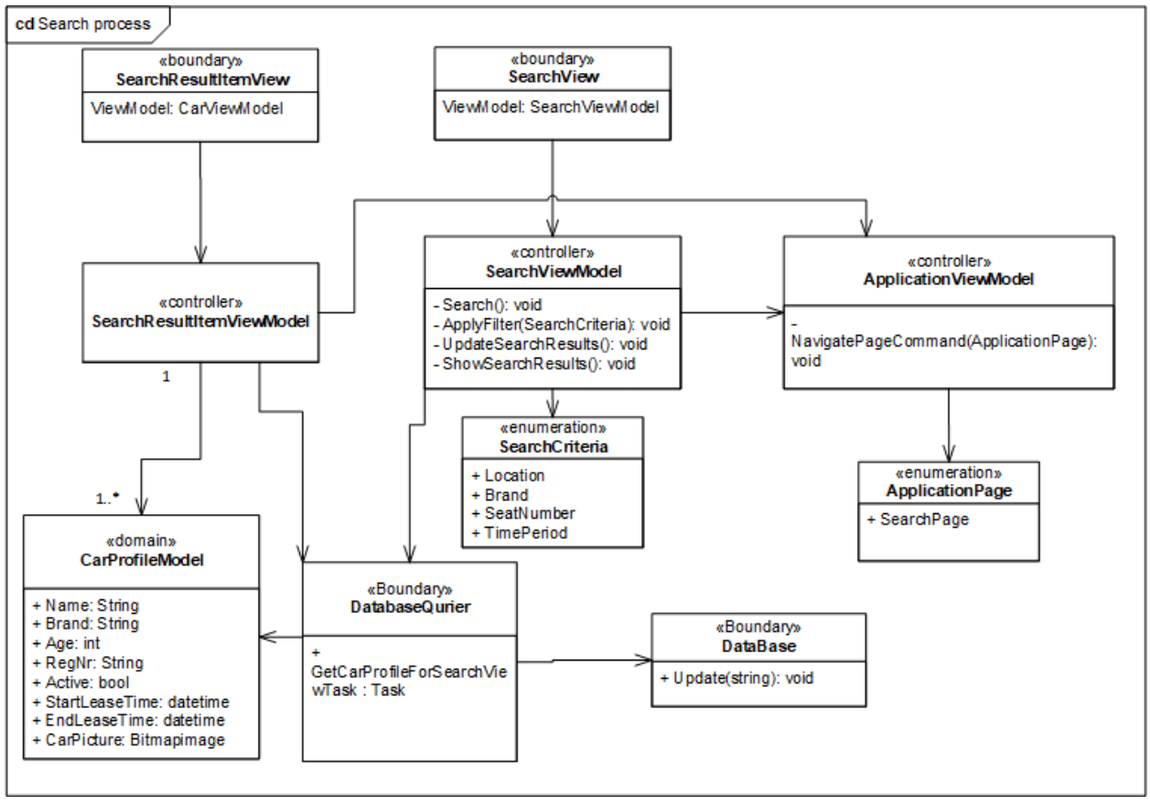
Figur 2.17: Her ses klassediagrammet for fjernelse af bil fra en brugerprofil.



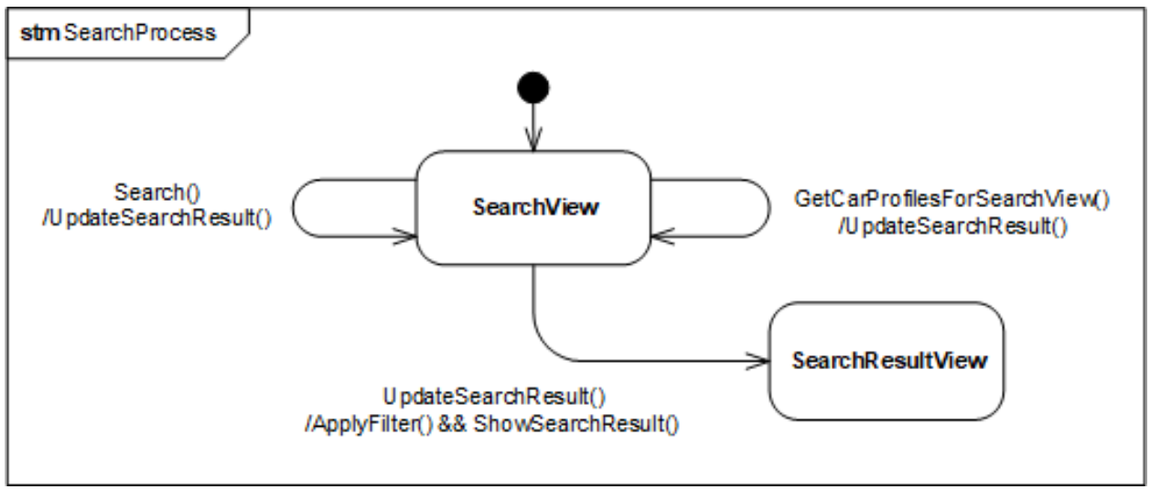
Figur 2.18: Her ses statemachine for fjernelse af bil fra en brugerprofil.

2.3.6 Søgning efter bilprofil

Når en lejer ønsker at leje en bil, så skal han søge efter en i applikationens katalog, hvor udlejer har sat dem til leje. Søgningsprocesen er først beskrevet med et klassediagram, som ses på figur 2.19, og et statemachine diagram, der ses på figur 2.20.



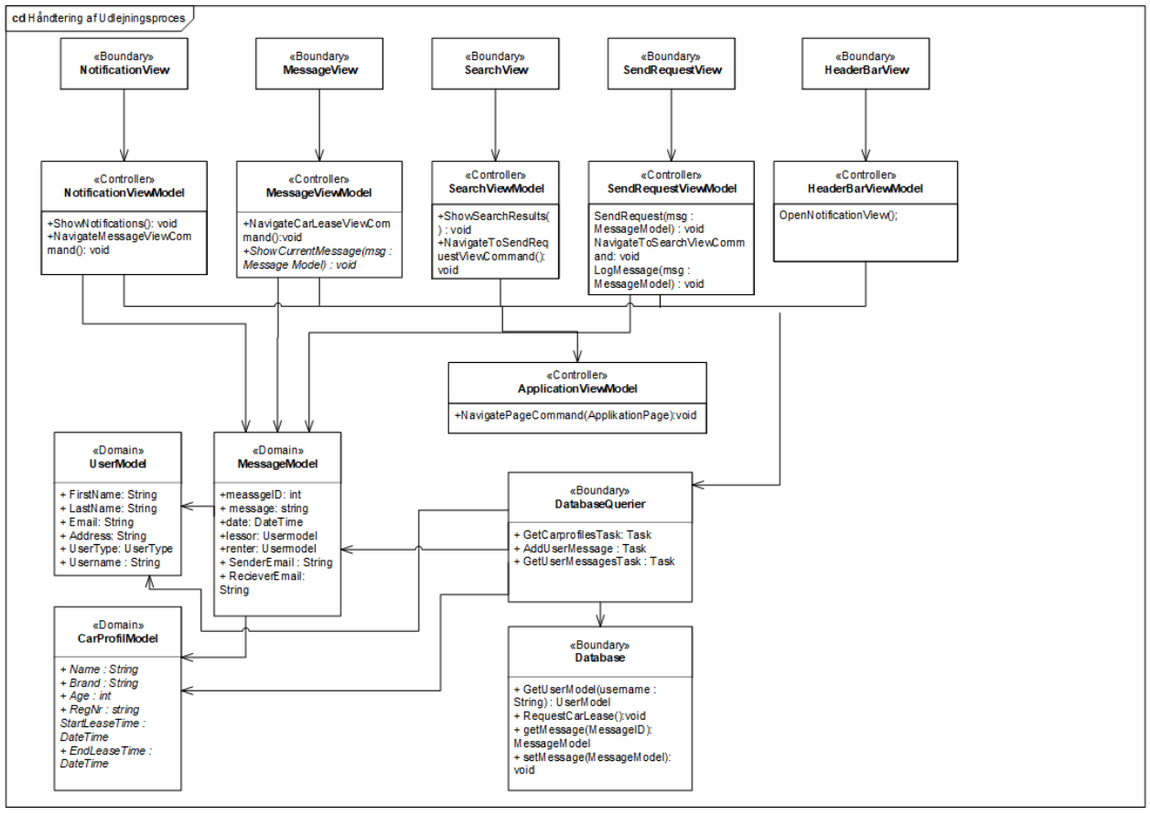
Figur 2.19: Klassediagram for søgning efter biler til leje.



Figur 2.20: Statemachine for søgning efter biler til leje.

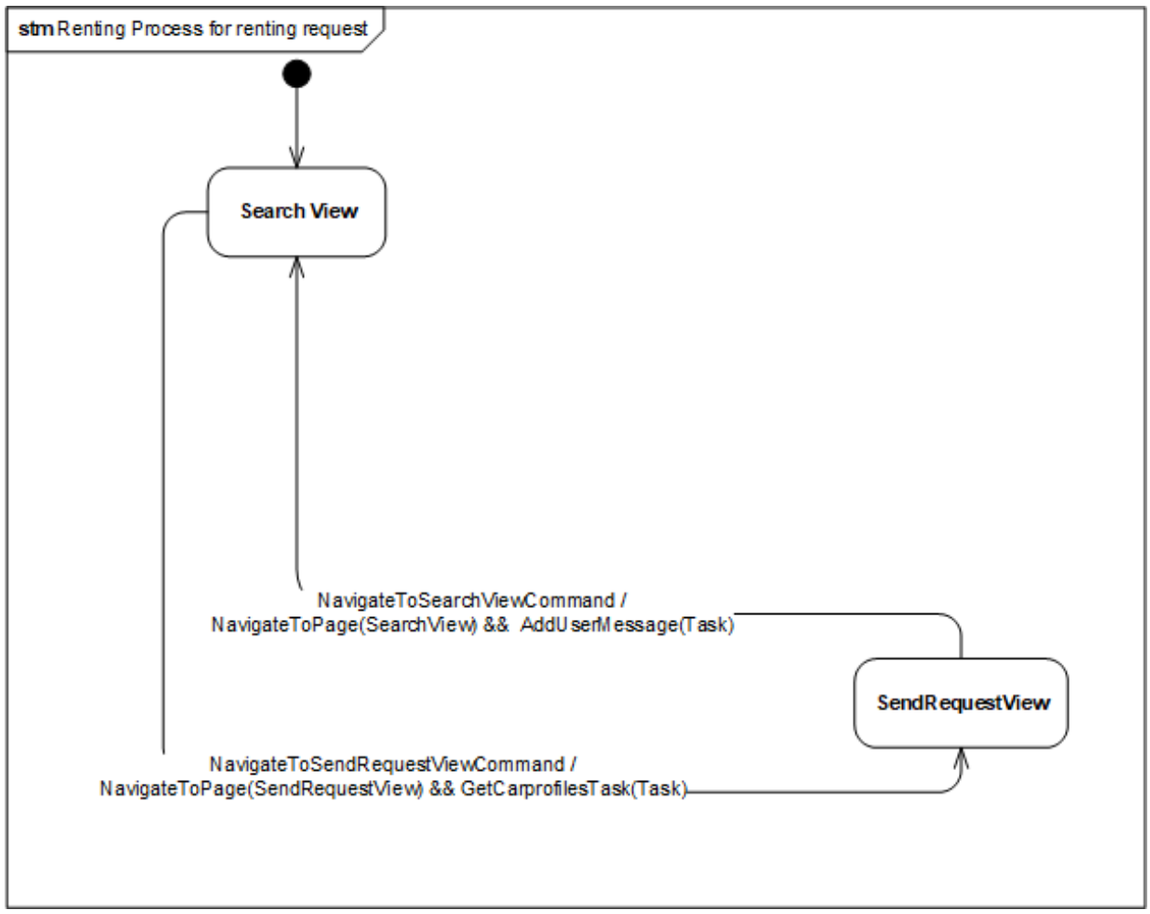
2.3.7 Håndtering af udlejningsproces

Efter at lejer har søgt i applikationens katalog, og har fundet en bil, som lejer ønsker at leje, så starter udlejningsprocessen. Nedenfor ses et klassediagram på figur 2.21 og et statemachinediagram på figur



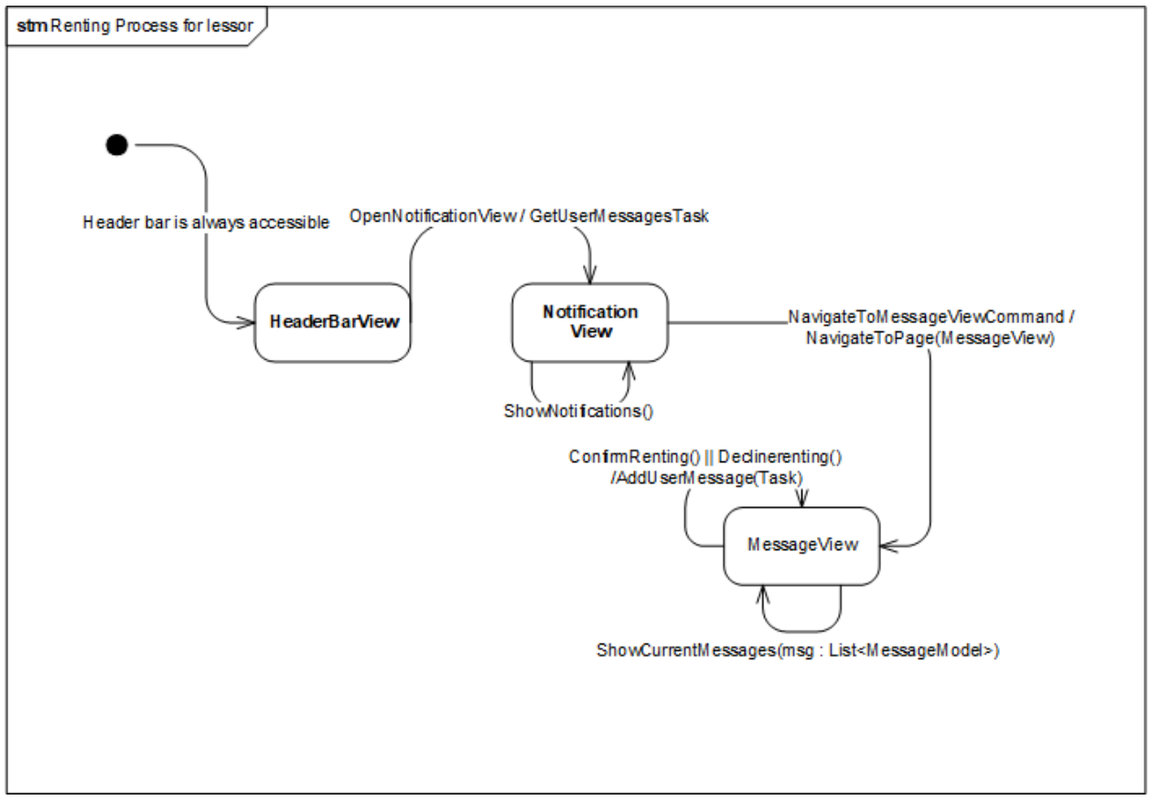
Figur 2.21: Klassediagram for håndtering af udlejningsprocessen.

Herefter er der lavet en opdeling i 3 statemachines, da det giver en bedre forståelse for lige netop denne proces. Det gør den, fordi der er forskellig fokus alt efter om man er *Lessor* eller *Renter* i udlejningsprocessen. Startes der ud med en statemachine for at Renter sender en anmodning til Lessor, så ses det i figur 2.22.



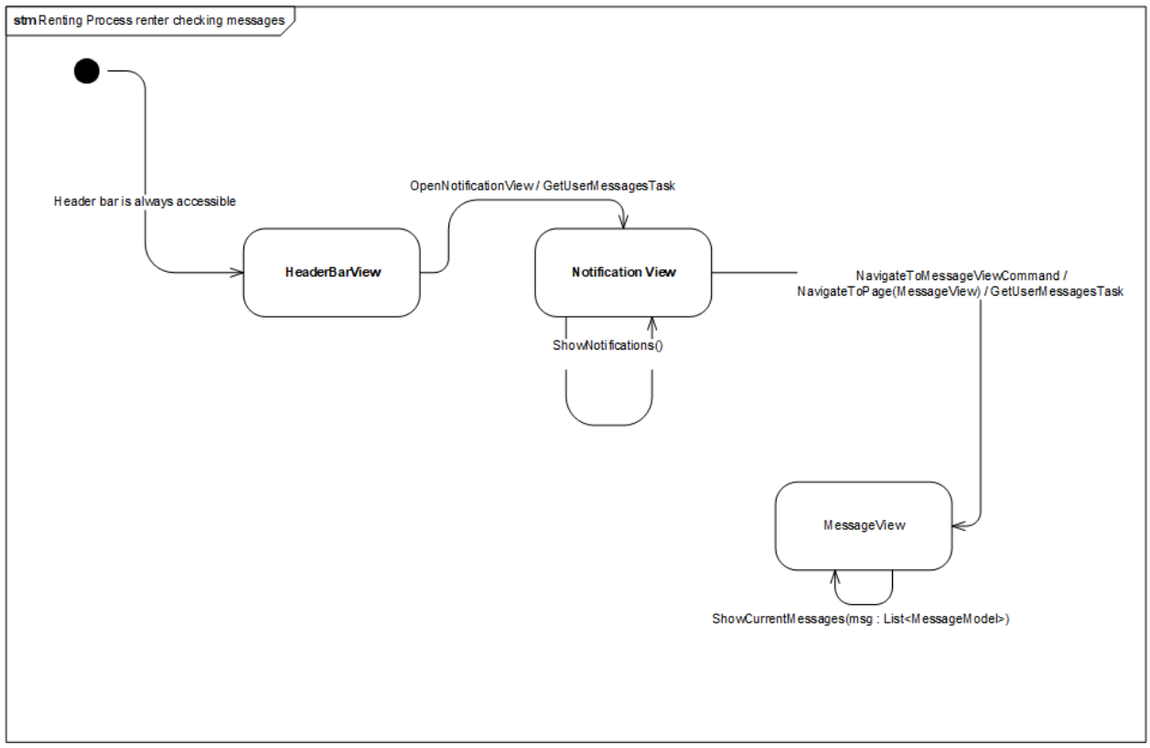
Figur 2.22: Tilstandsdiagram for håndtering af udlejningsprocessen, når lejeren requester at leje en bil.

Det er herefter relevant at kigge på, hvordan det ser ud hos Lessor, når denne for en anmodning om billeje tilsendt. Dette ses i figur 2.23.



Figur 2.23: Tilstandsdiagram for håndtering af udlejningsprocessen, når udlejeren tjekker sine notifikationer eller beskeder, og giver svar til lejeren.

Til slut er det værd at kigge på det eksempel, hvor Renter modtager et svar på den anmodning som Renter har sendt. Dette ses i figur 2.24.

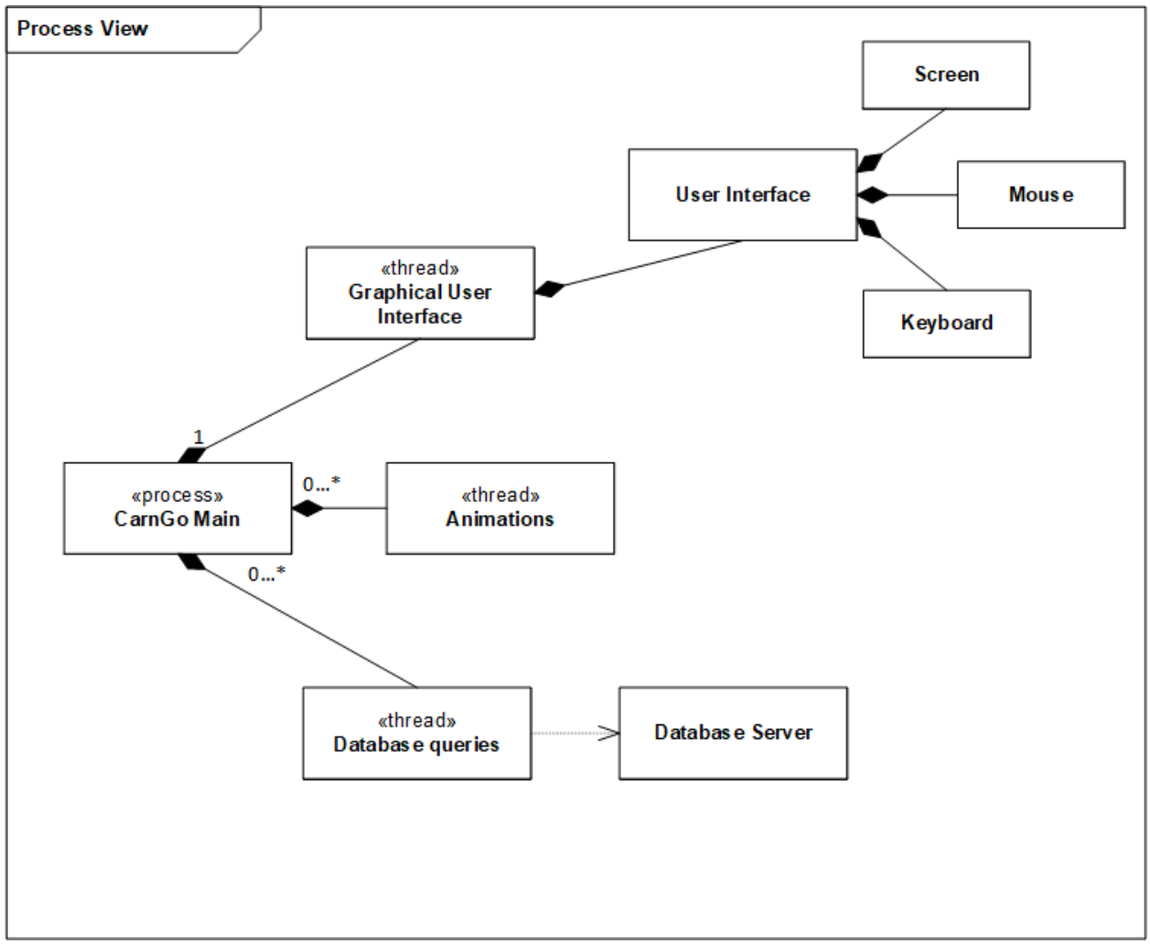


Figur 2.24: Tilstandsdiagram for håndtering af udlejningsprocessen, når lejeren får respons tilbage fra udlejeren.

2.4 Process View

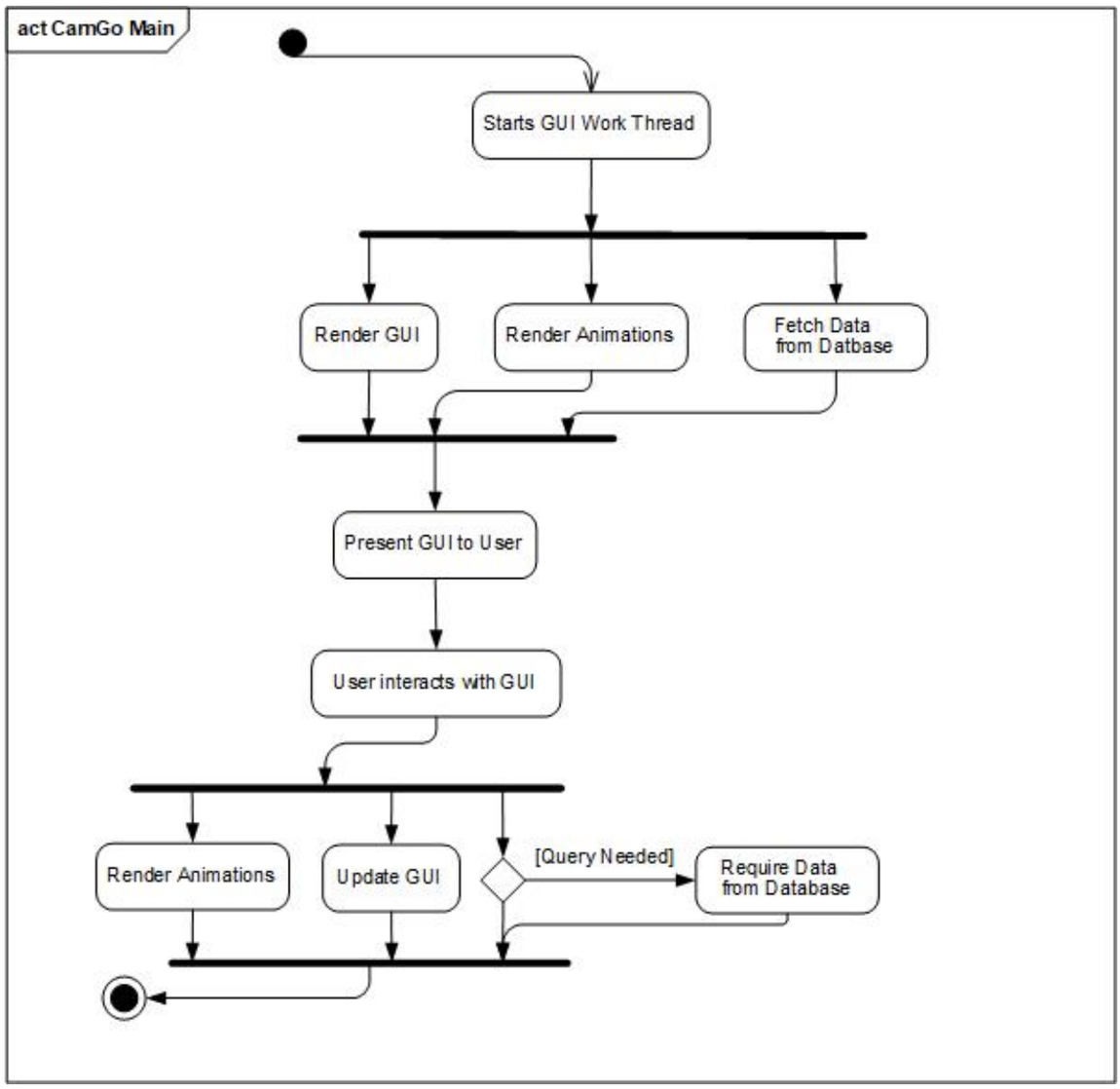
I dette afsnit beskrives systemets dynamiske aspekter. Her angives systemprocesserne, og hvordan de kommunikerer med hinanden (med fokus på systemets runtimeadfærd). Figur 2.25 illustrerer procesnedbrydning af systemet. Her angives aspekter for programmet, som er trådkontrolleret. Systemprocessen har 3 hovedelementer:

- **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 brugeroplevelse 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 til 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 2.25: Komponenter angives som 'kapsler'. En kapsel repræsenterer et trådkontrolleret aspekt af systemet

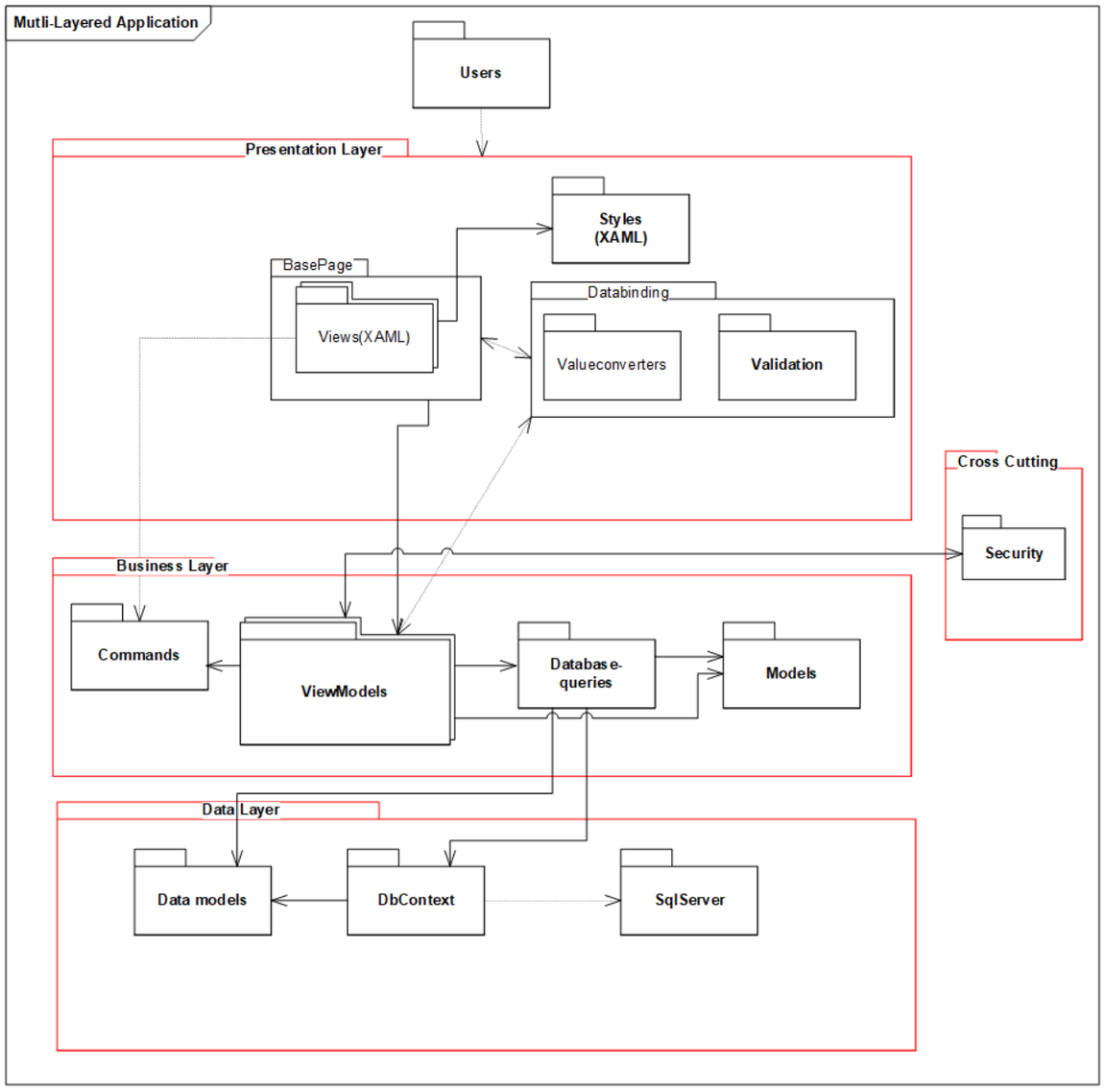
De abstrakte trådkontrolleret aspekter og hvordan de afvikles illustreres i et aktivitetsdiagram (Se figur 2.26). Her ses det hvordan at de grafiske elementer og database queries sker samtidigt. Den grafiske brugeroverflade vil løbende blive indlæst og præsenteret til brugeren. Hvis der er data fra queries, som skal præsenteres til brugeren, vil det også blive indlæst på den grafiske brugeroverflade, men det vil være uafhængigt af GUI-tråden.



Figur 2.26: Aktivitetsdiagram for hovedprocessen for CarnGo

2.5 Development View

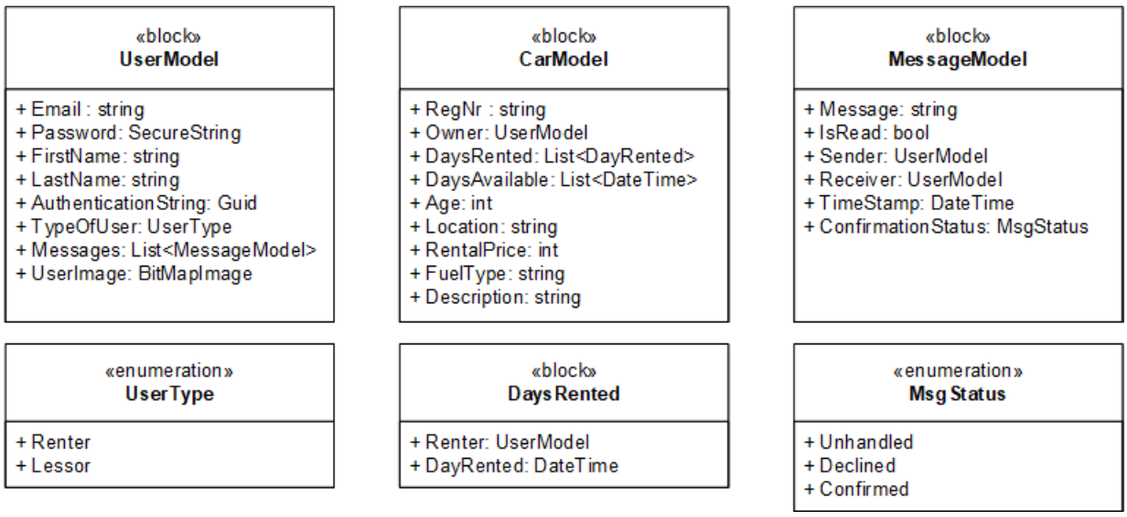
Formålet med dette view er at få overblik over hele applikation. Her illustreres de forskellige lag for applikation og tilhørende moduler og deres forbindelser. Figur 2.27 viser et Model Package diagram, som viser koblingerne mellem lagene og de interne komponenter.



Figur 2.27: Model diagram for Multi-Layered Applikation. En stiplet linje indikerer en forbindelse, som skabes gennem WPF frameworket - eksempelvis databinding

2.5.1 Arkitektur for Modeller

Databasens arkitektur laves ud fra de applikationsmodeller, der er beskrevet tidligere og domæne-modellen2.2. Grunden til dette er, at der her er specificeret domæner for systemet, som skal realiseres i databasen. Der samles derfor de attributter, der er blevet introduceret i de tidligere afsnit. Dette danner grundlag for den arkitektur for databasen, der skal designes ud fra. De samlede domæner med deres attributter beskrevet ses i figur 2.28.

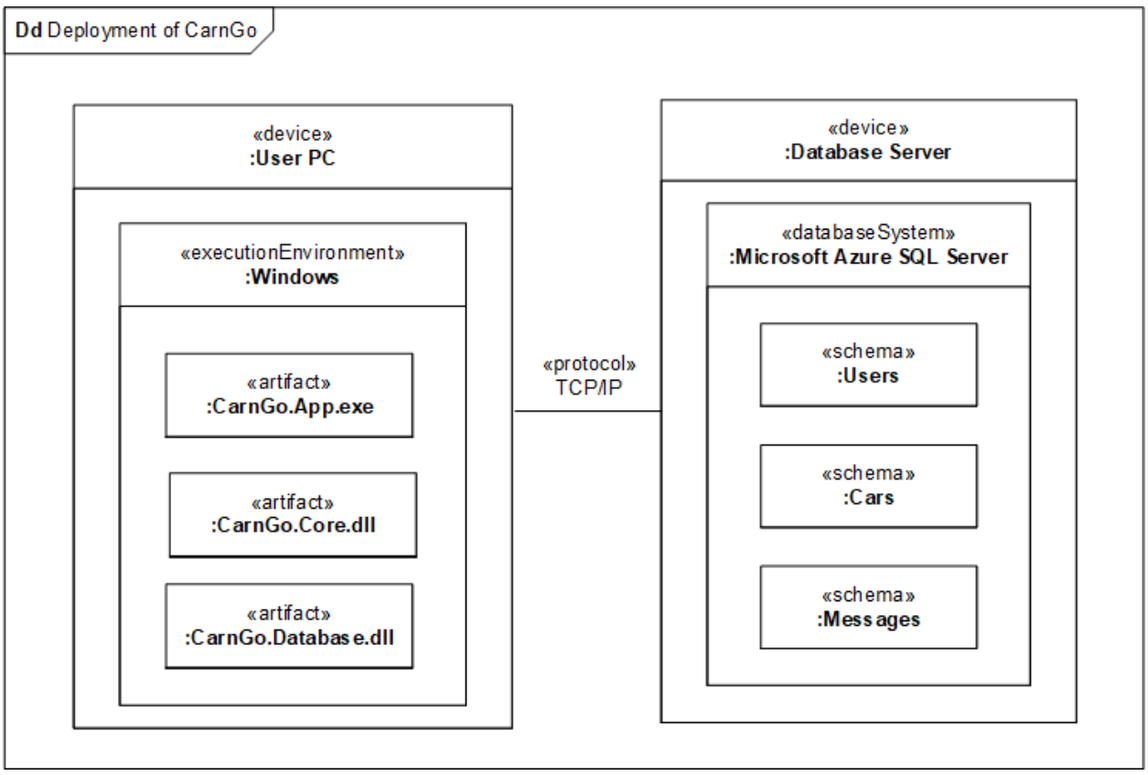


Figur 2.28: Arkitektur for databasens indhold

2.6 Physical View

2.6.1 Introduktion

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 2.29: Deployment View

Af figur 2.29 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 [1]. Dette er den simpleste form for arkitektur, men også den mest skrøbelige. Den er dog valgt, da den gør udviklingen hurtigere og den 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[2], 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.

2.7 Security View

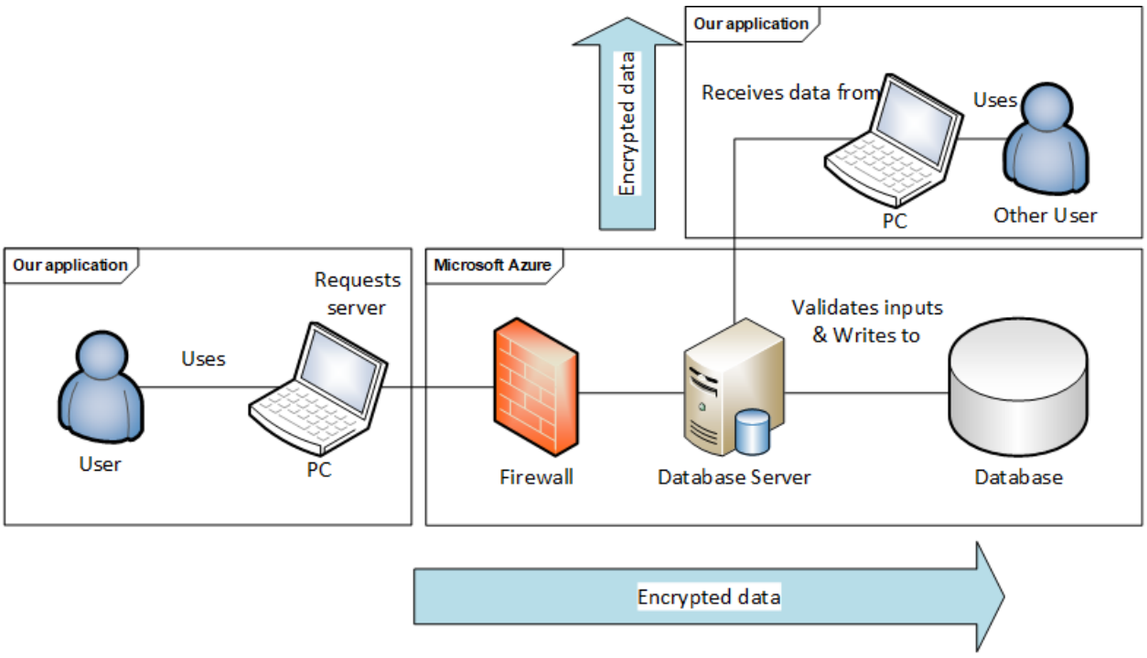
Systemet har en del fortrolige og personfølsomme oplysninger. Det er derfor essentielt, at det fastslås hvilke oplysninger der opbevares, hvordan det skal behandles, samt hvilke trusler der findes, og hvordan det kan løses. I dette afsnit beskrives hvordan systemet kun er tilgængeligt for dem, der har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

2.7.1 Koncepter om brugersikkerhed

Da denne applikation håndterer sensibel data om sine brugere i form af navne, adresser, registreringsnummer på biler osv., så er det 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.

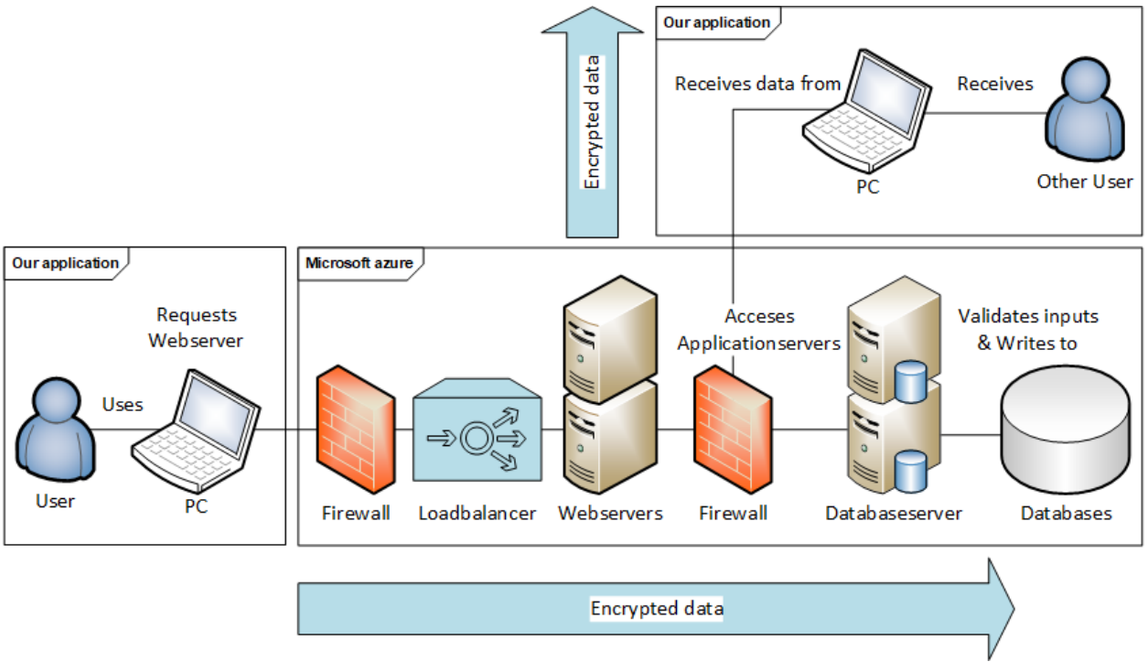
- Brugerens adgangskode holdes ikke i hukommelsen af programmet.
- Brugeren er påduttet en adgangskodesikkerhed ved, at adgangskoden skal indeholde både tal og bogstaver.
- Brugeren er påduttet en adgangskodesikkerhed ved, at adgangskoden skal være længere end 6 karakterer.

Da der ikke selv et system til opbevaringen af data, men i stedet anvendes en Microsoft Azure SQL database fås en bedre sikkerhed mod både ondsindede angreb og for opbevaring af brugernes data[3].
Når en bruger skal ændre, tilføje eller slette information så foregår det også over en sikker https-forbindelse, der sender krypteret data til databaseserveren[4]. I den tidlige arkitektur er der lagt op til det simpleste system, der giver værdi for brugeren. Dette afspejles i figur 2.30.



Figur 2.30: Securitydiagram 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. Derfor specificeres en N-tier arkitektur[5], der efter udviklingen af den første arkitektur, kan arbejdes henimod. Denne ses på figur 2.31.

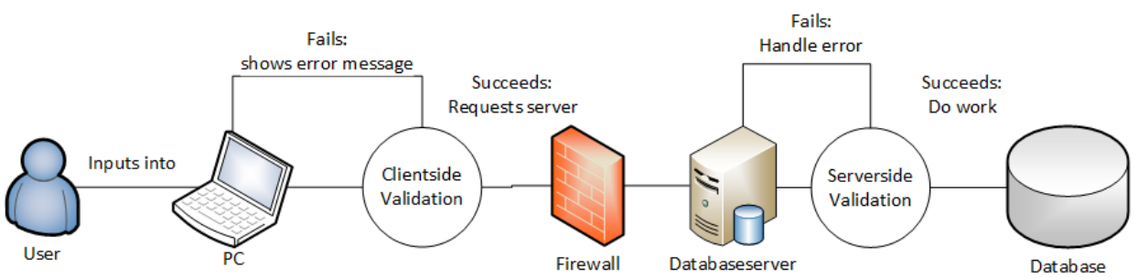


Figur 2.31: Securitydiagram for produktions bruger-til-server interaktion

Grunden til at denne måde er at fortrække i et produktionsmiljø er at en eventuel trussel ikke kan nedlægge hele systemet ved at nedlægge en enkelt webserver. Dog bliver meget af denne sikkerhed igen givet, da der anvendes en ekstern service til at hoste database og eventuel en webserver.

2.7.2 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 email om det er en valid email 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 2.32.



Figur 2.32: Datavalidationdiagram for clientside og serverside valideringer

Da der anvendes en ekstern server til database og eventuel webserver, så er det altså muligt at putte delegere flere valideringsopgaver til serveren, hvilket gør at brugerens applikation får bedre performance.
Et eksempel på en type af validering der kan løses server side, er at kun en bruger kan være logget ind af gangen, så det altid er den nyeste information, der bliver opbevaret i databasen.

3 Database beskrivelse

Databasen for CarnGo skal kunne håndtere biludlejere (Herefter refereret til som til som udlejere), bil lejere (Herefter refereret til som lejere) og biler. Udlejere ejer biler, og lejere kan leje biler i bestemte tidsintervaller valgt af udlejer. Biler skal være tilknyttet en udlejer, som kun kan lejes til **en** lejer ad gangen og skal kunne blive midlertidigt reserveret af lejere uden samtykke fra udlejere imens en anmodning om at leje bilen behandles. Bilen skal frigives fra midlertidigt reservering, hvis udlejer afviser udlejningen. Hvis udlejningen godkendes, skal bilen som udlejet i det angivne tidsinterval. Biler skal have nummerplade og i hvilken stand sidste bruger af den efterlod den i. Biler skal også have et tidsintervaller hvori det er muligt at leje den, så lejere kan søge efter den bil de vil have og i hvilke af de mulige tidsintervaller bilen allerede er udlejet. Udlejere må have så mange biler de vil, og lejere må leje så mange biler de vil. Databasen skal også understøtte funktionaliteten hvor lejer skriver en besked til Udlejer om at leje en af deres biler i et bestemt tidsinterval og udlejer skriver en besked tilbage hvis de ikke afviser anmodningen.Dette skal kunne foregå igennem databasen.

Referencer

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

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

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

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

[5] Tejas Dakve, „Software Integrity Blog“, *Synopsys*, s. 6, 2017. side: https://www.synopsys.com/blogs/software-security/attributes-of-secure-web-application-architecture/.