**Nordic Motorhome Project**

2. semester Interdisciplinary Exam Project

KEA Computer Science

Uddannelsessted: Københavns Erhvervsakademi – Datamatiker

Dato: 4 juni 2020

Studerende: Jonathan Kasper Clement Iversen

Underskrift: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Studerende: Jonas Mørkhøj Hansen

Underskrift: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Studerende: Jacob Ravn

Underskrift: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Studerende: Tobias Salling Jensen

Underskrift: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Indhold

[Problemformulering 4](#_Toc41910575)

[Samarbejdsaftale (Alle) 5](#_Toc41910576)

[**ITO** 6](#_Toc41910577)

[Feasibility Study (Alle) 6](#_Toc41910578)

[Risk Analysis (Alle) 7](#_Toc41910579)

[SWOT Analysis for Nordic Motorhome (Alle) 8](#_Toc41910580)

[Stakeholder Analysis (Alle) 8](#_Toc41910581)

[**Software design** 10](#_Toc41910582)

[Phase Plan (Alle) 10](#_Toc41910583)

[Unified process 10](#_Toc41910584)

[Supplementary Specification (Non-functional requirements) (Alle) 11](#_Toc41910585)

[Use Case Diagram 12](#_Toc41910586)

[Use Cases 12](#_Toc41910587)

[Brief (Use Case: Different Drop-off) (Alle) 12](#_Toc41910588)

[Brief (Use Case: View Details) (Alle) 13](#_Toc41910589)

[Casual (Use Case: Register Pick-up) (Alle) 13](#_Toc41910590)

[Fully Dressed (Use Case: Rent Motorhome) (Alle) 13](#_Toc41910591)

[Fully Dressed (Use Case: Cancel Motorhome) (Alle) 15](#_Toc41910592)

[Fully Dressed (Use Case: Return Motorhome) (Alle) 16](#_Toc41910593)

[Conceptual Model 18](#_Toc41910594)

[System Sequence Diagram (Use Case: Rent Motorhome) 19](#_Toc41910595)

[Sequence Diagram (Use Case: Rent Motorhome) 20](#_Toc41910596)

[System Sequence Diagram (Use Case: Return Motorhome) 21](#_Toc41910597)

[Sequence Diagram (Use Case: Return Motorhome) (Jonathan) 22](#_Toc41910598)

[Design Class Diagram (før vi begyndte at kode) (Jonathan) 23](#_Toc41910599)

[Design Class Diagram (efter vi var færdige med at kode) (Jonathan) 24](#_Toc41910600)

[GRASP Responsibilities 25](#_Toc41910601)

[GRASP diagram polymorphism (Jonathan): 25](#_Toc41910602)

[GRASP diagram controller: 25](#_Toc41910603)

[GRASP diagram create motorhome: 26](#_Toc41910604)

[**Software Construction** 27](#_Toc41910605)

[Java, Controllere, HTML og CSS (kommenteret udklip) 27](#_Toc41910606)

[http error håndtering (Jonathan): 27](#_Toc41910607)

[Filter ift. forskellige parametre: 28](#_Toc41910608)

[Bilag 30](#_Toc41910609)

[Risk Analyse (bilag 1) 30](#_Toc41910610)

[Design klasse diagram (efter færdig kode) (bilag 2) 31](#_Toc41910611)

# Problemformulering

# Samarbejdsaftale (Alle)

I forbindelse med vores første års eksamensprojekt vedrørende case om Nordic Motorhome Project, har vi udarbejdet en samarbejdsaftale for bedre at kunne administrere vores gruppearbejde.

Reglerne er som følgende:

§ 1 Jonas fungerer som projektorganisator, alt beslutningsmæssigt fastlås af projektorganisatoren.

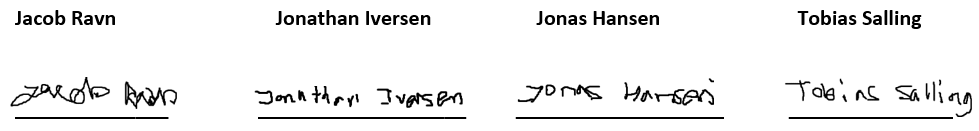
§ 2 Hvis der skulle være uenigheder, henvises der til § 1.

§ 3 Der bliver taget udgangspunkt i faseplanen, i løbet af projekt arbejdet.

Stk. 1. Hvis en milepæl ikke bliver nået, vil dette skulle indhentes i næste fase.

§ 4 Der vil være minimum 2 obligatoriske samlinger hver uge, hvor der bliver arbejdet sammen.

Stk. 1. Disse samlinger kan foregå hjemmefra, ved brug af discord. Grundet nuværende Covid-19 situation.



# **ITO**

## Feasibility Study (Alle)

Hvis man kigger på om vores projekt er teknisk feasible, så kan vi hurtigt konkludere at det ikke er et særligt stort produkt, dog kan visse dele være en smule komplicerede. Dette betyder at vi teknisk set ikke burde støde ind i nogle problemer. Da vi allerede har arbejdet med java og Spring før og lavet applikationer der minder meget om, er det heller ikke et problem at producere produkter. Produktet kommer også til at være meget sikkert da den opgave vi har fået, ikke kræver at der er noget online, oveni dette arbejder vi også med SQL-databaser hvilket er en stabil måde at opbevare sin data uden at frygte tab.

Kigger man i stedet på om det er økonomisk feasible, så kan man igen hurtigt se at produktet er meget simpelt og det hurtigt kan produceres. I forhold til cost/benefit er produktet billigt at producere, billigt at vedligeholde og billigt at opretholde, dog skal man også tage højde for hvis applikationen skal rulles ud og ligge på en online server. Hvis man ønsker dette, er der andre økonomiske trin på vejen, dog ikke noget der springer banken. Derimod er det ikke en særligt kompleks applikation og hvis man beslutter sig for at man gerne vil have nye funktioner kan dette nemt implementeres.

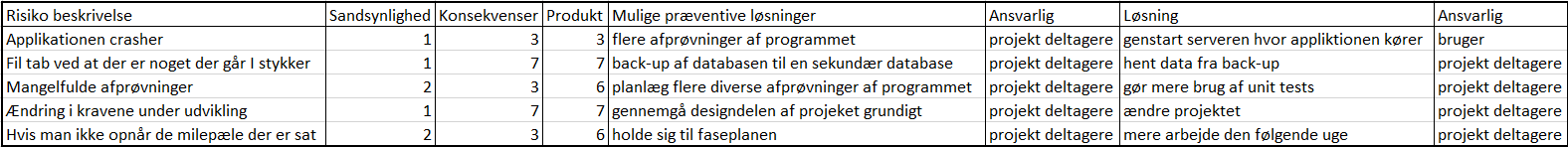
I forhold til de operationelle krav der er til produktet, er de som sagt ikke særligt komplicerede og derfor burde der ikke være nogle problemer. Ledelsen er fuldt ombord på produktet og vi kan nemt gå i dialog angående mulige problemstilling der kunne opstå. Dette gælder også efter produktet originalt er blevet deployed hvis der skulle findes fejl. Dog forventer vi ikke at der kommer til at være problemer efter ”launch” og vi forventer at det kommer til at løse det forretningsmæssige behov.

Den eneste åbne problemstilling for produktet er persondataloven, dog burde dette ikke være et problem da det hele foregår offline, vi skal dog også tage højde for hvis applikationen skal rulles ud til en online funktion. Vi har valgt at tage højde for dette ved at sikre vores database fra SQL injections og lignende. Dog skal man også tage forbehold til licenser og lignende, vi sælger webapplikationen og licensen til Nordic Motorhome og de får derefter ansvaret for vedligeholdelse og andre udgifter der kunne fremkomme. Dog fikser vi eventuelle bugs eller fejl der er hvis det er nødvendigt.

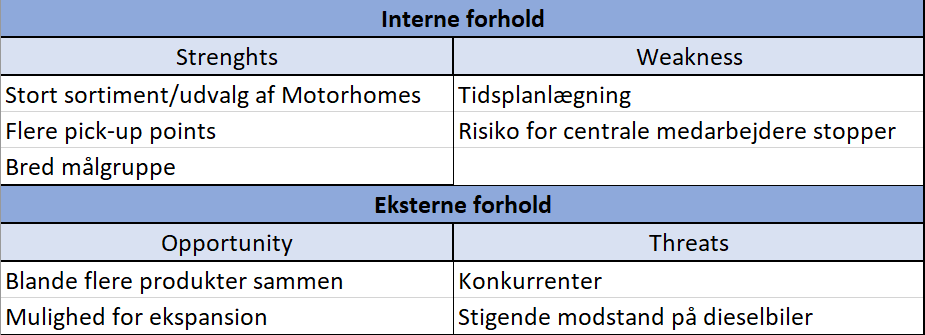
Vi foreslår som sædvanlig at checke om der allerede eksisterer applikationer der kunne løse de administrative problemer kunden har da det er et almindeligt problem, der bare skal specialiseres. Fordelene ved at de får lavet deres egen applikation er at det kommer til at være mere simpelt og derved mere brugervenligt.

For at konkludere til sidst om det er et go eller no go til applikationen vil vi nok foreslå kunden at undersøge om der ikke allerede eksisterer en applikation der lever op til deres behov. Det kommer til at være nemmere og hurtigere for dem at finde en applikation og nemmere på længere sigt i forhold til vedligeholdelse. Hvis de beslutter sig for at få lavet applikationen hos os vil det blive mere simpelt og brugervenligt i forhold til deres behov. Dog er produktet ellers muligt at lave hvis de ikke finder en gyldig applikation.

## Risk Analysis (Alle)

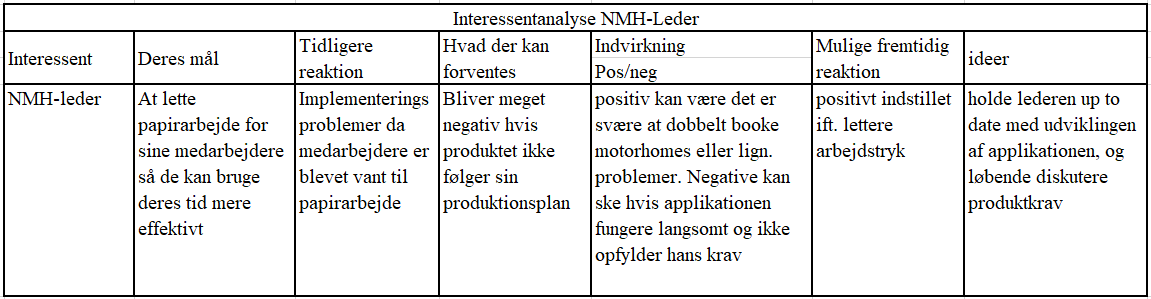
Vi har som udviklere af vores applikation til Nordic Motorhome valgt at lave en risikoanalyse, det har vi gjort for at være så godt forberedte som muligt til at gribe vores opgave an og for at danne os en oversigt over de mulige risici der kunne være forbundet med projektet. Vi har sammen brainstormet og fundet frem til hvilke risikoer der kunne være forbundet med dette projekt, dette har vi gjort under kolonnen *”Risiko beskrivelse”*, derefter har vi givet den givne risiko en sandsynlighed fra værdien 1 – 5 og en konsekvens i værdierne 1, 3, 7, 10 (1 ubetydelig – 10 katastrofal). Sandsynligheden ganger vi med vores konsekvens og får derved et produkt. Herefter beskriver vi vores mulige præventive løsninger til risikoen, hvem der er ansvarlig, løsningen og de ansvarlige for risikoen.

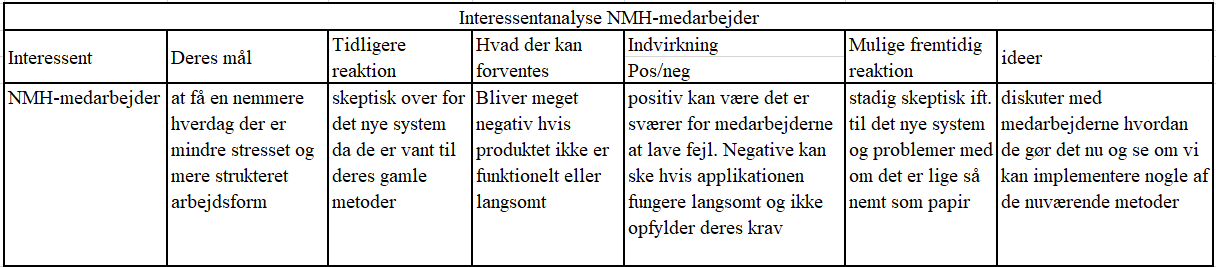
## SWOT Analysis for Nordic Motorhome (Alle)

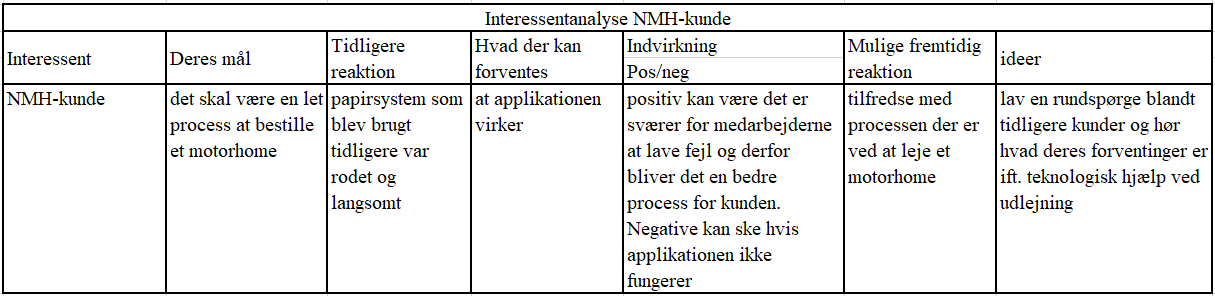
I vores gruppe har vi valgt at lave en SWOT-analyse for virksomheden Nordic Motorhome. Dette har vi gjort for at få skabt et bedre overblik over deres virksomhed, samt en bedre forståelse over deres forskellige forretningsområder og stillinger de har i virksomheden. En SWOT-analyse hjælper også virksomheden med at vurdere et nyt produkt og i dette tilfælde har vi vurderet at Nordic Motorhomes svage side er tidsplanlægning, fordi de har meget papirarbejde og manuelt arbejde i at indtaste bookinger, kunder og registrering af motorhomes. Derfor ville det give god mening at lave et program der kan sørge for dette. Under Nordic Motorhomes eksterne forhold har vi valgt at sige at de har mulighed for ekspansion, til at starte med i de skandinaviske lande og deraf vil vores program også være relativt let at opdatere på sproget og udenlandske valutaer.

## Stakeholder Analysis (Alle)

En interessentanalyse er et godt værktøj til at få et overblik og beskrive de forskellige interessenter der kunne have en interesse i det projekt vi bygger til deres organisation. Derfor har vi valgt at lave en interessentanalyse over Nordic Motorhomes ledere, medarbejdere og deres kunder. Nordic Motorhomes ledere er en vigtig interessent i vores projekt da de netop har taget fat i os for at optimere deres virksomheds daglige arbejdsgang, mht. at gøre arbejdet mere struktureret og overskueligt for deres medarbejdere. Nordic Motorhomes medarbejdere har vi også valgt at lave en interessentanalyse over, da det er dem som kommer til at arbejde mest med vores færdige produkt, her vil det primært være salgsassistenterne og bookkeeper som kommer til at bruge vores produkt. Til sidst har vi valgt at lave en interessentanalyse over Nordic Motorhomes kunder, da de også vil være interesseret i en nem og praktisk fremgangsmåde hvor alle tingene forløber lige til ved udlejning af et motorhome.

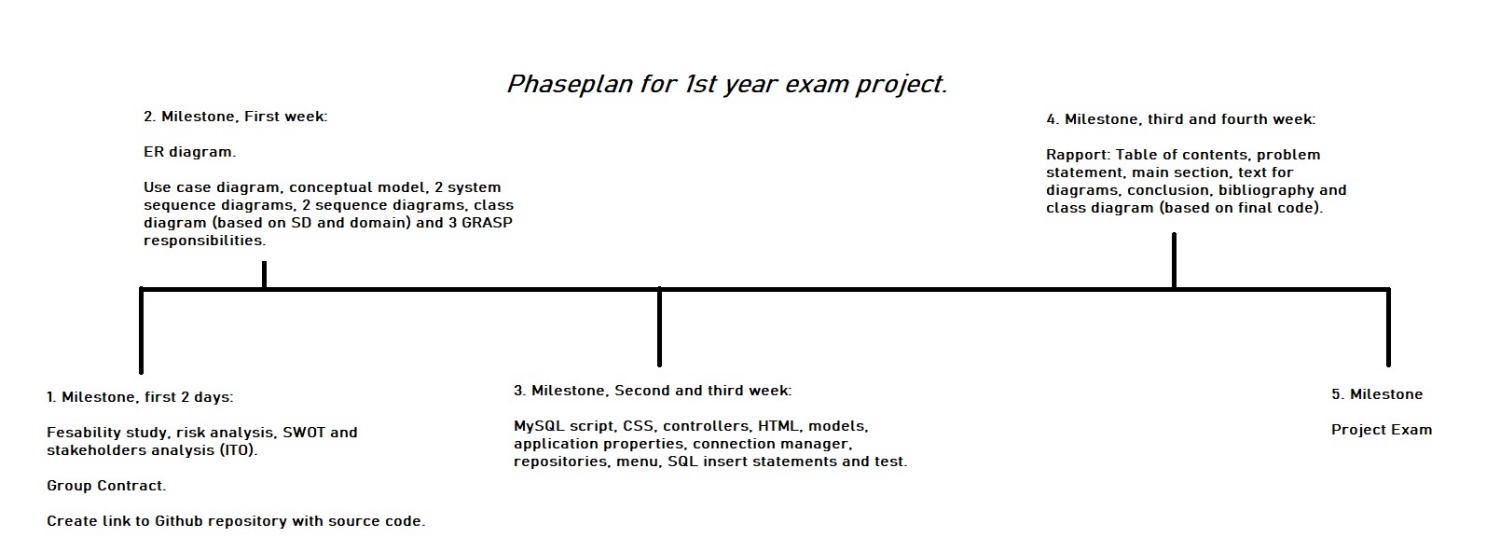
Interessentanalyse for Nordic Motorhome lederne/ejerne:

Interessentanalyse for Nordic Motorhome medarbejderne:

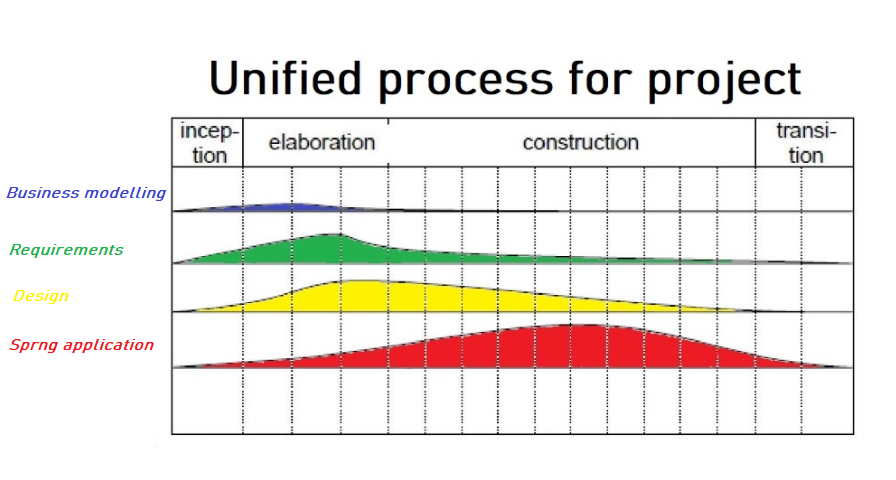
Interessentanalyse for Nordic Motorhomes kunder:

# **Software design**

## Phase Plan (Alle)



## Unified process

Vi har lavet vores Unified Process i starten af vores projekt, lige efter vores faseplan. Unified Process er en model som giver et godt overblik over hvordan vi kommer til at gå igennem de fire faser, Inception, Eloboration, Construction og Transition. Forberedelsen (Inception) er den fase der giver os det fulde overblik over hvilke krav vores system skal have implementeret. Etableringsfasen (Elaboration) er der hvor vi har fokus på at udvikle de centrale dele af systemet, vi ser herunder en stor del på vores requirements til Nordic Motorhomes krav til vores projekt. Under Elaboration kigger vi også på design, hvor vi udvikler Use Cases og diagrammer indenfor SWD. I konstruktionsfasen (Construction) udvikler vi på vores kode i Java, MySQL, HTML og CSS. I vores overdragelse af programmet (Transition) bliver de sidste små rettelser i koden lavet og programmet overdrages til Nordic Motorhome, i form af projektfremlæggelse og report.

## Supplementary Specification (Non-functional requirements) (Alle)

**Usability:**

Vores program skal være let at bruge, man skal ikke igennem en lang proces for at udføre trivielle funktioner, som at oprette en lejeaftale. Det skal derfor også være nemt at tilgå redigering af aktive motorhomes, tilføjelser af nye motorhomes når firmaet udvider eller lignende. Det er en del af projekt at det er skalerbart og derfor skal det også være nemt at udvide det. Her kunne man tænke at Nordic Motorhome kunne overveje at udvide til andre skandinaviske lande og dertil vil det være nemt at implementere programmet så det er tilpasset andre skandinaviske lande. Nordic Motorhome kunne også overveje at lave en portal til deres kunder, så de selv ville kunne leje et motorhome. Dette vil være en større udviklingsfase, men bestemt muligt da mange af funktionerne og metoderne vil blive brugt igen.

**Reliability:**

Applikationen kommer til at læne sig op ad en SQL server ift. at gemme sine motorhome og kunde data, i sammenhæng med SQL sikrer vi os også mod SQL injections og andre problemer SQL kan medføre.

**Performance:**

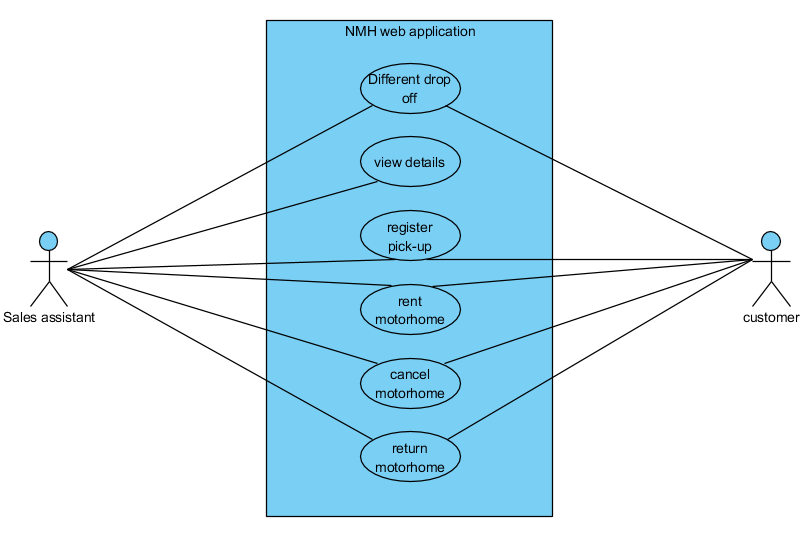
Applikationen skal være skalerbar og skal derfor være brugbar på et større plan så vel som et småt, der er derfor taget højde for memory management.

**Supportability:**

Applikationen skal være nem at ændre uden der skal redigeres i mange forskellige klasser. Oven i dette skal det også være nemt at tilføje funktioner, så det tager vi også højde for.

## Use Case Diagram

Vores Use Case diagram er et simpelt og godt diagram som viser brugernes interaktion med vores system. I Use Case diagrammet har vi derfor vores ”Actor” til venstre for diagrammet, som i dette tilfælde er Nordic Motorhomes Salgs assistent. Vores salgsassistent har associationer til alle vores Use Cases, da de kommer til at have en indflydelse igennem hele vores program. På højre side af vores diagram er Nordic Motorhomes customers som vi har valgt at associationer til nogle af vores Use Cases, da de har indflydelse på at den specifikke Use Case kan fungere.



## Use Cases

### Brief (Use Case: Different Drop-off) (Alle)

Salgs Assistenten åbner programmet og vælger fra Homepage ”*Hand In Motorhome*”. Derefter vælger medarbejderen ”*Edit details about drop-off*” for det gældende Motorhome. Sales Assistant ændrer ”*Drop-off location*” til den nye lokation og trykker derefter ”*Submit*”

### Brief (Use Case: View Details) (Alle)

Salgs Assistenten åbner programmet og vælger fra Homepage ”*Rent Motorhome*” og trykker ”*See Details*” på det specifikke Motorhome. Salgs Assistenten tjekker information på Motorhomet og lukker derefter programmet.

### Casual (Use Case: Register Pick-up) (Alle)

Main success scenario: Salgs Assistenten åbner programmet og vælger fra Homepage ”*Register Pick-up*” vælger det Motorhome som kunden ønsker. Salgs Assistenten indtaster derefter

nødvendige data på kunden og bliver derefter promptet med ”*Correct Motorhome*” og ”*Correct Data*”. Hvis data er korrekt oprettes en pick-up til kunden.

Alternative scenarios:

1. Hvis den indtastede data er forkert.

a.1) Salgs Assistenten redigerer dataene fra ”*waiting for Pick-up*” listen.

a.2) Accepterer data er korrekt og gemmer.

1. Hvis der efter Pick-up, bliver fundet fejl i indtastningen af Pick-up.

b.1) Salgs Assistenten vælger ”*See active Motorhomes*”.

b.2) Redigerer data for gældende motorhome og gemmer.

### Fully Dressed (Use Case: Rent Motorhome) (Alle)

**Use case:** Rent Motorhome

**Scope**: Nordic Motorhome Webapplikation.

**Level:** User goal

**Primary actor:** Salgs Assistenten

**Stakeholders and interest:**

* Salgs Assistent: Ønsker let brug af program, ved indtastning af data.
* Ejer: Ønsker at webapplikationen virker nemt og let. Ønsker også at der ikke er fejl der leder til problemer ved Motorhomes når de skal afleveres eller data skal ændres.
* Kunder: Ønsker nem proces ved registrering af lejeperiode.

**Preconditions:** Salgs Assistenten og kunde har aftalt at der skal lejes et eller flere motorhomes.

**Success guarantee (postcondition):** registreringen af leje af motorhome er oprettet og kunden har fået prisen.

**Main success scenario (basic flow):**

1. Ansat modtager lejeanmodning og skal oprette ny lejeperiode for motorhome.
2. Vælger fra homepage ”*Rent Motorhome*”.
3. Indtaster gældende dato for leje.
4. Vælger et af de tilgængelige motorhomes.
5. Indtaster kundens data og andre kontraktmæssige forhold.
6. Submitter oplysninger.
7. Den ansatte får prisen og videregiver den.

**Extension (alternative flow):**

6.a) Skriver forkert information.

a.1) Vælger ”*See active motorhomes*”.  
 a.2) Vælger motorhome der skal redigeres.  
 a.3) Retter sin fejl.

**Special requirements:**

* Salgs Assistenten skal bruge en pc.
* En database over motorhomes.

**Frequency of occurrence:**

* Hver gang en ny anmodning kommer ind.

**Open issues:**

* GDPR.
* Der kan kun vælges motorhome ud fra dato, og ikke ud fra motorhome-type

### Fully Dressed (Use Case: Cancel Motorhome) (Alle)

**Use case:** Cancel Motorhome

**Scope**: Nordic Motorhome Webapplikation.

**Level:** User goal

**Primary actor:** Salgs Assistent.

**Stakeholders and interest:**

* Salgs Assistent: Ønsker let brug af program, ved indtastning af data.
* Ejer: Ønsker at webapplikationen virker nemt og let. Ønsker også at der ikke er fejl der leder til problemer ved Motorhomes når de skal afleveres eller data skal ændres.
* Kunder: Ønsker nem proces ved aflysning af lejeperiode.

**Preconditions:** Salgs Assistent og kunde har aftalt at der skal aflyses et eller flere motorhomes og der er oprettet en lejekontrakt.

**Success guarantee (postcondition):** registreringen af aflysning af motorhome er gennemført og kunden har fået en pris.

**Main success scenario (basic flow):**

1. Ansat modtager aflysnings anmodning og skal aflyse lejeperiode for motorhome.
2. Vælger fra homepage ”*See active Motorhome*”.
3. Vælger det gældende motorhome.
4. Trykker ”*Cancel Motorhome*”.
5. Bliver promptet og accepterer.
6. Den ansatte får pris for aflysning og videregiver information.

**Extension (alternative flow):**

3.a) Hvis Salgs Assistenten vælger et forkert Motorhome.

a.1) Annullerer og går tilbage.  
 a.2) Vælger det korrekte motorhome og aflyser.

5.b) Aflyser forkert motorhome.

b.1) Vælger ”*Rent Motorhome*” fra homepage.  
 b.2) Vælger ”*Create from archive*”.  
 b.3) Vælger et gældende motorhome fra liste og opretter igen.  
 b.4) Går til ”*Cancel Motorhome*” siden og aflyser det rigtige motorhome.?????

**Special requirements:**

* Salgs Assistenten skal bruge en pc.
* En database over motorhomes.

**Frequency of occurrence:**

* Hver gang en ny anmodning kommer ind.

**Open issues:**

* GDPR.

### Fully Dressed (Use Case: Return Motorhome) (Alle)

**Use case:** Return Motorhome.

**Scope**: Nordic Motorhome Webapplikation.

**Level:** User goal.

**Primary actor:** Salgs Assistenten.

**Stakeholders and interest:**

* Salgs Assistent: Ønsker let brug af program, ved indtastning af data.
* Ejer: Ønsker at webapplikationen virker nemt og let. Ønsker også at der ikke er fejl der leder til problemer ved Motorhomes når de skal afleveres eller data skal ændres.
* Kunder: Ønsker nem proces ved aflysning af lejeperiode.

**Preconditions:** Lejekontrakt er udløbet og kunden har afleveret motorhomet

**Success guarantee (postcondition):** motorhome er afleveret og kunden har fået prisen

**Main success scenario (basic flow):**

1. Vælger fra homepage ”*See active Motorhome*”.
2. Vælger det gældende motorhome.
3. Trykker ”*Return Motorhome*”.
4. Indtaster kørte kilometer, tank status og om der er nogle skader.
5. Bliver promptet og accepterer.
6. Den ansatte får pris for lejekontrakten og står for indkrævning.

**Extension (alternative flow):**

2.a) Vælger forkert motorhome.

a.1) Annullerer og går tilbage.  
a.2) Vælger det korrekte motorhome og returnerer.

5.b) Indtastet data forkert.

b.1) trykker nej ved prompt og indtaster korrekt data.

5.c) Returnerer forkert motorhome.

c.1) Vælger ”*Rent Motorhome*” fra homepage.  
c.2) Vælger ”*Create from archive*”.  
c.3) Vælger et gældende motorhome fra liste og opretter igen.  
c.4) Går til page til return siden og returnerer det rigtige motorhome.

**Special requirements:**

* Salgs Assistenten skal bruge en pc.
* En database over motorhomes.

**Frequency of occurrence:**

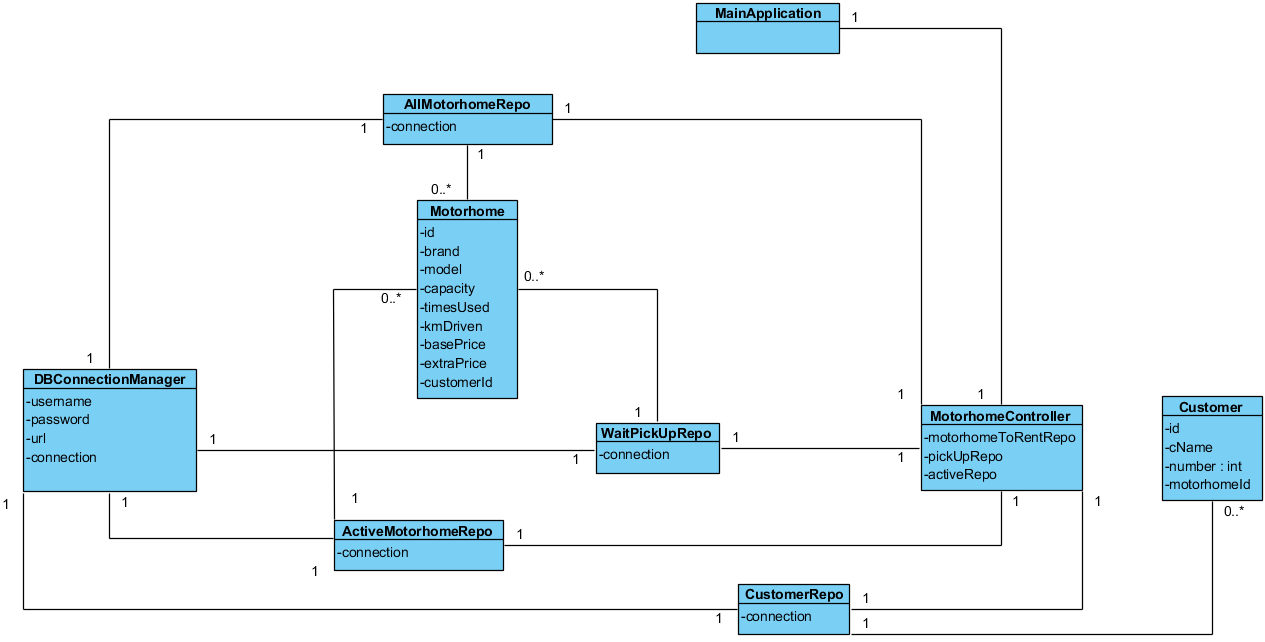
* Hver gang et nyt motorhome skal returneres.

**Open issues:**

* GDPR.

## Conceptual Model

Den konceptuelle model repræsenterer vores system, som vi har tænkt os at lave til Nordic Motorhome. Den konceptuelle model giver andre personer en god viden og forståelse for hvordan vi vil opbygge vores program. Modellen indeholder vores controllere, models og repositories, som vi har tænkt os at udvikle i Java. Hver klasse indeholder både attributter og associationer til andre klasser. Denne model er lavet inden vi begyndte at skrive selve koden og vi har efterfølgende i vores kode tilføjet og ændret flere controllere, models og repositories, herunder bl.a. en CustomerController, SeasonRepository, Season class osv. Disse ville kunne ses i vores [Design Class Diagram](#_Design_Class_Diagram).

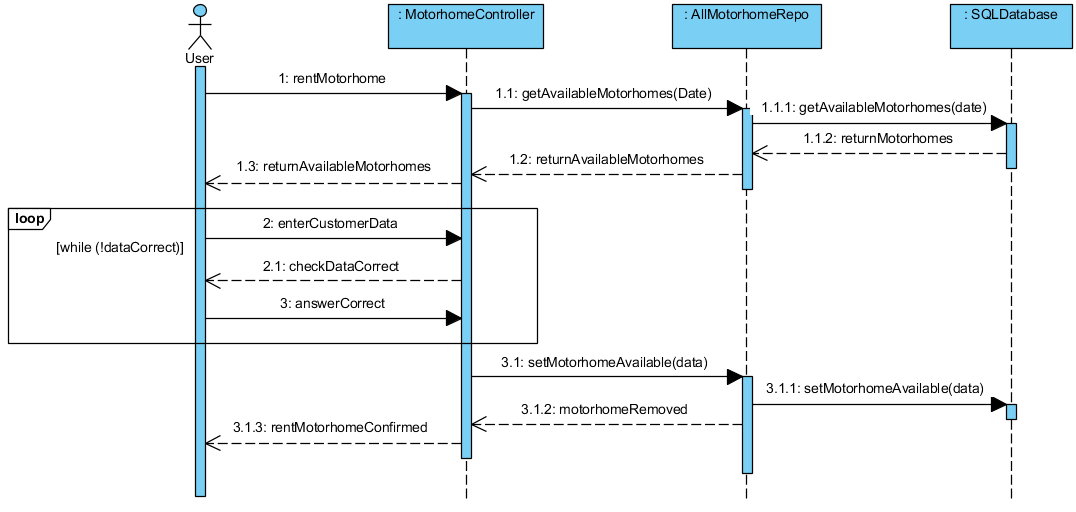


## System Sequence Diagram (Use Case: Rent Motorhome)

Vores System Sekvens Diagram er lavet over vores Use Case ”Rent Motorhome”. System Sekvens diagrammet viser en sekvens af et input og output mellem vores Actor og Systemet. Vores actor starter med at klikke på en knap så vores rentMotorhome metode aktiveres. Derefter kører et loop indtil alt data er udfyldt korrekt.

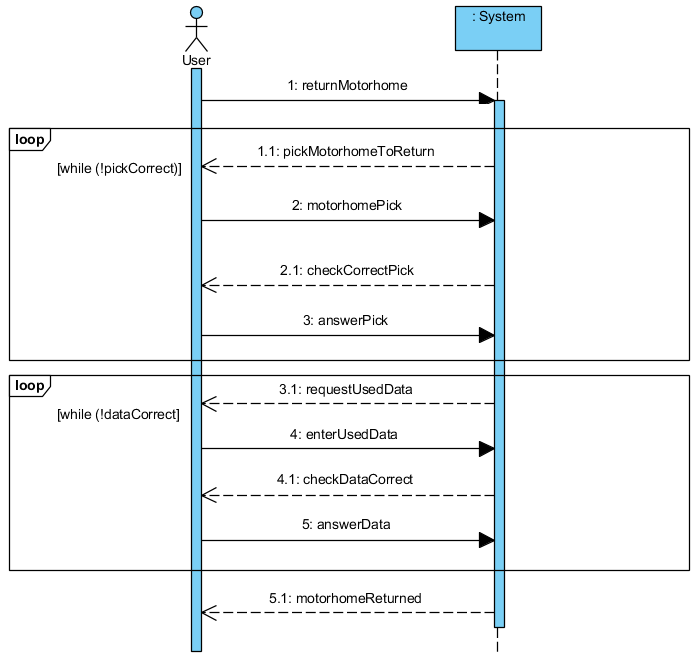
## Sequence Diagram (Use Case: Rent Motorhome)

Vores sekvens diagram har vi valgt at lave over Use Casen *”Rent Motorhome”*, sekvensdiagrammet viser os interaktionerne mellem vores Actor og de tre klasser / objekter. I vores sekvens diagram har vi en *MotorhomeController*, *AllMotorhomeRepo* og en *SQLDatabase*. Vores Actor (User: SalgsAssistent) ville i vores system vælge at leje et Motorhome for en kunde. *MotorhomeController* arbejder sammen med *AllMotorhomeRepo* for at få en liste over tilgængelige Motorhomes. *AllMotorhomeRepo* arbejder så sammen med vores *SQLDatabase*, som returner tilgængelige motorhomes. Salgsassistenten bliver herefter promptet for at indtaste lejerens personlige data korrekt, denne interaktion sker kun sammen med *MotorhomeControlleren* klassen og interfacet. Hvis dataene er korrekte så godkende salgsassistenten at ville leje det valgte Motorhome, og motorhomet bliver herefter sat ind i et nyt table i *SQLDatabasen* som viser at det gældende motorhome er aktiv med specifik kunde.



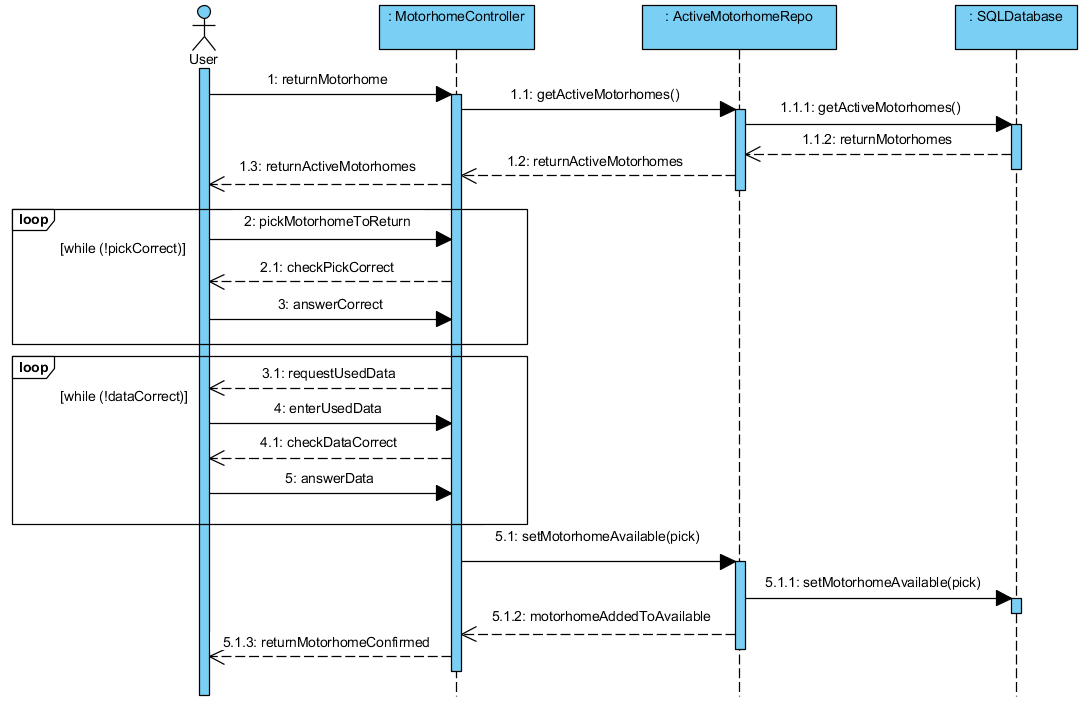
## System Sequence Diagram (Use Case: Return Motorhome)

Vores andet system sekvens diagram har vi valgt at lave over use casen *”Return Motorhome”*. Hvor vi har vores actor på venstre side og vores System på højre side. Vores actor (User: SalgsAssistent) salgsassistenten klikker i interfacet på knappen ”Return Motorhome” så interaktionen med vores system begynder, vores system sender en besked tilbage til vores user omkring hvilket Motorhome som skal returneres. Sådan kører interaktionerne mellem vores user og system hele vejen igennem. Vi har også gjort brug af et loop, de steder hvor vi mener at dette er nødvendigt. Loopet kører indtil at salgsassistenten bekræfter dataene.



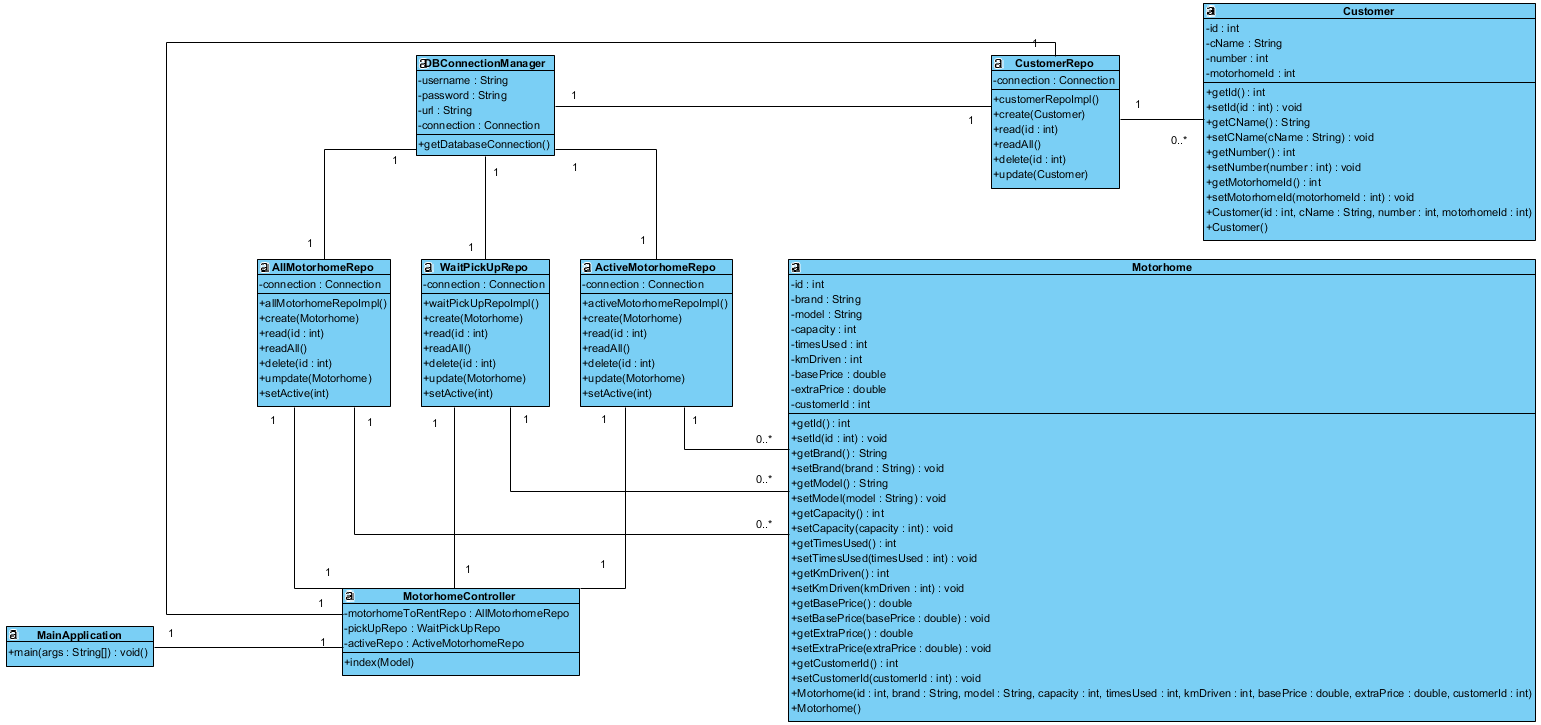
## Sequence Diagram (Use Case: Return Motorhome) (Jonathan)

Vores andet sekvens diagram har vi valgt at lave over use casen *”Return Motorhome”*. Hvor vi har vores actor (User: SalgsAssistent) på venstre side og vores MotorhomeController, ActiveMotorhomeRepo og vores SQL-database til sidst. Vores actor, salgsassistenten klikker i interfacet på knappen ”Return Motorhome” så interaktionen med vores system begynder, vores controller sender en besked til ActiveMotorhomeRepo om afhentning af alle aktive motorhomes som bliver videresendt som query til SQL-databasen, når den query er kørt, bliver der sendt en liste hele vejen tilbage til brugeren. Vores User går så igennem den liste og vælger det motorhome der skal returneres. Når man har valgt det rigtige motorhome, bedes brugeren indtaste de forskellige brugsdata for leje perioden, dette køres indtil brugeren har bekræftet at dataene er korrekt. Dataene sendes så fra controlleren til ActiveMotorhomeRepo som sender en query til SQL-databasen om at det gældende motorhome skal fjernes fra activeMotorhome tablet i databasen. Hvis setMotorhomeAvailable(pick) ikke fejler og returnerer true, sendes der en besked til controlleren som siger til user’en at motorhomet er blevet returneret.



## Design Class Diagram (før vi begyndte at kode) (Jonathan)

Nedenstående design klasse diagram repræsenterer vores system, som vi har tænkt os at lave til Nordic Motorhome. Diagrammet bruges til at give os et godt overblik og forståelse for hvordan vi vil opbygge vores program. Diagrammet indeholder vores controller og de basis modeller vi har tænkt os at lave i selve vores applikation. Man kan også se vores repositories som er en af de vigtigste dele af vores program. Hver klasse indeholder både attributter og metoder med hvad for nogle attributter der skal bruges. Udover dette er der også associationer til andre klasser. Mange af disse associationer er 1 til 1 hvilket indikerer at der kun skal eksistere en instans af det objeekt for den gældende association. Dog kan man også se at vores to modeller Motorhome og Customer har 1 til 0..\*, 0..\* betyder at der kan eksistere alt fra 0 instanser af det objekt til mange, dette bruger vi da de repositories de er forbundet til har både kan slet og opret metoder. Dette diagram er lavet inden vi begyndte at skrive selve koden og vi har efterfølgende i vores kode ændret i stort set alle klasserne, her i blandt controllere, models og repositories. Vi er også blevet nød til at tilføje mange nye klasser herunder bl.a. en CustomerController, SeasonRepository, Season class osv. Mange af disse ændringer sker som følge af GRASP responsibilities da der på daværende tidspunkt var et problem med bl.a. high coupling, vi kommer til at skrive mere om dette under [GRASP-diagrammerne](#_GRASP_Responsibilities).



## Design Class Diagram (efter vi var færdige med at kode) (Jonathan)

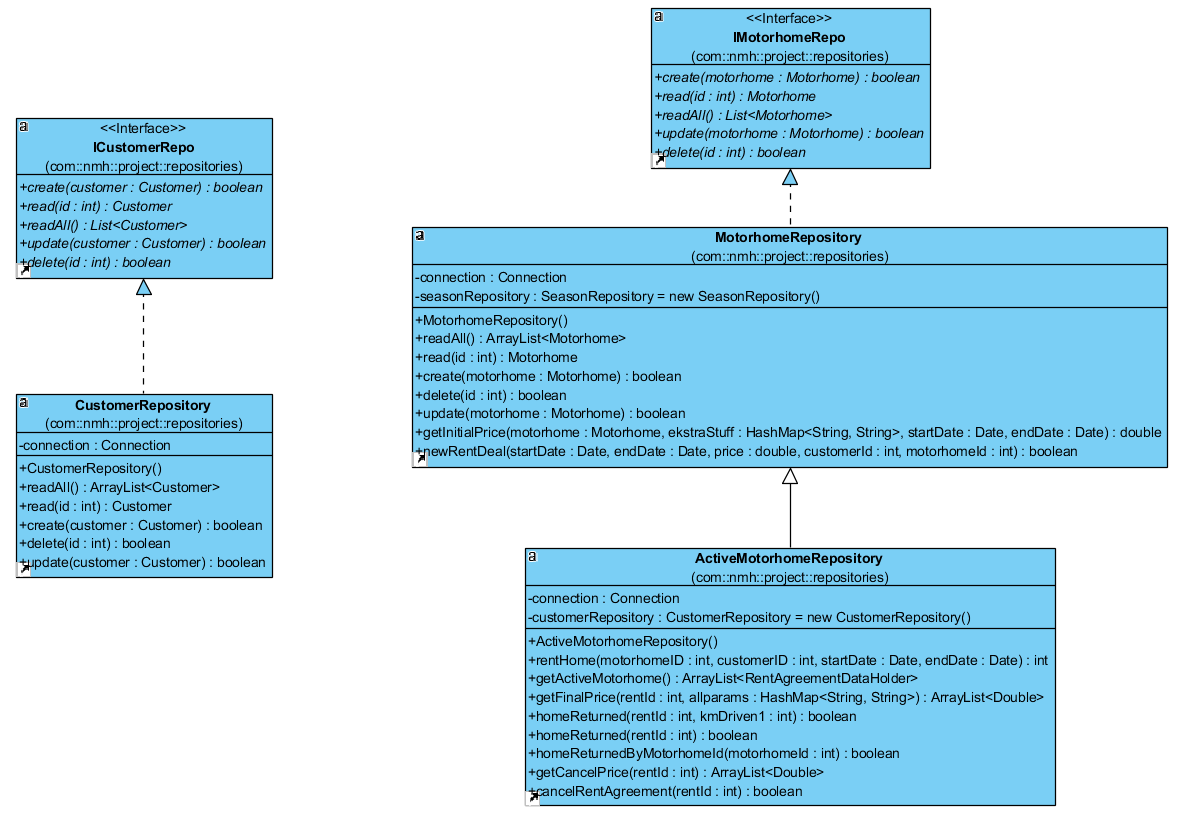
Grundet størrelsen på diagrammet er det vedlagt som bilag, under [bilag 2](#_Design_klasse_diagram), det kommer også til at følge med som billede med afleveringen da det kan være svært at se i Word.

Diagrammet kan bruges til at give os og andre der skal se kode-delen af vores et godt overblik og forståelse for hvordan programmet er opbygget. Diagrammet indeholder vores fem controllere, der i det originale design klasse diagram kun blev vist som et. Det viser også vores to nye modeller Season og RentAgreementDataHolder, disse bruges henholdsvis til at fastlægge en pris ud fra sæson periode og til at opbevare midlertidige data for en lejeperiode. Med de nye modeller er vi også blevet nød til at ændre og lave flere repositories, dette gjordes også for at holde fokus på GRASP. De nye repositories er DamageRepo, FilterRepo og SeasonRepo som bruges flere steder igennem vores pprogram. Sammenligner man igen de to klassediagrammer kan man også se at ActiveMotorhomeRepo har fået en række nye metoder, dette kommer af at vi ikke har arbejdet så meget med SQL databaser før og derfor har set anderledes på hvordan vi ville kreere objekter i lister og lignende. Man kan også se at ActiveMotorhomeRepo inheriter fra vores MotorhomeRepo, dette er illustreret med den fuldt optrukkede linje og en hvid pil, dette gør vi da den bl.a. bruger sin super klasses metode. Et andet punkt hvor vi også bruger polymorphismen er ved vores interfaces. Både MotorhomeRepo og CustomerRepo har et interface og dette er indikeret med en stiplet linje og en blå pil. Vi bruger interfaces for at sige hvad de forskellige klasser skal kunne. Udover dette har vi også tilføjet vores DatabaseConnectionManager som er den klasse der står for at forbinde til vores SQL-database, kigger man på diagrammet kan man se at alle dens attributter og metoder er understregede, dette indikerer at de er statiske. Vi bruger statiske metoder og attributter for at vi ikke behøver at oprette en instans af klassen for at bruge den, og for at sørge for at attributterne kun kan ændres ved brug af den gældende metode.

## GRASP Responsibilities

### GRASP diagram polymorphism (Jonathan):

Nedenstående diagram bruges til at vise de steder vi har brugt polymorphism i vores program. Vi bruger i vores kode både interface og inheritance. Vi bruger interfaces for at fastslå et basisgrundlag for hvad vores klasser skal kunne. Og vi bruger inheritance i vores ActiveMotorhomeRepo for at undgå at skulle genskrive kode unødvendigt og derved øge reuseability. Interfacet er vist med en stiplet linje og blå pil og inheritance bliver vist med den fuldt optrukkede linje og hvide pil.

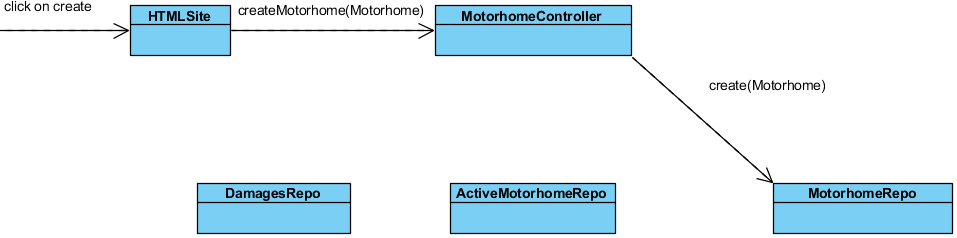


### GRASP diagram controller (Jacob):

I GRASP design bruges controller til at håndtere den data der kommer fra UI. Det er den første klasse data møder, som er ansvarlig for at håndtere og videresende information. Et helt konkret eksempel vil være: Hvis man trykker på login, hvor bliver password og login modtaget I en teoretisk applikationen?

I en Spring applikation er det rimelig let at identificere hvilke java klasser som er controllere, fordi Spring kræver at bruger en @Controller annotation ved Controller klasser. De ligger alle sammen I samme mappe og har formodentlig controller som en del af deres navn. Det gøres så spring ved, at her er en klasse som modtager html requests. Springs UI er naturligvis alle html filerne med tilhørende css filer, og det er her controllererne får deres information fra. Informationen kan være <form> submit data eller et link til en ny html adresse.

I vores applikation har vi 5 controller klasser, hvor 1 af dem kun håndterer /error, nemlig HomeController. De andre controller klasser er CustomerController, MotorhomeController, RentMotorhomeController og ActiveMotorhomeController. De fleste af controllerklassernes funktion giver sig selv, CustomerController håndterer alle html sider hvor Customer er I fokus, fx at oprette en ny kunde I systemet. MotorhomeController håndterer siderne hvor modellen Motorhome er I fokus, fx at hvis NMH har købt et nyt motorhome. RentMotorHome beskæftiger sig med at oprette nye leje aftaler, og modtager altså information såsom dato fra starten af lejeaftalen. ActiveMotorhomeController er måske den, hvor navnet giver mindst mening, hvis man ikke ved hvad der menes med active. I den her applikation skal active forstås som leje aftaler, som er oprettet og ligger I systemet. Derfor håndterer ActiveMotorhomeController ting som fx at aflyse en allerede eksisterende aftale eller afslutte en aftale når kunden returnerer et motorhome.   
De modtager alle sammen information fra html siderne, som de sender videre til de relevante repositories. På nedenstående figur kan man se et diagram over hvordan controller modtager information og sender den videre.

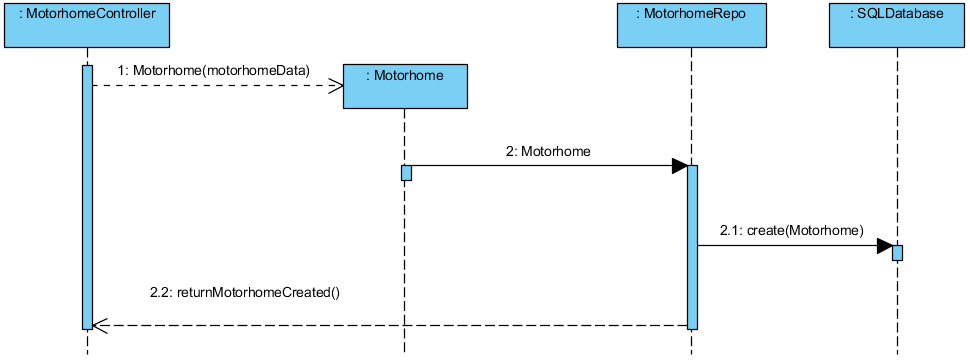


### GRASP diagram create motorhome:

Creator GRASP design princippet beskæftiger sig med, hvem opretter hver enkelt klasse. Hvem har ansvaret for at skrive new. Det er naturligvis vigtigt, at det er ikke er tilfældigt hvem der gør det, både for læsbarheden af koden, men også så man har en klasse der er information expert, så den klasse der opretter et objekt også har den information den skal bruge for at oprette klassen.

I vores application kan dette bedst ses ved brug af Controller klasserne, som har ansvaret for at instantiere de respektive repositories. Det giver bedst mening, at netop disse klasser instantiere repositories, fordi det er her klasserne skal bruges.   
Det bliver også brugt når de forskellige klasser, både controllerer og repositories, opretter modeller som motorhome, customer og lignende. De bliver oprettet I klassen selv, da det er her de skal bruges og det er her de har informationen. Det er altså ikke en handler klasse eller lignende, som står for at lave nye Motorhome modeller.

Hvis man skal se på både creator og controller sammen, vil det være controller klassen som får data, og her bruger den informationen til enten at oprette modeller som den selv bruger, eller sender den relevante information videre til et repository som vil overtage ansvaret.



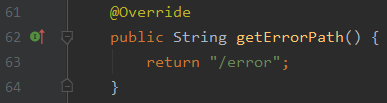
# **Software Construction**

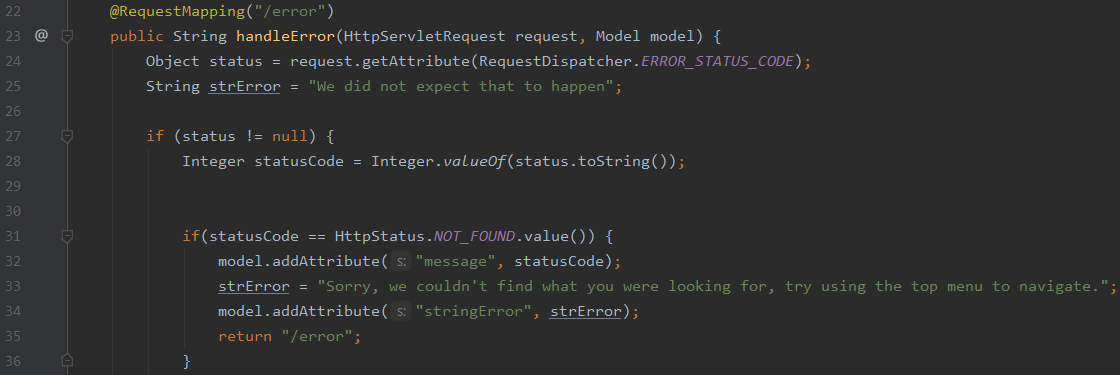
## Java, Controllere, HTML og CSS (kommenteret udklip)

### http error håndtering (Jonathan):

Det er dvs. meget usandsynligt at vores program ikke fejler så derfor bliver vi nød til at tage højde for de forskellige fejl der kunne komme. Når det kommer til http, kan der opstå mange fejl på forskellige tidspunkter, vi håndterer i vores kode 4 generelle fejl som vi har stødt ind i undervejs i programudviklingen. Udover de fire har vi også lavet en generel fejl håndtering, så hvis den fejl der opstår, ikke er fejl 404, 500, 403 eller 400 så vises der bare en generel besked.

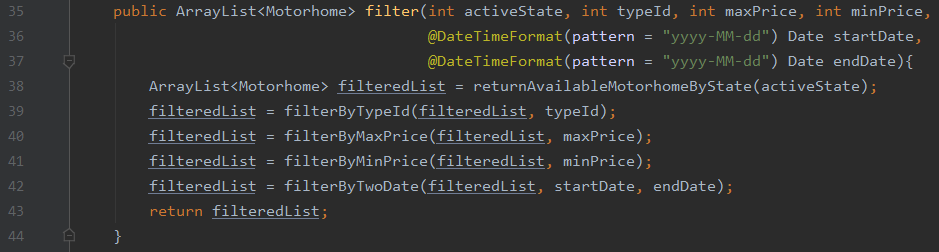
Den klasse vi håndterer fejlene i har vi kaldt HomeController og den implementerer et spring interface der hedder ErrorController. For at vi kan lave en custom error side bliver vi nød til at returnere den html side vi ønsker at vise når der opstår en fejl. Det er det vi gør fra linje 61 til 64. Vi bruger så RequestMapping og laver en metode der håndterer hvad der sendes når vi får en fejl. På linje 24 ses at vi bruger et af metodens parametre til at finde statuskoden på den fejl vi har modtaget, denne bruges så igen på linje 28 og laves om til en integer så vi kan lave logik på den. Udover at fortælle brugen hvilken fejl kode der er blevet produceret vil vi også fortælle lidt om den, til dette opretter vi først en String på linje 25 som er vores generelle fejlkode, denne ses kun hvis den fejl der opstår, ikke er en af de 4 fejl vi har valgt at håndtere. Er det dog en af de fejl der opstår så laves der en custom besked til den fejl, dette kan man se på linje 33. Så hvis der opstår en fejl aktiveres vores handleError() metode, og så længe status koden ikke er null går programmet videre. Hvis fejl koden f.eks. er 404 er if-statementet på linje 31 sandt og derfor bruger vi vores model til at sende både statuskoden og vores custom besked til error siden.





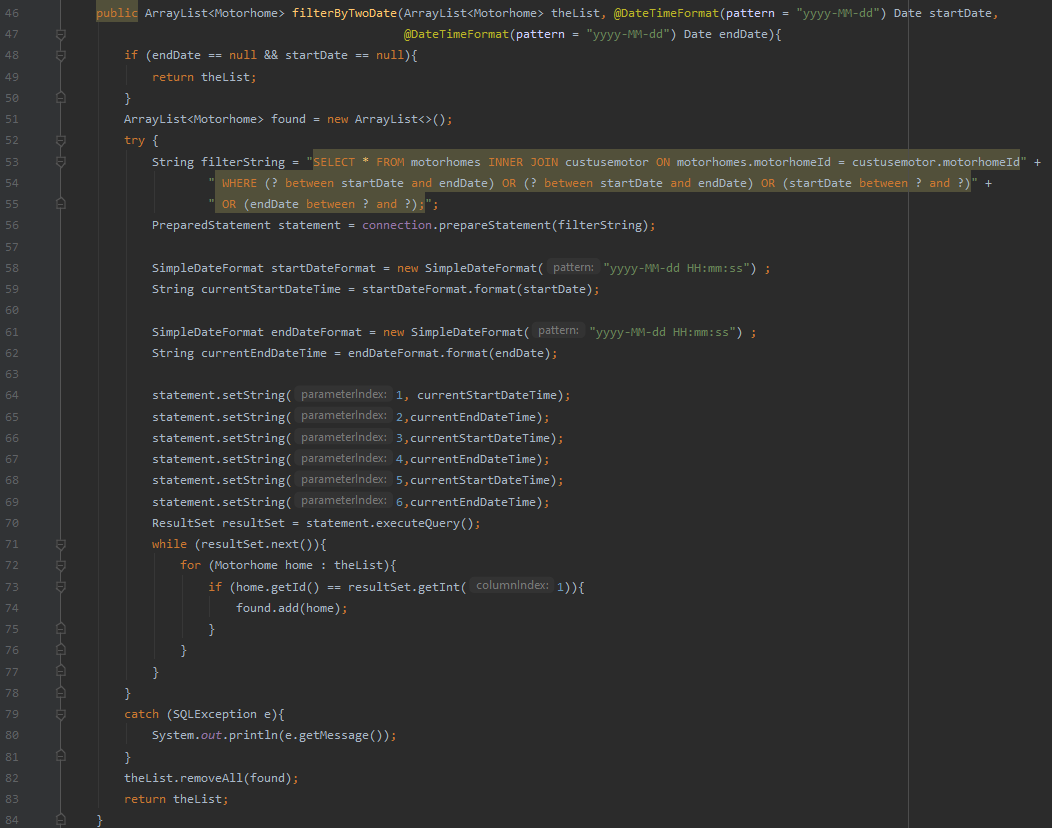
### Filter ift. forskellige parametre:

Grundlaget for vores produkt gør det næsten umuligt ikke på et eller andet tidspunkt at filtrere ud fra hvad det er kunden eller andre skulle have lyst til at se. På grund af dette har vi selvfølgelige lavet en filter metode. Vores filter metode bruges når der skal udlejes et motorhome og ved hjælp af de parametre man indtaster, sender den så en liste tilbage. På nuværende tidspunkt kalder den bare en masse andre metoder, som modtager en liste og fjerner motorhomes fra den liste ud fra de gældende parametre. Lad os kigge på en af de metoder.



(billede af koden på næste side)

Metoden filterByTwoDate() modtager to datoer og en ArrayList. Til at starte med checker den så at det er to gyldige datoer, er det ikke det returnerer den bare listen uden at fjerne nogle motorhomes fra den. Er de derimod gyldige oprettes der en ny tom liste kaldet found, denne bruges til at holde styr på hvilke motorhomes der skal fjernes fra listen senere i metoden. Da vi arbejder med SQL bliver vi også nød til at være forberedte på mulige fejl der kan opstå, dette gør vi med vores try-catch. Vi forbereder så en String vi kan indsætte i vores statement på linje 53, denne String skal bruges til at finde data i vores SQL fra vores custUseMotor table som passer med vores parametre. Vores filterString indsættes i vores PreparedStatement på linje 56 så vi senere kan eksekvere den med vores SQL-database. Fra linje 58 til og med 62 konverterer vi vores datoer til Strings så de kan forstås af SQL, dette kan gøres på andre måder, men vi mente dette var mest readable i tilfældet. Disse Strings indsættes så i vores PreparedStatement og vi eksekverer vores query. Vores data bliver indsat i et ResultSet, vi bruger så vores ResultSet til at gennemgå dataene og checker om nogle af de motorhomes der er i vores ArrayList også er i ResultSettet. Hvis de findes, tilføjes de til found listen, som så bruges til at fjerne dem fra den originale liste. Når de er fjernet, returneres listen og så er metoden slut.



# Bilag

## Risk Analyse (bilag 1)

## Design klasse diagram (efter færdig kode) (bilag 2)

