

Nordic Motorhome (Rapport)

Anno 4/06/2020, dat19B



Nordic Motorhome Project

Udarbejdet af:

Dejan Alija	22/07-1998
Heritier Mugisha	14/02-1998
Mohamad Elhady	22/01-1998
Oliver Schönnemann Bramsen	29/10-1997

KEA

Interdisciplinary Exam Project

Nordic Motorhome Rental

Sem 2 2020©

Indledning	3
Problemformulering	3
Afgrænsning af projekt	3
Værktøjer	4
Glossary	5
ITO	6
Feasibility Study	6
Virksomhedens vision og koncept	6
Løsning	6
Undersøgelse	6
Konklusion	8
Risk Analysis	9
Risikomomenter	9
Risikoplan	9
Udvidet Risiko tabel	10
SWOT Analysis	11
Stakeholder Analysis	13
Konklusion	15
Software Design	16
Faseplan	16
Supplementary Specification	17
Use Case Diagram	18
Fully Dressed	19
Opret motorhome	19
Opret kunde	21
Opret udlejning	23
Domain Model	24
SD og SSD diagrammer	25
Opret kunde	25
Opret udlejning	27
Klassediagram	28
GRASP	30
Controller	31
Creator	31
Low coupling	32
Softwarekonstruktion	33
E/R diagram	33

SQL beskrivelse	34
Data Definition Language	34
Data Manipulation Language	35
Normalform	37
Kode og klasse beskrivelse	39
HTML og CSS	46
Fejlbeskrivelse & Forbedringer	48
Design forbedringer	50
Konklusion	52
Bibliografi	53
Henvendelser (links)	54

Indledning

(DA)

Nordic Motorhome Rental (NMR) er en udlejningsvirksomhed der blev grundlagt i 2019 lidt udenfor København. Der er indtil videre 8 ansatte udover de to ejere, og der er planer om at ansætte flere, grundet deres store vækst og succes. Derfor er vi, som KEA studerende, ansat til at udvikle et system til NMR. Vores hovedmål er, at evaluere en prototype og det sker ved fokus på at udvikle en hjemmeside vha. Spring Boot, og en lokal database-forbindelse til MySQL. Inden da, er der tegnet analyser, diagrammer og gennemførlighedstests, der viser om dette er muligt og gennemførligt for både virksomheden og for os.

Problemformulering

(OSB)

I de kommende år, vil NMR muligvis ikke være det eneste udlejningsfirma på markedet i Danmark. NMR er en lille samling af 8 ansatte fordelt ud med hver deres stilling. Alt administrativ arbejde, såsom informationer, udlejninger og papirarbejde, kræver strukturering og sikkerhed på baggrund af GDPR – altså kundens personlige data. Er sikkerheden hos NMR sårbar nok til at investere i et system, og er det økonomisk gennemførligt for dem?

Afgrænsning af projekt

(DA)

Ved afgrænsning af projektet har vi først og fremmest beskrevet de forretningsmæssige aspekter i projektets sammenhæng. Dette gjorde konceptet nemmere at forstå og derved skabe en sammensluttende betydning for både os og virksomheden. Dette skete vha. ITO analyser og UML diagrammer i en Unified Process.

Værktøjer

(OSB)

Inden for softwareudvikling er der mange velfungerende og nødvendige værktøjer. Vores gruppe har valgt at bruge nedenstående værktøjer til at fuldføre vores eksamensprojekt. Der vil blive forklaret kort om hvad diverse gør og hvad man bruger dem til.

IntelliJ

Under vores projekt gør vi os brug af IntelliJ IDEA Ultimate Edition. Det er alle i gruppen komfortable med at bruge, samt er det godt og nemt da vi gør os brug af Spring Boot, som er let tilgængeligt i IntelliJ og er let at sætte et projekt op med Spring i.

GitHub

Da vi er sammenlagt fire gruppemedlemmer, har vi valgt at gøre os brug af Git. Det er et udmærket stykke værktøj at gøre sig brug af. Koden vil altid være "up to date" samt alle kan skrive inde i projektet på samme tid. Når vi i gruppen koder med Git, kan der nemmere kombinere teamets kode, hvis der er blevet arbejdet med forskellige ting. Der er struktur over projektets arbejde, orden og styr på tingene.

Discord - Teams

I gruppen har vi valgt at bruge Discord og Teams. Begge platforme er rigtige gode kommunikationsredskaber. De findes både som applikation og browser. Discord blev primært brugt til hurtig opsummering, da det er nemmere at have med at gøre fordi, vi har oprettet vores egen kanal til privat brug af eksamensprojektet. Teams store feature som discord ikke anvender er "Anmodning om styring". Det blev flittigt brugt hvis et medlem havde problemer med diverse.

Google Drev

Google drev var et perfekt værktøj til lagring af dokumenter. Google drev er et cloud-baseret-lager og fildeling. Vi kunne gemme diagrammer, filer, billeder og links. Det ligger alt sammen i en cloud, så de er tilgængelige overalt. Alt dette gjorde gruppen brug af,

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

ved at organisere filer i mapper. En mappe til hver ting fra ITO, Software Design og Software Konstruktion. Det fandt vi meget nyttigt grundet, at det hele skal indsættes i en endelig rapport. Dertil kan man hoppe ind på hvad man skal bruge og indsætte. Google drev i sig selv var også nødvendig til den skriftlige del af eksamensprojektet. Vi gjorde os brug af Google Drevs skriveprogram, Google Docs. Reelt set Word. Det blev brugt til at alle i gruppen igen kunne få lov til at skrive på samme tid. Alt sammen en god feature til at optimere ens tid.

MySQL

Under andet semester blev vi introduceret til MySQL. MySQL er et open source SQL relationsdatabase styringssystem, som alle i gruppen er meget positive med, har en god forståelse for og arbejder godt omkring. Vi har brugt MySQL til lagring af data og testet om nye funktioner i vores hjemmeside går i gennem korrekt og ned i databasen.

Glossary

NMR	Nordic Motorhome Rental	Her er nogle forkortelser vi bruger henover rapporten.
GDPR	General Data Protection Regulation	
SD & SSD	Sekvensdiagram & systemsekvensdiagram	
DB	Database	
DA	Dejan Alija	
OSB	Oliver Schöнемann Bramsen	
ME	Mohamad Elhady	
HM	Heritier Mugisha	

ITO

Feasibility Study

(ME, DA)

Virksomhedens vision og koncept

Nordic Motorhome Rental (NMR) er under vækst, og ønsker dermed at udvikle et program der kan behandle alle informationer vedrørende motorhome, udlejning, medarbejdere og kunder. De har behov for at ansætte flere medarbejdere, som kan håndtere kunderne. Derudover ønsker virksomheden i den nære fremtid at blive bemærket online vha. en hjemmeside.

Løsning

Den første løsning er den nuværende løsning. Ejerne benytter sig af Office-pakken og papirarbejde. Det er helt klart den billigste måde, men hvad angår sikkerhed og optimering, er dette ikke den bedste løsning.

Et alternativ ville derfor være et system som er tilpasset dem kun. Den store fordel ved et personligt program er, at ejerne får præcist det, de ønsker for at 1) gøre administrativt arbejde nemmere og 2) større sikkerhed ift. opbevaring af data. I denne situation ser man, at systemet kører offline, og at data dermed bliver gemt lokalt. Dette mindsker overbelastningen samt evt. hacking. En anden pointe er, at man senere hen kan tilføje og ombygge programmet nemmere.

Undersøgelse

I undersøgelsen tager vi udgangspunktet i om projektet er:

1. Teknisk feasible

Det tekniske aspekt af projektet har vi, som udviklere, frit adgang til alt form for software og hardware, og dermed ikke en ekstra omkostning for virksomheden. Software der bruges er MySQL-database og IntelliJ editor, og begge software er velkendte og testet. Systemet kører offline og data er månedligt sikkerhedskopieret, samt bliver gemt lokalt på en server.

2. Økonomisk feasible

Da vi er en gruppe studerende, er det en langt mindre økonomisk udfordring for virksomheden at ansætte os. Alt vores software samt licenser er velkendte og står frit til adgang på baggrund af vores studie på KEA. Systemet gemmes i en database på MySQL og selve systemet bliver skrevet på Java i IntelliJ – altså to store og veltestet programmer. Dog skal der tilkøbes en server.

3. Operational feasible

Vi bestræber os efter at skrive et brugervenligt system, der kræver minimal oplæring herved. Da vi kun er studerende, giver vi gerne NMR en rundvisning og sørger også for 10-timers gratis teknisk support.

4. Retlig og juridisk feasible

Som bruger af systemet, skal man have et login; dette øger sikkerheden. Derudover kører systemet offline og mindsker chancen for evt. hacking af data. At systemet er offline betyder det, at alt data bliver gemt lokalt.

Konklusion

NMR er under vækst og har planer om at skaffe et system, derfor ses det gerne, at systemet også er overkommeligt. Deres nuværende situation er, at man bruger Office-pakken – altså et færdiglavet program. Dette bedømmer vi ikke til at være holdbart og sikkert på længere sigt. Her konkluderer vi, at det er langt mere sikkert at få et system, der er designet til ejernes behov. Dette betyder også at, systemet ikke bliver overbelastet på nogle tidspunkter, samtidig har kunden mulighed for at tilføje nye features i fremtiden hvis det ønskes.

Det er derudover både teknisk, økonomisk og operationelt gennemførligt, da vi er en gruppe studerende på KEA, og dermed har frit adgang til velkendte og testede programmer samt licenser. I systemet herunder, bliver data gemt offline og lokalt. Dette resulterer i formindsket risiko for evt. hacking.

Vi kan således konkludere, at vores feasibility-undersøgelse afsluttes med et klart “go”.

Risk Analysis

(ME, DA)

Risikomomenter

- Sygdom/Barsel - NMR er en lille virksomhed som kun har 8 ansatte. og derfor er der ikke så mange til at tage over i tilfældet af barsel eller sygdomme, ligesom covid-19 eksempelvis.
- Vedligeholdelse af motorhome- for at sikre at deres kvalitet er i top.
- Systemnedbrud - Hvis systemet går ned, kan NMR ikke håndterer deres digitale motorhome servicebog, udlejningen og kunderne.

Risikoplan

- Sygdom og barsel
 1. Forebyggende:

Da det er i deres plan, at ansætte flere, er dette et forebyggende element. Et alternativ kunne også være, at have tæt kontakt med et vikarbureau.
 2. Skade-reduktion:

At man vælger at ansætte 3 medarbejder på fuldtidskontrakter og 2 medarbejdere på deltidskontrakter.
 3. Krisehåndtering

Ved sygdom ringes der til et vikarbureau og ved barsel ansættes en medarbejder på en midlertidig kontrakt for at fylde personens arbejdstider.
- Vedligeholdelse af motorhome
 1. Forebyggende:

Sørge for at vedligeholde motorhomes, så de står klar til brug. Dette omfatter rengøring og serviceeftersyn. Et alternativ kunne være, at leje motorhome under peak-season, så de ikke er står under mangel.
 2. Skade-reduktion:

Sørge for at, motorhome bliver klargjort ordentligt, så den er klar til brug.
 3. Krisehåndtering:

Sørge for have kontakt til mekanikeren hvis der er funktionsfejl.

- System nedbrud
 1. Forebyggende:
NMR's data bliver sikkerhedskopieret med jævne mellemrum.
 2. Skade-reduktion:
Data gendannes.
 3. Krisehåndtering:
Ved systemnedbrud, tilkaldes en tekniker hurtigst muligt, som kan anskaffe den seneste sikkerhedskopi.

Udvidet risiko tabel

(ME, DA, OSB, HM)

Risikom oment	Sandsyn lighed	Konse kvens	Produkt	Præventive tiltag	Ansvarlig	Løsningsforslag	Ansvarlig
Sygdom/ barsel	2	7	14	Sørge for at have kontakt med et vikarbureau der kan fylde arbejdsroller i tilfælde af sygdom/barsel. Ansætte flere medarbejdere	Ejere	Der findes en afløser hurtigst muligt evt. andet personale.	Ejere
Vedligeh oldelse af MH	1	7	7	Sørge for at vedligeholde motorhomes.	Ejere	Sørger for have kontakt til mekanikeren hvis der opstår funktionsfejl	Ejere
System nedbrud	1	3	3	Der laves backup af NMR data med jævne mellemrum.	Ejere	Ved systemnedbrud, tilkaldes en tekniker hurtigst muligt, som kan anskaffe den seneste sikkerhedskopi.	Udviklere

SWOT Analysis

(OSB, HM)

Interne situation	
Stærke sider (strengths)	Svage sider (weaknesses)
<ul style="list-style-type: none">- Placering (København)- Godt omdømme (populære)- Medarbejder forøgelse	<ul style="list-style-type: none">- Mangel på ansatte i fremtiden- Risiko ved for få MH- Syge medarbejdere ift. Covid-19- Administrative papirarbejde
Eksterne situation	
Muligheder (opportunities)	Trusler (threats)
<ul style="list-style-type: none">- Flere indlandsrejser grundet Covid-19	<ul style="list-style-type: none">- Covid-19 (færre turister, syge medarbejder)- Konkurrenter

Stærke sider

De er placeret tæt på København, hvilket gør deres levering af motorhomes nem for dem og kunderne. De leverer dem til lufthavnen (hvor turister har nem og hurtig adgang til MH), hoteller (jf. førnævnte) etc. mod en betaling på €0,70 pr. km.

Det vil gøre virksomheden populær og eftertragtet. Covid-19 medføre at flere potentielle kunder, der normalt vil rejse ud, vil ende med at blive potentielle faste kunder fremadrettet. De har planer om at ansatte flere medarbejder, hvilket kan blive en realitet til sommer, da sandsynligheden for at efterspørgslen vil stige, er relativ høj. Ydermere er medarbejder forøgelsen især vigtig nu med henblik på Covid-19, i tilfælde af sygdom i forbindelse med virussen.

Svage sider

Hvis der kommer en større efterspørgsel på Motorhomes (MH) end førhen (Grundet Covid-19), er antallet af ansatte (8 ansatte) et problem. Større efterspørgsel vil selvfølgelig føre til mere arbejde og mere arbejde kræver flere hænder.

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

Ydermere er det problematisk hvis medarbejder(e) bliver smittet med corona. I det de kun er 8 ansatte vil en medarbejders fravær medføre et tab på 12,5% af arbejdsstyrken. Særligt problematisk er det, hvis deres eneste mekaniker, der er ansat i virksomheden bliver syg. I den forbindelse ville de skulle bruge en ekstern mekaniker, som med al sandsynlighed ville være dyre. Der kunne også i den forbindelse blive skabt et logistisk problem i med, at den eksterne mekaniker ikke ville foretage arbejdet på Nordic Motorhomes Rental værksted.

En anden væsentlig svaghed, er antallet af Motorhomes, som de er i besiddelse af - nemlig 32. Hvis efterspørgslen stiger, ville de ikke have kapacitet i form af motorhomes til at dække efterspørgslen. Dermed vil de tabe potentielle kunder, som med stor sandsynlighed vil søge hen mod konkurrenterne.

Muligheder

Covid-19 medfører færre udlandsrejser/ferier for danskere, hvilket kan være en fordel for dem, da danskerne i forhåbentlig vil foretage indlandsferie. Idet befolkningen har været isoleret i så mange måneder, trænger de til at komme ud, hvilket MH kan drage fordel af med den rette marketing. Endvidere grundet Covid-19 er olieprisen faldet på markedet, hvorfor omkostningerne for Nordic Motorhomes på benzin/diesel automatisk vil falde og i det lange løb hive nogle penge hjem.

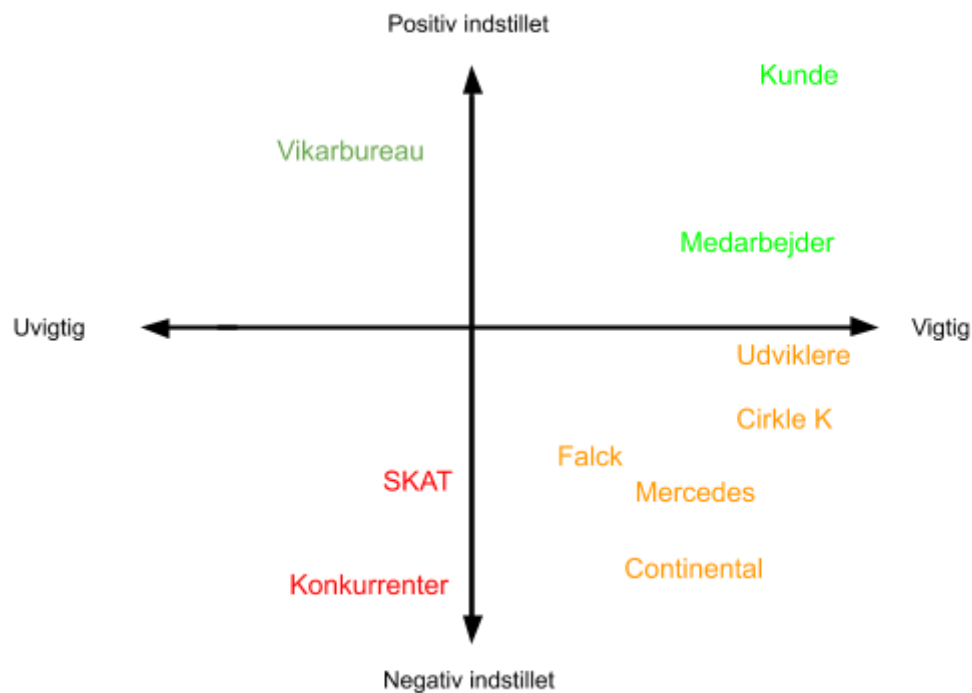
Trusler

Covid-19 i form færre turister, syge medarbejdere. Dertil er der 2 konkurrenter i Storkøbenhavn, som vil være en trussel for Nordic Motorhomes evt. færre kunder til sommeren 2020.

Lavere priser i lavsæson for at sikre en bedre udnyttelse af deres kapacitet (i form af deres produkt). Høje priser i peak season, så de kan få største udbytte.

Stakeholder analysis

(OSB, HM, ME, DA)



	Indflydelse	Succeskriterier
Falck	De hjælper NMR kunder, hvis der opstår pludselige funktionelle fejl eller uheld med motorhomes.	Der sker funktionelle eller tekniske fejl med motorhomes, som NMR ikke kan løse.
Continental dæk forhandler	NMR indkøber dæk til fast lav pris, hos en autoriseret dækforhandler i tilfælde af punktering, slitage, sommer/vinterdæk, etc.	NMR vil dertil forblive faste kunder hos den autoriseret dækforhandler. Det vil for fremtiden skabe gode tilbud i form af stort indkøb samt, en forsikrede leverandør og organisation.
Circle K	NMR får en særaftale med Circle K således, at der ligger et Circle K kort i bilerne som kunderne kan anvende til tankning.	NMR har en særaftale med Circle K, hvor de får en procentvis rabat på bla. brændstof, bilvask, etc. Deres interesse ligger i at, NMR ikke vælger at få en anden aftale et andet sted.
Mercedes forhandler	NMR indkøber eller lejer Motorhomes til en fast lav pris hos Mercedes.	NMR forbliver en loyal kunde hos Mercedes, hvorpå at alle indkøb eller leje af Motorhomes foretages hos Mercedes.

Kunden	NMR har behov for kunder for at drive forretningen og tjene penge.	kunden er interesseret i at leje gennem NMR fordi de leverer god kvalitet til god pris.
Kunden	NMR har kamp om kunderne, hvilket medfører en konstant opmærksomhed på konkurrenternes træk på markedet.	Konkurrenterne har til formål at slå os på markedet. Få større andel af markedet samt kunderne.
Udviklere	NMR har brug for en aftale med udviklere i tilfælde af, de vil opdatere deres IT-systemer eller tilføje flere funktioner til programmerne	De bibeholder deres aftaler og sørger for tilfredse kunder for at tjene deres penge ind.
Skattemyndighederne	Der skal beskattes både medarbejderløn, indkøb i form af moms, firmaskat etc.	Så længe vores forretning kører, får skattemyndighederne inddrevet de nødvendige skatter.
Medarbejdere	De sørger for at holde forretning kørende. Uden medarbejder, er der ingen virksomhed.	Medarbejderne har et arbejde samt en løn, så de kan leve.
Vikarbureau	I form af underbemanding, er et vikarbureau en stor hjælp for NMR. Især inden for rengøring. Det er vigtigt at have nogle der kan rengøre motorhomes hurtigt og effektivt.	NMR som en virksomhed kan både skabe og give godt omdømme med et sådant et samarbejde, samtidig med at de holder en standard indenfor rengøring eksempelvis.

Delkonklusion

(ME, HM)

Ud fra vores undersøgelse om feasibility studies er vi kommet frem til at projektet er gennemførligt og afsluttes med et "go". I forhold til vores risikoanalyse kan vi konkludere, at der for få risici til ikke at gennemføre projektet. Dog skal der gøres opmærksom på, at i og med virksomheden har få ansatte anbefales der, at de følger deres plan om ansættelse af flere medarbejdere. Vores SWOT analyse vurderes, at NMR har stærke egenskaber i form af eksempelvis placering og godt omdømme. Svage sider i form af eksempelvis mangel på ansatte i fremtiden, administrativt papirarbejde og risiko for få motorhomes. Vores interessentanalyse peger på forskellige aktører der har interesser i virksomheden.

Software Design

Faseplan

(DA)

Undervejs i dette projekt har vi arbejdet i unified process (UP). UP er slags iterativ process – det betyder at vi arbejder i en gentagende udvikling. Selve udviklingen er organiseret i en serie af korte mini-projekter – det er dette der menes med iteration. De centrale ideer herunder er: trinvis udvikling, use case-drivende, tackling af høj risici og højeværdige issues tidligt i iterationen og tidlig samt ofte tests. Ved brug af UP reducerer vi risici, da vi allerede identificerer risikoen tidligt.

Iteration	Periode	Disciplin	Artefakter	Anmeldelse (Efter iteration)
I1	Startede 11/05 Sluttede 18/05	Analyse	Feasibility Study	100%
			Fully dressed	100%
		Business Modelling	Use case diagram	100%
			Domain Model	100%
E1	Startede 19/05 Sluttede 21/05	Krav	Supplementary Specification	100%
		Design	Class Diagram	80%
E2	Startede 01/06 Sluttede 02/06	Design	Class Diagram	20%

Supplementary specification

(ME, DA, OSB, HM)

Functionality	<p>Programmet skal kunne:</p> <ul style="list-style-type: none">- Håndtere kunder, hvor det bliver muligt, at oprette, opdatere og slette kunder.- Håndtere udlejning, hvor det bliver muligt, at oprette, opdatere og slette motorhomes.- Håndtere motorhome, hvor det bliver muligt, at oprette, opdatere og slette motorhomes.- Vise oversigt over kunder, udlejning og motorhomes.- Hente den sidst opdateret sikkerhedskopi fra den lokale server
Usability	<ul style="list-style-type: none">- Brugervenlig med et enkelt og nemt design- <u>Brugermanual hvori, der kan læses om de forskellige fejlmeddelelse, der skulle opstår samt en vejledning til, hvordan det løses.</u>
Reliability	<ul style="list-style-type: none">- Oppetid på 98% af tiden- Backup server
Performance	<ul style="list-style-type: none">- Systemets svartid må maksimalt være på 15 sekunder, men vi forventer, at det holder sig på gennemsnitligt 5 sekunder.
Supportability	
++	<ul style="list-style-type: none">- Skrives i Java & SQL

Use Case Diagram

(OSB, DA)

Vi vælger her, at både medarbejder og ejere kan logge ind i samme system, men har anderledes funktioner grundet af GDPR. Det har vi gjort for, at medarbejder ikke kan oprette, opdatere eller slette en anden medarbejder. Derfor har medarbejderen 3 menuer, hvor ejeren har de samme tre, plus den fjerde, hvor man kan oprette en medarbejder.

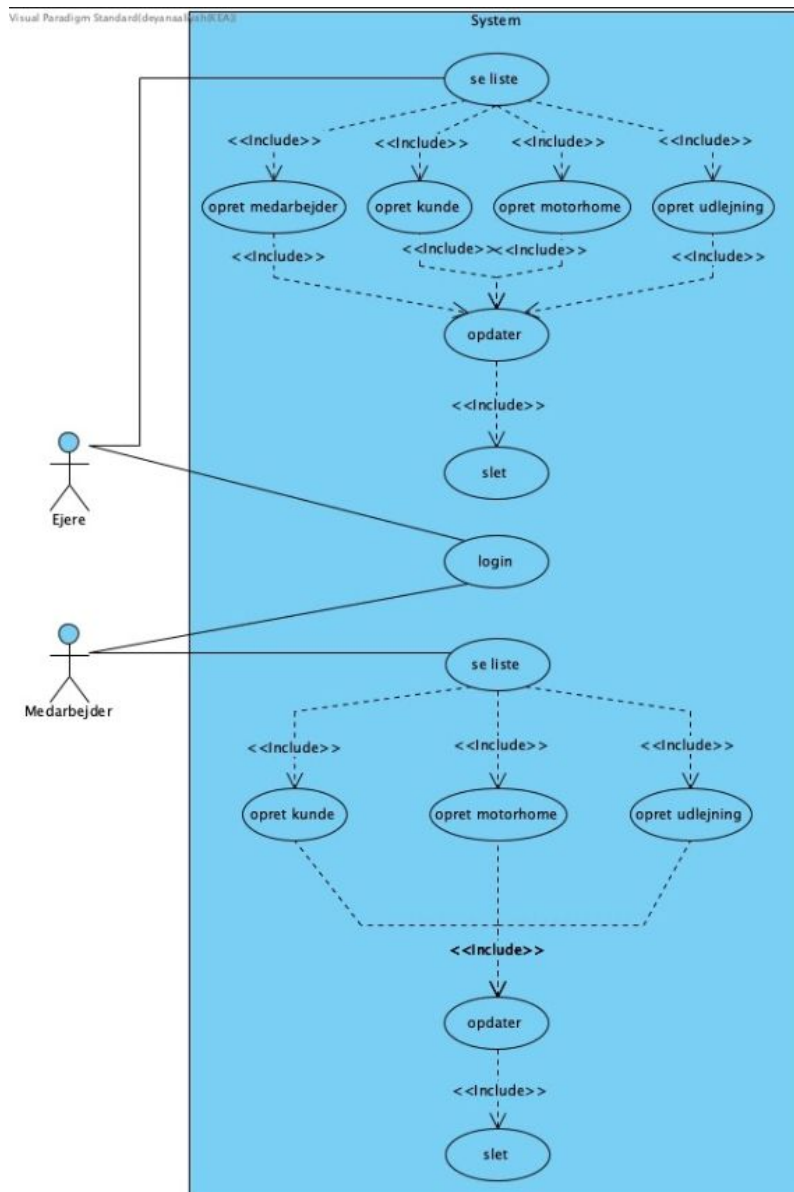


Diagram 1: Use case diagram

Således udvalgte vi tre use cases hvoraf vi skrev en fully dressed til dem.

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

Fully Dressed

(ME, DA, OSB, HM)

Opret motorhome

Use Case Name	Opret motorhome
Scope	Administrativt it-system
Level	User goal
Primary Actor	<ul style="list-style-type: none">- Ejere- Medarbejdere
Stakeholders and Interest	<ul style="list-style-type: none">• Ejerne har interesse i at opdatere flåden og få indkøbt nye motorhomes og/eller ekstra.• Forhandlere har interesse i at sælge og bibeholde deres partner.• Kunder har interesse i at leje den bedste kvalitet til den billigste pris.• Skat har interesse i at inddrive skatter
Preconditions	At brugeren er logget ind, og står i 'Håndter motorhomes/opret køb'-menuen.
Success Guarantee	<ul style="list-style-type: none">- Køb bliver registreret i systemet- Køb bliver gemt
Main Success Scenario	<ol style="list-style-type: none">1. Brugeren tilgår systemet og vælger menupunktet "Håndter motorhomes".2. Systemet beder om indtastning af stamdata for den indkøbte motorhomes der skal registreres.3. Brugeren får mulighed for at indtaste kundens stamdata i følgende rækkefølge: ID, pris, nummerplade, mærke, model, årgang, farve, sengepladser4. Brugeren godkender bekræftelsen.5. Systemet fører brugeren tilbage til "Motorhome Oversigt."

Extensions	<p>3a. Efter hver indtastning af et info-punkt gives brugeren mulighed for at godkende indtastningen.</p> <ol style="list-style-type: none"> 1. Brugeren <i>godkender</i>, og systemet giver mulighed for oprettelse af ny Motorhome. <p>3b. Efter hver indtastning af et info-punkt gives brugeren mulighed for at godkende indtastningen:</p> <ol style="list-style-type: none"> 1. Brugeren <i>afviser</i>, og bedes af systemet til at genindskrive en korrekt information. <p>3c. Hvis der er sket en fejl under oprettelsen af de to kunder:</p> <ol style="list-style-type: none"> 1. Brugeren informeres om fejlen. 2. Brugeren bringes tilbage til 'Opret Motorhome' menuen. <p>4a. Hvis brugeren ikke godkender bekræftelsen:</p> <ol style="list-style-type: none"> 1. Der oprettes ingen nye Motorhomes i systemet. 2. Brugeren bringes tilbage til 'Håndter Motorhomes' menuen.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Almost constantly
Miscellaneous	

Opret kunde

(ME, DA, OSB, HM)

Use Case Name	Opret kunde
Scope	Administrativt it-system
Level	User-goal level
Primary Actor	<ul style="list-style-type: none"> - Ejere - Medarbejdere
Stakeholders and Interest	<p>Ejerne har interesse i at kunder bliver oprettet da flere kunder øger indtægten.</p> <p>Medarbejdere har interesse i oprette kunder på en let og overskuelig måde</p> <p>Kunder har interesse i at deres data bliver gemt til senere genbrug.</p>
Preconditions	At brugeren er logget ind, og står i 'Håndter motorhomes/opret køb'-menuen.
Success Guarantee	<ul style="list-style-type: none"> - Kunden bliver oprettet - Kunden bliver gemt
Main Success Scenario	<ol style="list-style-type: none"> 1. Brugeren tilgår systemet og vælger menupunktet "Håndter Kunde". 2. Systemet beder om indtastning af stamdata for den oprettede kunde. 3. Brugeren får mulighed for at indtaste kundens stamdata i følgende rækkefølge: ID, fornavn, efternavn, adresse, kørekort nr, CPR 4. Brugeren godkender bekræftelsen. 5. Systemet fører brugeren tilbage til 'Kunde Oversigt'.
Extensions	<p>3a. Efter hver indtastning af et info-punkt gives brugeren mulighed for at godkende indtastningen.</p> <ol style="list-style-type: none"> 1. Brugeren <i>godkender</i>, og systemet giver mulighed for oprettelse af ny kunde.. <p>3b. Efter hver indtastning af et info-punkt gives brugeren mulighed for at godkende indtastningen:</p> <ol style="list-style-type: none"> 1. Brugeren <i>afviser</i>, og bedes af systemet til at genindskrive en korrekt information.

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

	<p>3c. Hvis der er sket en fejl under oprettelsen af de to kunder:</p> <ol style="list-style-type: none"> 1. Brugeren informeres om fejlen. 2. Brugeren bringes tilbage til 'Opret Kunde' menuen. <p>4a. Hvis brugeren ikke godkender bekræftelsen:</p> <ol style="list-style-type: none"> 1. Der oprettes ingen nye kunder i systemet. 2. Brugeren bringes tilbage til 'Håndter Kunde' menuen.
--	---

Opret udlejning

(ME, DA, OSB, HM)

Use case name	Opret udlejning
Scope	Administrativt IT-system
Level	User goal
Primary actor	<ul style="list-style-type: none"> - Ejere - Medarbejdere
Stakeholders and interests	<ul style="list-style-type: none"> • NMR- Vil have præcis og fejlfri gennemførsel af booking. Der til et regnskab der bliver gemt og oprettet i systemet. • Kunde - Vil gennemføre en reservation og køb hurtigt og med minimal indsats. Vil nemt kunne se sine mulige reservation(er) • Staten - Vil modtage SKAT og moms osv. for hvert salg.
Precondition	Systemet er oppe og køre, brugeren er logget ind og at brugeren står i 'Håndter udlejning/opret udlejning '
Success Guarantees	Reservationen bliver bekræftet i systemet. Betaling er godkendt. Købet er noteret og regnskab opdateret.
Main Success Scenario	<ol style="list-style-type: none"> 1. Brugeren tilgår systemet og vælger menupunktet 'Opret udlejning' 2. Brugeren vælger at oprette udlejning. 3. Brugeren noterer kundens stamdata (ID, kørekort nr., navn, alder, CPR, adresse). 4. Brugeren færdiggøre udlejningen og den indtastede data bliver gemt.
Extensions	<p>1a Brugeren logger ind med forkert info.</p> <ol style="list-style-type: none"> 1. Systemet beder om brugerens korrekte info, og trinnet gentages <p>2a Brugeren kan ikke oprette en udlejning pga. mangel af ledige MH.</p> <ol style="list-style-type: none"> 1. Systemet viser meddelelse. 2. Reservation kan ikke gennemføres, og reservationen afsluttes. <p>3a Brugeren har ikke de gyldige oplysninger for at oprette en udlejning.</p> <ol style="list-style-type: none"> 1. Systemet viser meddelelse. 2. Systemet kan ikke gennemføres, og reservationen afsluttes. <p>4a Brugeren færdiggøre udlejningen, pga kunden fortryder.</p> <ol style="list-style-type: none"> 1. Systemet gemmer ikke dataen.

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

Domain Model

(ME, DA, OSB, HM)

Endvidere tegnede vi et domain model, hvor vi fik en bedre forståelse af virksomhedens koncept, og hvordan virksomheden fungerer.

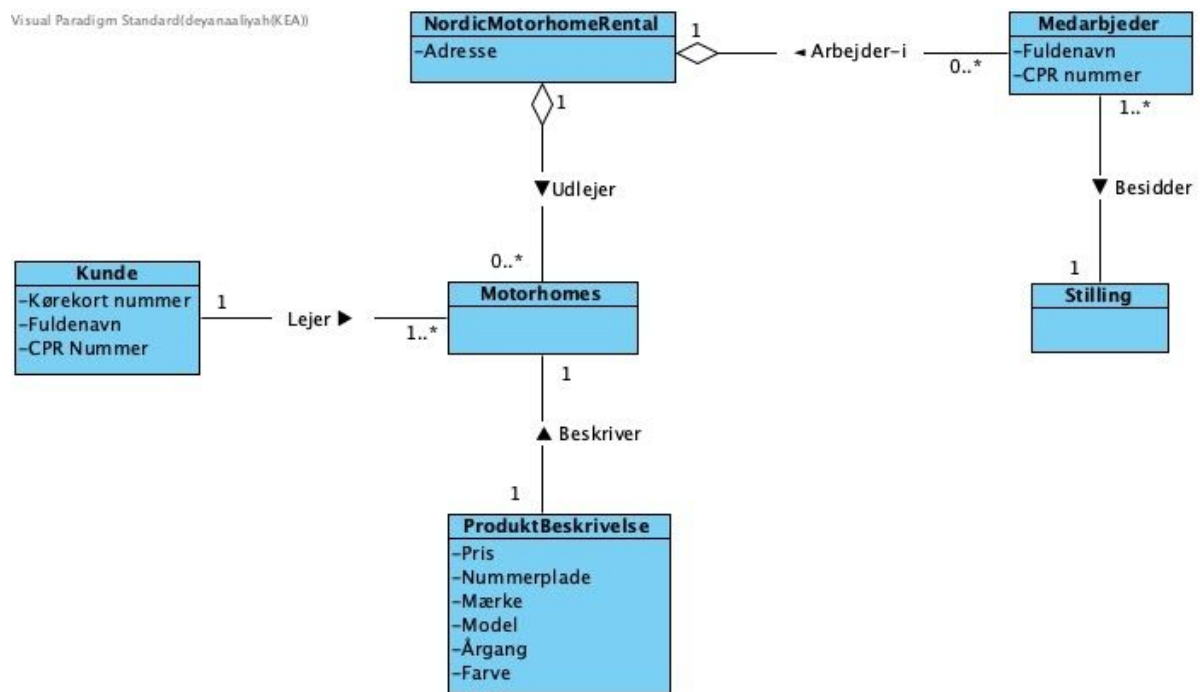


Diagram 2: Domain Model diagram

Efter dette step kunne vi tegne de næste diagrammer – vores SD og SSD.

SD og SSD diagrammer

Opret kunde

(HM, ME)

Som det fremgår på nedenstående billede af **System Sekvens Diagrammet**, er brugeren af systemet en medarbejder. Når medarbejderen skal oprette en kunde, får de først fremvist alle input felterne, hvor dataen om kunden indskrives.

Derefter godkender medarbejderen de indtastede oplysninger på kunden, hvorefter Systemet beder medarbejderne om at bekræfte godkendelsen. Herefter sender systemet medarbejderen retur til menuen 'håndter kunde'.

sd SSD-opretKunde

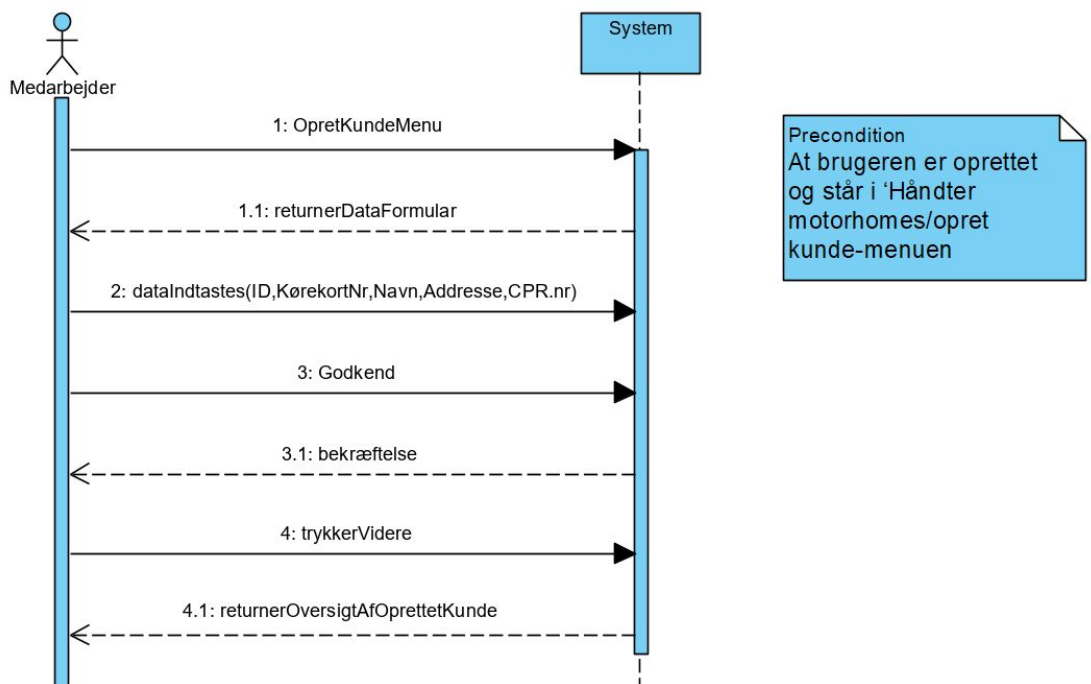


Diagram 3: System Sekvensdiagrammet over opret kunde

Som det fremgår på nedenstående billede af **Sekvensdiagrammet** for “opret-kunde” har vi en medarbejder, som er aktøren, der klikker på knappen “opret kunde”. Herefter bliver der sendt en operation i gang i software klassen “CustomerController”. Operationen bliver videresendt til Interfacet “ICustomer”, der implementeres i klassen “CustomerRepositoryImpl”, hvor metoden “create(Customer customer)” bliver kaldt. Parameteren “customer”, bliver sendt som en generisk type. Den nyoprettede kunde, bliver videresendt fra “CustomerRepositoryImpl” direkte til vores database i SQL. Slutteligt returneres “CustomerOverview.html” via vores “CustomerController”.

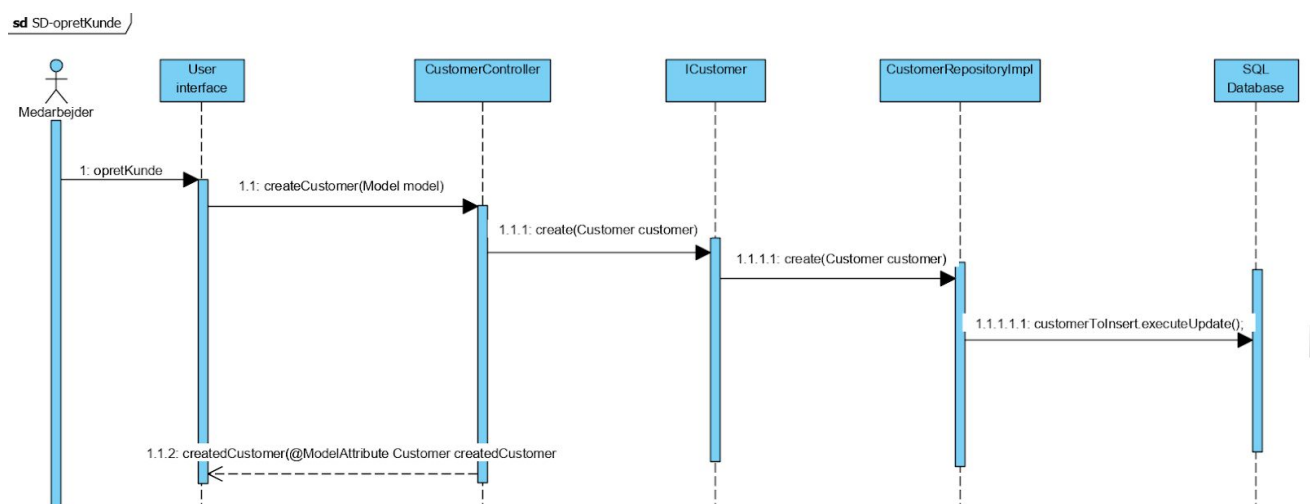


Diagram 4: Sekvens diagram over opret kunde.

Opret udlejning

(OSB, DA)

I det nedenstående billede af **System Sekvens Diagrammet**, er brugeren af systemet en medarbejder.

Medarbejderen vil oprette en udlejning til systemet og systemet vil returnere en udlejningsform, med alle diverse informationer nedenfor. Medarbejderen skal dertil godkende de indtastede oplysninger på udlejningen, hvorefter systemet beder om medarbejderen bekræfte godkendelsen. Dertil returnere systemet medarbejderen tilbage til udlejningsoversigt.

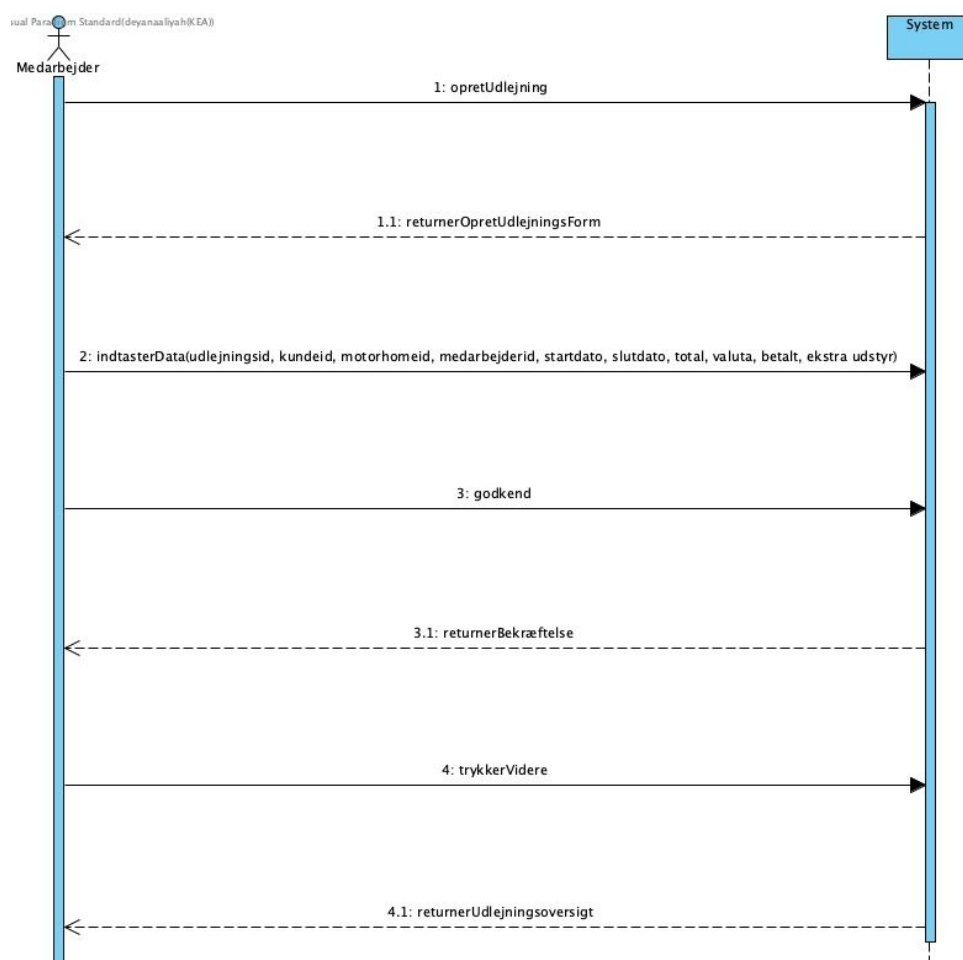


Diagram 5: Sekvens diagram over opret udlejning

I det nedenstående billede af **Sekvensdiagrammet** for opret-udlejning har vi en medarbejder, som er aktøren, der klikker på knappen “opret-udlejning”. Dertil bliver man videresendt til en operation i klassen “RentalController”. Den operation vil så blive videresendt til Interfacet “IRental”. som bliver implementeret i klassen

KEA

Interdisciplinary Exam Project

Nordic Motorhome Rental

Sem 2 2020©

“RentalRepositoryImpl”, hvor metoden “create(Rental rental)” bliver kaldt. Parameteret “customer”, bliver dertil sendt som en generisk type. Den oprettede udlejning bliver videresendt fra “RentalRepositoryImpl” til “nordic_motorhme_rental.rental”. Til sidst returneres rentalAdded.

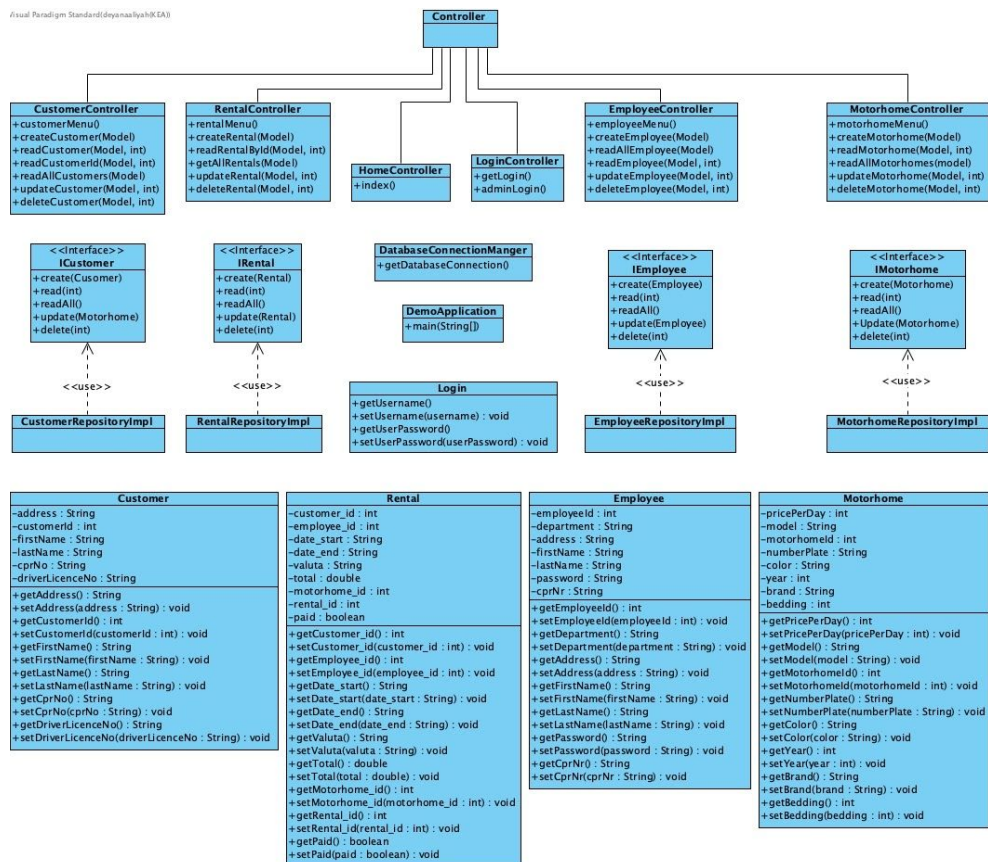


Diagram 6: System sekvens diagram over opret udlejning

Klassediagram

(DA, OSB)

Her vises et endeligt klassediagram af vores java projekt. Diagrammet blev opdateret løbende undervejs, da vi arbejdede i unified process. Det ses, at hvert repository implementation klasse har sin egen controller-, model- og interface-klasser. Vores login system er ikke højt optimalt, og har en statisk attribut – det betyder, at det ikke kan laves om. Derved fungerer den ikke som de andre controllere, der er forbundet med vores database.



KEA

Interdisciplinary Exam Project

Nordic Motorhome Rental

Sem 2 2020©

GRASP

(ME, DA, OSB, HM)

General Responsibility Assignment Software Pattern bedre kendt som GRASP, er et af mange principper, der anvendes i design til at give et overblik i hvilke klasser, der har hvilket ansvar. I bund og grund handler det om, at man skriver en klar og tydelig kode som gør det lettere at bygge på, for både en selv og for andre programmører i projektet. De forskellige underkategorier, der repræsenteres i GRASP, er følgende:

Creator som er et af de GRASP-principper, der kan beslutte, hvilken klasse skal være ansvarlig for at oprette nye instanser af en anden klasse. Dette resulterer i at have færre linjer koder, men derimod også langt mere ansvar. Dette kaldes for objektorienteret kode, og er en af de mest grundlæggende egenskaber i GRASP.

Information expert er det grundlæggende princip, hvorpå man tildeler ansvar til objekter. Et ansvar har brug for information for at det kan opfyldes. Det kan være informationer fra andre objekter, et objekts egen tilstand og information objektet kan udlede m.m.

Low coupling bygger på et princip om, at klasserne skal være selvstændige. Altså en klasse skal ikke være afhængige af en anden klasse. Hvis den ene klasse skal slettes, redigeres etc. skal det have minimal eller ingen indvirkning på de andre klasser.

High cohesion bygger på et princip om at reducere og opdele store klasser til mindre klasser. Det står i forlængelse til "Low coupling", hvor det er essentielt, at man reducerer den store indflydelse en klasse ændring har på de andre klasser. Hvis din klasse er High Cohesion, betyder det i bund og grund, at klassen skal have minimal funktion - med andre ord, skal den ikke både kunne beregne et matematikstykke og samtidig vise resultatet. Her inddeler du funktion i to klasser, hvor en klasser beregner (logisk-handling) og en anden klasse udprinter resultatet (UI-handling).

Controller er en slags ansvarlig koordinator mellem frontend og backend. Controlleren kan derfor udføre en eller flere handlinger, og kan returnere forskellige typer af

handlingsresultater fra en bestemt anmodning. Controlleren modtager input via UI og behandler derefter dataen og videresender resultaterne til et nyt view.

Ovenstående GRASP principper er de mest anvendte ud af de 9, der eksisterer. I rapporten har vi valgt at fokusere på følgende tre GRASP principper:

Controller

På figur 1, fremgår et udsnit af vores klassediagram. Vi har valgt at anvende GRASP i form af Controller princippet. Dertil har vi valgt at oprette følgende controller klasser:

MotorhomeController, KundeController, UdlejningController, MedarbejderController etc. De koordinerer vores frontend og backend ved hjælp af Java, HTML og CSS. Controller klasserne kan eksempelvis blive anmodet om input fra brugeren i form af udfyldning af formularer i forbindelse med oprettelsen af en kunde, udlejning eller medarbejder. Ydermere sørger de for at indhente data fra databasen og ud til et vilkårlig UI.

Creator

Der findes fire klasser i vores system der fungerer som Creator princippet i GRASP. Disse fire klasser er følgende:

- KundeRepositoryImpl
- MedarbejderRepositoryImpl
- MotorhomeRepositoryImpl
- UdlejningRepositoryImpl

De fire klasser opretter nye instanser af klasserne: Kunde, Medarbejder, Motorhome og Udlejning vha. en implementation af fire andre interface-klasser. I interface-klasserne findes der metoder der omhandler CRUD. Klasserne, hvoraf der bliver oprettet nye instanser, indeholder værdiløse attributter såsom:

```
private int id;  
private String motorhome;  
private double price; // og så videre
```

For en renere forståelse af dette eksempel, er dette klassen hvor man bruger Get og Set metoder.

Low coupling

I vores program har vi benyttet os af low coupling princippet i vores fire hovedklasser herunder (Kunde, Medarbejder, Motorhome og Udlejning). Dvs. at de nævnte klasser er uafhængig af andre klasser, men associerer med klasser som [x]RepositoryImpl. At de er uafhængige betyder med andre ord, at hvis der skulle ændres i [x]RepositoryImpl-klasserne, ville de fire low coupling-klasser ikke påvirkes.

Delkonklusion

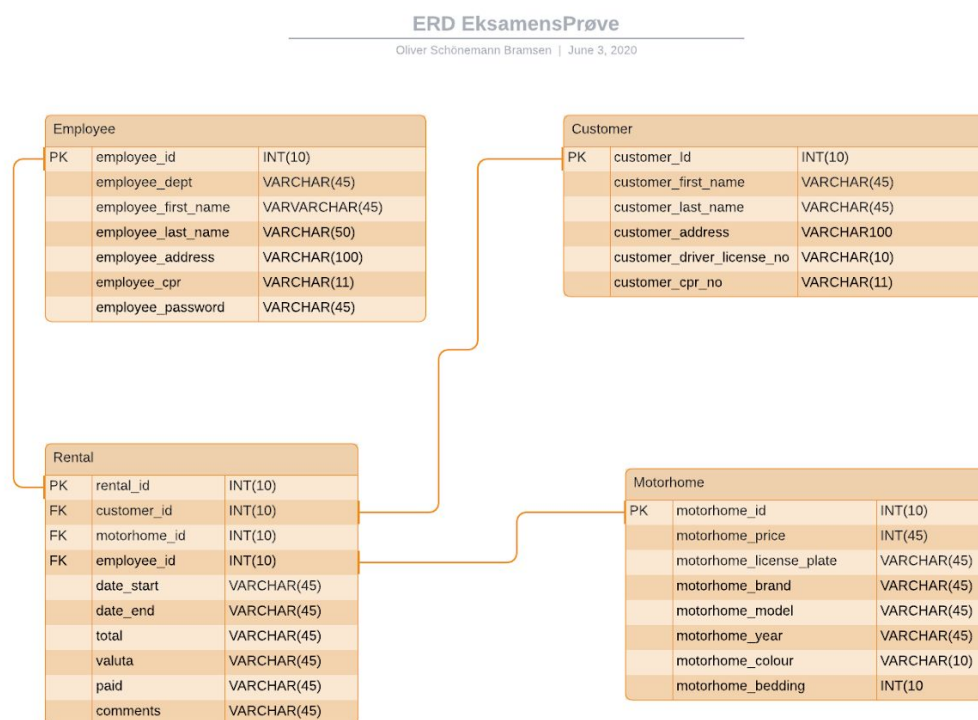
Vores faseplan giver et overblik over vores arbejdsprocess. I denne forbindelse har vi udarbejdet og oprettet modeller og diagrammer herunder FURPS+. Dertil følger at vi har lavet et use-case diagram af vores use-cases. Vi har således udvalgt 3 centrale use-cases som er beskrevet i fully dressed format. Herefter har vi lavet vores system sequence diagrammer baseret på vores fully dressed main success scenarios. Ydermere har vi en oversigt af vores klasser i form af et klassediagram som vi løbende har opdateret undervejs. I takt med kodning har vi fokuseret på at overholde GRASP principperne i form af Creator, Controller og Low coupling. Denne process blev gennemført iterativt og trinvist.

Softwarekonstruktion

E/R diagram

(OSB, DA)

E/R 'entitet relation' diagram. Vi anvender E/R for at kunne illustrere relationerne mellem employee, customer, rental og motorhome. E/R diagrammet bruges ofte til at designe eller fejlsøge i relationelle databaser.



E/R diagram

Vores E/R diagram viser at vi har 4 tabeller, employee, customer, rental og motorhome, hvori alle fire tabeller har sit eget primary key: id. Det bruger vi til at identificere vores data. Dertil har primary keys og foreign keys en til en relation med hinanden i rental databasen.

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

SQL beskrivelse

(HM)

Data Definition Language

```
CREATE TABLE employee(  
  
employee_id          INT(10)          NOT NULL,  
employee_dept        VARCHAR(45)       NOT NULL,  
employee_first_name  VARCHAR(45)       NOT NULL,  
employee_last_name   VARCHAR(45)       NOT NULL,  
employee_address     VARCHAR(100)      NOT NULL,  
employee_cpr         VARCHAR(11)       NOT NULL,  
employee_password    VARCHAR(45)       NOT NULL,  
  
PRIMARY KEY (employee_id)  
);
```

På ovenstående illustration fremgår et udsnit af vores database, hvor vi anvender en DDL Command i form af CREATE.

Syntax:

```
CREATE DATABASE <DATABASE_NAME>;
```

```
CREATE TABLE <table_name>;
```

Vi gør brug af kommandoen til at oprette en database ved navn `nordic_motorhome_rental` samt en tabel ved navn `employee`, som gemmer på data af medarbejderne. Ydermere er der kreeret kolonnenavne tilhørende hver deres data type.

```
DROP TABLE <kunde>;
```

På ovenstående illustration er der gjort brug af DDL kommandoen **DROP**. Den anvendes til at slette en tabel eller en database.

Syntax:

```
DROP DATABASE <DATABASE_NAME>;
```

```
DROP TABLE <table_name>;
```

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

Vi har gjort brug af kommandoen til at slette en tabel ved navn `kunde`, da alle kolonnenavne stod på dansk og skulle omskrives til engelsk.

Data Manipulation Language

```
INSERT INTO employee  
(employee_id, employee_dept, employee_first_name, employee_last_name,  
employee_address, employee_cpr, employee_password)
```

```
VALUES  
( '12341', 'Admin', Rasmus', 'Nielsen', 'Frederiksvej 2, 2730 Herlev',  
'220798-7351', 'passwordadminet');
```

I forbindelse med oprettelsen af databasen `nordic_motorhome_rental` og tabellen `employee` er der her anvendt en DML Command i form af **INSERT**. Dens funktion består i hjælp til at indsætte den tilhørende data.

Syntax:

```
INSERT INTO table_name VALUES(data1, data2, ...)
```

Man skal være opmærksom på, at rækkefølgen på værdien der indskrives i **VALUES(..)** ikke er tilfældig. Derimod afhænger det af rækkefølgen, på ovenstående kolonnenavne.

```
@Override  
public boolean create(Employee employee) throws SQLException {  
    PreparedStatement newEmployee = conn.prepareStatement("INSERT INTO employee  
VALUES (?, ?, ?, ?, ?, ?, ?)");  
  
    try{  
        newEmployee.setInt(1, employee.getEmployeeId());  
        newEmployee.setString(2, employee.getDepartment());  
        newEmployee.setString(3, employee.getFirstName());  
        newEmployee.setString(4, employee.getLastName());  
        newEmployee.setString(5, employee.getAdress());  
        newEmployee.setString(6, employee.getCprNr());  
        newEmployee.setString(7, employee.getPassword());  
  
        newEmployee.executeUpdate();  
    }catch (SQLException e){
```

KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

```

        e.printStackTrace();
    }
    return true;
}

```

Kommandoen er også at finde i vores program, hvor en medarbejder skal oprettes i en klasse ved `EmployeeRepositoryImpl`. På linje 23, fremkommer kommandoen, hvor spørgsmålstegnene i **VALUES**(..) indikerer, at der afventes input fra brugeren.

```

@Override
public boolean update(Employee Employee) throws SQLException {
    try{
        PreparedStatement updateEmployee = conn.prepareStatement("UPDATE employee
SET employee_dept=?, employee_first_name=?, employee_last_name=? WHERE
employee_id=?");

        updateEmployee.setString(1, Employee.getDepartment());
        updateEmployee.setString(2, Employee.getFirstName());
        updateEmployee.setString(3, Employee.getLastName());

        updateEmployee.executeUpdate();
    }catch (SQLException e){
        e.printStackTrace();
    }
    return true;
}

```

På dette billede fremgår en anden DML Commando – **UPDATE**. Den anvendes til at opdatere indholdet i en kolonne.

Syntax:

```
UPDATE table_name SET column_name = ? WHERE some_condition=?;
```

I klassen `EmployeeRepositoryImpl` bruges den også i samme forbindelse. Her giver programmet mulighed for, at man kan redigere/opdatere oplysningerne på en medarbejder.

```

@Override
public boolean delete(int id) {
    try {
        //Prepare et statement som sendes til MySQL workbench
        PreparedStatement employeeToDelete = conn.prepareStatement("DELETE FROM
employee WHERE employee_id = ?");
    }
}

```

KEA

Interdisciplinary Exam Project
 Nordic Motorhome Rental
 Sem 2 2020©

```

    // "?" er lig med den medarbejderens id der skal slette alt dens data
    employeeToDelete.setInt(1, id);

    //Execute statement
    employeeToDelete.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
return true;
}

```

Vi gør ligeledes brug af DML kommandoen **DELETE**. Den bruges også til at slette data fra en tabel.

SYNTAX: **DELETE FROM** table_name

I klassen *EmployeeRepositoryImpl* bruges den også i samme forbindelse. Her giver programmet mulighed for, at man kan slette en medarbejder.

Normalform

	rental_id	customer_id	motorhome_id	employee_id	date_start	date_end	total	valuta	paid
▶	20201	12020	101	7842	23-02-2020	04-02-2020	4,389	DKK	true
	20202	22020	801	4672	24-02-2020	07-03-2020	4,420	DKK	true
	20203	32020	301	8942	24-02-2020	09-03-2020	5,385	DKK	true
	20204	42020	401	1328	27-02-2020	13-04-2020	5,385	DKK	true
	20205	52020	701	4672	27-02-2020	15-05-2020	6,103	DKK	true
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Som det fremgår på ovenstående billede: læser man fra venstre mod højre, er primary key fra følgende tabeller: *customer*, *motorhome* og *employee* samlet i *rental* tabellen. I dette tilfælde bliver de således foreign key.

Vi har gjort brug af tredje normalform, da hver af foreign keys giver et syn på, hvilken kunde, medarbejder eller motorhome, der er tale om. *rental* tabellen har sin primary key – *rental_id*, som refererer til hvilken kunde, medarbejder og motorhome der er involveret i udlejningen. Dog skal der gøres opmærksom på, at vores yderligere kolonner: *date_start*, *date_end*, *total*, *valuta* og *paid* modvirker vores tredje form, da de kunne inddeles i henholdsvis:

- *date_id* I form af: *date_id*, *date_start* & *date_end*
- *price_id* I form af : *price_id*, *total*, *valuta* & *paid*

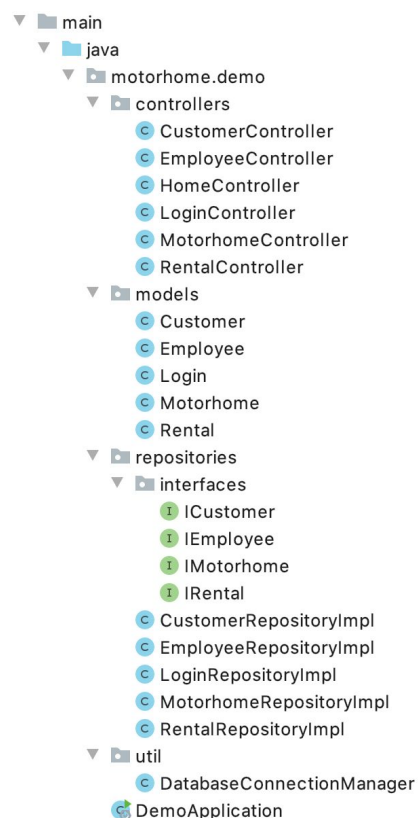
KEA

Interdisciplinary Exam Project
 Nordic Motorhome Rental
 Sem 2 2020©

Herefter fremgår det i **rental** tabellen er medvirkende til alle kolonnerne ville være uafhængige af hinanden. Fordelen ligger i, at dataen undgår at blive duplikeret og give et bedre overblik over dataen.

Kode og klasse beskrivelse

(DA)



Billedet til venstre, viser klassernes plads i vores IntelliJ projekt. Vi har samlet de tilsvarende klasser i deres repræsentative mapper. Mapperne er opdelt i models, controllers, repositories hertil liggende interfaces.

I kodedelen af projektet har vi brugt Spring Boot for at lave hjemmesiden. Spring Boot gør det let for os som programmører at bygge applikationer, da den har en vigtig hjælpende hånd i infrastrukturen. Den læser kode og klasser og downloader frameworks når de mangler. Derudover konfigurerer den automatisk en Tomcat der gør det muligt at køre ens Java kode i en webserver.

Udover Spring Boot bruger vi Thymeleaf som er en slags skabelon engine indenfor web. Grunden til vi bruger Thymeleaf blandt de engines vi har i projektet er fordi, Thymeleaf gør det muligt for os transformere data til

display eller modtage input via webserveren. Nedenfor ses det, at vi har brugt Thymeleaf til at printe nogle dataer fra vores database. Dem henter vi ved hjælp af vores model-, repository implementation- og controller-klasser i vores HTML.

```
<tr th:each="r : ${rentals}">
  <th th:text="${r.rental_id}"></th>
  <th th:text="${r.customer_id}"></th>
  <th th:text="${r.motorhome_id}"></th>
</tr>
```

I vores model klasser instantierer vi attributter der er tæt inspireret af attributters navne i databasen. Det er selvfølgelig i forhold til deres datatyper – om det er en String, int eller boolean, etc.

```
public class Rental {  
  
    private int rental_id;  
    private int customer_id;  
    private int motorhome_id;  
    private int employee_id;  
    //getter & setter og toString() instantieret nedenfor...  
}
```

Dem kan vi manipulere ind i vores repository implementation klasser. Nedenfor ses metoden der læser alle udlejninger fra databasen.

```
@Override  
public List<Rental> readall() {  
    //Opretter en arrayliste som gemmer tabellen fra database  
    List<Rental> rentalInArraylist = new ArrayList<Rental>();  
  
    try {  
        //Prepare et statement som sendes til MySQL workbench  
        PreparedStatement ps = conn.prepareStatement("SELECT * FROM rental");  
        //Execute den given prepared statement  
        ResultSet rs = ps.executeQuery();  
        while (rs.next()) {  
            //... så længe der er noget i tabellen, så udfør de næste linjer koder  
            Rental rentals = new Rental();  
            rentals.setRental_id(rs.getInt(1)); //X  
            rentals.setCustomer_id(rs.getInt(2));  
            rentals.setMotorhome_id(rs.getInt(3));  
            rentals.setEmployee_id(rs.getInt(4));  
  
            //Endvidere skal disse dataer gemmes i arraylisten  
            rentalInArraylist.add(rentals);  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
    return rentalInArraylist;  
}
```

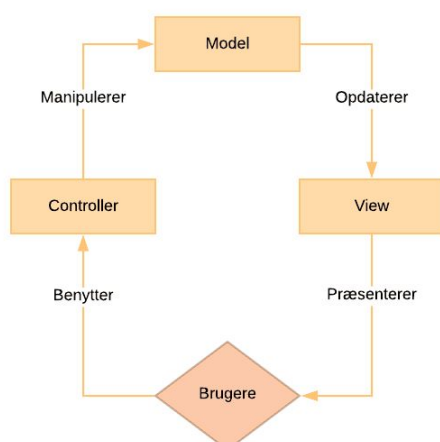
Her satte man den første attribut i databasen til at være lig med `rental_id` via `setRental_id()`; metoden på linjen hvor der er udkommenteret "X". Inde i metodens parameter, sammen med de andre set metoder (`customer_id`, `motorhome_id` og `employee_id`), skriver vi, at vores `ResultSet` (rs) kalder på vores get metoder og sætter

dem i databasens enkelte attribut – en ad gangen, så al’ data tilføjes i java. Altså vi kalder på set metoderne – attributten skal ændres. Vi giver metoden parameter, der gør det muligt at hente attributter fra databasen (de er indgivet i tal 1, 2, 3 og 4) og det er dét, de blive omdannet til. Lige nu er der hul igennem, og vi kan referere til databasens data via vores model klasse: Rental og RentalRepositoryImpl.

Hertil skal der gøres opmærksom på, at der anvendes en annotering af `@Override` linjen ovenfor `public List<Rental> readall()`. Annoteringen sker af den årsag, at repository klassen implementerer fra et interface. Dette interface er fra `IRental` klassen. Den er meget simpel og har fem metoder. Et interface er en klasse med metoder uden body. Det betyder, at der findes metoder uden funktioner. Funktionerne bliver kreeret i repository implementation klasserne, hvorfor den har de samme metoder. Metoderne har parametre og de implementeres også i repository implementation klasserne. Dette er vores IRental interface:

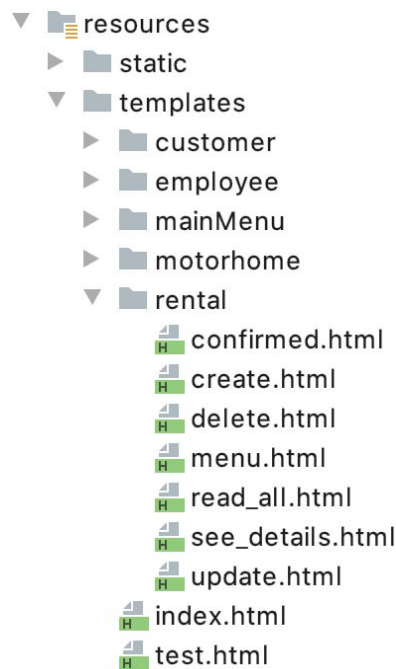
```
public interface IRental {  
  
    public boolean create(Rental udlejning);  
    public Rental read(int rental_id);  
    public List<Rental> readall();  
    public boolean update(Rental rental);  
    public boolean delete(int rental_id);  
}
```

Lige nu er vores applikation fuld funktionel. Vi kan oprette, læse, redigere og slette data som ændres direkte i databasen. Det betyder, at vi nu har instantieret vores model. Model findes i forkortelsen MVC som står for Model View Controller. Model i denne sammenhæng er vores database. View er de visuelle elementer vi mangler, modellen skal også modtage input og controlleren er logikken der håndterer end-userens input – altså controllerens hovedfunktion er at navigere. For at fuldende MVC, mangler vi view og controller.



Vi har tegnet et diagram der visuelt viser hvordan MVC arbejder. Brugeren er på websitet og derved benytter controlleren, der har logikken og navigerer brugeren til de ønskede requests. Controlleren manipulerer modellen – i vores tilfælde er det databasen den ændrer i. Og dernæst opdaterer

vores model de visuelle elementer der til sidst præsenteres til brugeren.



Vores view i denne sammenhæng består af HTML filer, hvori vi bl.a. bruger Thymeleaf til at hente data fra databasen. Alle filer er delt op i mapper for bedre oversigt og orden, og i mapperne customer, employee, motorhome og rental findes der fem forskellige HTML-filer der understøtter create, read, update og delete. Der findes også en HTML der hedder confirmation – som end-user af applikationen, viser dén egentligt bare en “Opgave vellykket” besked til brugeren, når han har oprettet eller rettet data. Derudover findes der også en menu HTML der lader brugeren vælge et af de 6 valgmuligheder – netop create, read, read all, update og delete. Det vil sige, at der er syv HTML-filer i hvert mappe.

I “update.html” under mappen “rental”, er dette den vigtigste del af HTML-filen. Når man vælger “opdater udlejning” vil kontrolleren navigerer brugeren til dette view. I viewet vises der den samme form indtastning som i “create” – men i denne er placeholderen lavet om til de selve eksisterende attributter. Først og fremmest kan form indtastningen håndteres vha. af attributterne `th:action="@{/opdater-udlejning}"`. Denne handling `th:action` bruges til at angive URL. De individuelle felter/data sker vha. attributten `th:field="*{rentalToUpdate.customer_id}"`, hvor navnet er objektets matchende property.

Udlejningen kan opdateres ved en input formular. Print alle udlejningsdata og modtag ændringerne. Front-end ser det således ud:

```
<h1>Opdater udlejning</h1>
<table>
  <form th:r="${rentalToUpdate}" th:action="@{/opdater-udlejning}" method="post">
    <tr>
      <label for="cid">Kunde ID</label>
      <input id="cid" type="number" th:field="*{rentalToUpdate.customer_id}">

      <label for="mid">Motorhome ID</label>
      <input id="mid" type="number" th:field="*{rentalToUpdate.motorhome_id}">
```

```
<label for="eid">Medarbejder ID</label>
<input id="eid" type="number" th:field="*{rentalToUpdate.employee_id}">
```

Og som bruger af websiden ser det sådan ud:
Dette er det visuelle element der ses, når der klikkes “redigér” ved en udlejning. Det vil sige, at man først skal se alle udlejninger, og så klikke “redigér” på den, som skal ændres. Denne handling stammer fra den samme HTML `th:r="${rentalToUpdate}"`. Den kommer fra HTML'en `read_all` i rental-mappen.

I den HTML, der gør, at vi kan læse alle udlejninger: har vi sat alle dataer op i kolonner og rækker – altså i en tabel. Det gør vi i `<tr>` tags – dette er rækken, hvor man inde i dét tag, skriver en `<th>` tag som er vores kolonne. På linje to har vi udkommenteret ordet “fast”, og det betyder egentligt, at alt under lige præcis det tag er der fast og øverst i tabellen. Vi slutter tagget efter `<th id="edit">ændr</th>` og starter på et nyt tag, der ser lidt anderledes ud. Det næste `<tr>` tag har en loop der itererer vores database, som skal printes ud. Tagget ser således ud `<tr th:each="r : ${rentals}">`. `rentals` som står inde i tuborg-klemmerne og efter `$`-tegnet, stammer fra vores `RentalController` klasse. Nu hvor man har skabt en iteration i `<tr>` tagget, så kan man anvende dem frit herunder. Det gør vi selvfølgelig. Vi kalder på `rental_id`, `customer_id`, `motorhome_id` og så derudaf – bemærk at det ikke er hele tabellen der printes ud. På grund af sikkerhed, tænker vi, at det er mest sikkert at printe vigtige, men ikke sårbare data. De kan printes ved at trykke på detaljer. Og det sker ved `<td>` tagget. Indeni det tag, har vi tre knapper – meget simpel vis: se detaljer, rediger og slet. Det funktionelle i knapperne er `<a>` tagget som har en `th:href`. Kort fortalt er det en attribut, og når den er behandlet, beregner den URL'en der skal bruges og indstiller værdien. Og vigtigst tillader det os til at have en statisk href attribut i vores HTML, så dens links forbliver navigerbar af en browser. In a nutshell: måden vi kan se detaljer, opdatere eller slette den specifikke udlejning, så har MVC brug for at vide hvilken præcis udlejning skal ændres. Præcis dét, sker ved

`...(id=${r.rental_id})}`". Ved de tre punktummer henvises der til hvilken funktion der ønskes. Eksempel ses på næste side.

```
<a th:href="@{/udlejning-detajler(id=${r.rental_id})}">
  <button>Detaljer</button>
</a>
```

Her henvises der til “udlejning-detajler” domænet fra vores controller.

Her ses HTML'en for read_all i rental mappen:

```
<tr>
  <th>Udlejnings ID</th> <!-- fast -->
  <th>Kunde ID</th>
  <th>Motorhome ID</th>
  <th>Dato for udlejning</th>
  <th>Dato for returnering</th>
  <th>Total</th>
  <th id="edit">Ændr</th>
</tr>
<tr th:each="r : ${rentals}"> <!-- iteration -->
  <th th:text="${r.rental_id}"></th>
  <th th:text="${r.customer_id}"></th>
  <th th:text="${r.motorhome_id}"></th>
  <th th:text="${r.date_start}"></th>
  <th th:text="${r.date_end}"></th>
  <th th:text="${r.total}"></th>
  <td>
    <a th:href="@{/udlejning-detajler(id=${r.rental_id})}">
      <button>Detaljer</button>
    </a>
    <a th:href="@{/opdater-udlejning(id=${r.rental_id})}">
      <button>Redigér</button>
    </a>
    <a th:href="@{/slet-udlejning(id=${r.rental_id})}">
      <button>Slet</button>
    </a>
  </td>
</tr>
```

Vores create HTML ligner lidt det samme som “redigér udlejning” fordi den har meget af den samme funktionalitet. `th:field="*(rentalToBeAdded.rental_id)"` går igen, og det virker ligeså godt. Andre ting der går igen er bl.a. `method="post"`. Den stykke linje fortæller at mængden af dataen ikke har en limitation, ikke kan ses ud fra URL og derved heller ikke blive bogmærket – en form for en sikker transaktion af data.

På næste side ses vores create HTML i rental mappen.

```

<h1>Opret udlejning</h1>
<form th:r="${rentalToBeAdded}" th:action="@{/opret-udlejning}" method="post">
  <tr>
    <label for="rid">Udlejnings ID</label>
    <input id="rid" placeholder="Indtast udlejnings ID..." type="number"
      th:field="*{rentalToBeAdded.rental_id}">

    <label for="cid">Kunde ID</label>
    <input id="cid" placeholder="Indtast kunde ID..." type="number"
      th:field="*{rentalToBeAdded.customer_id}">

    <label for="mid">Motorhome ID</label>
    <select id="mid" multiple th:field="*{rentalToBeAdded.motorhome_id}">
      <option value="101">Marco polo (2 pladser)</option>
      <option value="102">Marco polo (3 pladser)</option>
      <option value="103">Marco polo (4 pladser)</option>
      <option value="104">Marco polo (5 pladser)</option>
    </select>

    <label for="eid">Medarbejder ID</label>
    <input id="eid" placeholder="Indtast medarbejder ID..." type="number"
      th:field="*{rentalToBeAdded.employee_id}">

```

Det næste er selvfølgelig controlleren, som navigerer os de rigtige steder. I en controller klasse annoterer man at det er en controller klasse ved at skrive `@Controller` over ens klasse som vist nedenfor.

```

@Controller
public class RentalController {
  private IRental rentalRepository;
  public RentalController(){ rentalRepository = new RentalRepositoryImpl(); }

  @GetMapping("/udlejninger")
  public String getAllRentals(Model model){
    model.addAttribute("rentals",rentalRepository.readAll());

    return "rental/read_all";
  }

  @GetMapping("/opdater-udlejning")
  public String updateRentalForm(Model model, int id){
    model.addAttribute("rentalToUpdate", rentalRepository.read(id));

    return "rental/update";
  }

  @PostMapping("/opdater-udlejning")
  public String rentalUpdated(@ModelAttribute Rental rental){
    rentalRepository.update(rental);
  }
}

```

```
        return "rental/confirmed";  
    }  
}
```

Dernæst ses det, at metoderne enten har en `@GetMapping` eller `@PostMapping`. Ved hjælp af `GetMapping` sender brugeren et request som controlleren returnerer til viewet. Lad os antage, dette var `/opdater-udlejning`. Her ville den returnere `"rental/update.html"` som er vores opdateringsformular. Når man ændre udlejningen og submitter sender brugeren en post request, hvor systemet reagere ved et id som stammer fra det forrige get request. Så `GetMapping` er en get request hvor systemet nemt navigere dig, og `PostMapping` er når systemet kræver et id, så vi kan opdatere den specifikke udlejning.

HTML og CSS

(ME)

Vores hjemmeside er kodet i HTML og CSS, herunder er CSS-grid framework benyttet i form af `<header>`, `<main>` og `<footer>`. Desuden er grid også benyttet under vores CRUD-boks knapper som kan ses på billedet nedenfor.

```
/* CRUD-boksen */
.CRUD-box {
  display: grid;
  grid-gap: 2%;
  margin: 2% auto;
  padding: 2%;
  width: 40%;
  height: 20em;
  text-align: center;
  background-color: lightgrey;
  opacity: 0.8;
  border-radius: 2%;
}
```

Vi har sørget for at få struktureret vores kode bedst muligt. Det har vi gjort ved at tilføje kilde kommentarer hele vejen igennem i både vores Controllere, Java Klasser, HTML og CSS filer. på nedenstående billedet ses der et eksempel af vores velstruktureret html fil.

```

<!DOCTYPE html>
<!-- Language of website parameter-->
<html lang="DA" xmlns:th="http://www.thymeleaf.org">
<head>
  <!-- Basic Page Needs -->
  <!-- ***** -->
  <meta charset="UTF-8">
  <title>NMR</title>

  <!-- Mobile Specific Metas -->
  <!-- ***** -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <!-- CSS -->
  <!-- ***** -->
  <link rel="stylesheet" type="text/css" href="/css/styles.css">

  <!-- Favicon -->
  <!-- ***** -->
  <link rel="icon" href="/img/fav_icons.png" type="image">
</head>

<body>
<!--Navigation -->
<!-- ***** -->
<header>
  <div class="container">
    <nav>
      <ul>
        <a href="/home"></a>
        <li><a href="/motorhome">Motorhome</a></li>
        <li><a href="/medarbejder">Medarbejder</a></li>
        <li><a href="/kunde">Kunde</a></li>
        <li><a href="/udlejning">Udlejning</a></li>
      </ul>
    </nav>
  </div>
</header>

```

For at gøre vores kode læsevenligt og let at bygge videre på, har vi lavet flere fælles klasser til flere HTML elementer i de forskellige filer såsom knapper, input formular osv. Herefter har de fleste HTML elementer fået nogle unikke ID'er i tilfældet af, de skal få lavet et anderledes layout end de generelle regler.

Der skal lige siges at en klasse kan bruges gentagne gange og flere steder i de enkelte html filer hvorimod en id er et unikt navn som kun kan bruges en enkelt gang i hvert HTML fil. I forhold til vores kode har vi sørget for, at de overordnede/generelle CSS regler bliver designet i form af klasser. Der vises et eksempel af vores HTML knapper og CSS design på nedenstående billeder.

```
<div class="navigation-buttons">
  <a href="/motorhome"><button id="backBtn" type="button">Tilbage</button></a>
  <a href="/motorhome-overview"> <button id="continueBtn" th:method="post"
type="submit">Videre</button></a>
</div>
```

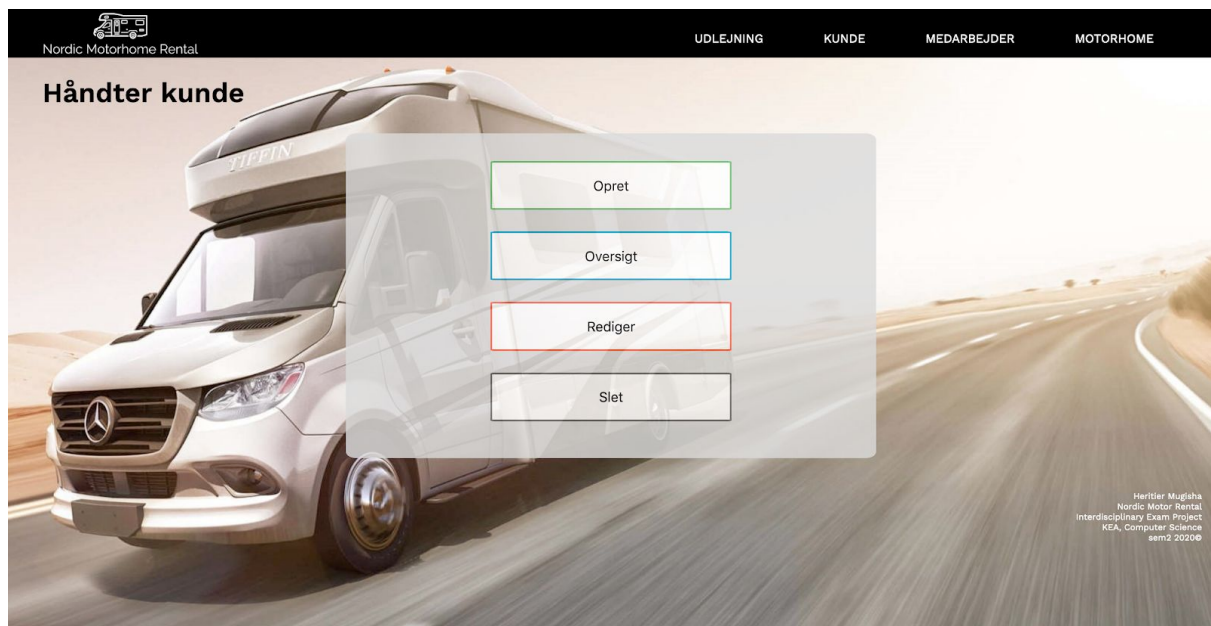
```
.navigation-buttons button {
  background-color: #4CAF50;
  width: 15%;
  text-align: center;
  margin: auto;
  color: white;
  padding: 12px ;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
/* tilbage knappen */
#backBtn{
  float: left;
  margin: 1%;
  background-color: red;
}
/* videre knappen */
#continueBtn{
  float: right;
  margin: 1%;
  background-color: green;
}
```

Fejlbeskrivelse & Forbedringer

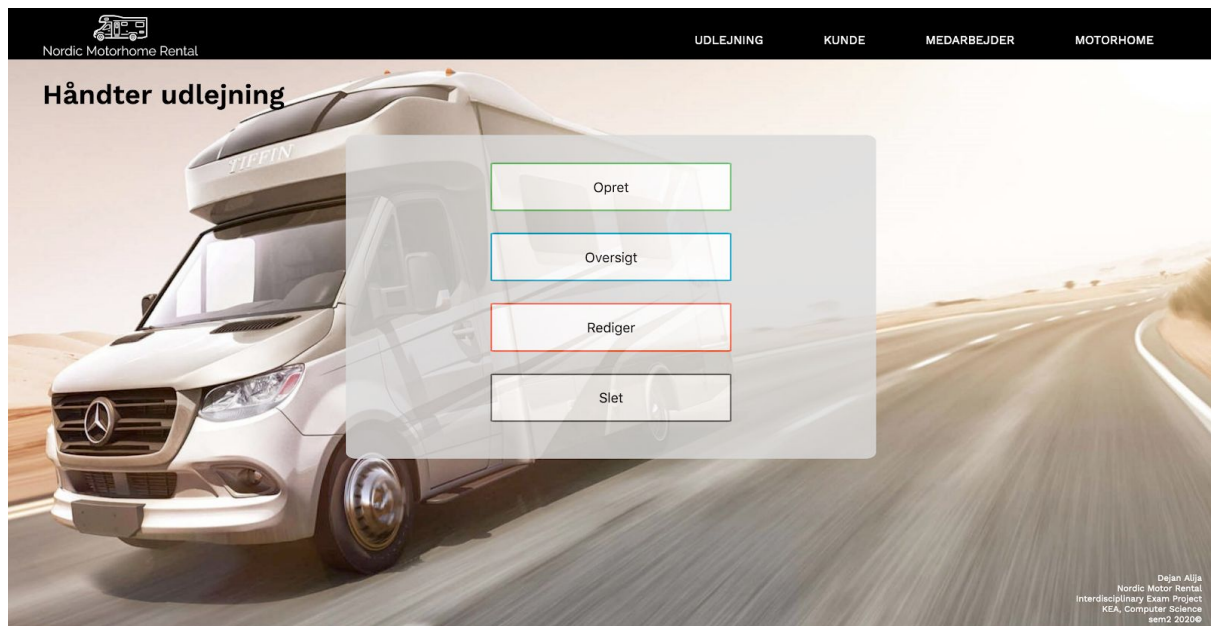
(HM)

Da programmet ikke er et fuldt funktionelt program, fremgår der fejl og mangel. Vi har valgt at fokusere på at gøre OPRET KUNDE og OPRET UDLEJNING fuldt funktionelle. Med henvisning til vores fully dressed, der beskriver deres funktionalitet, er kravene hertil opfyldt og yderligere krav er opfyldt, som det fremgår af nedenstående *Billede#1* & *Billede#2*.

Billede#1



Billede#2



KEA

Interdisciplinary Exam Project
Nordic Motorhome Rental
Sem 2 2020©

Det efterlader derfor mangel på nogle af vores andre klassers funktionalitet, såsom `Rental` og `Employee`.

```
@Controller
public class LoginController {

    @GetMapping("/")
    public String getLogin(){
        return "index";
    }

    @PostMapping("/")
    public String adminLogin(@ModelAttribute(name= "loginInput") Login
loginInput, Model model) {

        String username = loginInput.getUsername();
        String password = loginInput.getUserPassword();

        if("admin".equals(username) && "admin".equals(password)) {
            model.addAttribute("invalidSomething",false);
            return "mainMenu/mainMenu";
        }
        else{
            model.addAttribute("invalidSomething",true);
            return "index";
        }
    }
}
```

På den ovenstående illustration af vores `LoginController` optræder der en del mangel. Dels har vi ikke en forbindelse til vores database, hvilket resulterer i, at der kun findes et brugernavn og password. Login informationerne er ikke tilknyttet en bestemt medarbejder, men derimod gælder den samme for alle. Det giver en sikkerhedsbrist, da enhver med kendskab til det nemme `username` & `password` som er henholdsvis `"admin"` & `"admin"` kan få adgang til systemet. Det står klart, at en login information tilknyttet den enkelte medarbejder øger sikkerheden, hvorfor det er en forbedring som vil være åbenlys.



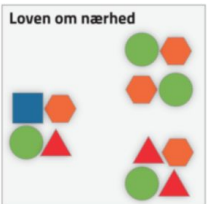
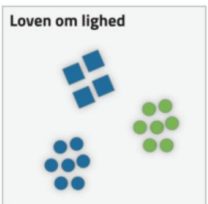
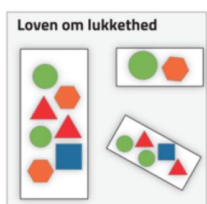
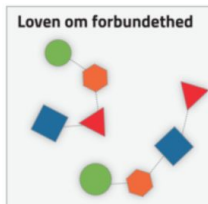

En anden mangel som medvirker til at sikkerhedsbristen forværres består i, at man fra login siden i søgefeltet kan tilføje følgende URL eksempel: `/kunde` efter portnummeret med andre ord - `8090/kunde`. Man vil herefter blive ført til siden som fremgår på **billede#1** længere

oppe. Det gør vores login system mindre sikkert, da man sagtens kan omgå den for at tilgå systemet.

Design forbedringer

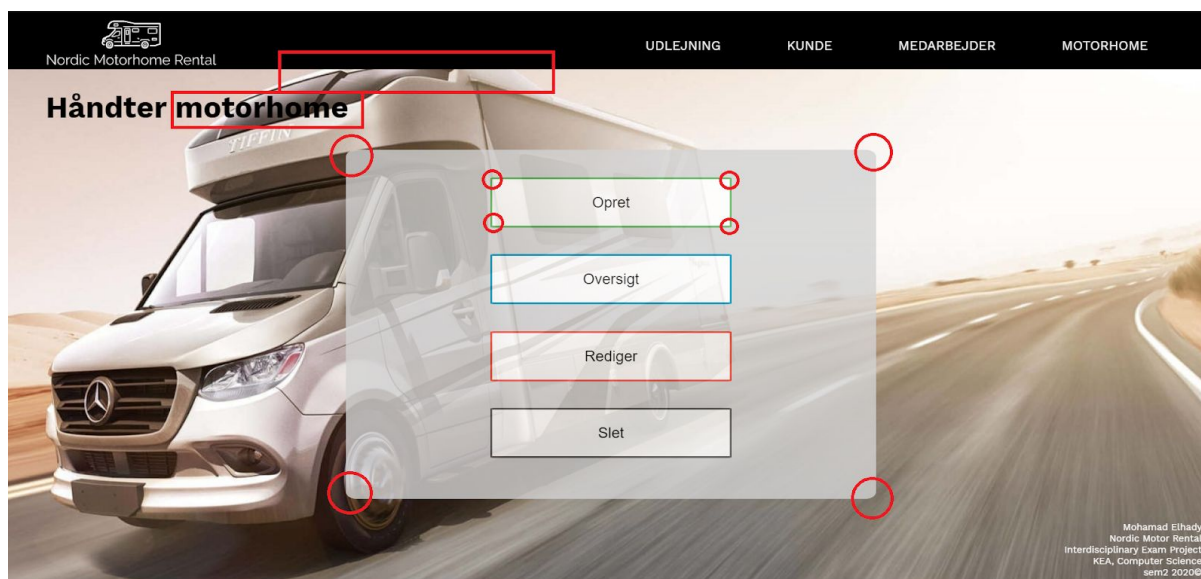
(ME)

I forhold til vores design af hjemmesiden har vi desværre ikke haft fokus på design regler såsom Gestaltlove. Der skal lige nævnes at Gestaltlove øger brugervenligheden på programmer, hjemmesider osv., ved at hjælpe brugerne med at fokusere på indholdet. De enkelte Gestaltlove handler om hvordan vi som mennesker opfatter og organiserer elementer i vores hjerner. Gestaltlove omfatter følgende love:

Nærhed	Lighed	Lukkethed	Forbundtlighed	Figur & baggrund
Symboler der er anbragt sammen, opfattes som hørende sammen	Symboler der ligner hinanden, opfattes som hørende sammen	Symboler der står i samme ramme opfattes som hørende sammen	Symboler der er forbundet, opfattes som hørende sammen	Den mindste afgrænsede figur på arealet vil først blive opfattet som figuren
				

Ved at sammenligne Gestaltlove med vores design af hjemmesiden har vi noteret følgende mangel som kan finjusteres.

På nedenstående billede kan man eksempelvis se at boksen omkring vores CRUD-knapper har en **border-radius** hvorimod knapperne ikke gør brug af **border-radius**, hvilket modsiger loven om lighed.



En anden bemærkning er, at det kunne være irriterende for øjnene at overskriften ligger oven på baggrundsbilledet.

Man kan også finde ud af om hvilke billeder og farver der passer bedst til sådan en platform og evt. teste programmet på rigtige brugere for at sikre brugervenligheden er i top.

Delkonklusion

Vi har udarbejdet et E/R diagram der viser forholdet mellem vores tabeller i databasen. Der er lavet en SQL beskrivelse af både DDL og DML og deres tilhørende kommandoer. Derudover er der blevet udformet en analyse af normalisering til 3 normalform. I forhold til kode delen har vi lavet en beskrivelse af vores kode i Java samt HTML og CSS – herunder forklaring på MVC. Slutteligt er der blev lavet et afsnit af fejlbeskrivelser og forbedringer.

Konklusion

(DA, ME, HM, OSB)

NMR er under vækst og har planer om at investere i et system, derfor ses det gerne, at systemet også er overkommeligt. Deres nuværende situation er, at man bruger Office-pakken – altså et færdiglavet program. Dette bedømmer vi til at være mindre holdbart og under minimal sikkerhed på længere sigt. Her konkluderer vi, at det er langt mere sikkert at få et system, der er designet til ejernes behov – det vil sige det administrative arbejde, som kun er for medarbejdere og ikke offentligt. Dette betyder også at, systemet ikke bliver overbelastet på nogle tidspunkter, og systemet kan opdateres og få nogle features – og måske endda bruges til at programmere et nyt system for det offentlige (kunder) også.

Det er derudover både teknisk, økonomisk og operationelt gennemførligt, da vi er en gruppe studerende på KEA, og dermed har frit adgang til velkendte og testede programmer samt licenser. I systemet herunder, bliver data gemt offline og lokalt. Dette resulterer i formindsket risiko for evt. hacking. Vi kan således konkludere, at vores feasibility-undersøgelse afsluttes med et klart “go”.

Ud fra vores undersøgelse om feasibility studies er vi kommet frem til, at projektet er lettere gennemførligt og afsluttes med et “go”. I forhold til vores risikoanalyse kan vi konkludere, at der er for få risici til at undlade gennemførsel af projektet. Dog skal der gøres opmærksom på, at i og med virksomheden har få ansatte anbefales der, at de følger deres plan om ansættelse af flere medarbejdere. Vores SWOT analyse vurderer, at NMR har stærke egenskaber i form af eksempelvis placering og godt omdømme. Svage sider er eksempelvis mangel på ansatte i fremtiden, administrativt papirarbejde og risiko for få motorhomes. Vores interessentanalyse peger på forskellige aktører der har interesser i virksomheden.

Vores faseplan giver et overblik over vores arbejdsprocess. I denne forbindelse har vi udarbejdet og oprettet modeller og diagrammer herunder FURPS+. Dertil følger at vi har lavet et use-case diagram af vores use-cases. Vi har således udvalgt 3 centrale use-cases som er beskrevet i fully dressed format. Herefter har vi lavet vores system sequence diagrammer baseret på vores fully dressed main success scenarios. Ydermere har vi en oversigt af vores klasser i form af et klassediagram som vi løbende har opdateret undervejs.

I takt med kodning har vi fokuseret på at overholde GRASP principperne i form af Creator, Controller og Low coupling. Denne process blev gennemført iterativt og trinvist.

Endvidere har vi udarbejdet et E/R diagram der viser forholdet mellem vores tabeller i databasen. Der er lavet en SQL beskrivelse af både DDL og DML og deres tilhørende kommandoer. Derudover er der blevet udformet en analyse af normalisering til 3. normalform. I forhold til kode delen har vi lavet en beskrivelse af vores kode i Java samt HTML og CSS – herunder forklaring på MVC –, der tæt forklarer de vigtige koder i vores system. Slutteligt er der blev lavet et afsnit af fejlbeskrivelser og forbedringer.

Bibliografi

	Bog	Forfatter	Udgave	Sider
Software design	Applying UML and Patterns	Craig Larman	3rd	131,171, 173-180,
ITO	Organisation	Hans Jørgen Erik Staunstrup Peter Storm-Henningsen	5rd	
Software konstruktion	https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html (Thymeleaf)			
	https://dzone.com/articles/add-login-to-your-spring-boot-app-in-10-mins (Login system)			
Diagrammer	Visual Paradigm			
	https://app.lucidchart.com/invitations/accept/11e0060b-c25f-4767-ada1-d0038c07d696 (E/R diagram)			
	https://app.lucidchart.com/invitations/accept/7c80ab1f-df90-4821-bba8-c5ef04771939 (MVC diagram)			

Henvendelser (links)

Platform	Links	Beskrivelse
GitHub	https://github.com/deyanaaliyah/exam_project_sem_2	Vores final kodeprojekt findes her.
Floobits	https://floobits.com/Mo2201/eksamensProj	Floobits er et plugin i IntelliJ der gør det muligt at flere personer kan kode i samme projekt.
Discord	App på Desktop	Discord er en chat kommunikationsplatform som har speciale indenfor lyd, video og skærmdeling.
Teams	App på Desktop	Teams er en anden chat kommunikationsplatform som har speciale indenfor lyd, video og skærmdeling.