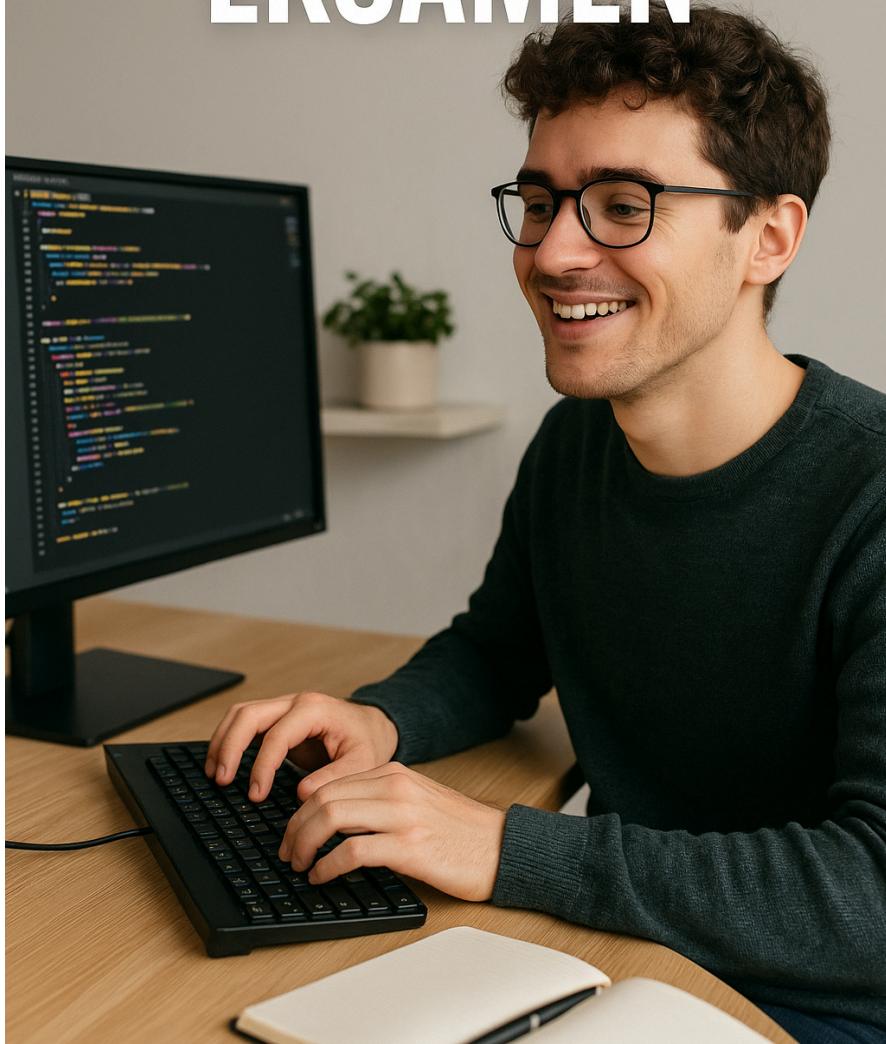


2. SEMESTER EKSAMEN



Udarbejdet af:
Kasper Sandholt
Martin L. Herrmann
Oliver Kronborg
Rasmus Dyring Raavig

Link til GitHub: (Dette projekt er kun så i kan se processen, da der er fejl vi ikke har
kunne fikse) https://github.com/Agentblank66/AutoM_2s

Link til Github: (Virkende Project) https://github.com/Agentblank66/AutoMeagler_s2_v2

Indholdsfortegnelse

Indledning: (Martin)	5
User Stories - Alle	6
<i>Story Points estimering: - Oliver</i>	7
Value Chain Analysis(VCA): (Martin)	7
Secondary Activities:	8
<i>Infrastruktur:</i>	8
<i>Human Resources / HR Management:</i>	8
<i>Technology Development:</i>	8
<i>Procurement:</i>	9
Primary Activities:	9
<i>Indkommende Logistik:</i>	10
<i>Operationer:</i>	10
<i>Udgående Logistik:</i>	10
<i>Markedsføring og Salg:</i>	10
<i>Service:</i>	11
Agile (Martin)	11
Mockup: (Kasper)	11
<i>Header:</i>	12
<i>Body:</i>	12
<i>Footer:</i>	12
<i>Header:</i>	12
<i>Body:</i>	13
<i>Footer:</i>	13
Sprint Backlog (Martin)	13
Sprints	14
Sprint 0 - Rasmus	14
Sprint 1	14
<i>Sprint Backlog - Rasmus</i>	14
<i>Daily Scrum - Rasmus</i>	15
<i>Sprint Review (Rasmus)</i>	16
<i>Sprint Retrospective (Rasmus)</i>	16
Sprint 2 - Rasmus	17
<i>Sprint Backlog</i>	17
<i>Daily Scrum - Rasmus</i>	17
<i>Sprint Review - Rasmus</i>	18
<i>Sprint Retrospective - Rasmus</i>	18
Sprint 3	18
<i>Sprint Backlog - Rasmus</i>	18
<i>Daily Scrum - Rasmus</i>	19
<i>Sprint Review - Rasmus</i>	19
<i>Sprint Retrospective - Rasmus</i>	19
<i>Increment (Kasper)</i>	19
Sprint 4	20
<i>Sprint Backlog - Rasmus</i>	20
<i>Daily Scrum - Rasmus</i>	20
<i>Sprint Review - Rasmus</i>	22

<i>Sprint Retrospective - Kasper</i>	22
<i>Increment - Kasper</i>	22
Kode Dokumentation	23
<i>Car</i> Kasper/Oliver.....	23
<i>Order (Martin)</i>	29
<i>User - Rasmus</i>	35
<i>NuGet (Kronborg)</i>	46
Diagram Dokumentation	46
<i>Domain Model</i>	46
<i>Domain Model- Første udkast: Bilag 2 (Kasper)</i>	46
<i>Domain Model- Andet udkast: Bilag 3 (Kasper)</i>	46
<i>Domain Model 2. sprint: Bilag 4 (Kasper)</i>	47
<i>Domain Model 3. sprint: Bilag 5 (Oliver)</i>	47
<i>Domain Model 4. sprint: Bilag 6 (Oliver)</i>	47
<i>Class Diagram</i>	47
<i>Class diagram – Første udkast: Bilag 7 (Oliver)</i>	47
<i>Class diagram – Andet udkast: Bilag 8 (Oliver)</i>	48
<i>Class diagram – 2 sprint: Bilag 9 (Oliver)</i>	48
<i>Class diagram – 3 sprint: Bilag 10 (Oliver)</i>	48
<i>Class diagram – 4 sprint: Bilag 11 (Oliver)</i>	48
<i>Razor page Diagram (Martin)</i>	49
BMC (Kasper)	49
E/R (Kronborg)	52
<i>Car</i>	52
<i>Images</i>	53
<i>Employee</i>	53
<i>Customer</i>	53
<i>Order</i>	53
<i>Leasing</i>	53
<i>Sale</i>	53
<i>Buy</i>	54
<i>Booking</i>	54
SWOT (Kronborg)	54
Unit Test (Martin)	56
Videreudvikling (Martin og Rasmus)	56
Refleksion (Kasper)	56
Konklusion (Kronborg)	57
Bilag (Alle)	57
<i>Bilag 1: User Stories</i>	57
<i>Domain Model Bilag</i>	61
<i>Bilag 2: Første udkast af domæne model - sprint 1</i>	61
<i>Bilag 3: Andet udkast af domæne model - sprint 1</i>	62
<i>Bilag 4: Udkast på domain model - Sprint 2</i>	63
<i>Bilag 5: Udkast på domain model - Sprint 3</i>	64
<i>Bilag 6: Slut domain model - Sprint 4</i>	64
<i>Class Diagram Bilag</i>	65
<i>Bilag 7: Første udkast på Class Diagram - Sprint 1</i>	65

<i>Bilag 8: Anden Udkast på Class Diagram - Slut Sprint 1.....</i>	66
<i>Bilag 9: Udkast på Class Diagram - Sprint 2.....</i>	67
<i>Bilag 10: Udkast på Class Diagram - Sprint 3.....</i>	68
<i>Bilag 11: Slut Class Diagram - Sprint 4.....</i>	69
<i>Bilag 12: Scrum syklus.....</i>	69
Sprint BackLog (Jira).....	70
<i>Bilag 13: Sprint Backlog - Sprint 1.....</i>	70
<i>Bilag 14: Sprint Backlog - Sprint 2.....</i>	71
<i>Bilag 15: Sprint Backlog - Sprint 3.....</i>	72
<i>Bilag 16: Sprint Backlog - Sprint 4.....</i>	73
Mockup (CKasper og Kronborg).....	74
<i>Bilag 17: Mockup Startside.....</i>	74
<i>Bilag 18: Mockup Køb Bil side.....</i>	74
BMC Model.....	75
<i>Bilag 19: BMC Model - sprint 1.....</i>	75
VCA.....	76
<i>Bilag 20: VCA - Secondary Activities.....</i>	76
<i>Bilag 21: VCA - Primary Activities.....</i>	76
ER Diagram (Kronborg).....	77
<i>Bilag 22: ER Diagram - sprint 1.....</i>	77
Razor Page Diagrammer.....	78
Sprint 1.....	78
<i>Bilag 23: Pages/Order.....</i>	78
<i>Bilag 24: Pages/Car.....</i>	79
<i>Bilag 25: Pages/Booking.....</i>	80
<i>Bilag 26: Pages/User.....</i>	81
Sprint 2.....	82
<i>Bilag 27: Pages/Order.....</i>	82
<i>Bilag 28: Pages/Car.....</i>	83
<i>Bilag 29: Pages/User.....</i>	84
Sprint 3.....	85
<i>Bilag 30: Pages/Order.....</i>	85
<i>Bilag 31: Pages/Car.....</i>	86
<i>Bilag 32: Pages/User.....</i>	87
Sprint 4.....	88
<i>Bilag 33: Pages/Order.....</i>	88
<i>Bilag 34: Pages/Car.....</i>	89
<i>Bilag 35: Pages/User.....</i>	90
Database (Kronborg).....	90
<i>Bilag 36: Tabel diagram - sprint 1.....</i>	90

Indledning: (Martin)

Dette program er et Web Applikation, der er udviklet ved brug af ASP.NET Core Razor Pages. Programmet er lavet til at bruge og tage imod data, der handler om biler, køb af biler, salg af biler og leasing af biler. Der bruges brugerinteraktion til at eksekvere mange af programmets funktioner, vi har gjort vores bedste når det kommer til at gøre det simpelt og nemt anvendeligt for brugeren. Vi har haft det i mente at programmet skulle have modulært arkitektur, nemt at vedligeholde og nemt at skaler op hvis behovet opstår.

Programmet integrerer flere af Microsofts Entity Frameworks Core database kontekster, der skulle hjælpe os med at håndtere de forskellige typer data, som disse tre anvender; biler, ordrer og brugere. Strukturen af programmet er sat op på en lagdelt service-setup måde, dette kan ved at vi anvender transient, scoped og singleton services til at tilgå vores data og vores logik. Vi anvender også autentificering med cookie baseret login for øget sikkerhed.

User Stories - Alle

Bilag 1: User Stories

Business Points	User Stories	Acceptance Criteria(Customer)	Acceptance Criteria(Coder)	Story Points
1.	Jeg som ejer, ønsker at se en oversigt over alle biler, med det formål at se vores sortiment	Test at man kan se alle biler. Test at man ikke kan se en bil.	Test at data fra databasen bliver udskrevet på siden. Test at ingen data bliver udskrevet.	5
2.	Jeg som bruger, ønsker at kunne se alle biler der kan leases, med det formål at have en oversigt over alle biler der kan leases.	Test at man kan lease en bil. Test at man ikke kan lease en bil.	Test at en leaset bil bliver oprettet i en table. Test at den ikke tilføjer en leaset bil til tabellen ogcaster en exception.	5
3.	Som ejer ønsker jeg at kunne tilføje biler til vores sortiment, med det formål at kunne sælge flere biler.	Test at man kan oprette en bil, Test at man ikke kan oprette uden at udfyldte alt info	Test at der oprettes et element i databasen. Test der ikke oprettes	4
4.	Jeg, som ejer, ønsker at kunne fjerne biler fra hjemmesiden, med det formål at kunne fjerne solgte biler fra vores sortiment.	Test at den sletter Test at der kun slettes det ønskede	Test at den sletter fra databasen Test at der kun slettes det ønskede	4
5.	Som ejer, ønsker jeg at kunne redigere oplysninger om biler, med det formål at kunne ændre i bilens status.	Test at man kan redigere oplysninger på biler. Test at når man redigerer oplysninger på biler og det slår fejl.	Test at den opdaterer fra databasen Test at der kun opdatere det ønskede	4
6	Jeg, som kunde, ønsker at kunne se informationer om bilen, med det formål at tjekke bilens oplysninger.	Test at man kan se bilens oplysninger. Test at man ikke kan se bilens oplysninger.	Test at GetCar printer en bil. Test at GetCar ikke printer en bil.	4
7	Som ejer, ønsker jeg at kunne lave en ny ordre, med det formål at ordenen bliver gemt i databasen.	Test at ordenen er blevet gemt. Test at ordenen ikke bliver gemt.	Test at ordenen bliver gemt i databasen. Test at ordenen ikke bliver gemt i databasen.	4
8	Som ejer, ønsker jeg at kunne se alle ordrer der er blevet lavet, med det formål at kunne se tidligere ordrer.	Test at alle ordrer vises i tabellen. Test at ingen ordrer kan ses i tabellen.	Test at GetOrders() metoden returnerer alle ordrer. Test at GetOrders() ikke returnere noget.	5

Story Points estimering: - Oliver

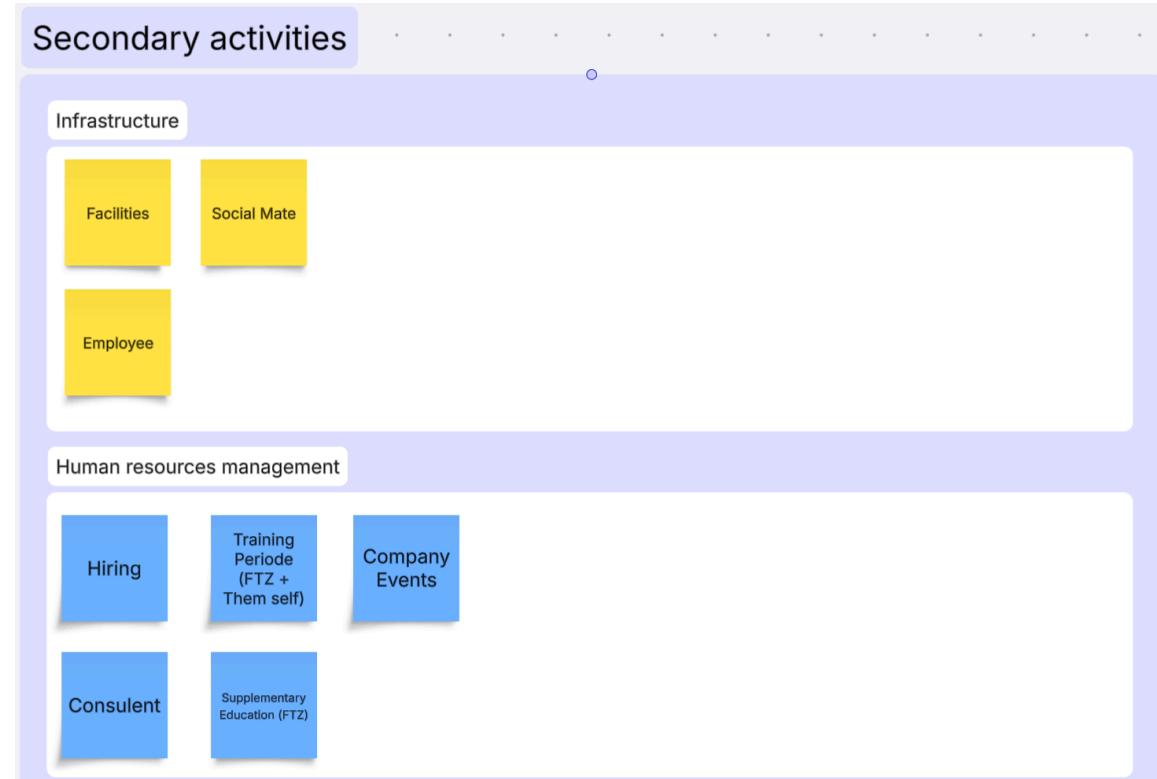
- Vi har valgt at bruge ti tals skala for at vurdere story points. Vi brugte Planning Poker metoden til at estimere story points, fordi vi gerne ville have hinandens subjektive meninger på hvor lang tid hvert ville tage.
 - 1 = Minutter
 - 2 = 10. Minutter
 - 3 = 20. Minutter
 - 4 = Time
 - 5 = Timer
 - 6 = Halv dag
 - 7 = Hel dag
 - 8 = To dag
 - 9 = Uge
 - 10 = Uge+

Value Chain Analysis(VCA): (Martin)

VCA er et diagram der bruges til at vurdere de forskellige ting et firma bruger, giver eller sender ud. Ud fra disse aktiviteter kan firmaet se hvor der er muligheder for forbedringer og så kan de foretage sig de overvejelser der er brug for i forhold til at tilføje eller fjerne værdi fra det færdige produkt eller service.

Vi har udført en VCA på vores produkt ejers firma Sjællands Automægler, og en del af den ende med at se sådan ud:

Bilag 20: VCA - Secondary Activities



Secondary Activities:

VCA - Secondary Activities er opbygget af fire sektioner plus margin. Disse sektioner hedder Infrastructure, Human Resources Management, Technology Development og Procurement:

Infrastruktur:

Inkludere firmaets systemer, opbygning og direktion afdeling. I dette firma er der følgende ting:

- Bilværksted: Bygning der tilhører hovedbygningen.
- Bilgård: Et område der ligger bag hovedbygningen, hvor bilerne står.
- Lager: Et område på firmaet grund, hvor der bliver opbevaret materiale af forskellige typer.
- Klargørelse Center: Et center hvor biler bliver klargjort til afhentning.
- Regnskabsafdeling: En afdeling hvor alt regnskab bliver håndteret.
- Administration: Det er en afdeling der står for at lave/tage opkald, tage mod kunder osv.
- Ansatte: De skal hjælpe firmaet med deres dag til dag opgaver.
- Social Mate (Markedsføring): Et andet firma der står for at designe meget af markedsføringen i samarbejde med CEO.
- Social Mate (Offentligt Billede): Et andet firma der står for at holde øje og håndtere det offentlige billede, i samarbejde med CEO.

Human Resources / HR Management:

Denne sektion hjælper med at håndtere hyring, træning, interne medarbejders problemer osv.:

- Hyring: Står for at hyre nye medarbejdere med de rigtige kompetencer.
- Træning af nye: Står for at lære nye, hvordan procedurer foregår på deres arbejdsplads osv.
- Extra Kurser: Hvis en medarbejder skal påtage sig nyt arbejde, kan det være personen skal anskaffe sig ny viden fra ekstra kurser hos FTZ.
- Konsulent: Er en sparringspartner til CEO, med forretnings og markedsføring ideer.
- Firmaevents: Er noget som firmaet holder for medarbejderne for at holde medarbejderne tilfredshed oppe.

Technology Development:

Dette er en afdeling hvor der bliver lavet eller forsket nye måder at designe eller udvikle procedurer og teknikker i forhold til deres processer.

- Market Tilpasning: Dette er den måde at firmaet udvikler og tilpasser sig markedsudviklingen.
- Internt IT System: Dette er det IT system som de har internt, som de selv laver forbedringer på.
- Eksternt IT System: Dette er det stykke IT system som de har udefra, som de ikke selv laver forbedringer til.

Procurement:

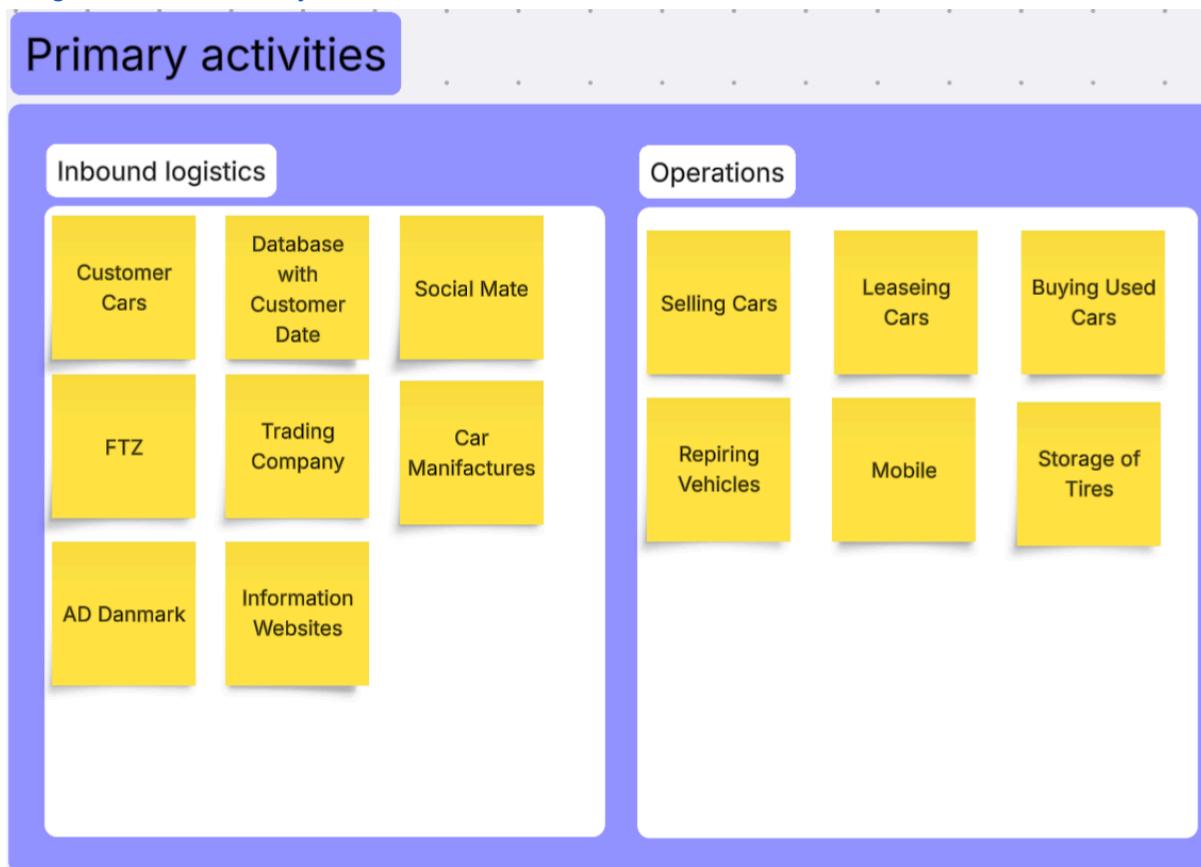
Denne procedure handler om, hvordan firmaet fremskaffer materiale.

- FTZ: Rigtig meget af der materiale firmaet anskaffer sig kommer fra FTZ, hvis FTZ ikke har det, går de til nogle af deres andre samarbejdspartnere.
- BTB: Dette er samarbejdspartnere som kun sælger til andre virksomheder.
- Teleselskab: Det er det abonnement som firmaet har aftalt med teleselskabet, hvor det opfylder de behov som firmaet har.
- Bilbasen: Er en hjemmeside, hvor de biler som firmaet har på deres hjemmeside, også bliver vist.
- Bilinfo: Dette er en side hvor firmaet tager eller sammenligner information omkring biler, hvis de ikke selv har informationen.
- Kunder (Biler): Det er de biler som firmaet køber af deres kunder, for at sælge dem videre til nye kunder.
- Bilproducenter: Hvis firmaet ikke har specifikke dele til en bil, kontakter de bilens firma og så bestil de dele som de skal bruge.
- Handelsvirksomheder (Værktøjer): Alle værktøjer som firmaet anskaffer sig kommer fra firmaer ligesom Elgiganten, Biltema osv.

Primary Activities:

De primære aktiviteter er dem som bliver brugt direkte til at lave produktet eller give deres service:

[Bilag 21: VCA - Primary Activities](#)



Indkommende Logistik:

Er de aktiviteter der involverer modtage, opbevaring og lagerstyring af materialer og komponenter.

- Kunde biler: De biler som firma køber af kunder.
- Bil Basen: En hjemmeside hvor firmaet henter information og lægger annoncer op omkring deres biler.
- Database med kundedata: Hvor der står et par af deres informationer og hvad der er blevet lavet på deres bil hos firmaet.
- FTZ: Et firma hvor de køber mange af deres ressourcer og materialer.
- Social Mate: Et firma som hjælper med at lave markedsføring og deres sociale billede.
- Bilinfo: En hjemmeside hvor firmaet finder information omkring biler.
- Handelsvirksomheder: Et firma der sælger redskaber og materialer.
- Bilproducenter: Hvis firmaet skal have fat i specifikke bildele.
- AD Danmark: Et firma hvor de køber materiale fra.

Operationer:

Aktiviteter der hjælper med at omdanne materialer og komponenter ind til færdige produkter.

- Selling Cars: De sælger de biler, som der er blevet købt af kunderne.
- Leasing Cars: De leaser biler ud til private og erhverv.
- Buying Used: De køber biler fra deres kunder.
- Reparation af biler: Kunder kan få deres biler fikset hos firmaet.
- Dækklager: Firmaet kan opbevare kunders dæk for dem.
- Mobile: De kan køre ud til deres kunder.

Udgående Logistik:

Det er de aktiviteter der involverer levering, pakning og forsendelse.

- Lager: Opbevare materiale og ressourcer. Dæk for kunder.
- Ordrestyringssystem: ?
- Levering: ?

Markedsføring og Salg:

Disse er de aktiviteter, der bruges i forhold til markedsføring og salg af firmaets produkt eller service.

- Kort Videoer: Små videoer der forklarer lidt om hvad firmaet gør og hvordan de kan hjælpe kunder og mange andre typer reklamer.
- Trustpilot: En hjemmeside hvor kunder deler deres oplevelser med firmaet både gode og dårlige.
- Deres egen website: Her kan kunder finde meget information om firmaet, og hvis de ikke kan finde svar på deres spørgsmål, kan de kontakte kundeservice.
- Bilnøgler: Når du køber eller leaser en bil hos dem, får man en lille keychain med deres firma logo og navn på med.
- Firma-bil: Firmaet har nogle biler med logo og navn på, en form for selv reklamation.

- Netværk: De er i nogle grupper, hvor de står sammen om at støtte det lokale samfund såsom sportsklubber osv.

Service:

Det er de aktiviteter der tager plads efter og under salget eller servicen. Dette kan være (installation, træning, kvalitetskontrol, reparationer og kunde service).

- Kender deres kunder: De tager sig tid til at lære deres kunder, så de får et meget personligt forhold til hinanden.
- Kunde goder: Til deres stamkunder, giver de nogle gange små venlige rabatter eller andre godter.
- Dårlige anmeldelser: Hvis en kunde giver en dårlig anmeldelse eller konstruktiv kritik, beder firmaet om en forklaring, hvad de ikke kunne lide og hvad de kunne ændre.
- Personlig Service: De vil gerne have alle kunder ned i butikken, hvor de så får personlig betjening af en medarbejder.
- Kunder aftaler: Firmaet indgår nogle gange individuelle aftaler med deres kunder hvor kunder kan få unikke aftaler på priser eller andre ting.
- Ordentlige priser: Firmaet går meget op i, at deres priser er rimelige og fair.

Agile (Martin)

Vi har tænkt os at anvende scrum til at kunne holde øje med hvordan projektet forløber sig. Scrum er bygget på tre ting; (Gennemsigtighed, Inspektion og Adaptivitet) hvilket gør det fleksibelt. Ud fra det vi lærer kan vi tilpasse hvordan vi gør ting og hvordan vi kommer til at gøre ting.

Scrum Master

❖ Rasmus Dyring Raavig

Development Team

Kasper Sandholt, Oliver Kronborg, Martin Herrmann og Rasmus Dyring Raavig.

Mockup: (Kasper)



Bilag 17

Header:

Her ses vores startside. Til vores startside, har vi draget inspiration fra virksomhedens allerede eksisterende hjemmeside. Vores header består af de 3 mest essentielle ting. Køb bil, sælg bil og leasing. Grunden til at de er placeret i midten, er at øjet altid fanger det som står i toppen. Vi har nærstuderet andre bilforhandleres hjemmesider, og fællesnævneren for alle siderne er at de vigtigste rubrikker ligger i toppen. Vi har en rubrik som hedder Login. Her skal det være muligt for medarbejdere og kunder at logge ind.

Body:

I kroppen har vi et stort billede af Product Owner, som står foran sin virksomhed. Grunden til vi har valgt et stort billede, er at det styrker troen på at det er en pålidelig virksomhed. Det første man ser når man kommer ind på hjemmesiden er dette billede, og det er vigtigt at skabe et godt førstehåndsindtryk. Når man har et stort billede, viser det en form for selvtillid og selvsikkerhed, man sikkert kan handle hos forhandleren. En visuel tiltalende hjemmeside er kun positiv, og giver brugeren lyst til at komme tilbage igen.

Footer:

I vores footer har vi diverse oplysninger om virksomheden. Alt fra åbningstider til et link til deres sociale medier. Vi har taget de informationer, som står på den eksisterende hjemmeside, og givet dem vores eget præg.



Bilag 18

Header:

I vores header har vi samme layout som på startsiden. I billedet har vi glemt at inkludere rubrikkene Køb bil, salg af bil og leasing. Disse rubrikker skal være synlige på alle siderne, så man ikke skal på startsiden hver gang.

Body:

Når man trykker på "Køb Bil", bliver man henvist til denne side. I vores body har vi lavet nogle sorteringsfiltrer.

"Søg efter bil" er en søgbar, hvor man kan indtaste navn på bilmærke, hvor den søger i en liste, og viser de biler som søges efter.

I "Vælg Mærke", har vi lavet det som en drop-down menu. I den drop-down menu, kommer der en lang række bilmærker frem. Når man vælger en bil, kommer bilen/bilerne frem.

I "Vælg drivmiddel", har vi gjort det samme som at vælge mærke, og lavet en drop-down menu, hvor den viser de forskellige drivmidler, så man kan sortere efter benzin, diesel, hybrid og elbil.

I "Vælg Type", kommer der en menu som dukker op fra højre side, hvor man kan sortere efter type bil. Det vil sige SUV, Stationcar, Sedan osv. Teksten indeholder et billede, som gør layoutet behageligt at kigge på. Her har vi draget inspiration fra Bilbasen.

Under søg bil, har vi lavet en bar, hvor man kan justere prisen. Denne bar fungerer på den måde, at mindsteprisen altid justerer efter billigste bil, og højeste pris, altid er justeret efter dyreste bil.

Samt har vi en drop-down menu, hvor man kan sortere efter diverse ting. Her kan man sortere efter pris, kilometer osv.

Den nederste del af bodyen viser den default biler, som vises før søgekriterierne er indtastet.

Footer:

I vores footer har vi diverse oplysninger om virksomheden. Alt fra åbningstider til et link til deres sociale medier. Vi har taget de informationer, som står på den eksisterende hjemmeside, og givet dem vores eget præg.

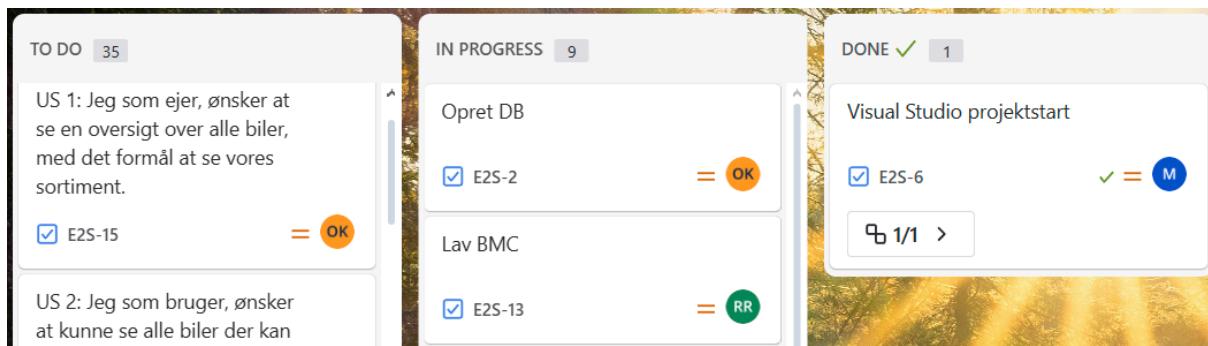
Sprint Backlog (Martin)

Vi har valgt at bruge Jira, et program der har samme funktion som Trello. Dette program bruges til at holde styr på hvad der skal laves, hvornår og af hvem. Selvfølgelig er intet skrevet i sten, navnet på opgaver er bare den person som står for at opgaver bliver lavet, man er altid velkommen til at søge hjælp udefra.

Vi har opdelt vores opgaver i tre kategorier (To Do / Opgaveliste, In Progress / Undervejs og Done / Færdig).

- ❖ Alt hvad der er i opgavelisten, er ting vi gerne vil arbejde på, men opgave er ikke blevet uddelt til nogen i nu.
- ❖ Alt, hvad der er i undervejs, er opgaver som er blevet arbejdet på af en eller flere gruppemedlemmer.
- ❖ Alt, hvad der er i Færdig, er opgaver som er blevet færdiggjort.

Her har vi et snippet af et større billede af vores sprint backlog af sprint 1:



Vi kan se på billedet at vi er påbegyndt diagrammerne og vi har oprettet vores Visual Studio projekt, men vi er ikke påbegyndt at implementere user stories i nu. [Bilag 13 - Sprint 1](#), [Bilag 14 - Sprint 2](#), [Bilag 15 - Sprint 3](#), [Bilag 16 - Sprint 4](#).

Sprints

Sprint 0 - Rasmus

Vi mødtes med Product Owner den 9 april klokken 9:30, hvor vi snakkede med ham omkring hvordan han vil have sin hjemmeside skal se ud. Vi stillede ham nogle spørgsmål vi havde forberedt, og han snakkede omkring hans forretning. Da han havde en hjemmeside i forvejen, sagde han at vi skulle have mere frie tøjler, til at lave en hjemmeside, som vi syntes kunne være bedst, og så ville han bedømme den, i forhold til sin egen. Vi begyndte også på User Stories.

Sprint 1

Sprint 1 - 2 uger:

Sprint Backlog - Rasmus

Sprint 1 varer 2 uger, fra den 22/4-2025 til den 4/5-2025. Vi skal have lavet Product Backlog. Vi skal lave et udskrift af alle diagrammerne, Business Analysis, Business Modeling og, analyse og design. Vi skal også have lavet Daily Scrum og Burndown Chart. Vi skal også have sat Github, Simply og Trello op, så det er klart til, når vi skal programmere. Vi skal også have lavet SWOT Analyse.

Daily Scrum - Rasmus

Dag 1 - 22/4-2025:

- ❖ I dag skal vi fokusere på product backlog, her skal vi skrive user stories og acceptance criteria. Vi skal skrive story points og prioritere user stories med business value.

Dag 2 - 23/4-2025:

- ❖ I går lavet vi Product Backlog sammen, samt diagrammer, så der var ikke så meget individualitet. Vi hyggesnakkede lidt for meget, så det er noget vi skal gøre lidt mindre af. I dag skal vi køre videre med Product Backlog og "færdiggøre" den. Samtidig skal vi have lavet diagrammerne færdigt og gå i gang med Business Model.

Dag 3 - 24/4-2025:

- ❖ I går fik vi lavet Product Backlog færdigt. Vi fik lavet et Domain Model og Class Diagram udkast. I dag skal vi have lavet et udkast til et Sequens Diagram. Samtidig skal vi gå i gang med Business Model og Business Analysis, samt beskrive hvad vi har lavet/laver og hvorfor.

Dag 4 25/4-2025:

- ❖ I går fik vi lavet UML diagrammer færdigt, som alle arbejdede på, samt lavet på BMC, lavet af Kasper og Rasmus, VCA, lavet af Martin, og SWOT, lavet af Oliver. Der var ingen forhindringer. I dag skal alt nævnt færdiggøres.

Dag 5 28/4-2025:

- ❖ Sidst fik vi lavet BMC færdigt. Dog har vi nogle spørgsmål til Product Owner, så den kan blive ændret. Vi havde også nogle problemer med value chain analysis(VCA), hvor vi har nogle spørgsmål til product owner. Her skal vi så aftale et møde med ham. I dag skal vi have lavet på Razor Page Class Diagram. Vi skal også have sat GitHub og Trello op.

Dag 6 29/4-2025:

- ❖ Vi fik lavet på razor page class diagrammet, dog har vi nogle spørgsmål, som vi vil have svar på fra Henrik i dag. Ellers fik vi sat GitHub og Trello op. I dag skal vi begynde at tilføje Class'er, lave videre på Razor Page Class Diagram og få oprettet Simply.

Dag 7 30/4-2025:

- ❖ I går havde vi problemer med GitHub, så Martin fik lavet et nyt GitHub. Oliver blev færdig med SQL og fik lavet Class'er i projektet. I dag begynder alle at kode. Diagrammer bliver ændret lidt på, da en User class bliver tilføjet.

Dag 8 1/5-2025:

- ❖ Oliver fik lavet en database, samt ændret i sql til mysql. Martin fik begyndt på Order processen, Kasper fik begyndt på Car processen og lavet lidt på layout, og Rasmus fik begyndt på User processen, hvor der var et problem med noget af det Martin havde lagt op og at Rasmus ikke kunne få det ned. I dag skal vi fortsætte med programmering. Vi skal lave mockup, samt fortsætte med opbygning af layout.

Dag 9 2/5-2025:

- ❖ Kasper og Oliver begyndte på layout, ved at lave muckups på hvordan hjemmesiden skal se ud, samt skrev code til det. Martin færdiggjorde alle order class'es. Rasmus lavede videre på alle User class'es. I dag skal vi alle skrive dokumentation på alt det vi har lavet indtil videre.

Sprint Review (Rasmus)

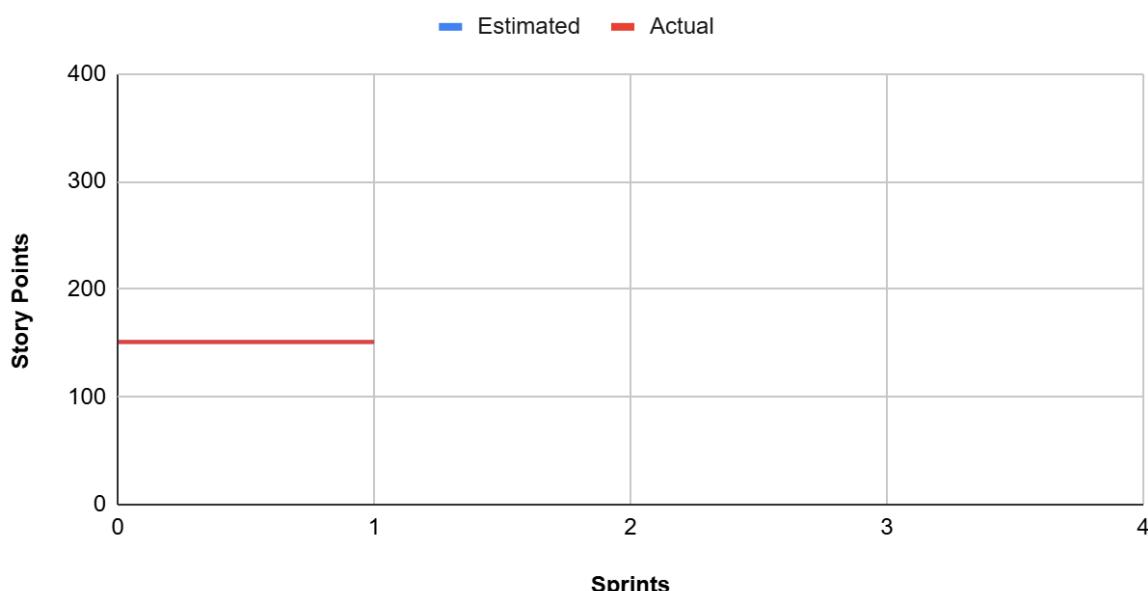
Vi startede med at lave Product Backlog, User Stories, Acceptance Criteria, Story Points og Business Points. Efterfølgende lavede vi Domain Model, Class Diagram, Sequence Diagram, BMC, Razor Page Class Diagram, VCA, SWOT og ER Diagram. Vi satte så Github, Trello og Simply op. Der blev også sat Database op.

Vi er noget længere end vi regnede med, da vi er begyndt på at kode før tid, hvor vi kastede os over nogle User Stories. Dog har vi ikke noget alt vi satte os for at gøre, da vi begyndte sent, at dokumentere det vi har lavet. Samtidig er der nogle diagrammer, vi regnede med at lave færdigt, som vi ikke har nået.

Sprint Retrospective (Rasmus)

Vi skal skrive dokumentation efter vi har færdiggjort noget i det kommende sprint, i forhold til sidste sprint, hvor vi begyndte på det hele til sidst, hvilket vi gerne vil undgå, da der bliver for meget. Vi skal også blive bedre til at bruge Trello.

BurnDown Chart



Sprint 2 - Rasmus

Sprint 2 - 1 uge:

Sprint Backlog

Vi skal blive færdig med BMC og VCA. Vi skal opdatere alle andre diagrammer. Vi skal skrive for første sprint i BurnDown Chart og Sprint Backlog(Trello). Vi skal også fortsætte med at kode. Vi skal snakke med Product Owner.

Daily Scrum - Rasmus

Dag 10 - 5/5-2025:

- ❖ Vi fik lavet på dokumentationen af første sprint sidst. I dag skal vi lave Sprint Review, Sprint Retrospective og Sprint Planning for Sprint 2. Vi fortsætter så med dokumentation af første sprint. Vi skal også opdatere Trello og lave på BurnDownChart. Til sidst i dag skal vi mødes med Product Owner klokken 14 og snakke med ham omkring det vi har lavet indtil videre, vores diagrammer, samt har vi spørgsmål til ham omkring vores forretnings diagrammer.

Dag 11 - 6/5-2025:

- ❖ Vi fik snakket med Product Owner, hvilket var meget produktivt i forhold til vores diagrammer. Vi skrev nogle nye User Stories, da vi indså at vi manglede nogle. Dem skal vi have færdiggjort i dag. Vi skal have skrevet dokumentation på det vi har lavet indtil videre færdigt i dag, samt få lavet BurnDown Chart, da vi ikke nåede det i går.
- ❖ Vi snakkede med Product Owner om vores BMC og VCA, han gav os noget feedback på dem. Den feedback handlede om nogle af vores beskrivelser og fortolkninger af hans forretning, ud fra hans feedback udarbejdede vi nogle ændringer på både BMC og VCA, så de passede bedre i hans syn af firmaet.

Dag 12 7/5-2025:

- ❖ Vi fik lavet User Stories færdigt, skrevet Acceptance Criteria til det, Story Points og Business Points. Da vi skulle lave BurnDown Chart, indså vi at vi ikke havde færdiggjort nogle User Stories, hvilket gjorde at vi ikke kunne vise noget på BurnDown Charted. I dag skal vi færdiggøre dokumentation. Der bliver lavet videre på Front-end.

Dag 13 08/05-2025:

- ❖ Kasper står for dokumentation for Mock-ups. Sidste dokumentation bliver skrevet af kode, af det vi har lavet, af Rasmus og Oliver. Martin færdiggøre VCA dokumentation. Hvis vi er færdig før tid, skriver Martin OrderService dokumentation på. Oliver laver videre front-end for cars car.cshtml.cs. Rasmus laver på DbService.

Dag 14 09/05-2025:

- ❖ I går fik vi færdiggjort dokumentation på hvad vi har lavet indtil videre. I dag skal vi kører videre med at kode. Martin laver Order processen. Rasmus laver videre på User. Oliver og Kasper laver front-end for salg, køb og leasing af biler.

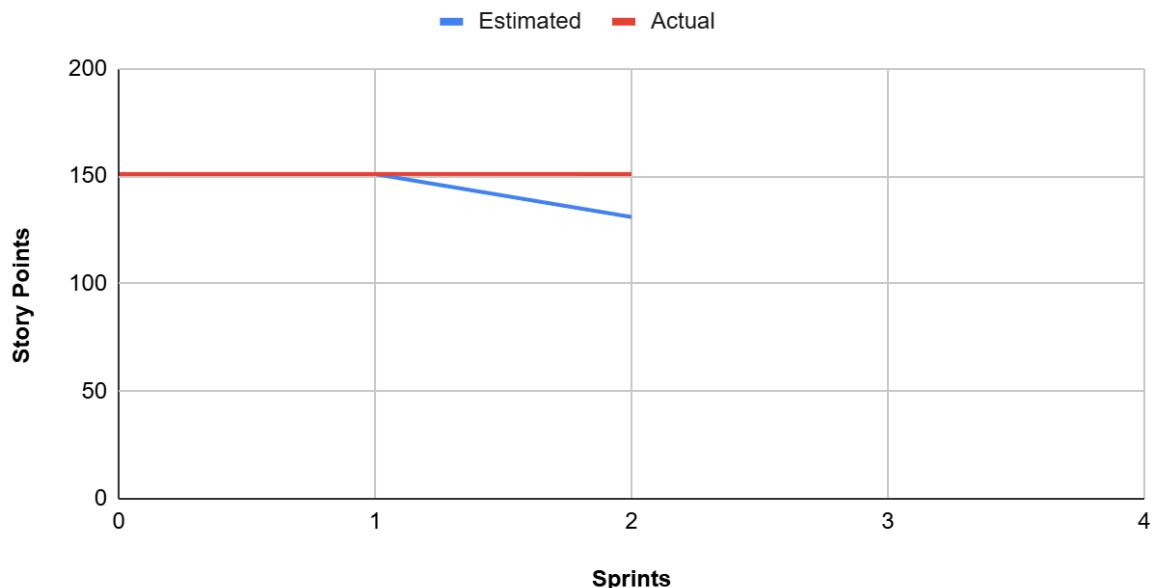
Sprint Review - Rasmus

I den foregående periode har vi videreudviklet fundamentet, som fører til en solid løsning på problemstillingen. Der er ikke blevet løst mange story points, men der er mange User Stories som er nær færdiggjorte. Vi har fået dokumenteret meget af koden og diagrammerne.

Sprint Retrospective - Rasmus

Vi har nået meget af det vi satte os for, dog fik vi ikke opdateret diagrammerne, hvilket vi er nødt til at få gjort i Sprint 3.

BurnDown Chart



Sprint 3

Sprint 3 - 1 uge:

Sprint Backlog - Rasmus

Vi regner med at få lavet Order, User og Car klasserne færdige, hvor der også er blevet lavet et link til databasen. Vi skal have opdateret diagrammerne. Vi skal dokumentere koden og diagrammerne. Vi skal have opdateret Sprint Backlog(trello).

Daily Scrum - Rasmus

Dag 15 12/05-2025:

- ❖ Martin arbejder videre med ordre user stories, så tingene bliver implementeret i vores programmering. Oliver og Kasper laver CarService, og forbindelsen mellem DB og Car. Rasmus laver videre på User.

Dag 16 13/05-2025:

- ❖ Martin laver videre på ordre user stories, og retter fejl i programmeringen. Rasmus laver videre på user, og retter fejl i programmeringen, samt dokumentation. Oliver og Kasper laver videre på forbindelsen mellem klassen "Car" og DB.

Dag 17 14/05-2025

- ❖ Kasper laver front-end. Oliver laver back-end for Car. Rasmus laver videre på User. Martin laver videre på Order.

Dag 18 15/05-2025

- ❖ Rasmus havde nogle problemer med EditUser, men fandt en løsning. Martin fortsætter på Order, hvor han begynder på Razor Page. Rasmus fortsætter på User og kigger på Admin. Oliver laver videre på Car. Kasper laver videre på front end, så vi kan få implementeret Car i Razor Page.

Dag 19 16-05-2025

- ❖ Rasmus har haft problemer med at gøre Employee til Admin, så kun de kan se GetAllUsers siden. Det bliver committed og de andre kigger på det. Vi laver diagrammer og trello. Når man er færdig med det, kan man lave videre på kode.

Sprint Review - Rasmus

Vi har i det foregående sprint, fået dokumenteret koden der er blevet lavet. Vi har lavet videre på Order, User og Car. Vi har fået lavet front-end for hjemmesiden, så det står bedre.

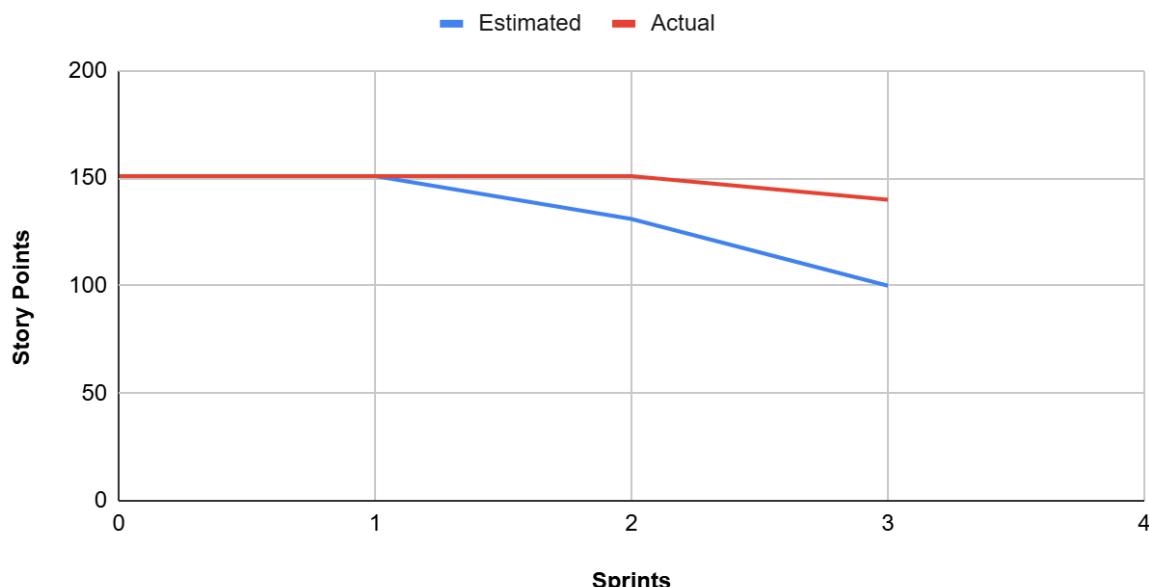
Sprint Retrospective - Rasmus

Vi skal være bedre til at administrere vores tid, i forhold til at vi skal være bedre til at få lavet det vi sætter os for noget hurtigere.

Increment (Kasper)

Vi har tilføjet en "Log In" funktion, som gør det muligt at tilgå Ordre på hjemmesiden. Samt har vi fået lavet en footer med de essentielle oplysninger som åbningstider, og informationer om virksomheden. Vi har implementeret og testet "CreateCar" med succes, og man kan se informationer om bilen. Vi har også testet og færdiggjort "Sælg Bil".

BurnDown Chart



Sprint 4

Sprint 4 - 1,5 uge:

Sprint Backlog - Rasmus

Vi skal lave dokumentation på kode og alle diagrammer. Vi skal lave Razor Pages færdigt, front-end og back-end. Her skal vi lave Order, User og Car færdigt. Vi skal lave burndown chart færdigt. Vi skal have gennemgået ALLE user stories, og være sikker på at de alle bliver opfyldt, eller dokumentere/argumentere for hvorfor nogle ikke er blevet implementeret.

Daily Scrum - Rasmus

Dag 20 19/05-2025

- ❖ Martin er syg i dag. Rasmus skal lave videre på User, hvor Search skal laves færdigt. Hvis det kan nås, bliver der lavet på CreateUser. Oliver og Kasper laver videre på Car, hvor de laver forbindelsen mellem car og database.

Dag 21 20/05-2025

- ❖ Kasper og Oliver arbejder videre på sammenhængen mellem class "Car" og Razor Page. Rasmus arbejder videre med class "User". Martin fortsætter med "Order", så det bare spiller.

Dag 22 21/05-2025

- ❖ I går fik vi skabt forbindelse til domænet. Kasper kigger på Om os og front-end. Oliver tester for problemer og dokumentation. Rasmus kigger på CreateUser, og hvis det kan nås, kigger på Authenticate Employee, så kun en medarbejder kan komme ind på GetAllUsers, DeleteCar, CreateCar og EditCar. Martin laver videre på Order.

Dag 23 22/05-2025

- ❖ Martin mangler meget lidt med Order. Når han er færdig, begynder han at teste med data. Kasper laver på LeasingCar og front-end. Oliver laver ENV.

Dag 24 23/05-2025

- ❖ Kasper og Oliver laver ImageService for Car, samt front-end på leasing og andet relateret til Car. Martin laver Front-end for Order, samt test af metoder. Hvis han når at blive færdig, hjælper han os andre. Rasmus tilslutter User til Databasen.

Dag 25 26/05-2025

- ❖ Der var nogle problemer med EditUser, så det blev lavet to separate klasser, EditEmployee og EditCustomer. Der var også lidt problemer med Hashedpassword, men det er fikset. Rasmus skriver dokumentation på resten af User. Martin færdiggørende Order. Kasper og Oliver færdiggørende Image.
- ❖ I vores sidste møde med Product Owner, klokken 8:30, vidste vi ham det af hjemmesiden vi har fået. Det var svært for ham at kommentere på køb- og leasing af bil, da programmet i var helt færdigt. Han kom med kommentarer for hvordan man eventuelt kunne videreudvikle den. Ellers var han tilfreds med hvad vi havde opnået.
- ❖ Product owner snakkede om nogle ting han godt kunne tænke sig at se hvis vi videreudviklede på programmet. Det var primært en funktion hvor en kunde kunne specificere specifikke KM tal osv. andre ting var meget omkring layout.

Dag 26 27/05-2025

- ❖ I dag fokuserer vi alle på rapporten. Der skrives dokumentation for kode og diagrammer. Burndown Chart bliver lavet færdigt. Diagrammerne bliver også lavet færdigt. Rapporten bliver rettet igennem. Hvis der er tid, bliver der kigget på kode.

Dag 27 28/05-2025

- ❖ I dag laver alle rapport. Kode dokumentation skrives færdig. Laver diagrammer færdigt og skriver dokumentation på dem. Da vi fik lidt mere tid, valgte vi at vente med Burndown Chart til i dag. Rapporten rettes igennem. Der laves Sprint Review, Sprint Retrospective og Increment.

Dag 28 29/05-2025

- ❖ I dag laves Burndown Chart færdigt. Vi skriver på, hvad der kan videreudvikles med, og refleksion og konklusion på projektet. Der skrives Sprint Review, Sprint Retrospective og Increment. Vi retter Razor Page diagrammet færdigt og skriver dokumentation.

Dag 29 30/05-2025 (DEADLINE)

- ❖

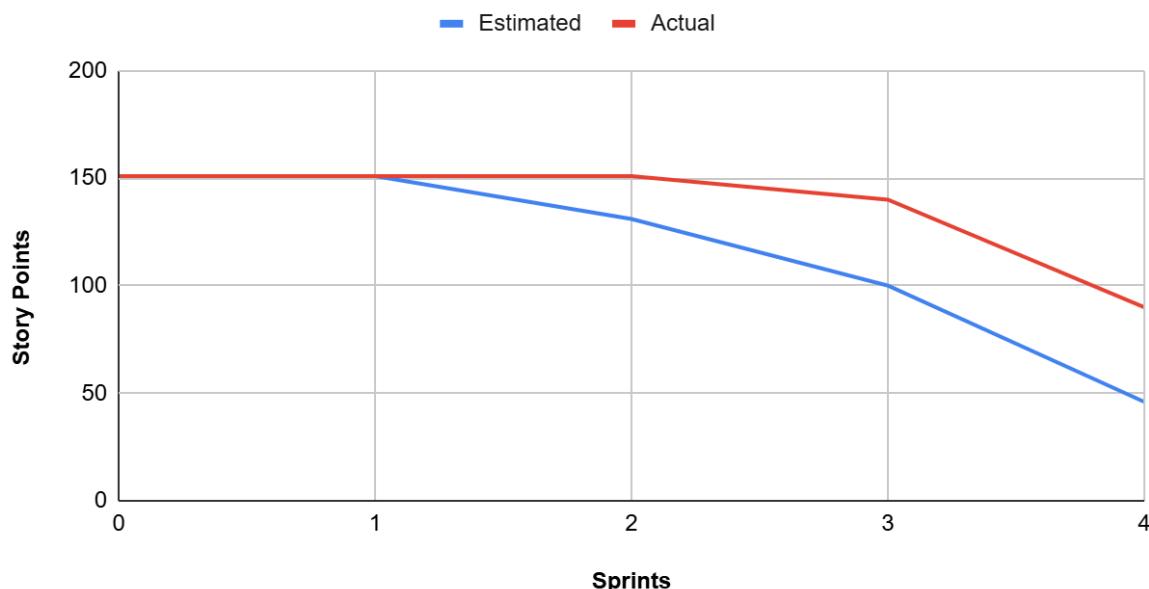
Sprint Review - Rasmus

Vi har i dette sprint fået lavet flere User Stories færdigt, da der var mange der var igang på samme tid. Vi har fået skrevet en masse dokumentation på kode og diagrammer. Vi har fået skabt et link til databasen med User, Car og Order. Vi har færdiggjort vores rapport.

Sprint Retrospective - Kasper

Nu når projektet er "færdigt", kan vi se ting som skal færdigudvikles, og der er noget vi vil fokusere på skal være færdigt, inden vi skal forsøre vores projekt. Meget af vores tid i det sidste sprint er gået til rapporten, og vi vil efter afleveringen få lavet så meget på hjemmesiden som muligt, så det kører som det skal. Vi er selv tilfredse med arbejdsindsatsen i sidste sprint.

BurnDown Chart



Increment - Kasper

- "User" er lavet færdigt og testet. Alt funktionalitet på "Ordre" virker, men ikke færdig implementeret på siden. "CreateCar" bliver videreudviklet, så der kan tilføjes et billede til bilen.

Kode Dokumentation

Car Kasper/Oliver

Car Class:

- ❖ Den har en masse properties som er information om en bil. Den har en default constructor og en constructor der initialiserer alle properties. Vi bruger [Required] til alle vores properties. Den angiver, at en bestemt egenskab (property) **skal udfyldes** den må ikke være null, tom eller manglende. Den bruges især i modelklasser til at sikre, at brugeren har angivet nødvendige oplysninger, før dataene gemmes eller behandles. Samt bruges [StringLength] også til mange af vores properties. StringLength er den som angiver hvor lang en streng må være – og evt. minimumslængde også.

```
[Table("Car")]
57 references
public class Car
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    /// <summary>
    /// Properties of the Car class.
    /// </summary>
    [Required]
    18 references
    public int Id { get; set; }
    [Required]
    [StringLength(40)]
    13 references
    public string TypeOfCar { get; set; }
    [Required]
    [StringLength(40)]
    17 references
    public string Brand { get; set; }
    [Required]
    [StringLength(40)]
    15 references
    public string Model { get; set; }
```

En **constructor** er en funktion, som **kører automatisk**, når et objekt af en klasse oprettes.

```
// Constructor til KØB BIL
2 references
public Car(int id, string type, string brand, string model, string color, string fuel, int modelYear,
{
    Id = id;
    TypeOfCar = type;
    Brand = brand;
    Model = model;
    Color = color;
    Price = price;
    Fuel = fuel;
    ModelYear = modelYear;
    Price = price;
    Mileage = mileage;
    KmPrL = kmPrL;
    TopSpeed = topSpeed;
    Doors = doors;
    HorsePower = horsePower;
    Gear = gear;
    Cylinders = cylinders;
    MotorSize = motorSize;
    ZeroToOneHundred = zeroToOneHundred;
    Length = length;
    NumOffWheels = numOffWheels;
    MaxPull = maxPull;
    Weight = weight;
    Status = status;
}

}
```

ICarService Interface:

- ❖ Dette interface forud bestemmer, hvilke metoder, klasser der arver fra den skal implementeres. Dette inkluderer CRUD, filter og search metoder.

```
14 references
public interface ICarService
{
    /// <summary>
    /// Methods that any class implementing this interface must implement.
    /// </summary>
    1 reference
    public List<Car> GetCars();
    1 reference
    public List<Car> GetBuyCars();
    2 references
    public List<Car> GetLeasingCars();
    2 references
    public void AddCar(Car car);
    3 references
    public void UpdateCar(Car car);
    11 references
    public Car GetCar(int id);
    2 references
    public Car DeleteCar(int id);
    2 references
    public Task<List<Car>> SearchCarsAsync(string? searchTerm, string? brand, string? fuel, int? maxPrice, int? maxKm, List<string?>? selectedTypes, bool? forSaleOnly);
}
```

CarService class:

- ❖ Dette benytter interfacet IcarService som forud bestemmer, hvilke metoder, der skal implementeres. CarService indeholder en liste kaldet _cars og en DBCarService _dBCarSercie

```

3 references
public class CarService: ICarService
{
    /// <summary>
    /// List of cars in the car service.
    /// </summary>
    private List<Car> _cars;

    //----- In test
    private DBCarService _dbcarservice;

    0 references
    public CarService(DBCarService dbCarService)
    {
        _dbcarservice = dbCarService;
        // _items = MockItems.GetMockItems();
        // _items = _jsonFileItemService.GetJsonObjects().ToList();
        //_dbService.SaveObjects(_items);
        _cars = _dbcarservice.GetCars().Result.ToList();
    }

    0 references
    public CarService()
    {
    }
}

```

DBCarService class:

- ❖ DBCarService bruges som en service som håndterer kommunikationen mellem controller og databasen. En controller er en klasse som håndtere HTTP-forespørgsler, og styrer hvordan applikationen skal reagere.
Vores DBCarService intereagerer med databasens (CRUD) samt henter og gemmer data på Car.
- ❖ Kommunikationen mellem DBCarService, CarDbContext og CarService spiller sammen på følgende måde:
 - ❖ **DBCarContext giver adgang til databasen, ved brug af CarService**
 - ❖ **CarService giver adgang til databasekald, ved brug af CarController**
 - ❖ **CarController modtager HTTP-anmodninger, ved brug af CarService.**

```
4 references
public class DBCarService
{
    private readonly CarDbContext _context;

    0 references
    public DBCarService(CarDbContext context)
    {
        _context = context;
    }

    1 reference
    public async Task<List<Car>> GetCars()
    {
        return await _context.Cars.ToListAsync();
    }

    1 reference
    public async Task<List<Car>> GetBuyCars()
    {
        return await _context.Cars
            .Where(car => car.ForSale)
            .ToListAsync();
    }

    1 reference
    public async Task<List<Car>> GetLeasingCars()
    {
        return await _context.Cars
            .Where(car => !car.ForSale)
            .ToListAsync();
    }
}
```

CarDbContext.cs:

Denne kode er forbindelsen til databasen, hvor denne arbejder med Cars. der benyttes Entity Framework Core til at gøre det nemt at arbejde med database i C#.

IConfiguration er et interface, vi bruger til at læse opsætninger fra appsettings.json hvor connection stringen er.

når der skal oprettes forbindelse til databasen så kaldes OnConfiguring, hvor der på linje 16 hentes forbindelsensstrengen, ved at sætte en variabel connectionstring lig med "DefaultConnection" som hentes fra appsettings.json ved at kalde GetConnectionString("DefaultConnection") på _configuration.

på linje 17 siger vi at den skal bruge MySQL som database og at den selv skal finde ud af version og den skal bruge variablen fra ovenover.

linje 19 fortælles, at der findes en tabel i databasen med biler, og at vi vil kunne arbejde med den som en liste af car-objekter.

```
1 using AutoMaegler.Models;
2     using Microsoft.EntityFrameworkCore;
3     using Microsoft.Extensions.Configuration;
4
5     namespace AutoMaegler.EFDbContext
6     {
7         4 references | OKronborg00, 2 days ago | 2 authors, 2 changes
8         public class CarDbContext : DbContext
9         {
10             0 references | Agentblank66, 4 days ago | 1 author, 1 change
11             private readonly IConfiguration _configuration;
12             public CarDbContext(IConfiguration configuration)
13             {
14                 _configuration = configuration;
15             }
16             0 references | Agentblank66, 4 days ago | 1 author, 1 change
17             protected override void OnConfiguring(DbContextOptionsBuilder options)
18             {
19                 var connectionString = _configuration.GetConnectionString("DefaultConnection");
20                 options.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString));
21             }
22         }
23     }
```

BuyCar:

BuyCar er en klasse eller en Razor Page, der håndterer oprettelsen af købstransaktioner for biler. Den giver brugeren mulighed for at vælge en bil og udfylde de nødvendige oplysninger for at gennemføre et køb. Funktionaliteten omfatter validering, databehandling og overførsel til databasen via relevante services.

Car.cshtml.cs:

Car.cshtml.cs er en kode-behind-fil for en Razor Page, som i sin nuværende form kun indeholder en tom `OnGet()`-metode. Det betyder, at siden ikke udfører nogen logik ved indlæsning, men er forberedt til at håndtere visning eller redigering af bilrelaterede data.

Cars.cshtml.cs:

Cars.cshtml.cs er kode-behind-filen for Razor Page-siden, der viser en oversigt over biler. Den indeholder aktuelt kun en tom `OnGet()`-metode og er forberedt til senere implementering, hvor biler kan hentes og vises, fx fra CarService.

CreateCar:

CreateCar er en Razor Page, hvor brugeren kan oprette en ny bil i systemet. Siden indeholder en formular til indtastning af informationer som mærke, model, årgang og pris. Data bindes til en bilmodel og valideres, inden den nye bil tilføjes databasen gennem CarService.

DeleteCar:

DeleteCar er en Razor Page, der giver brugeren mulighed for at slette en eksisterende bil. Den viser bilens informationer og spørger brugeren, om de vil bekræfte sletningen. Når bekræftelsen er givet, slettes bilen fra databasen via CarService og DBCarService.

EditCar:

EditCar er en Razor Page, hvor brugeren kan redigere oplysninger om en eksisterende bil. Siden viser en formular med nuværende værdier, som brugeren kan opdatere. Når ændringerne gemmes, opdateres bilobjektet i databasen via CarService.

LeasingCar:

LeasingCar er en klasse eller Razor Page, der håndterer leasing af biler. Den fungerer tilsvarende BuyCar, men indeholder felter og logik specifikt for leasingaftaler, såsom leasingperiode og månedlig ydelse. Data indsamles, valideres og gemmes via de relevante services.

Order (Martin)

Order Class:

- ❖ Order-klassen er en abstrakt basisklasse, der modellerer en generel ordre i systemet, såsom leasing, køb eller salg af en bil. Klassen definerer fælles egenskaber såsom ordretype, ordre-ID, bil, medarbejder og kunde. Bruger data annoteringer. Klassen indeholder en konstruktør til at initialisere disse egenskaber og implementere sammenligning logik baseret på ordre-ID. Specifikke ordretyper arver fra denne klasse for at tilføje mere specialiseret funktionalitet.

```
[Display(Name = "En kunde")]
[Required(ErrorMessage = "Der skal være en kunde tilknyttet ordren")]
25 references | Agentblank66, 3 days ago | 1 author, 1 change
public Customer Customer { get; set; }

/// <summary>
/// Constructor for the Order class.
/// </summary>
/// <param name="id"></param>
/// <param name="car"></param>
/// <param name="employee"></param>
/// <param name="customer"></param>
/// <param name="type"></param>
4 references | Agentblank66, 3 days ago | 1 author, 1 change
public Order(int id, Car car, Employee employee, Customer customer, OrderType type)
{
    Id = id;
    Car = car;
    Employee = employee;
    Customer = customer;
    Type = type;
}
```

OrderSale Class:

- ❖ Denne klasse er en subklasse af Order klassen, hvilket betyder den anvender det som der er i superklassen Order, men også tilføjer sine egne properties hvor det initialiseres i constructoren.

```
public OrderSale(int id, Car car, Employee employee, Customer customer, OrderType type, double salePrice, int year, int month, int day)
    : base(id, car, employee, customer, type)
{
    SalePrice = salePrice;
    SaleDate = new DateTime(year, month, day);
}

/// <summary>
/// Default constructor for the OrderSale class, initializing with default values.
/// </summary>
1 reference | Agentblank66, 11 hours ago | 1 author, 1 change
public OrderSale() : base(0,new Car(), new Employee(), new Customer(),OrderType.Sale )
{
    SalePrice = 0.0;
    SaleDate = DateTime.Now;
    Year = DateTime.Now.Year;
    Month = DateTime.Now.Month;
    Day = DateTime.Now.Day;
}
```

OrderLeasing Class:

- ❖ Dette er en subklasse af Order klassen, den bruger alt fra dens superklasse, mens den også har sine egne properties som bliver initialiseret i constructoren.

```

public OrderLeasing(int id, Car car, Employee employee, Customer customer, OrderType type, double depositum, int startYear, int endYear, int startMonth, int endMonth, int startDay, int endDay, double monthlyPayment)
{
    Depositum = depositum;
    LeasingDateStart = new DateTime(startYear, startMonth, startDay);
    MonthlyPayment = monthlyPayment;
    LeasingDateEnd = new DateTime(endYear, endMonth, endDay);
}

1 reference | 0 authors, 0 changes
public OrderLeasing() : base(0,new Car(), new Employee(), new Customer(), OrderType.Leasing)
{
    Depositum = 0.0;
    LeasingDateStart = DateTime.Now;
    LeasingDateEnd = DateTime.Now.AddYears(1);
    MonthlyPayment = 0.0;
    StartYear = DateTime.Now.Year;
    EndYear = DateTime.Now.Year + 1;
    StartMonth = DateTime.Now.Month;
    EndMonth = DateTime.Now.Month + 1;
    StartDay = DateTime.Now.Day;
    EndDay = DateTime.Now.Day;
}

```

OrderBuy Class:

- ❖ Dette er en subklasse af Order klassen, den bruger alt fra dens superklasse, mens den også har sine egne properties som bliver initialiseret i constructoren.

```

public OrderSale(int id, Car car, Employee employee, Customer customer, OrderType type, double salePrice, int year, int month, int day)
: base(id, car, employee, customer, type)
{
    SalePrice = salePrice;
    SaleDate = new DateTime(year, month, day);
}

/// <summary>
/// Default constructor for the OrderSale class, initializing with default values.
/// </summary>
1 reference | Agentblank66, 11 hours ago | 1 author, 1 change
public OrderSale() : base(0,new Car(), new Employee(), new Customer(), OrderType.Sale )
{
    SalePrice = 0.0;
    SaleDate = DateTime.Now;
    Year = DateTime.Now.Year;
    Month = DateTime.Now.Month;
    Day = DateTime.Now.Day;
}

```

IOrderService:

- ❖ IOrderService er et interface, der definerer de metoder, som enhver ordreservice klasse skal implementere i systemet. Det sikrer, at alle klasser, der håndterer order, følger den samme struktur og funktionalitet.

```

void UpdateOrder<T>(T order) where T : Order;
5 references | Agentblank66, 3 days ago | 1 author, 1 change
Order GetOrder(int id, Order.OrderType type);
2 references | Agentblank66, 3 days ago | 1 author, 1 change
T DeleteOrder<T>(int id, Order.OrderType type) where T : Order;
2 references | Agentblank66, 21 hours ago | 1 author, 1 change
IEnumerable<T> NameSearch<T>(string str) where T : Order;
2 references | Agentblank66, 3 days ago | 1 author, 1 change
IEnumerable<Order> PriceFilter(double minPrice, double maxPrice, double minLeasing, double maxLeasing);

```

OrderService:

- ❖ OrderService klassen er en serviceklasse, der håndterer alle operationer omkring ordrer i systemet. Den arbejder med lister som er forskellige ordretyper (leasing, køb og salg) og implementere metoder til at hente, tilføje, opdatere, slette, søge og sortere ordrer. Klassen kan både arbejde med mock data og med en databasen via DbOrderService, så ændringer gemmes i.

```

10 references | Agentblank66, 3 days ago | 1 author, 1 change
public List<Order> _orders { get; set; }
4 references | Agentblank66, 3 days ago | 1 author, 1 change
public List<OrderBuy> _orderBuys { get; set; }
4 references | Agentblank66, 3 days ago | 1 author, 1 change
public List<OrderLeasing> _orderLeasings { get; set; }
4 references | Agentblank66, 3 days ago | 1 author, 1 change
public List<OrderSale> _orderSales { get; set; }
5 references | Agentblank66, 3 days ago | 1 author, 1 change
public DbOrderService _dbOrderService { get; set; }

/// <summary>
/// Constructor for the OrderService class.
/// </summary>
0 references | Agentblank66, 3 days ago | 1 author, 1 change
public OrderService(DbOrderService dbOrderService)
{
    _dbOrderService = dbOrderService;
    _orders = MockOrders.GetMockOrders();
    // _orders = _dbOrderService.GetOrders<Order>().Result;
    _dbOrderService.SaveOrder(_orders);
}

0 references | Agentblank66, 3 days ago | 1 author, 1 change
public OrderService()
{
    _orders = MockOrders.GetMockOrders();
}

```

DBOrderService:

- ❖ DbOrderService-klassen fungerer som bindeledd mellem applikationen og databasen, når det gælder ordrer. Den tilbyder metoder til at hente, tilføje og gemme ordrer i databasen, så data kan lagres og genfindes på tværs af sessioner. Klassen bruges af OrderService til at sikre, at ændringer i ordrer ikke kun sker i hukommelsen, men også bliver gemt permanent i databasen. Dermed håndterer DbOrderService kommunikation og synkronisering af ordredata mellem applikationen og databasen.

```

public async Task AddOrder<T>(T order) where T : Order
{
    using (var context = new OrderDbContext())
    {
        context.Set<T>().Add(order);
        await context.SaveChangesAsync();
    }
}

/// <summary>
/// The method that saves the changes to an order from the database
/// </summary>
/// <param name="orders"></param>
/// <returns> Saves a list of orders in the database and saves the changes </returns>
3 references | Agentblank66, 3 days ago | 1 author, 1 change
public async Task SaveOrder<T>(List<T> orders) where T : Order
{
    using (var context = new OrderDbContext())
    {
        context.Set<T>().AddRange(orders);
        await context.SaveChangesAsync();
    }
}

```

CreateOrder.cshtml:

- ❖ Er en Razor Page, der indeholder formularen til at oprette en ny ordre. Siden viser forskellige input felter afhængigt af, hvilken type ordre brugeren vælger (leasing, køb eller salg). Formularen bruger data binding og validering for at sikre, at alle nødvendige oplysninger indtastes korrekt.

```

<h1> Opret Ordre </h1>

<div class="form-group">
    <label asp-for="SelectedOrderType" class="control-label"></label>
    <select asp-for="SelectedOrderType" class="form-control" asp-items="Model.OrderTypes"></select>
    <span asp-validation-for="SelectedOrderType" class="text-danger"></span>
</div>

<form method="post">
    @if (Model.SelectedOrderType == Models.Order.OrderType.Leasing)
    {
        <div class="form-group">
            <label asp-for="Depositum" class="control-label"></label>
            <input asp-for="Depositum" class="form-control" />
            <span asp-validation-for="Depositum" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="startYear" class="control-label"></label>
            <input asp-for="startYear" class="form-control" />
            <span asp-validation-for="startYear" class="text-danger"></span>
        </div>
    }
</form>

```

CreateOrder.cshtml.cs:

- ❖ Er koden bag filen, der håndterer logikken bag oprettelsen af en ordre. Her behandles brugerens input, valideres data, og der oprettes en ny ordre baseret på de indtastede oplysninger. Kode-behind-filen sørger også for at kommunikere med services, så orden gemmes korrekt i systemet.

```

[BindProperty]
5 references | Agentblank66, 22 hours ago | 1 author, 1 change
public int SaleDay { get; set; }

/// <summary>
/// Initializes the page with the available order types for selection.
/// </summary>
/// <returns> The current page </returns>
0 references | Agentblank66, 22 hours ago | 1 author, 2 changes
public IActionResult OnGet()
{
    OrderTypes = Enum.GetValues(typeof(Models.Order.OrderType))
        .Cast<Models.Order.OrderType>()
        .Select(ot => new SelectListItem
    {
        Value = ot.ToString(),
        Text = ot.ToString()
    });

    return Page();
}

```

GetAllOrders.cshtml:

- ❖ Er en Razor Page, der viser en oversigt over alle ordrer i systemet. Siden indeholder funktionalitet til at søge efter ordrer baseret på kundens navn, filtrere ordrer efter pris og leasingpris, samt sortere ordrer efter ID, navn eller pris. Brugeren kan også oprette, redigere eller slette ordrer direkte fra oversigten via knapper.

```

<tr class="btn-secondary active">
    <th>
        <div class="d-flex justify-content-between p-0">
            <div><h5> Id </h5></div>
            <div>
                <form method="post" class="d-inline">
                    <button type="submit" asp-page-handler="SortById" class="btn btn-link p-0" title="Sorter Via Id">
                        <i class="fa fa-angle-up"></i>
                    </button>
                    <button type="submit" asp-page-handler="SortByIdDescending" class="btn btn-link p-0" title="Sorter Via Id Faldende">
                        <i class="fa fa-angle-down"></i>
                    </button>
                </form>
            </div>
        </div>
    </th>

```

GetAllOrders.cshtml.cs:

- ❖ Er kode bag filen, der håndterer alt logikken bag visningen. Den henter ordrer fra services, håndterer søgninger, filtrering og sorterings, og sørger for at de rigtige data vises på siden. GetAllOrders modtager input fra brugeren (fx søgeord eller filter værdier), opdaterer listen af ordrer og sender dem tilbage til visningen.

```

public IActionResult OnPostNameSearch()
{
    Orders = _orderService.NameSearch<Models.Order>(SearchString).ToList();
    return Page();
}

/// <summary>
/// Filters the price of the orders
/// </summary>
/// <returns> To the same newly updated page </returns>
0 references | Agentblank66, 3 days ago | 1 author, 1 change
public IActionResult OnPostPriceFilter()
{
    Orders = _orderService.PriceFilter(MinPrice, MaxPrice, MinLeasing, MaxLeasing).ToList();
    return Page();
}

```

EditOrder.cshtml:

- ❖ Er en Razor Page, hvor brugeren kan redigere oplysninger af en eksisterende ordre. Siden viser en formular med felter for ordre ID (kun læsbar), kundens fulde navn og bilens pris. Brugeren kan ændre relevante oplysninger og gemme ændringerne.

```

<p>
    <div class="form-group">
        <input type="submit" value="Rediger Ordre" class="btn btn-primary" />
    </div>
</p>
</form>

<p>
    <a class="btn btn-primary" asp-page="/Order/GetAllOrders"> Tilbage </a>
</p>

```

EditOrder.cshtml.cs:

- ❖ Er kode-behind-filen, der håndterer logikken for at hente den valgte ordre, vise dens nuværende data i formularen og opdatere orden, når brugeren indsender ændringerne. Her valideres input, og opdateringen gemmes via den relevante service.

```

[BindProperty]
11 references | Agentblank66, 3 days ago | 1 author, 1 change
public Models.Order Order { get; set; }

/// <summary>
/// The search string for the orders
/// </summary>
/// <param name="id"></param>
/// <param name="orderType"></param>
/// <returns> Redirects to an error page or current page </returns>
0 references | Agentblank66, 3 days ago | 1 author, 1 change
public IActionResult OnGet(int id, Models.Order.OrderType orderType)
{
    Order = _orderService.GetOrder(id, orderType);
    if (Order == null)
        return RedirectToPage("/NotFound");
    return Page();
}

```

DeleteOrder.cshtml:

- ❖ Er en Razor Page, der viser en bekræftelses dialog, hvor brugeren bliver spurgt, om de er sikre på, at de vil slette den valgte ordre. Siden viser centrale oplysninger om ordren, såsom ordrenummer, kunde og bil, og giver mulighed for at bekræfte eller annullere sletningen.

```

<h1> Slet Ordre </h1>
<div class="alert alert-danger">
    <h4> Er du sikker på at du vil slette denne ordre? </h4>
    <p> Ordrenummer: @Model.Order.Id </p>
    <p> Kunde: @Model.Order.Customer.FullName </p>
    <p> Bil: @Model.Order.Car </p>
    <form method="post">
        <button type="submit" class="btn btn-danger"> Ja </button>
        <a class="btn btn-primary" asp-page="/Order/GetAllOrders"> Nej </a>
    </form>
</div>

```

DeleteOrder.cshtml.cs:

- ❖ Er kode bag filen, der håndterer logikken bag sletningen. Den henter den relevante ordre baseret på ID, viser oplysningerne på siden, og hvis brugeren bekræfter, slettes ordren via den relevante service. Herefter omdirigeres brugeren tilbage til oversigten over alle ordrer.

```

/// <summary>
/// A method which is called when the page is posted, where it deletes an order.
/// </summary>
/// <param name="id"></param>
/// <param name="orderType"></param>
/// <returns> Redirects to error page or wanted page </returns>
0 references | Agentblank66, 3 days ago | 1 author, 1 change
public IActionResult OnPost(int id, Models.Order.OrderType orderType)
{
    Models.Order order = _orderService.DeleteOrder<Models.Order>(id, orderType);
    if (order == null)
        return RedirectToPage("/NotFound");

    return RedirectToPage("GetAllOrders");
}

```

User - Rasmus

User Class:

- ❖ Vi har valgt at User klassen skal være abstract, da vi ikke skal lave objekter af den.
- Vi har valgt disse properties, da Product Owner ikke ville have for meget information om kunderne, og Customer klassen og Employee klassen har disse properties tilfælles. Vi har lavet et Enum UserTypes, som indeholder Customer og Employee, så vi har en liste over hvilke typer brugere vi har på hjemmesiden og det gør der mere overskueligt og kode besparende. Samtidig har vi lagt flere data annotationer på propertiesende, så det kan bruges i Databasen, det der bliver vist er på dansk og at det skal bruges. Den indeholder en konstruktør som initialiserer dens egenskaber.

```
public enum UserType
{
    [Display(Name = "Kunde")]
    Customer,
    [Display(Name = "Medarbejder")]
    Employee,
}
/// <summary>
/// Properties of the User class.
/// </summary>
[NotMapped]
5 references | Rasmus Dyring Raavig, 2 days ago | 2 authors, 2 changes
public UserType UserTypes { get; set; }
[Key]
[DatabaseGenerated(DatabaseGeneratedOption.None)]
[Display(Name = "ID")]
[Required(ErrorMessage = "Der skal angives et Bruger Id.")]
24 references | Rasmus Dyring Raavig, 2 days ago | 2 authors, 2 changes
public int Id { get; set; }
```

```
public User(int id, string firstName, string lastName, string email, string password)
{
    Id = id;
    FirstName = firstName;
    LastName = lastName;
    Email = email;
    Password = password;
}
```

Employee Class:

- ❖ Employee klassen er en subklasse af User klassen, så den anvender det som der er i User klassen. Dog har Employee klassen en Type property, da der flere typer er medarbejdere. Den har en konstruktør som initialiserer dens egenskaber, samt basisklassens. I konstruktøren bliver UserType sat til at være Employee, så Enummen kan genkende klassen som en Employee. Den har en default konstruktør, for at sætte default værdier. Den har en ToString metode som overrider basisklassens metode.

```

/// <summary>
[Display(Name = "Type")]
[Required(ErrorMessage = "Der skal angives en Bruger Type.")]
11 references | Rasmus Dyring Raavig, 2 days ago | 2 authors, 2 changes
public string Type { get; set; }

/// <summary>
/// Constructor for the Employee class with default values.
/// </summary>
3 references | Rasmus Dyring Raavig, 2 days ago | 2 authors, 2 changes
public Employee(): base(0, "", "", "", "")
{
}

public Employee(int id, string firstName, string lastName, string type, string email, string password): base(id, firstName, lastName, email, password)
{
    UserTypes = UserType.Employee;
    Type = type;
}

```

Customer Class:

- ❖ Customer klassen er en subklasse af User klassen, så den anvender det som der er i User klassen. Dog har Customer klassen en PhoneNumber property, da det er information, som Product Owneren synes skulle være med. Samtidig har den en WishToSell property, som fortæller om en kunde har en bil til salg eller ikke. De bliver så initialiseret i konstruktøren, samt basisklassen. I konstruktøren bliver UserType sat til at være Customer, så Enummen kan genkende klassen som en Customer. Den har en default konstruktør, for at sætte default værdier. Den har også en ToString metode som overrider basisklassens metode.

LoginPage.cshtml.cs:

- ❖ I LoginPageModel har vi bundet to properties, UserName og Password, da det er dem der skal bruges til at logge ind med. Samt har vi en Message property, for hvis login informationerne er forkerte, og en property for UserType, da der er to typer af brugere som skal logge ind. Vi laver så dependency injection af UserService i konstruktøren. Onpost metoden går igennem listen af enten Customers eller Employees. Den tjekker om brugernavnet passer til en Customer/Employee Email. Så laver den en PasswordHasher, som validerer brugerens adgangskode. Det gøres så adgangskoden ikke gemmes rå. Den laver en liste af Claims, som hvem brugeren er, som så bliver brugt til at lave en ClaimsIdentity. Brugeren logges ind med en cookie authentication og gemmes.

```

public async Task<IActionResult> OnPost()
{
    List<Customer> customers = _userService.Customers;
    foreach (Customer customer in customers)
    {

        if (UserName == customer.Email)
        {

            //LoggedInCustomer = customer;

            var passwordHasher = new PasswordHasher<string>();

            if (passwordHasher.VerifyHashedPassword(null, customer.Password, Password) == PasswordVerificationResult.Success)
            {
                var claims = new List<Claim>
                {
                    new Claim(ClaimTypes.Email, customer.Email),
                    new Claim(ClaimTypes.Name, customer.Email),
                    new Claim(ClaimTypes.Role, "Customer")
                };

                var claimsIdentity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
                await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(claimsIdentity));
                return RedirectToPage("/Index");
            }
        }
    }
}

```

LoginPage.cshtml:

- ❖ Vi har lavet en besked, så hvis man skriver de forkerte oplysninger, kommer der en rød besked frem. Under den har vi lavet to skriveauelt, så man skriver sin email, da vi ikke synes man skulle have et navn stående, som er ens UserName, og et Password. Under det har vi et knap til siden CreateUser, hvilket skal rykkes til siden hvor alle brugere står. Der er lagt input felter ind, så UserName og Password kan skrives, så brugeren logger ind.

```

✓ <div class="container">
✓   <div style="color: #red">
✓     @Model.Message
✓   </div>
✓   <form method="post">
✓     <div class="form-group">
✓       <label asp-for="UserName" class="col-form-label text-black"> Brugernavn</label>
✓       <div>
✓         <input asp-for="UserName" />
✓       </div>
✓     </div>
✓     <div class="form-group">
✓       <label asp-for="Password" class="col-form-label text-black"> Adgangskode</label>
✓       <div>
✓         <input asp-for="Password" type="password" />
✓       </div>
✓     </div>
✓     <div class="form-group mt-1">
✓       <button class="btn btn-outline-light text-black">Log ind</button>
✓     </div>
✓   </form>
</div>

```

LogOutPage.cshtml.cs:

- ❖ Vi har lavet en OnGet() metode, som bruges til at håndtere en Get Request. Den nulstiller enten en Customer eller en Employee. await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme) logger brugeren ud, ved at fjerne cookien som bruges til godkendelse af brugeren.

```

public async Task<IActionResult> OnGet()
{
    //LogInPageModel.LoggedInCustomer = null;
    //LogInPageModel.LoggedInEmployee = null;

    await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return RedirectToPage("/Index");
}

```

CreateUser.cshtml.cs:

- ❖ Der er blevet lagt properties ind for de som Customer og Employee har tilfælles, samt dem som er unikke for dem selv. Samtidig har vi tilføjet UserService og lavet dependency injection af den i konstruktøren, og det samme med PasswordHasher. I OnPost metoden starter vi med at fjerne valideringsfejlen fra Type, hvis det er en Customer og PhoneNumber og WishToSell, hvis det er en Employee. Så tjekker den så om det er en Customer eller en Employee, hvor den så tilføjer dem via UserService metoden AddUser.
- ❖ Da vi ikke kunne få siden til at fungere, valgte vi at gøre det mere simpelt og lave to separate sider. En for at lave en Employee og en for at lave en Customer.

```

public IActionResult OnPost()
{
    if (UserType == Models.User.UserType.Customer)
    {
        ModelState.Remove(nameof(Type));
    }
    else if (UserType == Models.User.UserType.Employee)
    {
        ModelState.Remove(nameof(PhoneNumber));
        ModelState.Remove(nameof(WishToSell));
    }

    if (!ModelState.IsValid)
    {
        return Page();
    }

    if (UserType == Models.User.UserType.Customer)
    {
        _userService.AddUser(new Customer(Id, FirstName, LastName, PhoneNumber, Email, passwordHasher.HashPassword(null, Password)));
    }
    else if (UserType == Models.User.UserType.Employee)
    {
        _userService.AddUser(new Employee(Id, FirstName, LastName, Type, Email, passwordHasher.HashPassword(null, Password)));
    }

    return RedirectToPage("/Users/Admin/GetAllUsers");
}

```

CreateUser.cshtml:

- ❖ I CreateUser Razor page laver vi en dropdown, hvor metoden GetEnumSelectList tager listen af UserType og giver disse valgmuligheder. Du trykker så på en knap og herfra kommer input felterne som passer til enten en Customer eller Employee, da den tjekker om det er en Customer eller en Employee under if-statementene.

```

<div class="row">
    <div class="col-md-6">
        <form method="post" asp-page-handler="Select">
            <div class="form-group">
                <label asp-for="UserType" class="text-white">Brugertype</label>
                <select asp-for="UserType" asp-items="Html.GetEnumSelectList<Models.User.UserType>()" class="form-control">
                    <option value="">- Vælg type -</option>
                </select>
                <span asp-validation-for="UserType" class="text-danger"></span>
            </div>

            <button type="submit" class="btn btn-secondary">Vis filter</button>
        </form>

        <hr class="bg-white" />
        <form method="post">
            @if (Model.UserType == AutoMaegler.Models.User.UserType.Customer
                || Model.UserType == AutoMaegler.Models.User.UserType.Employee)
            {
                <div class="form-group">
                    <label asp-for="Id" class="control-label text-white">Id</label>
                    <input asp-for="Id" class="form-control" />
                    <span asp-validation-for="Id" class="text-danger"></span>
                </div>
            }
        </form>
    </div>

```

GetAllUsers.cshtml.cs:

- ❖ For klassen er det kun Employee, som er autoriserede til at gå ind på den, da man kan se alle brugere, hvilket Customers ikke skal kunne, samt ændre, slette og lave brugere. Vi har lavet en property af UserService og lavet dependency injection af den i konstruktøren, samt lavet properties for vores søge metode af brugerne, som er SearchedUsers og en liste af FilteredUsers. I OnPost metoden tjekkes der er noget i søgerfeltet. Den kalder metoden SearchByName, som ligger i UserService og ligger dem i listen FilteredUsers. Hvis listen er tom returnere den til GetAllUsers siden.

```

0 references | Agentblank66, 5 days ago | 1 author, 1 change
public IActionResult OnPost()
{
    if (string.IsNullOrWhiteSpace(SearchedUsers))
    {
        return RedirectToPage("/Users/Admin/GetAllUsers");
    }

    FilteredUsers = UserService.SearchByName(SearchedUsers);

    if (FilteredUsers == null)
    {
        return RedirectToPage("/Users/Admin/GetAllUsers");
    }

    return Page();
}

```

GetAllUsers.cshtml:

- ❖ Vi har valgt at man skal kunne se to lister. En af alle Employees og en af alle Customers, hvor man samtidig har muligheden for at redigere dem, slette dem eller oprette en ny af dem. Når man skal søge efter navn, kommer der en liste op af de søgte brugere. Der bliver tjekket igennem om hvilke typer brugere der er, da de har forskellige felter. Ved Edit og Delete tjekker den også om det er en Employee eller Customer med GetType metoden.

```

59     <tbody>
60         @foreach (var user in Model.FilteredUsers)
61     {
62         var isCustomer = user is Customer;
63         var isEmployee = user is Employee;
64
65         <tr class="text-black">
66             <td>@user.FirstName</td>
67             <td>@user.LastName</td>
68             <td>@user.Email</td>
69
70             <td>@(isCustomer ? ((Customer)user).PhoneNumber: "")</td>
71             <td>@(isEmployee ? ((Employee)user).Type: "")</td>
72
73             <td>
74                 <a class="btn btn-primary btn-sm" asp-page="/Users/EditUser" asp-route-id="@user.Id" asp-route-userType="@user.GetType().Name">Rediger</a>
75                 <a class="btn btn-danger btn-sm" asp-page="/Users/DeleteUser" asp-route-id="@user.Id" asp-route-userType="@user.GetType().Name">Slet</a>
76             </td>
77         </tr>
78     }
79 </tbody>

```

EditUser.cshtml:

- ❖ Der er sat inputfelter af dataen for enten en Employee eller Customer. Her har de begge et Id, som er sat fast, så det ikke kan ændres, da det er tilknyttet til databasen. Ellers er det sat op så man kan ændre på dataen fra brugeren.

```

<div class="form-group text-white">
    <label asp-for="Email" class="control-label"></label>
    <input asp-for="Email" class="form-control" />
    <span asp-validation-for="Email" class="text-danger"></span>
</div>
@if (Model.UserType == Models.User.UserType.Customer)
{
    <div class="form-group text-white">
        <label asp-for="PhoneNumber" class="control-label"></label>
        <input asp-for="PhoneNumber" class="form-control" />
        <span asp-validation-for="PhoneNumber" class="text-danger"></span>
    </div>
}

```

EditUser.cshtml.cs:

- ❖ En model klasse for siden hvor man redigere en bruger. UserService og Customers/Employees egenskaber som properties. Dependency injection af UserService i konstruktøren. Man henter brugeren med et id og hvilken UserType det er, gennem OnGet() metoden. Så når man har ændret på noget, gemmes dataen så, gennem OnPost() metoden, når man trykker på Opdater knappen.

```
0 references | Rasmus Dyring Kaavig, 2 days ago | 2 authors, 2 changes
public IActionResult OnGet(int id, UserType userType)
{
    var user = _userService.GetUser(id, userType);
    if (user == null)
        return RedirectToPage("/NotFound");

    UserType = user.UserTypes;
    Id = user.Id;
    FirstName = user.FirstName;
    LastName = user.LastName;
    Email = user.Email;
    Password = user.Password;
    if (user is Customer customer)
    {
        PhoneNumber = customer.PhoneNumber;
    }
    else if (user is Employee employee)
    {
        Type = employee.Type;
    }

    return Page();
}
```

UserDbContext:

- ❖ Her arver UserDbContext fra DbContext, så den kan interagere med databasen. Her bruger vi metoden OnConfiguring(), som bruges til at konfigurere databasen. Samtidig har vi tilføjet DbSet af Customer og Employee, som oversætter mellem klasserne og tabellerne af vores objekter.

```
0 references | Rasmus Dyring Kaavig, 2 days ago | 2 authors, 2 changes
protected override void OnConfiguring(DbContextOptionsBuilder options)
{
    var connectionString = _configuration.GetConnectionString("DefaultConnection");
    options.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));
}
```

EditEmployee/EditCustomer.cshtml.cs:

- ❖ Vi havde problemer med at få EditUser til at fungere, så vi valgte at lave to nye Edit Razor pages, EditCustomer og EditEmployee. Vi har valgt at det kun skal være Employee, som skal kunne tilgå siden, da det kun er medarbejdere der skal kunne redigere kundeoplysninger. OnGet henter den Customer/Employee, som skal redigeres, ud fra Id og UserType, da GetUser metoden, kigger på om det er en Customer eller en Employee. I OnPost checker ModelState om det kun er gyldig data der bliver behandlet, hvor den så bliver opdateret gennem UserService metoden UpdateUser.

```

public IActionResult OnGet(int id, User.UserType userType)
{
    var user = _userService.GetUser(id, userType);
    if (user is Employee employee)
    {
        Employee = employee;
    }

    if (Employee == null)
        return RedirectToPage("/NotFound");

    return Page();
}

```

EditEmployee/EditCustomer.cshtml:

- ❖ Vi har sat på Id, readonly, da id'et ikke skal kunne redigeres, så forbliver det samme i databasen. Vi har brugt hidden for Password, så den ikke kan ses for brugeren, men den ikke kan f.eks. nulstilles ved en fejl, men vi viser den stadig er der. Vi har brugt et post-form, så det er muligt at redigere brugeren og sende de oplysninger tilbage til serveren via OnPost metoden. Så har vi brugt asp-for, som genererer inputfelter for Customer/Employee dataen.

```

<Form method="post">
    <input type="hidden" asp-for="Customer.Password" />
    <div class="form-group text-white">
        <label asp-for="Customer.Id" class="control-label text-black"></label>
        <input asp-for="Customer.Id" class="form-control" readonly="@{true}" />
    </div>
    <div class="form-group text-white">
        <label asp-for="Customer.FirstName" class="control-label text-black"></label>
        <input asp-for="Customer.FirstName" class="form-control" />
        <span asp-validation-for="Customer.FirstName" class="text-danger"></span>
    </div>
    <div class="form-group text-white">
        <label asp-for="Customer.LastName" class="control-label text-black"></label>
        <input asp-for="Customer.LastName" class="form-control" />
        <span asp-validation-for="Customer.LastName" class="text-danger"></span>
    </div>

```

CreateEmployee/CreateCustomer.cshtml.cs:

- ❖ I CreateEmployee/CreateCustomer modellen har vi valgt at kun en Employee kan tilgå siden, da kun en medarbejder skal kunne lave en ny bruger og ikke kunderne selv. Så har vi lavet Dependency Injection af UserService i konstruktøren, så vi kan tilgå dens metoder. I OnPost checker ModelState om det kun er gyldig data der bliver behandlet. Her bruger vi så PasswordHasher, hvor vi kryptere kode ordet, det ikke bliver gemt i klar tekst, men er hashed. Efter bliver brugeren tilføjet, gennem AddUser metoden fra UserService.

```

public IActionResult OnPost()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }
    Employee.Password = _hasher.HashPassword(Employee.Email, Employee.Password);
    _userService.AddUser(Employee);
    return RedirectToPage("/Users/Admin/GetAllUsers");
}

```

CreateEmployee/CreateCustomer.cshtml:

- ❖ Her har vi brugt et post-form, så det er muligt at redigere brugerne og sende de oplysninger tilbage til serveren via OnPost metoden. Vi bruger asp-for for at binde inputfeltet til Employee/Customer modellen. Under Password har vi brugt type="password" for at skjule når man skriver adgangskoden.

```
<hr class="bg-white" />
<form method="post">
    <div class="form-group">
        <label asp-for="@Model.Employee.FirstName" class="control-label text-black">Fornavn</label>
        <input asp-for="@Model.Employee.FirstName" class="form-control" />
        <span asp-validation-for="@Model.Employee.FirstName" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="@Model.Employee.LastName" class="control-label text-black">Efternavn</label>
        <input asp-for="@Model.Employee.LastName" class="form-control" />
        <span asp-validation-for="@Model.Employee.LastName" class="text-danger"></span>
    </div>
```

AccessDenied.cshtml:

- ❖ Vi har lavet siden, så når der opstår en fejl, kommer den frem og ikke en kedelig side.

IUserService interface:

- ❖ Dette interface forudbestemmer, hvilke metoder, klasser der arver fra den skal implementeres. Dette inkluderer CRUD og en search metode.

```
public void AddUser<T>(T user) where T : User;

public void DeleteUser<T>(T user) where T : User;

public void UpdateUser(User user);

public User GetUser(int id, User.UserType? user);

public List<User> SearchbyName(string name);
```

UserService Class:

- ❖ UserService går ud på at holde på logikken for brugerne. I UserService klassen har vi valgt at have en Customer liste og en Employee liste som har properties, da her skal alle vores Employees og Customers ligge, når de bliver tilføjet, opdaterede, hentet, slettet og søgt på. I konstruktøren skaber vi en forbindelse mellem listerne og databasen. AddUser metoden tilføjer en user til dens givne liste og tabel i databasen. Det samme for DeleteUser, bare hvor den sletter. I UpdateUser, bruger den DbUserService UpdateUser metoden, så den opdatere i databasen og i listen. SearchByName returnerer en ny liste af de fundne users.

```

0 references | Kasimir Dyring Kaavig, 2 days ago | author, 1 change
public UserService(DbUserService dbUserService)
{
    _dbUserService = dbUserService;

    Customers = _dbUserService.GetCustomers().Result.ToList();
    Employees = _dbUserService.GetEmployees().Result.ToList();

    //_dbUserService.SaveUsers(Employees.Cast<User>().ToList()).Wait();
}

```

DbUserService:

- ❖ DbUserService går ud på at alle metoder for at gemme, slette, lave, opdatere osv. brugere i databasen sker et sted. Generalt er metoderne meget simple. Der bliver tjekket om det er en Customer eller Employee og så tilføjer/fjerner og gemmer brugeren i databasen. Der er også to for at hente Employee og Customer. UpdateUser er dog lidt anderledes. Den kigger om det er en Employee eller Customer. Her bruger vi FindAsync() for at finde den gamle version af brugeren ud fra dens Id og sætter det til et objekt. Hvis den eksisterer, så tager den den gamle version via Entry(existing) og kopierer de opdaterede oplysninger på den. Så skal man ikke opdatere hvert felt enkeltvis. Herefter gemmer den det.

```

public async Task UpdateUser(User user)
{
    if (user is Customer customer)
    {
        var existing = await _userDbContext.Customers.FindAsync(customer.Id);
        if (existing != null)
        {
            _userDbContext.Entry(existing).CurrentValues.SetValues(customer);
            await _userDbContext.SaveChangesAsync();
        }
    }
    else if (user is Employee employee)
    {
        var existing = await _userDbContext.Employees.FindAsync(employee.Id);
        if (existing != null)
        {
            _userDbContext.Entry(existing).CurrentValues.SetValues(employee);
            await _userDbContext.SaveChangesAsync();
        }
    }
    else
    {
        throw new ArgumentException("No users found");
    }
}

```

_Layout.cshtml:

- ❖ Denne fil er det layout som alle sider har tilfælles. oppe i head tagget har vi tre links til stylesheets. så de virker på alle vores pages. body består af navigations baren hvor vi har et billede som linker til index, vi har links til undersiderne. der er link til loogind siden og neden for ses koden som siger vis User har rollen Employee så skal linked til GetAllUsers siden som viser brugere være synlig.
- ❖ Under nav har vi Main hvor kroppen af af alle sider loades ind ved at kalde @RenderBody()
- ❖ Så kommer et Style tag hvor der skrives css til at definere udseende på footeren, der bliver styret ved at bruge classes som vises som et punktum, der bliver defineret en styling på nogle h tags som befinner sig inden for en klass og der bruges # som er et id
- ❖ Næsten i bunden findes footer taget som er den nederste boks på alle siderne. når hjemmesiden vises. der benyttes en div med class="footer-container" som er vores store boks som består af mindre div bokse som har class="footer-column". disse div tags indeholder et h4 tag som er en lille overskrift efterfuldt af p tags som er tekst. den sidste footer-column indeholder a tags med target="_blank" for at disse links åbner i nyt vindue fordi de er til sociale medier.
- ❖ Helt i bunden findes de tre links til javascript filer. disse benytter script tags

```
<ul class="navbar-nav">
    @if (User.IsInRole("Employee"))
    {
        <li class="nav-item">
            <a class="nav-link text-white" asp-area="" asp-page="/Users/Admin/GetAllUsers">Brugere</a>
        </li>
    }

    <li class="nav-item">
        <a class="nav-link text-white" asp-area="" asp-page="/OmOs/OmOs">OM OS</a>
    </li>
</ul>
</div>
```

Program.cs:

- ❖ Øverst ses using-direktiver, som gør det muligt at anvende funktionalitet fra både frameworket og applikationens egne komponenter. Her hentes der EFDbContext mappen som benyttes til database forbindelse. Der ses også Service fra projektet, Microsoft ASPNetCores og EntityFrameWorkCore.
- ❖ Der oprettes en WebApplicationBuilder
- ❖ Der indlæses konfigurationsfiler, som inkluderer appsettings.json og en json fil som er miljø bestemt.
- ❖ Der bygges CarDBContext som henter connectionstring og der specificeres at der skal forbindes til en MySQL database.
- ❖ Der laves Dependency Injection til at tilføje services
 - builder.Services.AddSingleton<IOrderService, OrderService>();
- ❖ Applikationen bruger en cookie baseret autentification
- ❖ Så kommer der HTTP request pipeline konfigurationer. Dette sikrer bl.a. HTTPS, statiske filer, autentificering og routing til Razor Pages.

NuGet (Kronborg)

Vi benytter os af følgende 4 NuGet packages:

Microsoft.EntityFrameworkCore
Microsoft.EntityFrameworkCore.Design
Microsoft.EntityFrameworkCore.Tools
Pomelo.EntityFrameworkCore.MySql

Pomelo.EntityFrameworkCore.MySql som er et open source bibliotek benyttes som en databaseprovider til Entity Framework Core (EF Core), som tillader .NET-applikationer at kommunikere med en MySQL-database.

Diagram Dokumentation

Domain Model

Domain Model- Første udkast: [Bilag 2](#) (Kasper)

- ❖ På vores Product Owners hjemmeside har han valgt at man kan søge efter bilens mærke, type og drivmiddel, hvilket er grunden til at vi har valgt at give Entiteten Car propertiesene Brand, Type of Fuel. Ellers har vi valgt at have med så meget vi kan i forhold til hvordan bilen ser ud, hvad der er i den og hvad den kan.
- ❖ Vi har en Booking entity. Da vi snakkede med Product Owner, kunne han godt tænke sig at man fik en mail eller besked på at man havde bookeet en prøvetur i en bil. Her har vi en customer property, hvor man har en customers, som er logget ind, information, hvor man kan sende en mail og en sms.
- ❖ Vi har valgt at have en Order entity, hvor er tre entiteter, Buy, Sale og Leasing, som arver fra Order, da de er ordretyper selv.
- ❖ Vi har valgt at have en Customer og Employee entity. Det har vi valgt, da vi synes man skal kunne se hvilken ansat, som har stået for handlen af bilen, samt hvilken kunde som har købt, solgt eller lejet bilen.

Domain Model- Andet udkast: [Bilag 3](#) (Kasper)

- ❖ I anden udgave af vores Domain Model, er der blevet tilføjet en User Entity. Det har vi gjort, da både customer og employee er brugere af hjemmesiden og mange af propertiesene går igen i begge entities. På den måde undgår man duplikation.
- ❖ Der er blevet tilføjet en masse properties på Car entitien, da vi gerne ville have flere detaljer på bilen.

Domain Model 2. sprint: [Bilag 4](#) (Kasper)

- ❖ I vores sprint nr. 3 har vi ændret i vores entity "Sale", og ertsattet property "Price", med 2 nye "SalePrice" og "SaleDate". De to nye properties skal hjælpe med at simplificere initialiseringen af koden, så man har mere specifik information om købet. Det samme er gjort under entity "Buy", hvor "Price" er ændret til "BuyPrice" og "BuyDate".
- ❖ Der er tilføjet "PhoneNumber" til entity "Customer", da vi skal bruge kundens telefonnummer for at få kontakt angående henvendelser.
- ❖ Under entity "Leasing" har vi tilføjet "MontlyPayment", da man på hjemmesiden skal kunne indtaste hvad den månedlige rate skal ligge på, så kunden også kan se hvad de skal betale månedligt for leasing af bilen.
- ❖ Under entity "User", har vi tilføjet "UserType". UserType skal bruges til at identificere, hvilken bruger vi har med at gøre. Medarbejder, ejer eller kunde.
- ❖ I entity "Order", har vi fået tilføjet properties "Car: Car", "Employee: Employee" og "Costumer: Costumer". Dette har vi gjort, da vi skal bruge properties fra entity Car, Employee og Costumer.

Domain Model 3. sprint: [Bilag 5](#) (Oliver)

- ❖ Car tilføjer Model attribut
- ❖ User FullName bliver tilføjet
- ❖ Customer Address er fjernet i sprint 3

Domain Model 4. sprint: [Bilag 6](#) (Oliver)

- ❖ Image klassen er blevet tilføjet og indeholder Id, ImageString, CarId, Url, Car og har en relation med car. nul til mange billeder til en bil
- ❖ Car tilføjer PriceMonth, LeasingPeriod og KmIncluded
- ❖ Customer har fået Id, FirstName, LastName, FullName, Email, UserType
- ❖ Leasing har tilføjet YearStart, YearEnd, MonthStart, MonthEnd, DayStart, DayEnd
- ❖ Sale tilføjer SaleYear, SaleMonth, SaleDay
- ❖ Buy tilføjer BuyYear, BuyMonth, BuyDay

Class Diagram

Class diagram – Første udkast: [Bilag 7](#) (Oliver)

- ❖ Klasse Diagrammet er en forlængelse af Domain Modellen. Her går vi så over i klasser, som har metoder. Her har vi i alle klasser, tilføjet CRUD metoder, altså Add, Delete, Update og Get. Da vi var usikre på hvilke metoder hver klasse skulle bruge, startede vi simpelt ud med noget som alle klasserne har brug for.

Class diagram – Andet udkast: [Bilag 8](#) (Oliver)

- ❖ Der er blevet tilføjet en GetAllCars() metode i Car og en GetOrders() metode i Orders, da vi kunne se at vi har brug for at kunne se alle ordre som employee og at man skal kunne se alle biler som bruger.

Class diagram – 2 sprint: [Bilag 9](#) (Oliver)

- ❖ User har fjernet Name og tilføjet FirstName, LastName og FullName
- ❖ Car har skiftet navn på metoderne RemoveCar() til DeleteCar() og GetAllCars() til GetCars()
- ❖ Employee har fjernet List<Employee> og har tilføjet en ToString()
- ❖ Customer har tilføjet en ToString()
- ❖ Order har tilføjet Søge metoderne NameSearch<T>(), SortById<T>(), PriceFilter<T>()
- ❖ Sale og Buy har skiftet Price ud med SalePrice og BuyPrice

Class diagram – 3 sprint: [Bilag 10](#) (Oliver)

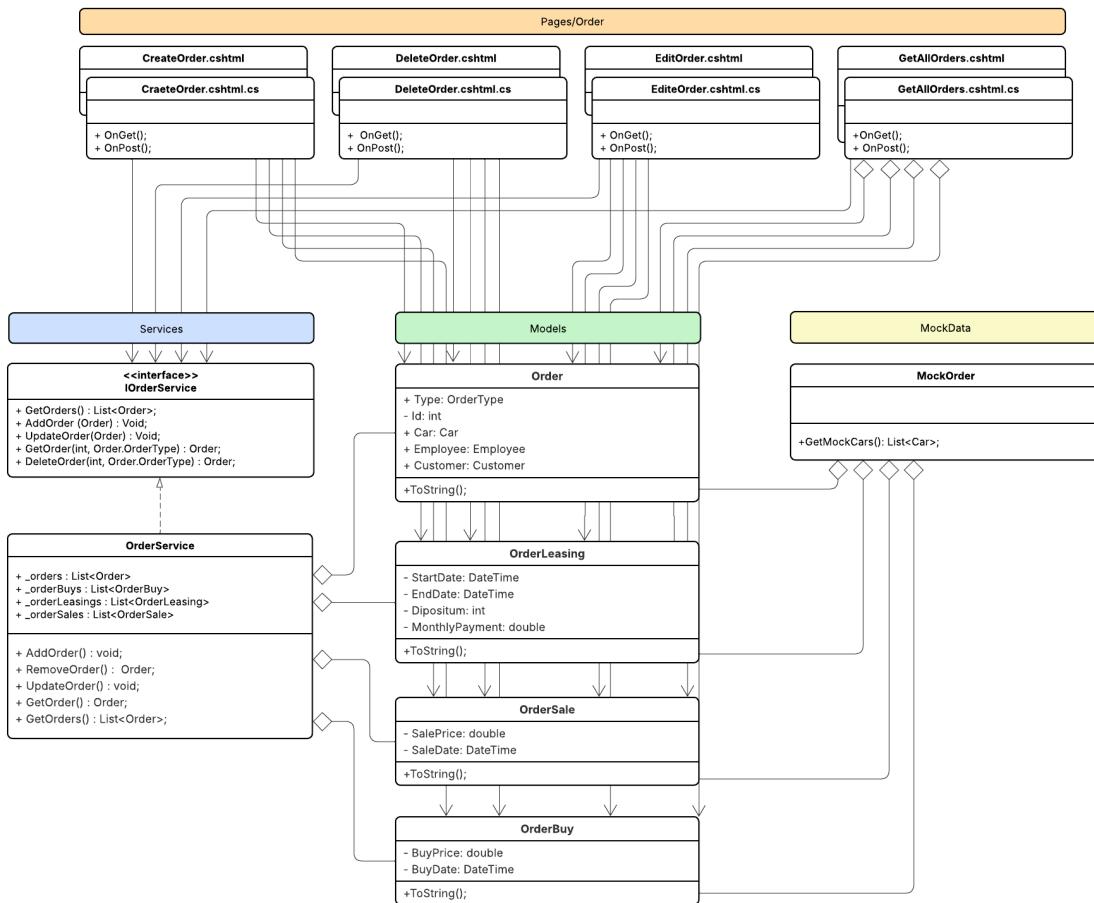
- ❖ Car har fået skiftet CarModel til Model
- ❖ GetOrderById<T>(List<T>, int) er blevet tilføjet til Order
- ❖ User har fået fjernet FullName og fået tilføjet metoden SearchById()
- ❖ Customer har fået fjernet Address attribut

Class diagram – 4 sprint: [Bilag 11](#) (Oliver)

- ❖ Sammenlignet med sprint 3 er diagrammet i sprint 4 blevet mere ryddeligt og fokuserer i højere grad på datamodellen frem for funktionalitet. Flere metoder er fjernet fra diagrammet i sprint 4, mens enkelte strukturelle og semantiske ændringer er blevet foretaget.
- ❖ Add(), Get(), Update() og Delete() metoder er blevet fjernet fra Car, Order, User og Booking
- ❖ Image klassen er blevet tilføjet og har en nul til mange billedere til en bil relation.

Razor page Diagram (Martin)

I (Bilag X) ses vores første udkast af et Razor Page Diagram. Formålet med et Razor Page diagram, er at illustrere hvordan en lille del af vores Razor page snakker sammen, og hvordan den er bygget op. I vores første udkast af Razor Page Diagram, har vi lavet en forudsigelse for hvordan vi forventer en del af vores Razor Page med "Car" kommer til at køre.



BMC (Kasper)

Bilag 19

BusinessModelCanvas er en essentiel model som bruges til at skabe et overblik over ændringer, som kan blive lavet til et eksisterende firma eller til en helt ny forretningside. I vores tilfælde har vi lavet en BMC til et eksisterende firma.

1. **Key Partners** - Key partners er de individer som hjælper med at opfylde **Key Activities**, ved hjælp af **Key Resources**.

- a. Vi har valgt at inkludere vores kunder som en nøglefaktor i at vores projekt lykkedes. Det er essentielt at kunder sælger og køber fra os, da det er virksomhedens omsætning.
 - b. Vi har samtidig også valgt vores mekanikere som en essentiel nøgle i at virksomheden kan køre rundt. Deres funktion er at vedligeholde biler, og sørge for de står i god stand, som gør det attraktivt for kunder.
 - c. Vores salgsafdeling er den vigtigste afdeling, da de står for salg og køb af bil. De har dialogen med kunderne, og giver dem den service og vejledning der skal til.
2. **Key Activities** - Dette områder dækker over de daglige opgaver som får tingene til at køre rundt.
- a. Først har vi valgt at vedligeholdelse af biler, som står på adressen, spiller en stor factor. Som nævnt ovenfor, så er salgsraten for biler væsentligt højere, hvis bilen er i god stand.
 - b. En ting som vores salgsafdeling fokuserer på, er at jævnligt tjekke mails i indbakken, i tilfælde at kunder har skrevet ind angående prøvetur, spørgsmål til biler, eller bare generel interesse.
 - c. Vi har et rengøringshold som ugentligt kommer ind og rengøres og pusser af, så butikken står nydeligt og pænt.
3. **Key Resources** - Dette hjælper med hvilke værktøjer der skal bruges for at få klaret hverdagens opgaver.
- a. Som tidligere nævnt bruger vi meget vores mekanikere, da deres primære funktion er vedligeholdelse af biler
 - b. Salgs afdelingens primære funktion er kommunikation med kunder, samt folk som skal sælge deres biler.
 - c. En vigtig ting som gør vores virksomhed lukrativ, er selvfølgelig vores biler. Bilerne er årsagen til virksomhedens eksistens.
 - d. Vi har et samarbejde med virksomheden FTZ, som er stor distributør af billede. Billede er essentielle i den forstand, at hvis en bil skal have skiftet tandrem, luftfilter osv, kan FTZ hurtigt levere nogle nye dele.
4. **Value Proposition** - Her beskrives hvad vi skaber som giver værdi for kunden.
- a. Processen for testkørsel er følgende; kunde går ind på hjemmesiden, og udfylder et ark med oplysninger, samt den bil, man ønsker at prøvekøre. Når dette er udfyldt, sendes der en mail ind til virksomheden, hvorefter virksomheden ringer til kunden for at aftale tid og dato. Dette er en nogenlunde holdbar proces, men virksomheden ønsker at de i fremtiden kan få tilføjet et system hvor kunden kan se ledige datoer på hjemmesiden, hvor de kan komme forbi til en prøvekørsel, hvorefter tiden så skal blive bekræftet af salgsafdelingen. Ved bekræftelse sendes der en SMS til kunden om bekræftelse.
 - b. Virksomheden fokuserer primært på at sælge brugte biler, men har tidligere solgt nye biler. De ønsker at sælge flere nye biler i fremtiden.

5. **Customer Relationship** - Hvordan får vi kunder? Hvordan holder vi på vores kunder? Hvordan fortsætter vi med at skabe værdi for vores kunder?
 - a. Grunden til at ideen om automægler fungerer godt, er at sælgende kunder kan få lov til at beholde deres bil, imens vi sælger den. Det eneste krav vi har til kunden er at de vedligeholder bilen i salgsprocessen.
 - b. Når kunder har deres bil på værksted, eller deres bil er til udstilling for en købene kunde, får den sælgende kunde en lånebil udleveret.
 - c. Når sælgende kunder skal sælge deres bil, får de 10.000-30.000 kr. retur oven i bilens salgsværdi.
 - d. Vi tager ansvaret for alt salg af bilen. Det eneste, den sælgende kunde skal gøre, er at aflevere bilen, hvis en potentiel køber ønsker at se bilen.
 - e. Hvis bilen ikke er solgt inden for 30 dage, tager virksomheden ikke penge for tjenesten af salg.
6. **Customer Segments** - Vores vigtigste kunder.
 - a. Vores kundesegment er alle kunder som har kørekort, eller folk som ønsker at skille sig af med deres bil.
7. **Channels** - Hvordan leverer vi vores forslag. Er det online eller gennem en fysisk butik. Her skal vi definere, hvilken måde er den bedste.
 - a. Facebook. Facebook hjælper med at ramme et stort antal potentielle kunder, da mange mennesker er på facebook. Algoritmen på Facebook hjælper med at skabe kunder som søger/har søgt efter salg af bil. Virksomheden bruger meget Facebook, da de ofte lægger billeder og videoer op om virksomheden.
 - b. Hjemmeside. Hjemmesiden er vores hovedprodukt. Det er stedet alle bliver henvist til, når de viser interesse for virksomhedens produkt. Hjemmesiden viser alt hvad kunden eftersøger, samt har en behagelig UI, så kunden ikke bliver skræmt væk, eller ikke er bange for at trykke på noget.
 - c. Trustpilot. Det kommer ikke bag på nogle af Trustpilot hyppigt bliver brugt, hvad angår nye kunder. Virksomheden kigger ofte på Trustpilot for at se anmeldelser af virksomheden, men det er ikke noget de tager så tungt, hvis der kommer en dårlig. Årsagen til en dårlig anmeldelse af virksomheden, skyldes ofte at virksomheden ikke har informeret kunden nok om diverse ting hvad angår salg af bil. De svarer hyppigt på de dårlige anmeldelser med grunde til den oplevelse de har fået.
 - d. Fysisk butik. I butikken skal det mærkes, at deres system er blevet optimeret. Dette vil sige mere overskud, og mindre travlhed. Mere overskud betyder mere tid til at modernisere, eller optimere butikken.
8. **Cost Structures** - Her skal vi have den primære udgift. Ud fra det kan vi overveje, hvad vores "return on investment (ROI)" skal være.
 - a. Den største udgift er selve processen for køb og salg af bil. I tilfælde af at bilen skal repareres, har de selvfølgelig også en udgift. ROI kommer når bilen

- skal sælges. Som virksomhed har virksomheden en fordel, da de kan sælge bilen til samme pris som en privat sælger, men at de sælger bilen med garanti, eller forsikringer. Kunder vælger at købe hos forhandler, da de kan tilbyde disse ydelser.
- En udgift som selvfølgelig også spiller en rolle, er den løn som skal betales til medarbejderne.
 - Virksomheden laver ikke selv deres reklamer. Dette har de en virksomhed "Social Mate" i Aalborg som sørger for. Virksomheden kommer månedligt på besøg, og tager nogle billeder, samt laver en video eller 2. Annoncer online, og i aviser står virksomheden også for. Dette vil sige at ALT markedsføring foregår gennem Social Mate.

9. Revenue Streams - Her ligger vores potentielle indtægt.

- Hvis en bil bliver solgt inden for 30 dage, bliver der betalt en difference fra kunden, som ender ud i profit for virksomheden.
- En stor indkomst kommer også fra værkstedet. Reservedele er som regel billige at bestille, men det som kunden betaler for, er mandetimerne som bliver brugt på bilen.
- Når virksomheden køber en bil, slår de cirka 15% af købsprisen af bilen, så det giver dem mulighed for at tjene på bilen. Det vil sige, at hvis en bil har en markedsværdi på 250.000kr, køber de den for 215.000kr.
- Leasing er en månedlig indkomst som virksomheden tjener penge på. Der ligger dog ikke en stor fortjeneste i leasing, men det skaber en stabil passiv indkomst.

E/R (Kronborg)

Bilag 22

Vores MySQL bliver hostet af simply.com og nedenstående entity relationship diagram afspejler databasens opbygning som er opbygget for at facilitere en automægler. Datamodellen dækker information om biler, kunder, medarbejdere og de fire handlinger, der tilbydes: lessing, sale, byg og booking. Alle tabellerne har et Id som primærnøgle som automatisk tæller op ved at bruge Identity(1,1) eller AUTO_INCREMENT.

Car

Denne tabel indeholder information om de enkelte biler.

Primærnøgle: Id

Fields: Id, TypeOfCar, Brand, Color, Model, Fuel, ModelYear, Price, Mileage, KmPrl, TopSpeed, Doors, HorsePower, Gear, Cylinders, MotorSize, ZeroToOnehundred, Length, NumOffwheels, MaxPull, Weight, StockStatus, Status, PriceMonth, LeasingPeriod, KmIncluded, ForSale

Relation: Sale, Leasing, Buy, booking Via Car.Id og CarId som er i de andre tables.

Relation: Images table via Image.CarId og Car.Id

Images

Denne tabel indeholder billederne af bilerne

Primærnøgle: Id

Fields: Id, ImageString, CarId

Relation: Car table via.Images.CarId og Car.Id

Employee

Denne tabel indeholder den information om medarbejderne der skal bruges

Primærnøgle: Id

Fields: Id, FirstName, LastName, Type, Email, Password

Relation: Sale, Leasing, Buy Via Employee.Id og EmployeeId som er i de andre tables.

Customer

Denne tabel indeholder informationer om kunder

Primærnøgle: Id

Fields: Id, FirstName, LastName, PhoneNumber, Email, WishToSell, Password

Relation: Sale, Leasing, Buy, booking Via Customer.Id og CustomerId som er i de andre tables.

Order

Denne tabel indeholder alle transaktions handlinger

Primærnøgle: Id

Fields: Id, LeasingId, SaleId, BuylId

Relation: Sale via SaleId og Id i Sale table

Relation: Leasing via LeasingIdog Id i Leasing table

Relation: Buy via BuylId og Id i Buy table

Leasing

Denne tabel indeholder informationer der skal bruges når der skal laves leasing af bil

Primærnøgle: Id

Fields: Id, StartDate, EndDate, Dipositum, CarId, CustomerId, EmployeeId

Relation: Car table via CarId og Id i Car table

Relation: Customer table via CustomerId og Id i Customer table

Relation: Employee table via EmployeeId og Id i Employee table

Relation: TheOrder table via Id og LeasingId i Leasing table

Sale

Denne tabel indeholder informationer der skal bruges i et salg af bil

Primærnøgle: Id

Fields: Id, Price, CarId, CustomerId, EmployeeId

Relation: Car table via CarId og Id i Car table

Relation: Customer table via CustomerId og Id i Customer table

Relation: Employee table via EmployeeId og Id i Employee table

Relation: TheOrder table via Id og SaleId i Sale table

Buy

Denne tabel indeholder informationer der skal bruges i et køb af bil

Primærnøgle: Id

Fields: Id, Price, CarId, CustomerId, EmployeeId

Relation: Car table via CarId og Id i Car table

Relation: Customer table via CustomerId og Id i Customer table

Relation: Employee table via EmployeeId og Id i Employee table

Relation: TheOrder table via Id og BuylId i Buy table

Booking

Denne tabel indeholder information om bookings så man kan bestille en prøvetid

Primærnøgle: Id

Fields: Id, CarId, CustomerId, Date

Relation: Car table via CarId og Id i Car table

Relation: Customer table via CustomerId og Id i Customer table

SWOT (Kronborg)

S	W
Mægler Koncept med høj kundetilfredshed Gennemsigtighed og ærlighed Mindre Virksomhed	Bilpriser Salg af bil
O	T
Udvidet tilstedeværelse online Omstillingsparat i udvalg af drivmiddel Erhvervs samarbejdspartnere	Konkurrence fra etablerede bilforhandlere Ændringer i kundernes præferencer

S-trenght: En styrke de har er deres unikke mægler koncept, hvor man kan sælge sin bil gennem dem, imens man stadig kører rundt i den. der garanteres solgt eller gratis, som betyder at kunden for solgt sin bil eller stoppe forsøget på salg hvis det ikke er lykkes inden for 30 dage. med over 4000 solgte biler siden 2005 har de erfaring og som de reklamere med så har de stor kundetilfredshed. Som mindre virksomhed har man et mere personligt forhold til kunderne, samt processen ikke er lige så firkantet, så man kan lettere give tilbud og bedre service til ens stamkunder.

W-eakness: En svaghed kunne være når man skal sælge en bil. Hvis der går lang tid og bilen ikke er blevet solgt, bliver man nødt til at sætte prisen ned, hvilket gør at firmaet taber penge på den. Ellers fordi markedet ændrer sig, falder priserne på nogle typer biler, hvilket gør at man kan taber penge på biler man har købt.

O-pportunity: Virksomheden kan styrke sin position på markedet gennem øget digital tilstedeværelse, hvor målrettet markedsføring og synlighed på sociale medier tiltrækker en bredere kundekreds. Samtidig åbner den grønne omstilling med fokus på elbiler og bæredygtige transportløsninger, nye muligheder i voksende segmenter samt B2B muligheder som kunne være aftale med mad udbringning, virksomhed eller andre erhverv som indebærer kørsel i et vis omfang.

T-hreat: Trusler er at de er en lille virksomhed sammenlignet med de store digitale platforme der er i danmark så som Bilbasen.dk og Biltorvet.dk som er store platforme med flere resourcer samt autouncle.dk hvor man kan sælge sin bil gennem dem gratis. Nye regler og forventninger i forhold til teknologi kan være kosteligt på tid og økonomi. Skift i økonomien hos købe segmentet kan resultere i fald af gennemførte transaktioner og voksende konkurrence på markedet kan også presse på prisen.

Svært at finde nye mekanikere

Unit Test (Martin)

Vi har lavet en lille unit test på employee på et par af den properties, som tjekker om navnet på en employee passer med det forventede og hvis det ikke er ens for man en error message.

Videreudvikling (Martin og Rasmus)

Vi har tænkt os at videreudvikle på programmet, ved at få de nuværende funktioner til at fungere.

Men hvis vi skulle videreudvikle vores program vil vi gerne ud at have booking implementeret så når du klikker på en bil, kan man bestille en tid hvor man kan komme forbi butikken og få en prøve tur.

Vi kunne også godt tænke os at implementere en køber database. Dette er en funktion hvor kunden skal have adgang til en skabelon, som de kan udfylde i forhold til hvordan deres ønskede bil ser ud (bilmærke, størrelse, motor, afstand kørt osv.). Når en bil så dukker op med matchende information, kan en medarbejder så tage kontakt til den kunde.

At kunden har nulstil adgangskode. Kunden skal have adgang til et link til en separat side, hvor man kan nulstille og lave en ny adgangskode.

Når man laver en bruger og når man logger ind, skal der være en knap, så man kan se den adgangskode man skriver, da den er gemt væk nu.

Lave create user siden færdig, så man kan vælge mellem enten en medarbejder eller kunde og lave brugeren på en side, i stedet for to separate.

Lave edit-user siden færdig, så når man trykker på linket, kan den se om det er en kunde eller en medarbejder og deres informationer kommer frem med de rigtige inputfelter.

Refleksion (Kasper)

Når vi kigger tilbage på alle vores User Stories, er der noteret nogle som ikke ender med at blive implementeret. Den primære årsag er, at det i praksis ikke vil fungere. En ting vi har bidt mærke i, er at vores forventning til forløbet, er at det ville starte langsomt med udvikling og opsætning, og går hurtigt til sidst, men dette var ikke tilfældet. Alt er gået i et relativt tempo, men ikke hurtigt nok. Vi har haft nogle få komplikationer som har holdt os tilbage, blandt andet med yderligere udvidelser undervejs, som har forsinket processen for det igangværende projekt. Vi har nået mindre end vi håbede på. "Car" er ikke lavet færdig til deadline, og der mangler stadig at blive implementeret en del ting til "Car". Funktioner som er essentielle for hjemmesiden. "User" er færdigt, dog var der flere komplikationer undervejs.

Konklusion (Kronborg)

I starten af projektet var der flow, første iteration af diagrammer blev designet, user stories blev skrevet og programmet blev startet op. Som tiden gik og vi begyndte at starte at implementere tingene fra vores diagrammer, fandt vi ud af at vi ikke var gode nok til at opdatere vores diagrammer og sprint backlog (Jira) med hvad der blev arbejdet på, hvad var færdig lavet og hvad vi manglede.

Undervejs har der været en del problemer med at få de forskellige klasser / metoder til at gøre som vi gerne vil have, her til sidst stødte vi på en problem hvor github tilføjede en nye mappe, hvorefter mange fejl begyndte at dukke op og en masse filer som hele tiden bliver overskrevet hver gang der bliver committed fra et gruppemedlem.

Så vi kan konkludere at det gik fint indtil her til sidst hvor et eller andet er gået galt og det har ødelagt det hele.

Bilag (Alle)

Bilag 1: User Stories

Business Points	User Stories	Acceptance Criteria(Customer)	Acceptance Criteria(Coder)	Story Points
1.	Jeg som ejer, ønsker at se en oversigt over alle biler, med det formål at se vores sortiment	Test at man kan se alle biler. Test at man ikke kan se en bil.	Test at data fra databasen bliver udskrevet på siden. Test at ingen data bliver udskrevet.	5
2.	Jeg som bruger, ønsker at kunne se alle biler der kan leases, med det formål at have en oversigt over alle biler der kan leases.	Test at man kan lease en bil. Test at man ikke kan lease en bil.	Test at en leaset bil bliver oprettet i en table. Test at den ikke tilføjer en leaset bil til tabellen ogcaster en exception.	5
3.	Som ejer ønsker jeg at kunne tilføje biler til vores sortiment, med det formål at kunne sælge flere biler.	Test at man kan oprette en bil, Test at man ikke kan oprette uden at udfylde alt info	Test at der oprettes et element i databasen. Test der ikke oprettes	4
4.	Jeg, som ejer, ønsker at kunne fjerne biler fra hjemmesiden, med det formål at kunne fjerne solgte biler fra vores sortiment.	Test at den sletter Test at der kun slettes det ønskede	Test at den sletter fra databasen Test at der kun slettes det ønskede	4
5.	Som ejer, ønsker jeg at kunne redigere oplysninger om biler, med det formål at kunne ændre i bilens status.	Test at man kan redigere oplysninger på biler. Test at når man redigerer oplysninger på biler og det slår fejl.	Test at den opdaterer fra databasen Test at der kun opdaterer det ønskede	4
6	Jeg, som kunde, ønsker at kunne se informationer om bilen, med det formål at tjekke bilens oplysninger.	Test at man kan se bilens oplysninger. Test at man ikke kan se bilens oplysninger.	Test at GetCar printer en bil. Test at GetCar ikke printer en bil.	4
7	Som ejer, ønsker jeg at kunne lave en ny ordre, med det formål at ordren bliver gemt i databasen.	Test at orden er blevet gemt. Test at orden ikke bliver gemt.	Test at ordenen bliver gemt i databasen. Test at orden ikke bliver gemt i databasen.	4
8	Som ejer, ønsker jeg at kunne se alle ordrer der er blevet lavet, med det formål at kunne se tidligere ordrer.	Test at alle ordrer vises i tabellen. Test at ingen ordrer kan ses i tabellen.	Test at GetOrders() metoden returnerer alle ordrer. Test at GetOrders() ikke returnere noget.	5
9	Som ejer, ønsker jeg at kunne slette ordrer, med det formål at fjerne unødvendige ordrer.	Test at ordenen bliver slettet. Test at ordenen ikke bliver slettet.	Test at DeleteOrder() fjerner ordren fra databasen. Test at DeleteOrder() ikke fjerner ordren fra databasen.	4

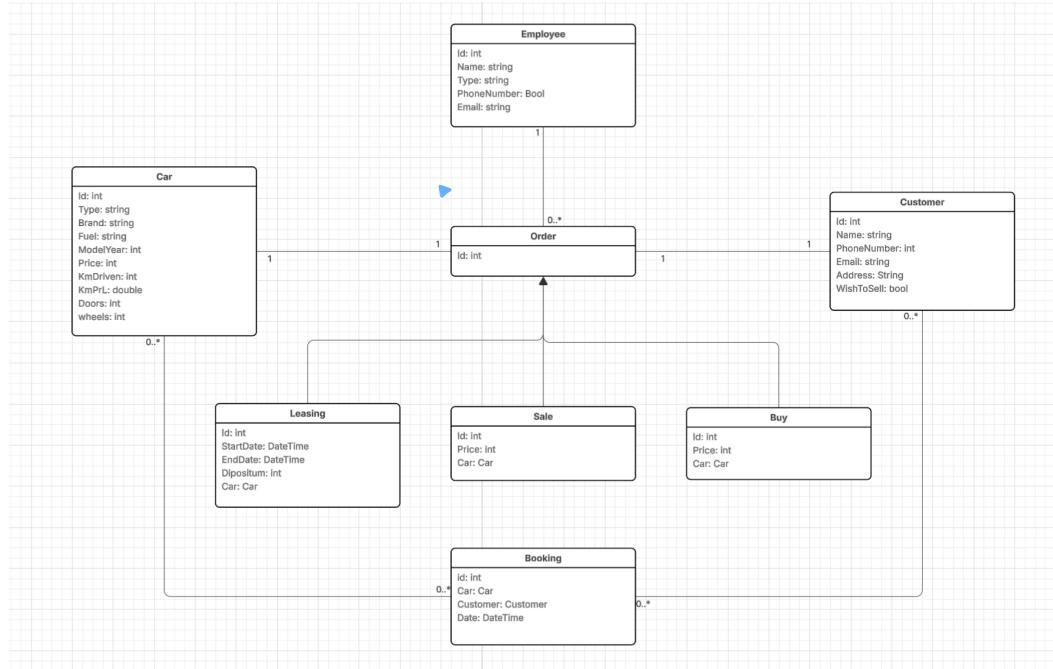
10	Som ejer, ønsker jeg at kunne se alle leasing ordrer der findes i systemet, med det formål at kun at se dem.	Test at kun leasing ordrer kan ses. Testen viser ikke kun leasing ordrer.	Test at GetOrders() kun returnerer leasing orders. Test at GetOrder() ikke returnerer leasing orders.	4
11	Som ejer, ønsker jeg at kunne se alle købs ordrer der findes i systemet, med det formål at kun at se dem.	Test at kun købs ordrer kan ses. Testen viser ikke kun købs ordrer.	Test at GetOrders() kun returnerer købs orders. Test at GetOrder() ikke returnerer købs orders.	4
12	Som ejer, ønsker jeg at kunne se alle salgs ordrer der findes i systemet, med det formål at kun at se dem.	Test at kun salgs ordrer kan ses. Testen viser ikke kun salgs ordrer.	Test at GetOrders() kun returnerer salg orders. Test at GetOrder() ikke returnerer salg orders	4
13	Som ejer, ønsker jeg at kunne finde en specifik ordre frem, med formålet at jeg skal kigge alle ordrer igennem.	Test at den rigtige order bliver fundet frem. Test at den ikke kan finde den rigtige ordre frem.	Test at GetOrder() returnere den rigtige ordre. Test at GetOrder() ikke returnere den rigtige ordre.	4
14x	Som medarbejder, ønsker jeg at kunne logge ind på hjemmesiden, med det formål at have en bruger.	Test at man kan logge ind som medarbejder. Test at man ikke kan logge ind som medarbejder.	Test at OnPost() metoden virker. Test at OnPost() metoden ikke virker.	3
15x	Som medarbejder, ønsker jeg at kunne lave en bruger for en kunde, med det formål at gemme kundens oplysninger og ordrehistorik.	Test at man kan tilføje en bruger. Test at man ikke kan tilføje en bruger.	Test at AddUser() metoden virker. Test at AddUser() metoden ikke virker.	4
16	Som medarbejder, ønsker jeg at kunne se en kundes bruger, med det formål at kunne se hvad deres oplysninger er.	Test at man kan finde en bruger. Test at man ikke kan finde en bruger.	Test at GetUser() metoden virker. Test at GetUser() metoden ikke virker.	4
17	Som medarbejder, ønsker jeg at kunne slette en bruger, med det formål at fjerne en kunde fra databasen.	Test at man kan slette en bruger. Test at man ikke kan slette en bruger.	Test at DeleteUser() metoden virker. Test at DeleteUser() metoden ikke virker.	4
18	Jeg som bruger, ønsker at kunne søge efter alle biler der kan købes, med det formål at kunne finde biler der kan købes.	Test at de ønskede biler kommer frem.	Test at man kan printe en liste af biler. Test at man ikke kan bruge SQL injektion	6
19	Jeg som bruger, ønsker at kunne filtrere efter forskellige mærker, typer, drivmidler og pris af biler, med det formål at kunne filtrere efter specifikke biler.	Test at man kan filtrere efter mærke, type, drivmiddel og pris. Test at ikke man kan filtrere efter mærke, type, drivmiddel og pris.	Test at man kan lave en Sort() som sorterer efter diverse søgekriterier. Test at Sort() ikke virker.	6
20	Jeg som ejer, ønsker at tilføje billeder af vores biler, med det formål kunder kan se bilerne.	Test at der kommer billeder frem af bilerne. Test at der ikke kommer billeder frem af bilerne.	Test af når vi indsætter link til billede af bil, at billedet vises.	5

21.	Jeg som ejer, ønsker alle åbningstider og kontaktilinformation nederst på hver side, med det formål at hjælpe kunden med at finde den information let.	Test at informationen kommer frem på alle sider. Test informationen ikke kommer frem på alle sider.	Test at de ønskede oplysninger bliver vist i den korrekte Header.	2
22.	Jeg som kunde, ønsker at kunne få en vurdering af min bil for det formål at måske sælge bilen	Test at bilen bliver vurderet. Test at biler der ikke lever op til kravene bliver afvist	Test at der er en visuel repræsentation af vurderingen Test	4
23.	Som ejer, ønsker jeg at gemme på kundens information under bilens nummerplade, med det formål at vide hvem der ejer bilen og have deres information.	Test at som admin, at man kan finde en kundes information frem. Test at man som bruger, ikke kan finde en kundes information frem. Test at man, som admin, ikke kan finde en kundes information frem.	Test at man med en søgefunktion, kan søge efter nummerplade, og printe oplysninger på kunder.	3
24.	Jeg som kunde, ønsker at kunne booke en prøvetur, med det formål at se om bilen er noget for mig.	Test at man kan booke en prøvetur af en bil. Test at man ikke kan booke en prøvetur af en bil.	Test at funktionen BookTestDrive() virker, og at den viser ledige tider i Razor.	7
25	Som ejer, ønsker jeg at lede efter en ordre ved hjælp af kundenavn, med det formål at jeg kun kan huske navn på kunde.	Test at det er den rigtige kundes ordre der kommer frem. Test at det ikke er den rigtige kundes ordre, der kommer frem.	Test at NameSearch() finder de ordrer frem som høre til kunden. Test at NameSearch() ikke finder de ordrer frem som høre til kunden.	5
26	Som ejer, ønsker jeg at kunne finde ordrer ud fra priser, med det formål at finde alle biler /leasede biler i en hvis prisklasse.	Test at den viser alle biler imellem den søgte prisklasse. Test at den ikke viser alle biler imellem den søgte prisklasse.	Test at PriceFilter() returnerer alle de biler i den søgte prisklasse. Test at PriceFilter() ikke returnerer alle de biler i den søgte prisklasse.	5
27	Som ejer, ønsker jeg at kunne opdatere ordrer, med det formål at ændre detaljer i orden.	Test at den nye information er gemt. Test at den nye information ikke bliver gemt.	Test at UpdateOrder() opdaterer og gemmer den nye information. Test at UpdateOrder() ikke opdaterer eller gemmer den nye information.	4
28.	Jeg, som ejer, ønsker en brugervenlig hjemmeside, med det formål at kunder nemt kan begå sig rundt på hjemmesiden, og ikke er "bange" for at trykke på noget.	Test at hjemmesiden er brugervenlig. Test at hjemmesiden ikke er brugervenlig	generate en Lighthouse report test at det er UCD overvej Stepwise Refinement	10
29.	Jeg som kunde, ønsker viden omkring jeres forretning, med det formål at finde sikkerhed hos jer som bilsælger og forhandler.	Test at der står information på hjemmesiden omkring forretningen. Test at informationen ikke kommer frem på hjemmesiden.	Test at den ønskede information står tydeligt sted. Test at intet information bliver udskrevet.	3

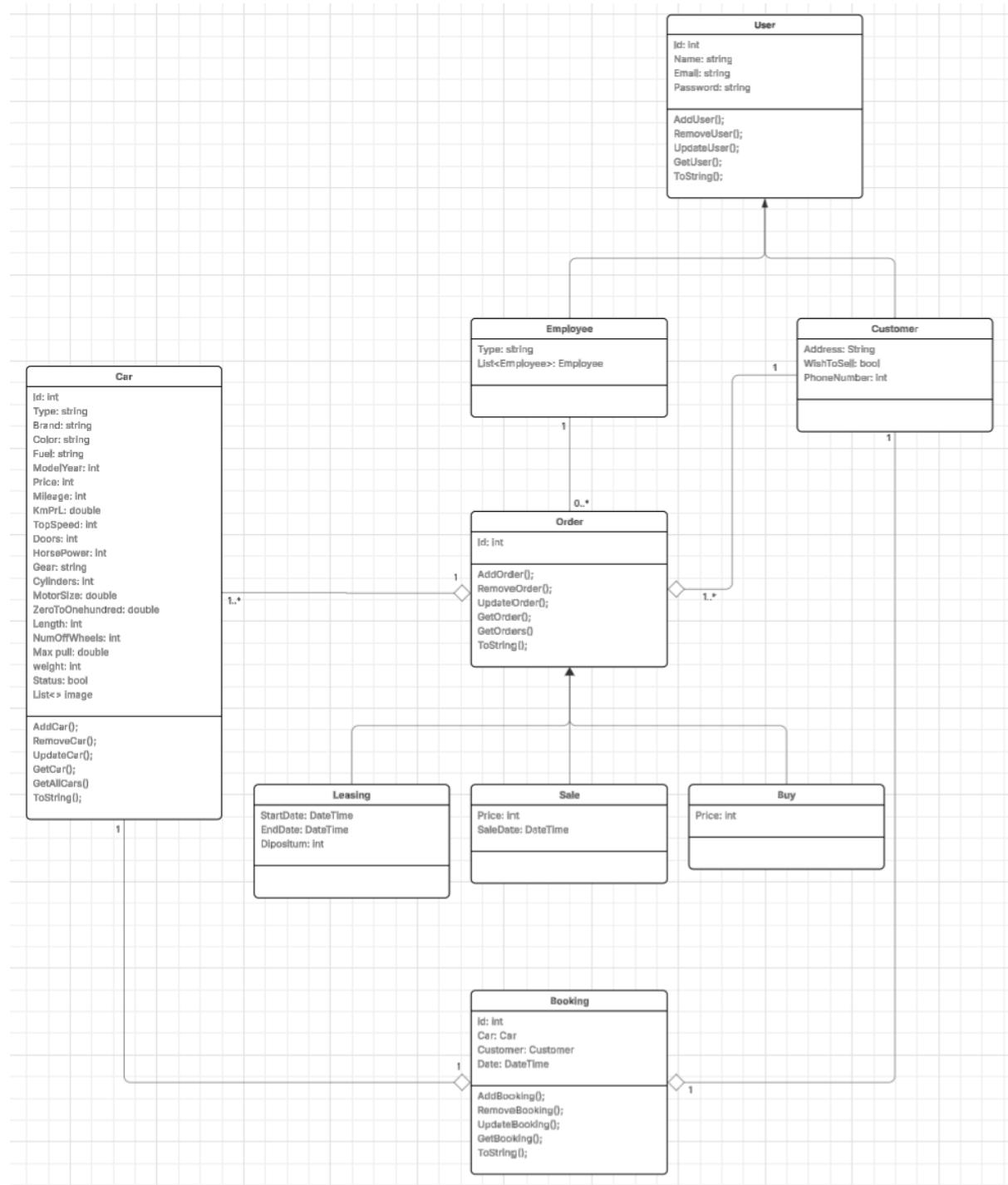
30.	Jeg som kunde, ønsker information omkring processen for hvordan I sælger biler, med det formål at vide hvordan i gørt tingene.	Test at man informationen kommer frem på hjemmesiden. Test at informationen ikke kommer frem på hjemmesiden.	Test at der står tekst i Header på hvordan salgsprocessen er.	3
31	Som kunde, ønsker jeg at kunne logge ind på hjemmesiden, med det formål at have en bruger.	Test at man kan logge ind som kunde. Test at man ikke kan logge ind som kunde.	Test at OnPost() metoden virker. Test at OnPost() metoden ikke virker.	3
32	Som medarbejder, ønsker jeg at kunne ændre på kundens oplysninger, med det formål at hvis f.eks. en email er blevet ændret.	Test at man kan redigere i en bruger. Test at man ikke kan redigere i en bruger.	Test at UpdateUser() metoden virker. Test at UpdateUser() metoden ikke virker.	4
33.	Jeg som ejer, ønsker at have en kort video omkring forretningens koncept på en side, med det formål at hurtigt og let forklarer til kunder, hvad vi går ud på.	Test at man kan se videoen. Test at man ikke kan se videoen.	test videoen vises test af lyd test funktionerne	4
34.	Jeg som kunde, ønsker at se alt udstyr på bilen.	Test at man kan se alt udstyr på bilen	Test at man kan se en liste af udstyr i Header under billede,	3
35.	Jeg som ejer, ønsker der kommer ikoner til tekst med vigtig data.	Test at der kommer ikoner til tekst	Test at der kommer ikoner til tekst, som indeholder vigtig data.	4
			Total:	151

Domain Model Bilag

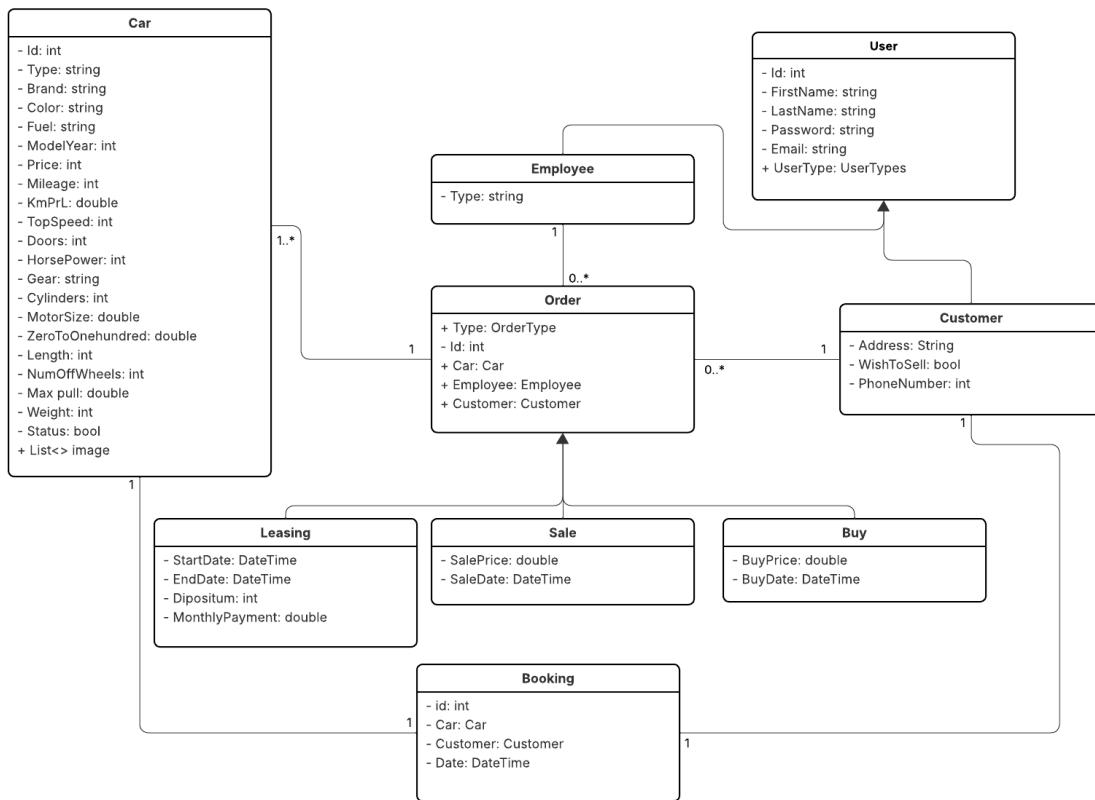
Bilag 2: Første udkast af domæne model - sprint 1



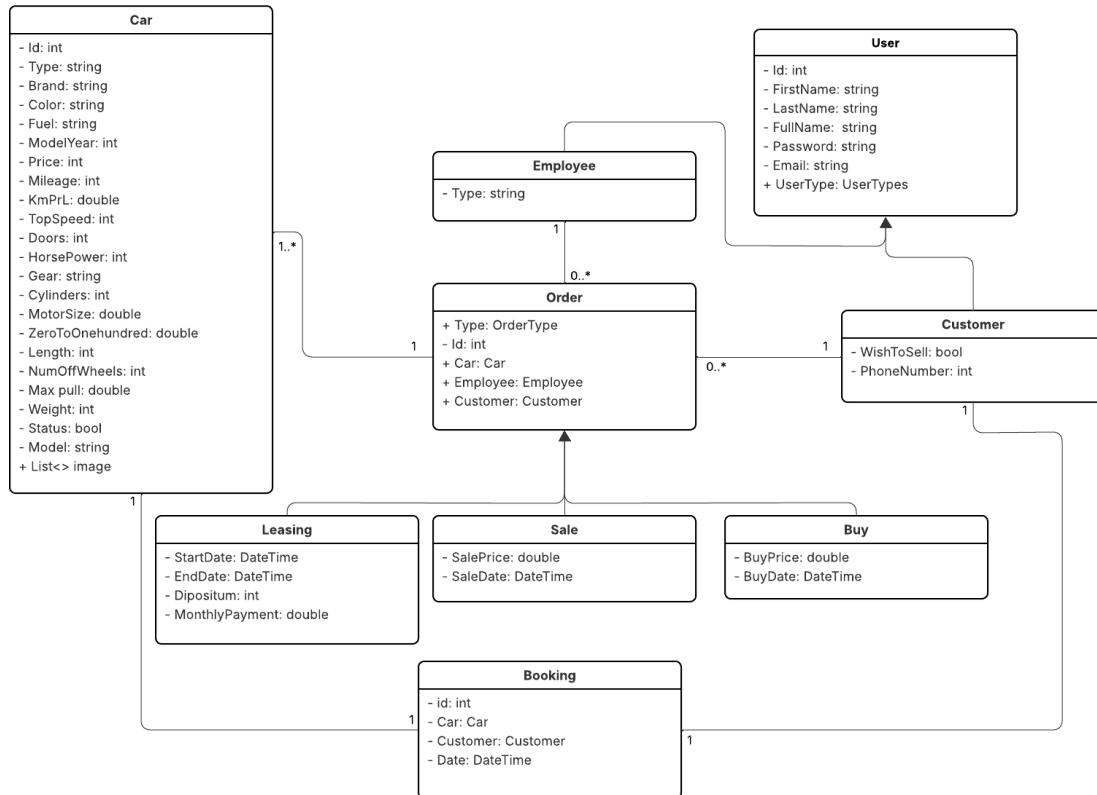
Bilag 3: Andet udkast af domæne model - sprint 1



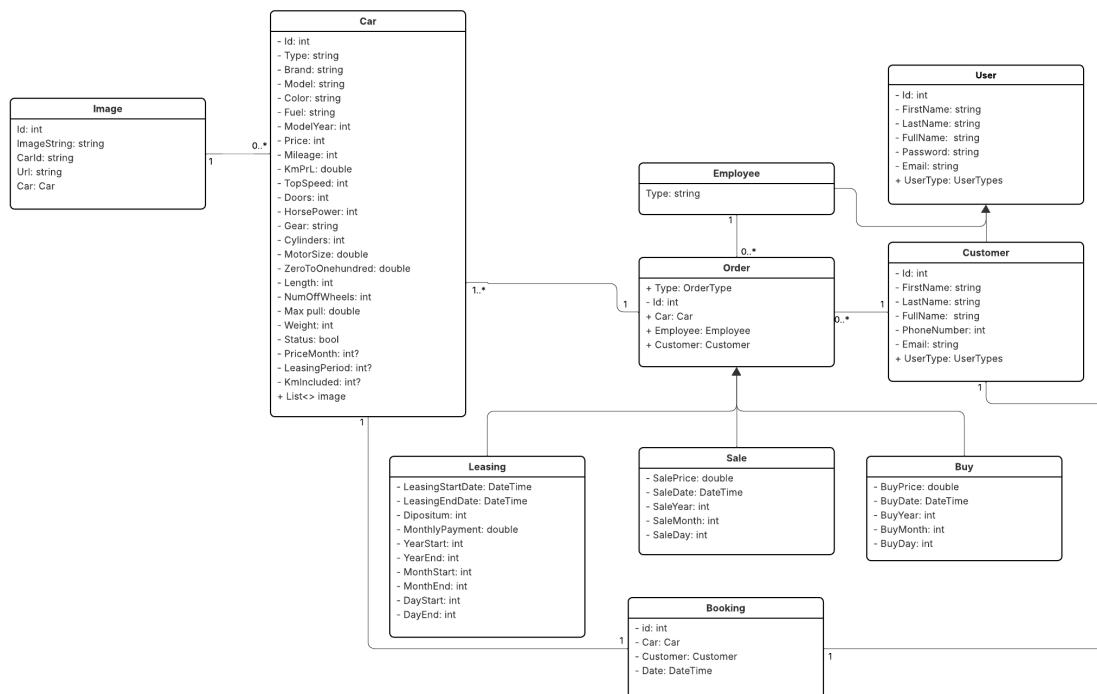
Bilag 4: Udkast på domain model - Sprint 2



Bilag 5: Udkast på domain model - Sprint 3

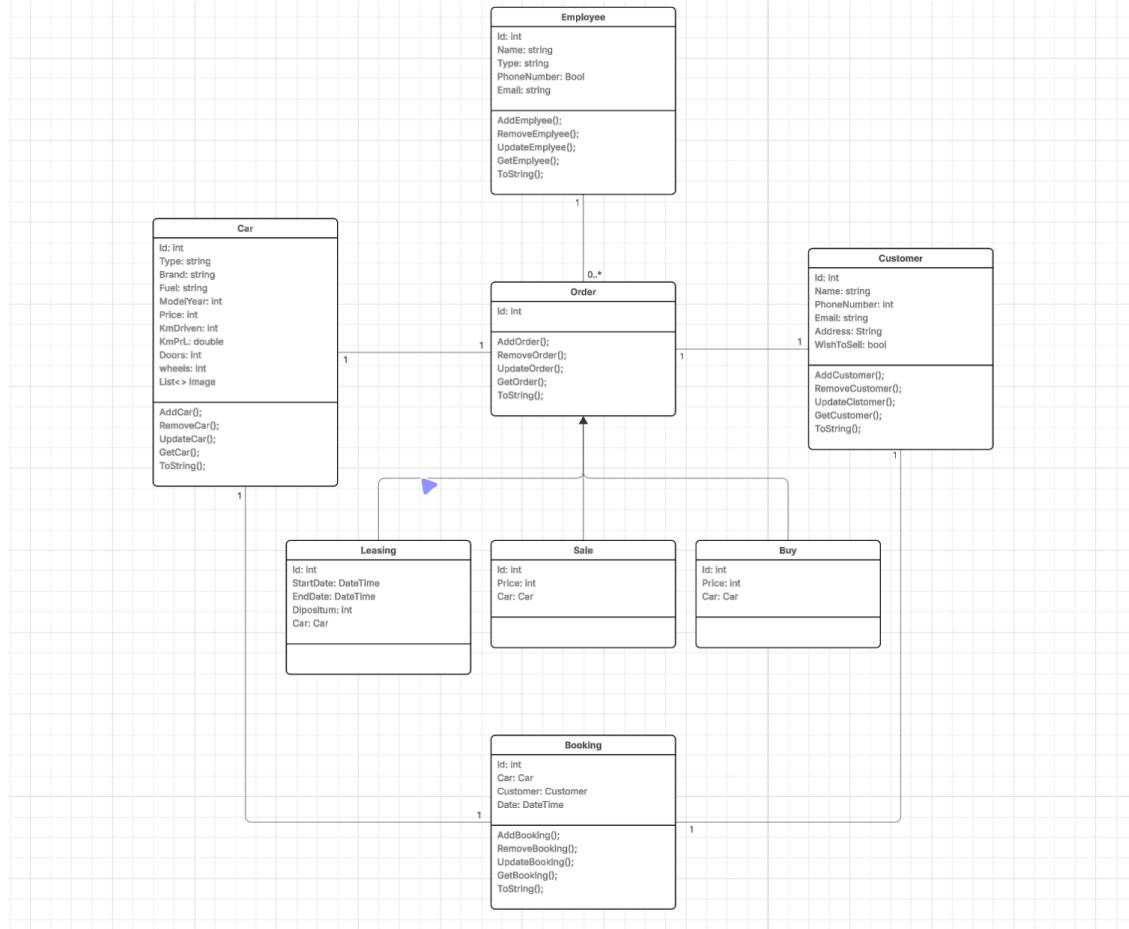


Bilag 6: Slut domain model - Sprint 4

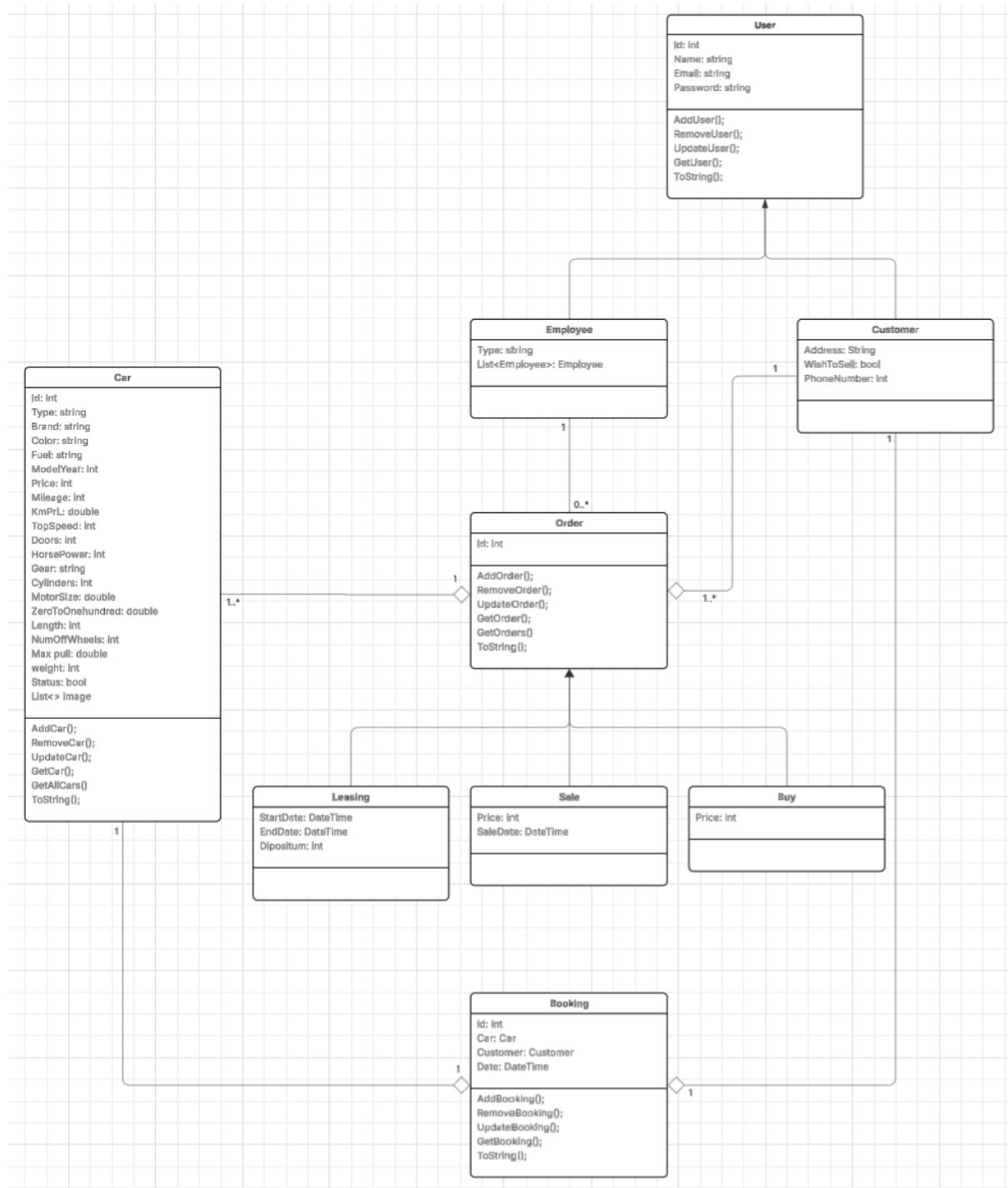


Class Diagram Bilag

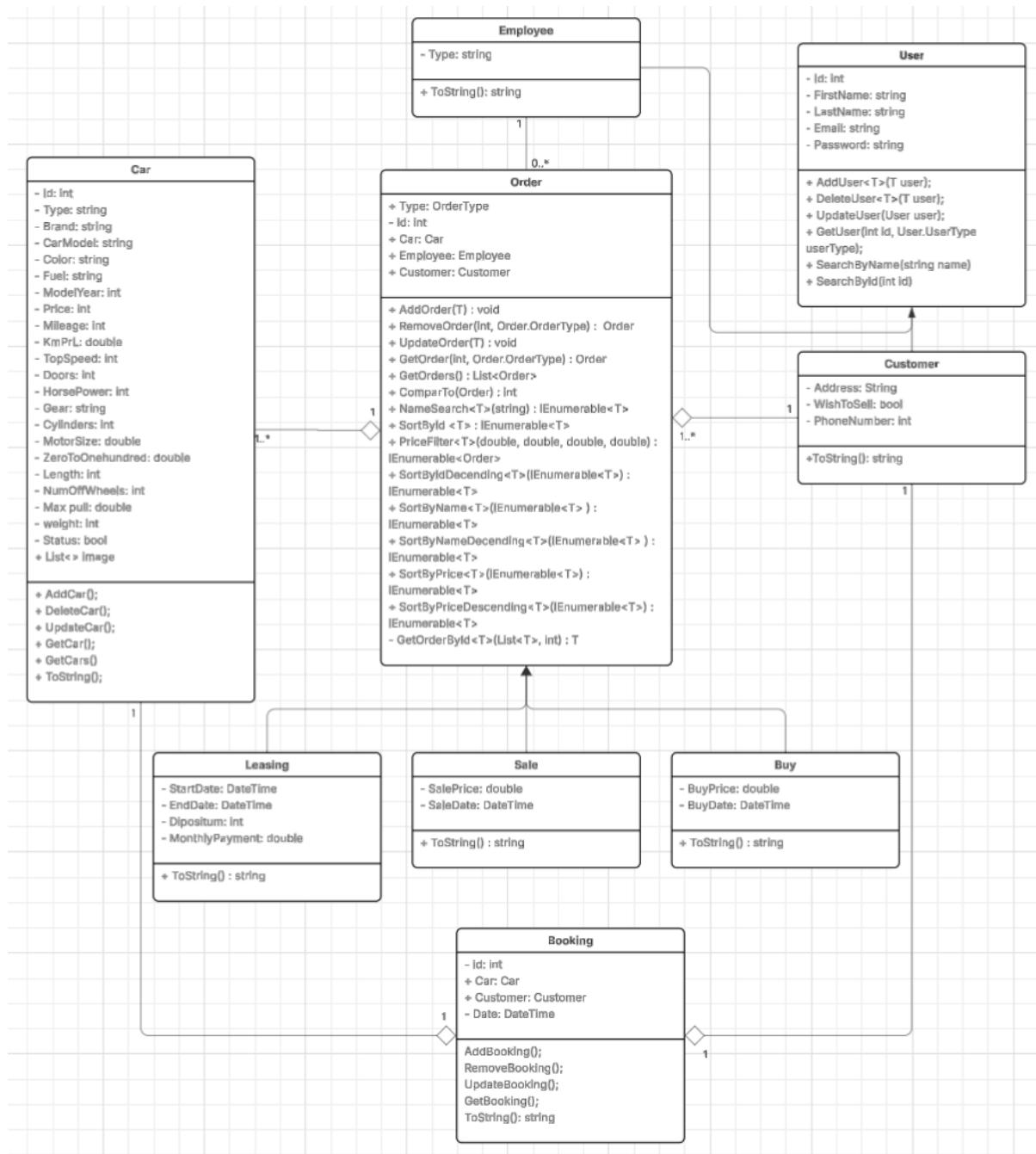
Bilag 7: Første udkast på Class Diagram - Sprint 1



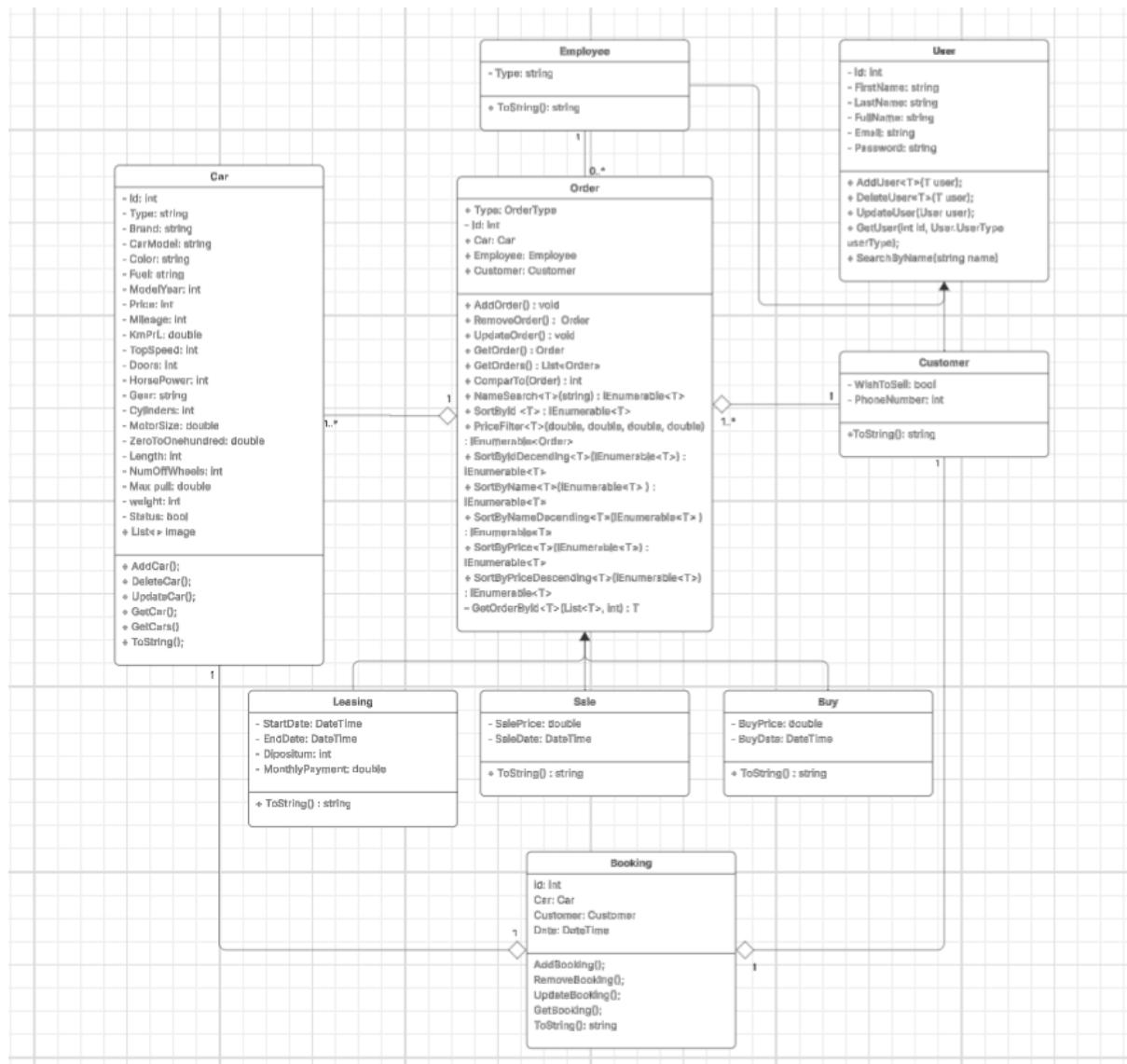
Bilag 8: Anden Udkast på Class Diagram - Slut Sprint 1



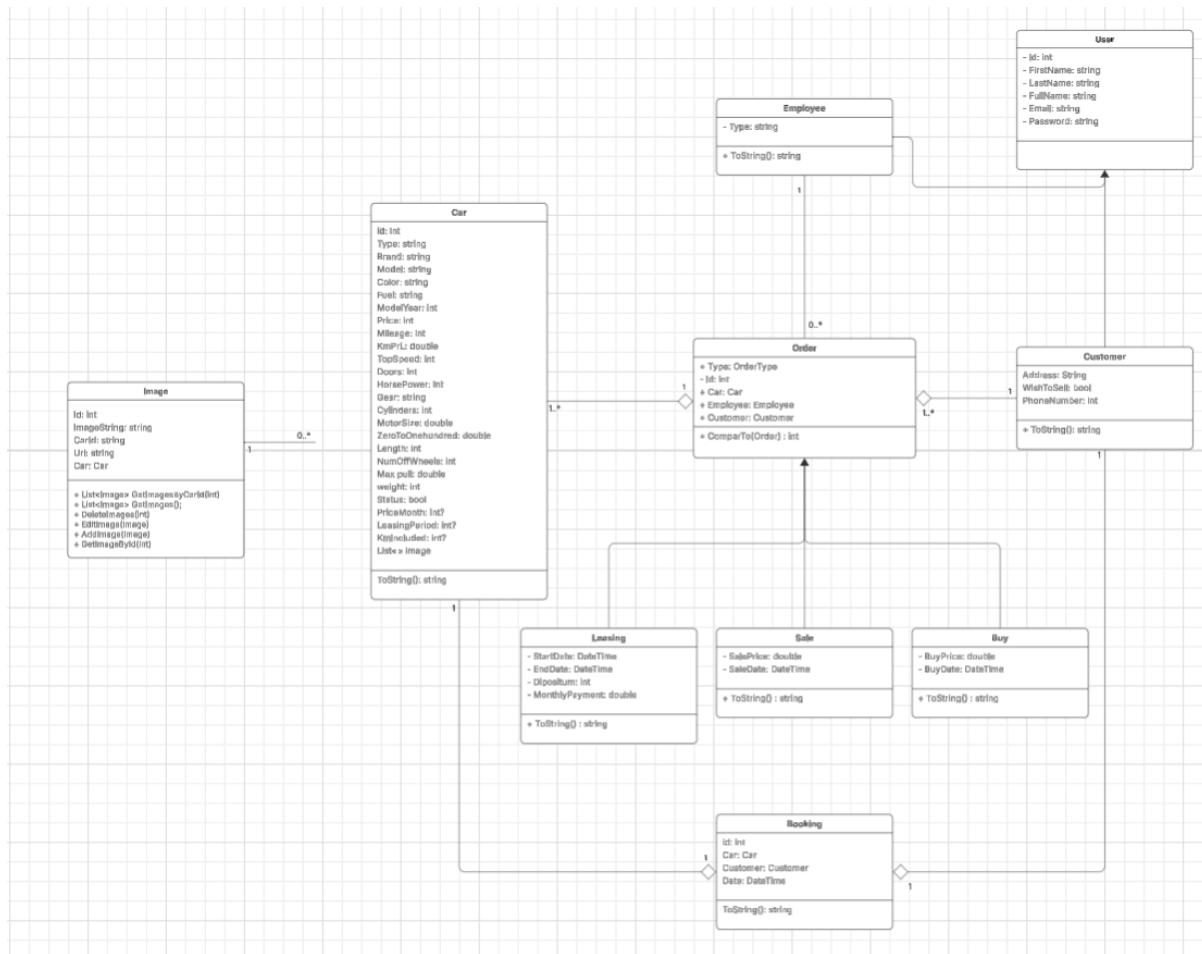
Bilag 9: Udkast på Class Diagram - Sprint 2



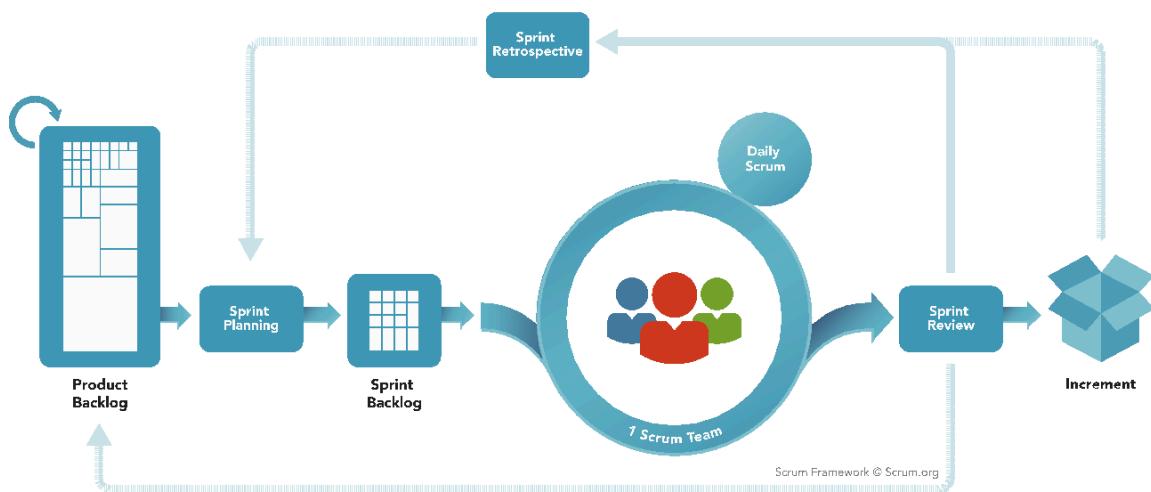
Bilag 10: Udkast på Class Diagram - Sprint 3



Bilag 11: Slut Class Diagram - Sprint 4



Bilag 12: Scrum syklus



Sprint BackLog (Jira)

Bilag 13: Sprint Backlog - Sprint 1

TO DO [35]

- US 1: Jeg som ejer, ønsker at se en oversigt over alle biler, med det formål at se vores sortiment.
 - E2S-15 = OK
- US 2: Jeg som bruger, ønsker at kunne se alle biler der kan leases, med det formål at have en oversigt over alle biler der kan leases.
 - E2S-16 = KS
- US 3: Som ejer ønsker jeg at kunne tilføje biler til vores sortiment, med det formål at kunne sælge flere biler.
 - E2S-17 = KS
- US 4: Jeg, som ejer, ønsker at kunne fjerne biler fra hjemmesiden, med det formål at kunne fjerne solgte biler fra vores sortiment.
 - E2S-18 = OK
- US 5: Som ejer, ønsker jeg at kunne redigere anlægningerne

+ Create

IN PROGRESS [9]

- Opret DB
 - E2S-2 = OK
- Lav BMC
 - E2S-13 = RR
- Simply.com
 - E2S-5 = OK
- Lav ER Diagram
 - E2S-1 = OK
- Lav VCA
 - E2S-14 = M
- Razor Page Diagram
 - E2S-4 = RR
- Lav SWOT
 - E2S-11 = OK
- Lav Domain Model

DONE [1]

- Visual Studio projektstart
 - E2S-6 = M

1/1 >

Bilag 14: Sprint Backlog - Sprint 2

JOBLISTE [34]

- US 7: Jeg som bruger, ønsker at kunne filtrere efter forskellige mærker, typer, drivmidler og pris af biler, med det formål at kunne filtrere efter specifikke biler.
✓ E2S-21 = OK
- US 2: Jeg som bruger, ønsker at kunne se alle biler der kan leases, med det formål at have en oversigt over alle biler der kan leases.
✓ E2S-16 = OK
- US 19: Jeg som ejer, ønsker der kommer ikoner til tekst med vigtig data.
✓ E2S-33 = KS
- US 31: Som kunde, ønsker jeg at kunne logge ind på + Opret

I GANG [3]

- Burndown chart
✓ E2S-8 = KS
- Lav SWOT
✓ E2S-11 = KS
- Lav Razor Page Diagram
✓ E2S-4 = KS

FÆRDIG ✓ [7]

- Lav ER Diagram
✓ E2S-1 ✓ = KS
- Lav BMC
✓ E2S-13 ✓ = KS
- Simply.com
✓ E2S-5 ✓ = KS
- Lav VCA
✓ E2S-14 ✓ = KS
- Opret DB
✓ E2S-2 ✓ = KS
- Lav Class Diagram
✓ E2S-12 ✓ = KS
- Lav Domain Model

Bilag 15: Sprint Backlog - Sprint 3

The image shows a digital sprint backlog board with three columns: JOBLISTE, I GANG, and FÆRDIG.

JOBLISTE (24 items):

- mea vigtig data.
✓ E2S-33 = KS
- US 8: Jeg, som kunde, ønsker at kunne se informationer om bilen, med det formål at tjekke bilens oplysninger.
✓ E2S-22 = OK
- US 20: Jeg som ejer, ønsker at tilføje billeder af vores biler, med det formål kunder kan se bilerne.
✓ E2S-34 = OK
- US 23: Som ejer, ønsker jeg at gemme på kundens information under bilens nummerplade, med det formål at vide hvem der ejer bilen og have deres information.
✓ E2S-37 = RR
- + Opret

I GANG (9 items):

- ✓ E2S-32 = KS
- US 6: Jeg som bruger, ønsker at kunne søge efter alle biler der kan købes, med det formål at kunne finde biler der kan købes.
✓ E2S-20 = KS
- US 7: Jeg som bruger, ønsker at kunne filtrere efter forskellige mærker, typer, drivmidler og pris af biler, med det formål at kunne filtrere efter specifikke biler.
✓ E2S-21 = KS
- US 9: Jeg som ejer, ønsker at tilføje billeder af vores biler, med det formål kunder kan se bilerne.
✓ E2S-23 = KS
- US 14: Jea. som ejer.

FÆRDIG (11 items):

- Lav ER Diagram
✓ E2S-1 ✓ = KS
- US 31: Som kunde, ønsker jeg at kunne logge ind på hjemmesiden, med det formål at have en bruger.
✓ E2S-45 ✓ = KS
- US 10: Jeg som ejer, ønsker alle åbningstider og kontaktilinformation nederst på hver side, med det formål at hjælpe kunden med at finde den information let.
✓ E2S-24 ✓ = KS
- US 16: Jeg som kunde, ønsker information omkring processen for hvordan I sælger biler, med det formål at vide hvordan i gør tingene.

Bilag 16: Sprint Backlog - Sprint 4

The image shows a digital sprint backlog board with three columns: JOBLISTE, I GANG, and FÆRDIG.

JOBLISTE (14 items):

- US 4: Jeg som ejer, ønsker at kunne fjerne biler fra hjemmesiden, med det formål at kunne fjerne solgte biler fra vores sortiment.
E2S-18 (status: OK)
- US 5: Som ejer, ønsker jeg at kunne redigere oplysninger om biler, med det formål at kunne ændre i bilens status.
E2S-19 (status: OK)
- US 33: Jeg som ejer, ønsker at have en kort video omkring forretningens koncept på en side, med det formål at hurtigt og let forklarer til kunder, hvad vi går ud på.
E2S-47 (status: OK)
- + Opret

I GANG (7 items):

- Burndown chart
- E2S-8 (status: KS)
- US 27: Som ejer, ønsker jeg at kunne opdatere ordrer, med det formål at ændre detaljer i orden.
E2S-41 (status: KS)
- US 20: Jeg som ejer, ønsker at tilføje billeder af vores biler, med det formål kunder kan se bilerne.
E2S-34 (status: KS)
- US 6: Jeg som bruger, ønsker at kunne søge efter alle biler der kan købes, med det formål at kunne finde biler der kan købes.
E2S-20 (status: KS)
- US 9: Jeg som ejer, ønsker at tilføje billeder af vores

FÆRDIG (23 items):

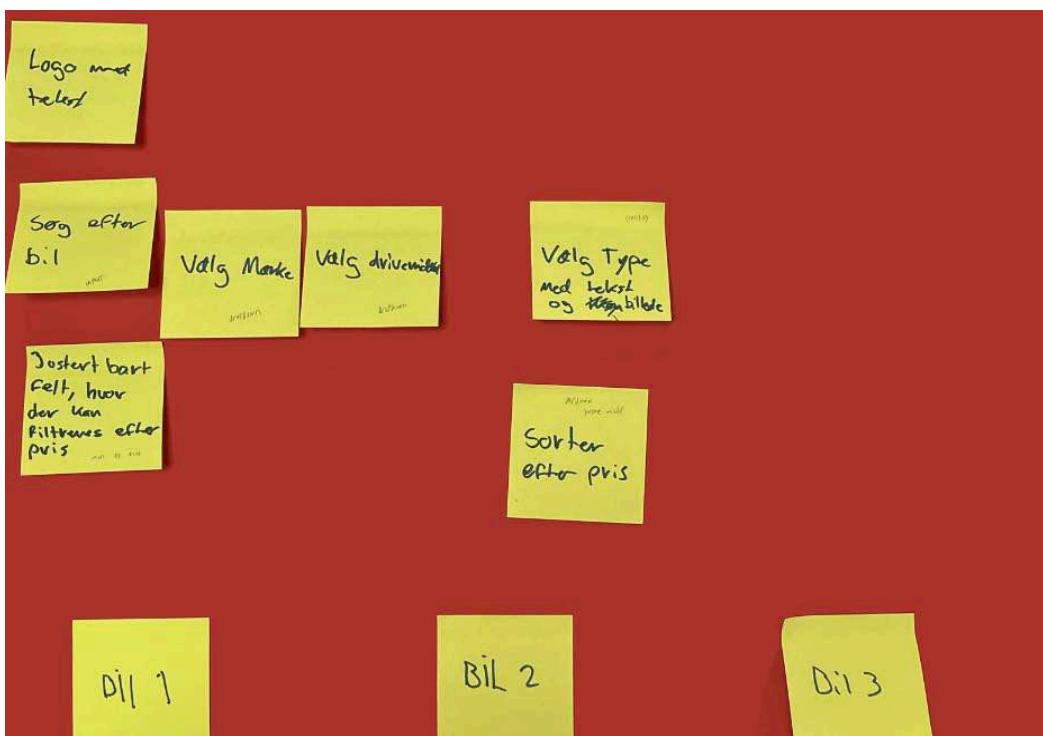
- US 31: Som kunde, ønsker jeg at kunne logge ind på hjemmesiden, med det formål at have en bruger.
E2S-45 (status: ✓ = KS)
- US 11: Jeg som kunde, ønsker at kunne få en vurdering af min bil for det formål at måske sælge bilen.
E2S-25 (status: ✓ = KS)
- US 18: Jeg som kunde, ønsker at se alt udstyr på bilen.
E2S-32 (status: ✓ = KS)
- US 34: Jeg som kunde, ønsker at se alt udstyr på bilen.
E2S-48 (status: ✓ =)
- US 30: Jeg som kunde,

Mockup (CKasper og Kronborg)

Bilag 17: Mockup Startside

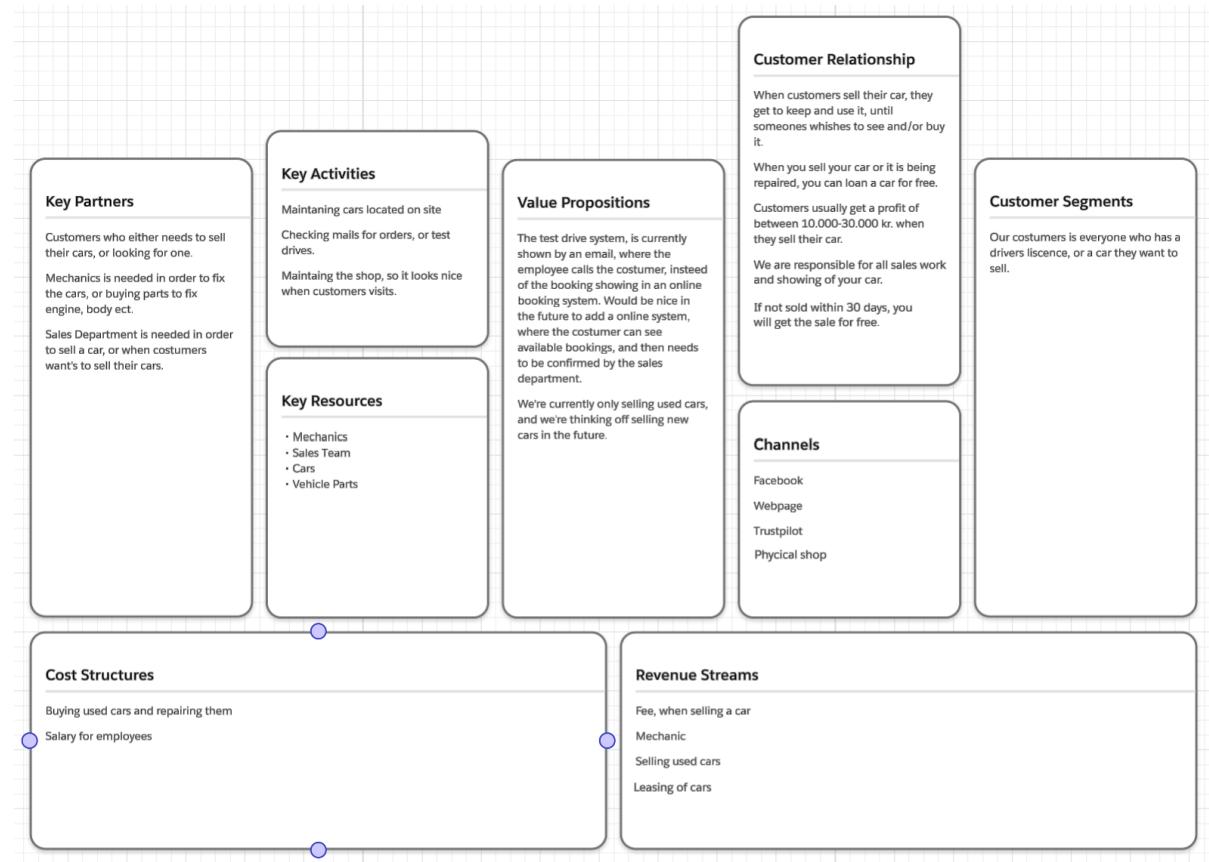


Bilag 18: Mockup Køb Bil side



BMC Model

Bilag 19: BMC Model - sprint 1

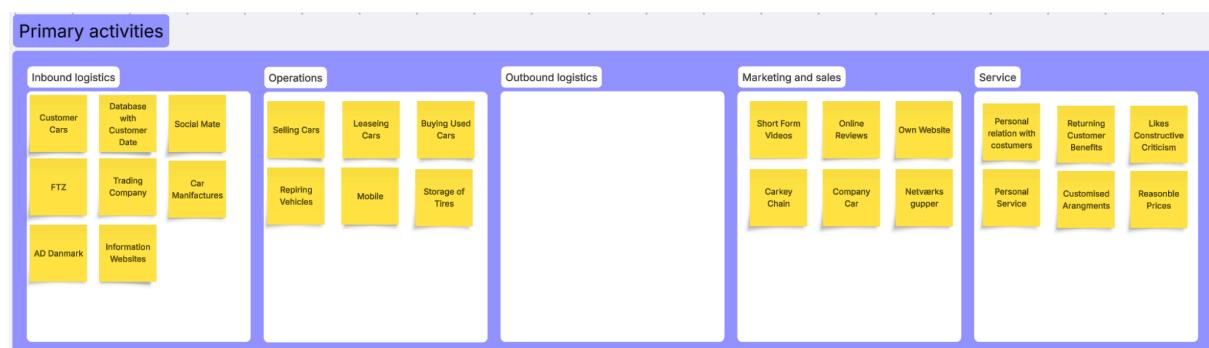


VCA

Bilag 20: VCA - Secondary Activities

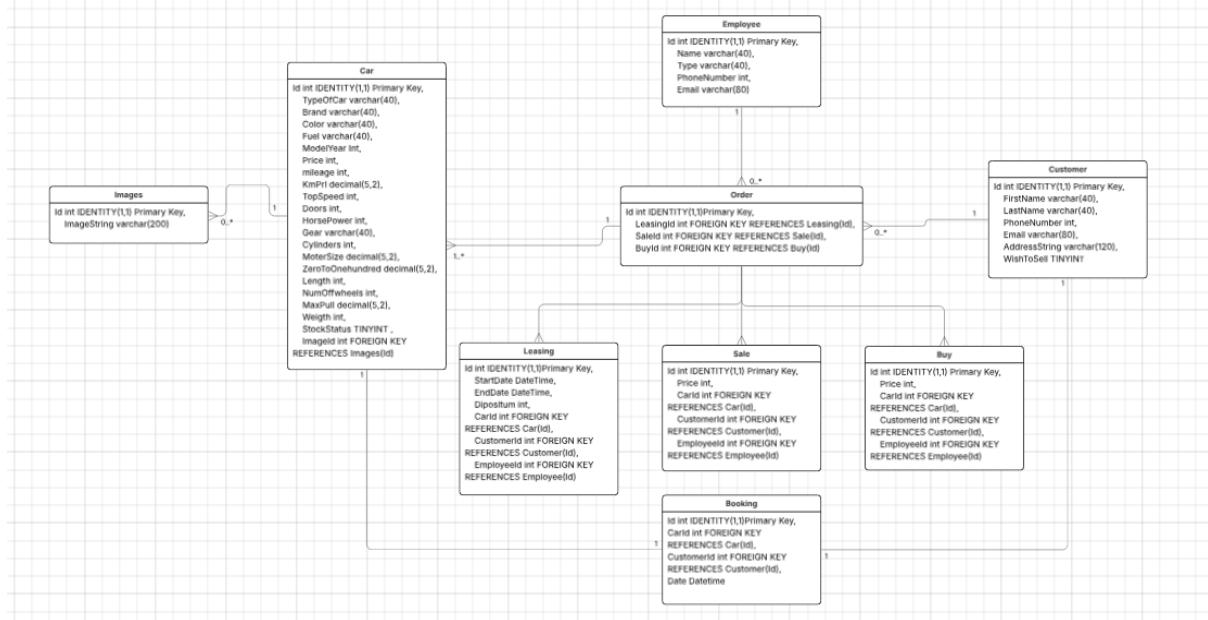


Bilag 21: VCA - Primary Activities



ER Diagram (Kronborg)

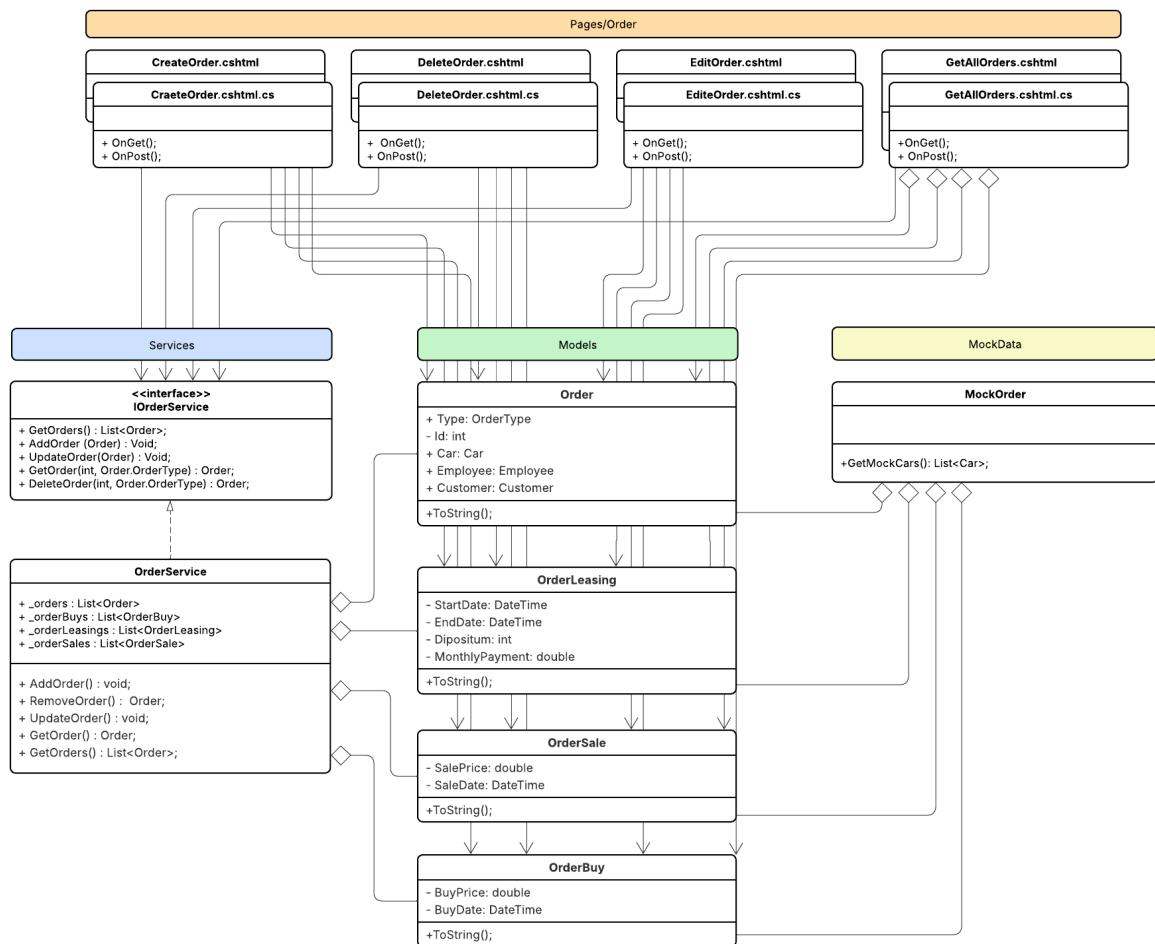
Bilag 22: ER Diagram - sprint 1



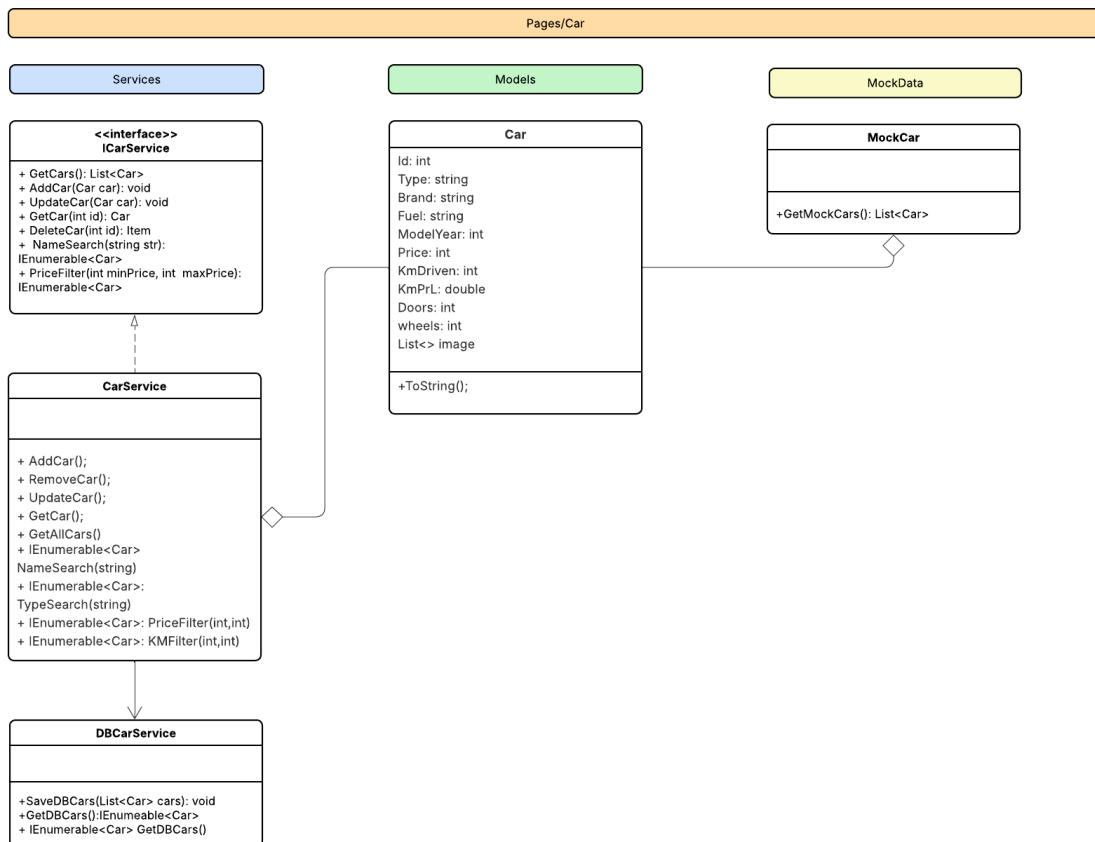
Razor Page Diagrammer

Sprint 1

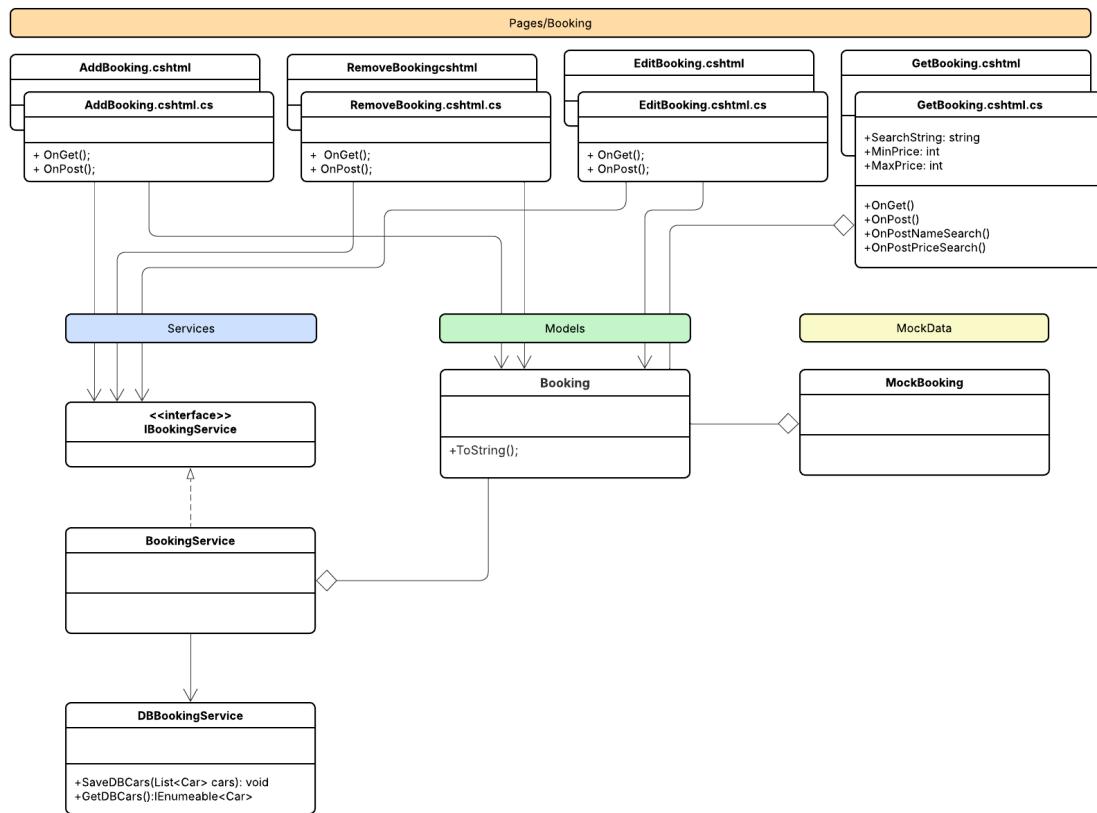
Bilag 23: Pages/Order



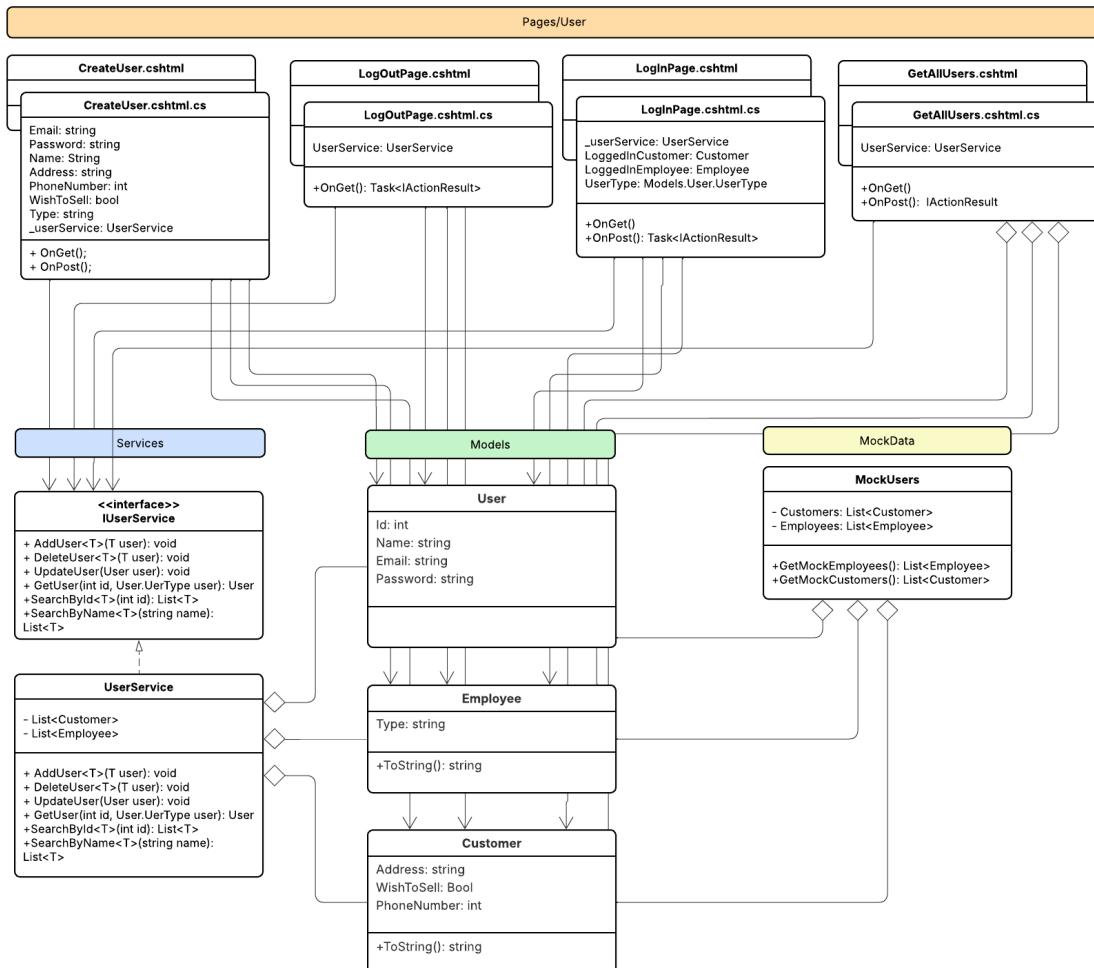
Bilag 24: Pages/Car



Bilag 25: Pages/Booking

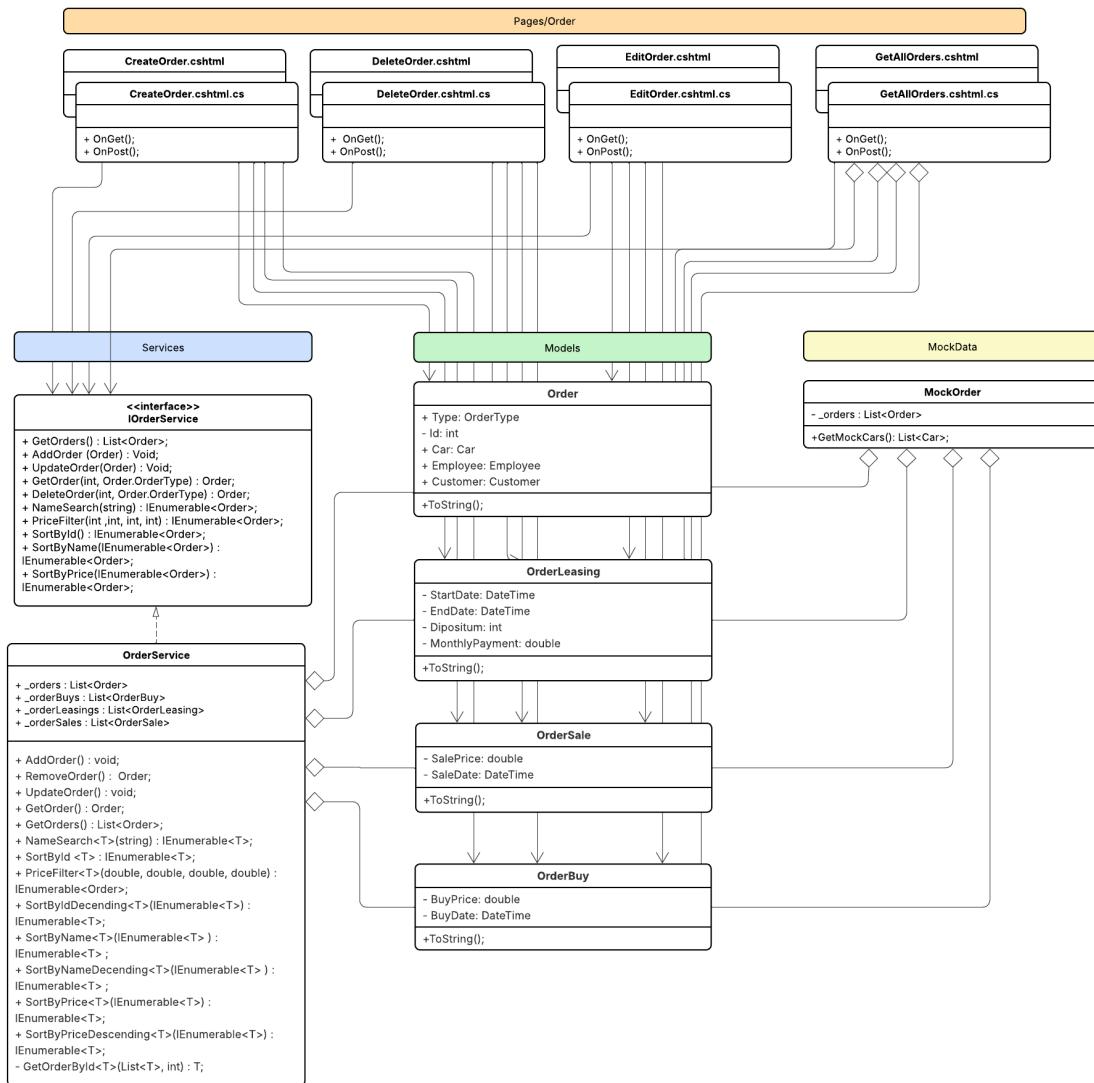


Bilag 26: Pages/User

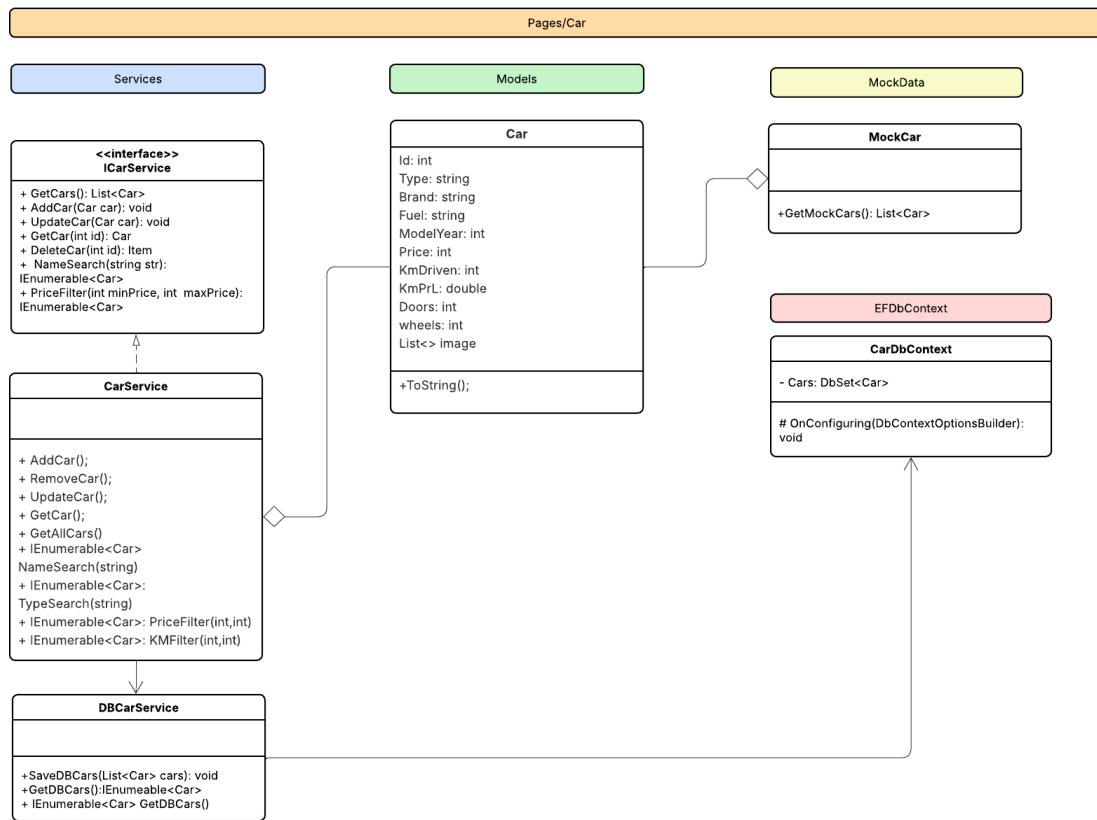


Sprint 2

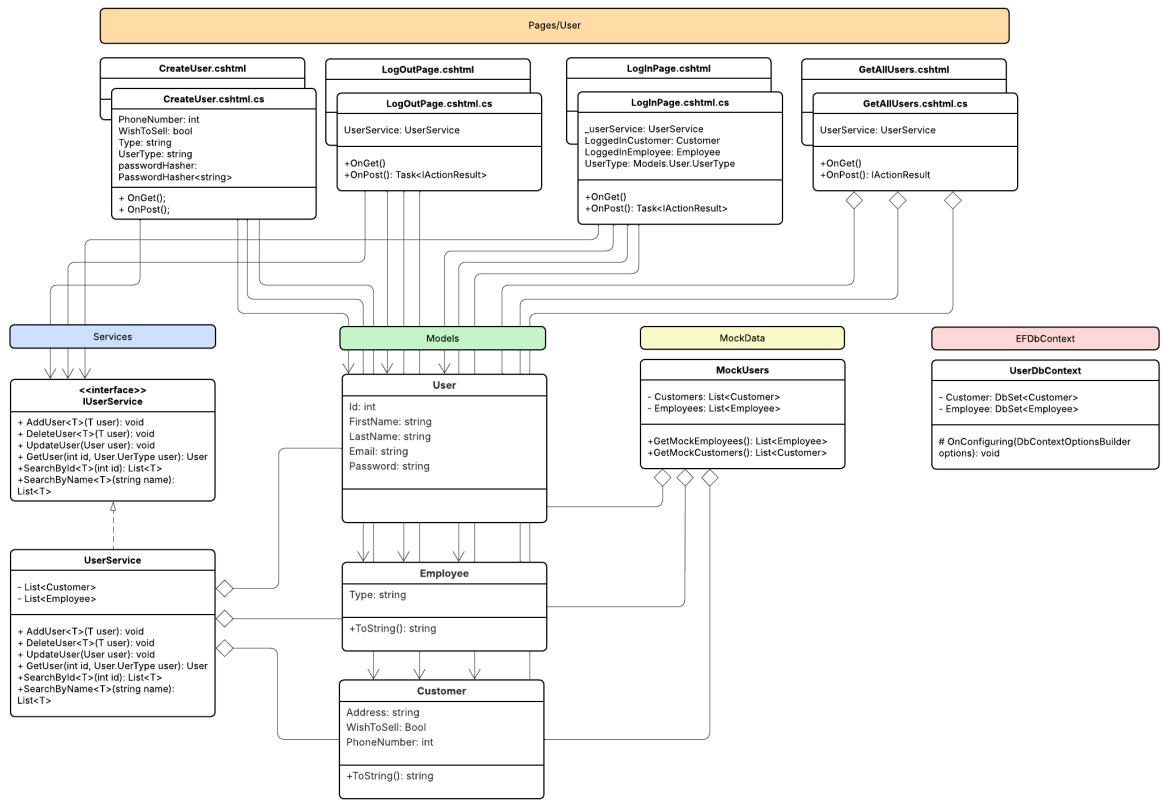
Bilag 27: Pages/Order



Bilag 28: Pages/Car

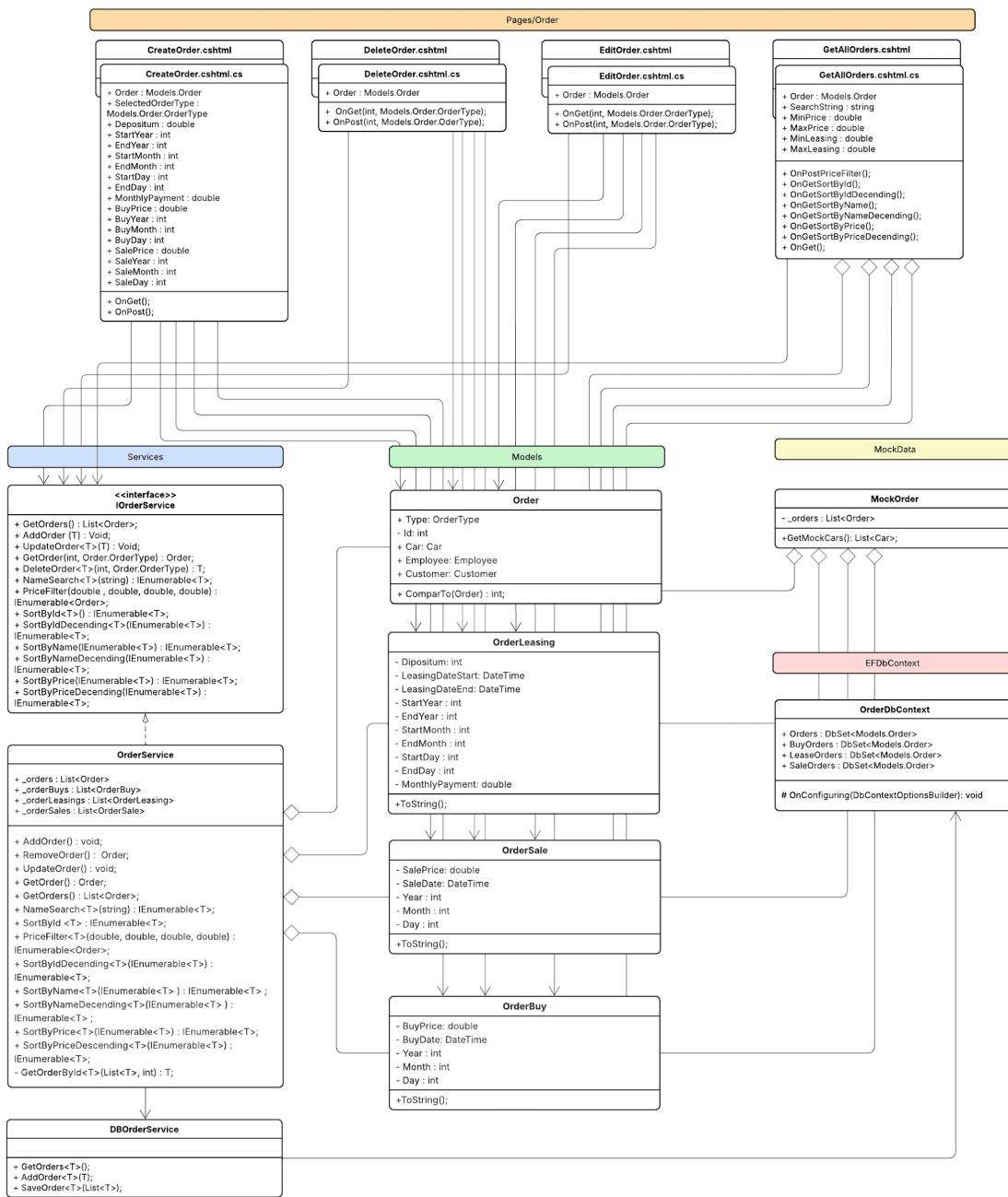


Bilag 29: Pages/User

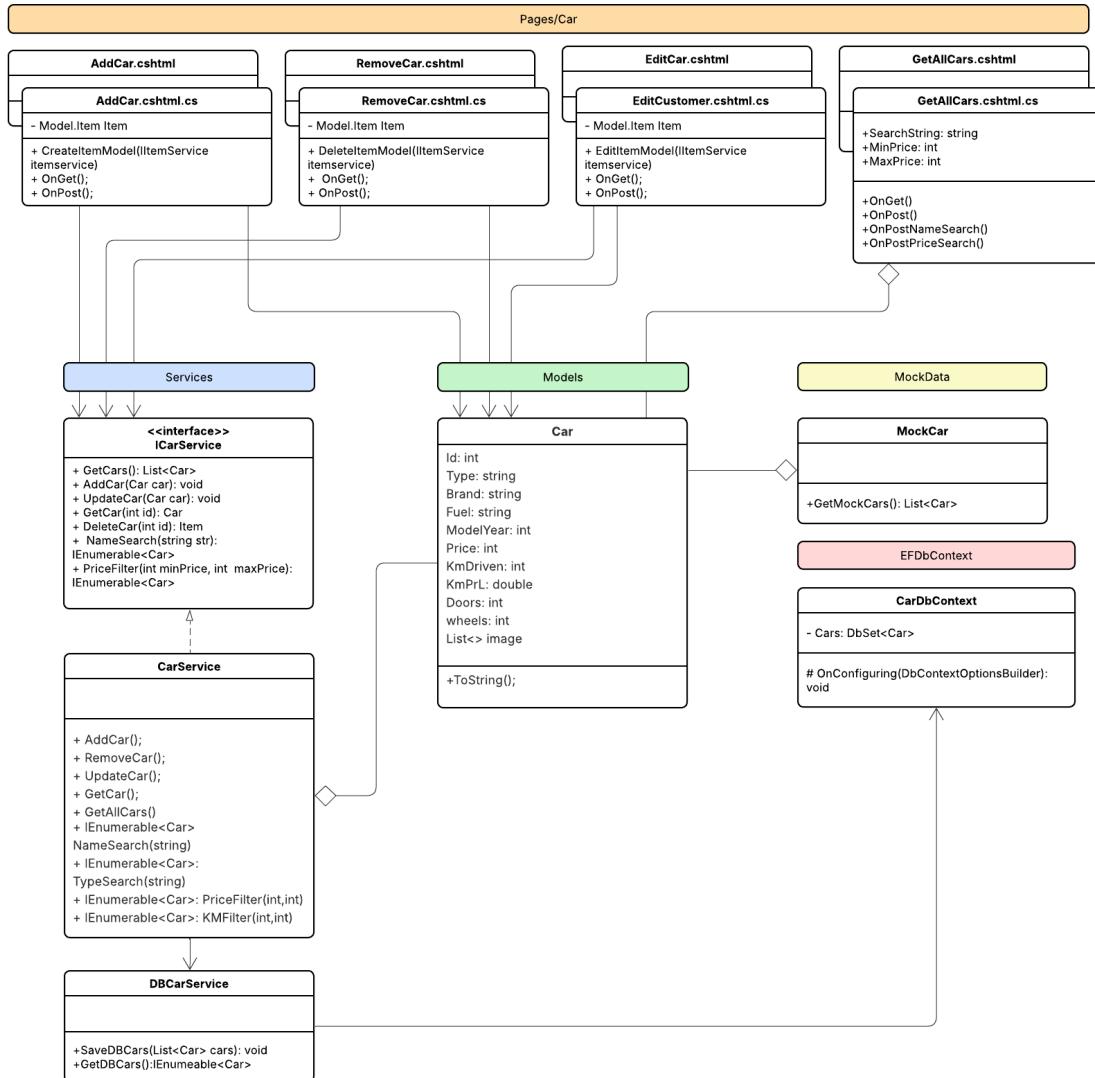


Sprint 3

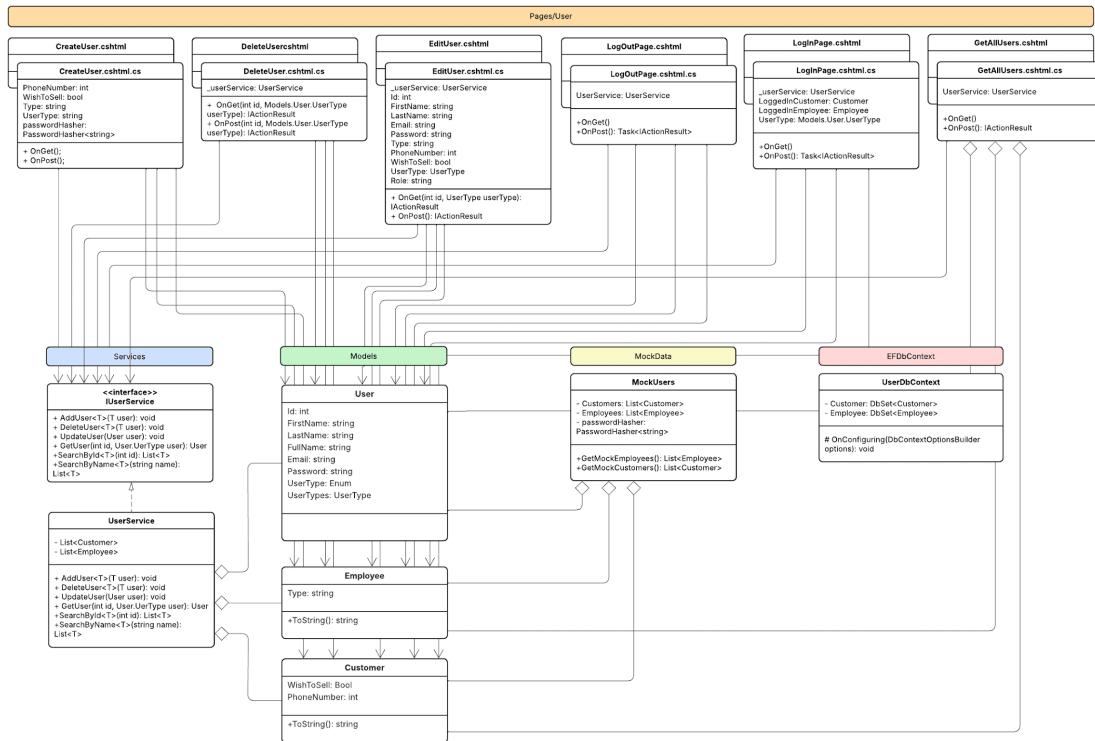
Bilag 30: Pages/Order



Bilag 31: Pages/Car

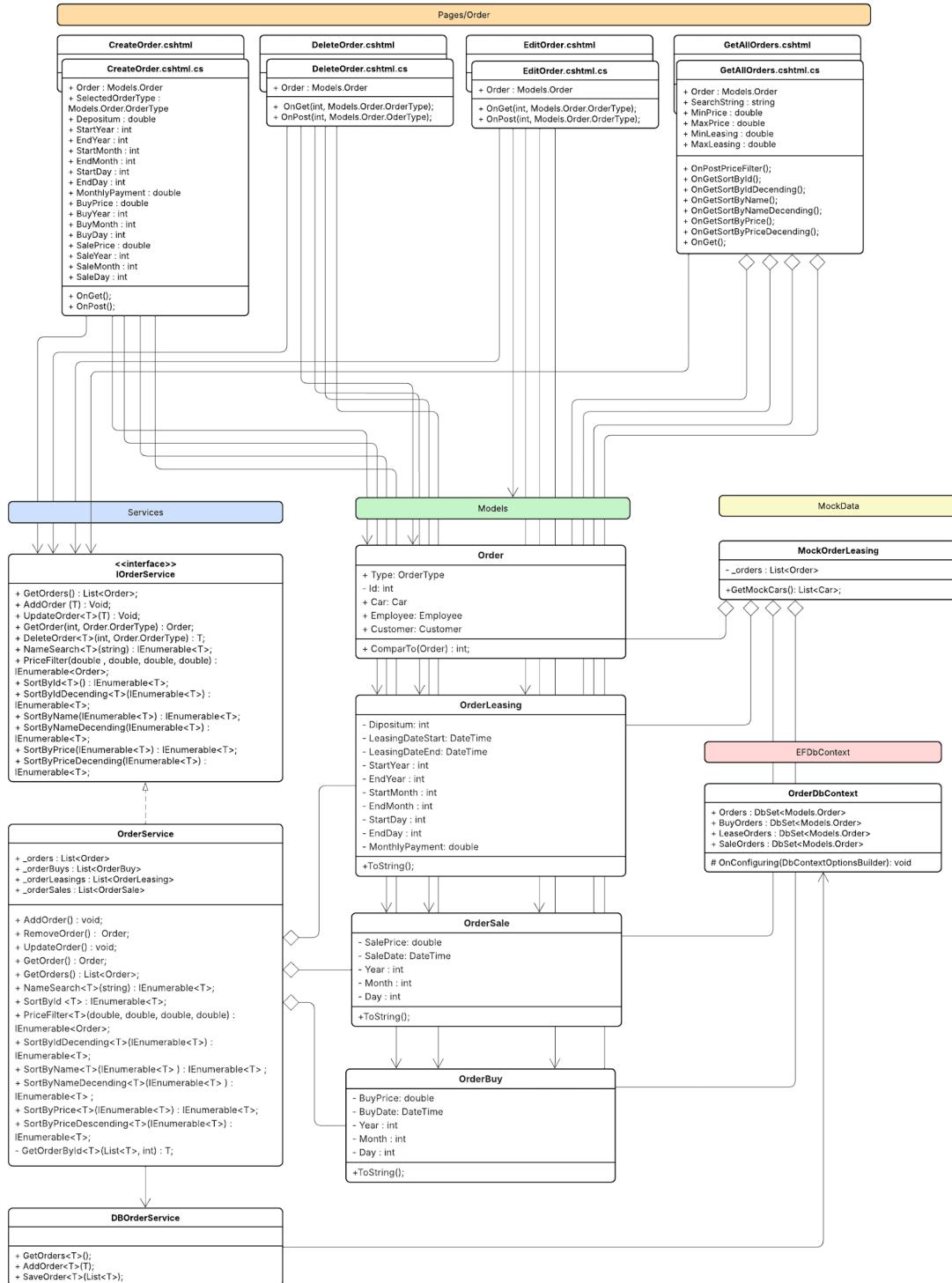


Bilag 32: Pages/User

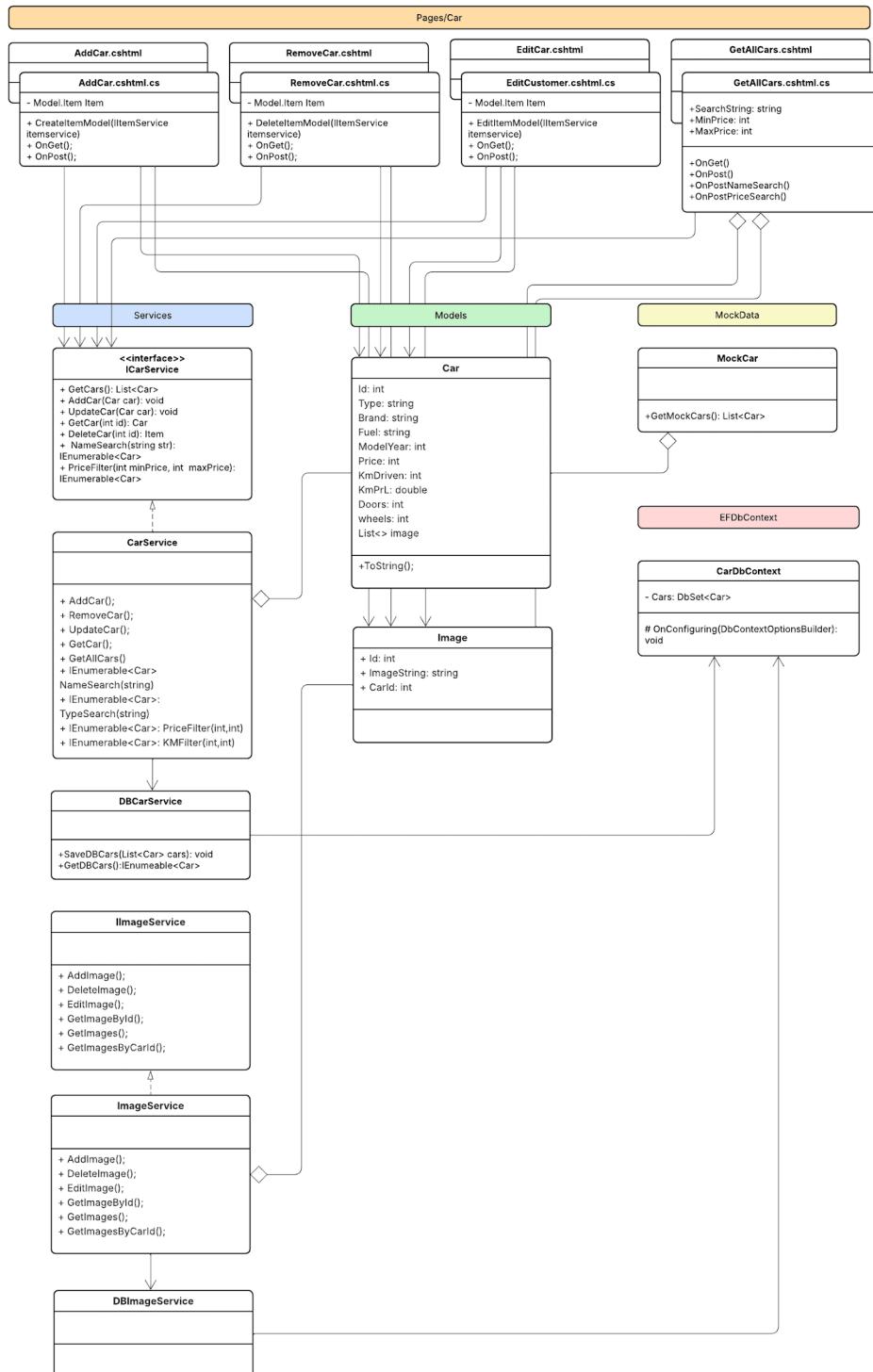


Sprint 4

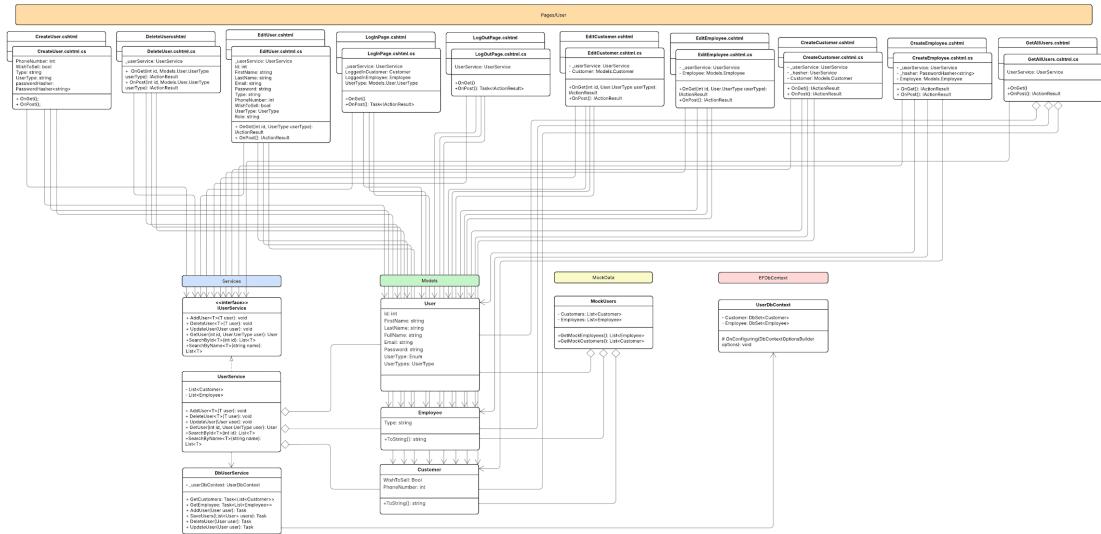
Bilag 33: Pages/Order



Bilag 34: Pages/Car

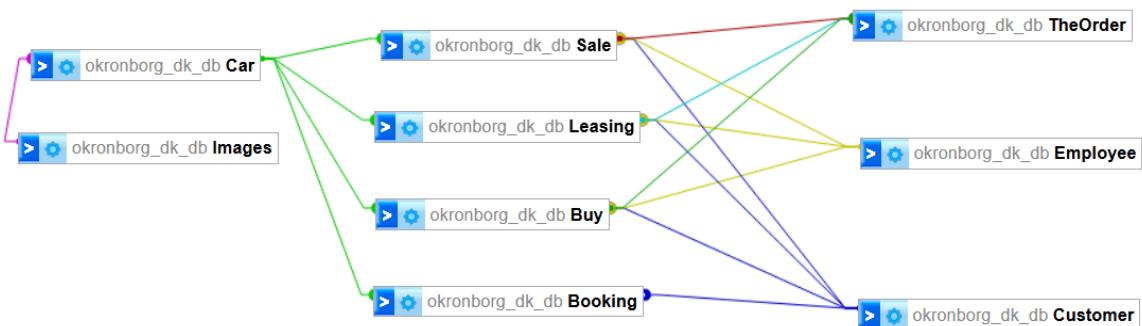


Bilag 35: Pages/User



Database (Kronborg)

Bilag 36: Tabel diagram - sprint 1



```
-- phpMyAdmin SQL Dump
-- version 5.2.1
-- https://www.phpmyadmin.net/
--
-- Vært: mysql62.unoeuro.com
-- Genereringstid: 28. 05 2025 kl. 12:40:42
```

```
-- Serverversion: 8.0.37-29
-- PHP-version: 8.1.29

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

-- 
-- Database: `okronborg_dk_db`
--



-----



-- 
-- Struktur-dump for tabellen `Booking`


-----



CREATE TABLE `Booking` (
  `Id` int NOT NULL,
  `CarId` int DEFAULT NULL,
  `CustomerId` int DEFAULT NULL,
  `Date` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;



-----



-- 
-- Struktur-dump for tabellen `Buy`


-----



CREATE TABLE `Buy` (
  `Id` int NOT NULL,
  `Price` int DEFAULT NULL,
  `CarId` int DEFAULT NULL,
  `CustomerId` int DEFAULT NULL,
  `EmployeeId` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;



-----




```

```
-- Struktur-dump for tabellen `Car`
-- 

CREATE TABLE `Car` (
  `Id` int NOT NULL,
  `TypeOfCar` varchar(40) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT
NULL,
  `Brand` varchar(40) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `Color` varchar(40) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `Model` varchar(40) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `Fuel` varchar(40) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `ModelYear` int NOT NULL,
  `Price` int NOT NULL,
  `Mileage` int NOT NULL,
  `KmPrl` decimal(5,2) NOT NULL,
  `TopSpeed` int NOT NULL,
  `Doors` int NOT NULL,
  `HorsePower` int NOT NULL,
  `Gear` varchar(40) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `Cylinders` int NOT NULL,
  `MotorSize` decimal(5,2) NOT NULL,
  `ZeroToOnehundred` decimal(5,2) NOT NULL,
  `Length` int NOT NULL,
  `NumOffwheels` int NOT NULL,
  `MaxPull` decimal(5,2) NOT NULL,
  `Weight` int NOT NULL,
  `StockStatus` tinyint DEFAULT NULL,
  `Status` tinyint DEFAULT NULL,
  `PriceMonth` int DEFAULT NULL,
  `LeasingPeriod` int DEFAULT NULL,
  `KmlIncluded` int DEFAULT NULL,
  `ForSale` tinyint(1) NOT NULL DEFAULT '1'
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
-- 
-- Struktur-dump for tabellen `Customers`
-- 

CREATE TABLE `Customers` (
  `Id` int NOT NULL,
  `FirstName` varchar(100) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT
NULL,
  `LastName` varchar(100) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT
NULL,
  `PhoneNumber` int NOT NULL,
  `Email` varchar(100) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
```

```
'WishToSell` tinyint NOT NULL,
`Password` varchar(210) CHARACTER SET latin7 COLLATE latin7_general_cs NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Struktur-dump for tabellen `Employees`
```

```
CREATE TABLE `Employees` (
`Id` int NOT NULL,
`FirstName` varchar(100) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT
NULL,
`LastName` varchar(100) NOT NULL,
`Type` varchar(40) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
`Email` varchar(80) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
`Password` varchar(210) CHARACTER SET latin7 COLLATE latin7_general_cs NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Struktur-dump for tabellen `Image`
```

```
CREATE TABLE `Image` (
`Id` int NOT NULL,
`ImageString` varchar(255) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT
NULL,
`CarId` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Struktur-dump for tabellen `Leasing`
```

```
CREATE TABLE `Leasing` (
`Id` int NOT NULL,
`StartDate` datetime DEFAULT NULL,
`EndDate` datetime DEFAULT NULL,
`Dipositum` int DEFAULT NULL,
`CarId` int DEFAULT NULL,
`CustomerId` int DEFAULT NULL,
`EmployeeId` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Struktur-dump for tabellen `Sale`  
  
CREATE TABLE `Sale` (  
    `Id` int NOT NULL,  
    `Price` int DEFAULT NULL,  
    `CarId` int DEFAULT NULL,  
    `CustomerId` int DEFAULT NULL,  
    `EmployeeId` int DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Struktur-dump for tabellen `TheOrder`  
  
CREATE TABLE `TheOrder` (  
    `Id` int NOT NULL,  
    `LeasingId` int DEFAULT NULL,  
    `SaleId` int DEFAULT NULL,  
    `BuyId` int DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Begrensninger for dumpede tabeller
```

```
--  
-- Indeks for tabel `Booking`
```

```
--  
ALTER TABLE `Booking`  
    ADD PRIMARY KEY (`Id`),  
    ADD KEY `CarId` (`CarId`),  
    ADD KEY `CustomerId` (`CustomerId`);
```

```
--  
-- Indeks for tabel `Buy`
```

```
--  
ALTER TABLE `Buy`  
    ADD PRIMARY KEY (`Id`),  
    ADD KEY `CarId` (`CarId`),  
    ADD KEY `CustomerId` (`CustomerId`),  
    ADD KEY `EmployeeId` (`EmployeeId`);
```

```
--  
-- Indeks for tabel `Car`  
  
--  
ALTER TABLE `Car`  
ADD PRIMARY KEY (`Id`);  
  
--  
-- Indeks for tabel `Customers`  
  
--  
ALTER TABLE `Customers`  
ADD PRIMARY KEY (`Id`);  
  
--  
-- Indeks for tabel `Employees`  
  
--  
ALTER TABLE `Employees`  
ADD PRIMARY KEY (`Id`);  
  
--  
-- Indeks for tabel `Image`  
  
--  
ALTER TABLE `Image`  
ADD PRIMARY KEY (`Id`),  
ADD KEY `CarId` (`CarId`);  
  
--  
-- Indeks for tabel `Leasing`  
  
--  
ALTER TABLE `Leasing`  
ADD PRIMARY KEY (`Id`),  
ADD KEY `CarId` (`CarId`),  
ADD KEY `CustomerId` (`CustomerId`),  
ADD KEY `EmployeeId` (`EmployeeId`);  
  
--  
-- Indeks for tabel `Sale`  
  
--  
ALTER TABLE `Sale`  
ADD PRIMARY KEY (`Id`),  
ADD KEY `CarId` (`CarId`),  
ADD KEY `CustomerId` (`CustomerId`),  
ADD KEY `EmployeeId` (`EmployeeId`);  
  
--  
-- Indeks for tabel `TheOrder`  
  
--  
ALTER TABLE `TheOrder`
```

```
ADD PRIMARY KEY (`Id`),  
ADD KEY `LeasingId` (`LeasingId`),  
ADD KEY `SaleId` (`SaleId`),  
ADD KEY `BuyId` (`BuyId`);  
  
--  
-- Brug ikke AUTO_INCREMENT for slettede tabeller  
--  
  
--  
-- Tilføj AUTO_INCREMENT i tabel `Booking`  
--  
ALTER TABLE `Booking`  
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;  
  
--  
-- Tilføj AUTO_INCREMENT i tabel `Buy`  
--  
ALTER TABLE `Buy`  
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;  
  
--  
-- Tilføj AUTO_INCREMENT i tabel `Car`  
--  
ALTER TABLE `Car`  
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;  
  
--  
-- Tilføj AUTO_INCREMENT i tabel `Customers`  
--  
ALTER TABLE `Customers`  
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;  
  
--  
-- Tilføj AUTO_INCREMENT i tabel `Employees`  
--  
ALTER TABLE `Employees`  
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;  
  
--  
-- Tilføj AUTO_INCREMENT i tabel `Image`  
--  
ALTER TABLE `Image`  
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;  
  
--  
-- Tilføj AUTO_INCREMENT i tabel `Leasing`  
--
```

```
ALTER TABLE `Leasing`
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;

-- 
-- Tilføj AUTO_INCREMENT i tabel `Sale`
-- 

ALTER TABLE `Sale`
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;

-- 
-- Tilføj AUTO_INCREMENT i tabel `TheOrder`
-- 

ALTER TABLE `TheOrder`
    MODIFY `Id` int NOT NULL AUTO_INCREMENT;

-- 
-- Begrænsninger for dumpede tabeller
-- 

-- 
-- Begrænsninger for tabel `Booking`
-- 

ALTER TABLE `Booking`
    ADD CONSTRAINT `Booking_ibfk_1` FOREIGN KEY (`CarId`) REFERENCES `Car` (`Id`),
    ADD CONSTRAINT `Booking_ibfk_2` FOREIGN KEY (`CustomerId`) REFERENCES
`Customers` (`Id`);

-- 
-- Begrænsninger for tabel `Buy`
-- 

ALTER TABLE `Buy`
    ADD CONSTRAINT `Buy_ibfk_1` FOREIGN KEY (`CarId`) REFERENCES `Car` (`Id`),
    ADD CONSTRAINT `Buy_ibfk_2` FOREIGN KEY (`CustomerId`) REFERENCES
`Customers` (`Id`),
    ADD CONSTRAINT `Buy_ibfk_3` FOREIGN KEY (`EmployeeId`) REFERENCES
`Employees` (`Id`);

-- 
-- Begrænsninger for tabel `Image`
-- 

ALTER TABLE `Image`
    ADD CONSTRAINT `Image_ibfk_1` FOREIGN KEY (`CarId`) REFERENCES `Car` (`Id`)
ON DELETE CASCADE;
```

MySQL connection: string connStr =
"Server=mysql62.unoeuro.com;Port=3306;Database=okronborg_dk_db;User
ID=okronborg_dk;Password=gnb6xtyDdc3eafE9zkrh;";