

Førsteårsprøve
DMVE231 - Datamatiker 2. Semester

Skole: UCL Erhvervsakademi og Professionshøjskole -
Boulevarden

Titel: IT-System til LawHouse



Undervisere:
Henrik Steen Krogh
Brian Soltau
Christina Rehmeier
Per Larsen

Skrevet af : Bilal Kinalli, Lucas Ordell MacQuarrie & Rasmus Skov

Afleveret d. 31/05-2024

Anslag: 107883

Indledning.....	5
Problemstilling.....	5
Problemformulering.....	5
Tilgangsvinkler.....	6
Projektledelse.....	6
Scrum (Lucas).....	6
Atlassian/Jira (Lucas).....	7
Risikoanalyse (Lucas).....	7
Pair Programming (Lucas).....	8
Før start (Lucas).....	9
Under sprints (Lucas).....	9
Sprint 0: D. 22 - 25 (Lucas).....	9
Sprint retrospective (Lucas).....	11
Sprint 1: D. 25 - 02 (Lucas).....	12
Sprint Retrospective (Lucas).....	13
Sprint 2: D. 02 - 09 (Lucas).....	14
Sprint Retrospective (Lucas).....	15
Sprint 3: D. 09 - 16 (Lucas).....	16
Sprint retrospective (Lucas).....	17
Sprint 4: D. 16 - 23 (Lucas).....	17
Sprint retrospective (Lucas).....	18
Sprint 5 D. 23 - 30 (Lucas).....	18
Sprint retrospective (Lucas).....	19
IT-Værktøjer.....	19
Github (Bilal).....	19
Microsoft Visual Studio (Bilal).....	19
Microsoft SQL Server (Bilal).....	20
Framework & NuGet Packages (Bilal).....	20
Discord (Lucas).....	21
Analyse.....	22
Use-Cases (Lucas).....	22
Fully Dressed Use Case (Rasmus).....	22
Use Case Diagram (Rasmus).....	24
System Sekvens Diagrammer (Rasmus & Bilal).....	25
Domæne Model (Rasmus).....	28
Ikke-funktionelle krav (Bilal, Lucas & Rasmus).....	29
Design.....	30
Systemets Arkitektur (Rasmus).....	30
Pakke Diagram (Lucas).....	32
Klasse Diagram (Rasmus).....	33
Sekvensdiagrammer (Lucas & Rasmus).....	34
Entity Relationship Diagram (Rasmus).....	36

Mapping af ERD (Lucas).....	38
Database.....	38
Database udbyder (Bilal).....	38
App.Config (Bilal).....	38
Entity Framework (Lucas).....	39
Table per type - TPT (Lucas).....	39
SQL Client (Lucas).....	40
Trigger (Rasmus).....	43
Overvejelse til databasekald.....	44
.Include med Entity Framework (Lucas).....	44
Program C#.....	45
Asynkron Programmering (Rasmus).....	45
Modeller og konvertering (Rasmus).....	46
Validering (Rasmus).....	47
Postnummer/By fra CSV (Bilal, Lucas & Rasmus).....	49
Login (Bilal).....	49
Oprettelse som kunde (Bilal).....	52
Navigering i forsiden (Lucas, Bilal, Rasmus).....	57
Håndtering af Bruger Privilegier (Lucas, Bilal, Rasmus).....	58
Advokater.....	62
Oprettelse af advokat (Bilal).....	62
Min side (Bilal).....	66
Visning af ansatte (Bilal).....	68
Håndtering af sager (Rasmus).....	72
Oprettelse af sag (Rasmus).....	72
Tilføj ydelse til sag (Rasmus).....	77
Tilføj timer til ydelse (Rasmus).....	80
Afslutning af sag og ydelser (Rasmus).....	84
Klientens oplevelse (Lucas).....	86
Opfølgning på sine sager (Lucas).....	86
Oprettelse af Abonnement (Lucas).....	89
Beregninger (Lucas).....	92
Køb af formularer (Lucas).....	94
Oversigt over advokater (Bilal).....	96
Distance Matrix API (Bilal, Lucas & Rasmus).....	98
Rapport til tekstfil (Bilal, Lucas & Rasmus).....	100
Åbning af pdf (Bilal, Lucas & Rasmus).....	102
NUnit Unit Test (Bilal, Lucas & Rasmus).....	103
Overvejelser i programmet.....	106
Instanser → Dependency Injection/Singleton (Rasmus).....	106
Kun få delete funktioner (Lucas).....	106
Brug af Enums (Lucas).....	107
Brugervenlighed.....	108
FURPS+ (Lucas, Bilal & Rasmus).....	108

Gestaltlove (Lucas).....	108
Brugertest/BlackBox test (Lucas, Bilal).....	109
Bidrag til koden.....	110
Konklusion.....	111
Perspektivering.....	112
Litteraturliste.....	112
Bilag.....	113
Use Cases.....	113
Scrum Board.....	115
Sprint 1.....	115
Sprint 2.....	117
User Stories.....	120
Sprint 3.....	121
User Stories.....	124
Sprint 4.....	125
Risikoanalyse.....	126
Sprint 1.....	126
Sprint 2.....	128
Sprint 3.....	130
Sprint 4.....	132
Sprint 5.....	134
Screenshots af program design.....	135
LoginPageView CreateUserView.....	135
LawyersOverview(for clients).....	136
AdminPage.....	136
MyPage(Lawyer).....	137
AdminCreateLawyer.....	137
EmployeesOverview.....	138
LawyersOverview.....	138
Forside.....	139
My Page klient.....	139
CaseDetailsView klient.....	140
CaseDetailsView advokat/sekretær.....	140
CreateCasePage.....	141
ServiceDetailsForm.....	142
AddLawyerView.....	142
DistanceCalculator.....	143
Subscription View.....	143
Beregning af ydelse form.....	144
Formularer.....	144

Indledning

Denne rapport omhandler udviklingsprocessen af et IT-system til advokatfirmaet LawHouse. Der er under udviklingen blevet arbejdet med den agile arbejdsproces SCRUM. Rapporten er struktureret således, at der først præsenteres vores arbejdsproces, efterfulgt af analyse og design. Dernæst vil rapporten præsentere de væsentlige implementationer og undervejs diskutere resultater og alternativer. Afslutningsvis evalueres resultaterne, og vi perspektiverer til fremtidige forbedringer af systemet.

Problemstilling

Advokatfirmaet LawHouse er i vækst og oplever store udfordringer med at håndtere og optimere deres arbejdsprocesser. Deres nuværende system er utilstrækkeligt til at registrere ydelser på sager, administrere klient- og medarbejderdata samt understøtte advokaternes efteruddannelse.

Derudover stiger behovet for at tilbyde kunderne selvbetjeningsmuligheder, såsom beregningsværktøjer, køb af formularer og evt. tilgå sine sager. For at imødekomme disse behov og forbedre arbejdsgangene, skal der udvikles et nyt IT-system, hvilket skal være fleksibelt, brugervenligt og sikre datasikkerhed og ydeevne.

Problemformulering

Hvordan kan det kommende IT-System være med til at øge effektiviteten for firmaets sags- og kundehåndtering?

Hvilke funktioner skal der implementeres i IT-Systemet for at automatisere arbejdsgangene for LawHouse?

Hvilke features skal systemet indeholde for at forbedre selvbetjeningsmuligheder for kunder?

Hvordan kan SCRUM anvendes til effektiv projektstyring under udvikling af dette IT-System?

Hvordan kan vi sikre at systemet er brugervenligt, fleksibelt og fremtidssikret?

Tilgangsvinkler

Igennem vores rapport vil vi, ved brug af SCRUM og vores IT-Værktøjer, beskrive vores tilgangsvinkler til udviklingen af dette projekt.

Projektledelse

Scrum (Lucas)

Vi har i vores gruppe valgt at bruge SCRUM til udvikling af vores produkt. SCRUM er en agil arbejdsmetode. Til udvikling af it-produkter er det rigtigt smart at gøre brug af agile arbejdsprocesser, da det giver fleksibilitet og hurtig tilpasning af produktet. SCRUM tilbyder en god struktureret ramme for projektledelsen da der er nogle møder som skal ”overholdes” heriblandt:

Sprint planning:

Før hvert sprint er gruppen samlet, planlægger og beslutter hvad der skal laves og hvem der laver hvad.

Daily scrum møde:

Et dagligt møde hvor arbejde koordineres og justeres, dette møde er med til at vi hurtigt kan identificere hindringer.

Sprint retrospective:

Til sidst for hvert sprint kommer sprint retrospective hvor vi får reflekteret over sprintet, hvad gik godt/dårligt så vi kan gøre det endnu bedre sprintet efter.

Derudover tilbyder scrum også nogle værdifulde artefakter som hjælper med projektledelsen

Product backlog:

Product backloggen er en liste over alle features som skal implementeres i vores produkt, den danner et godt overblik over hvad programmet egentlig skulle kunne. Samt at en potentiel produkt owner også kan se hvad der egentligt bliver udarbejdet.

Sprint Backlog:

Ved sprint planning gør vi alle i gruppen overvejelse om hvor meget vi kan nå at lave fra hele produkt backloggen og ligger det over i vores sprint backlog dette giver et godt overblik over hvad der skal gøres færdig indenfor sprintet.

Atlassian/Jira (Lucas)

Vi har til håndtering af SCRUM , herunder Product backlog, Sprint backlogs og scrumboards brugt Atlassian Jira software. Jira giver mulighed for at oprette de enkelte sprints med start og slut tid og hvert sprint har sin egen sprint backlog. Så er der selvfølgelig den overordnede product backlog hvorfra vi nemt kan hente de enkelte user stories ind i de enkelte sprints. Derudover tilbyder Jira en masse andre features, heriblandt rapporter og oversigter, hvor man nemt kan følge med i hvor langt vi egentlig er nået med hele projektet.

Risikoanalyse (Lucas)

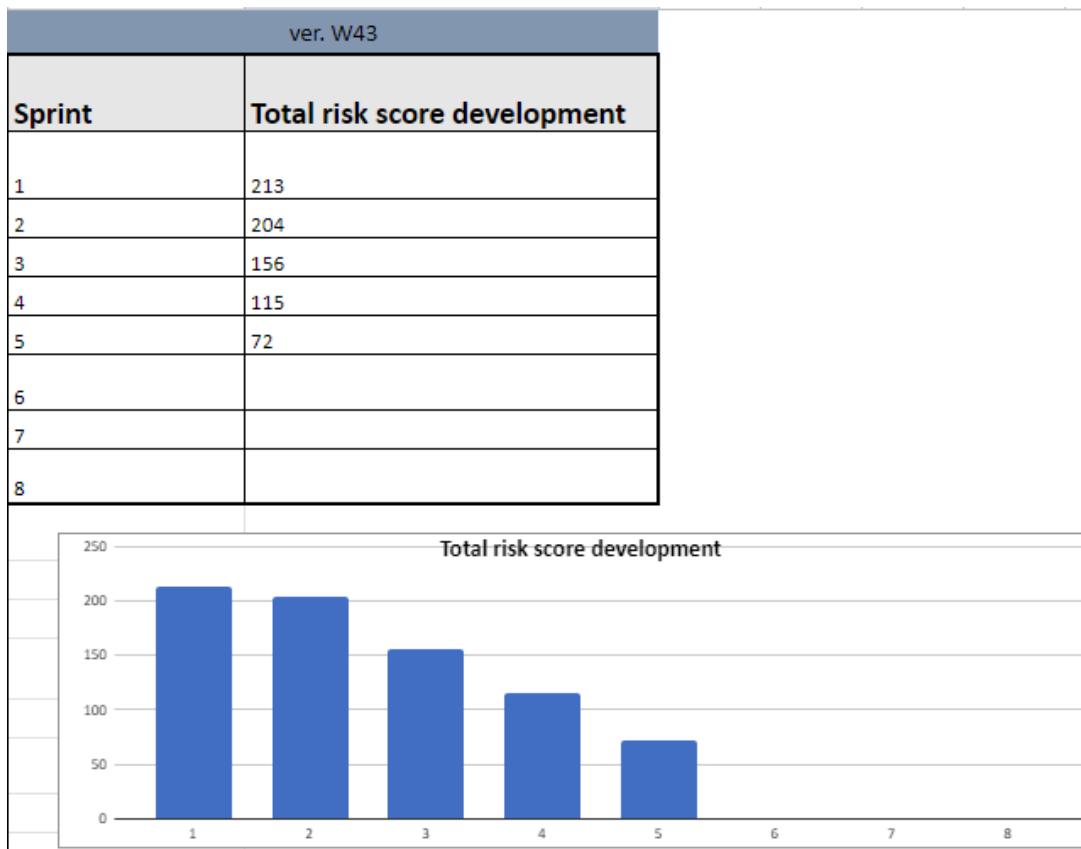
Ledelsen har efterspurgt en risikoanalyse, så de har en ide om hvilke risici der er involveret i udviklingen af projektet. Herunder ses et billede af de risici vi mener der indgår i udviklingen af produktet(Se Figur 1,1).

Risk Analysis - Matrix				
ID	Risk: What can go wrong?	Probability (1-5)	Consequence / impact (1-5)	Risk score
1	Teknisk risiko : Databasen er utilgængelig	4	5	20
2	Teknisk risiko : Brug af nuget-pakker	1	2	2
3	Teknisk risiko : Exception håndtering	5	4	20
4	Teknisk risiko : Konflikter ved merging	3	2	6
5	Teknisk risiko : SQL injection	4	5	20
6	Teknisk risiko : Fejl i model-konverteringer	2	3	6
7	Teknisk risiko : Fejhåndtering af brugerprivilegier	4	5	20
8	Ressourcemæssig risiko: Mangel på tilstrækkelig tid eller ressourcer til at implementere eller teste systemet	4	5	20
9	Ressourcemæssig risiko: Overestimering af teamets evne og tid til at efterkomme kravene i hver sprints	5	3	15
10	Ressourcemæssig risiko: Brug af for mange ressourcer på "Nice to have"	5	5	25
11	Ressourcemæssig risiko : Fraværende gruppe medlem ved sygdom	4	4	16
12	Ressourcemæssig risiko : For meget arbejde ved siden af skole, arbejde forskudt af hinanden	4	2	8
13	Kvalitets risiko: Principper for "Clean Code" overholdes ikke	4	2	8
14	Kvalitets risiko: Ustrukturert og dårlig design af kode	3	4	12
15				0
16				0
17	Usikre krav: Fejlfortolkning af Case og opgaven	3	5	15
				213

Figur 1,1 - Risikoanalyse, Risici

Billede af hele risiko analyse med Preventative og Mitigation action ses her¹

¹ [Risikoanalyse - Sprint 1](#)



Figur 1,2 - Total Risk score development

Hvis vi kigger på risk score development igennem hele projektet (figur 1,2) Kan man se at mellem sprint 1 og 2 ikke er den store udvikling. Grunden til dette er at vi endnu ikke på det tidspunkt er gået igang med at kode selve programmet. Dette starter ved sprint 2 og der ses en større udvikling pr sprint derefter.

Pair Programming (Lucas)

Pair Programming fra Extreme Programming har været en stor del af vores projekt. Vi har oftest sat på discord og snakket sammen. Discord har den smarte funktion, at man kan dele sin skærm. Så hvis noget mere tungt kode er blevet skrevet eller man har haft spørgsmål til små dele, har vi delt skærme og løst problemerne sammen. Derudover har vi hver især sat med mere end en skærm og programmeret, ofte har vi haft hinandens skærmedelinger til at køre på secondary skærm, så imens vi selv har kodet vores egne dele af programmet, har vi fulgt godt med i hvad hinanden præcist lavede samtidigt og kunne rigtig nemt hurtigt spørge hinanden om hjælp. Vi har i gruppen været rigtig taknemmelige for Pair programming.

Før start (Lucas)

Før starten på første sprint (sprint 0) har vi som gruppe sat sammen læst casen igennem og delt den op i mindre bider, fundet ud af hvad det er der bliver bedt om og derfra har vi lavet en Epic ved navn “To-Do” hvortil vi har tilføjet issues for alt hvad vi forventede der skulle laves. Herunder udformning af skitser, udarbejdelse af user stories, tegning af modeller osv. Alt der ikke indgår til direkte kodning af programmet og der ikke laves user stories på.

Dette gav os et godt grundlag for hvor vi kunne starte og sørgede for at vi fra starten fik et godt overblik over hvad der skulle laves til dette projekt.

Under sprints (Lucas)

Vi har under hele projektet ikke mødtes hver dag fysisk. Vi har mødtes de dage hvor vi har været til vejledning. Men vi har hver dag sat sammen på discord og snakket sammen under udvikling af produktet, hvorfor vi også har gjort stor brug af Pair programming fra XP.

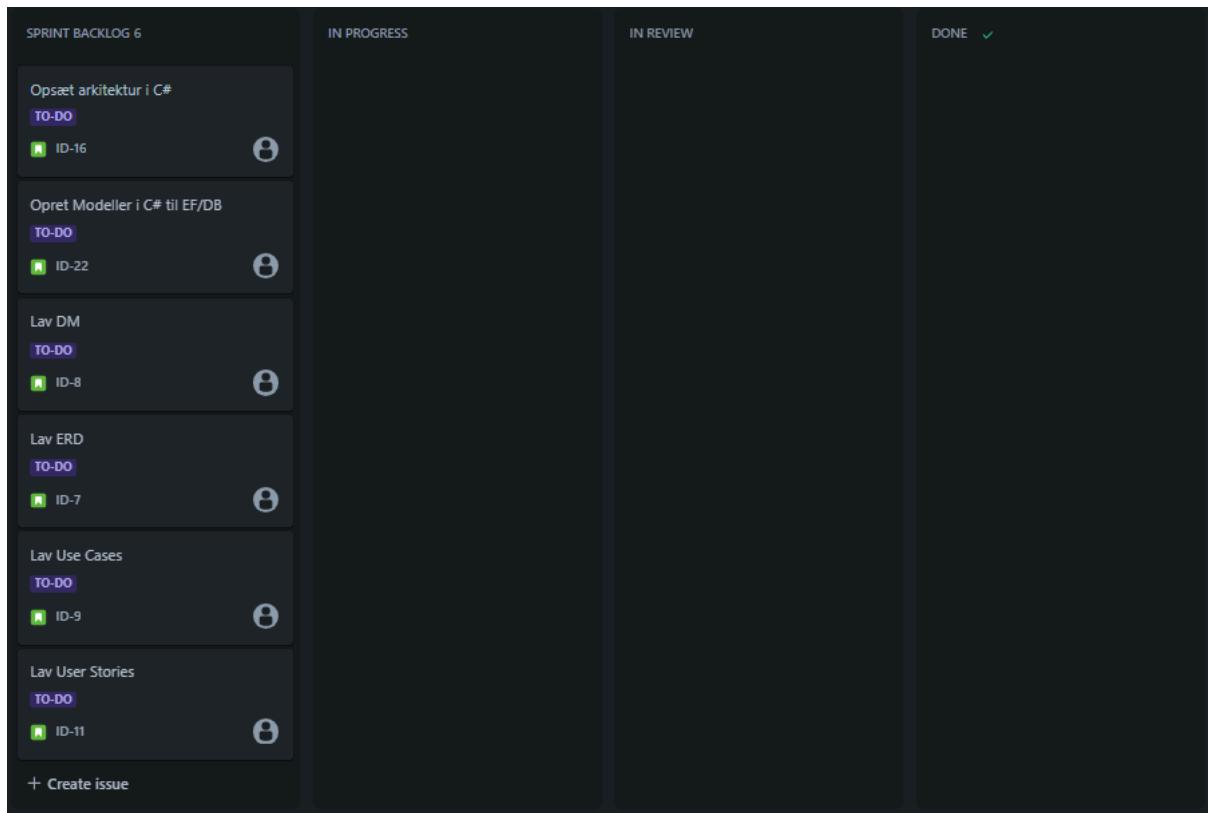
Vi har undervejs i alle sprints afholdt vores “daily standup meetings” hvor vi kort har gennemgået hvad der er blevet lavet dagen før hvad der skal arbejdes på for den gældende dag og eventuelle problemer, disse er afholdt over discord. Vi har alle arbejde ved siden af studiet som gør at disse standup meetings måske ikke altid har lagt starten af dagen, vi har måtte planlægge udfra arbejdstider tidspunkt hvor vi alle havde tid til at sætte os sammen på discord og afholde disse.

Grundet brug af SCRUM har vi haft en klar plan for hvad der skulle laves og hvem der skulle lave hvad. Det har givet mulighed for at vi også kunne arbejde individuelt, når andre fra gruppen var på arbejde.

Nogle af de samme To-Do “user stories” ligger under “in progress” i flere af vores sprints dette er grundet konstant udvikling og ændring på dem.

Sprint 0: D. 22 - 25 (Lucas)

Herunder vises et skærbillede af vores SCRUM Board i starten af sprint 0
(Se Figur 2)



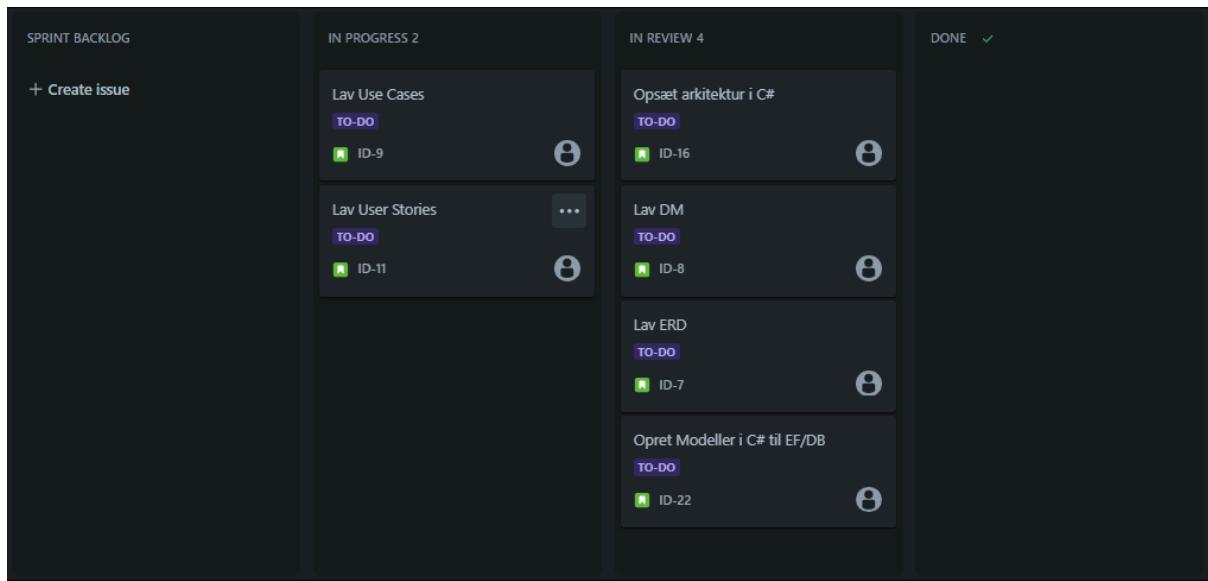
Figur 2 - SCRUM Board, Start Sprint 0 D. 22/04-24

Vi havde ved sprint 0 lige fået udleveret vores projekt, og har samtidigt undervisning. Så ved siden af skole og arbejde var tiden knap. Målet ved dette sprint var at få lagt en base for programmet, finde ud af hvad programmet skal indeholde. Og få analyseret casen ift hvordan databasen skal bygges op med relationer osv.

Så vi lavede lige en kort To-Do backlog som forklaret tidligere som vi bruger i vores sprint 0 så derfor ses ingen User stories på dette Scrum board.

Sprint 0 var vores “forberedelses” sprint også derfor kaldt sprint 0. Her i planlægger vi også længden af fremtidige sprints så dette er et kort sprint og varer kun i 3 dage.

Her ses skærbilledet af slut sprint 0 (Se figur 2,1)



Figur 2,1 - SCRUM Board, Slut sprint 0 d. 25/04-24

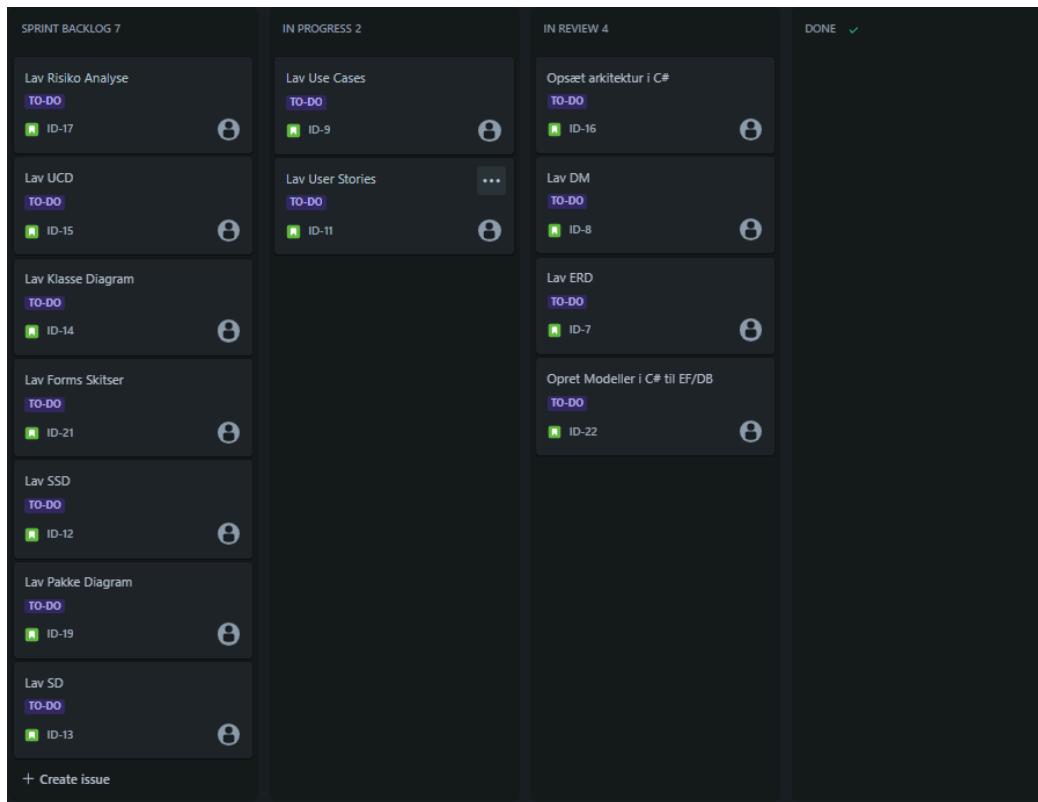
Sprint retrospective (Lucas)

Sprintet gik meget godt, vi nåede ikke alt hvad vi havde planlagt men det havde vi forudset, da vi lige var startet og vidste ikke hvad projektet helt indebar, samt at det var et meget kort sprint.

Vi fik opsat arkitekturen i c#, fik lavet vores første udkast til vores Domæne model og ERD, samtidigt fik vi lavet entiteterne/modellerne inde i c#

Intet er rykket over på done men står stadig "in review" da vi godt var klar over det nok ik var det endelige og at der ville forekomme en del ændringer da dette var første udkast. Men det er os en af fordelene ved Scrum at vi kan ændre det undervejs. User Stories og use cases nåede vi ikke så langt med dette er stadig igangværende.

Sprint 1: D. 25 - 02 (Lucas)



Figur 3 - Scrum Board, start sprint 1 d. 25/04-24

Herover ses et skærmbillede af vores Scrum Board start sprint 1 d. 25/04-24 (Se figur 3)

Her ses skærmbilleder af vores risiko analyse for sprint 1²

Sprint 1 foregik ligesom ved sprint 0 med undervisning og samtidigt med arbejde gjorde det tiden knap. I dette sprint er det endnu ikke planlagt at gå i gang med selve programmet men regner med at få de sidste diagrammer, og overvejelser på plads til at vi i sprint 2 kan starte med programmering.

Vi er nu kommet til midten i sprint 1³ her kan vi se arkitekturen er blevet ordnet i c# og vi er begyndt at få sat nogle basis ting på plads. User stories og use cases er blevet oprettet og vi er så småt igang med UCD, skitser til vores forms og begynder så småt at teste oprettelse af vores database ved hjælp af Entity Framework med Code First approach.

² [Risikoanalyse sprint 1](#)

³ [Scrum Board - Midt sprint 1](#)

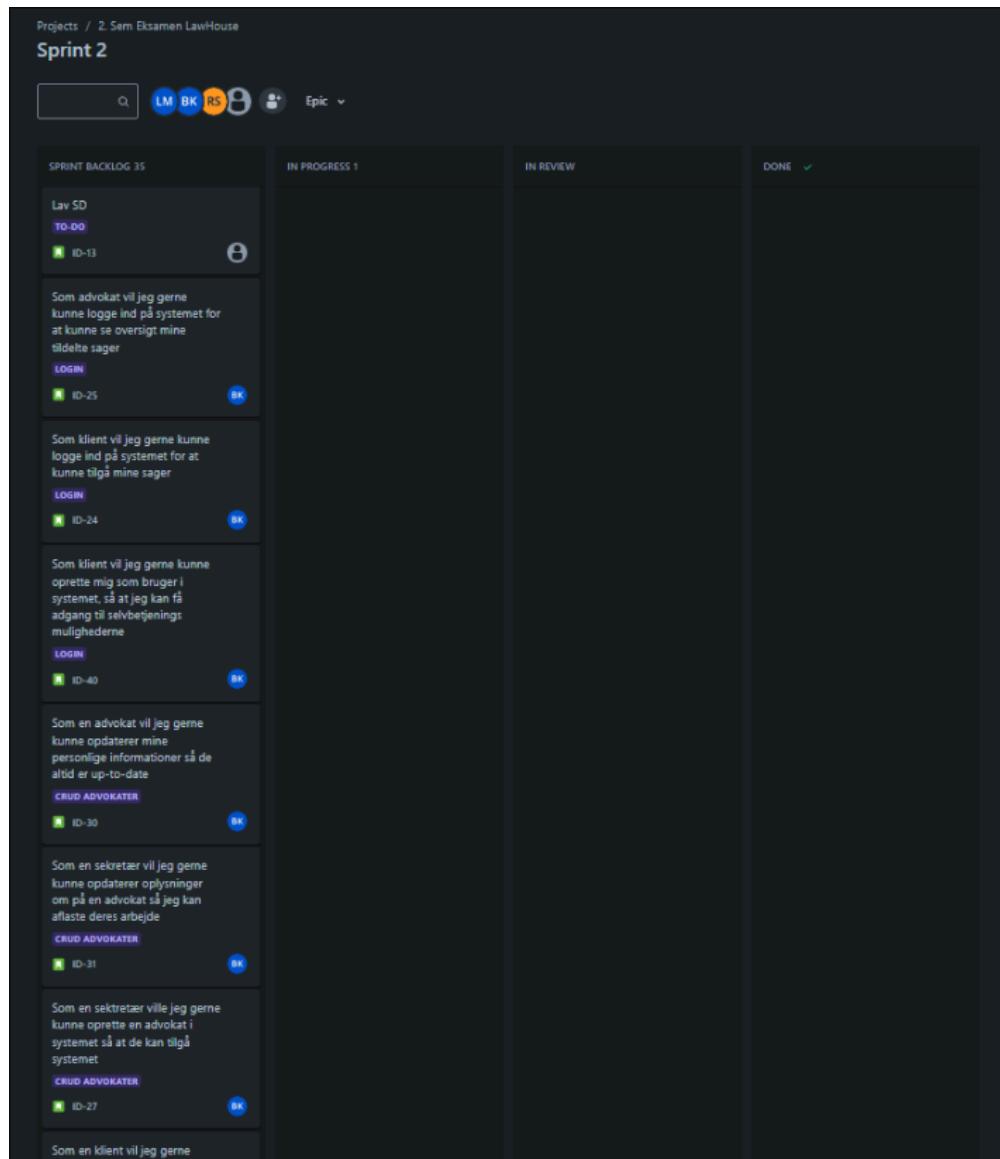
Hvis vi kigger på vores board slut i sprint 1⁴ kan vi se at vi begynder at få en masse basis ting på plads. User stories er lavet, use cases, use-case diagram, oprettelse af modeller i og modelkonverterer c#, og vi har fået oprettet vores første version af vores database igennem Entity framework Codefirst.

Sprint Retrospective (Lucas)

Overordnet set er det gået rigtigt godt i Sprint 1, vi mener der er kommet en masse grundsten på plads der gør at vi er ved at være klar til at gå i gang med at lave selve programmet. Sprint 1 har gået rigtigt meget på at finde ud af hvordan det skal sættes op, mange forskellige forslag til database har været oppe og vende, diskuteret og afprøvet. Vi føler ligesom at det er ved at komme godt på plads og det ligner noget vi kan bruge til vores program. Domain model står stadig i review da vi igen ikke er helt sikker på om det er vores endelige bud og det sagtens kan nå at ændre sig. Mange af de ting som er rykket over i done kan selvfølgelig stadig blive ændret på men vi føler at det er minimalt hvad der skal ændres og er derfor rykket over i done. Vi nåede ikke at blive færdige med nogle af vores diagrammer, så de kommer med over i sprint 2. Vi brugte sprint 1 som en sprint 0-V2. Vi havde så mange tanker og forskellige muligheder hvordan selve databasen skulle bygges op samtidigt med programmet hvordan det helt præcist skulle fungerer.

⁴ [Scrum Board - Slut sprint 1](#)

Sprint 2: D. 02 - 09 (Lucas)



Figur 4 - Scrum Board, start sprint 2 d. 02/05-24

Ovenover ses billede fra starten af sprint 2 (Se Figur 4).

Risiko analyse for sprint 2⁵

Nu er vi kommet til sprint 2, dette er sprintet hvor vi begynder på programmering af selve programmet. Der skal knækkes hul på kode delen og vi har sammen i gruppen, hver i sær udvalgt en god del user stories fra product backloggen som vi mener vi kan nå i dette sprint. Der er mange user stories og de kan ikke alle være på billedet, alle user stories til sprint 2 ses

⁵ [Risikoanalyse - Sprint 2](#)

her⁶. Der var stadig nogle få diagrammer fra sprint 1 der ikke blev nået at lave færdig, så de er selvfølgelig også inkluderet i dette sprint og medregnet på tiden.

Halvvejs igennem sprint 2⁷ syntes vi at vi er kommet rimelig godt igang og føler vi er godt med. Vi har færddigjort 14 ud af de 36 userstories 11 er igangværende og 9 tilbage i sprint backloggen.

Sprint Retrospective (Lucas)

I slutningen af sprint 2⁸ er der stadig 4 userstories som vi desværre ikke nåede at få gjort færdig, vi overestimerede måske lidt hvor hurtigt vi kunne kode nogle ting og har brugte lidt ekstra tid på lidt “nice to have” features.

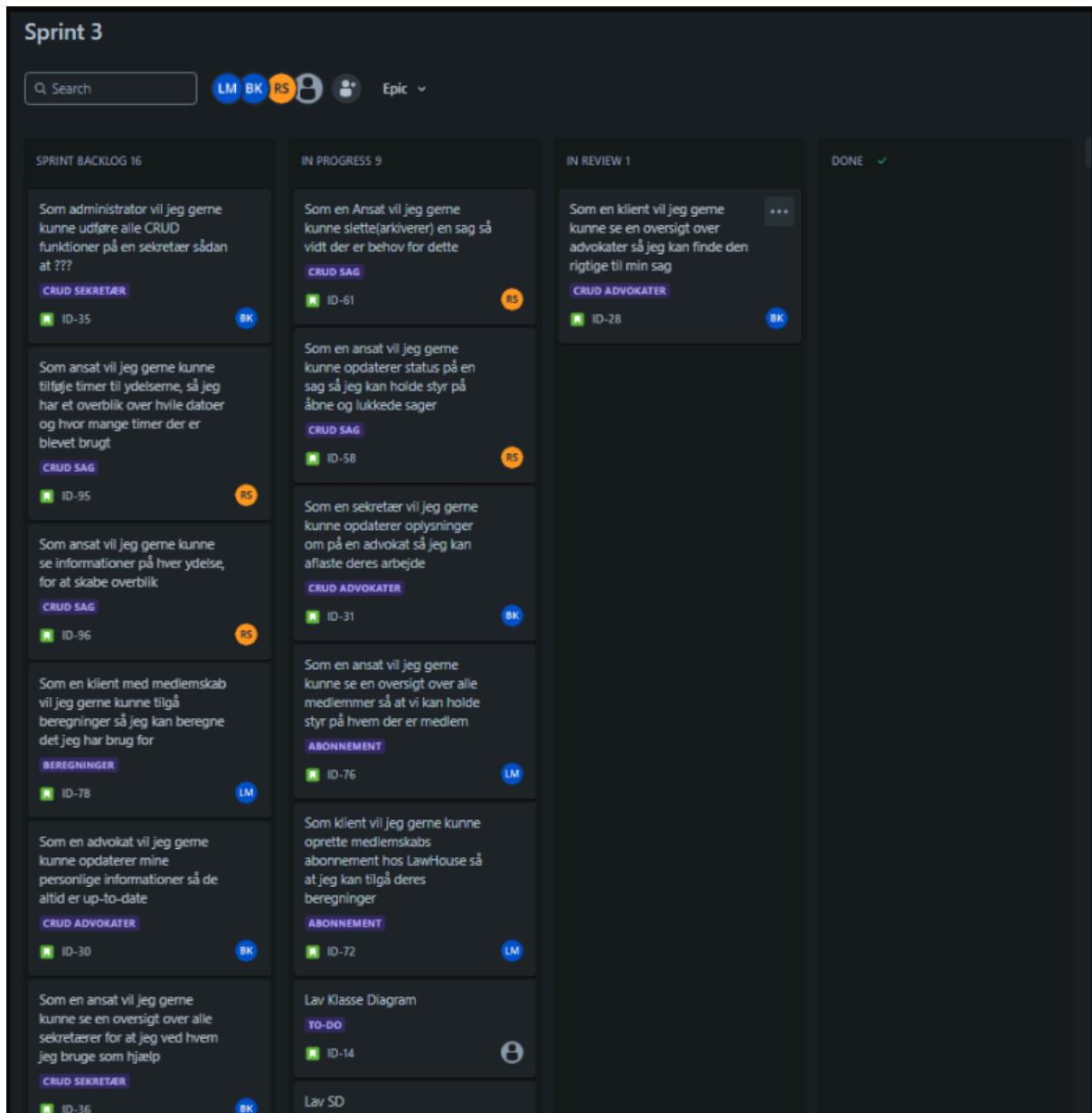
Vi har været meget produktive i sprint 2 rigtig mange features er blevet implementeret i vores system og der er blevet knoklet hårdt. Vi er tilfredse med resultatet efter dette sprint og ser frem til at starte på sprint 3.

⁶ [User stories - Sprint 2](#)

⁷ [Scrum Board - Midt Sprint 2](#)

⁸ [Scrum Board - Slut sprint 2](#)

Sprint 3: D. 09 - 16 (Lucas)



Figur 5 - Scrum Board, Start sprint 3

Risikoanalyse sprint 3⁹

Ovenfor ses billede af Scrum board for starten af sprint 3 (Figur 5)

Vi fortsætter fra sprint 2 med nogle enkelte user stories og vi har igen sat os ned som gruppe planlagt sprintet. Der er en del user stories, alle userstories ses her¹⁰

⁹ [Risikoanalyse - Sprint 3](#)

¹⁰ [User stories - Sprint 3](#)

Planen for sprint 3 er at alle grund features bliver lavet sådan at tingene fungerer, dvs alt er ikke finpudset og optimeret men det virker. Så næsten alle resterende user stories fra product backlog ligger i dette sprint.

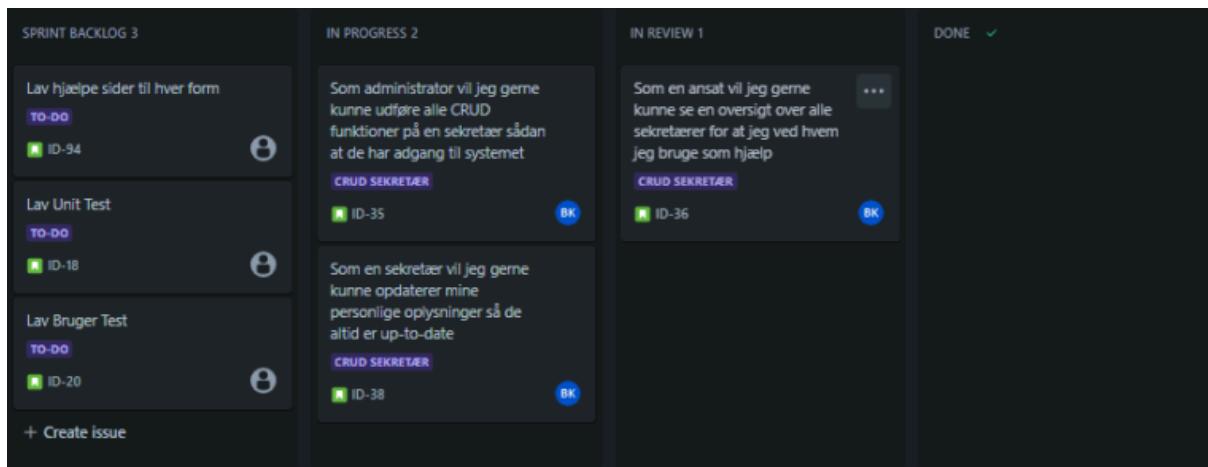
Halvejs igennem sprint 3¹¹ er vi kommet godt igang, det passer med at vi cirka har nået igennem halvdelen af sprint backloggen, dynamikken kører og vi arbejder på fuld skrue. Programmet begynder at udforme sig.

Sprint retrospective (Lucas)

Slut sprint 3 scrum board¹²

Sprintet er gået rigtigt godt der er lige et par user stories stadig in progress ellers er vi kommet godt igennem og rundt om alle user stories. Vi har igen også under sprintet brugt tid på nogle nice to haves og noget finpudsning rundt omkring. Det er blevet knoklet hårdt og samarbejdet køre rigtig godt.

Sprint 4: D. 16 - 23 (Lucas)



Figur 6 - Scrum Board, Start sprint 4

Risikoanalyse sprint 4¹³

¹¹ [Scrum Board - Midt sprint 3](#)

¹² [Scrum Board - Slut sprint 3](#)

¹³ [Risikoanalyse - Sprint 4](#)

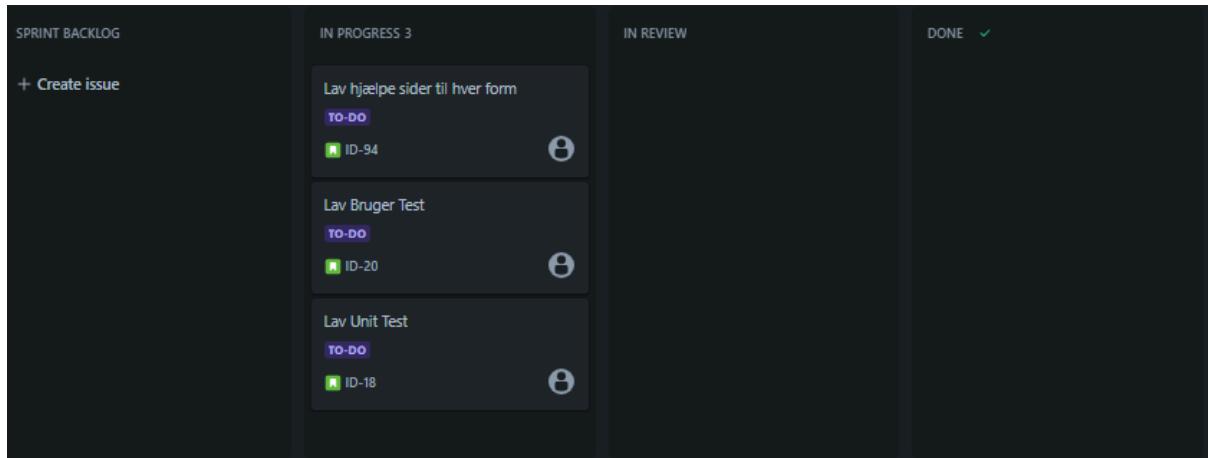
Så er vi nået til sprint 4 (Figur 6) Nu er der blevet tyndet godt ud i vores product backlog og vi er ved at nå slut fasen. Der er få user stories tilbage fra sprint 3 som skal færdiggøres i dette sprint. Formålet med dette sprint var derudover at lave en god gennemgang af programmet for fejl. Vi har undervejs i alle sprints skrevet lidt på rapporten, men her går vi også for alvor i gang med at skrive på rapporten.

Halvvejs igennem sprintet¹⁴ er vi færdige med de sidste user stories og der fortsættes med test og fin pudsnings

Sprint retrospective (Lucas)

I dette sprint havde vi meget arbejde ved siden af selve projektet så vi var godt forberedt på at indsatsen ikke var helt den samme. Vi har fået lavet alle programmets funktioner og rettet en masse fejl og lavet noget test så sprintet er overall gået godt.

Sprint 5 D. 23 - 30 (Lucas)



Figur 7 - Scrum Board, Start sprint 5

Risikoanalyse sprint 5¹⁵

Så er vi kommet til 5. og sidste sprint. Her kan man se vi er ved at nå ved vejs ende. Der mangler hjælpe pdf filer til alle forms. Der mangler at blive lavet brugertest og unit test. Derudover bruges meget af tiden i sprint 5 til at skrive rapporten færdig.

¹⁴ [Scrum Board - Midt sprint 4](#)

¹⁵ [Risikoanalyse - Sprint 5](#)

Sprint retrospective (Lucas)

Vi er nu nået til enden af projektet. Sprintet er gået super godt, der var ikke særligt meget arbejde ved siden af projektet, så alle har sat fuld tid på hver dag, for at kunne gennemføre det sidste sprint og få testet de sidste features. Det har alt i alt været et rigtigt sjovt og spændende projekt, der er blevet lagt en masse kræfter i det. Selvom det har været hårdt har vi hygget os rigtigt meget med det og nydt at have et rigtigt projekt at sidde med over en længere periode. - Tak for denne gang.

IT-Værktøjer

Github (Bilal)

Vi har anvendt Github til versionering, da det gør udviklingsprocessen meget mere gnidningsfri, når vi er et team der til tider arbejder på tværs af koden. Github gør samarbejdet effektivt og sporbar, når der skal merges, laves pull-request, kan ændringerne blive set og problemer håndteres, inden der går kludder i det hele. Skulle det alligevel finde sted, er det relativt nemt at gendanne og spare en masse tid og frustration.

Vi har ikke oplevet store problemer her, dog har der været tilfælde, hvis man ikke har merged i lang tid eller glemt at arbejde i branches, at manuel indgreb blev nødvendigt.

Microsoft Visual Studio (Bilal)

Visual Studio har været vores primære udviklingsmiljø, hvilket har gjort kodning og udvikling effektiv. Integration med værktøjer som NuGet-pakker, GitHub og EF Core har forbedret vores workflow. Debugging, kommentarer og WinForms har bidraget til fejlretning og brugeroplevelsen. Funktioner som API-integration, PDF-håndtering, unit testing og databaseadministration har været centrale, hvilket gør VS uundværlig for både udvikling og slutproduktet.

Microsoft SQL Server (Bilal)

SQL Server Management Studio (SSMS) programmet har været et godt værktøj til at se og bekræfte resultatet af de henvendelser vi kunne foretage os fra programmet. Da vi har oprettet databasen via Entity Framework Core, har vi undgået at foretage manuelle ændringer i databasen, udover at teste joins, data-udtrækninger og constraints.

Triggeren, nævnt senere i rapporten, blev dog oprettet gennem SSMS.

Framework & NuGet Packages (Bilal)

I vores system har vi valgt at gøre brug af flere forskellige NuGet pakker, som hjælper med at forbedre udviklingsprocessen. Nedenfor ses en oversigt over de forskellige og i hvilket lag de er integreret.

DataAccess

Microsoft.EntityFrameworkCore¹⁶

- Denne pakke anvendes til objekt-database mapning, og gør det muligt at interagere med databasen via .NET objekter.

Mircosoft.EntityFrameworkCore.SqlServer

- Denne pakke tillader os at bruge SQL Server som vores database.

System.Configuration ConfigurationManager¹⁷

- Denne pakke understøtter brug af XML-konfigurationsfiler (app.config).

System.Data.SqlClient

- Giver SQL Server database tilgængelighed for .NET applikationer.¹⁸

BusinessLogic

NewtonsoftJson

- Denne pakke bruges til at serialisere og deserialisere objekter til og fra JSON format, hvilket er brugt ved implementering af vores API.¹⁹

UI

FontAwesome.Sharp

¹⁶ [NuGet Gallery | Microsoft.EntityFrameworkCore 8.0.6](#)

¹⁷ [NuGet Gallery | System Configuration ConfigurationManager 9.0.0](#)

¹⁸ [System.Data.SqlClient Namespace | Microsoft Learn](#)

¹⁹ [Json.NET](#)

- Bruges til at anvende ikoner i vores Windows Forms, og giver et bedre visuelt udtræk, samt forbedrer brugervenligheden

Microsoft.EntityFrameworkCore.Tools

- Denne pakke bruges primært til at håndtere migrationer og til at generere en DbContext.²⁰

UnitTest

coverlet.collector

- Inkluderes ved brug af NUnit

MicroSoft.NET.Test.Sdk

- Understøtter testkørsler

Moq

- Vi prøvede ved brug af Moq, ved implementering af vores Unit tests, ved at teste et kald til databasen uden faktisk at bruge den.

NUnit

- Et enhedstest framework, giver struktur og metoder til at oprette og køre Unit Tests.²¹

NUnit.Analyzers

NUnit3TestAdapter

Discord (Lucas)

Discord har været en af de primære redskaber, gruppen har benyttet til specielt kommunikation. Discord giver mulighed for en effektiv og brugervenlig måde hvorpå vi kan kommunikere, dele sourcecode og andet relevant information vedr. projektet.

Vi har oprettet vores egen kanal, hvorigennem vi har opnået hvad discord kan tilbyde.

²⁰ [Microsoft.EntityFrameworkCore.Tools 8.0.4](#)

²¹ [NuGet Gallery | NUnit 4.1.0](#)

Analyse

Use-Cases (Lucas)

Vi lavede en række use-cases for at skabe et overblik over hvilke funktioner der skulle implementeres i systemet samt hvilke aktører der har gang til hvad.

Use Cases findes under bilag -> Use cases²².

Fully Dressed Use Case (Rasmus)

Fully Dressed Use Case: 8.1 Brug af beregningsalgoritmer

Use Case Navn: En klient med abonnement skal kunne gøre brug af beregningsalgoritmerne

Primær Aktør: Klienten

Aktørens Mål:

- Klienten ønsker at bruge beregningsalgoritmer til at få specifikke resultater, der kan hjælpe med deres behov.

Stakeholders and interests

- Klient: Ønsker at have mulighed for at kunne benytte sig af beregningsalgoritmer for at opnå nogle specifikke resultater.
- Systemadministrator: Ønsker at der sikres, at det kun er muligt at have adgang til beregningsalgoritmerne såfremt at klienten har et abonnement
- Ledelse: Ønsker at fastholde deres kunder og skabe værdi ved at tilbyde nogle beregningsalgoritmer gennem medlemskab.

Preconditions:

1. Klienten er oprettet i systemet
2. Klienten er logget ind
3. Klienten har et gyldigt medlemskab
4. Der er tilgængelige beregningsalgoritmer

²² [Use Cases](#)

Main Success Scenario (Basic flow)

1. Klienten logger ind i systemet
2. Klienten navigere til beregningsalgoritmerne på forsiden
3. Systemet viser en oversigt over de tilgængelige beregningsalgoritmer
4. Klienten vælger den ønskede beregningsalgoritme fra oversigten
5. Systemet viser de forskellige inputfelter som klienten skal udfylde.
6. Klienten udfylder de nødvendige inputfelter med gyldig oplysninger
7. Klienten trykker beregn
8. Systemet anvender de indtastede oplysninger til at foretage beregning ift. den valgte beregningsalgoritme
9. Systemet viser det beregnede resultat for klienten

Extensions

1. Klient har ikke medlemskab
 - a. Systemet nægter adgang til beregningsalgoritmerne og dermed tilbyder klienten at abonnere
2. Ugyldig oplysninger indtastet i inputfelt
 - a. Systemet fremhæver de felter, hvor klienten mangler eller har udfyldt ugyldigt data
 - b. System giver ikke klienten mulighed for at foretage en beregning før at de ugyldige informationer er rettet

Special Requirements

1. Systemet skal sikre at kun klienter der har et aktivt medlemskab kan gøre brug af biblioteket af beregningsalgoritmerne
2. Systemet skal have en brugervenlig grænseflade, for at sikre en god oplevelse for klienterne
3. Systemet skal udføre beregningerne præcist

Use Case Diagram (Rasmus)

Use case diagrammet er et vigtigt redskab inden for analyse af vores system. Det er blevet anvendt for at illustrere de forskellige krav og visualisere hvordan de forskellige aktører interagerer med systemet. I figur 8, ses use case diagrammet for vores system LawHouse.



Figur 8 - Use Case Diagram

I vores use case diagram har vi identificeret følgende primære aktører:

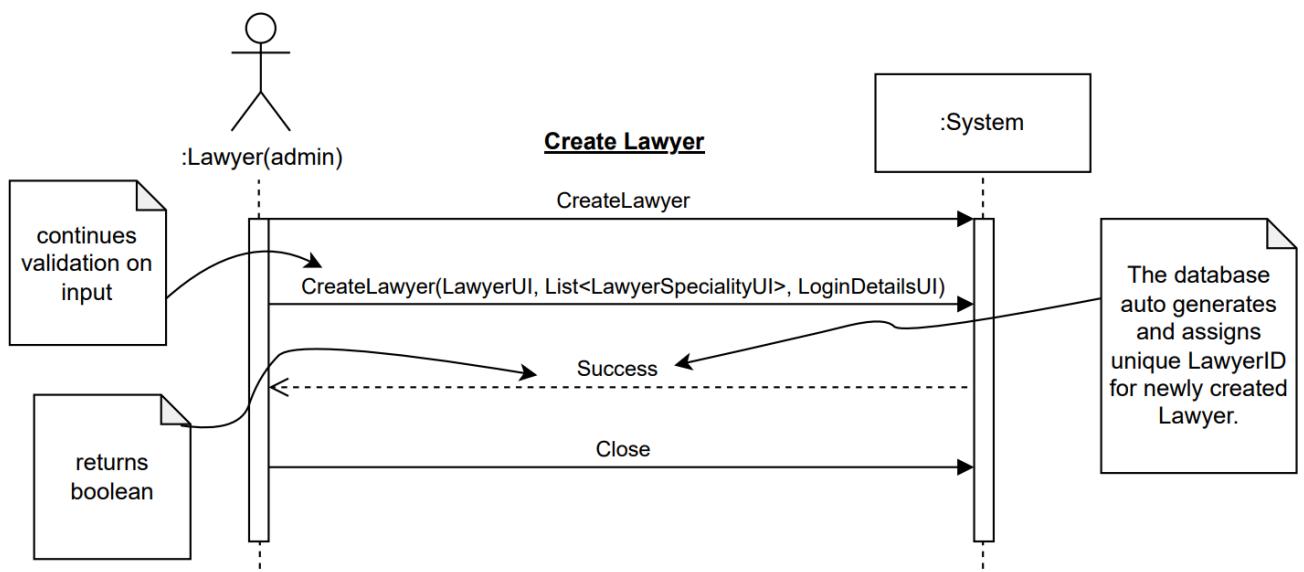
- **Admin:** En bruger med administratorrettigheder
- **Sekretær:** En bruger som assisterer advokater og klienter ved at hjælpe med de administrative opgaver
- **Advokat:** En bruger som arbejder med sager og klienter.
- **Klient:** En bruger, der gør brug af systemet for at få juridisk vejledning, se sager og købe formularer.
- **Subscribed Klient:** Er en udvidelse af klienten, hvis de har abonneret

Systemet er konstrueret således at en admin bruger har adgang til alle aspekter af vores system. En advokat og sekretær har adgang til mange af de samme funktioner, dog på undtagelse af UC 1.3 Update Secretary og UC 2.3 Update Lawyer, idet vi i vores konstruktion

af systemet har taget højde for, at det kun er en admin og en advokat selv der kan rette sine egne oplysninger, det samme gør sig gældende for sekretær.

Diagrammet viser, hvordan de forskellige aktører interagerer med systemet for at udføre forskellige funktioner. Linjerne, der forbinder en aktør til en use case, repræsenterer at aktøren har adgang til denne funktionalitet. F.eks. at en klient med medlemskab kan se alle de forskellige beregninger. Diagrammet hjælper med at forstå systemets krav og det giver en klar og konsistent måde at visualisere alle de forskellige funktioner. Derved sikre det at vi har en fælles forståelse af hvilke funktionaliteter systemet indebærer samt dets begrænsninger.

System Sekvens Diagrammer (Rasmus & Bilal)



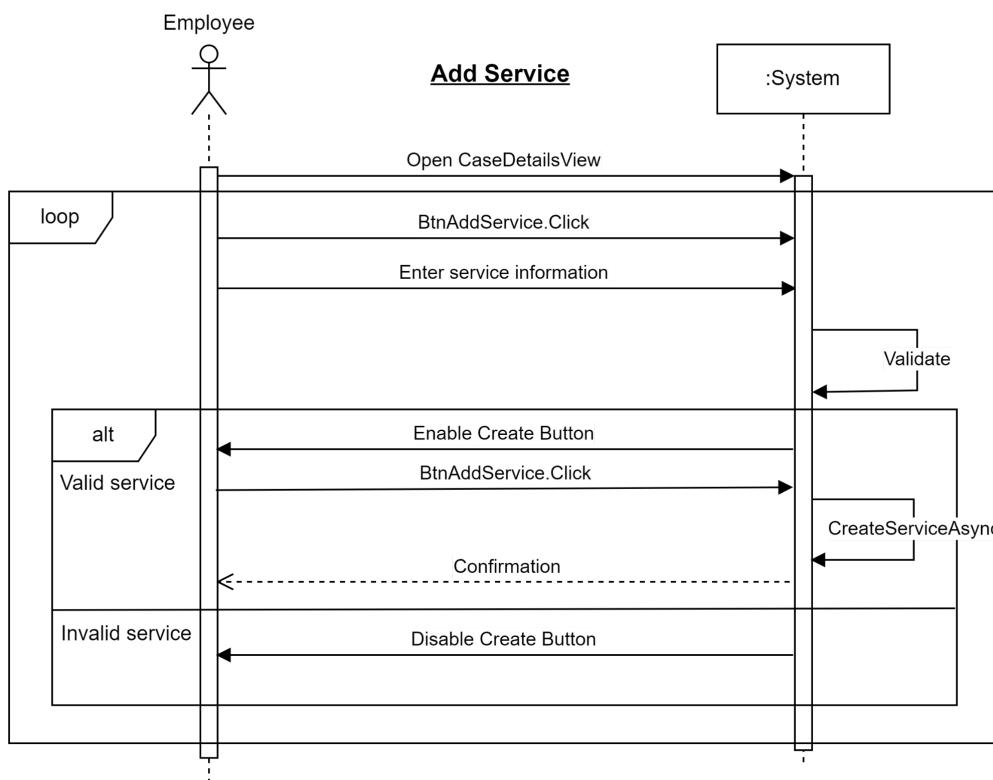
Figur 9 - SSD / Create Lawyer

Dette systemsekvensdiagram (SSD) tilhører use case **UC 2.1**, som illustrerer de trin aktøren "Administrator" foretager sig ved oprettelse af en advokat i systemet.

Aktøren interagerer med systemet og sætter "CreateLawyer" processen i gang som fremviser en form med felter, der gør det muligt at indtaste alle de relevante felter ind, så attributter som personoplysninger, advokattitel, specialer, ansættelsesdato og logindetaljer kan blive registreret.

Ved indtastning af disse oplysninger, vil brugergrænselaget foretage løbende validering af formatet for de respektive felter, for at informere brugeren om evt. fejl, hvor f.eks. postnummeret skal være på 4 cifre og navn ikke må indeholde andet end bogstaver og bindestreg. Når administratoren er færdig og alle felter er korrekt indtastet, vil ”Create Lawyer”-knappen blive aktiv og kan trykkes på, så processen kan fortsætte.

”CreateLawyer” metoden bliver nu kaldt med de viste input parametre og systemet opretter advokaten i databasen. Systemet returnerer en succesindikator, som indikerer om operationen var vellykket eller ej, og brugeren får tilbagemelding i form af en MessageBox. Ved vellykket operation vil systemet have oprettet og tildelt unik LawyerID til advokaten og LoginDetailsID.



Figur 10 - SSD, Add Service

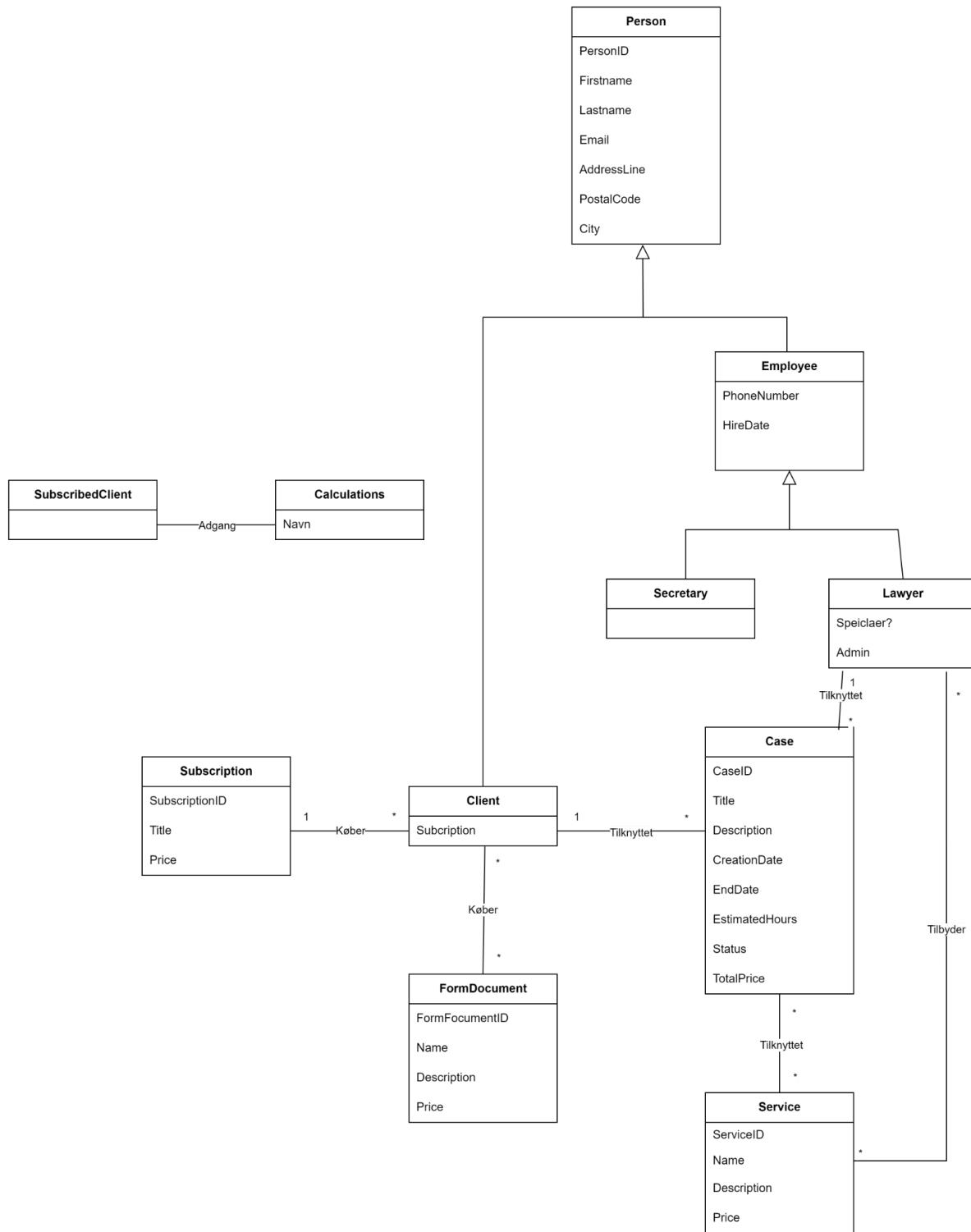
Dette systemsekvensdiagram (SSD) tilhører use case **UC 4.4**, som illustrerer processen ved at tilføje ydelser til en sag.

Aktøren starter med at åbne CaseDetailsView²³, hvorefter processen sættes i gang ved at klikke på 'AddService', her skal aktøren udfylde de nødvendige informationer vedrørende en sag. Der foretages dermed en validering af de respektive input-felter løbende i vores brugergrænseflade. Hvis validering lykkedes, aktiveres knappen og brugeren kan derefter tilføje servicen ved at kalde CreateServiceAsync. Der vil derefter blive vist en messagebox med en bekræftelse, med at servicen er blevet tilføjet korrekt. Hvis der opstår fejl under oprettelsen, vil der i stedet blive vist en fejlbesked. I tilfælde af at valideringen mislykkes, vil knappen blive deaktiveret, for at undgå yderligere handlinger.

I diagrammet er der implementeret et loop, som viser at aktøren kan gentage processen og tilføje flere services til en sag. Derudover er der en alt, hvilket beskriver hvad der sker hvis valideringen lykkes og hvis den ikke gør.

²³ [Screenshot af CaseDetailsView](#)

Domæne Model (Rasmus)



Figur 11 - Domæne model

Dette er vores domænemodel for LawHouse, der repræsenterer de forskellige entiteter, sammen med deres attributter og relationer inden for systemet. Formålet med

domænemodellen er at der skabes en struktureret visualisering af systemet, der gør det lettere at se hvordan de forskellige entiteter håndteres og interagerer med hinanden.

Domænemodellen viser hvordan entiteterne relaterer til hinanden, f.eks at en client kan være tilknyttet til flere cases eller at en case skal have en tilknyttet client og en lawyer.

Der er ligesom i vores ER-Diagram gjort brug af specialisering, idet vi igen har en overordnet entitet Person, hvorfra den specialiseres ned i Client og Employee, og Employee videre ned i Lawyer og Secretary. Dette er vist ved brug af arv-pile.

Ikke-funktionelle krav (Bilal, Lucas & Rasmus)

Vi har i gruppen skrevet en liste over ikke-funktionelle krav.

Ydeevne

- Kald til databasen skal køres inden for relativt kort tid, helst indenfor få sekunder.

Sikkerhed

- Alle database kald via SQLClient skal gøre brug af Parameters for at undgå SQL-Injection.
- Systemet skal sikres mod uautoriseret adgang
- Systemet skal have adgangskontrol baseret på brugertype, så brugeren kun har adgang til funktioner, de er autoriseret til.

Drift

- Der skal altid være adgang til databasen medmindre der er planlagt nedlagt tidsrum.
- Der skal være backup fil af databasen, i tilfælde af nedbrud.

Brugervenlighed

- Systemet skal have en brugervenlig grænseflade, som er intuitiv, læsbar, have klare introduktioner og hjælpematerialer i form af PDF.

Design

Systemets Arkitektur (Rasmus)

Vores projekts arkitektur er struktureret i flere lag, som er med til at sikre at der er en klar afkobling mellem lagene og opdeling af ansvar, hvilket bidrager til en mere effektiv datahåndtering. Her er en oversigt over de forskellige lag:

UI (User Interface)

Dette lag er primært ansvarlig for at formidle og præsentere data til brugerne, samt håndtere brugerinteraktioner. Det er implementeret gennem Windows Forms.

UI-Models

Dette lag indeholder alle vores UI-modeller, som repræsenterer alt det datam der skal præsenteres i vores UI.

Business Logic

Business Logic fungerer som et mellemled mellem vores UI og vores DataAccess. Det er ansvarlig for at implementere den logik, der styrer hvordan vores data behandles i systemet. Dette lag kalder til DataAccess for at udføre CRUD-operationer på databasen, hvorefter det modtager resultatet og sender det videre til UI.

Business logic-laget validerer de data, som kommer fra vores UI, før de konverteres og gemmes i databasen. Dette mindsker risikoen for fejl og sikrer at vi kun opererer med gyldige data. Dette lag håndterer også vores konvertering mellem UI-modellerne og entitetsmodellerne.

DataAccess

DataAccess-laget bruger entitets modeller og Entity Framework til at interagere med vores database. DataAccess-laget gør brug af interfaces, hvilket øger systemets fleksibilitet og udskiftning af andre data implementeringer uden at påvirke de andre lag.

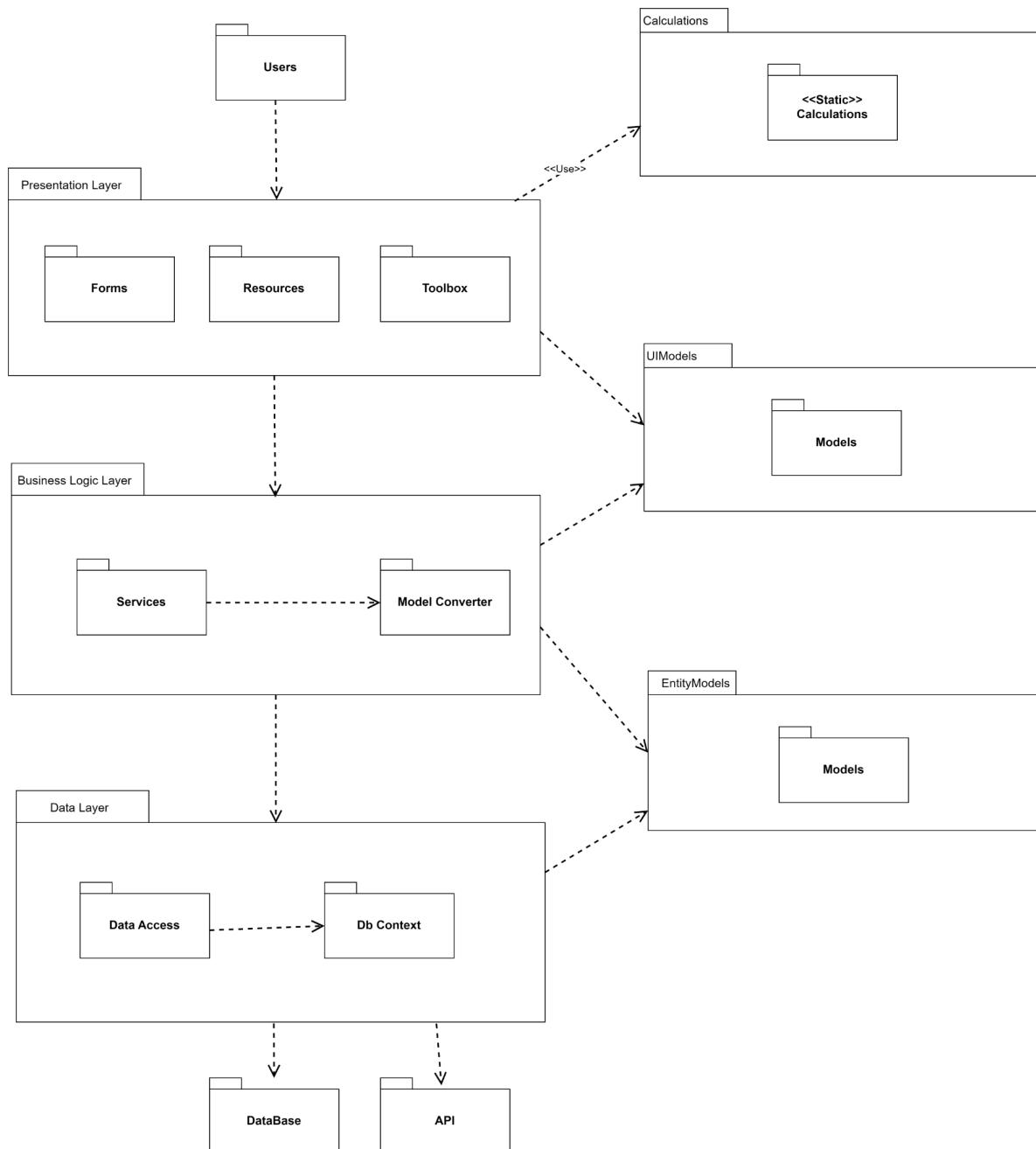
Ved at bruge denne arkitektur har vi en klar og tydelig opdeling af systemets ansvarsroller, hvilket gør det lettere at vedligeholde. Systemet er gjort meget skalerbart, idet hvert lag kan

opdateres og udskiftes uafhængigt af de andre, hvilket gør det nemmere at tilpasse nye krav, såsom at vores UI skulle udskiftes, kunne dette opnås uden at påvirke de andre lag.

Interfaces

For at opnå en løs kobling mellem vores forskellige lag, har vi valgt at implementere interfaces for vores DataAcces-lag. Dette gør, at vi for hver DbAcces kan definere en kontrakt, som de skal følge.

Pakke Diagram (Lucas)



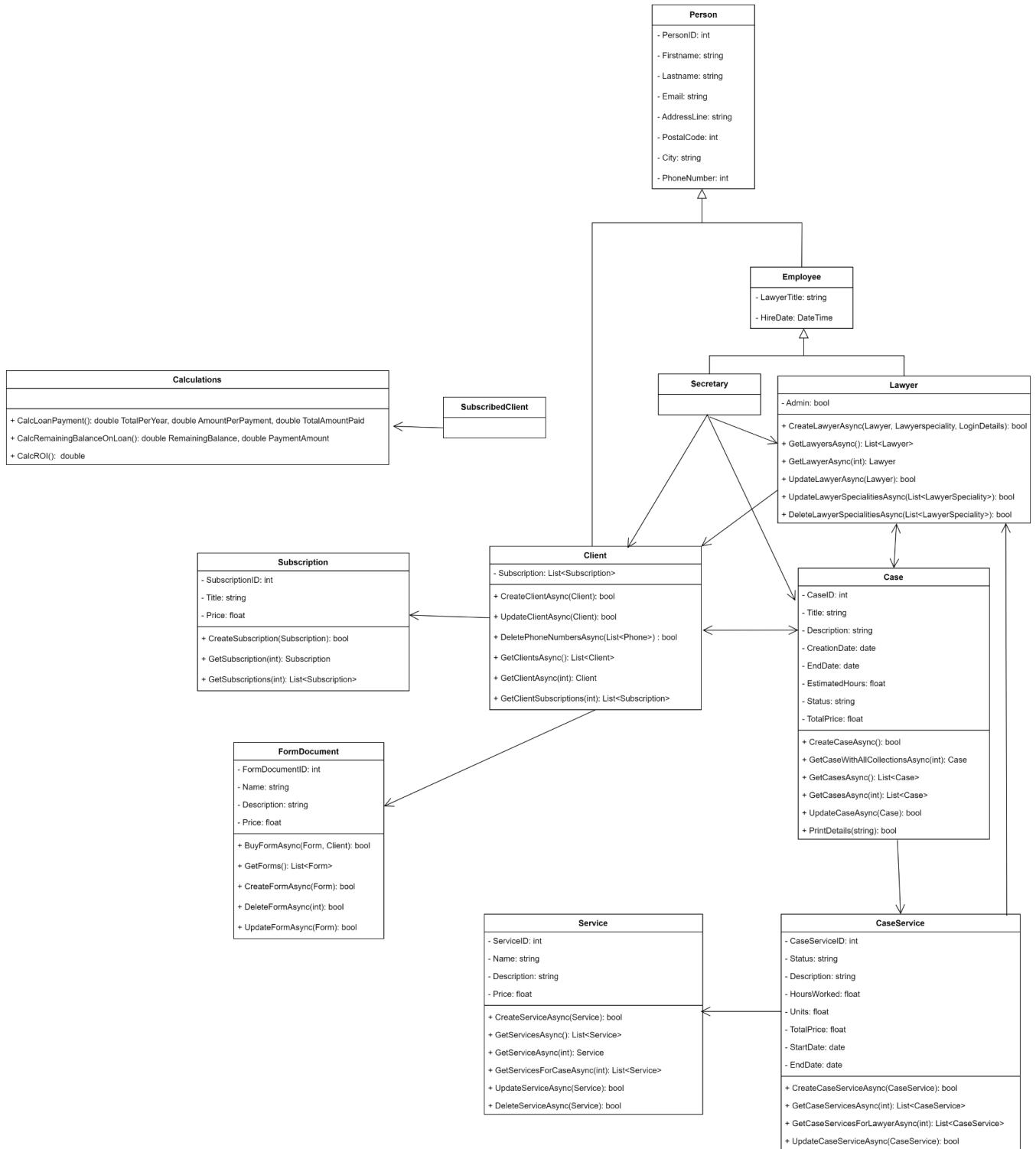
Figur 12 - Pakke diagram, LawHouse

Vi har valgt at lave et pakke diagram til LawHouse systemet. Dette giver en god visualisering af hvordan arkitekturen for programmet er opbygget. Beskrevet i dybden her²⁴

²⁴ [Systemets Arkitektur](#)

Klasse Diagram (Rasmus)

Vores klassediagram er en repræsentation af de klasser vi har i vores system, samt deres gældende attributter og metoder. Derudover viser det også relationen imellem dem.

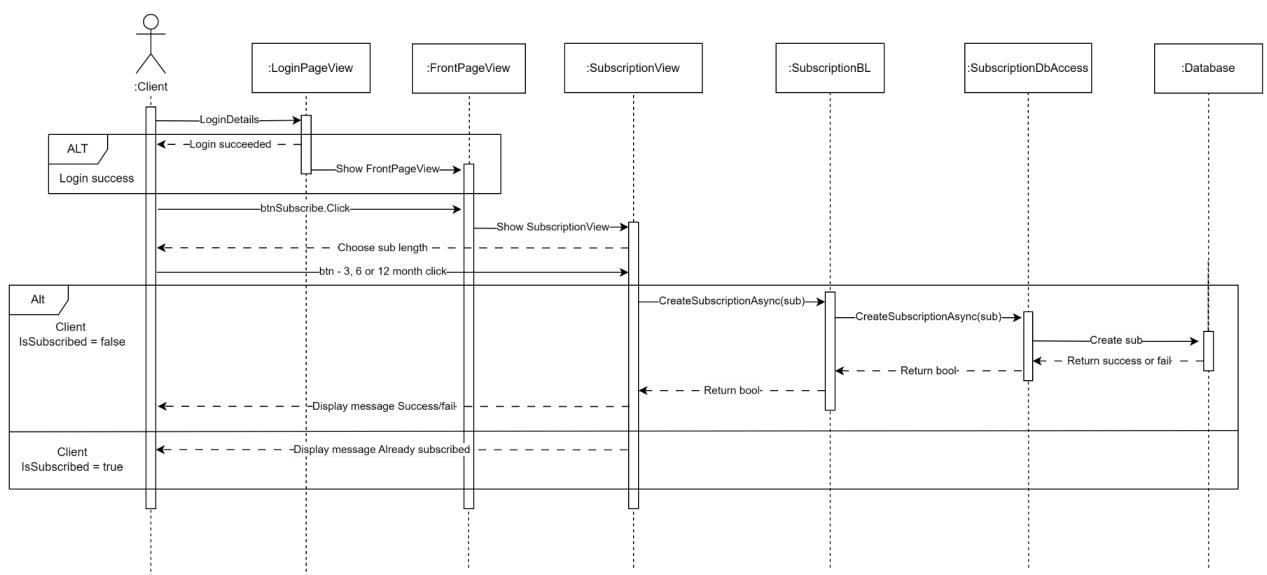


Figur 13 - Klasse diagram

Vi har i vores klassediagram valgt at inkludere alle CRUD-operationer i klassediagrammet, selvom implementeringen af disse metoder ligger i DataAccess-laget. Det har vi gjort på baggrund af, at det afspejler hvilke operationer som er tilgængelige for hver klasse. Det gør diagrammet mere forståeligt, idet det skaber et tydeligere overblik over systemets funktionalitet.

Ved brug af pile har vi valgt at vise hvordan de forskellige klasser har adgang til hinanden. F.eks. har vi valgt at vise sekretær har en relation til Cases, Lawyer og Client, dette skal forstås ved at en sekretær har adgang til metoderne i disse klasser.

Sekvensdiagrammer (Lucas & Rasmus)



Figur 14 - Sekvensdiagram, Client creates membership UC - 6.1

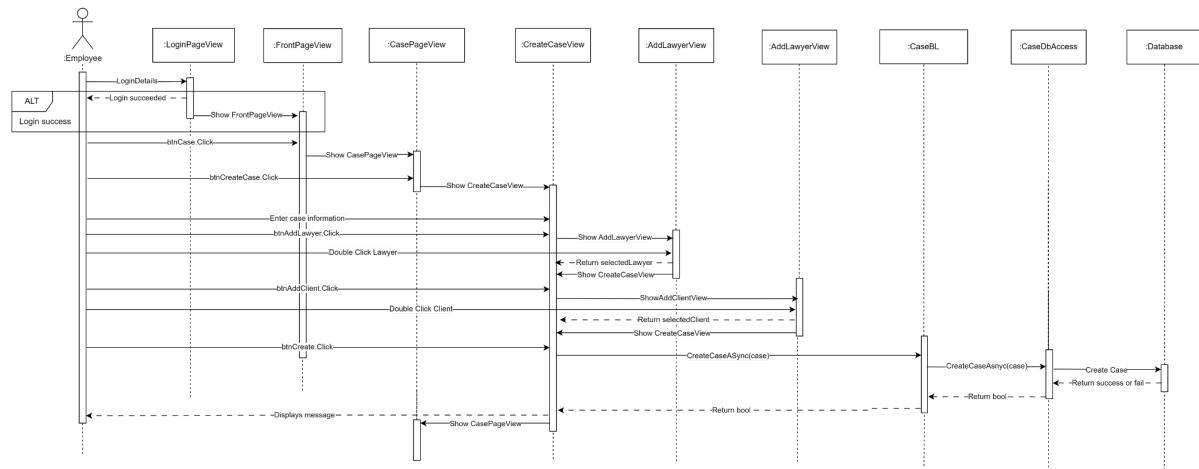
Dette sekvensdiagram illustrerer processen i **UC 6.1** (Figur 14), hvor aktøren "Client" skal oprette abonnement/medlemsskab hos LawHouse.

Klienten interagerer med systemet og logger ind, derefter bliver forsiden vist.

Klienten trykker derefter på knappen "Subscribe" i navigationsbaren, der bliver subscription-formen vist.

Klienten bliver præsenteret for 3 muligheder, at abonnere i 3, 6 eller 12 måneder. Klienten trykker på en af de 3 valgmuligheder, som kalder metoden `CreateSubscriptionAsync` fra businesslogic, med oplysninger om abonnement. Her bliver UI-Modellen for et abonnement konverteret entitets model. Derfra bliver entitets modellen sendt over til metoden `CreateSubscriptionAsync` i Data Access laget.

Data access laget sørger for at oprette abonnemennet i databasen og returnere hvorvidt der er skrevet nye rækker i databasen. Hvis abonnementet er oprettet succesfuldt i databasen bliver der returnerer true gennem alle lagene tilbage til UI hvor der så vil blive vist en besked til klienten at personen nu har medlemskab.



Figur 15 - SD, opret sag

Dette sekvensdiagram illustrerer processen i **UC 4.1** (figur 15), hvor en advokat og sekretær skal kunne oprette sag med tilknyttet klient og advokat.

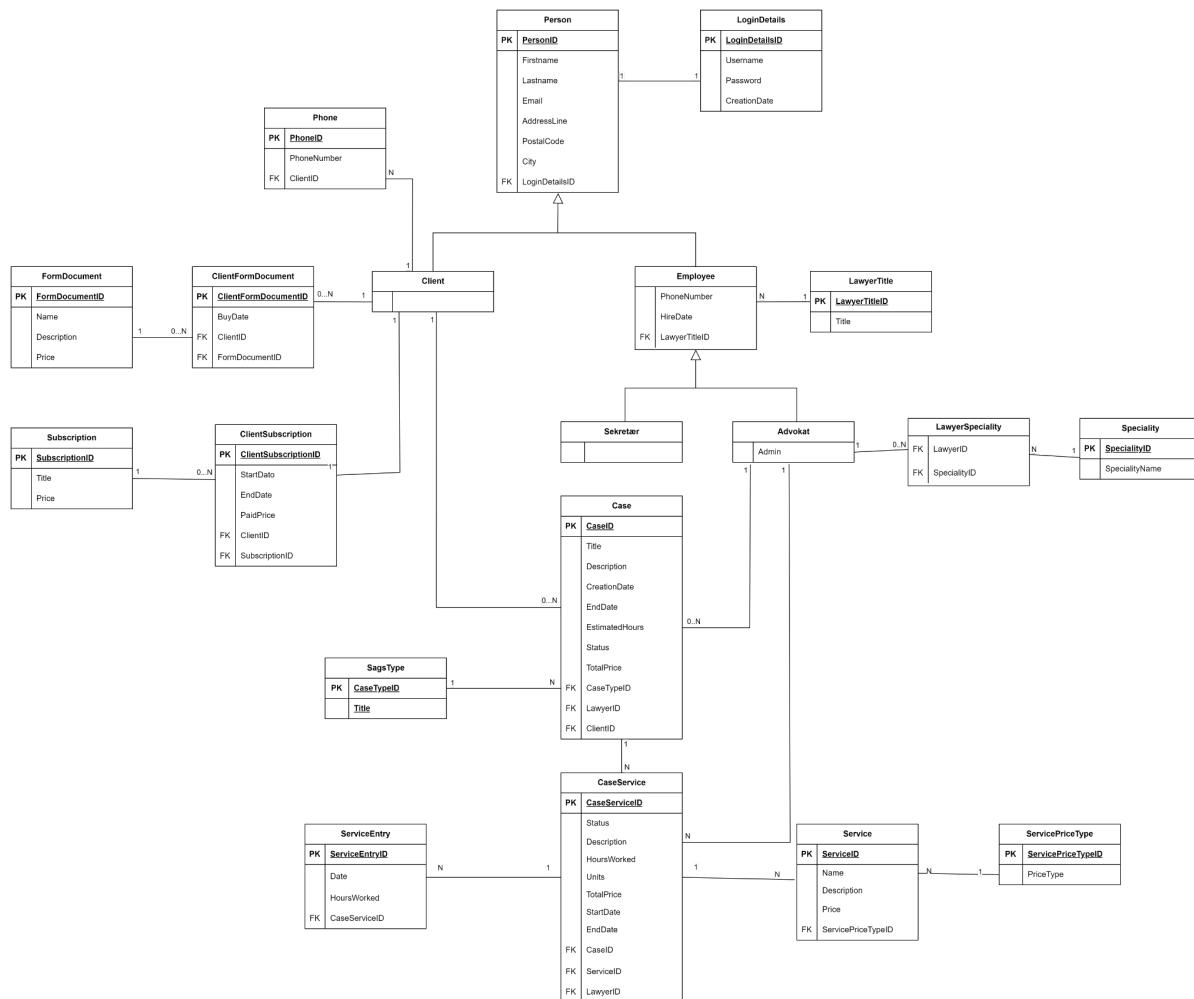
Aktøren starter interaktionen ved at åbne vores program, og logger ind. Derefter vises en frontpage med forskellige knapper og indhold. Aktøren interagerer med frontpagen ved at klikke på knappen Case i navigationsmenuen, hvorefter brugergrænsefladen reagerer på interaktionen ved at navigere til CaseViewPage. Aktøren nавigerer videre ved at trykke på Create Case. Herefter præsenteres de for CreateCasePage, hvor de skal udfylde nogle input-felter med information.

Dernæst skal der tilføjes en Lawyer og Client, disse to interaktioner har samme fremgangsmåde, det sker ved at der trykkes eks. 'Add Lawyer', hvorefter aktøren vælger en lawyer, og dernæst vises den tidligere brugergrænseflade samt advokaten bliver sendt med tilbage.

Aktøren trykker derefter Create Case, hvorefter der kaldes til BusinesLogic med metoden CreateCaseAsync. Businesslogic-laget konverterer UI-modellen til en entitetsmodel og samtidig kalder vores dataaccess-lag, som forsøger at oprette casen i databasen og returnerer en bool. Denne bool sendes igennem lagene tilbage til vores brugergrænseflade, hvor der vil blive vist en messagebox, baseret på om det er vellykket eller ej.

Entity Relationship Diagram (Rasmus)

Her vises vores Entitets-Relations Diagram (ERD) for LawHouse. ERD'et giver en visuel repræsentation af databasestrukturen og den sammenhæng der er mellem vores entiteter.



Figur 16 - ERD

I vores ERD er der en overordnet entitet ‘Person’ som indeholder fælles attributter for alle brugerne der er oprettet i systemet. Herfra specialiseres Person ned til Client og Employee, som yderligere specialiseres til mere specifikke arbejdsmråder som Lawyer og Secretary.

Diagrammet illustrerer disse hierarkiske relationer ved at bruge arve pile mellem entiteterne. Ved at inkludere disse arveforhold mellem entiteterne, skaber det en tydelig struktur af hierarkiet, hvilket gør det nemmere at forstå hvordan dataen er organiseret og yderligere hvordan de forskellige brugere er opdelt og differentieret i systemet.

Alt i alt giver ERD'et en struktureret og beskrivende visualisering af LawHouse opbygning af database struktur, hvilket er med til at sikre et effektivt databasedesign og en forståelse af systemets funktionalitet.

Mapping af ERD (Lucas)

Vi har i gruppen gjort brug af Code First tilgangen med Entity Framework. Dette tillader os at oprette tabellerne direkte i C# kode hvorefter entity framework genererer database strukturen.

Så vi har kørt igennem vores ERD fra en ende og defineret alle klasser i C# og tilføjet Data Notations. Vi har tilføjet alle relationerne i C# med hjælp fra Microsoft learn^{25 26 27}.

Tilgangen med Entity framework har gjort det meget nemt for os at oprette databasen.

TPT mapping strategy er også blevet brugt, Læs mere her²⁸

Database

Database udbyder (Bilal)

Da vi var tilfredse med strukturen og funktionaliteten af databasen, valgte vi at benytte os af serviceudbyderen “Simply²⁹”. Dette gav os en fælles online database, hvilket sikrede konsistent data og gjorde det muligt for os at teste yderligere tilgang til databasen gennem programmet. Ved at skifte til en ekstern database udbyder kunne vi også simulere et mere realistisk produktionsmiljø, hvor flere brugere har adgang til samme datakilde.

Denne ændring har forbedret vores testmuligheder og givet os en mere robust platform til at identificere og løse problemer, der kunne opstå under udviklingen. Derudover har det gjort samarbejder i teamet lettere, da vi alle havde adgang til de samme opdaterede data.

App.Config (Bilal)

I vores konfigurationsfil “App.Config”, har vi valgt at inkludere vores lokale connectionStrings, så det er nemmere at skifte mellem brug af servere, vores Google API-nøgle ligger også herinde for nem adgang.

²⁵ [One-to-many relationships - EF Core | Microsoft Learn](#)

²⁶ [One-to-one relationships - EF Core | Microsoft Learn](#)

²⁷ [Many-to-many relationships - EF Core | Microsoft Learn](#)

²⁸ [Table Per Type - TPT](#)

²⁹ [Simply.com](#)

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add
      name="Bilal"
      connectionString="Server=BILAL-KINALI;Database=LawHouseTest;Trusted_Connection=True;Encrypt=false;" />
    <add
      name="Lucas"
      connectionString="Server=DESKTOP-ANPNVL3;DataBase=LawHouse;Trusted_Connection=true;Encrypt=False;" />
    <add
      name="Simply"
      connectionString="Server=mssql16.unoeuro.com;Database=lawhouseblr_dk_db_lawhouse;User Id=lawhouseblr_dk;Password=km5xFBGRe2pErnDcg6h3;Encrypt=False;" />
  </connectionStrings>
  <appSettings>
    <add key="GoogleApiKey" value="AIzaSyDeUkeA7a7NReydk45u0sy8qnon_YjiLQQ"/>
  </appSettings>
</configuration>

```

Figur 17 - Konfigurationsfil, App.Config

Vi er opmærksomme på, at dette ikke er ideelt med hensyn til sikkerhed, da API-nøgler og adgangskoder bør lagres og beskyttes evt. ved hjælp af andre services og GitIgnore³⁰. Det kunne også tænkes at gøre brug af forskellige konfigurationsfiler, for at adskille test og udvikling, men da dette projekt er et skoleprojekt, har vi valgt denne løsning for enkelhedens skyld.

Entity Framework (Lucas)

Entity Framework er ORM Framework som bruges til at opbygge og administrere databaser. Vi har valgt at bruge EF til med Code first tilgang til at bygge vores database. Det giver os muligheden for at oprette databasen ved hjælp af C# klasser, som entity framework automatisk bruger til at generere databasen.

Når det handler om at hente data fra databasen, giver Entity framework også muligheden for en sikker, mere letlæselig og objektorienteret tilgang. Når data hentes fra databasen kan LINQ bruges som er meget letlæselig kontra SQLClient. Derudover giver EF os også en god mekanisme til at undgå SQL-Injections³¹

Table per type - TPT (Lucas)

Vi har til opbygning af databasen brugt TPT mapping strategy.

TPT er en “inheritance” strategi hvor at hver klasse i et arve hierarki får oprettet hver sin tabel, og de afledte klassers tabeller får en fremmednøgle der referer til baseklassens

³⁰ [Ignoring files - GitHub Docs](#)

³¹ [Prevent SQL Injection With EF Core, Dapper, and ADO.NET](#)

primære nøgle³². Vi har i vores database implementeret Arv fra Person til ansatte og klienter, illustreret i vores ERD³³. Så i vores tilfælde er det smart at der er brugt TPT så alle klasser i hierarkiet får deres egen klasse med fremmednøgler. Uden TPT ville både Client, Lawyer og Secretary indeholde alle person attributterne som skaber en masse redundans, og her hjælper TPT også med at overholde Normaliserings reglerne. En ulempe ved TPT er at det kan gøre databaseforespørgsler mere komplekse, da det kan kræve en masse joins for at hente eksempelvis alt data forbundet med en klient.

SQL Client (Lucas)

Der er blevet bedt om i casen at tilgangen til databasen for 2 tabeller skal kodes i SqlClient.

Vi startede med at lave alt database tilgang via Entity Framework.

Vi har derefter valgt at kode ClientDbAccess om til SQLClient da denne inkluderer tilgang til både Clients, Phones, logindetails og ClientSubscriptions tabellerne.

Da der er oprettet interface til alle vores dbAccess filer har det været nemt at udskifte ClientDbAccess med ClientDbAccessSqlClient da den implementere interfacet og der i stedet kodes med SQLClient.

Dette gør også at når ClientDbAccess instantieres i BusinessLogic laget oprettes der en instans af type IClientDbAccess og så kan der nemt skiftes mellem ClientDbAccess (kodet med EF) og ClientDbAccessSqlClient(kodet med SQLClient).

Herunder vil metoden CreateClientAsync(client) kodet i SQLClient blive dokumenteret.

Der bliver gjort brug af SqlParameter alle steder der kræver parametre for at undgå Sql Injections.

³² [Table Per Type Inheritance](#)

³³ [ERD](#)

CreateClientAsync(client)

Metoden er delt op i 3 billeder da den er på rimelig mange linjer (figur 18,1 - figur 18,2 - figur 18,3)

```

<references>
public async Task<bool> CreateClientAsync(Client client)
{
    using SqlConnection dbConn = new SqlConnection(connString);
    {

        try
        {
            //aabn db
            await dbConn.OpenAsync();

            //start transaction
            using (DbTransaction transaction = await dbConn.BeginTransactionAsync())
            {

                try
                {
                    //insert ind i login tabel
                    string insertLoginQuery = "INSERT INTO LoginDetails VALUES (@UN, @PW, @CD);"
                        + "SELECT SCOPE_IDENTITY()"; //selecter primary key

                    int loginDetailsID;

                    //opret command
                    using SqlCommand createLoginCMD = new SqlCommand(insertLoginQuery, dbConn, (SqlTransaction)transaction);
                    {
                        createLoginCMD.Parameters.AddRange(new SqlParameter[]
                        {
                            new SqlParameter("@UN", client.LoginDetails.Username),
                            new SqlParameter("@PW", client.LoginDetails.Password),
                            new SqlParameter("@CD", DateTime.Now),
                        });
                        //execute query
                        loginDetailsID = Convert.ToInt32(await createLoginCMD.ExecuteScalarAsync());
                    }
                }
            }
        }
    }
}

```

Figur 18,1 - CreateClientAsync() Del 1

Det første der sker er at vi opretter en ny SqlConnection med vores connection string og så åbner vi for database forbindelsen.

Så starter vi en transaction. For at oprette en klient skal der tilføjes data til LoginDetails, Persons, Clients og Phones tabellen. Da der skal tilføjes til flere tabeller er det smart med en transaction således at hvis der sker fejl i oprettelse ved en af tabellerne at den kan “rollback” så der ikke eksempelvis bliver oprettet en client uden logindetails.

Der startes med at blive indsats data i logindetails tabellen. I queryien er indsats linjen “SELECT SCOPE_IDENTITY()” denne command henter efter oprettelse af logindetails primary key for denne, så den kan bruges som foreign key senere.

Commanden bliver derfor executed ved brug af ExecuteScalar der executer query og retunerer første row der bliver læst (SCOPE_IDENTITY) som gemmes i int loginDetailsID

```

//INSERT ind i person tabellen
string insertPersonQuery = "INSERT INTO Persons VALUES (@FN, @LN, @EM, @AL, @PC, @C, @LDID); " +
    "SELECT SCOPE_IDENTITY()"; //selecter primary key

int personID;

//cmd create person
using DbCommand createPersonCMD = new SqlCommand(insertPersonQuery, dbConn, (SqlTransaction)transaction);
{
    createPersonCMD.Parameters.AddRange(new SqlParameter[]
    {
        new SqlParameter("@FN", client.Firstname),
        new SqlParameter("@LN", client.Lastname),
        new SqlParameter("@EM", client.Email),
        new SqlParameter("@AL", client.AddressLine),
        new SqlParameter("@PC", client.PostalCode),
        new SqlParameter("@C", client.City),
        new SqlParameter("@LDID", loginDetailsID),
    });

    //execute query
    personID = Convert.ToInt32(await createPersonCMD.ExecuteScalarAsync());
}

//INSERT ind i Client tabellen
string insertClientQuery = "INSERT INTO Clients VALUES (@PID);";

using DbCommand createClientCMD = new SqlCommand(insertClientQuery, dbConn, (SqlTransaction)transaction);
{
    createClientCMD.Parameters.AddRange(new SqlParameter[]
    {
        new SqlParameter("@PID", personID),
    });

    //execute query
    bool success = await createClientCMD.ExecuteNonQueryAsync() > 0;
    if (!success)
    {
        await transaction.RollbackAsync();
        return false;
    }
}

```

Figur 18,2 - CreateClientAsync() Del 2

Efter oprettelse af LoginDetails bliver der oprettet en person i databasen (Figur x,2) her gøres brug af primary key for logindetails, da person tabellen har en foreignkey dertil.

Igen gøres brug af "SELECT SCOPE_IDENTITY()" så primary key fra den oprettede Person gemmes. Som bruges nedenunder til oprettelse af klienten i Clients tabellen.

```
//INSERT ind i Phones tabellen
string insertPhonesQuery = "INSERT INTO Phones VALUES (@PN, @CID)"; // @CID = ClientID/PersonID

foreach (Phone phone in client.Phones)
{
    using DbCommand createPhonesCMD = new SqlCommand(insertPhonesQuery, dbConn, (SqlTransaction)transaction);
    {
        createPhonesCMD.Parameters.AddRange(new SqlParameter[]
        {
            new SqlParameter("@PN", phone.PhoneNumber),
            new SqlParameter("@CID", personID),
        });

        //execute query
        bool success = await createPhonesCMD.ExecuteNonQueryAsync() > 0;
        if (!success)
        {
            await transaction.RollbackAsync();
            return false;
        }
    }
}

//commit
await transaction.CommitAsync();

return true;
}
catch (Exception)
{
    //rollback
    await transaction.RollbackAsync();
    return false;
}
}

catch (Exception)
{
    return false;
}
finally
{
    //luk db
    await dbConn.CloseAsync();
}
```

Figur 18,3 - CreateClientAsync() Del 3

Til sidst skal klientens telefonnumre også tilføjes til phones tabellen.

En klient har en `ICollection` af `Phones`, der loopes igennem denne collection og så bliver der inserted til `Phones` tabellen, og igen bruges foreign key fra `Person`.

Der er sat en try catch blok rundt om alt inde i transaction så hvis at der kommer en exception så kører den rollback på transactionen, ellers committer den til databasen.

Trigger (Rasmus)

I systemet har vi gjort brug af en SQL trigger, som vi har oprettet med det formål at automatisere opdateringen af den totale pris på en sag. Triggeren er navngivet 'updateCaseTotalPrice'.

Formålet med vores trigger er, at vi sikre at den samlede pris for en sag altid bliver

opdateret, idet den kører automatisk. Dette sikrer datakonsistens og derved gør det også at vi ikke har brug for behovet for manuelle opdateringer, eller flere databasekald fra vores system, hvilket også mindsker risikoen for eventuelle fejl.

```
CREATE TRIGGER update_case_total_price
ON CaseServices
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @total_price FLOAT;

    SELECT @total_price = SUM(TotalPrice) FROM CaseServices WHERE CaseID IN (SELECT CaseID FROM inserted);

    UPDATE Cases SET TotalPrice = @total_price WHERE CaseID IN (SELECT CaseID FROM inserted);
END
```

Figur 19 - SQL trigger

Triggeren oprettes på tabellen CaseServices og udløses efter hver INSERT og UPDATE. Vi deklarerer en variabel @total_price af typen float, som bliver brugt til at gemme den samlede pris. Derefter bruger vi select til at beregne den samlede pris, idet den summere totalprice fra alle de CaseServices som er connected til en sag. Dette gøres ved brug af SUM(TotalPrice) og en subquery, der henter CaseID fra den tabel der bliver indsat i (inserted). UPDATE opdaterer TotalPrice i Cases-tabellen, dette gøres ved brug af den deklarerede variabel, som vi tilegnede i starten.

Overvejelse til databasekald

.Include med Entity Framework (Lucas)

```
public async Task<List<Client>> GetClientsAsync()
{
    try
    {
        List<Client> clients = await db.Clients
            .Include(c => c.ClientSubscriptions)
            .Include(c => c.Phones)
            .ToListAsync();
        return clients;
    }
    catch (Exception)
    {
        return new List<Client>();
    }
}
```

Figur 20 - Join med Entity Framework

Entity Framework tilbyder en mulighed ift. udførelsen af joins ved brug af “.Include()” metoden (Figur 20) når dine klasser er mappet korrekt.

Vi blev i starten af projektet enige om ikke at .Include alt for mange collections, da det nok ville kræve nogle tungere databasekald. I stedet ville vi gøre brug af ”manuelt” Lazy loading, hvis vi på et tidspunkt fik brug for clientens ICollection af telefonnumre så at lave et yderligere kald til databasen der henter alle telefonnumre det matcher clientens ID.

Så hele programmet blev kodet op på denne måde.

En opgave fra casen var at kunne udskrive en sigende rapport angående en sag til en tekstfil. Her ville vi gerne følge GRASP principperne og derved ligge denne metode i selve klassen/modellen ”sag”.

Her blev det et problem at kalde Print() metoden udfra en instans af en sag da alle data ikke var tilgængelig.

Her blev vi enige om at det nok havde været smart at gøre brug af .include eller af entity frameworks egen Lazy loading funktion, i stedet for at lave 5 database kald for at kunne udskrive alt data for en sag.

Vi har derefter fået rettet nogle af vores database kald til at ”.Include” flere properties, dette er dog ikke tilfældet alle steder.

En fremtidig opdatering af systemet ville indeholde flere .Includes for nemmere datahåndtering i systemet.

Program C#

Asynkron Programmering (Rasmus)

I vores system har vi implementeret asynkron programmering, på tværs af alle vores lag. I UI-laget kaldes de asynkrone metoder fra BusinessLogic-laget og venter på den vha. ‘await’. Dette sikrer at vores UI-tråd ikke blokeres.

I BusinessLogic kører de asynkrone operationer, og venter på kaldet til DataAccess-laget uden at blokere den aktuelle tråd.

I DataAccess-laget udføres de egentlige database operationer asynkront ved brug af Entity Framework. Her bruges metoder som ‘AddAsync’ og ‘SaveChangesAsync’, for at udføre database operationerne asynkront. Dette sørger for at vi ikke blokere tråden.

Yderligere gør vi også brug af at deaktivere buttons ved databasekald for at sikre at kun en operation kører af gangen. Dette forhindrer at flere ens tråde igangsættes på samme tid, da knappen først bliver aktiveret igen efter at tråden er kørt færdig.

Vores implementering af asynkron programmering har stor indflydelse på at skabe en responsiv brugergrænseflade, som forbedrer brugeroplevelsen.

Modeller og konvertering (Rasmus)

I vores system benytter vi os af en Modelconverter klasse, denne har ansvar for at konvertere mellem UI-modeller og vores entitetsmodeller. Formålet ved at benytte denne konvertering, er at vi på den måde opnår en afkobling mellem den data vi får ind fra databasen og det som skal præsenteres i vores UI.

Klassen er konstrueret på den måde, at den indeholder en konvertering metode for hvert model som skal vises i UI, og ligeledes for at konvertere tilbage, når vi skal bruge det i vores DataAcces-lag.

```

210  public EmployeeUI ConvertFromEmployeeEntity(Employee employeeE)
211  {
212      EmployeeUI employeeUI = new EmployeeUI()
213      {
214          PersonID = employeeE.PersonID,
215          Firstname = employeeE.Firstname,
216          Lastname = employeeE.Lastname,
217          PhoneNumber = employeeE.PhoneNumber,
218          Email = employeeE.Email,
219          AddressLine = employeeE.AddressLine,
220          PostalCode = employeeE.PostalCode,
221          City = employeeE.City,
222          HireDate = employeeE.HireDate,
223          LawyerTitle = employeeE.LawyerTitle.Title,
224
225          //foreign keys
226          LoginDetailsID = employeeE.LoginDetailsID,
227          LawyerTitleID = employeeE.LawyerTitleID,
228      };
229      return employeeUI;
230  }

```

Figur 21 - ConvertFromEmployeeEntity

Denne metode viser et eksempel, hvor vi konverterer en Lawyer til en LawyerUI. Konverteringen af denne model er nødvendig for at kunne vise data på vores brugergrænseflade.

Ved at afkoble vores UI-modeller fra vores entitetsmodeller, gør vi systemet mere robust over for eventuelle fejl, da det oftest vil forekomme i vores UI og dermed ikke påvirke data i vores DataAccess-lag. Yderligere kan vi også tilpasse vores UI-modeller med ekstra attributter og metoder som kun er relevante ift. vores brugergrænseflade.

Ved at benytte os af vores Modelconverter klasse, sikre vi at vi effektivt kan konvertere alt data mellem vores DataAccess-lag og UI. Dette bidrager til at vi får en mere struktureret tilgang og at det bliver nemmere at eventuelt udvide det i fremtiden, samt vedligeholde.

Validering (Rasmus)

Vi har i vores system implementeret en Validation-mappe i vores BusinessLogic-lag som indeholder vores forskellige validerings klasser,. Valideringen af data sker på to forskellige niveauer, både i vores UI og BusinessLogic-lag.

I vores UI sker valideringen af hvert enkelt input-felt, når brugeren indtaster data. Dette har til formål at opfange fejlene tidligt og sørge for at de bliver rettet, inden de sendes videre. Derved giver vi ikke brugeren adgang til at udføre operationer med korrupte data.

I vores BusinessLogic-lag foretages der endnu en validering på hele objektet. Dette sker efter, at data er blevet sendt fra vores UI. Derved sikrer vi yderligere, at dataene er behandlet korrekt og opnår en ekstra sikkerhedskontrol, inden de sendes videre til vores DataAccess-lag.

```

13  public PersonValidator()
14  {
15      // Allows upper/lowercase + space (\s) + hyphen (-)
16      vName = new Regex("^[A-Za-zÆØÅæøåÜÖÜö-]+\\s?[A-Za-zÆØÅæøåÜÖÜ-]+$");
17      // Same as name + digits (0-9) + comma (,) + period (.)
18      vAddress = new Regex("^[A-Za-zÆØÅæøåÜÖÜö-9]\\s-, .]+$");
19      vDigit = new Regex("^-[0-9]+$");
20      // Left of @
21      // Right of @
22      // After a period min 2 letters
23      vEmail = new Regex("^[A-Za-z0-9-_]+@[A-Za-z0-9-.]+\\.[A-Za-z]{2,}$");
24      vPassword = new Regex("^[A-Za-zÆØÅæøå0-9-!.{4,30}$");
25  }

```

Figur 22 - PersonValidator Regex

I dette eksempel fra vores PersonValidator-klasse anvender vi regex til at validere forskellige attributter som navn, adresse, email og telefonnummer. Vi har valgt at implementere regex, idet det giver en nøjagtig og effektiv validering af data. Det er specielt effektivt ved brug af valideringen på email, idet det tillader fleksibilitet, da vi kan oprette et pattern som email skal følge.

Ved at have flere lag af validering minimerer vi risikoen for at der bliver opereret med forkerte data. Dette betyder også at hvis fejl skulle omgå vores validering i UI-laget, vil BusinessLogic stadig udføre en validering, hvilket tilføjer et ekstra lag af sikkerhed.

Det kunne have været en mulighed at gøre vores validerings klasser statisk, idet det vil have gjort metoderne lettere tilgængelige, da der ikke skal oprettes en instans. Det vil samtidig gøre vores kode mere simpel.

Postnummer/By fra CSV (Bilal, Lucas & Rasmus)

```

public static class GetCityFromPostalCode
{
    11 references | 0/4 passing
    public static string SetCityFromPostalCode(string postal)
    {
        PersonValidator pValidator = new PersonValidator();

        if (!pValidator.ValidPostalCode(postal))
            return "-";

        string location = "Resources\\postnumre.csv";

        try
        {
            string[] lines = File.ReadAllLines(location);

            foreach (string line in lines.Skip(1))
            {
                string[] temp = line.Split(',');
                if (temp[0] == postal)
                {
                    return temp[1];
                }
            }
            return "-";
        }
        catch (Exception)
        {
            return "-";
        }
    }
}

```

Figur 23 - SetCityFromPostal()

Vi har implementeret en kort statisk klasse med en metode der henter By navn ud fra postnummeret fra en csv fil. (figur 23)

Metoden har som input parameter string postnummer. Det metoden gør er at den henter data fra en CSV fil “postnumre.csv” hvor at kolonne [0] er postnummer og kolonne[1] er by navnet. Den går igennem alle rækker i filen indtil den rammer en række hvor at postnummeret matcher og så tager den index 1 fra denne række som er bynavnet og returnerer dette.

Login (Bilal)

Programmet starter op med en login-side³⁴, hvor brugere har mulighed for at oprette konto, gendanne glemt kodeord (simulering) og logge ind. For enkelthedens skyld har vi besluttet, at både ansatte og brugere logger ind samme sted. Det ville dog også have været muligt at

³⁴ [Screenshot af LoginPageView](#)

have en separat sektion for ansatte, eventuelt med en anden database, for at adskille kunder og ansatte.

Login-siden er en vigtig del af programmet, da den sikrer at kun brugere med konto får adgang til systemet, og bruger privilegier kan tilpasses. Det er designet med fokus på sikkerhed og brugervenlighed.

Funktionen vis/skjul adgangskode forbedrer brugeroplevelsen og gør processen mere sikker:

```

1 reference
123  private void PBoxEye_Click(object? sender, EventArgs e) // Skjul visning af brugerens kodeord
124  {
125      if (pboxEye.IconChar == FontAwesome.Sharp.IconChar.Eye)
126      {
127          pboxEye.IconChar = FontAwesome.Sharp.IconChar.EyeSlash;
128          txtPassword.PasswordChar = '\0'; // Deaktiver funktionen
129      }
130      else
131      {
132          pboxEye.IconChar = FontAwesome.Sharp.IconChar.Eye;
133          txtPassword.PasswordChar = '\u2022'; // Unicode for "bullet point" karakteren •
134      }

```

Figur 24 - LoginPageView | Vis/skjul adgangskode

Adgangskoden er som standard indstillet til at skjule kodeordet, hvis man ønsker at få det vist, kan man klikke på “øjet” og alt efter om ikonet er sat til “Eye” / “EyeSlash”, vises eller skjules koden med symbolet “•”.

Når brugernavnet og adgangskoden er indtastet i korrekt format, kan der trykkes på “Login” og denne kodeblok vil blive eksekveret:

```

1 reference
98   private async void BtnLogin_Click(object? sender, EventArgs e)
99   {
100      // Deaktiver Login knap, så det ikke er muligt at spamme knappen ved Login forsøg
101      btnLogin.Enabled = false;
102
103      // Returner LoginDetailsID og tjek for korrekt loginoplysninger
104      int userID = await loginBL.CheckUsernameAndPasswordAsync(txtUsername.Text, txtPassword.Text);
105
106      btnLogin.Enabled = true; // Aktiver knap efter endt Task
107
108      if (userID > 0)
109      {
110          new FrontPageView(userID, this).Show(); // Åbn programmets main page og medtag ID'et på bruger
111          Hide(); // Skjul loginpage
112          txtPassword.Text = string.Empty; // Slet koden
113      }
114      else
115      {
116          // Neutral besked ved forkert loginoplysninger, for at undlade at informere brugeren om
117          // brugernavn/email eksisterer i systemet
118          MessageBox.Show("Invalid username or password. Please try again.",
119                          "Login Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);
120      }
121  }

```

Figur 25 - LoginPageView | Login-klik

LoginBL er instantiseret i constructoren og async-metoden (linje 104) kaldes for at verificere brugerens login oplysninger, der tager brugernavn og adgangskode som inputparametre:

```

5   <namespace BusinessLogic
6   {
7     <references
8       public class LoginBL
9     {
10       ILoginDbAccess dbAccess;
11       PersonValidator pValidator;
12     }
13     public LoginBL()
14     {
15       dbAccess = new LoginDbAccess();
16       pValidator = new PersonValidator();
17     }
18     <reference
19     public async Task<int> CheckUsernameAndPasswordAsync(string username, string password)
20     {
21       if (!pValidator.ValidEmail(username) || !pValidator.ValidPassword(password))
22         return 0;
23       return await dbAccess.CheckUsernameAndPasswordAsync(username, password);
24     }

```

Figur 26.1 - LoginBL / CheckUsernameAndPasswordAsync

Selvom vi input-validerer i brugergrænsefladen, bliver der sørget for at valideringen også sker i BusinessLogic-laget, så ugyldig format begrænses så meget som muligt, inden vi når ud til DataAccess-laget. Dette er med til at øge sikkerheden og gør udskifteligheden af programmet nemmere.

Linjerne 20-21 sørger for at afslutte metoden tidligt, hvis ikke parametrene er valide, ved at returnere værdien 0, ellers fortsætter processen:

```

5   <namespace DataAccess
6   {
7     <references
8       public class LoginDbAccess : ILoginDbAccess
9     {
10       LawHouseDbContext db;
11       public LoginDbAccess()
12     {
13       db = new LawHouseDbContext();
14     }
15     <reference
16     public async Task<int> CheckUsernameAndPasswordAsync(string username, string password)
17     {
18       try
19       {
20         LoginDetails? user = await db.LoginDetails
21           .FirstOrDefaultAsync(u => u.Username == username && u.Password == password);
22         return user != null ? user.LoginDetailsID : 0;
23       }
24       catch (Exception)
25       {
26         return 0;
27       }
28     }

```

Figur 26.2 - LoginDbAccess / CheckUsernameAndPasswordAsync

Hvis databasen ikke kan tilgås eller en anden fejl skulle opstå (try-catch), bliver der returneret 0. Linje 22, returnerer brugerens primære nøgle LoginDetailsID, hvis ikke det er null.

Det returnerede int-værdi i Figur 26.2, bliver nu tilgængelig på linje 104.

Ved vellykket login: Hvis ID'et er større end 0, åbnes FrontPageView og brugerens ID videregives. Adgangskoden i feltet bliver ryddet og login-siden bliver skjult til senere brug, hvis brugeren ønsker at logge ud.

Ved fejlet login: Brugeren får feedback i form af MessageBox, hvor meddelelsen gives på en sikker måde, uden at afsløre om brugernavnet findes i systemet.

Har man glemt kodeordet, kan man trykke på "Forgot Password?" og gendanne dette.

- *Denne funktion skal anses som en simulering, da vi ikke har implementeret forsendelse af e-mail for at holde det simpelt, hvilket ville være normal praksis at gøre. Det er dog stadig muligt at få vist koden på skærmen ved indtastning af valid email i systemet, kun med henblik på at præsentere funktionen.*

Ved at gøre login-siden intuitiv og brugervenlig, skaber det værdi for kunden men også virksomheden, ikke nok med brugertilfredsheden øges, er det også med til at formindske unødvendige henvendelser vedr. tilgang til ens konto.

Oprettelse som kunde (Bilal)

Ønsker man at oprette en ny konto, kan der klikkes på "Register" som vil åbne en ny form CreateUserView³⁵, med felter til personoplysninger som skal udfyldes. Når email-feltet er udfyldt, bedes brugeren om at bekräfte dette ved at indtaste e-mailen igen i næste felt, for at sikre tastefejl ikke skaber forvirring.

Vi har besluttes os for, at username skal være brugerens indtastede e-mailadresse, hvilket bliver udfyldt automatisk efter e-mail detaljerne er indtastet korrekt:

³⁵ [Screenshot af CreateUserView](#)

```

239     ↓   | 1 reference
240     ↓   | private void TxtEmailConfirm_TextChanged(object? sender, EventArgs e)
241     ↓   |
242     ↓   | {
243     ↓   |     ValidateEmails(txtEmailConfirm, txtEmail);
244     ↓   |
245     ↓   | 1 reference
246     ↓   | private void TxtEmail_TextChanged(object? sender, EventArgs e)
247     ↓   |
248     ↓   | {
249     ↓   |     ValidateEmails(txtEmail, txtEmailConfirm);
250     ↓   |
251     ↓   | 2 references
252     ↓   | private void ValidateEmails(TextBox email, TextBox emailConfirm)
253     ↓   |
254     ↓   | {
255     ↓   |     bool isEmailValid = pValidator.ValidEmail(email.Text);
256     ↓   |     email.ForeColor = isEmailValid ? validFormat : invalidFormat;
257     ↓   |
258     ↓   |     bool isConfirmEmailValid = pValidator.ValidEmail(emailConfirm.Text);
259     ↓   |     emailConfirm.ForeColor = isConfirmEmailValid ? validFormat : invalidFormat;
260     ↓   |
261     ↓   |     if (isEmailValid && isConfirmEmailValid)
262     ↓   |     {
263     ↓   |       if (email.Text == emailConfirm.Text)
264     ↓   |       {
265     ↓   |         txtUsername.Text = email.Text;
266     ↓   |         email.ForeColor = validFormat;
267     ↓   |         emailConfirm.ForeColor = validFormat;
268     ↓   |       }
269     ↓   |     }
270     ↓   |     else
271     ↓   |     {
272     ↓   |       txtUsername.Text = string.Empty;
273     ↓   |       if (!string.IsNullOrEmpty(emailConfirm.Text))
274     ↓   |       {
275     ↓   |         email.ForeColor = invalidFormat;
276     ↓   |         emailConfirm.ForeColor = invalidFormat;
277     ↓   |       }
278     ↓   |     }
279     ↓   |     else
280     ↓   |     {
281     ↓   |       txtUsername.Text = string.Empty;
282     ↓   |     }
283     ↓   |     UpdateCreateButtonState();
284     ↓   |
285   }

```

Figur 27.1 - CreateUserView / ValidateEmails()

Hvis både email og bekræft e-mail felterne er indtastet korrekt og de er ens, assigner linje 261 username-feltet med denne email. Brugeren kan tilføje et alternativt telefonnummer, hvor det er muligt at tilføje flere i sin profil efter oprettelsen af kontoen.

Når brugeren har indtastet alle felter og valgt en adgangskode (der findes tooltip på accepteret input), kan der trykkes på “Create”, bliver CreateClientAsync() metoden kaldt:

```

66     |   1 reference
67     |   private async Task<bool> CreateClientAsync()
68     |   {
69     |   // Opret Client UI
70     |   ClientUI client = new ClientUI()
71     |   {
72     |   Firstname = txtFirstname.Text,
73     |   Lastname = txtLastname.Text,
74     |   Email = txtEmailConfirm.Text.ToLower(),
75     |   AddressLine = txtAddress.Text,
76     |   PostalCode = int.Parse(txtPostal.Text),
77     |   City = txtCity.Text,
78     |   };
79
80     |   // Opret LoginDetails UI
81     |   LoginDetailsUI loginDetails = new LoginDetailsUI()
82     |   {
83     |   Username = txtUsername.Text.ToLower(),
84     |   Password = txtPasswordConfirm.Text,
85     |   CreationDate = DateTime.Now, // Oprettelses dato sat til Now
86     |   };
87
88     |   List<PhoneUI> phoneUIs = new List<PhoneUI>();
89
90     |   // Opret Phone UI
91     |   PhoneUI phoneUI = new PhoneUI { PhoneNumber = int.Parse(txtPhoneMain.Text) };
92
93     |   phoneUIs.Add(phoneUI);
94
95     |   // Opret PhoneAlt UI - Alternativ telefonnummer tilføjes til listen
96     |   if (!string.IsNullOrWhiteSpace(txtPhoneAlt.Text) && pValidator.ValidPhone(txtPhoneAlt.Text))
97     |   {
98     |   PhoneUI phoneUIAlt = new PhoneUI { PhoneNumber = int.Parse(txtPhoneAlt.Text) };
99     |   phoneUIs.Add(phoneUIAlt);
100    |
101    }

```

Figur 27.2 - CreateUserView / CreateClientAsync()

Der oprettes UI-modeller til ClientUI, LoginDetailsUI og liste af PhoneUI, hvor attributterne får deres værdi ud fra tekstdoboksfelterne. CreationDate bliver sat til DateTime.Now, så det bliver håndteret automatisk.

Da vores validators tester for værdierne, bruger vi int.Parse og ikke den lidt mere påpasselige int.TryParse, som tester for om værdien kan konverteres.

Efter modellerne er blevet oprettet, kaldes clientBL metoden CreateClientAsync med modellerne som input parameter:

```

56      4 references
57
58  public async Task<bool> CreateClientAsync(ClientUI clientUI, LoginDetailsUI loginDetailsUI, List<PhoneUI> phoneUIs)
59  {
60      if(!pValidator.ValidClient(clientUI))
61      {
62          return false;
63      }
64
65      foreach (PhoneUI phone in phoneUIs)
66      {
67          bool succes = pValidator.ValidPhone(phone.PhoneNumber.ToString());
68          if (!succes)
69          {
70              return false;
71          }
72
73          if (!pValidator.ValidPassword(loginDetailsUI.Password))
74          {
75              return false;
76          }
77
78          Client tempC = modelConverter.ConvertFromClientUI(clientUI);
79
80          LoginDetails templ = modelConverter.ConvertFromLoginDetailsUI(loginDetailsUI);
81
82          List<Phone> phoneList = new List<Phone>();
83
84          foreach(PhoneUI phoneUI in phoneUIs)
85          {
86              Phone tmp = modelConverter.ConvertFromPhoneUI(phoneUI);
87              phoneList.Add(tmp);
88          }
89          tempC.Phones = phoneList;
90          tempC.LoginDetails = templ;
91
92      }
93  }

```

Figur 27.3 - ClientBL / CreateClientAsync()

Ligesom i Figur 26.3 LoginBL, afslutter vi metoden tidligt, i tilfælde af at valideringen af modellerne ikke er tilfredsstillende.

UI-modellerne bliver konverteret til entity-modeller og samlet i tempC, som er Client-entiteten og sendt videre ved at kalde dbAccess.CreateClientAsync metoden.

- Det kunne tænkes at begrænse oprettelse af variablerne og samle dem senere, med en mere optimeret tilgang:

```

tempC.LoginDetails = modelConverter.ConvertFromLoginDetailsUI(loginDetailsUI);
tempC.Phones = phoneUIs.Select(modelConverter.ConvertFromPhoneUI).ToList();

```

Selvom dette forenkler koden og reducerer antallet af midlertidige variabler, gør den originale kode det nemmere at debugge og forstå detaljerne.

For at returnere resultatet, skal vi have klienten oprettet i database, som bliver håndteret af dbAccess:

```

59     |    2 references
60     |    public async Task<bool> CreateClientAsync(Client client)
61     |    {
62     |        try
63     |        {
64     |            await db.AddAsync(client);
65     |            bool success = await db.SaveChangesAsync() > 0;
66     |            db.ChangeTracker.Clear();
67     |            return success;
68     |        }
69     |        catch (Exception)
70     |        {
71     |            return false;
72     |        }
73   }

```

Figur 27.4 - ClientDbAccess / CreateClientAsync()

Client-entiteten bliver nu tilføjet til vores DbContext og først gemt i databasen ved SaveChangesAsync()³⁶ kaldet, som returnerer Task<int> der fortæller hvor mange ændringer der er blevet lavet i databasen, hvilket her vil være over 0, da der bliver tilføjet. Resultatet bliver så svaret på om værdien er større end 0, hvilket er "true", hvis det lykkedes. Try-catch'en sørger for at melde fejl, ved at returnere false i tilfælde af at den fanger exceptions fra linjerne 63-64.

Figur 27.2 viste metoden, som bliver kaldt på linje 107 i vores Create_Click knap event:

```

104    |    1 reference
105    |    private async void BtnCreate_Click(object? sender, EventArgs e)
106    |    {
107    |        btnCreate.Enabled = false;
108    |        bool result = await CreateClientAsync();
109    |        btnCreate.Enabled = true;
110
111        |    if (result)
112        |    {
113            |        MessageBox.Show($"Your account was successfully created!", "Success",
114            |                         MessageBoxButtons.OK, MessageBoxIcon.Information);
115            // Ved successful oprettes, åbnes loginsiden med brugerensemail tastet ind i "username" feltet
116            |        loginPage.GetCreatedUsername(txtEmail.Text);
117            |        this.Close();
118        |    }
119        |    else
120        |    {
121            |        MessageBox.Show("Failed to create client. Please check the entered details and try again.", "Error",
122            |                         MessageBoxButtons.OK, MessageBoxIcon.Error);
123        |    }
    }

```

Figur 27.5 - CreateUserView / CreateClick

Resultatet af oprettelsen afgør nu, hvad kunden skal have returneret som feedback i form af MessageBox. Lykkedes operationen, lukkes oprettelsesformen og ens username bliver automatisk tilskrevet i loginform for nemmere login, ellers får man besked på at prøve igen.

Vi har gennem hele forløbet arbejdet på at gøre det nemt for brugeren ved at begrænse mulige indtastningsfejl. For eksempel bliver by-feltet automatisk udfyldt, så man undgår

³⁶ [DbContext.SaveChangesAsync Method \(System.Data.Entity\) | Microsoft Learn](#)

ugyldige oplysninger. Brugernavnet er begrænset til den indtastede email, og kun et alternativt telefonnummer kan tilføjes, så man undgår at stresse serveren med store datamængder, hvis intentionen skulle være der.

Dette er med til at minimere manuelle database korrektioner, opretholde en mere stabil database ydeevne og øge kundetilfredsheden.

Navigering i forsiden (Lucas, Bilal, Rasmus)

Programmets forside er opbygget således at der er en navigations bar til venstre med en masse knapper til diverse forms/sider og et stort panel (gråt område markeret på screenshot³⁷).

Når man trykker på en knap vil dens form vise ved på panelet ved at add formen som user control til panelet.

for at opnå dette har vi oprettet en metode PnlContextChange(Form f) (Figur x)

```
public void PnlContextChange(Form f)
{
    //clear controls fra panelForm
    pnlContext.Controls.Clear();
    f.TopLevel = false;
    //tilføj form som control til panelet
    pnlContext.Controls.Add(f);
    f.Show();
    pnlContext.Show();
}
```

Figur 28 - Skærmbillede af metode PnlContextChange()

Denne metode tager en form som input. Så clear den alle controls fra panelet. Sætter dens toplevel til false, så det bliver en “child control” og så tilføjer den formen som control og viser denne.

Her vises metoden i brug (Figur 28,1)

³⁷ [Screenshot af forside](#)

```
private void BtnAdminPage_Click(object? sender, EventArgs e)
{
    AdminPageView apv = new AdminPageView();
    PnlContextChange(apv);
    SetNavBtnColor(btnAdminPage);
}
```

Figur 28,2 - Skærbillede af PnlContextChange i brug

Her ser vi på et event der sker når man klikker på knappen “Admin Page” den opretter en ny instans af formen AdminPageView så bliver metoden PnlContextChange kaldt med AdminPageView som input parameter og så bliver formen vist i det grå panel. Herefter bliver SetNavBtnColor metoden kaldt som sætter knappen i navigationsbaren til en anden farve så man kan se hvilken side man er inde på.

Håndtering af Bruger Privilegier (Lucas, Bilal, Rasmus)

I systemet har en klient ikke samme privilegier som en ansat og omvendt. Herunder beskrives hvordan håndtering af dette fungerer i systemet.

Når en bruger er logget ind og FrontPageView åbnes køres GetPersonAsync(int loginDetailsID) (figur 29,1)

```
public async Task GetPersonAsync(int loginDetailsID)
{
    if (currentUser == null)
    {
        currentUser = await personBL.GetPersonAsync(loginDetailsID);
    }

    if (currentUser != null)
    {
        lblLoggedInAs.Text = $"Logged in as: {currentUser.Firstname} {currentUser.Lastname}";

        if (currentUser is ClientUI clientUI)
        {
            this.clientUI = clientUI;
            await SetupClientFormAsync();
        }
        else if (currentUser is LawyerUI lawyer)
        {
            this.lawyerUI = lawyer;
            await SetupLawyerFormAsync();
        }
        else if (currentUser is SecretaryUI secretary)
        {
            this.secretaryUI = secretary;
            await SetupEmployeeFormAsync();
        }
        else if (currentUser is EmployeeUI employeeUI)
        {
            this.employeeUI = employeeUI;
            await SetupEmployeeFormAsync();
        }
    }
}
```

Figur 29,1 - GetPersonAsync()

Denne metode tester først på om currentUser er null. currentUser er af datatypen PersonUI. Hvis currentUser er null så henter den currentUser ved brug af GetPersonAsync() i businessLogic der tager imod et logindetailsID som input (figur 29,2)

```

public async Task<PersonUI> GetPersonAsync(int id)
{
    if (id <= 0)
        return new PersonUI();

    try
    {
        Person person = await dbAccess.GetPersonAsync(id);

        if (person != null)
        {
            if (person is Client)
            {
                ClientUI clientUI = await new ClientBL().GetClientAsync(person.PersonID);
                return clientUI;
            }
            if (person is Lawyer)
            {
                LawyerUI lawyerUI = await new LawyerBL().GetLawyerWithCollectionsAsync(person.PersonID);
                return lawyerUI;
            }
            if (person is Employee)
            {
                EmployeeUI employeeUI = await new EmployeeBL().GetEmployeeAsync(person.PersonID);
                return employeeUI;
            }
        }
        return new PersonUI();
    }
    catch (Exception)
    {
        return new PersonUI();
    }
}

```

Figur 29,2 - GetPersonAsync(), BusinessLogic

Vi tester først på at det sendte id er over 0 ellers returneres en tom person.

Så henter metoden personen fra databasen der matcher på det indsendte LoginDetailsID.

Her tester metoden så på hvilken type af person vi har med at gøre. Er personen eksempelvis en advokat henter den advokaten der matcher på personID fra databasen og retunerer denne til UI.

Hvis man så kigger tilbage til (figur 29,1) GetPersonAsync() fra FrontPageView. Ser vi igen at der bliver testet på hvilken type currentUser, (*der er sendt retur fra GetPersonAsync*), er. I dette eksempel har vi med en lawyer at gøre så den sætter lawyerUI i lig med den person der er kommet retur fra GetPersonAsync(), og så bliver SetupLawyerFormAsync() kaldt (figur 29,3) havde dette været en klient ville SetupClientFormAsync() blive kaldt i stedet.

```
private async Task SetupLawyerFormAsync()
{
    // Knapper
    btnMyPageLawyer.Visible = true;
    btnEmployees.Visible = true;
    btnCase.Visible = true;
    btnClients.Visible = true;
    btnAdminPage.Visible = false;

    SetNavBtnColor(btnMyPageLawyer);

    //set mypage
    LawyerDetailsView lawyerDetailsView = new LawyerDetailsView(lawyerUI.PersonID, true, lawyerUI);
    if (pnlContext.Controls.Count == 0 || pnlContext.Controls[0].GetType() != typeof(LawyerDetailsView))
    {
        PnlContextChange(lawyerDetailsView);
    }

    if (lawyerUI.Admin)
    {
        btnAdminPage.Visible = true;
    }
}
```

Figur 29,3 - SetupLawyerFormAsync()

Det er her vi laver hele setuppet for hvordan forsiden skal se ud.

Da vi i dette tilfælde kigger på et advokat setup gør metoden det at den gør alle de knapper en lawyer har adgang til visible.

Herefter bliver LawyerDetailsView tilføjet som control til context så denne side bliver vist.

(*LawyerDetailsView fungerer også som "My page" til en advokat*)

herefter bliver der testet på om advokaten er en admin hvis dette er tilfældet sættes knappen "admin page" også til visible.

Det er generelt set hvordan vi har håndteret bruger privilegier i andre forms er currentUser sendt med som parameter hvor der så igen er testet på brugertypen, alt efter hvordan den specifikke form skal vises.

Advokater

Oprettelse af advokat (Bilal)

Advokater bliver oprettet af administratoren i systemet via AdminPage³⁸.

Ved at trykke på knappen “Create” under Lawyers i AdminPage, åbnes AdminCreateLawyer³⁹ formen, så bruger- og personoplysninger kan blive tastet ind i felterne.

Formen henter lister af advokattitler og specialer og deaktiverer diverse knapper:

```

102 1 reference
103 private async void AdminCUDLawyer_Load(object? sender, EventArgs e)
104 {
105     btnCreate.Enabled = false;
106     btnAddSpeciality.Enabled = false;
107     btnRemoveSpeciality.Enabled = false;
108     txtPassword.Text = "0000";
109
110     await FillTitleComboBox();
111     await FillSpecialityComboBox();
112 }
```

30.1 - AdminLawyerCreate / AdminCUDLawyerLoad

Combo-boksene bliver fyldt op med kald af disse metoder:

```

113 1 reference
114 private async Task FillTitleComboBox()
115 {
116     lawyerTitles = await lawyerTitleBL.GetLawyerTitles();
117
118     foreach (LawyerTitleUI title in lawyerTitles)
119     {
120         cboxTitles.Items.Add(title.Title);
121     }
122
123 1 reference
124 private async Task FillSpecialityComboBox()
125 {
126     specialities = await specialityBL.GetSpecialitiesAsync();
127
128     foreach (SpecialityUI speciality in specialities)
129     {
130         cboxSpecialities.Items.Add(speciality.SpecialityName);
131     }
132 }
```

Figur 30.2 - AdminLawyerCreate / FillComboBoxes

Data bliver hentet og konverteret igennem lagene som er vist tidligere, men da der ikke er noget særligt at lægge vægt på, har vi valgt at undlade at vise hele processen her.

³⁸ [Screenshot af AdminPage](#)

³⁹ [Screenshot af AdminCreateLawyer](#)

Vi valgt at have en ensformet e-mail oprettet til alle ansatte, hvor vi udtager de første 2 bogstaver fra fornavn og efternavn, samt de første 4 cifre i telefonnummeret og slutter af med emaildomænet “@lawhouse.com”:

```

135     ↓      3 references
136     ↓      private void SetEmail()
137     ↓      {
138         ↓      if (txtFirstname.ForeColor == validFormat &&
139             ↓      txtLastname.ForeColor == validFormat &&
140                 ↓      txtPhone.ForeColor == validFormat)
141         ↓      {
142             ↓      string email =
143                 ↓      ($"{"txtFirstname.Text.Substring(0, 2)}" +
144                     ↓      $"{"txtLastname.Text.Substring(0, 2)}" +
145                         ↓      $"{"txtPhone.Text.Substring(0, 4)}" +
146                             ↓      $"{emailDomain}").ToLower();
147             ↓      if (pValidator.ValidEmail(email))
148             ↓      {
149                 ↓      txtEmail.Text = email;
150                     ↓      txtEmailLogin.Text = txtEmail.Text;
151             }
152             ↓      else
153             ↓      {
154                 ↓      txtEmail.Text = string.Empty;
155             }
156             ↓      UpdateCreateButtonState();
157
158 }
```

Figur 30.3 - AdminLawyerCreate / SetEmail()

Hvis alle de tre punkter er udfyldt korrekt, får vi dannet en e-mail, som også videreføres til username under logindetaljer, hvor kodeordet som standard er indstillet til at være “0000”. Dette har vi tænkt at være fornuftigt, så den ansatte kan tilgå sin e-mail og sætte sit eget adgangskode.

Substring-funktionen her, starter fra index 0 og returnerer de kommende 2 tegn i string, og de første 4 i telefonnummeret.

Advokattitlen kan ikke skrives ind, men vælges fra kombo-boksen, så der undgås duplikater, stavfejl og inkonsistente data, samt gøre valideringen nemmere. Den samme tanke gælder, når advokaten skal have tildelt specialer. Her kræver det dog lidt mere end bare at vælge et og gå videre, vi har nemlig tænkt at det ville fremme brugeroplevelsen med en mere interaktiv og dynamisk tilgang til funktionen:

```

62     1 reference
63     private void BtnRemoveSpeciality_Click(object? sender, EventArgs e)
64     {
65         if (lboxSpecialities.SelectedItem != null)
66         {
67             cboxSpecialities.Items.Add(lboxSpecialities.SelectedItem);
68             lboxSpecialities.Items.Remove(lboxSpecialities.SelectedItem);
69             if (lboxSpecialities.Items.Count > 0)
70             {
71                 lboxSpecialities.SelectedIndex = lboxSpecialities.SelectedIndex + 1;
72             }
73             else btnRemoveSpeciality.Enabled = false;
74         }
75         UpdateCreateButtonState();
76     }
77     1 reference
78     private void BtnAddSpeciality_Click(object? sender, EventArgs e)
79     {
80         if (cboxSpecialities.SelectedItem != null)
81         {
82             lboxSpecialities.Items.Add(cboxSpecialities.SelectedItem);
83             cboxSpecialities.Items.Remove(cboxSpecialities.SelectedItem);
84             if (cboxSpecialities.Items.Count > 0)
85             {
86                 cboxSpecialities.SelectedIndex = cboxSpecialities.SelectedIndex + 1;
87             }
88             else btnAddSpeciality.Enabled = false;
89         }
90     }

```

Figur 30.4 - AdminLawyerCreate | Add / RemoveSpecialityClick

Admin kan tilføje og fjerne specialiteter ved hjælp af kombo-boksen og listeboksen. Listeboksen indeholder de specialer advokaten bliver tilskrevet, og bliver fjernet fra kombo-boksen, ved tilføjelse og omvendt. *Knapperne bliver genaktiveret, hvis de ikke er nul i to andre event, der ikke lige er med i billedet.*

HireDate kan vil vise dagens dato, når "Today" bliver trykket med en simpel metode:

```

214     1 reference
215     private void BtnToday_Click(object? sender, EventArgs e)
216     {
217         dtpHireDate.Value = DateTime.Now;
218     }

```

Figur 30.5 - AdminLawyerCreate | TodayClick

Vi havde overvejet at begrænse brugeren i at vælge en fremtidig dato eller dato, der ligger for langt tilbage, men gik fra det igen, da vi syntes friheden skulle være der for administratoren.

Hvis valideringen giver grønt lys, vil "Create" knappen være aktiv og processen kan fortsætte med:

```

158     |           1 reference
159     |   private async void BtnCreate_Click(object? sender, EventArgs e)
160     |   {
161         |       DialogResult dialogResult = MessageBox.Show($"Are you sure, that you want to reate this Lawyer? " +
162         |           $"{txtFirstname.Text} {txtLastname.Text}?", "Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
163
164         |       if (dialogResult == DialogResult.Yes)
165         |       {
166             |           btnCreate.Enabled = false;
167
168             |           LawyerUI lawyerUI = new LawyerUI()
169             |           {
170                 |               Fristname = txtFirstname.Text,
171                 |               Lastname = txtLastname.Text,
172                 |               PhoneNumber = int.Parse(txtPhone.Text),
173                 |               Email = txtEmailLogin.Text,
174                 |               AddressLine = txtAddress.Text,
175                 |               PostalCode = int.Parse(txtPostal.Text),
176                 |               City = txtCity.Text,
177                 |               LawyerTitleID = lawyerTitles.FirstOrDefault(x => x.Title == cboxTitles.SelectedItem).LawyerTitleID,
178                 |               HireDate = dtpHireDate.Value
179             |           };
180
181             |           lawyerSpecialityUIs = new List<LawyerSpecialityUI>();
182
183             |           foreach (string item in lboxSpecialities.Items)
184             |           {
185                 |               LawyerSpecialityUI lawyerSpecialityUI = new LawyerSpecialityUI()
186                 |               {
187                     |                   SpecialityID = specialities.FirstOrDefault(s => s.SpecialityName == item).SpecialityID,
188                 |               };
189                 |               this.lawyerSpecialityUIs.Add(lawyerSpecialityUI);
190             |           };
191
192             |           LoginDetailsUI loginDetailsUI = new LoginDetailsUI()
193             |           {
194                 |               Username = txtEmailLogin.Text,
195                 |               Password = txtPassword.Text,
196                 |               CreationDate = DateTime.Now,
197             |           };
198
199
200             |           bool result = await lawyerBL.CreateLawyerAsync(lawyerUI, lawyerSpecialityUIs, loginDetailsUI);

```

Figur 30.6 - AdminLawyerCreate / CreateClick

UI-modeller bliver oprettet, ligesom i Figur 27.2 CreateUserView, dog med tilføjelse bliver titel og specialerne hentet fra tidligere hentet liste ved brug af LINQ, der matcher på ID.

Metoden returnere resultatet og brugeren bliver informeret:

```

202     |           if (result)
203     |           {
204         |               MessageBox.Show($"Lawyer '{lawyerUI}' was successfully created!", "Success",
205         |                   MessageBoxButtons.OK, MessageBoxIcon.Information);
206         |               this.Close();
207     |           }
208     |           else
209         |               MessageBox.Show("Failed to create Lawyer!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
210     }
211     |           btnCreate.Enabled = true;
212

```

Figur 30.7 - AdminLawyerCreate / CreateClick

Ved vellykket oprettelse, får brugeren vist meddeelse der indeholder nylig oprettet advokats fulde navn. Da vi i LawyerUI klasse har brugt override .ToString()-metoden, ved programmet at det er det fulde navn, som skal udskrives.

Min side (Bilal)

Når en advokat logger ind i systemet, bliver "MyPage⁴⁰" vist, som er en slags "dashboard" eller en mere tilpasset side, hvor der uddover er advokatens oplysninger, muligheder for ændringer af disse, har vi også vedlagt 2 datagridviews, der viser de åbne sager og sagssydelser advokaten er tilknyttet. Skulle man have interesse i at åbne og se detaljer om gamle sager og ydelser, kan man trykke på "Show All" check-boksene, og få dem vist.

Formålet med det, er at man nok højst sandsynligt vil bruge denne oversigt til netop at tilpasse sin arbejdssdag efter de opgaver der er i fokus:

```

105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
    2 references
public async Task SetupView(int id)
{
    // Hent og hvis den advokat der skal vises i formen
    displayedLawyerUI = await lawyerBL.GetLawyerWithCollectionsAsync(id);

    chboxServicesShowAll.Checked = false; // Start med at vise åbne sagssydelser

    if (displayedLawyerUI != null)
    {
        DisplayLawyer(displayedLawyerUI);

        SetCasesDgv(chboxCasesShowAll.Checked);
        SetServicesDgv(chboxServicesShowAll.Checked);
        SetupDgvCases();
        SetupDgvCaseServices();
        SetDgvStyle();
    }
}

```

Figur 31.1 - LawyerDetailsView / SetupView()

Advokaten bliver hentet med alle collections, hvor de hver især bliver konverteret i BusinessLogic, så det er nemmere at bruge disse kilder fra displayedLawyerUI variablen, end at lave flere databasekald, for at hente sager og ydelser hver for sig:

```

142
143
144
145
146
147
148
149
150
151
152
    2 references
private void SetCasesDgv(bool serviceShowAllCheck)
{
    if (serviceShowAllCheck)
        dgvCases.DataSource = displayedLawyerUI.Cases; // Vis alle sager
    else
        dgvCases.DataSource = displayedLawyerUI.Cases // Vis kun åbne sager
            .Where(c => c.Status == "Open")
            .ToList();

    SetupDgvCases();
}

153
154
155
156
157
158
159
160
161
162
163
    2 references
private void SetServicesDgv(bool serviceShowAllCheck)
{
    if (serviceShowAllCheck)
        dgvServices.DataSource = displayedLawyerUI.CaseServices; // Vis alle sagssydelser
    else
        dgvServices.DataSource = displayedLawyerUI.CaseServices // Vis kun åbne sagssydelser
            .Where(cs => cs.Status == "Open")
            .ToList();

    SetupDgvCaseServices();
}

```

⁴⁰ [Screenshot af MyPage\(Lawyer\)](#)

Figur 31.2 - LawyerDetailsView / SetDgv() Cases / Services

Her bestemmer check-boks tilstanden indholdet i de nævnte datagridviews.

Da vi bruger LawyerDetailsView for andre advokater, har vi oprettet en user-control der bruges som en slags kort med den pågældende advokat, hvilket gør at vi oprette en standariseret visning og håndtering af klasserne:

```
124 |   | 1 reference
125 |   | private void DisplayLawyer(LawyerUI lawyer)
126 |   | {
127 |   |     // Tilføj hentede advokat til panel
128 |   |     pnlLawyerDetails.Controls.Clear();
129 |   |     // Her bruges en user-control "LawyerCardDisplay" for at standardisere opsætning
130 |   |     pnlLawyerDetails.Controls.Add(new LawyerCardDisplay(displayedLawyerUI));
```

Figur 31.3 - LawyerDetailsView / DisplayLawyer()

Ligeledes, har vi en LawyerCardEdit user-control, når en advokat oplysninger skal opdateres.

Nogle attributter som email, titel og specialet er deaktiveret, så kun administrator kan ændre disse:

```
132 |   | 1 reference
133 |   | private void BtnEditDetails_Click(object? sender, EventArgs e)
134 |   | {
135 |   |   btnEditDetails.Enabled = false;
136 |   | 
137 |   |   pnlEdit.Controls.Clear();
138 |   |   // En user-control til redigering af advokatoplysninger
139 |   |   pnlEdit.Controls.Add(new LawyerCardEdit(this, displayedLawyerUI, isAdmin));
140 |   | }
```

Figur 31.4 - LawyerDetailsView / EditDetailsClick()

Her tager control'en formen, advokaten og boolean isAdmin med som input parameter, da der skal tages højde for, om redigeringsmuligheden skal være aktiveret.

Vil man åbne sagsdetaljer, kan man dobbeltklikke på sag eller service i tabellerne:

```
65 |   | 1 reference
66 |   | private void DgvServices_CellDoubleClick(object? sender, DataGridViewCellEventArgs e)
67 |   | {
68 |   |   // Double-klik på sagsydelse åbner sagsdetaljer med medbringer nødvendige parametre
69 |   |   if (e.RowIndex >= 0) // Ignorer header rækken
70 |   |   {
71 |   |     if (dgvServices.Rows[e.RowIndex].DataBoundItem is CaseServiceUI caseServiceUI)
72 |   |     {
73 |   |       CaseDetailsView detailsView = new CaseDetailsView(caseServiceUI.CaseID, false, true);
74 |   |       detailsView.ShowDialog();
75 |   |     }
76 |   | }
```

Figur 31.5 - LawyerDetailsView / DgvServices_CellDoubleClick

CaseDetailsView vil blive åbnet, vilkårligt om det er under cases eller services man dobbeltklikker. Denne form tager sagsnummeret, om det er klient og om den bliver åbnet fra MyPage variabler som input parameter:

```
CaseDetailsView CaseDetailsView(int selectedCaseID, bool isClient, bool isMyPage)
```

Sagsnummeret bruges naturligvis til at hente og vise sags detaljerne, hvorimod isClient bruges til at deaktivere nogle funktioner i formen. "isMyPage" sørger for at tilpasse vinduet, så den enten åbner som popup eller inde i frontpage panelet, uden de øverste knapper til lukning.

Designet af LawyerDetailsView gør det fleksibelt at bruge i MyPage og andre steder, der skal vises informationer om en advokat. Formålet med MyPage har primært været at hjælpe advokaten med at tilpasse sin arbejdsgang, ved at give en god oversigt over opgaverne, adgang til håndtering af personoplysninger, samt adgang til gamle sager og ydelser udført af vedkommende. Denne tilgang munder ud i effektiv planlægning, øget produktivitet, bedre tidsstyring og samlet set et bedre arbejdsflow.

Visning af ansatte (Bilal)

Vi har lavet en samling af oversigter EmployeesOverview⁴¹, hvor brugeren har mulighed for at se lister over alle ansatte, advokater eller advokatsekretærer. I stedet for at oprette forskellige navigations muligheder i menuen, er oversigterne til rådighed i en kombo-boks "Show".

⁴¹ [Screenshot af EmployeesOverview](#)

```

443     |   1 reference
444     |   private void CboxShowEmployees_SelectedIndexChanged(object? sender, EventArgs e)
445     |   {
446     |       // Assign selectedShow med det valgte "Show" item værdi, så den kan bruges andre steder
447     |       // istedet for at indlæse cboxShow.SelectedItem.ToString() data hver gang
448     |       if (cboxShow.SelectedItem != null)
449     |           selectedShow = cboxShow.SelectedItem.ToString();
450
451     |       txtSearch.Text = string.Empty; // Slet søgningsfelt
452     |       lblShow.Focus();
453     |       lblTotalEmployees.Text = "Loading..."; // Hvis loading tekst mens data indlæses
454
455     |       ResetFilters();
456     |       ResetSorts();
457
458     |       // Tilpas DGV og andre funktioner alt efter hvilken oversigt der er valgt
459     |       switch (selectedShow)
460     |       {
461     |           case showEmployees:
462     |               FillComboBoxesForEmployees();
463     |               SetupDgvWithEmployees();
464     |               break;
465     |           case showLawyers:
466     |               FillComboBoxesForLawyers();
467     |               SetupDgvWithLawyers();
468     |               break;
469     |           case showSecretaries:
470     |               FillComboBoxesForSecretaries();
471     |               SetupDgvWithSecretaries();
472     |               break;
473     |       }

```

Figur 32.1 - EmployeesOverview | CboxShowEmployees_SelectedIndexChanged()

Denne event metode sørger for at nulstille og tilpasse oversigten til det ønskede valg. De tre variabler efter case (showEmployees osv.) er const string datatyper, så der ikke sker ændringer i dem, og da de bliver brugt flere steder, synes vi at det var smart at gøre det på den måde, end at skrive string værdien hver gang.

Den overordnede sorteringsmetoden ser sådan ud:

```

202     |   6 references
203     |   private void SortDgv()
204     |   {
205     |       // Korrekte sorterings metode bruges alt efter hvilken oversigt der er valgt
206     |       if (selectedShow == showEmployees)
207     |       {
208     |           SortEmployees();
209
210     |           dgvEmployees.CurrentCell = null;
211     |           SetupDgvWithEmployees();
212     |       }
213     |       else if (selectedShow == showSecretaries)
214     |       {
215     |           SortSecretaries();
216
217     |           dgvEmployees.CurrentCell = null;
218     |           SetupDgvWithSecretaries();
219     |       }
220     |       else if (selectedShow == showLawyers)
221     |       {
222     |           SortLawyers();
223
224     |           dgvEmployees.CurrentCell = null;
225     |           btnTrashFilter.Visible = flpnlFilters.Controls.Count > 0;
226     |           SetupDgvWithLawyers();
227     |       }

```

Figur 32.2 - EmployeesOverview | SortDgv()

Employees

Som default, bliver "Employees"-tabellen valgt og et datagridview med alle ansatte i virksomheden bliver vist. Alle relevante kontaktoplysninger og hyringsdato vises i tabellen, så man nemt kan kontakte en kollega, hvis det skulle være nødvendigt. Ved at dobbeltklikke på en medarbejder, fremvises en mindre version af EmployeeCardDisplay user-control kortet, som også bruges i MyPage, hvis den ansatte ikke er advokat.

De søgningsmuligheder der er for denne visning er stort set på alle attributter, hvorimod sortering kun sker på Navn og HireDate.

Advokatsekretærer

Denne visning er stort set ens med Employees, uover at have advokattitel kolonnen tilføjet til tabellen.

Advokater

Advokat-oversigten "Lawyers"⁴² er mere omfattende, da vi synes at det kan bruges til at fordele workload, når den rette advokat skal findes for en specifik opgave eller sag. Uover kompetence forskellen, syntes vi at det var relevant, at vise hvor meget arbejde de har at se til. Her har vi nemlig skjult nogle ubetydelige persondata og tilføjet kolonner som åbne/lukkede sager og ydelser, og antal specialer advokaten har, så det skaber et bedre overblik over deres workload, når der skal fordeles arbejde.

Tanken var nemlig, at man filtrerer efter specialer og finder den/de advokater der kan foretage opgaven og evt. vælge den med mindst mængde af tildelte opgaver, så man undgår at overbelaste de samme personer. Dobbeltklikker man på en advokat, ser man det samme view som i MyPage(Lawyer) bilaget, dog uden redigeringsmuligheder, hvis man ikke er administrator. Man kan altså se en mere detaljeret informationsoversigt af personen, samt deres sager og ydelser.

⁴² [Screenshot af LawyersOverview](#)

Når sorteringsmetoden i Figur 32.2 bliver kaldt, sørger den for at vælge den rigtige sorteringsmetode, alt efter hvilken oversigt der er valgt, så den går videre med Figur 32.3, når oversigten er advokater:

```

347     1 reference
348     private void SortLawyers()
349     {
350         filteredLawyers = new List<LawyerUI>(lawyers);
351
352         string search = txtSearch.Text.Trim().ToLower();
353         if (!string.IsNullOrEmpty(search))
354         {
355             filteredLawyers = filteredLawyers
356                 .Where(lawyer =>
357                     lawyer.PhoneNumber.ToString().Contains(search) ||
358                     lawyer.LawyerTitle.ToLower().Contains(search) ||
359                     lawyer.PostalCode.ToString().Contains(search) ||
360                     lawyer.Firstname.ToLower().Contains(search) ||
361                     lawyer.Lastname.ToLower().Contains(search) ||
362                     lawyer.Email.ToLower().Contains(search))
363                 .ToList();
364         }
365     }

```

Figur 32.3 - EmployeesOverview / SortLawyers()

Denne metode opretter en ny liste af advokater som matcher søgningskriterierne, den er dog længere hvis der også skal filtreres for specialer:

```

365     1 reference
366     if (selectedFilters.Count > 0)
367     {
368         filteredLawyers = filteredLawyers
369             .Where(l => selectedFilters
370                 .All(filter => l.LawyerSpecialities
371                     .Any(ls => ls.SpecialityName == filter))).ToList();
372     }

```

Figur 32.4 - EmployeesOverview / SortLawyers()

Filtrerings kombo-boksen er kun aktiv når advokat oversigten vises, da der kan filtreres på advokatspecialer. Her bruges den samme liste af advokater som søgningsfunktionen tidligere muligvis havde ændret på, så den ikke bliver overskrevet. Den nye liste kommer så til at bestå af advokater, som har alle de specialer som er blevet valgt, ved at gå hver advokat og deres specialer igennem. Ønsker man så at sortere efter navn, åbne sager, åbne ydelser, eller specialer eksekveret denne kode:

```
373     switch (cboxSort.SelectedItem)
374     {
375         case "Name":
376             sortByNameCount++;
377             if (sortByNameCount % 2 == 1)
378             {
379                 filteredLawyers = filteredLawyers
380                     .OrderBy(lawyer => lawyer.Firstname)
381                     .ThenBy(lawyer => lawyer.Lastname)
382                     .ToList();
383             }
384         else
385         {
386             filteredLawyers = filteredLawyers
387                     .OrderByDescending(lawyer => lawyer.Firstname)
388                     .ThenByDescending(lawyer => lawyer.Lastname)
389                     .ToList();
390         }
391     }
392 }
```

Figur 32.5 - EmployeesOverview | SortLawyers()

Her bruges en switch for at skabe lidt bedre overblik end et if-statement, hvilket er længere med de andre valgmuligheder, dog begrænset siden operationerne er ens.

Skal der sorteres på navn, tjekker linje 377 på om resultatet er 0 eller 1, for at differentiere om der skal vises i ascending eller descending rækkefølge (her kunne en simpel boolean også klare opgaven).

Datagridview'et bliver tilpasset alt efter hvilken oversigt, søgnings-, filtrerings- og sorteringsmuligheder der er valgt. Dette er med til at fremvise en kompakt og brugbar del af programmet, som medarbejderne kan anvende til at træffe beslutninger om arbejdsfordeling eller finde den rette advokat til den rette sag/opgave. Det skaber værdi for virksomheden, da ressourcer bliver brugt effektivt, og arbejdsprocesserne bliver håndteret af kvalificerede fagfolk til tiden, hvilket også kommer klienterne til gode.

Håndtering af sager (Rasmus)

Oprettelse af sag (Rasmus)

Dette afsnit har til formål at beskrive processen og vores implementering af oprettelsen af en ny sag igennem Windows Forms.

Oprettelsesprocessen af en ny sag, foregår i formen 'CreateCasePage'⁴³ hvor brugeren skal udfylde en række input-felter, som håndterer validering og kommunikerer med vores BusinessLogic-lag for at oprette nye sager i systemet.

Validering af bruger inputtet håndteres gennem en række forskellige event-metoder, der opdaterer farven på teksten i hvert input-felt baseret på gyldigheden. F.eks ændres farven på txtDescription til invalidFormat, hvis titlen ikke overholder valideringen. For yderligere information om vores valideringsproces, læs afsnittet om validering.⁴⁴

```

68 | 1 reference
69 | private void TxtDescription_TextChanged(object? sender, EventArgs e)
70 | {
71 |     txtDescription.ForeColor = cValidator.ValidDescription(txtDescription.Text) ? validFormat : invalidFormat;
72 |     btnCreateEnabled();
73 |

```

Figur 33,1 - Validering af bruger input

Knappen btnCreateCase er kun enabled, i det tilfælde at alle felter opfylder vores krav til valideringen. Dette kontrolleres af en metode 'BtnCreateEnabled'

```

98 | 1 references
99 | private bool BtnCreateEnabled()
100| {
101|     //Håndterer adfærdens på CreateCase knappen, så den kun er enabled, hvis alle felter overholder valideringen
102|     return btnCreateCase.Enabled =
103|         txtTitle.ForeColor == validFormat &&
104|         cbxCaseType.SelectedItem != null &&
105|         txtEstimatedHours.ForeColor == validFormat &&
106|         isEstimatedEndDateValid &&
107|         txtDescription.ForeColor == validFormat &&
108|         selectedClient != null &&
109|         selectedLawyer != null;

```

Figur 33,2 - BtnCreateEnabled()

Metoden har til formål at sikre alle input felter overholder vores validering, samt at der er valgt en lawyer og client. Det vil sige at knappen først er aktiveret, såfremt alle input felter overholder vores validering.

Brugeren skal yderligere have tilvalgt en advokat og klient til sagen, før at knappen bliver aktiveret, dette sker ved at klikke på knappen 'Add Lawyer', hvorefter en ny form åbnes. I denne nye form har vi oprettet en EventHandler som bruges til at sende et LawyerUI-objekt tilbage til vores 'CreateCasePage'

```
28 | public event EventHandler<LawyerUI> LawyerSelected;
```

⁴³ [Screenshot af CreateCasePage](#)

⁴⁴ [Validering](#)

I den nye AddLawyerView⁴⁵ vises et datagridview over alle de forskellige lawyers, for at tilkoble en advokat til en sag, skal brugeren dobbeltklikke på en celle.

```

62  private void DgvLawyerView_CellDoubleClick(object? sender, DataGridViewCellEventArgs e)
63  {
64      //kontroller om cellen er gyldig
65      if(e.RowIndex >= 0 && e.ColumnIndex >= 0)
66      {
67          DataGridViewRow dataGridViewRow = dgvLawyerView.Rows[e.RowIndex];
68
69          if (dataGridViewRow.Cells["PersonID"].Value != null)
70          {
71
72              if (int.TryParse(dataGridViewRow.Cells["PersonID"].Value.ToString(), out int id))
73              {
74                  //henter den selected lawyer fra listen som matcher på id, ud fra den lawyer der er blevet dobbeltklikket på
75                  LawyerUI selectedLawyer = originalLawyerList.FirstOrDefault(l => l.PersonID == id);
76
77                  //Udfører hændelsen LawyerSelected
78                  LawyerSelected?.Invoke(this, selectedLawyer);
79                  this.Close();
80
81              }
82          }
83      }
}

```

Figur 33,3 - Metode til at hente lawyer

Derefter udløses dette event, hvor den først tjekker om cellen er gyldig, derefter kontrollerer den at PersonID indeholder en værdi og forsøger at konvertere den til en int. Derefter henter vi den valgte advokat fra 'originalLawyerList' ved at søge efter en advokat hvor den matcher på det id vi har hentet fra datagridviewet.

Hvis der findes en advokat, udløses 'LawyerSelected', som bruger Invoke til at kalde hændelsen i 'CreateCasePage', som sender den valgte advokat, som parameter tilbage.

```

174  private void AddLawyerView_LawyerSelected(object? sender, LawyerUI e)
175  {
176      //metoden er en event handler som bliver kaldt, når hændelsen LawyerSelect
177
178      txtLawyerFirstName.Text = e.Firstname;
179      txtLawyerLastName.Text = e.Lastname;
180      txtLawyerEmail.Text = e.Email;
181      txtLawyerPhone.Text = e.PhoneNumber.ToString();
182
183      //gemmer den valgte advokat i en variabel selectedLawyer for senere brug
184      selectedLawyer = e;
185
186      BtnCreateEnabled();
187  }

```

Figur 33,4 - event for LawyerSelected

Når eventet er udløst, håndteres den i 'CreateCasePage', som opdaterer vores brugergrænseflade med information om den valgte advokat.

Implementeringen af processen for at vælge en klient, gør brug af samme fremgangsmåde som ved advokat.

⁴⁵ [Screenshot af AddLawyerView](#)

Når brugeren har indtastet valide data, klikkes der på knappen for at oprette en sag, som starter metoden 'BtnCreateCase.Click'. Metoden er asynkron og derved sikrer den at vores brugergrænseflade forbliver responsiv under hele processen.

```
111 private async void BtnCreateCase_Click(object? sender, EventArgs e)
112 {
113     btnCreateCase.Enabled = false;
114
115     CaseTypeUI selectedCaseType = (CaseTypeUI)cboxCaseType.SelectedItem;
116
117     CaseUI caseUI = new CaseUI()
118     {
119         Title = txtTitle.Text,
120         Description = txtDescription.Text,
121         CreationDate = DateTime.Now,
122         EndDate = dtpEstimatedEndDate.Value,
123         EstimatedHours = float.Parse(txtEstimatedHours.Text),
124         Status = "Open",
125         TotalPrice = 0,
126
127         CaseTypeID = selectedCaseType.CaseTypeID,
128         LawyerID = selectedLawyer.PersonID,
129         ClientID = selectedClient.PersonID,
130     };
131
132     DialogResult dialogResult = MessageBox.Show("Do you wish to create this new case?", "Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
133
134     if (dialogResult == DialogResult.Yes)
135     {
136         bool succes = await caseBL.CreateCaseAsync(caseUI);
137         if (succes)
138         {
139             MessageBox.Show("Case created");
140             await casePageView.SetDgvAsync();
141             f1.PnlContextChange(casePageView);
142         }
143         else
144         {
145             MessageBox.Show("Error! Couldnt create case");
146         }
147     }
148
149     btnCreateCase.Enabled = true;
150 }
```

Figur 33,5 - *BtnCreateCase_Click*, oprettelse af sag

Metoden opretter et nyt objekt af typen `CaseUI`, med de data som brugeren har indtastet og viser derefter en message box, hvor brugeren skal bekræfte at de vil oprette denne sag. Dette er gjort for at forbedre brugerens oplevelse. Hvis brugeren trykker ja, kaldes den asynkronne metode `'CreateCaseAsync'` fra `'CaseBL'`.

```
1 reference
27     public async Task<bool> CreateCaseAsync(CaseUI caseUI)
28     {
29         if(!cValidator.ValidateCase(caseUI))
30             return false;
31
32
33         Case temp = modelConverter.ConvertFromCaseUI(caseUI);
34         return await dbAccess.CreateCaseAsync(temp);
35     }
}
```

Figur 33,6 - CreateCaseAsync - Business Logic

I BusinessLogic-laget udfører metoden flere vigtige opgaver. Først valideres der på det CaseUI objekt som, er blevet sendt med fra vores brugergrænseflade, dette minimerer risikoen for korrupte data. Hvis valideringen mislykkes returneres false, og metoden brydes. Derimod hvis valideringen bliver overholdt, konverterer vores CaseUI-objekt til et

Case-entitets objekt, som bruges i vores DataAccess-lag. Til sidst kaldes den tilsvarende metode i datalaget asynkront.

```
21  public async Task<bool> CreateCaseAsync(Case caseE)
22  {
23      try
24      {
25          await db.Cases.AddAsync(caseE);
26          bool succes = await db.SaveChangesAsync() > 0;
27          db.ChangeTracker.Clear();
28          return succes;
29      }
30      catch (Exception)
31      {
32
33          return false;
34      }
35  }
```

Figur 33,7 - CreateCaseAsync - DataAccess

I DataAccess-laget kalder vi 'AddAsync' fra vores dbContext, og dermed tilføjer vores case-objekt til vores DatabaseContext. For at gemme vores Case i databasen, kaldes 'SaveChangesAsync' som er en asynkron metode, der gemmer ændringer, foretaget i DatabaseContexten.

Til sidst sendes resultatet af operationen tilbage til vores brugergrænseflade, hvor den har til formål at informere brugeren om processen er vellykket eller ej.

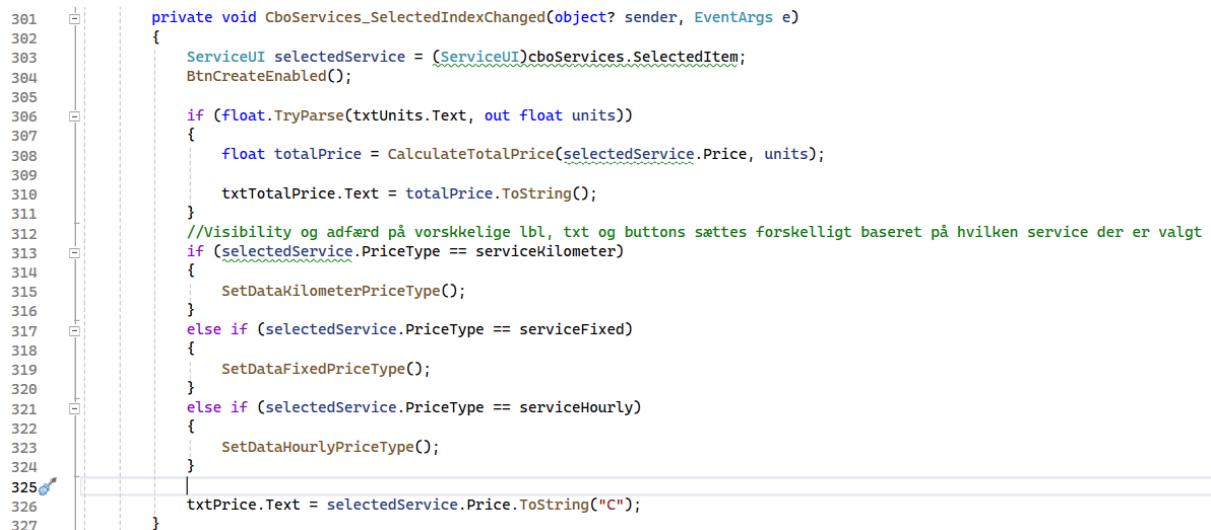
Denne implementering af vores oprettelse af en ny sag, sikrer at der gøres brug af asynkron programmering, validering af input og samtidig at vi har en dynamisk brugergrænseflade. Det forbedrer brugers oplevelse, idet at vores brugergrænseflade forbliver responsiv og tilpasser sig brugers input ift. validering. Samtidig forhindrer det fejl og korrupte data.

Tilføj ydelse til sag (Rasmus)

Dette afsnit beskriver vores implementering af funktionen, der gør det muligt for en advokat og sekretær at tilføje ydelser til en sag. Det er designet til at kunne håndtere forskellige pristyper, samt forskellige typer af ydelser.

Igennem vores oversigt af cases, kan en bruger dobbeltklikke på en given sag, hvorefter vores 'CaseDetailsView'⁴⁶ åbnes. Denne brugergrænseflade har til formål at skabe et overblik over casens forskellige informationer, såsom den advokat der er ansvarlig, samt klienten. Derudover kan man se en oversigt over de tilkoblede ydelser og tilføje flere.

For at navigere videre til processen for at tilføje en ydelse, klikker brugeren på 'Add Service', hvorefter der åbnes en ny form. Her vælger brugeren en ydelse fra ComboBoxen, hvorefter brugergrænsefladen tilpasses dynamisk baseret på ydelsen pristype (timepris, fast pris, kilometer). Dette sikrer at brugeren kun ser de relevante input-felter i forbindelse med en ydelse. Der skal samtidig tilføjes en advokat, som er ansvarlig for denne ydelse, det er præcis samme fremgangsmåde som i oprettelse af en sag.⁴⁷



```

301  private void CboServices_SelectedIndexChanged(object? sender, EventArgs e)
302  {
303      ServiceUI selectedService = (ServiceUI)cboServices.SelectedItem;
304      BtnCreateEnabled();
305
306      if (float.TryParse(txtUnits.Text, out float units))
307      {
308          float totalPrice = CalculateTotalPrice(selectedService.Price, units);
309
310          txtTotalPrice.Text = totalPrice.ToString();
311
312          //Visibility og adfærd på forskellige lbl, txt og buttons sættes forskelligt baseret på hvilken service der er valgt
313          if (selectedService.PriceType == serviceKilometer)
314          {
315              SetDataKilometerPriceType();
316          }
317          else if (selectedService.PriceType == serviceFixed)
318          {
319              SetDataFixedPriceType();
320          }
321          else if (selectedService.PriceType == serviceHourly)
322          {
323              SetDataHourlyPriceType();
324          }
325      }
326
327      txtPrice.Text = selectedService.Price.ToString("C");
328  }

```

Figur 34,1 - Ændring af adfærd ud fra combobox-event

Metoden tilpasser vores brugergrænseflade med de input-felter som er nødvendige for den valgte service, hvorefter den kalder en ny metode som ændrer adfærdens af de forskellige

⁴⁶ [Screenshot af CaseDetailsView](#)

⁴⁷ [Oprettelse af sag](#)

tekstfelter og design.

```
329      //adfærd for kilometer
330      1 reference
331      private void SetDataKilometerPriceType()
332      {
333          lblUnits.Visible = true;
334          txtUnits.Visible = true;
335
336          lblPrice.Text = "Price/km";
337          lblPrice.Visible = true;
338          txtPrice.Visible = true;
339
340          lblTotalPrice.Text = "Total price";
341          lblTotalPrice.Visible = true;
342          txtTotalPrice.Visible = true;
343          txtTotalPrice.Enabled = false;
344
345          lblHoursWorked.Visible = true;
346          txtHoursWorked.Visible = true;
347
348          btnCalculation.Visible = true;
349
350      }
351
```

Figur 34,2 - Metode til at ændre adfærdens

Denne metode tilpasser brugergrænsefladen for den valgte service, hvis den har en pristype af kilometer.

Validering af de forskellige input felter, har samme fremgangsmåde, som ved oprettelse af sag. Der gøres ligeledes brug af en BtnCreateEnabled metode til at aktivere knappen, hvis alle tekstfelter overholder vores validering, samt at der er en advokat tilknyttet.

Når en bruger klikker på 'Add Service' oprettes et nyt objekt af typen CaseServiceUI, som får tilegnet data ud fra hvilken pristype den valgte service har, der kan fx. ses i figur(x) at, hvis pristypen er kilometer, så sættes status til closed. Det har vi gjort grundet, at vi har taget forbehold for at der i oprettelse af en driving service skal tilføjes timer, men efter den er tilføjet, er dette ikke muligt.

```

211     private async void BtnAddService_Click(object? sender, EventArgs e)
212     {
213         //Henter den service der er valgt fra comboboxen
214         ServiceUI selectedService = (ServiceUI)cboServices.SelectedItem;
215         //Disabler knappen
216         btnAddService.Enabled = false;
217
218         //Oprettet en ny caseservice
219         CaseServiceUI caseServiceUI = new CaseServiceUI()
220         {
221             Description = txtServiceDescription.Text,
222             StartDate = DateTime.Now, // sortByNameCount
223
224             CaseID = selectedCase.CaseID,
225             ServiceID = selectedService.ServiceID,
226             LawyerID = selectedLawyer.PersonID,
227         };
228
229         //Forskellig oprettelse ift hvilken pricetype servicen har
230         if (selectedService.PriceType == serviceKilometer)
231         {
232             caseServiceUI.Status = caseClosed;
233             caseServiceUI.TotalPrice = float.Parse(txtTotalPrice.Text);
234             caseServiceUI.HoursWorked = float.Parse(txtHoursWorked.Text);
235             caseServiceUI.EndDate = DateTime.Now;
236             caseServiceUI.Units = float.Parse(txtUnits.Text, CultureInfo.InvariantCulture);
237         }
238         else if (selectedService.PriceType == serviceFixed)
239         {
240             caseServiceUI.Status = caseOpen;
241             caseServiceUI.TotalPrice = float.Parse(txtTotalPrice.Text);
242             caseServiceUI.HoursWorked = 0;
243             caseServiceUI.EndDate = null;
244             caseServiceUI.Units = 1;
245         }
246         else if (selectedService.PriceType == serviceHourly)
247         {
248             caseServiceUI.Status = caseOpen;
249             caseServiceUI.TotalPrice = 0;
250             caseServiceUI.HoursWorked = 0;
251             caseServiceUI.EndDate = null;
252             caseServiceUI.Units = 0;
253         }
254
255         DialogResult dialogResult = MessageBox.Show("Do you want to add this service?", "Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
256
257         if (dialogResult == DialogResult.Yes)
258         {
259             bool succes = await caseServiceBL.CreateCaseServiceAsync(caseServiceUI);
260             if (succes)
261             {
262                 MessageBox.Show("Service has been added");
263             }
264             else
265             {
266                 MessageBox.Show("Failed to add the service");
267             }
268         }
269
270         //Opdater den tidligere form
271         await caseDetailsView.SetCaseDataAsync();
272         caseDetailsView.SetDgvAsync();
273         this.Close();
274     }

```

Figur 34.3 - AddService

Dernæst vises en bekræftelse som brugeren skal interagere med for at fortsætte. Hvis brugeren trykker ja, tilføjes den nye CaseServiceUI asynkront til sagen. Dette sker ved at den først kalder vores BusinessLogic-lag.

```

27     public async Task<bool> CreateCaseServiceAsync(CaseServiceUI caseServiceUI)
28     {
29         if (!cValidator.ValidateCaseService(caseServiceUI))
30             return false;
31
32         CaseService temp = modelConverter.ConvertFromCaseServiceUI(caseServiceUI);
33         return await dbAccess.CreateCaseServiceAsync(temp);
34     }

```

Figur 34.4 - BusinessLogic, CreateCaseServiceAsync

I BusinessLogic-laget har metoden flere opgaver. Først valideres der på hele objektet, for at mindske risikoen for korrupte data. I tilfælde af det ikke overholder vores validering returneres false. Hvis valideringen bliver overholdt, konverteres vores CaseServiceUI-objekt

til et entitetsobjekt, som vi bruger i DataAccess-laget. Til sidst kaldes den tilsvarende metode i DataAccess-laget asynkront.

```
21  public async Task<bool> CreateCaseServiceAsync(CaseService caseService)
22  {
23      try
24      {
25          db.CaseServices.AddAsync(caseService);
26          bool succes = db.SaveChanges() > 0;
27          db.ChangeTracker.Clear();
28          return succes;
29      }
30      catch (Exception)
31      {
32          return false;
33      }
34  }
35
36
```

Figur 34,5 - DataAccess, CreateCaseServiceAsync

I DataAccess-laget kalder vi 'AddAsync' fra vores dbContext, og dermed tilføjer vores 'CaseService' til vores DatabaseContext. For at gemme det i databasen, kaldes 'SaveChangesAsync', som gemmer ændringerne i databasen asynkront.

Resultatet af denne operation vil blive sendt tilbage til vores brugergrænseflade, hvor en messagebox vil indikere om operationen er lykkedes eller ej.

Vores implementering af tilføjelse af en ydelse, er designet til at kunne håndtere komplekse interaktioner med brugeren. Den dynamiske håndtering af vores brugergrænseflade, gør at der kun vises de relevante input-felter som skal udfyldes, hvilket letter arbejdet for brugeren og forbedrer brugeroplevelsen og effektiviteten. Dette er også med til at automatisere arbejdsgangene omkring registrering af ydelser, hvilket var efterspurgt fra LawHouse.

Tilføj timer til ydelse (Rasmus)

Dette afsnit vil beskrive tilføjelse af timer til en ydelse, dette er en afgørende funktion i vores system, idet det hjælper med at skabe en præcis oversigt over arbejdstimer på en given sag, samt ydelser.

I vores CaseDetailsView vises alle ydelser som er tilkoblet denne sag i et DataGridView, ved at dobbeltklikke på en ydelse, åbnes vores ‘ServiceDetailsForm’⁴⁸, som tilpasser sig dynamisk efter hvilken service det er. Det er her brugeren kan tilføje timer til en ydelse.

Når vores ServiceDetailsForm initialiseres modtager den flere forskellige parametre, herunder en reference til vores (CaseDetailsView), den valgte CaseService og en boolsk værdi, som angiver om den nuværende bruger er en klient. Dette design er afgørende for, at vi kan tilpasse brugeroplevelsen af systemet, samt sikre at det kun er bestemte brugere som får adgang til visse funktioner. Derfor har vi valgt at skjule visse knapper og input-felter som kun er relevant for advokater og sekretærer, for at opretholde sikkerheden.

```

66      //fjerner knapper hvis bruger er client
67      if (isClient)
68      {
69          btnSubmit.Visible = false;
70          txtHoursWorked.Visible = false;
71          lblHoursWorked.Visible = false;
72          btnClose.Visible = false;
73      }

```

Figur 35.1 - Tilpasning af funktioner

Derudover tager vi også højde for om en ydelse er lukket eller åbnet, hvis en ydelse er lukket, kan denne ikke rettes i og visse funktioner gøres utilgængelige.

Vi har også valgt at tage hensyn til, at der ikke skal kunne registreres timer til kørsels-ydelser. Dette skyldes at disse ydelser ikke kræver den samme registrering af timer, såsom andre typer af ydelser. Dette bidrager til en bedre brugeroplevelse, idet unødvendige funktioner fjernes og der fokuseres på det vigtige.

```

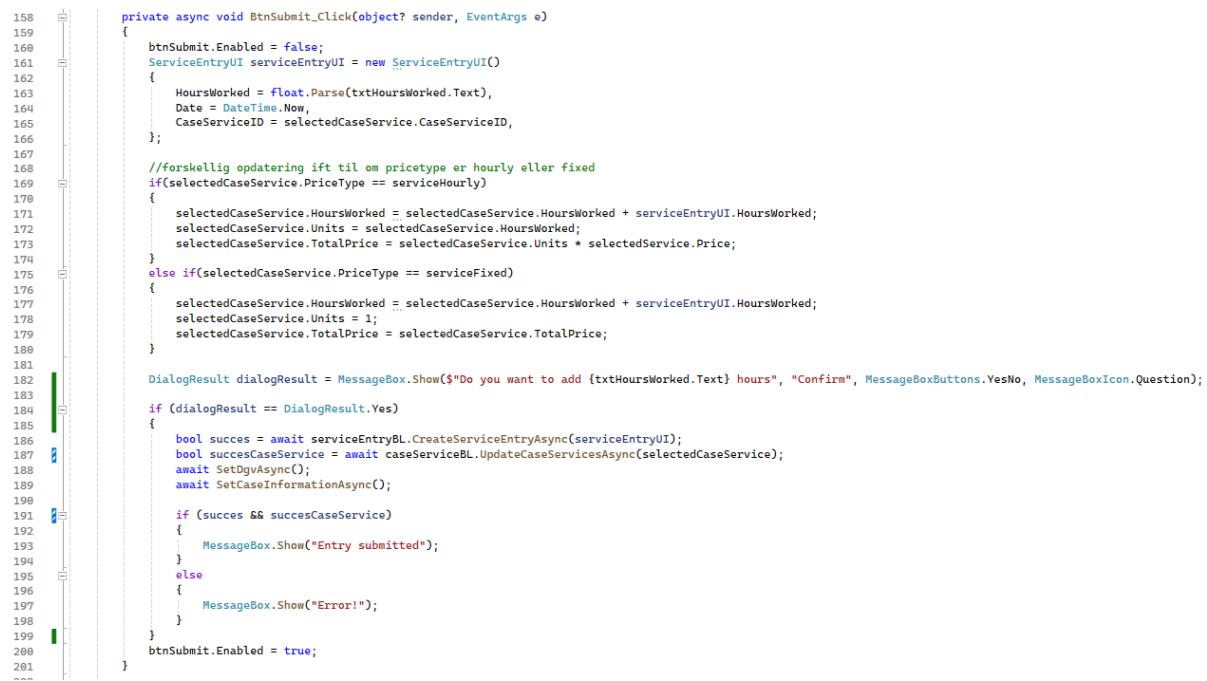
98      private void CheckStatus()
99      {
100         if(selectedCaseService.Status == "Closed")
101         {
102             txtHoursWorked.Enabled = false;
103             btnClose.Enabled = false;
104             btnSubmit.Enabled = false;
105         }
106     }

```

Figur 35.2 - CheckStatus

⁴⁸ [Screenshot af ServiceDetailsForm](#)

Hvis en bruger ønsker at tilføje timer til en ydelse, åbner de den relevante ydelse i 'ServiceDetailsView'. Vores brugergrænseflade præsenterer detaljer om ydelsen og de timer der allerede er registreret, herunder hvilken dag det blev registreret. Brugeren har mulighed for at registrere antallet af timer i forbindelse med den service, de har arbejdet med, dette skaber en mere effektiv håndtering af timer.



```

158     private async void BtnSubmit_Click(object? sender, EventArgs e)
159     {
160         btnSubmit.Enabled = false;
161         ServiceEntryUI serviceEntryUI = new ServiceEntryUI()
162         {
163             HoursWorked = float.Parse(txtHoursWorked.Text),
164             Date = DateTime.Now,
165             CaseServiceID = selectedCaseService.CaseServiceID,
166         };
167
168         //Forskellig opdatering ift til om pricetype er hourly eller fixed
169         if(selectedCaseService.PriceType == serviceHourly)
170         {
171             selectedCaseService.HoursWorked = selectedCaseService.HoursWorked + serviceEntryUI.HoursWorked;
172             selectedCaseService.Units = selectedCaseService.HoursWorked;
173             selectedCaseService.TotalPrice = selectedCaseService.Units * selectedService.Price;
174         }
175         else if(selectedCaseService.PriceType == serviceFixed)
176         {
177             selectedCaseService.HoursWorked = selectedCaseService.HoursWorked + serviceEntryUI.HoursWorked;
178             selectedCaseService.Units = 1;
179             selectedCaseService.TotalPrice = selectedCaseService.TotalPrice;
180         }
181
182         DialogResult dialogResult = MessageBox.Show($"Do you want to add {txtHoursWorked.Text} hours", "Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
183
184         if (dialogResult == DialogResult.Yes)
185         {
186             bool succes = await serviceEntryBL.CreateServiceEntryAsync(serviceEntryUI);
187             bool succesCaseService = await caseServiceBL.UpdateCaseServicesAsync(selectedCaseService);
188             await SetDgvAsync();
189             await SetCaseInformationAsync();
190
191             if (succes && succesCaseService)
192             {
193                 MessageBox.Show("Entry submitted");
194             }
195             else
196             {
197                 MessageBox.Show("Error!");
198             }
199         }
200         btnSubmit.Enabled = true;
201     }

```

Figur 35.3 - BtnSubmit

Når brugeren har indtastet timerne og klikker på knappen 'Submit', oprettes der et nyt objekt af typen 'ServiceEntryUI'. Dette objekt håndteres ud fra hvilken pristype, der er koblet til det. Dernæst kaldes den asynkrone metode CreateServiceEntryAsync, denne metode følger samme praksis som ved tilføjelse af en ydelse⁴⁹, eller oprettelse af en sag⁵⁰.

Den kalder også UpdateCaseServiceAsync fra CaseServiceBL, for at opdatere de samlede timer og den samlede pris. Det er også her vores trigger⁵¹ primært udløses, for at opdatere den samlede pris på en sag.



```

54     public async Task<bool> UpdateCaseServicesAsync(CaseServiceUI caseServiceUI)
55     {
56         if (!cValidator.ValidateCaseService(caseServiceUI))
57             return false;
58
59         return await dbAccess.UpdateCaseServiceAsync(modelConverter.ConvertFromCaseServiceUI(caseServiceUI));
60     }

```

Figur 35.4 - BusinessLogic, UpdateCaseServiceAsync

⁴⁹ [Tilføj ydelse til sag](#)

⁵⁰ [Oprettelse af sag](#)

⁵¹ [Trigger](#)

I BusinessLogic-laget ser metoden sådan ud for UpdateCaseServicesAsync, den følger samme struktur som create, ved første at validate på objektet. Dernæst konverterer den objektet og kalder den tilsvarende metode fra vores DataAccess-lag.

```

73  public async Task<bool> UpdateCaseServiceAsync(CaseService caseService)
74  {
75      try
76      {
77          db.CaseServices.Update(caseService);
78          bool succes = await db.SaveChangesAsync() > 0;
79          db.ChangeTracker.Clear();
80          return succes;
81      }
82      catch (Exception)
83      {
84          return false;
85      }
86  }

```

Figur 35.5 - DataAccess, UpdateCaseServiceAsync

I DataAcces-laget kalder vi Update fra vores dbContext, og dermed tilføjer voresvalgte caseservice til vores DatabaseContext. For at gemme det i databasen, kaldes SaveChangesAsync, som gemmer ændringerne i databasen asynkront. Hvis fejl skulle opstå fanges de i vores try-catch.

```

150 protected async override void OnClosing(CancelEventArgs e)
151 {
152     //Når formen lukkes så opdateres detailview for casen
153     await caseDetailsView.SetCaseDataAsync();
154     caseDetailsView.SetDgvAsync();
155     base.OnClosing(e);
156 }

```

Figur 35.6 - Override OnClosing

Når brugeren er færdig med at registrere timerne, lukkes formen. Denne metode er blevet overskrevet for at tilføje nogle specifikke handlinger, herunder at opdatere casens data og datagridviewet over ydelserne, med de korrekte data. Det sikrer at alt data altid er korrekt.

Funktionen ved at tilføje timer har til formål at forbedre brugervenligheden. Der skabes en mere struktureret og præcis måde at tilføje timer til en ydelse. De automatiserede processer bidrager til at der spares tid og det kan forbedre produktiviteten. Dette er med henblik på at skabe en mere effektiv arbejdssproces for de ansatte i LawHouse.

Afslutning af sag og ydelser (Rasmus)

Dette afsnit vil beskrive hvordan vi har valgt at håndtere afslutningen af en case og ydelserne, dette er en afgørende funktion i systemet, der sikrer at en sag ikke kan lukkes, før alle forholdsregler er opfyldt.

Når brugeren nавигerer til CaseDetailsView, ved at dobbeltklikke på en case i caseoversigten, sættes dataen alt afhængigt af casens status, dette sikrer at der ikke kan rettes i en case som er lukket, derudover sikrer det også at en case ikke kan lukkes igen.

Hvis den specifikke case ikke er lukket, kan brugeren klikke på Close Case og følgende metode vil blive eksekveret.

```

107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148

    private async void BtnCloseCase_Click(object? sender, EventArgs e)
    {
        //Hvis der er 0 caseservices tilføjet, kan sagen ikke closes
        if (caseServiceList.Count == 0)
        {
            MessageBox.Show("Cannot close case without any services connected");
            return;
        }

        //Hvis der er en caseservice med status åben, kan den ikke closes
        foreach (CaseServiceUI caseServiceUI in caseServiceList)
        {
            if (caseServiceUI.Status == "Open")
            {
                MessageBox.Show("Cannot close case with active services");
                return;
            }
        }

        //Disabler knappen
        btnUpdateCaseStatus.Enabled = false;

        //Messagebox
        DialogResult dialogResult = MessageBox.Show("Do you wanna close this case", "Close case", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if(dialogResult == DialogResult.Yes)
        {
            //Opdaterer status og enddate på casen
            selectedCase.Status = "Closed";
            selectedCase.EndDate = DateTime.Now;
            if (await caseBL.UpdateCaseSync(selectedCase))
            {
                MessageBox.Show("Case closed");
                await SetCaseDataAsync();
            }
            else
            {
                MessageBox.Show("Failed to close the case");
                btnUpdateCaseStatus.Enabled = true;
            }
        }
    }
}

```

Figur 36.1 - BtnCloseCase

Metoden starter med at kontrollere, hvorvidt der er tilføjet nogle ydelser til sagen. Hvis der ikke er tilføjet nogle, kan sagen ikke lukkes og der vises en besked med dette til brugeren.

Dernæst itererer metoden gennem listen af 'CaseServices' og kontrollere, om der findes en med en status som er 'Open', hvis dette er tilfældet kan sagen ikke lukkes og der vises igen en besked til brugeren.

Hvis alle forholdsregler er overholdt og der klikkes på Close Case, vises en messagebox, som beder brugeren om at bekræfte handlingen. Hvis brugeren bekræfter valget, vil sagens status opdateres til Closed og dens slutdato til det nuværende tidspunkt. Derefter kaldes metoden UpdateCaseAsync fra vores CaseBL.

```

72  public async Task<bool> UpdateCaseAsync(CaseUI caseUI)
73  {
74      if(!cValidator.ValidateCase(caseUI))
75          return false;
76
77      Case temp = modelConverter.ConvertFromCaseUI(caseUI);
78      return await dbAccess.UpdateCaseAsync(temp);
79  }
80
81 }
```

Figur 36.2 - BusinessLogic, UpdateCaseAsync

BusinessLogic-laget validerer objektet og returnerer false, hvis dette mislykkes. Ved vellykket scenario konverteres UI-modellen til en entitetsmodel som vores DataAccess-lag arbejder med. Dernæst kaldes den tilsvarende metode i DataAccess-laget.

```

106 public async Task<bool> UpdateCaseAsync(Case caseE)
107 {
108     try
109     {
110         db.Cases.Update(caseE);
111         bool succes = await db.SaveChangesAsync() > 0;
112         db.ChangeTracker.Clear();
113         return succes;
114     }
115     catch (Exception)
116     {
117         return false;
118     }
119 }
```

Figur 36.3 - DataAccess, UpdateCaseAsync

I DataAccess-laget kalder vi Update fra vores dbContext, og dermed tilføjer vores valgte case til vores DatabaseContext. For at gemme det i databasen, kaldes SaveChangesAsync, som gemmer ændringerne i databasen asynkront. Hvis fejl skulle opstå fanges de i vores try-catch. Dernæst returnes en boolsk værdi, som sendes tilbage til vores brugergrænseflade og viser om operationen er vellykket eller ej.

For at afslutte en service skal brugeren navigere til ServiceDetailsView hvor, de kan klikke på Close Service, som vil eksekvere denne metode

```

108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137

    private async void BtnClose_Click(object? sender, EventArgs e)
    {
        if(serviceEntryUIs.Count == 0)
        {
            MessageBox.Show("Must add entries to close this service");
            return;
        }

        btnClose.Enabled = false;
        DialogResult dialogResult = MessageBox.Show("Do you wanna close this service", "Close service", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (dialogResult == DialogResult.Yes)
        {
            selectedCaseService.Status = "Closed";
            selectedCaseService.EndDate = DateTime.Now;
            if (await caseServiceBL.UpdateCaseServicesAsync(selectedCaseService))
            {
                MessageBox.Show("Service closed");
                txtHoursWorked.Enabled = false;
                this.Close();
            }
            else
            {
                MessageBox.Show("Failed to close the service");
                btnClose.Enabled = true;
            }
        }
        else
            btnClose.Enabled = true;
    }
}

```

Figur 36.4 - BtnClose

Her kontrolleres der igen for om der er nogle 'ServiceEntries' (time-registreringer) tilføjet til denne service. Dernæst er fremgangsmåden i denne metode magen til forrige metode for at afslutte en sag.

Implementeringen af disse metoder har til formål at sikre at brugeren ikke kan lukke en sag før at den er færdigbehandlet, ligeledes gælder det ydelserne. Det minimerer risikoen for eventuelle fejl, hvilket er afgørende for sagsbehandlingen.

Klientens oplevelse (Lucas)

Opfølgning på sine sager (Lucas)

Vi har i systemet LawHouse tilføjet en funktion, der gør det muligt for en klient selv at følge i deres sager herunder status, pris osv. For at skabe mere værdi til både Lawhouse, men også at forbedre klientens oplevelse hos LawHouse.

Hvis klienten navigerer over til "My Page" vil de blive mødt af denne her side⁵².

Der bliver vist et DataGridView over cases der tilhører klienten. Til det er der lavet en metode SetCasesDGVAsync() (figur 37,1)

⁵² [Screenshot af My Page](#)

```

public async Task SetCasesDGVAsync()
{
    //henter alle cases hvor klienten er tilknyttet
    cases = await caseBL.GetCasesAsync(client.PersonID);

    //setup cases dgv
    dgvCases.DataSource = cases;
    dgvCases.Columns[2].Visible = false;
    dgvCases.Columns[3].Visible = false;
    dgvCases.Columns[4].Visible = false;
    dgvCases.Columns[8].Visible = false;
    dgvCases.Columns[9].Visible = false;
    dgvCases.Columns[10].Visible = false;
    dgvCases.Columns[11].Visible = false;
    dgvCases.Columns[12].Visible = false;
    dgvCases.Columns[13].Visible = false;
    dgvCases.Columns[14].Visible = false;
    dgvCases.ClearSelection();
}

```

Figur 37,1 - SetCasesDGVAsync()

Metoden fjerner kolonner som ikke skal være synlige fra DGvet og sætter datasource.

Den spændende del er at den starter med at hente alle cases som klientet er tilknyttet. Det gør den med metoden GetCasesAsync() fra CaseBL (business logic lag) som tager et klient id som input parameter (Figur 37,2)

```

public async Task<List<CaseUI>> GetCasesAsync(int clientID)
{
    List<CaseUI> cases = new List<CaseUI>();

    foreach(Case caseE in await dbAccess.GetCasesAsync(clientID))
    {
        cases.Add(modelConverter.ConvertFromCaseEntity(caseE));
    }
    return cases;
}

```

Figur 37,2 - GetCasesAsync() Business Logic

Denne metode opretter en liste af Cases af type UI-Modellen CaseUI. Så kører den en foreach loop igennem alle de cases den henter fra databasen ved hjælp af metoden GetCasesAsync(clientID) fra dbAccess (Figur x,3).

```

public async Task<List<Case>> GetCasesAsync(int ClientID)
{
    try
    {
        return await db.Cases.Where(c => c.ClientID == ClientID).ToListAsync();
    }
    catch (Exception)
    {

        return new List<Case>();
    }
}

```

Figur 37,3 - GetCasesAsync() DataAccess

Her laver den et database hvor den henter alle cases, ved hjælp af LINQ filtrerer vi så cases så den kun henter de cases hvor clientID på case matcher med clientID som bliver sendt med som parameter. Så returnerer den listen af alle de sager.

Hvis vi så går tilbage til BusinessLogic (figur 37,2) kører den en foreach loop igennem alle de cases den har fået ind og konverterer dem en af gangen fra Entitets model til UI model og tilføjer til vores liste af CaseUI. Denne liste bliver så retuneret til UI. Hvor metoden SetCasesDGVAsync() sætter den liste som datasource til datagridviewet.

En klient kan så dobbelt klik på en af af sagerne i DataGridViewet for at få yderligere detaljer på sagen. Her bliver metoden DgvCases_CellDoubleClick() kaldt (figur 37,4)

```

private void DgvCases_CellDoubleClick(object? sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        //declare
        CaseDetailsView caseDetailsView;

        //gemmer valgte case
        CaseUI selectedCase = cases[e.RowIndex];

        //tester på om den nuvrende bruger er client eller ikke og instanciere caseDetailsView
        if(currentUser is ClientUI)
            caseDetailsView = new CaseDetailsView(selectedCase.CaseID, true, true);
        else
            caseDetailsView = new CaseDetailsView(selectedCase.CaseID, false, true);

        //åben details på casen
        caseDetailsView.ShowDialog();
    }
}

```

Figur 37,4 - DgvCases_CellDoubleClick()

Her tester den på at den valgte celle har index 0 eller dereover. Den gemmer den valgte case fra DVet som "selectedCase" og så tester vi på om den nuværende bruger som er logget

ind er en Client. Dette er fordi at My page for klienter også fungere som “client details” for en ansat. Client detailsview tager imod en parameter IsClient som bool så hvis det er en klient sendes true med over ellers false.

og så åbnes instansen af caseDetailsView.⁵³

Grunden til vi sender IsClient med over ser vi på CaseDetailsView (Figur x,5)

```
//fjerner knapper hvis brugeren er client
if (isClient)
{
    btnAddService.Visible = false;
    btnUpdateCase.Visible = false;
    btnUpdateCaseStatus.Visible = false;
    txtTitle.Enabled = false;
    txtDescription.Enabled = false;
    dtpEstimatedEndDate.Enabled = false;
    txtEstimatedHours.Enabled = false;
}
```

Figur 37,5 - CaseDetailsView(), Constructor

I constructoren tester den nemlig på om IsClient er true hvis dette er tilfældet, fjerner den nogle knapper fra CaseDetailsView() som kun en Ansat har adgang til.

Nu kan klienten se flere detaljer på sin sag og hvad der er blevet arbejdet på. Nærmere detaljer om dette er beskrevet her⁵⁴. Her kan klienten også vælge at lave en udskrift til en txt fil der giver en oversigt over hele sagen, beskrevet mere i dybden her⁵⁵.

Oprettelse af Abonnement (Lucas)

I LawHouse systemet skal en klient kunne oprette abonnement hvorefter de så vil få adgang til et bibliotek af beregninger.

Når en klient navigerer til Subscription siden vil de blive vist med 3 forskellige valgmuligheder at abonnerer i 3, 6 eller 12 måneder⁵⁶, vi tager udgangs i at klienten klikker på 3 måneder så vil metoden BtnBuySubscriptionThreeMonth_Click blive kaldt (Figur x,1)

⁵³ [Screenshot af CaseDetailsView fra klientens side](#)

⁵⁴ [CRUD Sager](#)

⁵⁵ [Rapport til tekstfil](#)

⁵⁶ [Screenshot - Subscription view](#)

```

private async void BtnBuySubscriptionThreeMonth_Click(object? sender, EventArgs e)
{
    btnBuySubscriptionThreeMonth.Enabled = false;

    //tjekker om client allerede er sub
    if (client.IsSubscribed)
    {
        MessageBox.Show("You are already subscribed!");
        btnBuySubscriptionThreeMonth.Enabled = true;
        return;
    }

    DialogResult dialogResult = MessageBox.Show($"Are you sure, that you want to buy subscription for 3 months?", "Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (dialogResult == DialogResult.Yes)
    {
        //ellers create sub
        bool subscriptionIsCreated = await CreateSubscriptionOne(3);

        if (subscriptionIsCreated)
        {
            await frontPageView.GetPersonAsync(client.LoginDetailsID);
            MessageBox.Show($"Successfully subscribed for 3 months!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
            this.Close();
        }
        else
            MessageBox.Show("Failed!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    btnBuySubscriptionThreeMonth.Enabled = true;
}

```

Figur 38,1 - *BtnBuySubscriptionThreeMonth_Click()*

Hvis ikke man er abonneret vil en messagebox vise der spørger om man er sikker på man har lyst til at abonnere. Når klienten trykker ja vil metoden `CreateSubscriptionOne(int subLength)` (Figur 38,2) blive kaldt som tager et inputparameter, antal måneder man vil abonnere. Metoden hedder "One" til sidst da dette er for oprettelse af abonnement type 1. Systemet har nu kun en type af abonnement, men der tages forbehold for at der i fremtiden kunne være flere typer af abonnementer.

```

public async Task<bool> CreateSubscriptionOne(int subLength)
{
    //henter den valgte abonnement type //kun en findes nu
    SubscriptionUI subscriptionOne = await subscriptionBL.GetSubscriptionAsync(1);

    //berenger den betalte pris
    float PaidPrice = CalcPaidPrice(subscriptionOne, subLength);

    //opretter nyt køb af sub i UI
    ClientSubscriptionUI subscription = new ClientSubscriptionUI()
    {
        StartDate = DateTime.Now,
        EndDate = DateTime.Now.AddMonths(subLength),
        PaidPrice = PaidPrice,
        ClientID = client.PersonID,
        SubscriptionID = 1,
    };

    //opretter køb i db
    return await subscriptionBL.CreateSubscriptionAsync(subscription);
}

```

Figur 38,2 - *CreateSubscriptionOne()*

Der er lavet en metode `CalcPaidPrice` med input parameter for type af abonnement og længde. Denne beregner den pris der er betalt for abonnementet udfra prisen på

abonnementet og længden, Der kommer rabat pr måned hvis man vælger 6 eller 12 måneder. (Figur 38,3)

```
1 reference
private float CalcPaidPrice(SubscriptionUI subscription, int subLength)
{
    switch(subLength)
    {
        case 3:
            return subscription.Price * 3;
        case 6:
            return (subscription.Price - 5) * 6;
        case 7:
            return (subscription.Price - 10) * 12;
        default: return 0;
    }
}
```

Figur 38,3 - CalcPaidPrice()

Hvis vi kigger tilbage på CreateSubscriptionOne metoden (Figur X) Vil den nu oprette en instans af ClientSubscription. Instansen får tildelt StartDate som dags dato, End date får tildelt dags dato plus antal måneder klienten har valgt at abonnere, PaidPrice bliver tildelt den beregnede paidPrice og den opretter abonnementtypen med ID - 1.

Den kalder nu metoden CreateSubscriptionAsync(ClientSubscription) fra subscriptionBL klassen i businesslogic laget. (Figur 38,4)

```
public async Task<bool> CreateSubscriptionAsync(ClientSubscriptionUI subscriptionUI)
{
    ClientSubscription clientSubscription = modelConverter.ConvertFromClientSubscriptionUI(subscriptionUI);
    return await dbAccess.CreateSubscriptionAsync(clientSubscription);
}
```

Figur 38,4 - CreateSubscritonAsync Busines Logic

Denne metode vil først konvertere UI-Modellen af ClientSubscription om til en Entity-Model og derefter kalder CreateSubscritptionAsync fra DataAccess laget. (figur 38,5)

```
public async Task<bool> CreateSubscriptionAsync(ClientSubscription subscription)
{
    try
    {
        await db.AddAsync(subscription);
        return await db.SaveChangesAsync() > 0;
    }
    catch (Exception)
    {
        return false;
    }
}
```

Figur 38,5 - CreateSubscriptionAsync, Data Access Lag

Her kalder vi metoden AddAsync fra vores DbContext (db) og tilføjer så ClientSubscription til vores dbcontext. Så kaldes metoden SaveChangesAsync som gemmer det i databasen og retunerer entries skrevet i databasen hvis dette er over nul vil hele metoden CreateSubscriptionAsync returnere true tilbage til BL Laget, og videre til UI Laget.

Hvis man kigger tilbage på (Figur 38,1) så når oprettelse er lykkedes, kaldes metoden GetPersonAsync fra frontpageview som henter personen som er logget ind for at opdatere oplysningerne om at klienten nu er abonneret. og derefter viser den en besked "success".

Beregninger (Lucas)

For at skabe mere værdi til LawHouse systemet ved at forbedre klientens oplevelse er der bedt om at tilføje et bibliotek af beregnings algoritmer som klienter som abonnere kan gøre brug af.

For at afkoble algoritmerne fra resten resten af systemet har vi oprettet et ny projekt (class library) hvori der er oprettet klasser til beregnings algoritmer.

Alle klasser i dette library er lavet static, så behøver de ikke blive instansieret og er globalt tilgængelige.

Her vises et eksempel på en af vores algoritmer for beregning af ydelse på lån (figur 39,1)

```
public static class LoanPaymentCalculator
{
    1 reference
    public static (double TotalPerYear, double AmountPerPayment, double TotalAmountPaid) CalcLoanPayment (
        double loanAmount, double interestRate, double amountOfYears, double paymentsPerYear)
    {
        //udregning af betaling pr aar
        double totalPerYear = loanAmount * ((interestRate) / (1 - Math.Pow(1 + interestRate, -amountOfYears)));
        //afrunder svar til 2 decimaler
        double roundedTotalPerYear = Math.Round(totalPerYear, 2);

        //tager betaling pr aar og deler ud i antal ydelses pa aaret
        double amountPerPayment = totalPerYear / paymentsPerYear;
        //afrunde til 2 decimaler
        double roundedAmountPerPayment = Math.Round(amountPerPayment, 2);

        //berening af samlet betalt beløb
        double totalAmountPaid = totalPerYear * amountOfYears;
        //afrund til 2 decimaler
        double roundedTotalAmountPaid = Math.Round(totalAmountPaid, 2);

        return (roundedTotalPerYear, roundedAmountPerPayment, roundedTotalAmountPaid);
    }
}
```

Figur 39,1 - CalcLoanPayment()

Metoden tager imod lånets størrelse, årlig rente, antal af år lånet forløber sig over og antal betalinger om året.

Den har 3 return værdier den totale betaling pr. år, hvor meget man betaler pr betaling på lånet samt den totale betaling ialt når lånet er betalt af. Metoden beregner disse 3 returværdier og runder til 2 decimaler og sender så disse return.

Vi tager et kig på UI-Laget hvor metoden bliver kaldt (figur 39,2)

```
private void BtnCalculate_Click(object? sender, EventArgs e)
{
    if (client.IsSubscribed)
    {
        //lånets størrelse
        double loanAmount = double.Parse(txtLoanAmount.Text);
        //rentefod p.a
        double interestRate = double.Parse(txtAnnualInterestRate.Text) / 100;
        //antal aar
        double amountOfYears = double.Parse(txtAmountOfYears.Text);
        //antal ydelser pr aar
        double paymentsPrYear = double.Parse(txtPaymentsPrYear.Text);

        //kalder beregner med inputs
        (double totalPrYear, double amountPrPayment, double totalAmountPaid) result = LoanPaymentCalculator
            .CalcLoanPayment(loanAmount, interestRate, amountOfYears, paymentsPrYear);

        //udskriver total betaling pr aar i kroner
        lblTotalPrYear.Text = result.totalPrYear.ToString("C", new CultureInfo("da-DK"));

        //udskriver beløb pr betaling i kroner
        lblAmountPrPayment.Text = result.amountPrPayment.ToString("C", new CultureInfo("da-DK"));

        //udskriver total beløb betalt når lan er betalt ud
        lblTotalAmountPaid.Text = result.totalAmountPaid.ToString("C", new CultureInfo("da-DK"));
    }
    else
    {
        MessageBox.Show("You must be subscribed to use calculations");
    }
}
```

Figur 39,2 - *BtnCalculateClick()*

Når en klient navigerer over til beregninger og vælger beregningen ydelse på lån og har udfyldt felterne for beregningen⁵⁷ og trykker på “calculate knappen” køres metoden *BtnCalculate_Click()* (figur 39,2).

Da det kræver at klienten er abonneret for at tilgå beregninger tester vi først på om klienten er abonneret. Hvis klienten ikke har medlemskab vises beskedten “You must be subscribed to use calculations”.

⁵⁷ [Skærmbillede - Beregning af ydelse](#)

Hvis klienten har medlemskab bliver værdierne indsat i felterne parset til en “double”, og så kaldes metoden “CalcLoanPayment”, returværdien gemmes i en tuple “result” som indeholder 3 værdier (totalpryear, amountprpayment og totalamountpaid).

Dette gør det nemt når vi skal skrive svarene tilbage på brugergrænsefladen ved at sætte labels til eksempelvis “result.totalPrYear” hvis det er den totale betaling pr år der skal vises. På alle resultater bruges “ToString” metoden til at vise resultatet i kroner.

Køb af formularer (Lucas)

Som en yderligere funktion til klienter, er der i LawHouse systemet mulighed for at købe formularer som eksempelvis testamenter.

Når klienten nавигerer over til siden “forms” vil de blive vist med en liste af formularer samt mulighed for at se detaljer på den valgte formular⁵⁸.

Hvis den valgte formular ikke endnu er købt af klienten vil der blive vist en “buy” knap. Ved klik på denne knap køres metoden BtnBuy_Click() (figur 40,1)

```
private async void BtnBuy_Click(object? sender, EventArgs e)
{
    btnBuy.Enabled = false;

    DialogResult dialogResult = MessageBox.Show($"Are you sure, that you want to buy the form {selectedForm.Name}?", "Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (dialogResult == DialogResult.Yes)
    {
        //opretter ny ClientFormDocumentUI
        ClientFormDocumentUI clientFormBought = new ClientFormDocumentUI()
        {
            BuyDate = DateTime.Now,
            ClientID = client.PersonID,
            FormDocumentID = selectedForm.FormDocumentID,
        };

        //opretter køebet i db
        bool success = await clientFormBL.BuyFormDocumentAsync(clientFormBought);

        if (success)
        {
            await GetBoughtFormsAsync();
            SetDetails();
            MessageBox.Show($"Form has been bought and sent to: {client.Email}", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
            MessageBox.Show("Failed!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    btnBuy.Enabled = true;
}
```

Figur 40,1 - BtnBuy_Click()

Metoden viser først en messagebox der spørger om brugeren er sikker på de vil købe denne formular. Derefter oprettes en ClientFormDocument, som er modellen til tabellen ClientFormDocuments der viser hvilke klienter der ejer hvilke forms samt købsdato.

⁵⁸ [Screenshot - FormularerView](#)

Så skal dette objekt oprettes i databasen og metoden BuyFormDocumentAsync bliver kaldt fra business logic (Figur 40,2)

```
public async Task<bool> BuyFormDocumentAsync(ClientFormDocumentUI clientFormUI)
{
    ClientFormDocument clientForm = modelConverter.ConvertFromClientFormUI(clientFormUI);
    return await dbAccess.BuyFormDocumentAsync(clientForm);
}
```

Figur 40,2 - BuyFormDocumentAsync, Business Logic

Denne metode tager en ClientFormDocument som input parameter. Det første metoden gør at konvertere UI-Modellen om til en EntitetsModel og kalder så metoden BuyFormDocumentAsync() fra DataAccess (figur 40,3)

```
public async Task<bool> BuyFormDocumentAsync(ClientFormDocument clientForm)
{
    try
    {
        await db.ClientFormDocuments.AddAsync(clientForm);
        return await db.SaveChangesAsync() > 0;
    }
    catch (Exception)
    {
        return false;
    }
}
```

Figur 40,3 - BuyFormDocumentAsync, Data Access

Denne metode tager imod et objekt af Entitets modellen ClientFormDocument. Entiteten bliver nu tilføjet til dbContext og så kaldes SaveChangesAsync på vores dbContext for at gemme ændringerne i databasen. SaveChangesAsync returnerer Task<int> som er antallet af ændringer i databasen og da metoden returnerer en boolean tester vi på at der er sket mere end 0 ændringer i databasen og returnerer dette. Hvis en exception fanges returneres false. Denne boolean retunes hele vejen tilbage til UI igennem businesslogic.

Tager vi et kig på UI metoden BtnBuyClick() (figur 40,1) ser vi nu at hvis true er retuneret henter den igen en liste over købte formularer for klienten for at opdatere at den nu er købt, og viser så en besked hvor den siger køber er succesfuldt og er sendt til klientens email adresse.

Hvis en form allerede er købt skjules knappen “buy” og i stedet ses en “Resend Mail” knap.

Denne knap kalder metoden BtnResendMail_Click() (figur 40,4)

```
private void BtnResendMail_Click(object? sender, EventArgs e)
{
    //gensend mail til klientens mail
    MessageBox.Show($"The form has been resent to: {client.Email}");
}
```

Figur 40,4 - BtnResendMail_Click()

Denne metode er en simulering. Normalt ville man lave en email service der sendte formularen som eksempelvis pdf til klientens email. Dette er ikke implementeret og derfor simuleres dette ved at vise en besked om at formularen er sendt til klientens e-mail.

Oversigt over advokater (Bilal)

Kunder der har adgang til systemet, kan se en oversigt over advokater ved at klikke “Lawyers”⁵⁹ i menuen, så de kan finde den rette til deres interesseområde. Her har vi udeladt advokatsekretærer, da formålet var at vise advokater med deres kontaktoplysninger og specialer.

- Det ville dog give mening at vise andre ansatte også, så de evt. kan kontakte en advokatsekretær til møder, dokumenthåndtering, spørgsmål vedr. udfyldning af form og øvrige henvendelser.

⁵⁹ [Screenshot af LawyersOverview\(for clients\)](#)

```

167     |   1 reference
168     |   private void SetupPanelContent()
169     |   {
170     |   |   foreach (LawyerUI lawyer in lawyerUIs)
171     |   |   {
172     |   |   |   LawyerCardMini lawyerCard = new LawyerCardMini(lawyer);
173     |   |   |   // For hver child lawyerControl i LawyerCard lawyerControl, tilknyt klik event
174     |   |   |   foreach (Control control in lawyerCard.Controls)
175     |   |   |   {
176     |   |   |   |   control.Click += (sender, e) => LawyerCard_Click(lawyerCard, e);
177     |   |   |   }
178
179     |   |   lawyerCard.Margin = new Padding(23); // Mellemrum for hver user-control
180     |   |   flpnlLawyers.Controls.Add(lawyerCard);
181     |   |   OriginalLawyerControls.Add(lawyerCard);
182
183   }
184
185   |   1 reference
186   |   private void LawyerCard_Click(object? sender, EventArgs e)
187   |
188   |   |   // Når der klikkes på en advokat, vises detaljer i højre side panel
189   |   |   if (sender is LawyerCardMini lawyerCard)
190   |   |   {
191   |   |   |   if (lawyerCard != null)
192   |   |   |   {
193   |   |   |   |   pnlLawyerDetails.Controls.Clear();
194   |   |   |   |   // Find og vis den valgte advokat
195   |   |   |   |   pnlLawyerDetails.Controls
196   |   |   |   |   .Add(new LawyerCardDisplay(lawyerUIs.SingleOrDefault(x => x.PersonID == lawyerCard.LawyerID)));
197   |   |   |   }
198   |   }

```

Figur 41.1 - LawyersOverview / SetupPanelContent()

Efter at have hentet alle advokater og alle specialer, bliver panelet med advokaterne vist med LawyerCardMini user-control, for nemmere identiskt setup af visningen. Hver control bliver tilknyttet klik-eventet, der sørger for at vise detaljer om advokaten i panelet til højre, så man har et indblik i flere detaljer, samt specialerne.

Filtrerings-funktionen bliver håndteret sådan her:

```

1 reference
95     private void FilterLawyerControls()
96     {
97         switch (cboxSpecialities.SelectedIndex)
98         {
99             // Ved valg af tom item, vis alle advokater fra original listen
100            case 0:
101                foreach (Control control in OriginalLawyerControls)
102                {
103                    control.Show();
104                }
105                break;
106            // Ellers, skal der vises de advokater der har den valgte speciale
107            default:
108                string selectedSpeciality = cboxSpecialities.SelectedItem.ToString();
109
110                List<int> matchingLawyerIDs = new List<int>();
111
112                // Match speciale med alle advokater og gem deres ID i listen
113                matchingLawyerIDs = lawyerSpecialityUIs
114                    .Where(x => x.SpecialityName == selectedSpeciality)
115                    .Select(x => x.LawyerID).ToList();
116
117                // Vis/skjul relevante advokater
118                foreach (LawyerCardMini lawyerControl in OriginalLawyerControls)
119                {
120                    if (matchingLawyerIDs.Contains(lawyerControl.LawyerID))
121                        lawyerControl.Show();
122                    else
123                        lawyerControl.Hide();
124                }
125                break;
126
}

```

Figur 41.2 - LawyersOverview | FilterLawyerControls()

Når kunden har valgt en speciale fra kombo-boksen, bliver ID'erne på de advokater der har specialen, gemt i en liste ved brug af LINQ og vist ved at iterere over advokat-controls med matche på ID. Kunden kan også sortere listen på navn og by.

Panelet med advokaterne er et flowlayout panel, der gør det mere dynamisk at tilpasse. Vi synes at denne oversigt er et godt værktøj for kunden, så man hurtigt kan finde relevante advokater og deres informationer. Det kunne dog tænkes, at en søgefunktion ville hæve brugeroplevelsen, hvis der skulle være mange flere advokater i virksomheden, vi har dog valgt at udelade det.

Distance Matrix API (Bilal, Lucas & Rasmus)

Vi har implementeret et Google API⁶⁰ til at hjælpe arbejdsgange for advokater, når der skal registreres kilometer på en service, så beregningen sker præcist og advokaten ikke manuelt skal åbne andre servicer for at udregne dette. Det er også forsvarligt overfor klienten, da der

⁶⁰ [Distance Matrix API overview | Google for Developers](#)

faktureres for denne service, at man kan fremvise eller dokumentere, hvor værdien kommer fra.

```

3   <namespace> DataAccess.APIRequests
4   {
5       <reference>
6       <public class DistanceMatrixAPI>
7       {
8           HttpClient client;
9           string apiKey;
10      <reference>
11      <public DistanceMatrixAPI>()
12      {
13          client = new HttpClient();
14          client.DefaultRequestHeaders.Add("Accept", "application/json");
15          apiKey = ConfigurationManager.AppSettings["GoogleApiKey"];
16      }
17      <reference>
18      <public async Task<string> GetDistanceAsync(string origin, string destination)>
19      {
20          string url = $"https://maps.googleapis.com/maps/api/distancematrix/json?units=metric&origins=" +
21          $"{origin}&destinations={destination}&key={apiKey}";
22
23          HttpResponseMessage response = await client.GetAsync(url);
24
25          return await response.Content.ReadAsStringAsync();
26      }
27  }

```

Figur 42.1 - DistanceMatrixAPI / GetDistanceAsync()

Denne API request bruger vores API-nøgle og URL fremskaffet af Google tjenesten, som tager starts- og slutadressen og returnerer en json-string. Eksempel:

```

1   <{
2       "destination_addresses" :
3       [
4           "Los Angeles, CA, USA"
5       ],
6       "origin_addresses" :
7       [
8           "New York, NY, USA"
9       ],
10      "rows" :
11      [
12          {
13              "elements" :
14              [
15                  {
16                      "distance" :
17                      {
18                          "text" : "4,496 km",
19                          "value" : 4496132
20                      },
21                      "duration" :
22                      {
23                          "text" : "1 day 17 hours",
24                          "value" : 147896
25                      },
26                      "status" : "OK"
27                  ]
28              }
29          ],
30      "status" : "OK"
31  }
32

```

Figur 42.2 - Json-string output

Det returnerede json-string bliver konverteret (deserializes) til Distance Matrix-objektet, så vi kan udtrække de relevante værdier:

```

5   <namespace BusinessLogic.API
6   {
7     <references
8       public class DistanceMatrixBL
9       {
10         DistanceMatrixAPI db;
11         public DistanceMatrixBL()
12         {
13           db = new DistanceMatrixAPI();
14         }
15         <reference
16         public async Task<DistanceMatrix> GetDistanceAsync(string origin, string destination)
17         {
18           string jsonData = await db.GetDistanceAsync(origin, destination);
19           return JsonConvert.DeserializeObject<DistanceMatrix>(jsonData);
20         }
21       }
22     }

```

Figur 42.3 - DistanceMatrixBL / GetDistanceAsync()

Funktionen bruges i vores DistanceCalculatorView⁶¹, når kørselsservicen er valgt i en given case:

```

60   <reference
61   private async void BtnSearch_Click(object? sender, EventArgs e)
62   {
63     DistanceMatrix calcResult = await distanceMatrixBL.GetDistanceAsync(txtOrigin.Text, txtDestination.Text);
64     if (calcResult.status == "OK")
65     {
66       if (calcResult.rows[0].elements[0].status == "OK")
67       {
68         txtResult.Text = calcResult.rows[0].elements[0].distance.text;
69         txtDuration.Text = calcResult.rows[0].elements[0].duration.text;
70       }
71       else
72         MessageBox.Show("Couldn't find address", "Error",
73                         MessageBoxButtons.OK, MessageBoxIcon.Error);
74     }
75     else
76     {
77       MessageBox.Show("Could'nt connect to service, please try again later or contact costumer support",
78                         "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
79     }
80   }

```

Figur 42.4 - DistanceCalculatorView / Search_Click

Der vises duration og afstanden for turen, så man har en ide om hvor lang tid det tager. Advokaten kan vælge at trykke “Save” og tilskrive distance resultatet servicen.

⁶¹ [Screenshot af DistanceCalculator](#)

Rapport til tekstfil (Bilal, Lucas & Rasmus)

Udskrift af rapport til en txt-fil, sker i klassen CaseUI som genererer filen med detaljerede oplysninger om sag, de tilknyttede tjenester og interesserter.

```

27     1 reference
28     public async Task<bool> PrintDetailsAsync(string path)
29     {
30         int casePadding = 25;
31
32         List<string> caseHeader = new List<string>()
33         {
34             "CaseID", "Title", "Type", "Status", "Creation Date", "End Date", "Hours",
35             "Total Price", "Lawyer", "Lawyer Number", "Client", "Client Number"
36         };
37
38         List<string> caseData = new List<string>()
39         {
40             CaseID.ToString(), Title, CaseType.Title, Status.ToString(), CreationDate.ToString("d"), EndDate.ToString("d"),
41             EstimatedHours.ToString(), TotalPrice.ToString(), Lawyer.ToString(), Lawyer.PersonID.ToString(),
42             Client.ToString(), Client.PersonID.ToString()
43         };
44
45         // Padding the headers and caseData to ensure alignment
46         for (int i = 0; i < caseHeader.Count; i++)
47         {
48             caseHeader[i] = caseHeader[i].PadRight(casePadding).ToUpper();
49             caseData[i] = caseData[i].PadRight(casePadding);
        }
    
```

Figur 43.1 - CaseUI | PrintDetailsAsync() - Case

Kolonner/headers for sagen samles i en liste, og data/værdier samles i en anden liste, som bliver itereret over og Padding bliver tilført. Dette gør, at de får luft mellem bredden, så man opnår en ensartet struktur. Det samme gøres også for ydelserne:

```

51     int caseServicePadding = 25;
52
53     List<string> caseServiceHeader = new List<string>()
54     {
55         "ServiceID", "Name", "Price Type", "Status", "Start Date", "End Date", "Hours Worked",
56         "Units", "Price/Unit", "Total Price", "Lawyer", "Lawyer Number"
57     };
58
59     for (int i = 0; i < caseServiceHeader.Count; i++)
60     {
61         caseServiceHeader[i] = caseServiceHeader[i].PadRight(caseServicePadding).ToUpper().ToString();
62     }
63
64     List<List<string>> caseServiceData = new List<List<string>>();
65
66     foreach (CaseServiceUI caseService in CaseServices)
67     {
68         List<string> caseServiceLine = new List<string>()
69         {
70             caseService.ServiceID.ToString(), caseService.Service.Name, caseService.Service.PriceType.ToString(), caseService.Status, caseService.StartDate.ToString("d"),
71             caseService.EndDate?.ToString("d") ?? "null", caseService.HoursWorked.ToString(), caseService.Units.ToString(),
72             caseService.Service.Price.ToString(), caseService.TotalPrice.ToString(), caseService.Lawyer.ToString(), caseService.LawyerID.ToString()
73         };
74
75         for (int i = 0; i < caseServiceLine.Count; i++)
76         {
77             caseServiceLine[i] = caseServiceLine[i].PadRight(caseServicePadding).ToString();
78         }
79         caseServiceData.Add(caseServiceLine);
80     }
    
```

Figur 43.2 - CaseUI | PrintDetailsAsync() - Service

Her bliver der så brugt en løkke, for at tilpasse alle ydelser, hvis der skulle være flere tilknyttede til sagen.

```

83     // Combine headers and caseData into lines
84     string caseHeaderRow = string.Join("", caseHeader);
85     string caseDataRow = string.Join("", caseData);
86     string caseServiceHeaderRow = string.Join("", caseServiceHeader);
87
88     // Write to the file
89     List<string> rowsToPrint = new List<string>();
90     rowsToPrint.Add(caseHeaderRow);
91     rowsToPrint.Add(caseDataRow);
92     rowsToPrint.Add("");
93     rowsToPrint.Add("-----");
94     rowsToPrint.Add("");
95     rowsToPrint.Add(caseServiceHeaderRow);
96
97
98     foreach (List<string> serviceLine in caseServiceData)
99     {
100         rowsToPrint.Add(string.Join("", serviceLine));
101     }
102
103     try
104     {
105         await File.WriteAllLinesAsync(path, rowsToPrint);
106     }
107     catch (Exception)
108     {
109         return false;
110     }
111     return true;
112 }
```

Figur 43.3 - CaseUI / PrintDetailsAsync() - Print Rows

Til sidst, bruges der bl.a. `string.Join()`, for at danne en string for hver række og “printe” hver række ud med `File.WriteAllLinesAsync()` til en tekstfil.

CaseUI-klassen fungerer som *Information Expert*, da den håndterer udskrivningen af dens egne data, idet klassen har de nødvendige viden om sine attributter og relaterede klasser. Dette resulterer i *High Cohesion*, da metoden `PrintDetails` fokuserer på en enkelt opgave - udskrivning af sagsoplysninger. Samtidig opnår man *Low Coupling*, da metoden er mindre afhængig af andre klasser, hvilket reducerer kompleksiteten og forbedrer vedligeholdelsen.

Denne funktion skaber værdi for begge parter, da den kan tilgås af ansatte og klienter, så der nemt kan dannes en rapport over sager og tjenester, som virksomheden fakturerer for.

- Der er også implementeret en funktion hvorpå man kan udskrive en mere detaljeret rapport.

Åbning af pdf (Bilal, Lucas & Rasmus)

For at optimere alle brugere af dette systems oplevelse, er der implementeret en hjælpefunktion. På alle forms er der nederst i højre hjørne en blå understreget tekst “[Need Help?](#)” Når en bruger trykker på denne vises en pdf med en guide til hvordan den enkelte side bruges.

For at implementere dette er der i vores BusinessLogic lag oprettet en mappe HelpService hvor der er tilføjet en statisk klasse “OpenPDF” (Figur 44,1).

```
namespace BusinessLogic.HelpServices
{
    30 references
    public static class OpenPDF
    {
        30 references
        public static bool ShowPDF(string filename)
        {
            string filePath = $"Resources\\HelpPdfs\\{filename}.pdf";

            if (File.Exists(filePath))
            {
                Process.Start(new ProcessStartInfo(filePath) { UseShellExecute = true });
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

Figur 44,1 - OpenPDF Statisk klasse

Denne klasse er gjort statisk da der ikke anses nogen grund for instantiering af klassen. Ingen data bliver gemt den laver bare en udførsel nemlig at åbne en pdf, og samtidigt kan den benyttes globalt i projektet.

Metoden ShowPDF tager en string som input, filnavnet på pdfen der skal åbnes. Så oprettes Path til filen, alle vores hjælpe pdf'er ligger i UI projektet i en mappe der hedder resources og deri er der en mappe HelpPdfs. Så bliver der tjekket på om filen eksisterer på den gældende path.

Hvis filen eksisterer, gøres der brug af klassen Process og “Start()” metoden til at åbne et eksternt program/fil. UseShellExecute er sat til true så den bruger systemets shell, altså at

systemet bestemmer hvilken måde filen/programmet skal åbne på, eksempelvis vil et link på en windows nok åbnes i Microsoft edge og en Mac computer ville tage Safari. Den tager standarden for systemet. Hvis filen åbnes returneres true.

Herunder vises eksempel for brug af klassen (figur 44,2)

```
private void LblHelp_Click(object? sender, EventArgs e)
{
    if (!OpenPDF.ShowPDF("ClientViewHelp"))
    {
        MessageBox.Show("Couldnt open file, PDF was not found!");
    }
}
```

Figur 44,2 - Brug af OpenPDF klassen

Her testes på åbning af filen hvis metoden returnerer false meldes en fejlbesked til brugeren.

NUnit Unit Test (Bilal, Lucas & Rasmus)

Til unit testing er der blevet gjort brug af NuGet packagen “NUnit”.

Vi har valgt at køre unit test på vores metode SetCityFromPostalCode, som henter et By navn fra en csv fil udfra et givet postnummer.

Vi startede med at lave en test tabel for metoden (figur x,1).

Test Tabel for SetCityFromPostalcode

Testcase	input	Forventet resultat	Noter
indsæt negativ postnummer	postnr = -7000	string = “-”	tjek om svar ved negativ input
indsæt gyldig postnr 7000	postnr = 7000	string = “Fredericia”	tjek om korrekt svar ved gyldig postnr
indsæt postnummer der ikke existerer	postnr = 1000000000 0000000000 0000000000 0000	string = “-”	tjek resultat ved ikke existerende postnr
indsæt postnummer med bogstaver	postnr = “hej”	string = “-”	tjek resultat ved ugyldig postnr

Figur 45,1 - Test tabel for Unit Test på SetCityFromPostalCode

I denne tabel skrev de test cases vi ville test på ned hvordan vi ville teste samt det forventede resultat. Udfra denne tabel kunne vi oprette unit test på metoden.

```
namespace UnitTest
{
    0 references
    internal class SetCityFromPostalCodeTests
    {
        string postal;
        [SetUp]
        0 references
        public void Setup()
        {
        }

        [Test]
        0 references
        public async Task SetCityFromPostalCode_ValidInput_ReturnCity()
        {
            string excpectedResult = "Fredericia";
            postal = "7000";
            //act
            string result = GetCityFromPostalCode.SetCityFromPostalCode(postal);
            //Assert
            Assert.That(result, Is.EqualTo(excpectedResult));
        }

        [Test]
        0 references
        public async Task SetCityFromPostalCode_NegInput_ReturnHyphen()
        {
            string excpectedResult = "-";
            postal = "-7000";
            //act
            string result = GetCityFromPostalCode.SetCityFromPostalCode(postal);
            //Assert
            Assert.That(result, Is.EqualTo(excpectedResult));
        }
    }
}
```

Figur 45,2 - Unit Test

Figur 45,3 - Unit Test

Ovenfor ses de forskellige test kodet i C# (figur 45,2 og 45,3). Alle metoderne fungerer på samme måde der er oprettet en string expected result som er det svar vi regner med der

kommer tilbage fra det valgte input. Derefter har vi en variable postal som vi sender ind i metoden. Og så bruger vi Assert klasse til at teste på om det svar metoden giver tilbage matcher med det expectedresult.

Alle tests er gået igennem (figur x,3)

▲ ✓ SetCityFromPostalCodeTests (4)	62 ms
✓ SetCityFromPostalCode_Lettersl...	54 ms
✓ SetCityFromPostalCode_NegInp...	< 1 ms
✓ SetCityFromPostalCode_NonExi...	< 1 ms
✓ SetCityFromPostalCode_ValidIn...	8 ms

Figur 45,4 - Gennemført Unit Test

Overvejelser i programmet

Instanser → Dependency Injection/Singleton (Rasmus)

Vi har i udviklingen af vores projekt valgt at instantiere en ny instans af vores BusinessLogic i hver form. Dog har vi diskuteret fordelene og ulemperne ved denne praksis og hvad man ellers kunne have implementeret.

Vi har i overvejelserne snakket om brugen af Dependency Injection. Som kan forbedre testbarheden, kvaliteten af koden og genanvendelighed, ved at man skaber en løs kobling mellem komponenterne.⁶²

Vi har i vores nuværende program valgt at oprette en ny instans af vores DbContext for hver dataaccess. Vi har dermed haft overvejelser om denne tilgang kunne forbedres, idet det måske ikke er den mest optimale ift. ressourcestyring og performance.

⁶² [Architectural principles - .NET | Microsoft Learn](#)

Derfor har vi yderligere overvejet at implementere et Singleton-pattern, for at sikre at der kun arbejdes med en enkelt instans af vores DbContext. Ved at sikre, at der kun arbejdes med en enkelt instans, kan det forbedre ydeevnen.⁶³

Kun få delete funktioner (Lucas)

I systemet har vi ikke implementeret særlig mange Delete funktioner. Vi har overvejet meget hvordan man skulle håndtere sletning af eksempelvis klienter, hvis man slettede en klient skulle klientens sag dermed også slettes?

Vi blev enige om at man nok højst sandsynligt ikke ville slette data permanent i systemet, specielt i et advokatfirma, men nærmere arkivere.

Der gik diskussionen på hvordan dette ville udføres, et “soft delete” hvor man tilføjer attribut isArchived som bool? Oprette helt nye tabeller til arkiveret data, hvor man så slettede fra den aktive tabel og tilføjer til arkiveret? Eller om man skulle oprette en helt ny “arkiveret” database?

Vi konkluderede at den rigtige metode afhænger af systemets krav. Oprettelse af ny database eller nye tabeller kunne nemt håndteres med Transactions. Men den nemmeste metode ville være at lave soft deletes, hvor man ved hentning af data fra databasen ville test på WHERE isArchived = false, og at denne metode nok ville være tilstrækkelig for de fleste advokatfirmer.

Brug af Enums (Lucas)

I LawHouse systemet er der et par steder hvor der har været nogle faste typer med datatype string vi har diskuteret om hvordan skulle håndteres. En advokatydelse har attributten “price type” som lige nu har 3 værdier Hourly, fixed og kilometer. Samt på en sag og en sagsydelseslinje (CaseService) har en attribut der hedder “status” med to muligheder aktiv og lukket.

⁶³ [Dependency injection in ASP.NET Core | Microsoft Learn](#)

Her gik diskussionen på hvorvidt der skulle oprettes en ny tabel med en N-1 relation med de forskellige pristyper og status typer, eller der skulle gøres brug af Enum Tabeller i C#.

Her konkluderede vi at det afhænger af kravet til typerne. Har de et fast udvalg af muligheder der aldrig kan ændres ville der blive gjort brug af en Enum hvorimod hvis det skal være fleksibelt, mulighed for udvidelse skal det oprettes som en ny tabel.

Ved advokatydelsens price type har vi tilføjet en tabel "ServicePriceType", da vi godt kunne se andre ting priser ville blive beregnet på.

Ved status er der endnu ikke oprettet en tabel i databasen eller Enum men her ville en enum nok være mere passende da vi ikke ser andre muligheder en aktiv og lukket.

Brugervenlighed

FURPS+ (Lucas, Bilal & Rasmus)

1. **Functionality**: Programmet er hovedsageligt bygget op omkring CRUD operationerne. Systemet tillader brugere at logge ind med et brugernavn og adgangskode. Systemet tilbyder også klienter en selvbetjeningsdel, hvorpå de kan gøre brug af beregninger, købe formularer og følge op på sine sager.
2. **Usability**: Vi har ved implementering af vores system, taget højde for at der er blevet udviklet en brugervenlig grænseflade. Brugeren kan nemt og hurtigt navigere rundt, og finde de funktioner de skal bruge, uden den helt store kompleksitet.
3. **Reliability**: Systemet har en forbindelse til en SSMS server som altid er oppe i den relevante brugsperiode for projektet. Brugeren er begrænset til at indtaste forkert format af data når der logges ind, oprettes sag og andre features. Programmet forhindrer dette ved eksempelvis ikke at tillade oprettelse af sag, indtil alt data er gyldigt.
4. **Performance**: Systemets ydeevne er vigtig, da det gør at brugerne kan udføre deres operationer hurtigt og effektivt. For at forbedre ydeevnen, har vi haft stor fokus på asynkrone metoder, hvilken håndtere alle databasekald, uden at blokere hovedtråden.

Dette resulterer i at vores brugergrænseflade er responsiv og skaber en bedre brugeroplevelse.

5. **Supportability:** I vores system har vi implementeret hjælpeguides i form af en PDF, der guider brugeren igennem alle funktioner.

Gestaltlove (Lucas)

Gestaltlovene er en række love/principper der skaber en visuel orden, hvilket fører til at brugeroplevelsen øges markant.

Vi har igennem hele projektet forsøgt så vidt muligt at overholde gestaltlovene for et brugervenligt design.

Generelt set har vi igennem hele udviklingen sørget for at elementer, der passer sammen enten står nær, er visuelt forbundet, ligner hinanden eller er indrammet i bokse. Se evt bilag for screenshots af program design⁶⁴.

Brugertest/BlackBox test (Lucas, Bilal)

Vi har i gruppen valgt at opstille en “test” hvor brugeren skal udføre nogle forskellige operationer i systemet for at køre en brugertest.

Operationer:

1. Opret dig som klient og gør brug af en beregningsalgoritme
2. Find en advokat med specialiteten “Criminal law”
3. Log ind som advokat og opret en sag, tilføj ydelse kørsel (fra Fredericia til vejle)
4. Print en rapport over en sag.

Resultater:

1. Nem oprettelse forløb, brugeren var ikke i tvivl om hvad eller hvordan felterne skulle udfyldes.

Beregningssalgoritmen var nem at finde. Brugeren vidste dog ikke, at det krævede medlemskab at bruge beregneren, så det skulle lige ordnes efter meddelelsen.

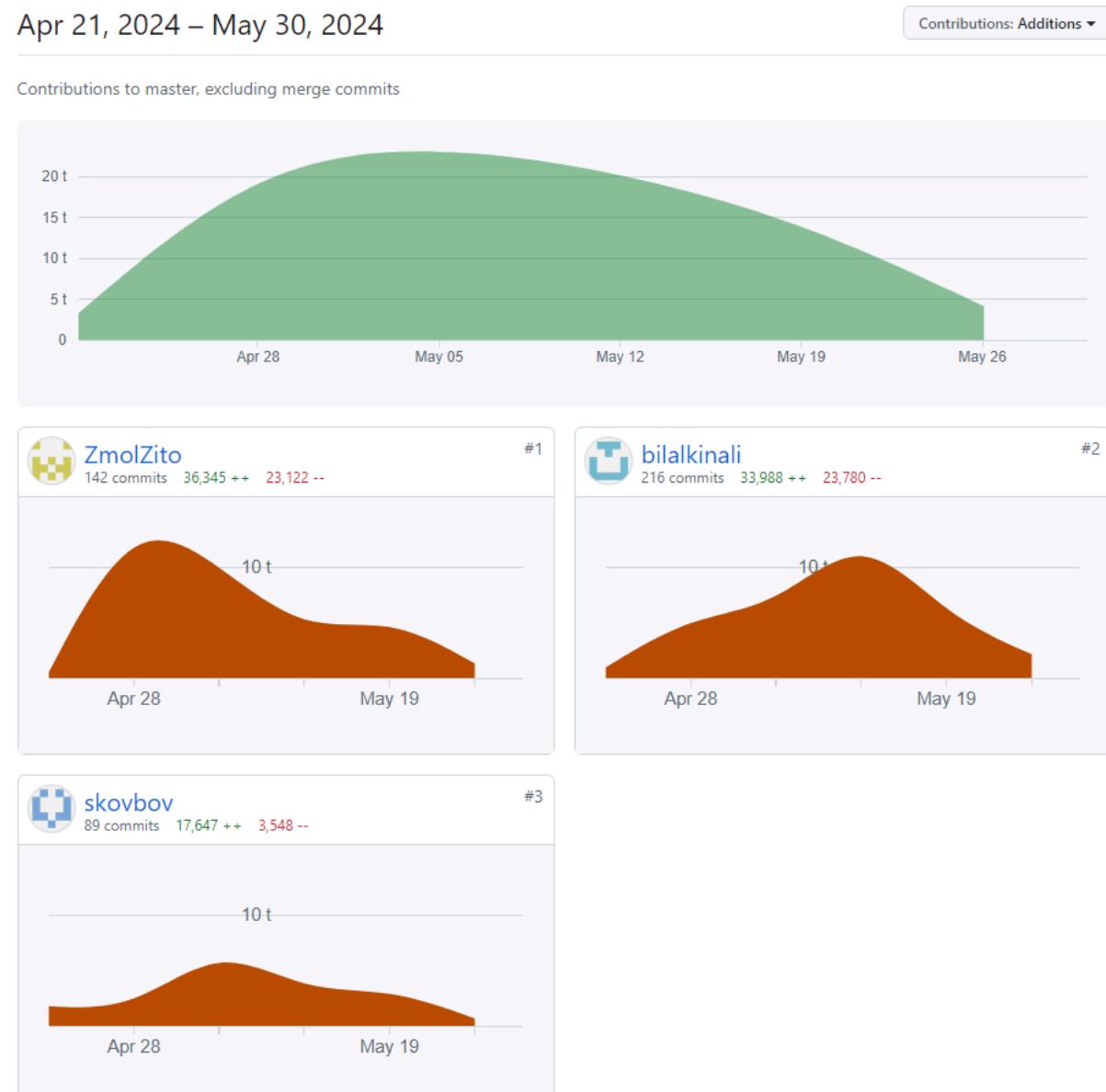
2. For at finde en advokat med en specifik speciale, trykkede brugeren på den korrekte menu-knap, og valgte advokaterne for at se deres detaljer. Filtreringsfunktionen blev ikke brugt (der var kun 2 advokater under testen).

⁶⁴ [Screenshots af program design](#)

3. Efter at give brugeren adgang som advokat, blev en sag oprettet og kørselsydelsen tilføjet. Her blev vores API funktion brugt uden problemer.
4. Brugeren forsøgte her først at lukke en sag, som udløste en fejlmeddeelse, selvom alt var som det skulle være. Udskriften af sagsoplysninger virkede godt, og brugeren var tilfredsstillet.

Denne test gav os et godt indblik i fejl, som vi ikke selv havde set, potentielt fordi der var lavet små ændringer i validering. Fejlene blev dog hurtigt rettet til.

Bidrag til koden



Figur 46 - Github, Additions to kode.

I figur 46 vises de enkelte medlemmers bidrag til koden via github. Disse tal stemmer ikke helt overens med hvem der har bidraget med. Der er gjort brug stort af pair programming, samt tilføjelse af migration, forms osv tilføjer en hel del autogenererede linjer kode.

Der er bred enighed om, at vi alle har bidraget ligeligt til projektet.

Konklusion

Rapporten præsenterer vores udvikling af IT-systemet til advokatfirmaet LawHouse, hvor vi har anvendt SCRUM til projektstyring. Formålet med systemet er at optimere sags- og klienthåndtering, for at opnå dette har vi implementeret flere funktioner, med det formål at automatisere arbejdsprocessen og forbedre selvbetjeningsmulighederne.

Ved brug af SCRUM har vi opnået en struktureret og effektiv udviklingsproces, der omfatter planlægning af sprints, daglige scrum-møder og sprint evalueringer. Projektledelsen blev understøttet af Atlassian Jira, som gav os et tydeligt overblik over opgaverne og fremskridtet.

I udviklingsfasen har vi anvendt forskellige teknologier og værktøjer såsom Microsoft Visual Studio, SQL Server og Entity Framework. Vores system er struktureret med en 3-lags arkitektur for at sikre fleksibilitet og skalerbarhed, hvor vi anvender asynkron programmering for at sikre en responsiv brugergrænseflade. Vores system gør brug af en valideringsklasse, som benyttes i både vores UI og BusinessLogic-lag, for at minimere risikoen for fejl.

Systemets funktioner inkluderer oprettelse af sager, tilføjelse af ydelser til sager, opdatering af status på sager, samt håndtering af brugere. Klienter kan følge med i deres sager, købe formularer samt benytte beregningsalgoritmer. Derudover har vi implementeret en funktion, der giver mulighed for at udskrive en rapport over en sag og kommet med forslag til fremtidige forbedringer såsom implementering af en arkiverings funktion til håndtering af data sletning.

Vi synes generelt, at der er kommet et godt resultat ud af projektet, og vi mener, at det nye system giver mange fordele til LawHouse ved at øge arbejdseffektiviteten og forbedre brugeroplevelsen.

Perspektivering

I systemet ville det være smart med en måde at arkivere data på. I vores system kan man ikke slette sager, personer osv⁶⁵. Her havde det været smart at implementere en måde hvorpå det var en mulighed at gemme data fra UI, ligesom at ”slette” det men at det så bliver arkiveret i stedet.

En fremtidig opdatering ville indeholde et mailsystem, hvor formularer blev sendt til, Hvor man kunne reset sin kode hvis den er glemt eller et system hvor man kunne skrive skrive ind til lawhouse.

Vi har diskuteret meget, hvorvidt der skulle implementeres et ”log” system. En log der holder styr på hvem der foretager hvilke handlinger i programmet. Det ville være smart hvis man kunne holde styr på hvem der oprettede en sag, opdaterede en sag, tilføjede en ydelse til en sag osv sådan at hvis der var blevet indtastet noget forkert eller tvivlsomt, at man nemt kunne gå ind at se hvem der har foretaget ændringen og spørge ind til det. Dette er ikke implementeret i systemet men ville helt klart være en del af en opdatering til systemet.

Litteraturliste

Bog: Trykt bog

Craig, Larman. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson.

<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore>

<https://www.nuget.org/packages/System.Configuration.ConfigurationManager/9.0.0-preview.4.2426.6.19>

⁶⁵ [Kun få delete funktioner](#)

<https://learn.microsoft.com/en-us/dotnet/api/system.data.sqlclient?view=net-8.0>

<https://www.newtonsoft.com/json>

<https://learn.microsoft.com/en-us/ef/core/modeling/relationships/one-to-many?source=recommendations>

<https://learn.microsoft.com/en-us/ef/core/modeling/relationships/many-to-many?source=recommendations>

<https://learn.microsoft.com/en-us/ef/core/modeling/relationships/one-to-one?source=recommendations>

<https://docs.github.com/en/get-started/getting-started-with-git/ignoring-files>

<https://www.simply.com/dk/>

<https://code-maze.com/prevent-sql-injection-with-ef-core-dapper-and-ado-net/>

<https://www.learnentityframeworkcore.com/inheritance/table-per-type>

<https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext.savechangesasync?view=entity-framework-6.2.0>

<https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles#dependency-injection>

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-5.0>

<https://developers.google.com/maps/documentation/distance-matrix/overview>

Bilag

Use Cases

UC1 - Sekretær

1.1 Create:

1.1 - En Admin skal kunne oprette en sekretær i systemet

1.2 Read:

1.2 - Sekretær og advokater skal kunne se en oversigt over alle sekretær

1.3 Update:

1.3 - En sekretær skal kunne opdatere sine egne oplysninger

UC2 - Advokat

2.1 Create:

2.1: En Admin skal kunne oprette en advokat i systemet

2.2 Read:

2.2: Advokat, Klient og sekretær skal kunne se et udvalg af advokater

2.3 Update:

2.3: En admin skal kunne opdatere advokaters oplysninger

- en advokat kan kun opdatere sine egne oplysninger

2.4 Tilkoble speciale:

2.4: En Admin skal kunne tilkoble et speciale på en Advokat

UC3 - Klient

3.1 Create:

3.1: En Klient, advokat og sekretær skal kunne oprette klient i systemet

3.2 Read:

3.2: En Advokat og sekretær skal kunne se en liste over klienter

- En klient kan kun se sine egne oplysninger

3.3 Update:

3.3: En Klient, sekretær og advokat skal kunne opdatere en klients oplysninger

- Klient kan kun opdaterer sine egne oplysninger

UC4 - Sag

4.1 Create:

4.1a: En advokat, (sekretær) skal kunne oprette sag med tilknyttet klient og advokat

4.2 Read:

4.2: En klient, advokat og sekretær skal kunne se en liste over sine sager

4.3 Update:

4.3: En advokat og en sekretær skal kunne opdaterer en sag med status

4.4 Tilknyt ydelse:

4.4: En sekretær og en advokat skal kunne tilknytte en ydelse til en sag

4.5 Print Case

4.5: Alle skulle kunne udskrive detaljer om en case

- En klient kan kun udskrive om sin egen

UC5 - Formular

5.1 Create:

5.1: En Admin skal kunne oprette nye formular typer i systemet

5.2 Read:

5.2: En Klient skal kunne se en oversigt over sine købte formularer

5.3 Update:

5.3a: En Admin skal kunne opdaterer en formular med titel og beskrivelse

5.4 Delete:

5.4a: En Admin skal kunne slette formularer fra systemet

5.5 Buy:

5.5a En klient skal kunne købe en formular

UC7 - Klient Abonnement -> Medlemskab

6.1 Create:

6.1: En Klient skal kunne oprette medlemskab hos LawHouse

6.2 Read:

6.2: En klient skal kunne se detaljer på sit eget medlemskab

6.2: En klient skal kunne se en oversigt over mulige abonnementer

UC8 - Beregninger

8.1 Read:

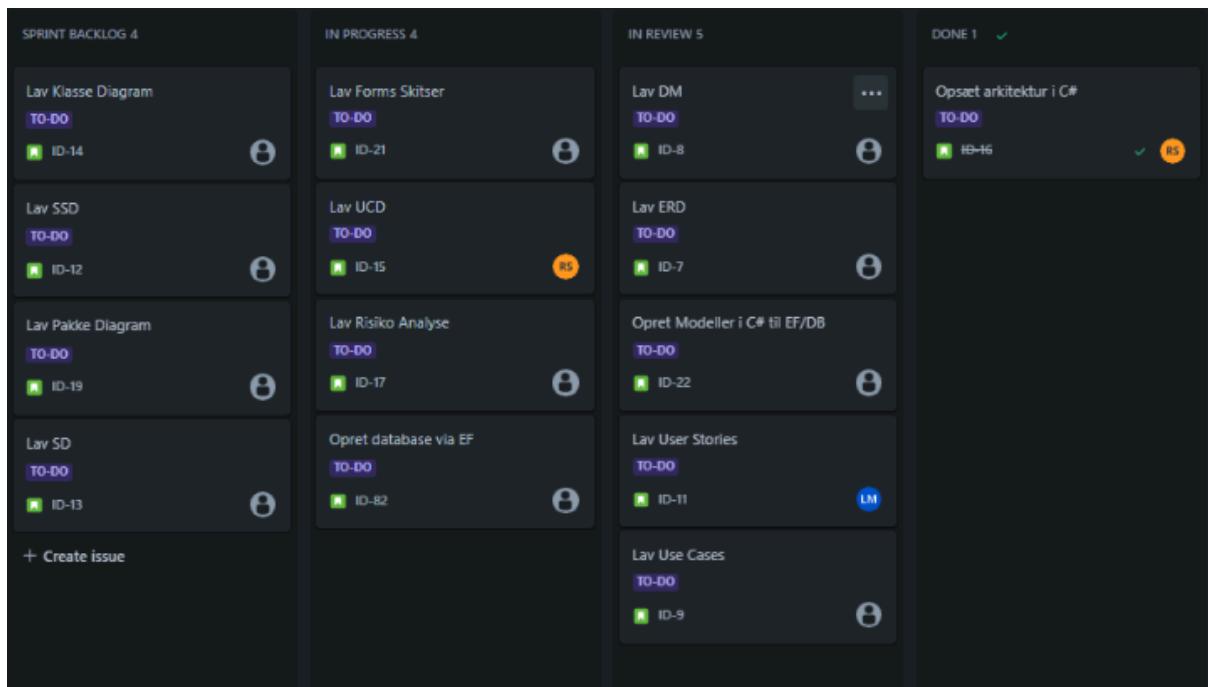
8.1: En klient med abonnement skal kunne gøre brug af beregningsalgoritmerne

Scrum Board

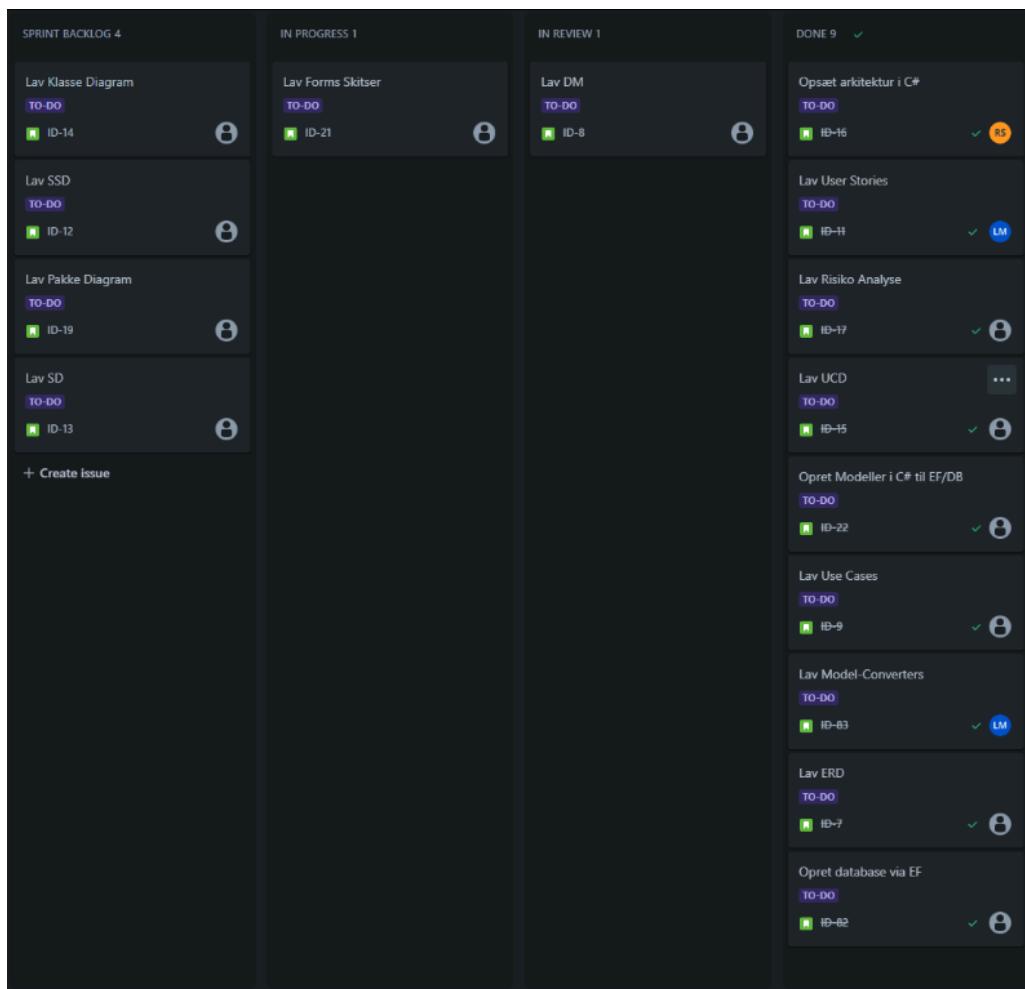
Sprint 1

SPRINT BACKLOG 7	IN PROGRESS 2	IN REVIEW 4	DONE ✓
Lav Risiko Analyse TO-DO ID-17	Lav Use Cases TO-DO ID-9	Opsæt arkitektur i C# TO-DO ID-16	
Lav UCD TO-DO ID-15	Lav User Stories TO-DO ID-11	Lav DM TO-DO ID-8	
Lav Klasse Diagram TO-DO ID-14		Lav Erd TO-DO ID-7	
Lav Forms Skitser TO-DO ID-21		Opret Modeller i C# til EF/DB TO-DO ID-22	
Lav SSD TO-DO ID-12			
Lav Pakke Diagram TO-DO ID-19			
Lav SD TO-DO ID-13			
+ Create issue			

Scrum Board, Start Sprint 1

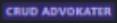
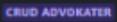
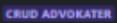


Scrum Board, Midt sprint 1



Scrum Board, slut sprint 1

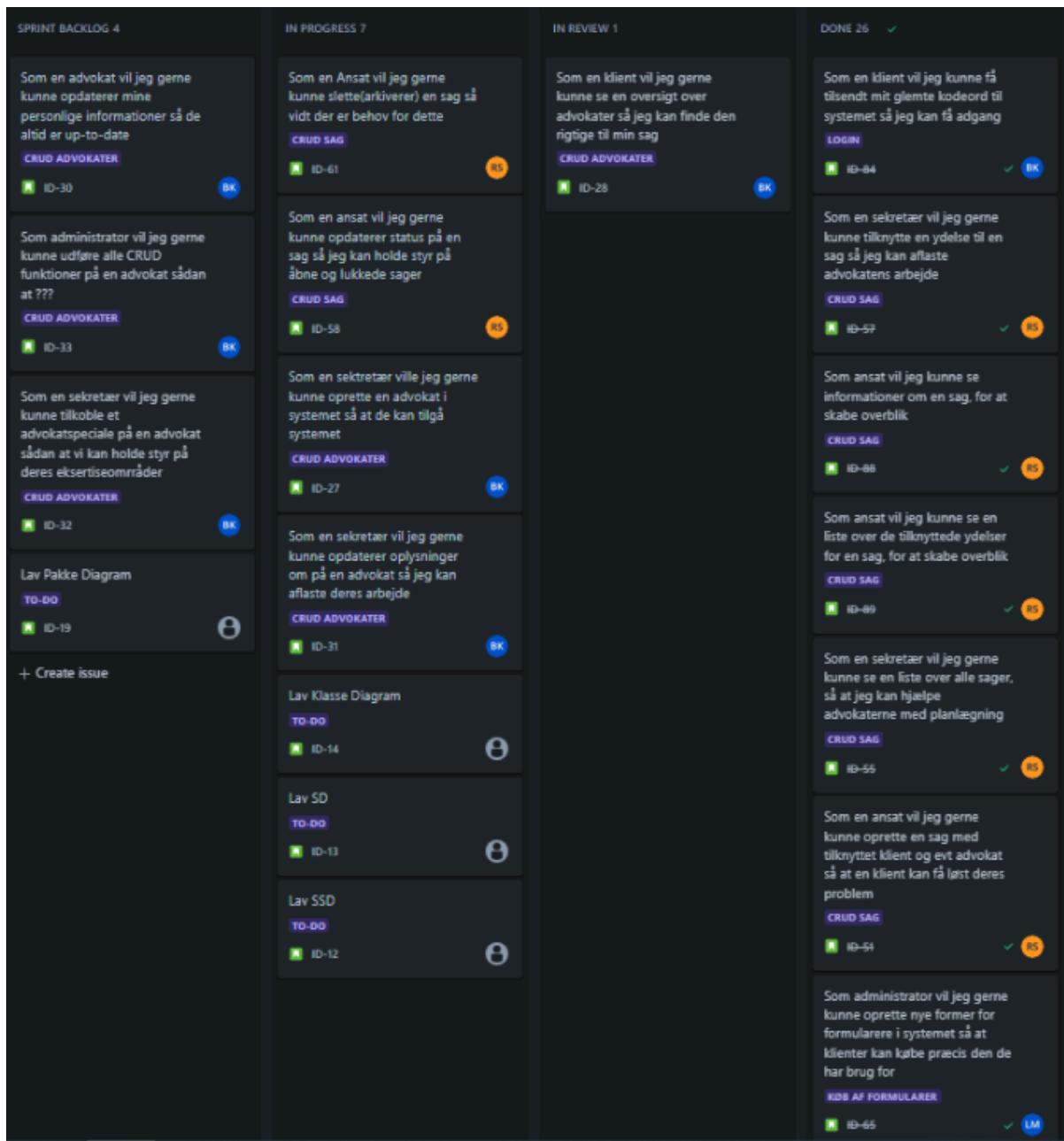
Sprint 2

SPRINT BACKLOG 35	IN PROGRESS 1	IN REVIEW	DONE ✓
Lav SD TO-DO  ID-13 			
Som advokat vil jeg gerne kunne logge ind på systemet for at kunne se oversigt mine tildekte sager  LOGIN			
 ID-25 			
Som klient vil jeg gerne kunne logge ind på systemet for at kunne tilgå mine sager  LOGIN			
 ID-24 			
Som klient vil jeg gerne kunne oprette mig som bruger i systemet, så at jeg kan få adgang til selvbetjenings mulighederne  LOGIN			
 ID-40 			
Som en advokat vil jeg gerne kunne opdaterer mine personlige informationer så de altid er up-to-date  CRUD ADVOKATER			
 ID-30 			
Som en sekretær vil jeg gerne kunne opdaterer oplysninger om på en advokat så jeg kan aflæste deres arbejde  CRUD ADVOKATER			
 ID-31 			
Som en sekretær ville jeg gerne kunne oprette en advokat i systemet så at de kan tilgå systemet  CRUD ADVOKATER			
 ID-27 			
Som en klient vil jeg gerne			

Scrum Board, Start sprint 2

SPRINT BACKLOG 9	IN PROGRESS 11	IN REVIEW 2	DONE 14 ✓
Som en advokat vil jeg gerne kunne opdaterer mine personlige informationer så de altid er up-to-date CRUD ADVOKATER ID-30 BK	Som en ansat vil jeg gerne kunne opdaterer status på en sag så jeg kan holde styr på åbne og lukkede sager CRUD SAG ID-58 BS	Som en sekretær vil jeg gerne kunne se en liste over alle sager, så at jeg kan hjælpe advokaterne med planlægning CRUD SAG ID-55 BS	Som en klient vil jeg kunne få tilsendt mit glemt kodeord til systemet så jeg kan få adgang LOGIN ID-84 ✓ BS
Som en sekretær vil jeg gerne kunne opdaterer oplysninger om på en advokat så jeg kan aflaste deres arbejde CRUD ADVOKATER ID-31 BK	Som en klient vil jeg gerne kunne se en oversigt over advokater så jeg kan finde den rigtige til min sag CRUD ADVOKATER ID-28 BK	Lav Forms Skitser TO-DO ID-21 B	Som en ansat vil jeg gerne kunne oprette en sag med tilknyttet klient og evt advokat så at en klient kan få løst deres problem CRUD SAG ID-64 ✓ BS
Som administrator vil jeg gerne kunne udføre alle CRUD funktioner på en advokat sådan at ??? CRUD ADVOKATER ID-33 BK	Som en klient vil jeg gerne kunne se en oversigt over alle formularer der eksisterer så at jeg kan se hvilke muligheder der er at købe KØB AF FORMULARER ID-67 LM		Som en sekretær vil jeg gerne kunne se en oversigt over advokater så jeg kan koble en passende advokat på en sag CRUD ADVOKATER ID-29 ✓ BS
Som en sekretær ville jeg gerne kunne oprette en advokat i systemet så at de kan tilgå systemet CRUD ADVOKATER ID-27 BK	Som ansat vil jeg kunne se en liste over de tilknyttede ydelser for en sag, for at skabe overblik CRUD SAG ID-89 BS		Som en Ansat vil jeg gerne kunne opdaterer en klients oplysninger i tilfælde af de ikke selv kan CRUD Klient ID-46 ✓ LM
Som en sekretær vil jeg gerne kunne tilkoble et advokatspecial på en advokat sådan at vi kan holde styr på deres eksertiseområder CRUD ADVOKATER ID-32 BK	Som en Klient vil jeg gerne kunne se en oversigt over alle mine købte formularer så jeg kan bruge dem jeg har købt KØB AF FORMULARER ID-66 LM		Som administrator vil jeg gerne udføre alle CRUD funktioner på en Klient sådan at ??? CRUD Klient ID-49 ✓ LM
Som en Ansat vil jeg gerne kunne slette(arkiverer) en sag så vidt der er behov for dette CRUD SAG ID-61 BS	Som en klient vil jeg gerne kunne købe en formular så jeg kan få dem jeg har brug for KØB AF FORMULARER ID-64 LM		Som en advokat vil jeg gerne kunne oprette en klient i systemet, så at jeg kan oprette en sag til dem CRUD Klient ID-44 ✓ LM
Som en Admin vil jeg gerne kunne slette formularer fra systemet så at de ikke kan købes længere KØB AF FORMULARER	Som ansat vil jeg kunne tilføje nye ydelser til en sag, så kunden har overblik over hvad der er blevet brugt af ydelser og timer CRUD SAG ID-90 BS		Som en bruger vil jeg kunne skjule mit kodeord ved indtastning så det forbliver hemmeligt LOGIN

Scrum Board, Midt sprint 2



Scrum Board, slut sprint 2

User Stories

Summary :

Lav SD
Lav Klasse Diagram
Lav SSD
Lav Pakke Diagram

Som en ansat vil jeg gerne kunne opdaterer status på en sag så jeg kan holde styr på åbne og lukkede sager

Som en advokat vil jeg gerne kunne opdaterer mine personlige informationer så de altid er up-to-date

Som en sekretær ville jeg gerne kunne oprette en advokat i systemet så at de kan tilgå systemet

Som en klient vil jeg gerne kunne se en oversigt over advokater så jeg kan finde den rigtige til min sag

Som en sekretær vil jeg gerne kunne opdaterer oplysninger om på en advokat så jeg kan aflaste deres arbejde

Som administrator vil jeg gerne kunne udføre alle CRU funktioner på en advokat sådan at bruge systemet

Som en sekretær vil jeg gerne kunne tilkoble et advokatspeciale på en advokat sådan at vi kan holde styr på deres eksertiseområder

Som en Ansat vil jeg gerne kunne slette(arkiverer) en sag så vidt der er behov for dette

Som klient vil jeg gerne kunne logge ind på systemet for at kunne tilgå mine sager

Som advokat vil jeg gerne kunne logge ind på systemet for at kunne se oversigt over mine tildelte sager

Lav Forms Skitser

Som en Klient vil jeg gerne kunne opdaterer mine personlige oplysninger så de altid er up-to-date

Som klient vil jeg gerne kunne oprette mig som bruger i systemet, så at jeg kan få adgang til selvbetjenings mulighederne

Som en advokat vil jeg gerne kunne oprette en klient i systemet, så at jeg kan oprette en sag til dem

Som en Sekretær vil jeg gerne kunne oprette en klient i systemet, sådan at jeg kan oprette en sag til dem

Som en ansat vil jeg gerne kunne oprette en sag med tilknyttet klient og evt advokat så at en klient kan få løst deres problem

Som en Ansat vil jeg gerne kunne se en liste over alle klienter så jeg kan få kontakt til alle klienter

Som en Ansat vil jeg gerne kunne opdaterer en klients oplysninger i tilfælde af de ikke selv kan

Som en sekretær vil jeg gerne kunne slette (arkiverer) en klient i systemet, så vidt de ønsker dette

Som administrator vil jeg gerne kunne udføre alle CRU funktioner på en Klient så at de kan få oprettet sager

Som en sekretær vil jeg gerne kunne se en liste over alle sager, så at jeg kan hjælpe advokaterne med planlægning

Som en sekretær vil jeg gerne kunne se en oversigt over advokater så jeg kan koble en passende advokat på en sag

Som en klient vil jeg gerne kunne købe en formular så jeg kan få dem jeg har brug for

Som en Klient vil jeg gerne kunne se en oversigt over alle mine købte formularer så jeg kan bruge dem jeg har købt

Som en klient vil jeg gerne kunne se en oversigt over alle formularer der eksisterer så at jeg kan se hvilke muligheder der er at købe

Som en Admin vil jeg gerne kunne slette formularer fra systemet så at de ikke kan købes længere

Som administrator vil jeg gerne kunne oprette nye former for formularer i systemet så at klienter kan købe præcis den de har brug for

Som en klient vil jeg kunne få tilsendt mit glemte kodeord til systemet så jeg kan få adgang

Som en bruger vil jeg kunne skjule mit kodeord ved indtastning så det forbliver hemmeligt

Som ansat vil jeg kunne se informationer om en sag, for at skabe overblik

Som ansat vil jeg kunne se en liste over de tilknyttede ydelser for en sag, for at skabe overblik

Som ansat vil jeg kunne tilføje nye ydelser til en sag, så kunden har overblik over hvad der er blevet brugt af ydelser og timer

Som en administrator vil jeg gerne kunne opdaterer en formular så at de passer til behov

Som en sekretær vil jeg gerne kunne tilknytte en ydelse til en sag så jeg kan aflaste advokatens arbejde

User stories for sprint 2

Sprint 3

Sprint 3

SPRINT BACKLOG (16)	IN PROGRESS (9)	IN REVIEW (1)	DONE (0)
<p>Som administrator vil jeg gerne kunne udføre alle CRUD funktioner på en sekretær sådan at ???</p> <p>CRUD SEKRETÆR</p> <ul style="list-style-type: none"> ID-35 (BK) ID-95 (RS) 	<p>Som en Ansat vil jeg gerne kunne slette(arkiverer) en sag så vidt der er behov for dette</p> <p>CRUD SAG</p> <ul style="list-style-type: none"> ID-61 (RS) ID-58 (RS) 	<p>Som en klient vil jeg gerne kunne se en oversigt over advokater så jeg kan finde den rigtige til min sag</p> <p>CRUD ADVOKATER</p> <ul style="list-style-type: none"> ID-28 (BK) 	
<p>Som ansat vil jeg gerne kunne tilføje timer til ydelserne, så jeg har et overblik over hvilke datoer og hvor mange timer der er blevet brugt</p> <p>CRUD SAG</p> <ul style="list-style-type: none"> ID-95 (RS) 	<p>Som en ansat vil jeg gerne kunne opdaterer status på en sag så jeg kan holde styr på åbne og lukkede sager</p> <p>CRUD SAG</p> <ul style="list-style-type: none"> ID-58 (RS) 		
<p>Som ansat vil jeg gerne kunne se informationer på hver ydelse, for at skabe overblik</p> <p>CRUD SAG</p> <ul style="list-style-type: none"> ID-96 (RS) 	<p>Som en sekretær vil jeg gerne kunne opdaterer oplysninger om på en advokat så jeg kan afliste deres arbejde</p> <p>CRUD ADVOKATER</p> <ul style="list-style-type: none"> ID-31 (BK) 		
<p>Som en klient med medlemskab vil jeg gerne kunne tilgå beregninger så jeg kan beregne det jeg har brug for</p> <p>BEREGNINGER</p> <ul style="list-style-type: none"> ID-78 (LM) 	<p>Som en ansat vil jeg gerne kunne se en oversigt over alle medlemmer så at vi kan holde styr på hvem der er medlem</p> <p>ABONNEMENT</p> <ul style="list-style-type: none"> ID-76 (LM) 		
<p>Som en advokat vil jeg gerne kunne opdaterer mine personlige informationer så de altid er up-to-date</p> <p>CRUD ADVOKATER</p> <ul style="list-style-type: none"> ID-30 (BK) 	<p>Som klient vil jeg gerne kunne oprette medlemskabs abonnement hos LawHouse så at jeg kan tilgå deres beregninger</p> <p>ABONNEMENT</p> <ul style="list-style-type: none"> ID-72 (LM) 		
<p>Som en ansat vil jeg gerne kunne se en oversigt over alle sekretærer for at jeg ved hvem jeg bruge som hjælp</p> <p>CRUD SEKRETÆR</p> <ul style="list-style-type: none"> ID-36 (BK) 	<p>Lav Klasse Diagram</p> <p>TO-DO</p> <ul style="list-style-type: none"> ID-14 (S) 	<p>Lav SD</p>	

Scrum Board, start sprint 3

SPRINT BACKLOG 7	IN PROGRESS 9	IN REVIEW 6	DONE 4 ✓
Som administrator vil jeg gerne kunne udføre alle CRUD funktioner på en sekretær sådan at ??? CRUD SEKRETÆR  ID-35 BK	Som en Ansat vil jeg gerne kunne slette(arkiverer) en sag så vidt der er behov for dette CRUD SAG  ID-61 RS	Som en ansat vil jeg gerne kunne se en oversigt over alle medlemmer så at vi kan holde styr på hvem der er medlem ABONNEMENT  ID-76 LM	Som klient vil jeg gerne kunne oprette medlemskabs abonnement hos LawHouse så at jeg kan tilgå deres beregninger ABONNEMENT  ID-72 LM
Som administrator vil jeg gerne kunne udføre alle CRUD funktioner på en Sag sådan at ??? CRUD SAG  ID-62 RS	Som ansat vil jeg gerne kunne tilføje timer til ydelse, så jeg har et overblik over hvilke datoer og hvor mange timer der er blevet brugt CRUD SAG  ID-95 RS	Som en klient med medlemskab vil jeg gerne kunne tilgå beregninger så jeg kan beregne det jeg har brug for BEREGNINGER  ID-78 LM	Som en sekretær vil jeg gerne kunne opdaterer oplysninger om på en advokat så jeg kan aflaiste deres arbejde CRUD ADVOKATER  ID-31 BK
Som en klient vil jeg gerne kunne danne en rapport over sagsomkostninger og dets ydelser så at jeg ved hvad min sag koster PDF  ID-92 E	Som en ansat vil jeg gerne kunne se en oversigt over alle sekretærer for at jeg ved hvem jeg bruge som hjælp CRUD SEKRETÆR  ID-36 BK	Som en advokat vil jeg gerne kunne se en liste over mine tildekte sager, så at jeg kan holde styr og planlægge arbejdstid CRUD SAG  ID-54 BK	Som en klient vil jeg gerne kunne se en oversigt over advokater så jeg kan finde den rigtige til min sag CRUD ADVOKATER  ID-28 BK
Som administrator vil jeg gerne kunne udføre alle CRUD funktioner på en advokat sådan at ??? CRUD ADVOKATER  ID-33 BK	Som en klient vil jeg gerne kunne se en liste over mine egne sager, så jeg kan holde styr på dem CRUD SAG  ID-53 LM	Som en sekretær vil jeg gerne kunne tilkoble et advokatspecial på en advokat sådan at vi kan holde styr på deres eksertiseområder CRUD ADVOKATER  ID-32 BK	Som ansat vil jeg gerne kunne se informationer på hver ydelse, for at skabe overblik CRUD SAG  ID-96 RS
Som en ansat vil jeg gerne kunne danne en rapport over sagsomkostninger for en given sag så at jeg kan holde styr på hvad der er lavet på sagen PDF  ID-93 E	Som en klient vil jeg gerne kunne detaljer på mit medlemskab så jeg kan holde styr på funktioner og udgangsdato ABONNEMENT  ID-73 LM	Som en ansat vil jeg gerne kunne opdaterer status på en sag så jeg kan holde styr på åbne og lukkede sager CRUD SAG  ID-58 RS	Som en advokat vil jeg gerne kunne opdaterer mine personlige informationer så de altid er up-to-date CRUD SAG
Som en sekretær vil jeg gerne kunne opdaterer mine personlige oplysninger så de altid er up-to-date	Som en sekretær ville jeg gerne kunne oprette en advokat i systemet så at de kan tilgå systemet CRUD ADVOKATER		

Scrum Board, Midt sprint 3

Projects / 2. Sem Eksamens LawHouse

Sprint 3

Search LM BK RS Epic

SPRINT BACKLOG	IN PROGRESS 6	IN REVIEW 1	DONE 16
+ Create issue	<p>Som administrator vil jeg gerne kunne udføre alle CRUD funktioner på en sekretær sådan at ??? CRUD SEKRETÆR ID-35</p> <p>Som en sekretær vil jeg gerne kunne opdaterer mine personlige oplysninger så de altid er up-to-date CRUD SEKRETÆR ID-38</p> <p>Lav Klasse Diagram TO-DO ID-14</p> <p>Lav SD TO-DO ID-13</p> <p>Lav SSD TO-DO ID-12</p> <p>Lav Pakke Diagram TO-DO ID-19</p>	<p>Som en ansat vil jeg gerne kunne se en oversigt over alle sekretærer for at jeg ved hvem jeg bruge som hjælp CRUD SEKRETÆR ID-36</p>	<p>Som en klient vil jeg gerne kunne danne en rapport over sagsomkostninger og dets ydelser så at jeg ved hvad min sag koster TEKSTFIL ID-92</p> <p>Som en ansat vil jeg gerne kunne se en oversigt over alle medlemmer så at vi kan holde styr på hvem der er medlem ABONNEMENT ID-76</p> <p>Som en ansat vil jeg gerne kunne danne en rapport over sagsomkostninger for en given sag så at jeg kan holde styr på hvad der er lavet på sagen TEKSTFIL ID-93</p> <p>Som klient vil jeg gerne kunne oprette medlemskabs abonnement hos LawHouse så at jeg kan tilgå deres beregninger ABONNEMENT ID-72</p> <p>Som en klient vil jeg gerne kunne se detaljer på mit medlemskab så jeg kan holde styr på funktioner og udløbsdato ABONNEMENT ID-73</p> <p>Som administrator vil jeg gerne kunne udføre alle CRUD</p>

Scrum Board, slut sprint 3

User Stories

Lav SD

Lav Klasse Diagram

Lav SSD

Lav Pakke Diagram

Som en ansat vil jeg gerne kunne se en oversigt over alle sekretærer for at jeg ved hvem jeg bruge som hjælp

Som en sekretær vil jeg gerne kunne opdaterer mine personlige oplysninger så de altid er up-to-date

Som administrator vil jeg gerne kunne udføre alle CRU funktioner på en sekretær sådan at de har adgang til systemet

Summary :

Som en klient vil jeg gerne kunne se en oversigt over advokater så jeg kan finde den rigtige til min sag

Som en advokat vil jeg gerne kunne opdaterer mine personlige informationer så de altid er up-to-date

Som en ansat vil jeg gerne kunne opdaterer status på en sag så jeg kan holde styr på åbne og lukkede sager

Som en sekretær vil jeg gerne kunne opdaterer oplysninger om på en advokat så jeg kan aflaste deres arbejde

Som administrator vil jeg gerne kunne udføre alle CRU funktioner på en advokat sådan at bruge systemet

Som administrator vil jeg gerne kunne udføre alle CRU funktioner på en Sag sådan at vi kan håndtere sager i systemet

Som en klient vil jeg gerne kunne se en liste over mine egne sager, så jeg kan holde styr på dem

Som en advokat vil jeg gerne kunne se en liste over mine tildelte sager, så at jeg kan holde styr og planlægge arbejdstid

Som en klient med medlemskab vil jeg gerne kunne tilgå beregninger så jeg kan beregne det jeg har brug for

Som klient vil jeg gerne kunne oprette medlemskabs abonnement hos LawHouse så at jeg kan tilgå deres beregninger

Som en klient vil jeg gerne kunne se detaljer på mit medlemskab så jeg kan holde styr på funktioner og udløbsdato

Som en ansat vil jeg gerne kunne se en oversigt over alle medlemmer så at vi kan holde styr på hvem der er medlem

Som en ansat vil jeg gerne kunne danne en rapport over sagsomkostninger for en given sag så at jeg kan holde styr på hvad der er lavet på ...

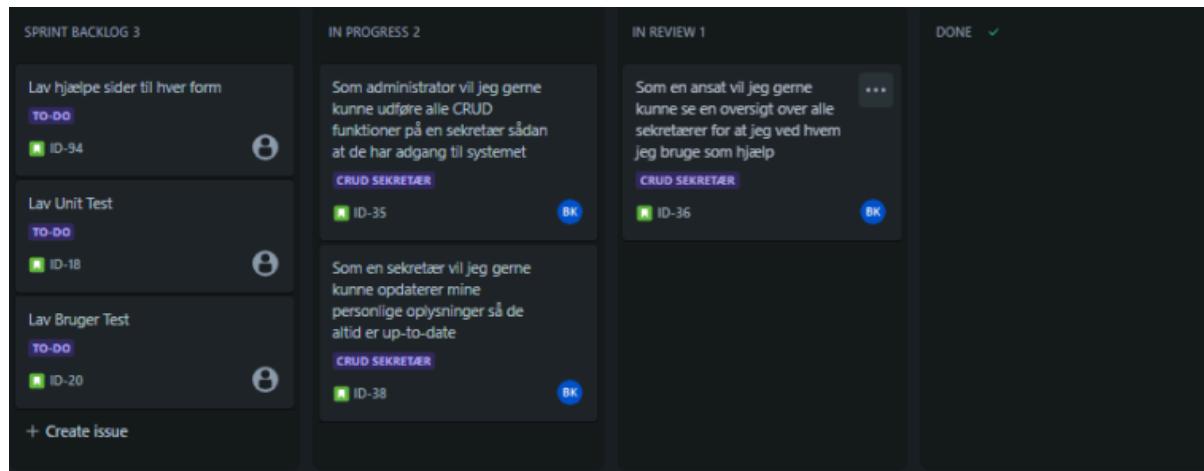
Som en klient vil jeg gerne kunne danne en rapport over sagsomkostninger og dets ydelser så at jeg ved hvad min sag koster

Som ansat vil jeg gerne kunne tilføje timer til ydelserne, så jeg har et overblik over hvile datoer og hvor mange timer der er blevet brugt

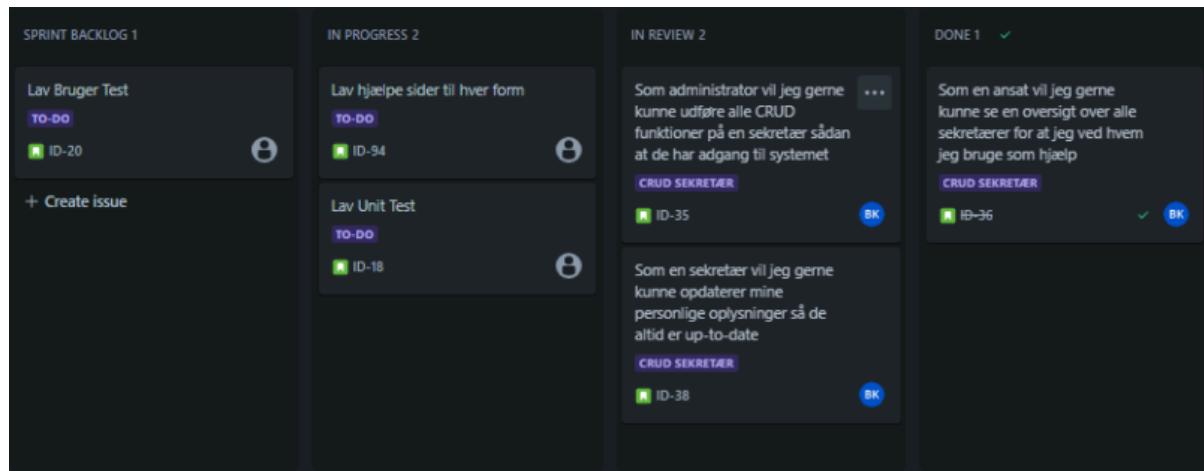
Som ansat vil jeg gerne kunne se informationer på hver ydelse, for at skabe overblik

User stories for sprint 3

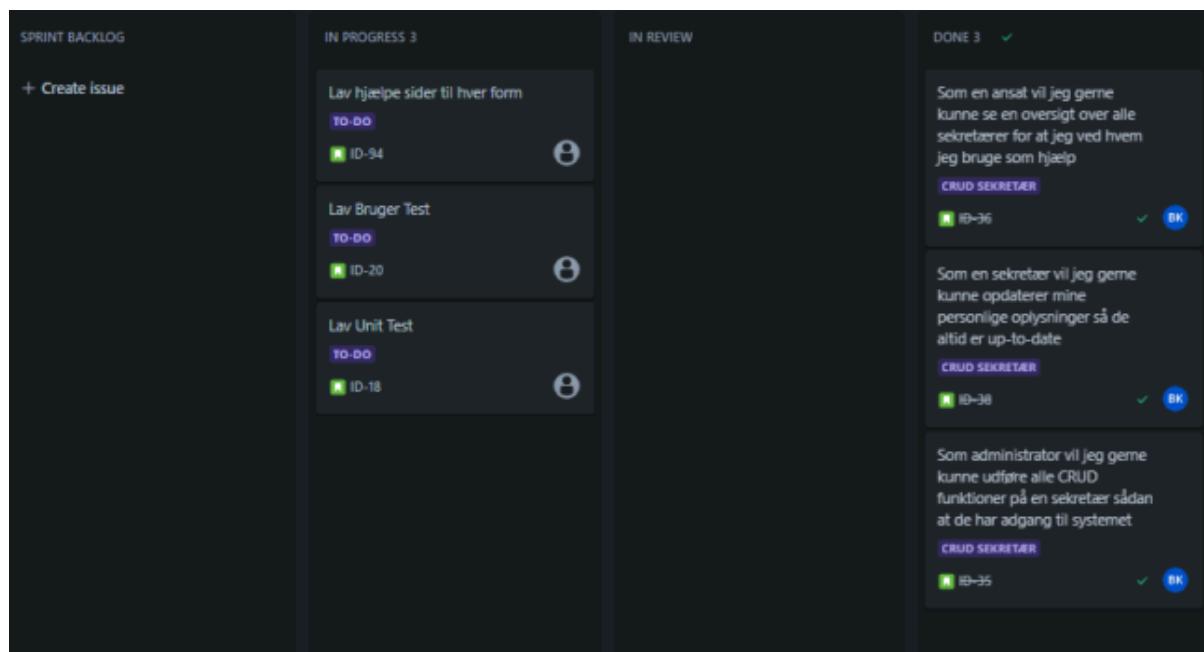
Sprint 4



Scrum Board, Start sprint 4



Scrum Board, Midt sprint 4

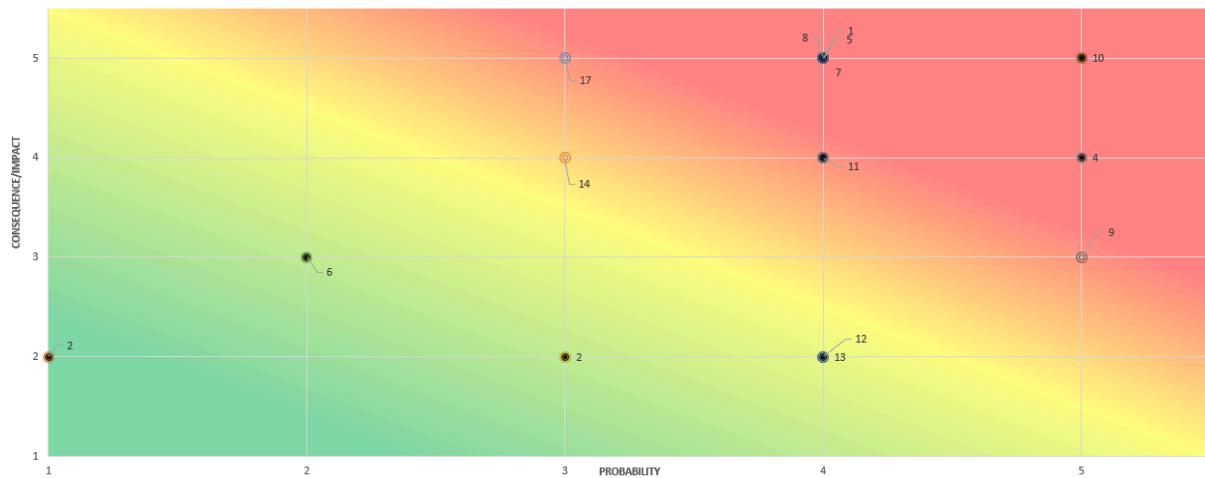


Scrum Board, Slutspint 4

Risikoanalyse

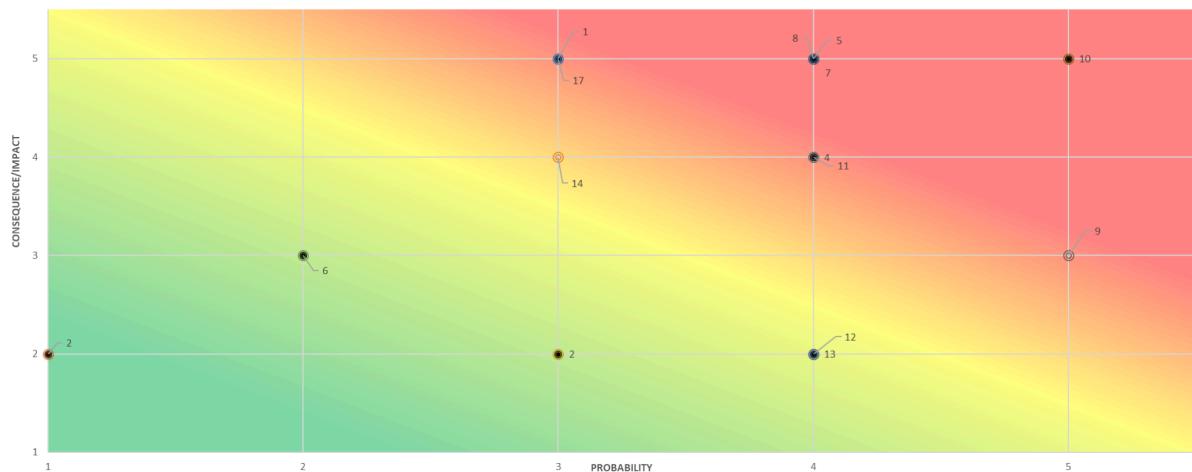
Sprint 1

Risk Analysis - Matrix						
ID	Risk: What can go wrong?	Probability [1-5]	Consequence / Impact [1-5]	Risk score	Resp.	
					Preventive action	Mitigation action
1	Teknisk risiko : Databasen er ikke sikker	4	5	20	(Alle) Generer öste backup af databasen	Gendan databasen fra backup fil, evt ny databasudbyder
2	Teknisk risiko: Brug af bug-rækker	1	2	2	(Alle) Brug nogen fra etablerede kilder og "stable" versioner	Brug af standard udviklerne nogen
3	Teknisk risiko: Exception håndtering	5	4	20	(Alle) Husk at bruge try/catch	Implementer try/catches der håndter exceptionen
4	Teknisk risiko: Konflikter ved merging	3	2	6	(Alle) Alt udvikling skal foregå i branches	Løs konflikterne - ellers revidér commit
5	Teknisk risiko: SQL injection	4	5	20	(Alle) Gør brug af parameters i SQL'en	Gør evt. brug af password hashing
6	Teknisk risiko: Fejl i modelkonverteringer	2	3	6	(Alle) Test på modellen efter konvertering	Konverter Kun modvendelige attributter - omstørkning
7	Teknisk risiko: Fejl i håndtering af bruger privilegier	4	5	20	(Alle) Test af privilegier ved hjælp fra Use Case Diagram	Indstillinge bruger privilegier
8	Ressourceneffektivitet risiko: Mangel på tilstrækkeligt tid eller ressourcer til at implementere etelte systemet	4	5	20	(Alle) Hæv opmærksomhed til at fokusere på det rigtige	Få et et minimums trav
9	Ressourceneffektivitet risiko: Overestimering af teamets evne og tid til at etekomme klarne i hver spids	5	3	15	(Alle) Gendring planlægning og potentielle kommunikation	Manglende opgaver tildeles til næste sprint
10	Ressourceneffektivitet risiko: Brug af for mange ressourcer på "nice to have"	5	5	25	(Alle) Hæv opmærksomhed med at høje fokus på "need to have"	Eventuelt trække nogen af "nice to have"
11	Ressourceneffektivitet risiko: Fraværende øvrigt medlem ved sigdom	4	4	16	(Alle) Hold kommunikation med gruppemedlem	Ondeliger juststødes
12	Ressourceneffektivitet risiko: For meget arbejde ved siden af andre arbejde	4	2	8	(Alle) Brug tiden for at arbejde formidlig	Prøve at få øjeblik vigtig væk
13	Kvalitets risiko: Principper for "Clean Code" overholder ikke forskrift af finandien	4	2	8	(Alle) Lav kodestandarder som alle skal overholde	Reaktionering af code
14	Kvalitets risiko: Ustrukturert og dårligt design af kode	3	4	12	(Alle) Overhold Grænsp. pattern, lav løbende evaluering på koden (code review)	Hæsset
15				0	(Alle)	
16				0	(Alle)	
17	Udskriv krav: Fejlfortolkning af krav og opgaven	3	5	15	(Alle) Tydelig kommunikation og konsensfindning	Spørgsætten
		213				



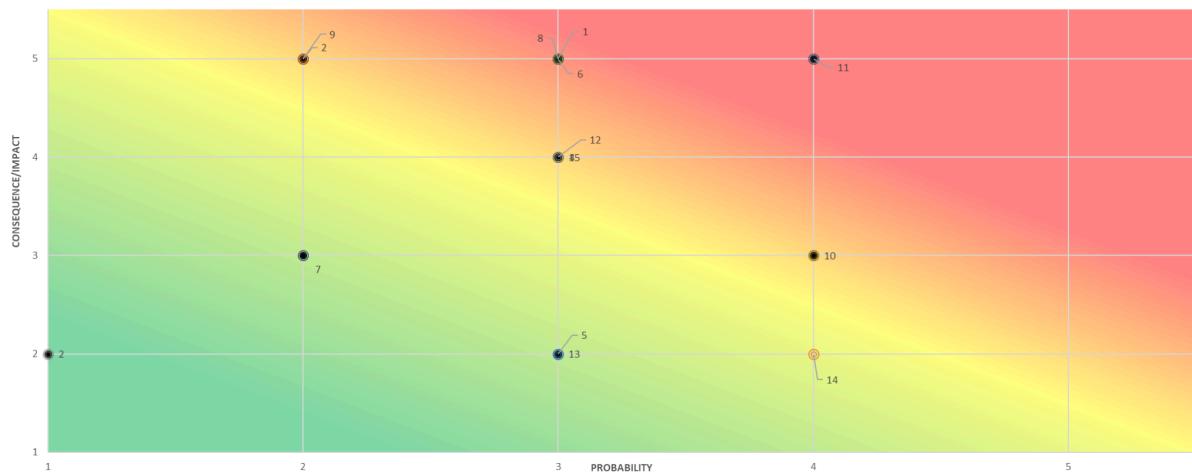
Sprint 2

Risk Analysis - Matrix							
ID	Risk: What can go wrong?	Probability (1-5)	Consequence / Impact (1-5)	Risk score	Resp.	Preventive action	Mitigation action
1	Teknisk risiko : Databasen er utiliggørlig	3	5	15	(Alle)	Gørere ofte backup af databasen	Gendan databasen fra backup fil, evt ny databaseudbyder
2	Teknisk risiko : Brug af nuge-pakker	1	2	2	(Alle)	Brug nuge fra tilsvarende kilder og "stable" versioner	Gør brug af standard/andetekniske nuges
3	Teknisk risiko : Exception håndtering	4	4	16	(Alle)	Hjælp at bugge i try/catch	Implementer try/catches der håndter exceptionen
4	Teknisk risiko : Konflikter ved merge	3	2	6	(Alle)	Alt udskilning skal foregå i branches	Løs konflikterne - eller revert/commit
5	Teknisk risiko : SQL injection	4	5	20	(Alle)	Gør brug af parameters i SQLclient	Gør evt. brug af password hashing
6	Teknisk risiko : Fejl i model-konverteringer	2	3	6	(Alle)	Test på modellen efter konvertering	Konverter Kun nødvendige attributter - omスキv konvertering
7	Teknisk risiko : Fejl i håndtering af brugerprivilegier	4	5	20	(Alle)	Test af privilegier ved hjælp fra Use Case Diagram	Få lavet minimums trav indstift rigtige brugerprivilegier
8	Ressourcemæssig risiko : Mangl. på tilstede/tilgængel. eller ressource til at implementere alle testene systemet	4	5	20	(Alle)	Hjælp hinanden med at fokusere på det rigtigste	Få lavet minimums trav
9	Ressourcemæssig risiko : Overestimering af teamets evne og tid til at efterkomme kravene i hver sprints	5	3	15	(Alle)	Grundig planlægning og idébønde kommunikation	Manglende opgaver tilføjes til næste sprint
10	Ressourcemæssig risiko : Brug af for mange ressourcer på "Nice to have"	5	5	25	(Alle)	Hjælp hinanden med at holde fokus på "need to have"	Eventuelt fratag nogle af "Nice to have"
11	Ressourcemæssig risiko : Fraværende gruppe medlem ved siddom	4	4	16	(Alle)	Hold kommunikation med gruppenmedlem	Omtaleget user stories
12	Ressourcemæssig risiko : For meget arbejde ved siden af skole og arbejde	4	2	8	(Alle)	Brug tiden før/efter arbejde for nytigt	Prøve at få blyttet vægtet væk
13	Kvalitets risiko: Principper for "Clean Code" overholder ikke	4	2	8	(Alle)	Lav kodestandarder som alle skal overholde	Refaktorering af kode
14	Kvalitets risiko: Ustabilitet og dårlig design af koden	3	4	12	(Alle)	Overhold GRASP patterns, lav/bedømme evaluering på koden (code review)	Metoder og dependencier skal tildeles til ansvarlige klasser
15				0	(Alle)		
16				0	(Alle)		
17	Udskriv kav: Fejlforklaring af Case og opgaven	3	5	15	(Alle)	Tydelig kommunikation og kvalitetskræfting	Sjældg. læsen



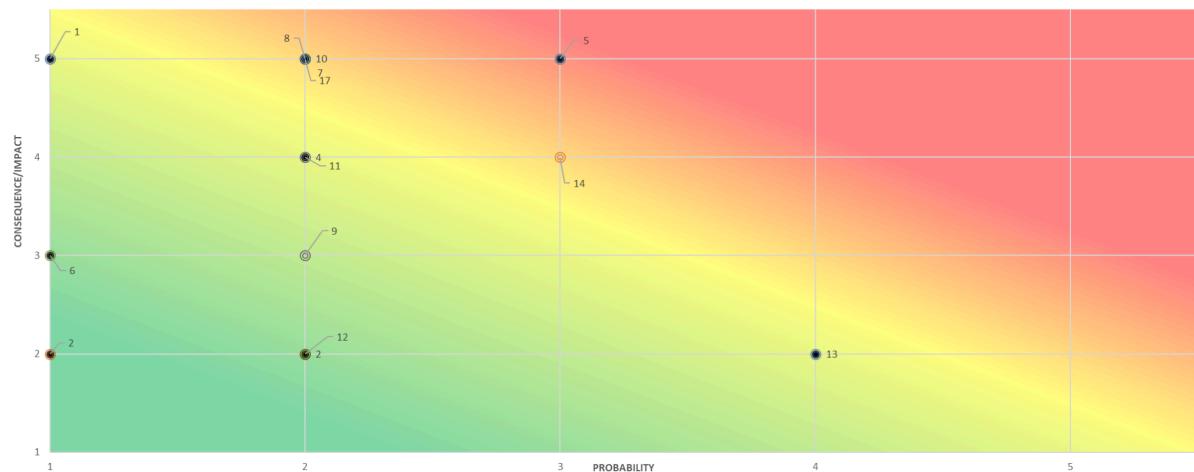
Sprint 3

Risk Analysis - Matrix							
ID	Risk: What can go wrong?	Probability (1-5)	Consequence / impact (1-5)	Risk score	Resp.	Preventive action	Mitigation action
1	Teknisk risiko: Databasen er utilgængelig	2	5	10	(Alle)	Generer ofte backup af databasen	Gendan databasen fra backup fil, evt. ny databaseudløber
2	Teknisk risiko: Brug af nugetpakker	1	2	2	(Alle)	Brug nuge fra troværdige tilde "stable" versioner	Gebrug af standard/andenheds nugets
3	Teknisk risiko: Exception håndtering	3	4	12	(Alle)	Husk at bruge try/catch	Implementeret try catches der hanter exceptionen
4	Teknisk risiko: Konflikter ved inngåing	3	2	6	(Alle)	Alt udvikling skal foregå branches	Løs konflikterne - ejers revert commit
5	Teknisk risiko: SQL injection	3	5	15	(Alle)	Gør brug af parameters i SQLclient	Gør evt. brug af password hashing
6	Teknisk risiko: Fejl i model-konverteringen	2	3	6	(Alle)	Test på modellen efter konvertering	Konverter kun holdværdige attributter - om skriv konvertering
7	Teknisk risiko: Fejl i håndtering af brugeren privilejer	3	5	15	(Alle)	Test af privilejer ved hjælp fra Use Case Diagram	Indstirge bruger privilejer
8	Ressourcenæssig risiko: Mangel på tilstrækkeligt tids eller ressource til at implementere eller teste systemet	2	5	10	(Alle)	Hjælp hinanden med at fokusere på det vigtigste	Få lavet minimumskrav
9	Ressourcenæssig risiko: Overestimation af teamets evne og tid til at efterkomme travene i hver sprints	4	3	12	(Alle)	Grundig planlægning og løbende kommunikation	Manglende opgaver tilføjes til næste sprint
10	Ressourcenæssig risiko: Bug af for mange ressourcer på "Nice to have"	4	5	20	(Alle)	Hjælp hinanden med at holde fokus på "need to have"	Eventuelt fravig af noget af "Nice to have"
11	Ressourcenæssig risiko: Fraærend grupperne medlem ved sigdom	3	4	12	(Alle)	Hold kommunikation med gruppemedlem	Omdeliger user stories
12	Ressourcenæssig risiko: For meget arbejde ved siden af skole, arbejde forskudt af hinanden	3	2	6	(Alle)	Brug tiden før/efter arbejde formidligt	Prøve at få bittet vægten væk
13	Kvalitets risiko: Principper for "Clean Code" overholdes ikke	4	2	8	(Alle)	Lav kodenstandarden som alle skal overholde	Rektorkortering af kode
14	Kvalitets risiko: Ustruktureret og dårligt design af kode	3	4	12	(Alle)	Overhold GRASP patterns, lav løbende evaluering på koden (code review)	Metoder og dependency skal tildeles til ansvarlige klasser
15				0	(Alle)		
16				0	(Alle)		
17	Ulske krav: Fejfortolkning af Case og opgaven	2	5	10	(Alle)	Tydelig kommunikation og kravskræftning	Sporø læren



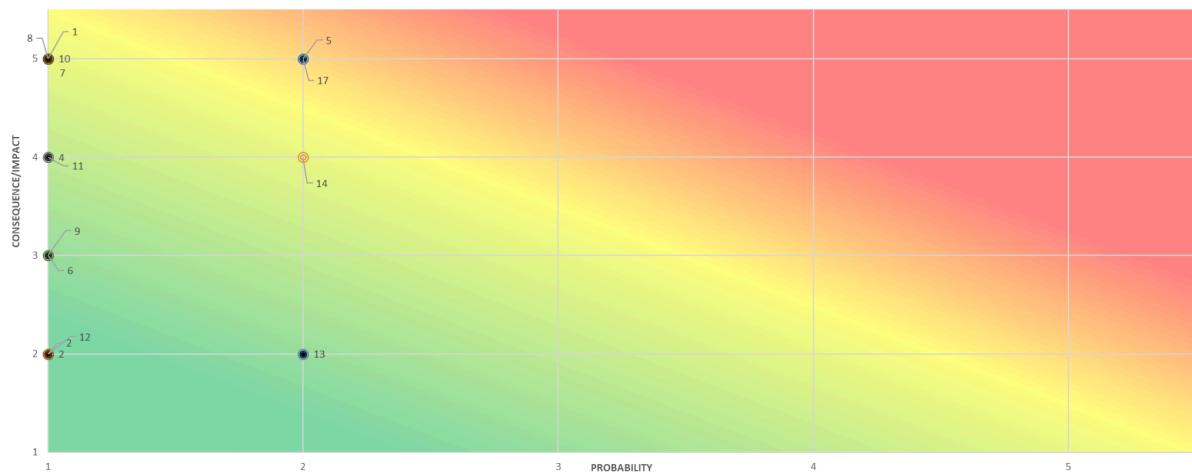
Sprint 4

Risk Analysis - Matrix							
ID	Risk: What can go wrong?	Probability (1-5)	Consequence / Impact (1-5)	Risk score	Resp.	Preventive action	Mitigation action
1	Teknisk risiko : Databasen er u tilgængelig	1	5	5	(Alle)	Generer ofte backup af databasen	Gendan databasen fra backup fil, evt ny databaseudbylder
2	Teknisk risiko : Brug af uregelmæssige kilder	1	2	2	(Alle)	Brug nuget fra troværdige kilder og "stable" versioner	Gøring af standard/standardiserede nugets
3	Teknisk risiko : Acceptationshåndtering	2	4	8	(Alle)	Husk at bruge try/catch	Implementeret try/catches der håndter exceptions
4	Teknisk risiko : Konflikter ved merging	2	2	4	(Alle)	Alt udvikling skal foregå branches	Løs konflikterne - ellers revert commit
5	Teknisk risiko : SQL-injection	3	5	15	(Alle)	Gøring af parameters i SQLclient	Gør evt brug af password hashing
6	Teknisk risiko : Fejl i model-konverteringer	1	3	3	(Alle)	Test på modellen efter konvertering	Konverteret Kun nødvendige attributter - om skriv konvertering
7	Teknisk risiko : Fehåndtering af bugger i initialiser	2	5	10	(Alle)	Test af prælægger ved hjælp fra Use Case Diagram	Indstillinge bruger prælægger
8	Ressourcemasaging risiko : Mængde på tilstrækkelig tid eller ressourcer til at implementere en ny testesystemet	2	5	10	(Alle)	Hjælp hinanden med at fokusere på det vigtigste	Få lavet minimums krav
9	Ressourcemasaging risiko : Overrestimering af tidsen egn sig tid til at efterkomme kravene i hvert sprints	2	3	6	(Alle)	Grundig planlægning og følgende kommunikation	Mængende opgaver til højde til næste sprint
10	Ressourcemasaging risiko : Brug af for mange ressourcer på "Nice to have"	2	5	10	(Alle)	Hjælp hinanden med at holde fokus på "need to have"	Eventuelt fravæg noge af "Nice to have"
11	Ressourcemasaging risiko : Fraværende gruppe medlem ved sygdom	2	4	8	(Alle)	Hold kommunikation med gruppenmedlem	Ombedlig userstories
12	Ressourcemasaging risiko : For meget arbejde ved siden af støtte, arbejde forstørrel af hinanden	2	2	4	(Alle)	Brug tiden før efter arbejde formelt	Prøve at få bittet nogen væk
13	Kvalitets risiko: Principper for "Clean Code" overholdes ikke	4	2	8	(Alle)	Lav kodestandarder som alle skal overholde	Refaktorering af kode
14	Kvalitets risiko: Ustrukturert og dårlig design af kode	3	4	12	(Alle)	Overhold GRASP patterns, Lav løbende evaluering på koden (code review)	Metoder og dependency skal tildeles til ansvarlige klasser
15				0	(Alle)		
16				0	(Alle)		
17	U sikre krav: Fejfortolkning af Case og opgaven	2	5	10	(Alle)	Tidelig kommunikation og konsensfåring	Sjørgen læren



Sprint 5

Risk Analysis - Matrix							
ID	Risk: What can go wrong?	Probability (1-5)	Consequence/ Impact (1-5)	Risk score	Resp.	Preventive action	Mitigation action
1	Teknisk risiko: Database er utilgængelig	1	5	5	(Alle)	Generer ofte backup af databasen	Gendan databasen fra backup fil, evt ny databaseudløber
2	Teknisk risiko: Bug af nutager-pakker	1	2	2	(Alle)	Bug i nogen fra trouzende kilder og "stabile" versioner	Gør bug i standard/andender kendte buggets implementer try catches der håndter exceptionen
3	Teknisk risiko: Execution Validation	1	4	4	(Alle)	Husk at bruge try/catch	
4	Teknisk risiko: Konflikter ved merge	1	2	2	(Alle)	Alt udvikling skal foregå i branches	Løs konflikter - eller etet commit
5	Teknisk risiko: SQL-injection	2	5	10	(Alle)	Gør brug af parameters SQLclient	Gør end brug af password hashing
6	Teknisk risiko: Fejl i model-konverteringer	1	3	3	(Alle)	Test på modellen efter konvertering	Konverter koden nødvendige attributter - om skriv konvertering indstifte brugere præuliæger
7	Teknisk risiko: Fejl i håndtering af brugertilgangsprivilegier	1	5	5	(Alle)	Test af præuliæger ved hjælp fra Use Case Diagram	Få lavet minimums krav
8	Ressourcenæssig risiko: Mangel på instruktører fra eller ressourcer til at implementere eller teste systemet	1	5	5	(Alle)	Hjælp hinanden med at fokusere på det vigtigste	
9	Ressourcenæssig risiko: Overstyring af teamets evne og tid til at få komme kravene i hver spids	1	3	3	(Alle)	Grundlig planlægning og godtende kommunikation	Møgende og gæve tilprojekt næste sprint
10	Ressourcenæssig risiko: Brug af for mange ressourcer på "Nice to have"	1	5	5	(Alle)	Hjælp hinanden med at holde fokus på "need to have"	Evetudt trænning noge af "Nice to have"
11	Ressourcenæssig risiko: Fraærende gruppe medlem ved sidom	1	4	4	(Alle)	Hold kommunikation med gruppenmedlem	Omdeliger userstories
12	Ressourcenæssig risiko: For meget arbejde ved siden af skole, arbejde forskudt af hinanden	1	2	2	(Alle)	Brug tiden før/efter arbejde fuldtigt	Prøve at få blyttet væk
13	Kvalitets risiko: Princippet for "Clean Code" overholdes ikke	2	2	4	(Alle)	Lav kodenstandarde som alle skal overholde	Refaktorering af kode
14	Kvalitets risiko: Ustrukturert og dårligt design af kode	2	4	8	(Alle)	Ovenstående GRASP patterns, Lav løbende evaluering på koden (code review)	Metode og depedency sat tildeles til ansvarlige kasser
15				0	(Alle)		
16				0	(Alle)		
17	Usikre krav, Fejlforklaring af Case og udgaven	2	5	10	(Alle)	Tydelig kommunikation og klar sammenhæng	Så lang læren



Screenshots af program design

LoginPageView

LAW HOUSE

LOG IN

Username (email)

Password

Forgot password? [Need help?](#)

Login

Don't have an account? [Register](#)

[Need help?](#)

CreateUserView

CREATE USER

Firstname Lastname

Email Enter email Confirm email

Phone number Alternative phone

Address

Postal code City

Username Username is the entered email

Password

Enter password Confirm password

Create

[Need help?](#)

LawyersOverview(for clients)

Lawyers

Filter: Speciality Sort by:

Bilal Kinali Associate Lawyer +4522840883	Rasmus Skov Corporate Lawyer +4528401072	Bilal Kinali
		<p>Title: Associate Lawyer City: Fredericia Phone: +4522840883 Email: biki2284@lawhouse.com</p> <p>Specialities: Family Law Immigration Law Personal Injury Law</p>

[Need help?](#)

AdminPage

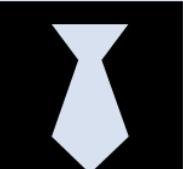
LAW HOUSE

Admin Page Logged in as: Bilal Kinali [sign out](#)

Clients Cases Employees	Forms	Services	Secretaries																
	Create	Create	Create																
	Update	Update																	
	Delete																		
<table border="1"> <thead> <tr> <th>Lawyers</th> <th>Lawyer Title</th> <th>Speciality</th> <th>CaseType</th> </tr> </thead> <tbody> <tr> <td>Create</td> <td>Create</td> <td>Create</td> <td>Create</td> </tr> <tr> <td></td> <td>Update</td> <td>Update</td> <td>Update</td> </tr> <tr> <td></td> <td>Delete</td> <td>Delete</td> <td></td> </tr> </tbody> </table>				Lawyers	Lawyer Title	Speciality	CaseType	Create	Create	Create	Create		Update	Update	Update		Delete	Delete	
Lawyers	Lawyer Title	Speciality	CaseType																
Create	Create	Create	Create																
	Update	Update	Update																
	Delete	Delete																	
Admin Page																			
My Page																			

MyPage(Lawyer)

Bilal Kinali



Title: Associate Lawyer
City: Fredericia
Phone: +4522840883
Email: biki2284@lawhouse.com
Specialities: Family Law
 Immigration Law
 Personal Injury Law

[Edit details](#)

Show All

Cases

CaseID	Title	Description	CreationDate	EndDate	EstimatedHours	TotalPrice	Status
10001	Divorce Pro...	Bilal handle...	24/05/2024 ...	08/06/2024 ...	20	3300	Open
10003	Asylum Ap...	Bilal travels ...	24/05/2024 ...	08/06/2024 ...	150	2079.3599	Open

Show All

Case Services

CaseServiceID	ServiceName	StartDate	PriceType	Units	HoursWorked	TotalPrice	Status
1	Document ...	25/05/2024 ...	Fixed	1	9	1200	Open
2	Court Repre...	25/05/2024 ...	Hourly	3	3	2100	Open
6	Representat...	26/05/2024 ...	Hourly	0	0	0	Open
8	Follow-up ...	26/05/2024 ...	Hourly	0	0	0	Open

AdminCUDLawyer

Name

Firstname

Lastname

Phone number

Enter 8-digit phone number

Email

auto-generated

Address

Postal code City

Title

Select title

Specialities

Add specialities << Remove Added specialities

Hire date

Wednesday, 29 May 2024

Email auto-generated

Password 0000

[Need help?](#)

EmployeesOverview

Filter: Sort by: Search:

Show: 2 Lawyers

LawyerID	Title	F_firstname	Lastname	OpenCases	OpenServices	Specialties	Phone	Email	Zip	ClosedCases	ClosedCaseS	Admin
101	Associate ...	Bilal	Kinali	2	4	3	22840883	biki2284@...	7000	0	1	<input checked="" type="checkbox"/>
104	Corporate ...	Rasmus	Skov	1	4	2	28401072	rask2840...	7000	1	1	<input type="checkbox"/>

[Need help?](#)

LawyersOverview

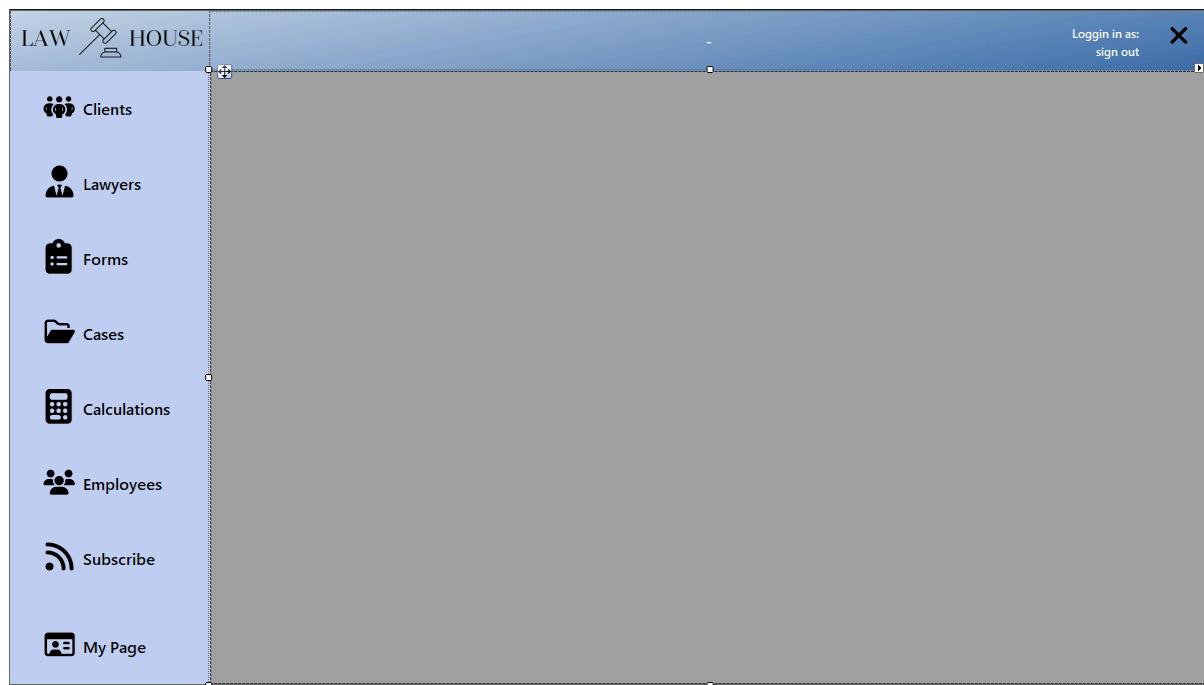
Filter: Sort by: Search:

Show: 2 Lawyers

LawyerID	Title	F_firstname	Lastname	OpenCases	OpenServices	Specialties	Phone	Email	Zip	ClosedCases	ClosedCaseS	Admin
101	Associate ...	Bilal	Kinali	2	4	3	22840883	biki2284@...	7000	0	1	<input checked="" type="checkbox"/>
104	Corporate ...	Rasmus	Skov	1	4	2	28401072	rask2840...	7000	1	1	<input type="checkbox"/>

[Need help?](#)

Forside



My Page klient

LAW HOUSE

My Page

Logged in as: Lucas MacQuarrie sign out

Lawyers

Forms

Calculations

Subscribe

My Page

Firstname: Lucas

Lastname: MacQuarrie

Email: maclucas07@hotmail.com

Address: Korskærevej 16

Postal Code: 7000

City: Fredericia

Subscribed: Yes - Expires in: 89 Days

Phone Numbers:

PhoneNumber	28993856
-------------	----------

Enter 8-digit phone number

Cases

CaseID	Title	EstimatedHour	Status	TotalPrice
10001	Divorce Proc...	20	Open	3300

Formulars

Name	Price
Agreement on Joint Custody	300
Child Testament (One Parent)	250

[Need help?](#)

Update

CaseDetailsView klient

CaseDetailsView

Case information Casenumber: 10001 Casetype: Divorce Title: Divorce Proceedings Description: Bilal handles legal proceedings for the dissolution of marriage for client Lucas MacQuarrie.				Case status Open Estimated End Date 24. maj 2024 Estimated Hours 20 Total hours 12 Total price 3300	Client information Fullname: Lucas MacQuarrie Client ID: 100 Email: maclucas07@hotmail.com Phone number: 28993856 Address: Korskærvej 16 Postal code: 7000 City: Fredericia																																		
Services <table border="1"> <thead> <tr> <th>ServiceName</th> <th>Status</th> <th>TotalPrice</th> <th>Units</th> <th>Hours</th> <th>PriceType</th> <th>StartDate</th> <th>EndDate</th> </tr> </thead> <tbody> <tr> <td>Document Preparation</td> <td>Open</td> <td>1200</td> <td>1</td> <td>9</td> <td>Fixed</td> <td>25-05-2024</td> <td></td> </tr> <tr> <td>Court Representation</td> <td>Open</td> <td>2100</td> <td>3</td> <td>3</td> <td>Hourly</td> <td>25-05-2024</td> <td></td> </tr> <tr> <td>Follow-up Consultations</td> <td>Open</td> <td>0</td> <td>0</td> <td>0</td> <td>Hourly</td> <td>26-05-2024</td> <td></td> </tr> </tbody> </table>								ServiceName	Status	TotalPrice	Units	Hours	PriceType	StartDate	EndDate	Document Preparation	Open	1200	1	9	Fixed	25-05-2024		Court Representation	Open	2100	3	3	Hourly	25-05-2024		Follow-up Consultations	Open	0	0	0	Hourly	26-05-2024	
ServiceName	Status	TotalPrice	Units	Hours	PriceType	StartDate	EndDate																																
Document Preparation	Open	1200	1	9	Fixed	25-05-2024																																	
Court Representation	Open	2100	3	3	Hourly	25-05-2024																																	
Follow-up Consultations	Open	0	0	0	Hourly	26-05-2024																																	
Lawyer information Firstname: Bilal Lastname: Kinali Phone number: 22840883 Email: biki2284@lawhouse.com																																							
Print Details Need help?																																							

CaseDetailsView advokat/sekretær

Case information Casenumber: 10002 Casetype: Franchise Dispute Title: Franchise Agreement Review Description: Rasmus reviews a franchise agreement for client Tristan MacQuarrie, after he decided to open a McDonalds.				Case status Open Estimated End Date 24. maj 2024 Estimated Hours 100 Total hours 0 Total price 1900	Client information Fullname: Tristan MacQuarrie Client ID: 102 Email: mactristan15@hotmail.com Phone number: 60200843 Address: Damvej 13 Postal code: 7000 City: Fredericia																																		
Services <table border="1"> <thead> <tr> <th>ServiceName</th> <th>Status</th> <th>TotalPrice</th> <th>Units</th> <th>Hours</th> <th>PriceType</th> <th>StartDate</th> <th>EndDate</th> </tr> </thead> <tbody> <tr> <td>Initial Franchise Agreeme...</td> <td>Open</td> <td>1900</td> <td>1</td> <td>0</td> <td>Fixed</td> <td>26-05-2024</td> <td></td> </tr> <tr> <td>Follow-up Consultations</td> <td>Open</td> <td>0</td> <td>0</td> <td>0</td> <td>Hourly</td> <td>26-05-2024</td> <td></td> </tr> <tr> <td>Negotiation with Franchi...</td> <td>Open</td> <td>0</td> <td>0</td> <td>0</td> <td>Hourly</td> <td>26-05-2024</td> <td></td> </tr> </tbody> </table>								ServiceName	Status	TotalPrice	Units	Hours	PriceType	StartDate	EndDate	Initial Franchise Agreeme...	Open	1900	1	0	Fixed	26-05-2024		Follow-up Consultations	Open	0	0	0	Hourly	26-05-2024		Negotiation with Franchi...	Open	0	0	0	Hourly	26-05-2024	
ServiceName	Status	TotalPrice	Units	Hours	PriceType	StartDate	EndDate																																
Initial Franchise Agreeme...	Open	1900	1	0	Fixed	26-05-2024																																	
Follow-up Consultations	Open	0	0	0	Hourly	26-05-2024																																	
Negotiation with Franchi...	Open	0	0	0	Hourly	26-05-2024																																	
Lawyer information Firstname: Rasmus Lastname: Skov Phone number: 28401072 Email: rask2840@lawhouse.com																																							
Add service Print Details Close case Need help?																																							

CreateCasePage

The screenshot displays a user interface for creating a new case. It is divided into several sections:

- Case information:** Contains fields for Title, Description, Estimated End Date (set to 29. maj 2024), Estimated Hours, and Casetype.
- Client information:** Contains fields for Firstname, Lastname, Email, and Phone number. It also includes an "Add client" button with a person icon.
- Lawyer information:** Contains fields for Firstname, Lastname, Email, and Phone number. It includes an "Add lawyer" button with a person icon.
- Central controls:** A large "Create case" button at the bottom center and a "Need help?" link in the bottom right corner.

ServiceDetailsForm

■ ServiceDetailsView

Service information

Name	Document Preparation
Description	Prepared all necessary divorce documents, including the petition, financial affidavits, and custody arrangements, and reviewed them with the client for accuracy.
Hours worked	9
Listed Price	1500
Agreed Price	1200

Close service

Tasks	
Date	HoursWorked
25-05-2024 23:33	2
25-05-2024 23:35	4
26-05-2024 00:14	3

Lawyer information

Firstname	Lastname
Bilal	Kinali
Phone number	22840883
Email	biki2284@lawhouse.com

Hours worked **Submit**

[Need help?](#)

AddLawyerView

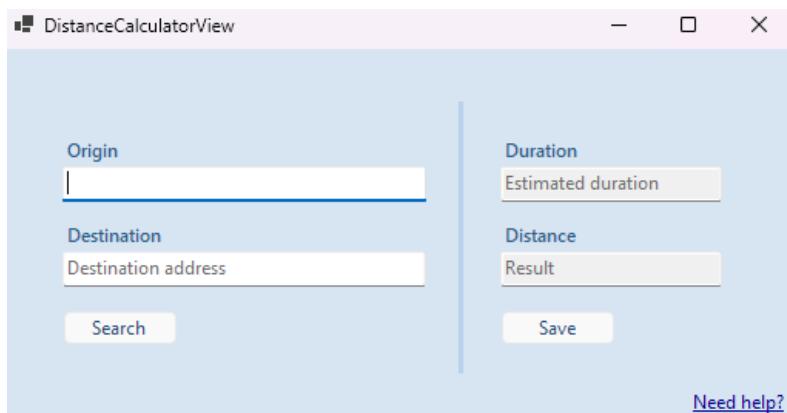
■ AddLawyerView

Search: **Sort by speciality:**

Firstname	Lastname	LawyerTitle	Email	AddressLine	PostalCode	City	Admin	PhoneNumbe
Bilal	Kinali	Associate La...	biki2284@la...	Rygaardsvej ...	7000	Fredericia	<input checked="" type="checkbox"/>	22840883
Rasmus	Skov	Corporate L...	rask2840@la...	Friggsvej 11	7000	Fredericia	<input type="checkbox"/>	28401072

[Need help?](#)

DistanceCalculator



Subscription View



Beregning af ydelse form

Calculate payment on loan

Loan amount:

Annual interest rate:

Payments pr year:

Amount of years:

Calculate **Clear all**

Amount pr payment: 480,16 kr.

Payment pr year: 5.761,90 kr.

Total amount paid: 11.523,81 kr.

[Need help?](#)

Formularer

Forms

Name	Price
Agreement on Joint Cus...	300
Agreement on Child Su...	300
Contract for the Sale of ...	200
Agreement on Spousal ...	350
Mutual Will for Unmarri...	300
Child Testament (One P...	250

Title:
Agreement on Child Support

Description:
This document is used to formalize an agreement regarding the amount and frequency of child support payments from one parent to the other.

Price: 300\$

Buy Form

[Need help?](#)