Syddansk Universitet, Det Tekniske Fakultet, Mærsk-Instituttet.

Civilingeniøruddannelsen i Software Engineering og Diplomingeniøruddannelsen i Softwareteknologi.

Semesterprojekt, 2. Semester: SI2-PRO: Projektrapport.

Projektperiode: Uge 13 2019 – Uge 22 2019.

SI2-PRO - Semesterprojekt - Compassio

Medlemmer

Afleverings dato for projektrapporten: 29. maj 2019

1 Resume

I denne rapport har gruppen valgt at skrive om sagsudredning, som er afgrænset i forhold til den udleverede case opgave, hvor der er tre afgrænsninger. Casen indebærer et objektorienteret samlet system, som sammensættes af EG Team Onlines to eksisterende systemer, Sensum og Sensum Bosted, inden for det specialiserede socialområde. Det opbyggede system kan tilgås fra forskellige brugere uden at det overskrider persondataloven, hvorved gruppen har udarbejdet en problemstilling som lyder følgende;

Kan gruppen udvikle et system, som samler systemerne EG Sensum og Sensum Bosted i et, mens der stadig afgrænses hvad brugeren kan se?

Ud fra denne problemstilling er der blevet udviklet et system, som indebærer de funktionelle krav, samt herefter er blevet arbejdet videre med i form af brugsmønster realisering, som beskrevet i Unified Process. Målene for dette udviklede system er et funktionelt system, som har forbindelse til en relationel database, der er sat op af gruppen. Projektets formål er at fremvise, at gruppens medlemmer kan styre et projekt efter UP sammen med en Scrum Master til at styre det konkrete arbejdsforløb. Udviklingen af softwaresystemet indebærer hele processen fra kravspecifikation til implementering, herunder;

Analyse som indebærer en statisk og en dynamisk del. Den statiske del omhandler domæne model, analyseklassediagram, samt viser selve systemets struktur. Hvor den dynamiske del viser, hvordan dele af systemet interagerer med hinanden. Den dynamiske del indebærer også interaktionsdiagrammer og operations kontrakter.

Hvor en anden del af processen er design, hvilket indebærer struktur og opsætning af softwarearkitektur, hvilket beskriver de forskellige lag i et software system og kommunikationen imellem dem ved hjælp af interfaces. Et underafsnit af design er subsystemdesign, som bruger disse interfaces igennem facader, og databasedesign, som bestemmer hvordan databasen er opbygget på en præsentabel måde, og hvilke midler der er med til at styrke den.

Processen inkluderer også test, hvor gruppen har implementeret fire unit tests, hvor der testes på alt lige fra log ind på systemet til diverse metoder, som er fundamentet for selve systemet. Den optimale test ville være at teste fra start processen i projektet, hvor der typisk ville opstå overraskelser alt efter hvad forventningerne til systemet ville være. Undervejs i iterationerne kom der mere styr på implementationen, der til sidst ville ende ud i en samlet test af hele systemets gennemgang.

Systemet har potentiale til at kunne indeholde flere af persondataforordningens principper, som der er blevet studeret gennem GDPR ordningens hjemmeside[1]. Der er flere årsager til, at systemet ikke indeholder alle principperne. I starten af processen, da gruppen begyndte at udvikle systemet, var fokus på at lavet noget funktionelt, der kunne vises til midtvejsseminaret. Derfor blev der ikke lagt vægt på at overholde persondataforordningen. I de senere implementeringer har dette vist sig at være vigtigere, og derfor er der herefter også lagt mere vægt på netop denne mangel.

[1] GDPR: Fulde dokument. Udgivet af eur-lex.europa.eu. Internetadresse: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679

2 Forord

Hensigten med denne rapport er at give et overblik over de givne brugsmønstre, aktører og de associerede risici, samt overblik over vores samlede system som er sammensat af de to EG Team Online systemer, EG Sensum og Sensum Bosted, hvor der stadig afgrænses hvad brugeren kan se i systemet. Der er fokuseret på sagsudredning som en konceptafprøvning på, at der kan tilgås fra forskellige brugere uden at det overskrider persondataloven.

Rapporten udgør i sammenhæng med systemudvikling og databaseopsætningen, semesterprojektet for 2. semester på uddannelserne civilingeniør i Software Engineering og diplomingeniør i Softwareteknologi på Syddansk Universitet.

Rapporten tiltænkes vejleder og lektor som afleveret delprodukt af semesterprojektet, hvoraf rapporten også udgør dokumentation for udført arbejde. Grundet projektets natur, kan indholdet også være relevant for medstuderende. Undervejs med projektstyringen har visse personer været behjælpelige med at assistere projektgruppen og dens arbejde.

Gruppen vil derfor gerne anerkende:

Afleveringsdato for rapport: 29. maj

Projektdeltagernes underskrifter:

3 Indholdsfortegnelse

4 Læsevejledning

Rapporten giver en beskrivelse af det forløb og de fremgangsmåder, der er blevet udført gennem arbejdsforløbet med projektet. Indholdsfortegnelsen kan bruges som navigationsværktøj, for at få overblik over rapportens indhold og struktur. Hovedteksten af rapporten indeholder en gennemgang af den inkrementelle implementering af de to forskellige iterationer, som gruppen har beskæftiget sig med. Det anbefales at denne hovedtekst læses kronologisk, da strukturen følger den løbende udvikling af programmet. Procesevalueringen indeholder overvejelser og evalueringer på samtlige aspekter ved selve projektforløbet. Procesevalueringen behøves da nødvendigvis ikke læses kronologisk; der kan blot vælges at læse de punkter som synes umiddelbart relevante. Der kan refereres til ordlisten indeholdt i rapporten for klargøring af relevante fagtermer. Rapporten er tiltænkt vejleder og censor for projektet.

5 Redaktionelt

Afsnit	*Ansvarlig*	*Bidrag fra*	*Kontrolleret af*
Resumé	Julie		Alle
Forord	Julie		Alle
Læsevejledning	Julie		Alle
Ordliste	Mads	Morten, Julie	Alle
Indledning	Fælles		Alle
Metode	Peter		Alle
Indledning - hovedtekst	Julie		Alle
Overordnet kravspecifikation	Julie	Mads, Frederik	Alle
Detaljeret kravspecifikation	Julie	Mads	Alle
Analyse	Mads, Frederik	Peter	Alle
Design	Frederik, Peter	Julie, Morten	Alle
GDPR	Mads		Alle
Implementering	Mads, Morten		Alle
Test	Bent	Julie	Alle
Diskussion	Peter		Alle
Konklusion	Frederik		Alle
Perspektivering	Mads		Alle
Læring og refleksion	Julie		Alle
Projektstyring	Peter		Alle
Identifikation, analyse og bearbejdning af problemer	Julie		Alle
Udviklingsprocessen	Julie	Peter	Alle
Formidling og kommunikation	Julie		Alle
Samarbejde i gruppen	Julie		Alle
Samarbejde med vejleder	Julie		Alle
Oversigt over projektets kildekode	Morten		Alle
Brugervejledning	Julie		Alle

6 Ordliste

Dette afsnit vil give en kort forklaring af de fagtermer der benyttes i løbet af rapporten.

Begreb	***Beskrivelse***
ACID	Akronym for atomicitet, konsistens, isolation og holdbarhed
Aktør	En person med en bestemt rolle
Analyseklassediagram	En finpudset udgave af domænemodellen, med flere detaljer.
Brugsmønster	En handling eller række af handlinger.
Burgermenu	Burger-menuen er de tre vandrette streger som ofte ses på mobile websites, systemer med mange menupunkter og i apps.
DAO	Data Access Object. Bruges til at tilgå data.
Database	En server, der kan opbevare data. Selv når programmet ikke kører.
Design mønster	Generel løsning på ofte forekommende problem.
Docker	Værktøj til brug af virtuelle containere.
Domænemodel	En første udgave af et diagram over problemområdet.
E/R diagram	Entity/Relation diagram er et diagram over database design, hvor en entity er objekt af data i database og relationen er sammenhæng mellem dataene.
FXML	Et markdown sprog til at designe brugerflader i Java.
JavaFX	En del af Java, hvori der kan laves applikationer med brugergrænseflade.
MoSCoW	MoSCoW er en prioriterings teknik til at hjælpe med at forstå og styre prioriteter. Bogstaverne står for: Må have, Skulle have, Kunne have og Vil ikke have.
Pakker	En samling af klasser eller andre pakker. Filstruktur i et Java program, på samme måde som i Windows.
Persondataforordningen (GDPR)	Lovgivning omkring opbevaring og behandling af persondata.
Plug-and-play	Et system hvor ved moduler let kan tilføjes eller fjernes, uden at der opstår problemer.
Postkonditioner	Konditioner der skal være sandt efter et brugsmønster er gennemført.
Primær aktør	Aktøren der igangsætter et brugsmønster.
Prækonditioner	Konditioner der skal være sande før et brugsmønster kan igangsættes.
Scrum	Scrum er en agil udviklingsmetode som tager meget fokus på projektledelse.
Sekundær aktør	Aktører der påvirker brugsmønsteret efter det er igangsat.

Softwarearkitektur	Den grundlæggende opbygning af et system og meget lidt detaljeret og giver kun det overordnede overblik over systemet.
SQL-forespørgsel	En anmodning om at hente specifikt data fra databasen.
Tre-lags-arkitektur	Er en arkitekturmodel, som består af tre lag, typisk GUI, logik og persistens. Lagene kan kun snakke nedad.
Tuple	En række af data i en relationel database.
Unit test	En måde at test enkelte dele selvstående kode for, om det virker korrekt.
UP	UP står for Unified Process og er en objektorienteret softwareudviklingsproces.
VPS	Virtual Private Server er en virtuel server.
VUM	VUM står for VoksenUdredningsMetoden. Metodehåndbogen gennemgår sagsbehandlingen i de forskellige trin og understøtter sagsbehandlerens anvendelse af metoden.

7 Indledning

I en lang årrække har EG Team Online været førende på softwareløsninger til det specialiserede socialområde. De har stået for at udvikle systemer til den sociale sektor. Til dette har de udviklet systemerne Sensum Bosted og EG Sensum, som bliver brugt i dag. Da institutionerne og bostederne vokser, er der opstået et behov for en sammenlægning af systemerne, så det samme system kan benyttes på tværs af flere institutioner og bosteder.

Det centrale problem er mangel på sammenhæng mellem de to systemer EG Sensum og Sensum Bosted. Denne mangel skaber førstehånds udfordringer for medarbejdere på både bostederne og i kommunen. I sidste ende vil det også skabe længere ventetid for patienter, der er afhængige af hjælp fra enten kommunen eller et bosted. For at kunne forbedre forståelsen for hvordan systemerne kan samles, og hvilke metoder der benyttes i systemerne, vil gruppen fokusere på sagsudredning som en konceptafprøvning på, at det kan tilgås fra forskellige brugere uden at det overskrider persondataloven, hvorved gruppen har udarbejdet en problemstilling som lyder:

Kan gruppen udvikle et system, som samler systemerne EG Sensum og Sensum Bosted i et, mens der stadig afgrænses hvad brugeren kan se?

I denne sammenhæng har gruppen udarbejdet nogle underspørgsmål for at udforske den overordnede problemstilling:

- Hvilke dele af persondataforordningen og persondataloven vil der blive inkorporeret i løsningsforslaget?
- Hvordan skal data håndteres?

Formålet med dette projekt er dermed at udarbejde et system, der kan løse denne problemstilling. Dette gøres ved først at udarbejde en businesscase og de funktionelle krav. Disse krav arbejdes der efterfølgende videre med i form af brugsmønsterrealisering, som beskrevet i Unified Process (UP). Målene for dette system er et funktionelt program, som har forbindelse til en relationel database. Projektet har til formål at vise, at medlemmerne af gruppen kan styre et projekt efter UP. og herunder også udvikle et software system, som indebærer en relationel database. Udviklingen af softwaresystemet indebærer hele processen fra kravspecifikation til implementering, herunder; analyse, design og test.

8 Metode

I dette afsnit vil gruppens arbejdsmetoder blive beskrevet. Gruppen har valgt at arbejde med både UP og Scrum. UP bestemmer projektforløbet og overordnede arbejdsopgaver. Scrum anvendes til styring af opgaver og fordelingen af disse.

8.1 Scrum

Gruppen har valgt Peter, som Scrum Master, da han har arbejdet med Scrum før. Scrum Masterens opgave er at sørge for, at Scrum ceremonierne bliver holdt. og at holdet ikke bliver forstyrret af eksterne indflydelser. Ifølge Scrum skal der arbejdes ud fra en product backlog. Product backloggen indeholder de opgaver, der skal løses i løbet af projektet. Gruppens product backlog er baseret på den overordnede kravspecifikation., og er derfor også sorteret efter MoSCoW.

Scrum har normalt fire ceremonier, men gruppen kommer kun til at arbejde med tre af dem. Ved Scrum planlægningsmøder besluttes det hvilke opgaver fra product backloggen skal laves i det kommende sprint, og hvor lang tid hver opgaver ca. tager.

Der afholdes Scrum møder i starten af hver arbejdsdag, så der bliver ikke afholdt dagligt, som der ellers normalt gøres i Scrum. Dette er på grund af, at gruppen ikke arbejder fuld tid, og det derfor ikke ville give mening at holde møde, uden at gruppen har udført noget arbejde.

I slutningen af hvert sprint afholdes der retrospektive møder. Ved disse møder diskuterer gruppen hvordan sprintet er gået og eventuelle problemer. Ud fra dette vil der også blive diskuteret forbedringer og ændringer til næste sprint.

Der vil ikke blive afholdt review møder, da gruppen heller ikke har en productowner. Productowner for projektet er en kombination af kravene fra projektkoordinator Lone Borgersen og input fra vejleder Henrik Lykkegaard. De kan dog ikke direkte betegnes som productowner, og det er derfor heller ikke muligt at holde review møder. Det tætteste gruppen kommer på disse er vejledermøder.

8.2 Unified Process

UP er en iterativ arbejdsproces til softwareudvikling. UP dikterer fire faser, som der arbejdes igennem i løbet af udviklingen af et system. Disse fire faser er; inception, elaboration, konstruktion og overdragelse. I dette projekt, vil der kun blive arbejdet med de første to faser på grund af tids- og ressourcebegrænsninger. Hver fase har nogle milepæle, der skal opnås for at færdiggøre fasen. Et andet centralt aspekt af UP er fokusset på at minimere risici. I UP forsøges det at adressere de største risici tidligt i forløbet.

UPs første fase er inceptionsfasen. Denne fase fokuserer på om projektet er muligt at fuldføre og om omkostninger er for store. Her produceres en businesscase, der vurderer om projektet er en god ide fra et forretningssynspunkt, og visionerne for projektet defineres og afgrænses. Kravspecifikationen udarbejdes ud fra brugsmønstre. For dette projekt er disse leverancer samlet i inceptionsdokumentet, som kan ses i internt bilag B.

Den næste fase er elaborationsfasen. Her arbejdes der videre med de mest systemkritiske brugsmønstre. Først analyseres brugsmønstrene og derefter skal der laves et design. Begge repræsenteres ved klasse- og sekvensdiagrammer. Ved udgangen af elaborationsfasen er der lavet grundlæggende funktionalitet og arkitekturdesign, som også er blevet implementeret.

8.3 UP og Scrum kombination

Arbejdsmetoden, som bruges i projektet, er en kombination af UP og Scrum. UP bruges til at styre arbejdsprocessen helt overordnet. UP dikterer i hvilken rækkefølge ting skal laves, hvad der er vigtigst at fokusere på i hvert stadie af projektforløbet. Når elaborationsfasen startes, er der udarbejdet en forretningsanalyse, og de grundlæggende krav er bearbejdet i form af både overordnede og detaljerede brugsmønstre. Brugsmønstrene bruges i elaborationsfasen til at skabe fælles forståelse for systemet, så programmeringen af en prototype kan begynde og klassediagrammer kan laves.

Det specifikke arbejde planlægges ved hjælp af Scrum. Dette betyder at arbejdet inddeles i sprint, som sørger for, at delmål opnås og gruppemedlemmerne ikke drukner i en masse uoverskuelige opgaver. Scrum fungerer godt med UP, da man kan lægge alle arbejdsopgaverne, der skal udføres i UP, ind i en product backlog. I hvert Scrum sprint, som varer cirka 3 uger, kan man så vælge arbejdsopgaver ud fra product backloggen. Dette sikrer, at opgaverne er mere overskuelige, og gør det også nemmere at vurdere, hvor meget gruppen kan nå at lave.

UP og Scrum virker derfor godt sammen, da UP giver en god måde at planlægge det overordnede projektforløb og Scrum inddeler projektet i en masse mindre arbejdsopgaver, hvor man undervejs vurderer hvor meget gruppen kan nå at lave.

8.4 Refleksion af metoderne

Det har virket som om, at der er stort potentiale med UP. Det har været udfordrende at bruge UP i dette projekt, da det har været en læringsproces, hvor undervisningsforløbet ikke altid har passet med projektforløbet. Gruppen har derfor skulle prøve sig lidt frem, men trods dette har det fungeret. Hele arbejdsprocessen omkring brugsmønstrene har gjort det nemmere at afgrænse en ellers meget stor opgave. Scrum har også hjulpet med at fokusere på mindre arbejdsopgaver. Ved at definere mindre arbejdsopgaver, har det været nemmere at delegere opgaver og skabe overblik over, hvor langt projektet er nået.

9 Persondataforordningen

Det nemmeste overblik over persondataforordningen er artikel 5[2]. Artikel 5 beskriver nogle principper, for hvordan man behandler persondata. Der er syv principper, som er følgende:

- Ansvarlighed
- Formålsbegrænsning
- Dataminimering
- Lovlighed, rimelighed, gennemsigtighed
- Rigtighed
- Integritet og fortrolighed
- Opbevaringsbegrænsning

9.1 Ansvarlighed

Ansvarlighedsprincippet handler om, at man kan påvise, at man efterlever persondataforordningen. Dog i dette projekt, er dette ikke noget der har nogen betydning og er derfor ikke videre begrundet.

9.2 Formålsbegrænsning

Princippet om formålsbegrænsning, handler om at begrænse indsamlingen af persondata til specifikke formål. Dataene må således ikke deles mellem flere formål, medmindre der er hjemmel for dette. I denne situation indsamles hjemmelen, hos dem der bruger systemet. De beder borgeren, der snakkes med, om samtykke til at opbevare og behandle deres persondata, og derfor er dette princip opfyldt. Man må heller ikke viderebehandle dataene til andre formål uden at oplyse borgeren om dette.

9.3 Dataminimering

Dataminimering handler om, at når man indsamler data, skal man tænke "need-to-have" og ikke "nice-to-have". Man skal kun indsamle de nødvendige data. I dette system behandles der kun CPR-numre, og der bliver dermed ikke indsamlet unødvendige persondata. Altså er dette princip også opfyldt.

[2] GDPR: De 7 principper. Udgivet af gdpr.dk. Internetadresse: https://gdpr.dk/persondataforordningen/de-7-principper/ - Besøgt d. 07.05.2019 (Internet)

9.4 Lovlighed, rimelighed og gennemsigtighed

Princippet om lovlighed, rimelighed og gennemsigtighed handler om, at persondata bliver behandlet lovligt. Det handler også om, at persondata bliver behandlet rimeligt, hvilket vil sige at dataene bliver behandlet sikkert og ifølge bedste praksis. Gennemsigtighed handler om, at man er klar over for borgerne om, hvordan man behandler dataene, og at det bliver formidlet på et letforståeligt og lettilgængeligt sprog. I dette projekt bliver dataene behandlet lovligt, hvilket vil sige at dataene ikke bliver brugt til ulovlige formål, de bliver til en vis grad behandlet rimeligt, idet der bliver begrænset, så brugere kun kan se data relevante for dem. Dog bliver gennemsigtigheden ikke dækket i dette projekt, da det er udenfor en programmørs ansvar.

9.5 Rigtighed

Rigtighedsprincippet går ud på at man til enhver tid sørger for, at de persondata man behandler, er korrekte. I dette system, bliver der sørget for at når dataene indsættes, så er de korrekte, dog er der ikke nogen ajourføring af dataene efterfølgende.

9.6 Integritet og fortrolighed

Integritet er som nævnt ovenfor, ikke noget der bliver tjekket efter at dataene er blevet indsat. Der er også blevet taget noget hensyn til fortrolighed. Der er så vidt muligt, sørget for at uvedkommende ikke kan få adgang til dataene. Dog er der ikke gået i langdrag med dette. Der er for eksempel ikke krypteret kommunikation til databasen.

9.7 Opbevaringsbegrænsning

Opbevaringsbegrænsning handler om hvor lang tid man skal opbevare data og når deres tales medicinal eller journal data, skal det opbevares i fem år ifølge GDPR. Det er der valgt ikke at tage hensyn til i dette projekt[3].

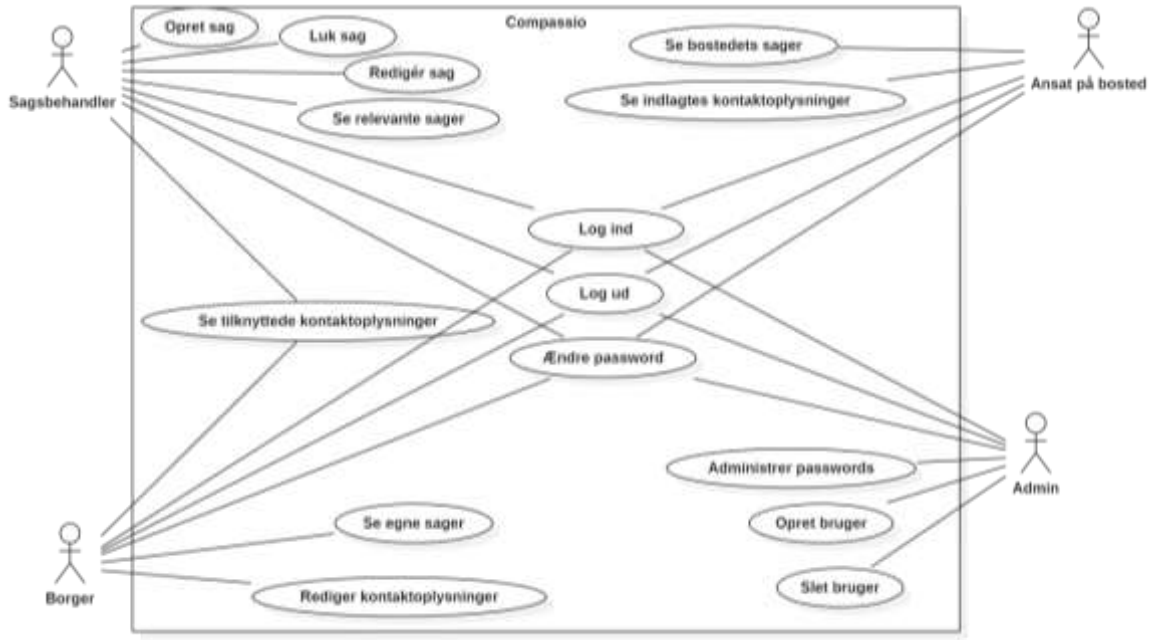
[3] GDPR: Fulde dokument. Udgivet af eur-lex.europa.eu. Internetadresse: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679

10 Hovedtekst

Hovedteksten indeholder herunder beskrivelser af blandt andet overordnet og detaljeret kravspecifikationer, som kommer ind på de forskellige brugsmønstre og aktører, statiske og dynamiske analyser, som indebærer analyseklassediagram, kontrakt og interaktionsdiagram, statiske og dynamiske designvalg, samt database design og softwarearkitektur, implementering af kode, database og til sidst test.

10.1 Overordnet kravspecifikation

I dette projekt er systemet afgrænset fra tre moduler til et. Der er fokus på hele forløbet gennem sagsudredning og samspillet mellem sagsbehandleren og systemet. Dog skal systemet også kunne tilgås af andre, så som administratorer, borger/patienter og ansatte på bostederne. Intentionen med systemet er at det skal være opbygget af moduler, som kan tilføjes og fjernes, som plug-and-play.



Figur 10.1.1 - Overordnet brugsmønstermodel

10.1.1 Aktører og brugsmønstre

Her ses en liste over alle aktører og brugsmønstre for systemet.

Aktører	***Brugsmønstre***
Sagsbehandler	Log ind Log ud Se relevante sager Opret sag Redigér sag Luk sag Ændre password Se tilknyttedes kontaktoplysninger
Ansat	Log ind Log ud Se bostedets sager Ændre password Se indlagtes kontaktoplysninger
Borger/patient	Log ind Log ud Se egne/pårørendes sager Ændre password Se kontaktoplysninger Rediger kontaktoplysninger
Admin	Log ind Log ud Opret bruger Slet bruger Ændre brugerrettigheder Administrer passwords

10.1.2 Kort beskrivelse af aktører

Nedenunder ses en liste med korte beskrivelser af aktørerne.

Aktør	*Beskrivelse*
Sagsbehandler	En sagsbehandler er ansat af kommunen og arbejder med sagerne fra start til slut. Sagsbehandleren kan derfor se alle sine egner sager gennem hele forløbet og interagere med dem.
Ansat	En ansat arbejder på et eller flere af kommunens bosteder. De kan derfor se relevante informationer om de borgere, som er tilknyttet det/de bosteder, hvor den ansatte arbejder.
Borger/patient	En borger/patient er enten de folk som er patient i en sag eller en borger som er pårørende til en patient. De har derfor altid mulighed for at se patientens data, hvis der er givet samtykke eller fuldmagt til det.
Admin	Admin er ansvarlig for at holde styr på alle de forskellige brugere i systemet. Admin kan oprette og slette brugere. De kan også administrere passwords og rettigheder for andre brugere.

10.1.3 Kort beskrivelse af brugsmønstre

Nedenunder ses en liste med korte beskrivelser af brugsmønstrene.

Brugsmønster	***Beskrivelse***
Log ind	Aktøren indtaster brugernavn og password. Herefter logges der ind med en vis rolle på den givne side.
Log ud	Aktøren skal kunne logge ud.
Ændre password	Aktøren skal kunne ændre sit eget password.
Administrere passwords	Admin skal kunne ændre andres password for andre.
Se kontaktoplysninger	Borgeren skal kunne se sine egne kontaktoplysninger.
Opret sag	Sagsbehandleren skal kunne oprette nye sager for en borger.
Rediger sag	Sagsbehandleren skal kunne ændre og tilføje oplysninger til en eksisterende sag.
Luk sag	Sagsbehandleren skal kunne afslutte en sag.
Se kontaktoplysninger	Sagsbehandler/Borger skal kunne se kontaktoplysninger på en borger/om sig selv.
Se bostedets sager	Ansatte skal kunne se de sager som er relevante for bostedet.
Se egne/pårørendes sager	Borgere skal se deres egne sager og sager hvor de er registreret som fuldmægtig.
Rediger kontaktoplysninger	Borgere skal kunne opdatere deres kontaktoplysninger.
Opret bruger	Admin skal kunne oprette brugere til nye medarbejdere, som sagsbehandlere og ansatte på bosteder.
Ændre brugerrettigheder	Administrator skal kunne redigere synligheds rettigheder for brugere.
Slet bruger	Admin skal kunne slette forældede brugere.
Se relevante sager	Sagsbehandleren skal kunne se sine egne sager.

10.1.4 Prioritering

Da der ikke er tid nok til at udarbejde alle brugsmønstrene, er der blevet prioriteret efter hvad der er mest relevant i forhold til afgrænsningen. Da sagsudredning er dette projekts afgrænsning som prototype til sammenlægning af Sensum og Sensum Bosted, vil prioriteringen omhandle den forretningskritiske funktionalitet som sagsbehandlere skal have. Logik som omhandler log ind, er derfor vigtigt, da brugere, som skal bruge systemet, kun skal have funktionalitet, som er tiltænkt dem. Derfor skal der være en måde, hvori der differentieres i brugerne af systemet. MoSCoW forklares i inceptionsdokumentet[4].

Prioritering	***Brugsmønstre***
Must	Log ind Log ud Opret sag Rediger sag Luk sag Se relevante sager Opret bruger
Should	Ændre password Se kontaktoplysninger Rediger kontaktoplysninger Slet bruger Rediger brugerrettigheder
Could	Administrer passwords Se pårørendes sager
Won't	Opret dagbog/indlæg i dagbog Automatisk opdatering af dagbog Opret aktivitetsplan

[4] Inceptionsdokument: Interne bilag, bilag A.

10.2 Detaljeret kravspecifikation

Herunder er et ud af fire udvalgte brugsmønstre, Log ind, Opret sag, Rediger sag og Se relevante sager. Det er de fire mest relevante brugsmønstrer, der er lavet detaljerede beskrivelser for. Opret sag, kan ses herunder, mens resten kan findes i bilag. Igennem hele denne rapport, vil arbejdet med dette brugsmønster være beskrevet. Det samme arbejde er også udført med de tre andre detaljerede brugsmønstre.

Brugsmønster: Opret sag	
ID:	1
Primære aktører:	Sagsbehandler.
Sekundære aktører:	Ingen.
Kort beskrivelse:	Sagsbehandleren skal kunne oprette nye sager for en borger.
Prækonditioner:	● Sagsbehandler skal være logget ind. ● Sagsbehandler skal have adgang til borgeres oplysninger.
Hovedhændelsesforløb:	1. Sagsbehandler opretter sag med borger oplysninger. 2. Hvis flere sagsbehandlere er med i sagen 2.1. Tilføjes de relevante sagsbehandlere. 2.2. Ekstern sagsbehandler modtager sag. 3. Sagsbehandler indtaster oplysninger.
Postkonditioner:	● Sagen er oprettet i systemet. ● Sagen vises hos relevante sagsbehandlere.
Alternative hændelsesforløb:	Ingen.

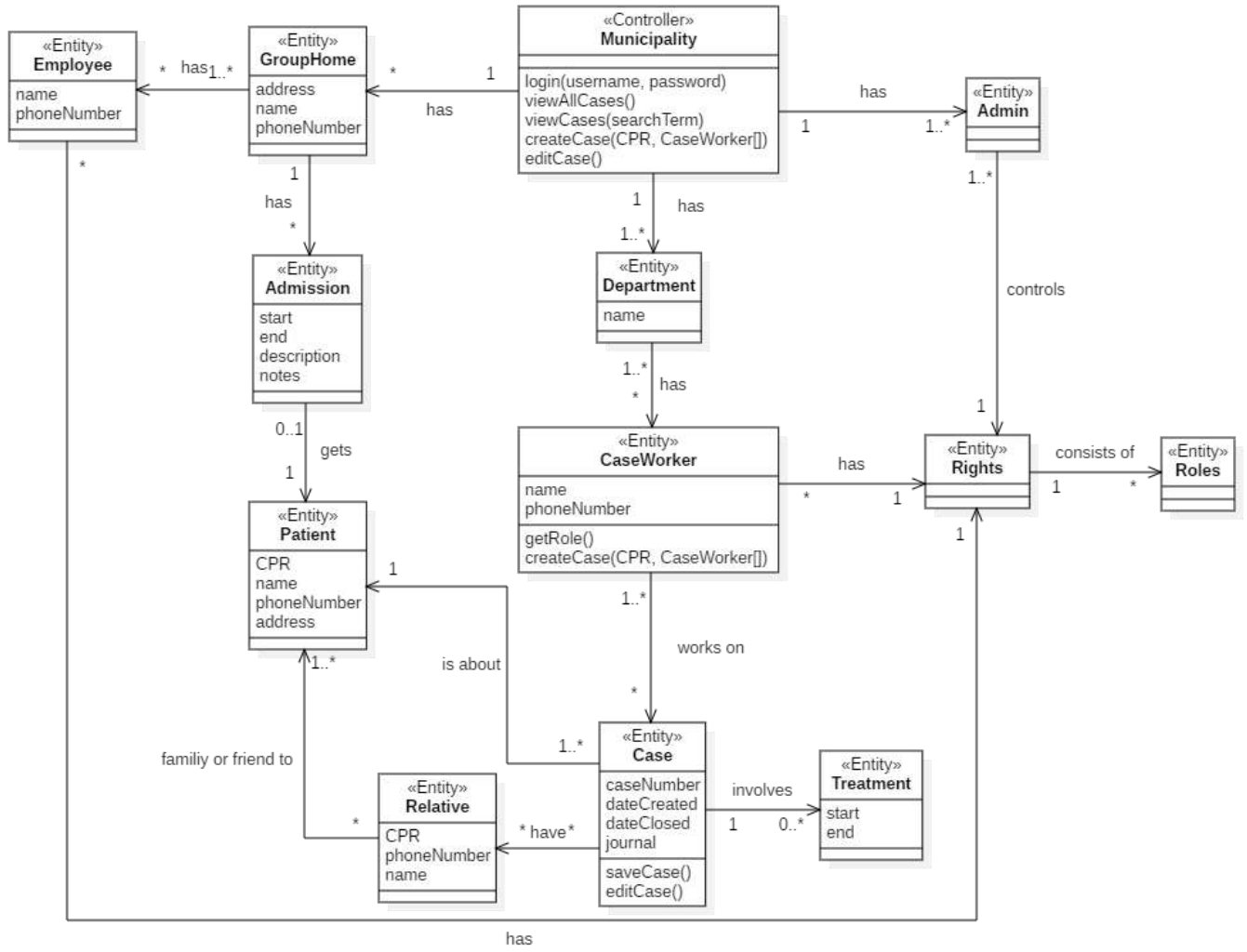
I den detaljerede beskrivelse ovenfor ses at den primære aktør er sagsbehandleren, hvilket vil sige at han er den eneste, der kan igangsætte dette brugsmønster. Der er to prækonditioner for brugsmønsteret. Den første er at sagsbehandleren skal være logget ind, og den anden er at sagsbehandleren skal have adgang til borgerens oplysninger. Begge disse skal være opfyldt for at brugsmønstret kan starte. Det, der skal være opfyldt, når brugsmønstret er gennemført, er at der skal være oprettet en ny sag i systemet, og at sagen bliver vist hos de relevante sagsbehandlere.

10.3 Analyse

Analyse består af to dele, den statiske og den dynamiske. Den statiske del indebærer domæne model og analyseklassediagram og viser systemets struktur. Den dynamiske del indebærer interaktionsdiagrammer og operations kontrakter, og viser hvordan systemets dele interagerer med hinanden indbyrdes.

10.3.1 Statisk analyse

Den statiske side af analysemodellen er analyseklassediagrammet. Analyseklassediagrammet er en viderebearbejdelse af domænemodellen. Analyseklassediagrammet består af analyseklasser, forbindelser mellem analyseklasserne, og eventuelle attributter og funktioner. Analyseklasserne kommer fra analyse af brugsmønstermodellen og domænemodellen.



Figur 10.3.1 – Analyseklassediagram.

Figur 10.3.1 viser analyseklassediagrammet. Det er baseret på domænemodellen, som kan ses i bilag D. Der er taget de samme klasser og herefter er det blevet tilpasset, så nogle klasser er blevet fjernet og der er tilføjet nogle metoder i klasserne efter den dynamiske analyse i afsnit Dynamisk analyse10.3.2. Efter analyseklasserne var blevet fundet, blev de opstillet, og der blev lavet logiske forbindelser mellem dem. Disse forbindelser fremgår også af figur 9.4.1 ovenfor. Herefter er der også blevet indsat UML stereotyper på klasserne. Af figur 9.4.1 fremgår at klassen Municipality er kontrollør, da denne klasse koordinerer handlinger mellem de andre klasser og det også er denne klasse som modtager system hændelser. Resten af klasserne har stereotypen Enhed, da de er enheder i systemet.

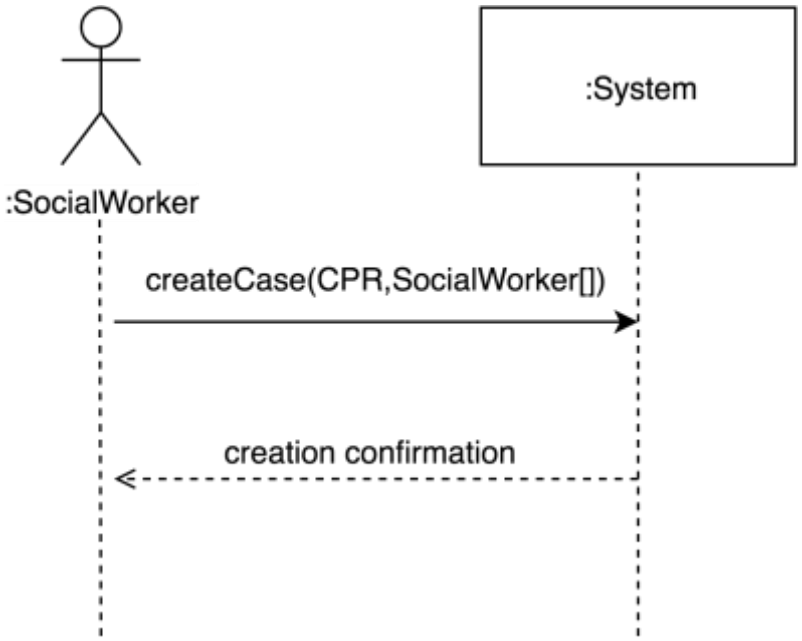
Klasserne har også fået tilføjet funktioner baseret på de detaljerede brugsmønstre. Som kontrollør, så har Municipality-klassen alle de overordnede funktionerne, som kaldes af f.eks. brugergrænsefladen. Den reelle implementering er fordelt ud på de relevante klasser. Her ses det at opret sag er blevet sat på CaseWorker, da det kun er sagsbehandlere, som kan oprette sager.

10.3.2 Dynamisk analyse

Den statiske del af en analysemodel har fokus på struktureringen af systemet, altså indgående klasser og relationerne imellem dem. Den dynamiske del fokuserer på instans-adfærden imellem klasser og relationerne mellem dem. Det vil sige, at den dynamiske del viser, hvordan et brugsmønster bliver udført inde i systemet. En dynamisk analyse indeholder flere dele. Beskrivelsen af adfærden gøres igennem en brugsmønsterrealisering. Der er en brugsmønsterrealisering til hvert brugsmønster.
En brugsmønsterrealisering består af flere dele. Det starter med, at et brugsmønster vælges, hvorefter der bliver lavet et systeminteraktionsdiagram på baggrund af brugsmønsteret. Systeminteraktionsdiagrammet viser de måder, hvorpå en aktør kan interagere med systemet. I denne sammenhæng skal systemet ses som en black box. Dette betyder at system interaktion diagrammet ikke viser andet, end hvordan aktøren interagerer med systemet, og hvad der kommer tilbage på baggrunden af interaktionen.
Næste skridt er at lave en operation kontrakt for de operationer, der er fundet i system interaktion diagrammet. Operationskontrakten kan ses som en formél beskrivelse af ansvaret, som en operation har. Det vil sige, hvilken effekt en operation skal have, og derfor også hvad den skal ændre.
Det tredje skridt er at lave et sekvensdiagram, der både viser operationen, som en aktør laver, og hvordan dette bliver håndteret i systemet. Sekvensdiagram er et white box interaktion diagram, som viser hele processen på baggrund af en operation.
Det sidste skridt er en opdatering af analyseklassediagrammet på baggrund af brugsmønster realiseringerne, så den passer overens med det ansvar de forskellige operationer har. [5]

[5] UML 2 and the Unified Process. Supplements by Lone Borgersen. Udgivet af Lone Borgersen. Sidst opdateret: 02.2017. Internetadresse: https://drive.google.com/open?id=0B3ME4awmGS9ac2tRelJCZ0Z0LWM - Besøgt d. 21.05.2019 (Internet)

I første iteration var der fokus på de must have brugsmønstre fra den tidligere udførte MoSCoW analyse. På baggrund af denne blev der lavet en brugsmønster realisering på baggrund af beskrivelsen ovenfor. Nedenfor ses en brugsmønsterrealisering for "Opret sag".

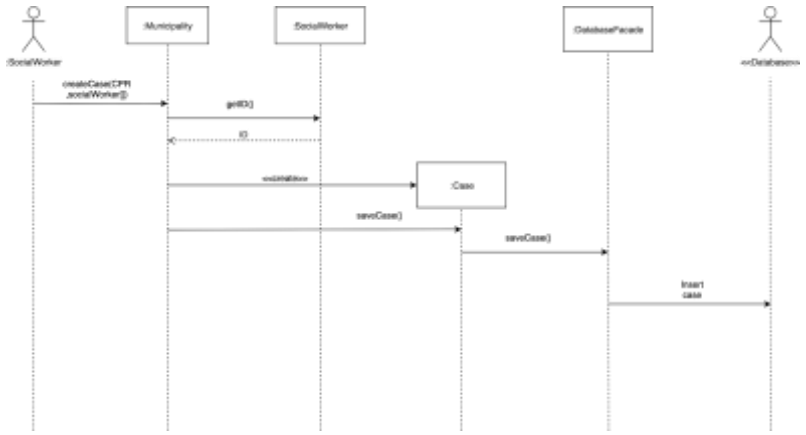


Figur 10.3.2 - Systeminteraktionsdiagram for opret sag.

Første skridt i en brugsmønsterrealisering er, som tidligere beskrevet, et systeminteraktionsdiagram. Dette ses i Figur 10.3.2. Her kan man se at en bruger kan oprette en sag ved at kalde operationen createCase() i systemet. Systemet skal bruge et CPR-nummer og en liste af sagsbehandlere for at udføre metoden. Systemet vil efterfølgende vise en bekræftelse til brugeren.

Kontrakt	
Operation	createCase(CPR, caseWorker[])
Krydsreference	Opret sag
Ansvar	er at oprette en sag omhandlende den givne patient. Hvis der er specificeret flere sagsbehandlere, tilknyttes disse også til sagen. er at udsende en bekræftelse til sagsbehandler med informationer om sagsnummer, patient, tilknyttede sagsbehandlere og oprettelsesdato. Derefter gemmes Case instansen i databasen.
Output	saveCase()
Prækonditioner	• En nuværende caseWorker instans eksisterer.
Postkonditioner	• En sag var oprettet. • Sagen var tilknyttet den nuværende sagsbehandler og eventuelt ekstra sagsbehandlere. • Sagen var tilknyttet patienten med det givne CPR-nummer. • Attributten oprettelsesdato på sagen var sat.

I operationskontrakten specificeres de nærmere detaljer omkring operationen. Udover de parametre operationen tager, så beskriver kontrakten også præ- og postkonditioner. I denne kontrakt ses for eksempel, at der skal eksistere en caseWorker instans før, at man kan udføre operationen. Kontrakten viser også hvilke outputs operationen har. Her menes kald ud af systemet, som ikke er responsen til brugeren. Ved denne kontrakt gemmes sagen i databasen, hvilket er et output.



Figur 10.3.3 - Sekvensdiagram for opret sag

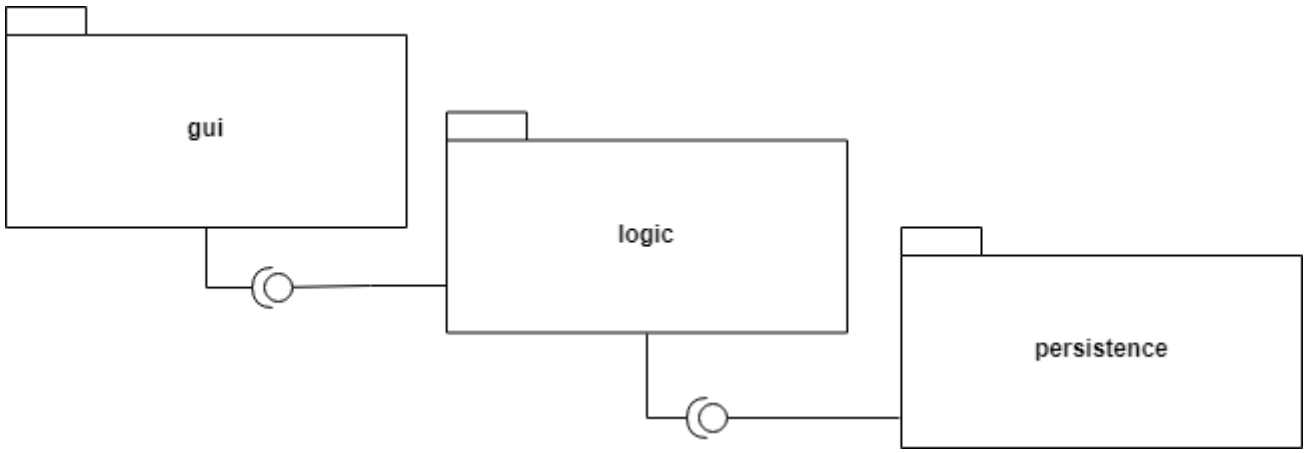
Endelig kan der ud fra kontrakten laves et sekvensdiagram, der beskriver eksekveringsforløbet. Her ses det hvordan systemet først henter brugerens ID for at kunne knytte brugeren til sagen. Herefter bliver sagen oprettet og gemt i databasen.

10.4 Design

Dette afsnit beskriver designovervejelser, beslutninger og resultater for systemets arkitektur, designmodeler og database.

10.4.1 Softwarearkitektur

Softwarearkitektur er fordelingen af klasser i pakker og kommunikationen imellem disse. Igennem arkitekturen kan man opnå gode software kvaliteter. Dette projekts system har en 3-lags arkitektur. En 3-lags arkitektur består, som navnet angiver, af tre lag; brugergrænsefladelaget, logiklaget og persistenslaget. De tre lag kan ses i nedenstående pakkediagram.



Figur 10.4.1 - Tre-lags arkitektur pakkediagram.

Brugergrænsefladelaget håndterer kommunikationen med brugeren. Dette lag håndterer alle input fra brugeren, både museklik og tekstinput. Det er også her, at systemet opdaterer hvad der vises på skærmen baseret på kald til logiklaget. Logiklaget indeholder implementeringer af alle de funktioner, som systemet skal kunne. Persistenslaget håndterer kommunikationen med databasen. Her ligger alle SQL-queries.

De forskellige lag kan kun kommunikere med hinanden igennem et interface. Lagene kan også kun kommunikere med hosliggende lag. Det vil sige, at der ikke er nogen kommunikation imellem brugergrænsefladelaget og persistenslaget uden, at det går igennem logiklaget. Dette gøres for at mindske koblingen af klasserne. En lavere kobling gør det nemmere at ændre i programmet i fremtiden, da en ændring af en klasse kræver færre ændringer i andre klasser.

10.4.2 Subsystemdesign

Som tidligere beskrevet, så indeholder programmet tre subsystemer; GUI, Logic og Persistence. Hvert lag har en facade, som kan kaldes igennem et interface. Facaderne bliver instantieret ved programmets opstart. Under opstarten, får facaderne også kendskab til den underliggende pakkes facade, så der kun bliver instantieret én af hver facade.

Interfacene agerer, som en slags kontrakt, der sikrer at hver pakke kan udføre en række funktioner, som de andre pakker har brug for. De sikrer også, at implantationen af en pakke kan ændres uden at et kald fra andre pakker skal ændres, så længe at de stadig opfylder kravene fra interfacet. Pakkerne kan dermed også let blive skiftet ud. Hvis man for eksempel ikke længere er interesseret i en relationel database, kunne man skifte persistenslaget ud med et, der kan forbinde til en anden slags database. Dette vil ikke have nogen indflydelse på resten af programmet, hvis laget stadig kan udføre de samme funktioner, som er specificeret i interfacet.

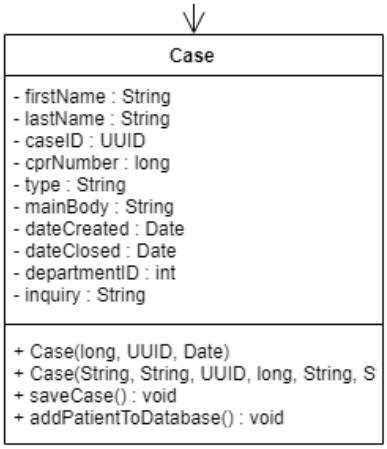
10.4.3 Designmodel

Ifølge UP består en designmodel, ligesom analysemodellen, af to sider; den statiske og den dynamiske side. Den statiske side af designmodellen beskriver et stillestående overblik over implementering. Hver klasse er beskrevet med både attributter, operationer og deres relationer med andre klasser. Dette sker i form af et designklassediagram.

Den dynamiske side af designmodellen viser interaktionerne mellem klasser under eksekveringen. Den kan både indeholde informationer om hvilke parameter en operation tager og hvad der bliver returneret. Fokus af kontrol kan også vises. Den dynamiske side bliver beskrevet i form af sekvensdiagrammer.

10.4.3.1 Statiske side af designmodel

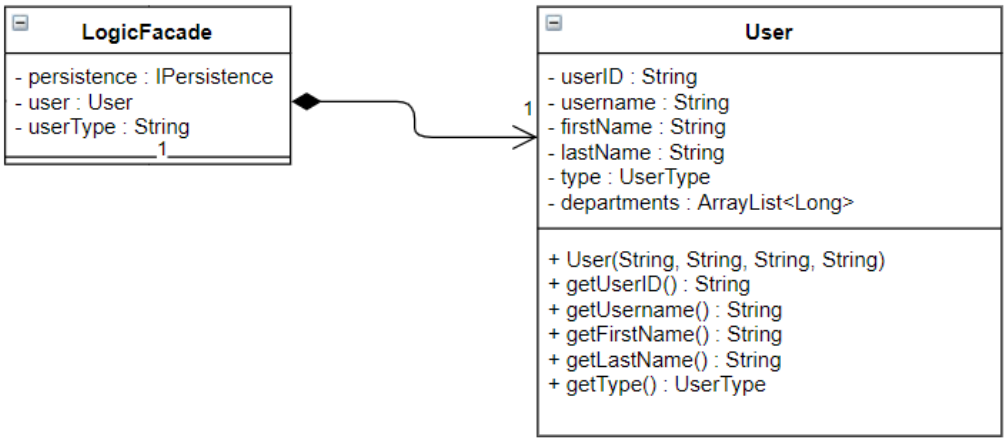
Det statiske aspekt, som er knyttet til den traditionelle forståelse af systembeskrivelse, kan være et objektdiagram. Objektdiagrammer beskriver klassens objekter, såsom under Logic laget hvor Case har flere objekter. Det statiske aspekt kan også beskrives i form af et klassediagram, som indeholder metoder, for eksempel under Logic laget hvor Case har fire metoder.
Design klassediagrammet bliver specificeret på et tilstrækkeligt niveau, hvorefter der kan implementeres uden at skulle tage for mange designvalg imens der implementeres. Design klassediagrammet kan ses i internt bilag I. Et eksempel på en designklasse er "Case".



Figur 10.4.2 - Case under logiklaget.

Klassen "Case" indeholder en masse attributter. Attributter har både en type og en tilgængelighedsmodifikator. Dette gør det muligt for den, der skal implementere at se, hvilke attributter og operationer klassen skal have. Den statiske side af designmodellen kan derfor ses som et slags kode skelet, hvori det eneste, der mangler, er en implementering af den givne opførsel, som klasserne skal have. Når der kigges på "Case", kan det ses, at "Case" f.eks. skal have et fornavn og efternavn på personen, som sagen skal handle om. Her er det blevet bestemt. at de skal være af typen String. Dette gør det klart for alle, der arbejder med implementering, at hvis der skal bruges et fornavn fra en sag på et tidspunkt, så ved alle, at det er af typen String.

Den statiske del af design viser også forholdet mellem forskellige klasser. Her er det især vigtigt at kigge på associationen mellem klasser. Et eksempel på en association, som er blevet specificeret, er mellem klasserne "LogicFacade" og "User":



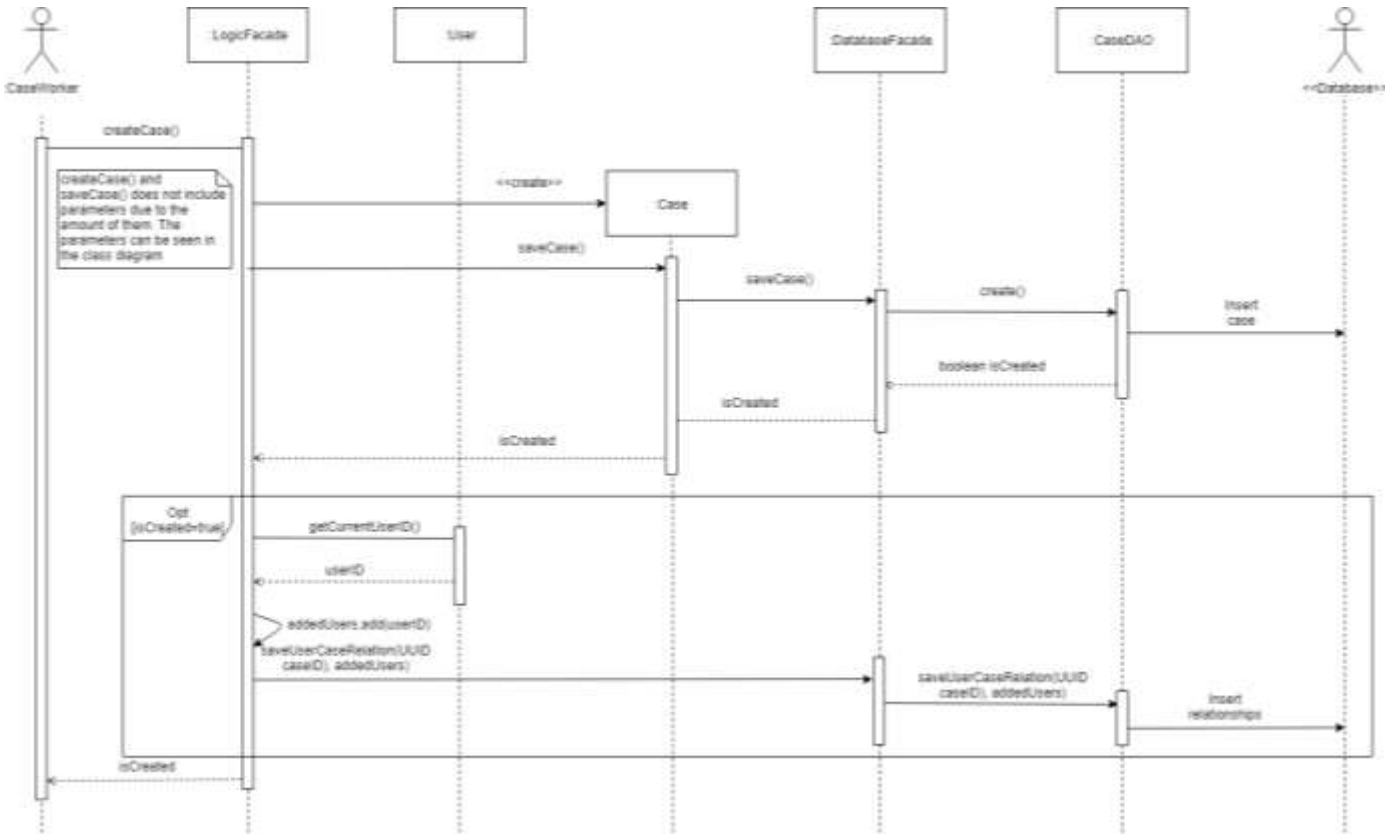
Figur 10.4.3 - Designklasser "LogicFacade" og "User" og forholdet mellem dem i logiklaget.

Denne associering kan ses, som værende gået fra en normal association til en specifik version, som i dette tilfælde er en komposition. Dette betyder at "User" instansen, der er koblet til "LogicFacade", vil forsvinde, når facaden stopper med at eksistere. Der er kun koblet én "User"-instans til facaden ad gangen. Det er ikke relevant at definere multiplicitet af "LogicFacade", der er koblet til "User". Dette skyldes både, at der kun er en instans af "LogicFacade" i hele systemet når det kører, og derudover dikterer en komposition i dette tilfælde, at "User" ikke kan være stærkt koblet til mere end en instans, da "parts" i kompositioner forsvinder når sin "composite" stopper med at eksistere.

Disse klasser kan ikke umiddelbart findes i analysemodellen. Når et analyseklassediagram skal blive til et designklassediagram, er det ikke alle klasser, som bliver overført en til en. For eksempel, kan user ses som en samling af alle mulige typer af brugere. Alt efter hvilken "UserType", som den aktive bruger er, bestemmes der hvad en bruger kan. Designet kan ses som en specifik realisering af analysemodellens mere generelle skitse af hvordan systemet skal realisere funktionaliteten. Den specifikke implementering af designmodellen viser, at alle de typer af arbejdere, der var i analysemodellen, er blevet sat sammen til en "User", hvor det er "UserType", der bestemmer, hvad der er muligt at gøre. Dette realiserer også de rettigheder, som en bruger skal have. Til sidst er nogle analyseklasser blevet til en relation i databasen, og er derfor ikke beskrevet i klassediagrammet.

Et eksempel på dette er "Departments" i analyseklassediagrammet. Den er ikke med i designklassemodellen, da den ikke havde nogen anden funktion end at holde på information om brugere. Derfor gav det bedre mening at gøre den til en relation, hvori en bruger der er koblet til en "Department", som heri er med til at bestemme hvad brugeren kan se og gøre.

10.4.3.2 Dynamiske side af designmodel



Figur 10.4.4 - Sekvensdiagram for opret sag.

Den dynamiske side af design modellen realiseres i sekvensdiagrammer. Sekvensdiagrammer beskriver eksekveringsforløbet i systemet. Ovenover ses sekvensdiagrammet for operationen createCase(). I diagrammet kan det ses, at systemet først opretter et Case objekt, og derefter kalder metoden saveCase() på den. Begge disse metoder indeholder flere parametre, der beskriver alle informationerne omkring en case, men disse er udeladt fra diagrammet, da de optog en masse plads og dermed gjorde diagrammet mindre læsevenligt. Derefter fortsætter eksekveringsforløbet ned igennem persistenslaget, og til sidst gemmes sagen i databasen. Efterfølgende skal sagsbehandleren også knyttes til sagen, da dette er gemt i en anden relation i databasen. Dette gøres ved at hente brugerens ID i User klassen, som derefter tilføjes til listen over de sagsbehandlere som brugeren har indtastet. De bliver derefter alle gemt i databasen som værende knyttet til den givne sag.

Der er flere grunde til, at det kan betale sig at lave et detaljeret sekvensdiagram for et brugsmønster. For det første, viser det, at der allerede er taget designbeslutninger om, hvorledes et brugsmønster skal udfolde sig. Det bliver altså beskrevet, hvilken funktion diverse instanser af klasser skal have ifølge et brugsmønster. Et sekvensdiagram giver et forslag til et specifikt løsningsforslag på, hvordan et brugsmønster kan realiseres. Dette gør det muligt at separere dem som skal implementere koden og dem som laver designet. Ved at have et sekvensdiagram at læne sig op ad, er det kun operationslogikken som mangler fra de operationer, der skal kaldes på forskellige instanser.

10.4.4 Design af persistens

Persistens lagets primære funktioner er at afkoble implementeringen af data adgang fra resten af systemet. Helt grundlæggende gør det nemmere at skifte databasesystemet ud, da implementeringen ikke er tæt koblet med forretningslogikken. For at opnå denne afkobling skal persistens laget implementere et interface, der definerer de operationer, som systemet er afhængigt af for at fungere. I persistens laget vil dette interface blive implementeret af en facade.

Persistens laget består yderligere af Data Access Objects. DAO design mønstret hjælper med at adskille kald til databasen fra resten af persistens laget, og dermed skjuler kompleksiteten bag lagringen af dataene fra resten af laget. Dermed opnås stor fleksibilitet til at gøre behandling af dataene fra en datakilde uafhængig af implementeringen af den datakilde. Design mønstret er med til at skabe yderligere abstraktion, indkapsling og afkobling. Design mønstret er også nemmere at udføre unit-tests på end en monolitisk datakilde klasse.

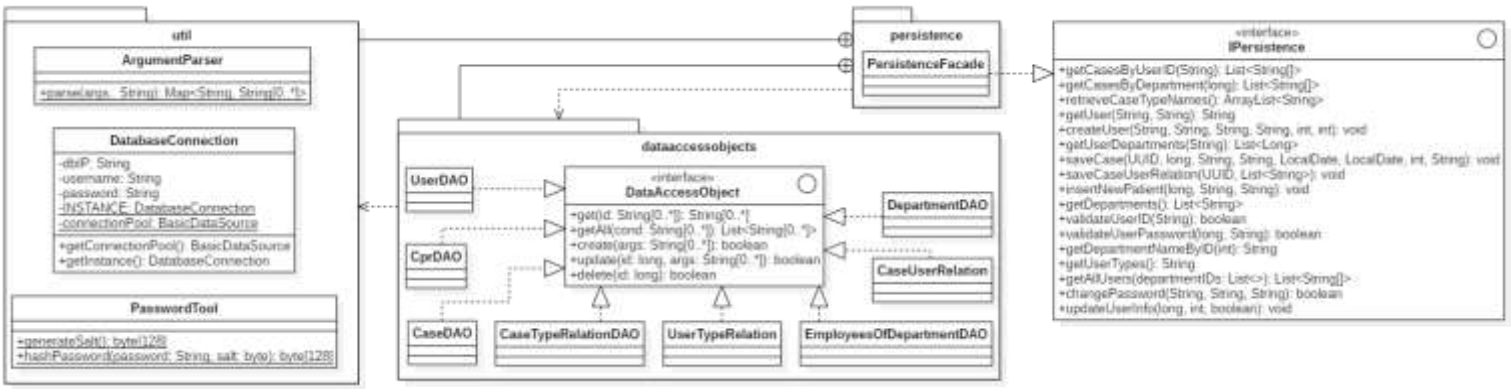
Ulempen ved DAO design mønstret er at det skaber ekstra netværkstrafik og ekstraarbejde for den underliggende datakilde.

Overordnet består Data Access Object design mønstret af følgende klasser og interfaces:

- DAO Interface - Sikrer at alle DAO klasser understøtter alle de fundamentale operationer for manipulation af data. Herunder oprettelse af nye rækker, aflæsning af eksisterende rækker, redigering af eksisterende rækker, og sletning af eksisterende rækker.
- DAO klasse - Konkret implementering af DAO interfacet. Klassen henter data fra en datakilde.
- Model objekt - Objekt med metoder til at gemme data hentet fra DAO klassen.

Alle Data Access Objekterne i systemet er tabel-centrale og følger dermed tabellerne i databasen 1:1.

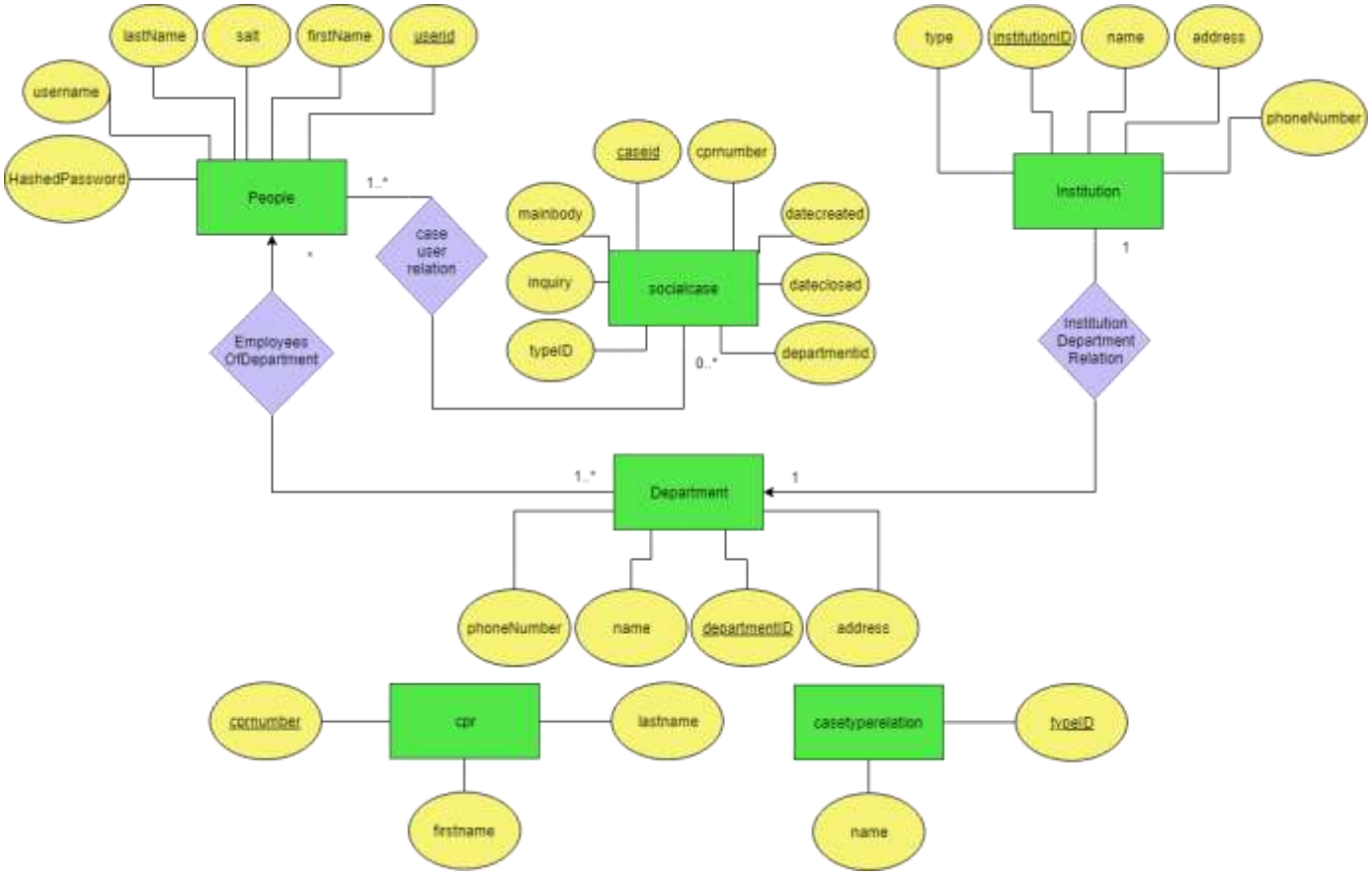
I dette system resulterer kombinationen af ovenstående designvalg i følgende arkitektur for persistens laget:



Figur 10.4.5 - Klassediagram for persistenslaget.

10.4.5 Databasedesign

Systemet benytter en PostgreSQL database. PostgreSQL er et relationelt databasemanagement system (RDBMS). Relationelle databaser benytter rækker og kolonner til at organisere data. Rækker kaldes også tupler og kolonner kaldes attributter. En relation repræsenterer en enhed, eg. en bruger. Hver tuple er dermed en specifik enhed og attributterne repræsenterer aspekter af eller informationer omkring den specifikke enhed.



Figur 10.4.6 - E/R diagram over databasen.

Ovenstående figur er et E/R diagram over systemets database. Databasen er struktureret efter 3. normal form. Dette betyder, at databasen skal overholde visse principper. Det første princip er, at der ikke må være duplikerede tupler. Dette sikres ved, at alle relationer har en eller flere primære nøgler. I E/R diagrammet er attributter, som også er primære nøgler, vist ved at være understreget.

Det andet princip er, at celler skal være atomiske. Det vil sige, at celler ikke må indeholde mere end en værdi. For at forhindre dette laves der nogle steder relationer, der beskriver sammenhængen mellem to andre relationer. Et eksempel på dette er caseuserrelation, der beskriver hvilke sagsbehandlere der er tilknyttet til en sag. En anden måde at lave dette på ville være at have en liste over tilknyttede sagsbehandlere, som en attribut af socialcase, men det ville ikke være atomisk.

Det tredje princip er, at der ikke må være delvise afhængigheder. En afhængighed er, når en attribut kan afledes fra en anden attribut. Delvise afhængigheder er, når en attribut kan afledes fra færre end alle de primære attributter i en relation. Hvis cpr og socialcase var en relation, hvor cpr og caseid begge var primære nøgler, så ville man for eksempel kunne aflede typeID ud fra caseid uden at kigge på hvad cprnumber er. Grunden til at delvise afhængigheder bør undgås er, at de gør det lettere at slette data, som ikke ønskes slettet.

Det sidste princip er, at der ikke må være transitive afhængigheder. Dette betyder, at attributter ikke må være afhængige af andre attributter end primære nøgler. Et eksempel på dette er casetype relationen, som godt bare kunne inkluderes i socialcase relationen, men så ville name være afhængigt af typeID, som ikke er en primær nøgle. Denne adskillelse sikrer også at en type ikke slettes, hvis alle socialcase enheder af en given type slettes.

10.5 Implementering

Dette afsnit vil beskrive de overvejelser og resultater der er opstået i forbindelse med implementering af design.

10.5.1 Kode

Systemet er implementeret i Java version 8. Denne version er valgt da GUI biblioteket, JavaFX, er blevet fjernet i Java version 11, som er den nyeste version. Java er et simpelt kodesprog, der tager den objektorienterede tilgang til programmering. Java tilbyder mange features og et nemt system at lave brugergrænseflade i. I projektet er LogicFacade klassen meget interessant, her er nemlig implementeret data afgrænsningen, som er fokus i projektet. Dataafgrænsningen omhandler, at brugere i systemet ikke må kunne se noget, som de ikke har adgang til.

```
@Override
public ArrayList < Case > getCases() {
   ArrayList < Case > response = new ArrayList < > ();
   if (this.user.getUserType() == this.userType.get(
         "CASEWORKER")) {
      ArrayList < String[] > cases = persistence.getCasesByUserID(
         user.getUserID());
      while (cases.size() > 0) {
         String[] singleCase = cases.remove(cases.size() - 1);
         if (singleCase[7] != null) {
            response.add(new Case(singleCase[0], singleCase[1],
               UUID.fromString(singleCase[2]), Long
               .parseLong(singleCase[3]),
               singleCase[4], singleCase[5], LocalDate.parse(
                  singleCase[6]),
               LocalDate.parse(singleCase[7]), Integer
               .parseInt(singleCase[8]), singleCase[9]));
         } else {
            response.add(new Case(singleCase[0], singleCase[1],
               UUID.fromString(singleCase[2]), Long
               .parseLong(singleCase[3]),
               singleCase[4], singleCase[5], LocalDate.parse(
                  singleCase[6]),
               null, Integer.parseInt(singleCase[8]),
               singleCase[9]));
         }
      }
   } else if (this.user.getUserType() == this.userType.get(
         "SOCIALWORKER")) {
      ArrayList < Long > departments = this.user.getDepartments();
      ArrayList < String[] > cases = new ArrayList < > ();
      departments.forEach(d - > {
         cases.addAll(persistence.getCasesByDepartment(d));
      });
      while (cases.size() > 0) {
         String[] singleCase = cases.remove(cases.size() - 1);
         response.add(new Case(singleCase[0], singleCase[1],
            UUID.fromString(singleCase[2]), Long.parseLong(
               singleCase[3]),
            singleCase[4], null, null, null, Integer
            .parseInt(singleCase[5]), singleCase[6]));
      }
   }
   return response;
}
```

Figur 10.5.1 - Implementering af getCases i LogicFacade.

Her ses implementeringen af metoden getCases. Det er i denne metode dataafgrænsningen foregår. Som det fremgår af Figur 10.5.1, så initialiserer metoden først en tom liste af Case. Dernæst tjekker den hvilken

brugertypen for den bruger, som er logget ind. Hvis brugeren er en sagsbehandler (Caseworker), så henter programmet alle sager, hvor sagsbehandleren er tilknyttet. Hvis brugeren derimod er af typen social medarbejder(Socialworker), så henter programmet alle sager, som er tilknyttet de afdelinger, som medarbejderen er tilknyttet. Desuden, så er det ikke hele sagen, som en social medarbejder kan se, men beskrivelsen af den ydelse, der ønskes fra bostedet.
De to essentielle metoder her er "getCasesByUserID" og "getCasesByDepartment". I disse metoder ligger der to næsten ens SQL-forespørgsler. De ser således ud:

```
SELECT * FROM SocialCase NATURAL JOIN CaseUserRelation NATURAL JOIN CPR NATURAL
JOIN CaseTypeRelation WHERE userID=(?)
```

Figur 10.5.2 - SQL-forespørgsel til getCasesByUserID.

```
SELECT * FROM SocialCase NATURAL JOIN CPR NATURAL JOIN CaseTypeRelation WHERE
departmentID=(?)
```

Figur 10.5.3 - SQL-forespørgsel til getCasesByUserID.

Disse to forespørgsler, sørger for at hente de relevante sager og ikke mere. Derved er der altid kun de data lokalt på computeren, som brugeren må se. Dette er den væsentlige del i dataafgrænsningen. En anden vigtig beslutning er at systemet er opbygget som tre-lags-arkitektur. Imellem de tre lag er der brugt interfaces mellem dem. Dette gør at det er nemt at skifte lagene ud, da interfacet siger hvilke metoder der skal kunne tilgås i det given lag. Eksempel på interface mellem GUI og logik lag:

```
public interface ILogic {

   public void injectPersistence(IPersistence PersistenceLayer);

   public ArrayList < Case > getCases();
   public ArrayList < String > retrieveCaseTypes();
   public boolean login(String username, String password);
   public void logout();
   public boolean createCase(String firstName, String lastName, long cprNumber,
      String type, String mainBody,
      LocalDate dateCreated, LocalDate dateClosed, int departmentID,
      String inquiry, ArrayList < String > socialWorkers);

   public ArrayList < String > getDepartmentInfo();
   public boolean checkUserID(String userID);
   public boolean checkUserPassword(String password);
   public String getUserID();
   public String getUserName();
   public String getUserType();
   public String[] getUserTypes();
   public ArrayList < UserInfo > getAllUsers();
   public String getDepartmentNameById(int departmentId);
   public Boolean changePassword(String newPassword, String oldPassword);
   public void updateUserState(long userID, String newRole,
      boolean newInactiveState);
   public void createUser(String firstName, String lastName, String username,
      String password, String type, int departmentid);
}
```

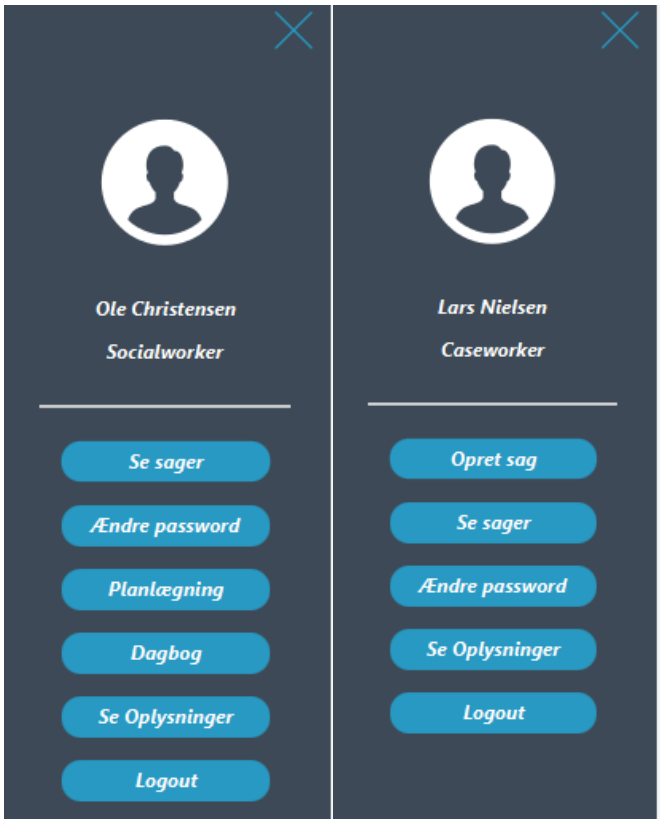
Figur 10.5.4 - Interface for logiklaget

Ved hjælp af denne klasse, ved GUI laget hvilke metoder det kan tilgå i logiklaget. Og så kan implementeringen af logik udskiftes efter behov uden at påvirke GUI laget.

10.5.2 Brugergrænseflade

Systemets brugergrænseflade er sat op efter hvem brugeren, der logger ind, er, samt hvilke rettigheder brugeren har. Der er opsat en menu bar, som indeholder diverse dynamiske menupunkter, og viser, alt efter brugerens rettigheder, menupunkter med relevante funktioner. Der er tilføjet menupunkter, som holder sig inden for projektets afgrænsning, samt menupunkter, der lægger op til funktions tilføjelse. Der er derfor mulighed for at kunne tilføje moduler med ønskede funktioner. Tilføjes flere moduler ville funktionerne fra disse moduler indsættes som nye knapper i menu baren. En anden måde den dynamiske del i brugergrænsefladen kan opbygges på, er ved at knapperne i menu baren ikke refererer funktioner fra forskellige moduler, men derimod refererer de forskellige moduler. Knapperne ville således indlæse en skærm med overblik over de forskellige funktioner indenfor for det givne modul. Dette ville gøre det nemmere at se funktionerne inden for et modul end, når alle funktionerne er i en lang liste. Denne løsning skaber også bedre brugervenlighed i et større program. Dog da programmet lige nu kun inkluderer et modul, og ikke er større er dette ekstra trin ikke implementeret. Det gør også programmet mere simpelt.

Måden, som dette dynamiske aspekt fungerer på, er ved at alle knapperne bliver oprettet, i det man logger ind, og så alt efter hvilken type bruger der logger ind, tilføjes de korrekte knapper til menu baren. Hver knap bliver så opsat således at når den klikkes, loader den en bestemt FXML-fil med den tilknyttede kontroller, som repræsenterer den givne funktion. Fremadrettet er det nemt at tilføje flere funktioner, da der bare skal oprettes en knap, tilføje knappen til de korrekte roller og implementere funktionaliteten. Til brugergrænsefladen er der også tilføjet genvejstaster, for at øge brugervenligheden for mere erfarne brugere, så de kan arbejde hurtigere.



Figur 10.5.5 - (Venstre) Menu bar for en socialarbejder; (Højre) Menu bar for en sagsbehandler

Ovenfor ses menu baren for to brugere med forskellige roller. Den ene med rollen socialarbejder og en med rollen sagsbehandler. Det ses, at socialarbejderen har adgang til nogle funktioner, mens sagsbehandleren har adgang til nogle andre. Det kan dog også ses, at nogle funktioner har begge roller adgang til. Dette er det dynamiske aspekt af brugergrænsefladen.

10.5.3 Databaseopsætning

Databasen er konfigureret på en VPS hostet på DigitalOcean[6]. Serveren kører Ubuntu version 18.04 med en instans af Docker version 18.09.2. Docker instansen kører et PostgreSQL image. En distribueret løsning er valgt for at gemme data centralt.

PostgreSQL er blevet valgt på baggrund af dens ACID-kompatibilitet. MySQL et lignende RDBMS er ikke ACID kompatibelt, og blev derfor valgt fra. ACID-kompatibilitet sikrer dataintegriteten i tilfælde af fejl. Evt. strømsvigt og netværksfejl vil ikke ramme dataene i databasen. Dette er vigtigt for dette system da der behandles private oplysninger. Kompatibilitet hjælper med at sikre privatpersonernes sikkerhed og privatliv, da risikoen for forkerte eller manglende informationer minimeres.

For at en database kan være ACID kompatibel skal den overholde følgende regler:

- Atomicitet: Transaktioner skal være atomiske. Alt eller intet.
- Konsistens: Data skal overholde databasens regler, altså være valid.
- Isolation: Samtidige transaktioner må ikke påvirke hinanden. Illusion om sekventiel eksekvering.
- Holdbarhed: Data er gemt efter en afsluttet transaktion.

10.5.4 Connection Pooling

Oprettelse af forbindelse til databasen er en relativt dyr handling og langsom proces. Systemet er voldsomt afhængigt af forbindelsen til databasen, og systemet opretter ofte forbindelse for at hente eller opdatere data. I første iteration af implementering med database oprettede metoderne i persistensen individuelt en forbindelse til databasen. Dette havde den betydning, at diverse operationer i systemet ikke var så responsivt som ønsket. Dermed blev der i anden iteration af implementeringen arbejdet med at implementere connection pooling. Med connection pooling kan forbindelsen bibeholdes og genbruges flere gange på tværs af systemet. Dette resulterer i en besparelse af ressourcer og overordnet bedre ydeevne. Connection pooling tillader også nemmere vedligeholdelse af database relaterede metoder, da forbindelserne oprettes et sted frem for i alle metoder.

I første omgang blev connection pooling implementeret med en klasse gruppen selv implementerede. Dog stod det hurtigt klart at denne implementering indeholdt en række fejl og mangler. Fordi prepared statements knyttes til en connection opstod der fejl. Derfor blev det besluttet at benytte et tredjeparts bibliotek. Det blev besluttet at benytte "Apache Commons Pool", da denne understøtter caching af prepared statements i forbindelse med connection pooling. Dette sikrer at prepared statements kan genbruges på flere connections.

[6] https://www.digitalocean.com/

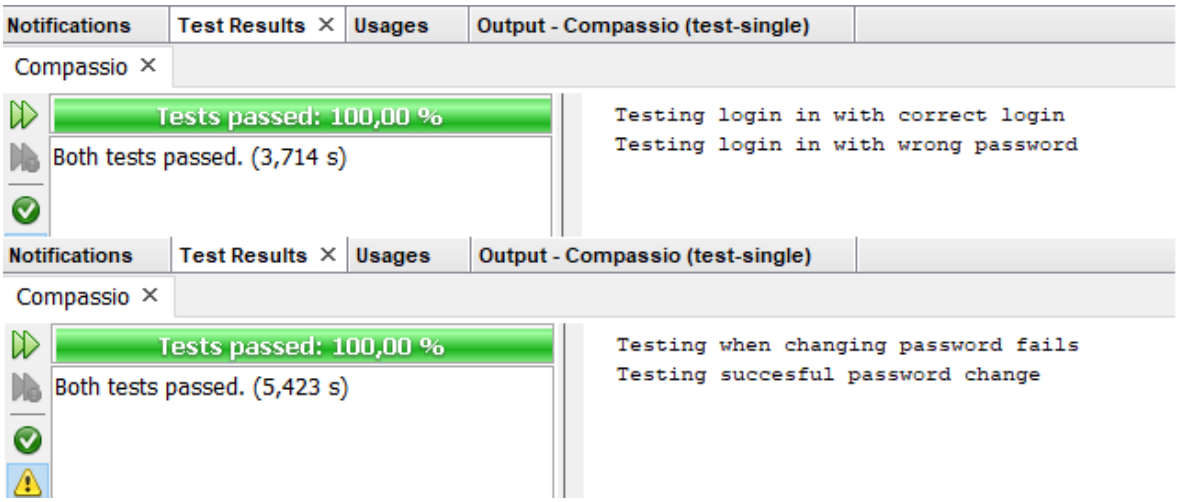
10.6 Test

I computerprogrammering er enhedsprøvning altså unit tests en software testmetode, hvor individuelle enheder benytter kildekode, lavet af et eller flere computerprogramsmoduler sammen med tilhørende kontroldata, hvor brugsprocedurer og driftsprocedurer testes for at afgøre, om de er egnede til brug i systemet. Der er under udvikling af systemet løbende blevet kørt komponents tests, hvor de enkelte enheder samles til komponenter, for at teste at de kommunikere korrekt med hinanden. Men også for at sikre at de implementerede interfaces fungere hensigtsmæssigt. Systemet som helhed er også løbende blevet brugertestet af gruppen, her har vi testet systemet for at sikre, at der ikke er uhensigtsmæssige interaktioner.

I systemet testes følgende dele af koden:

- login
- changePassword
- getCases
- saveCases

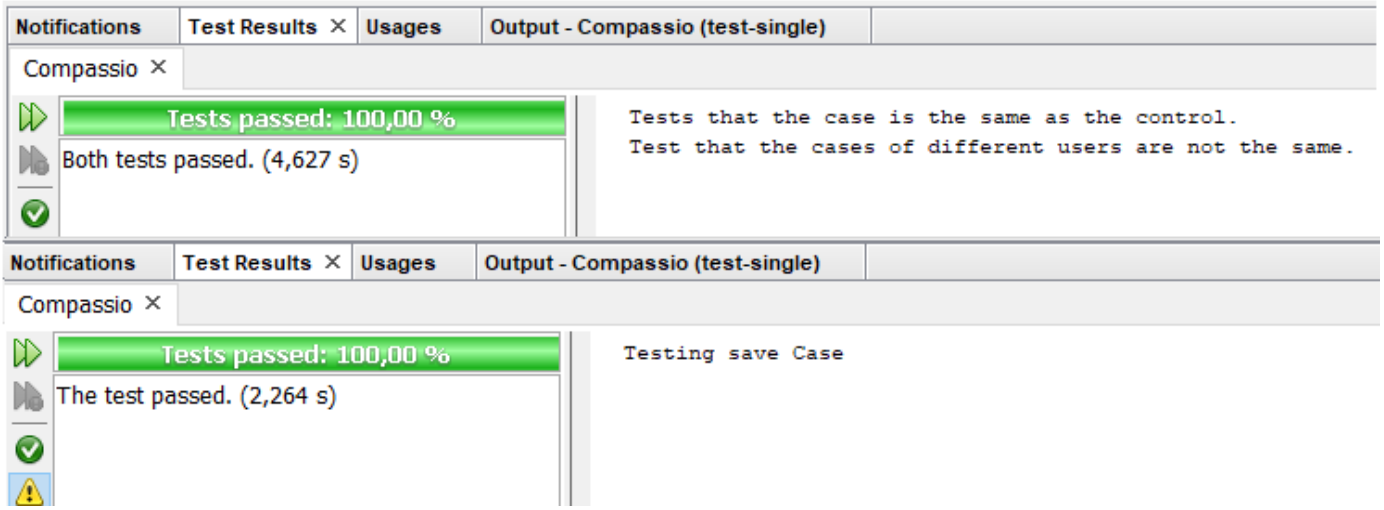
Ved flere af modulerne køres der flere tests.



Figur 10.6.1 - Resultatet af unit tests for login og change password

Login test modulet bliver der kørt 2 test. Ved første test "testWrongPassword" testes der, hvad der sker, når der logges ind med en forkert adgangskode. Login-metoden returnerer sandt når log ind er korrekt, og falsk hvis adgangskode eller bruger er forkert. Testen bruger en forkert adgangskode, så der forventes at resultatet returnerer falsk, så assertEquals tester, at resultatet af login er falsk, som gør at testen bestås. Det samme koncept ved "testCorrectLogin", her forventes at testen returnerer sandt, hvilket den gør, da der logges ind med korrekt bruger og adgangskode.

Under change password modulet køres der 2 tests, før hver test sikres der, ved at bruge @Before, at der bliver logget ind på testbrugeren. Ved "testChangePasswordFail" testes der, hvad der sker, når den gamle adgangskode ikke stemmer overens med den adgangskode, der er tilknyttet brugeren. Når dette sker, bliver adgangskoden ikke opdateret, og metoden returnerer falsk. Det er det resultat, der forventes i assertion af testen, og testen består. Når testen "testChangePassword" køres, sendes den korrekte adgangskode, der er tilknyttet brugeren, samt den nye adgangskode. Testen opdaterer brugerens adgangskode, og metoden returnerer sandt, at adgangskoden er blevet ændret. Det testes i assert some forventer at det lykkedes, så testen består. Når alle testene er blevet kørt, køres der en @After som opdaterer brugerens adgangskode og returnerer det til den gamle adgangskode, så testen kan køres igen.



Figur 10.6.2 - Resultatet af unit tests for getCase og saveCase.

Get cases modulet køre 2 tests. "testGetCases" går ind og henter den samme case fra samme bruger to gange. De 2 cases bliver så sammenlignet i compareTo metoden som ligger i case klassen for at sikre at indholdet er ens. Hvis dette er tilfældet, returnerer compareTo metoden sandt, som testen regner med at modtage, så den returnerer sandt, og testen består. I "testCasesNotEqual" testen sammenlignes to cases fra to forskellige brugere i compareTo metoden, da indholdet af de to cases er forskellige, så returnerer compareTo metoden falsk, som betyder, at de to cases ikke er ens. Hvilket testen forventer, så testen består.

Save cases modulet kører en test "saveCaseTest". Den logger ind på en testbruger som har testcases liggende. Den case henter informationerne i casen, hvor den tilføjer "Save this" til det der allerede står i main body. Derefter gemmes casen, hvis casen bliver gemt som den skal returnere saveCase metoden sandt, som er det assertion forventer. Da det stemmer overens med det testen forventer, så består testen.

11 Diskussion

Systemet omfatter de væsentligste aspekter, som gruppen ville have det til at kunne. Målet var at lave et system, hvor man kunne forskellige ting, alt efter hvordan man loggede ind. Dette har gruppen opnået. Hvis man logger ind som en sagsbehandler, kan man både se sine sager og oprette nye, hvorimod hvis man logger ind som en bostedsansat, så kan man kun se de sager, som er tilknyttet ens bosted. Dermed er systemet afgrænset efter brugerens beføjelser samtidigt med, at det både kan bruges af kommuner og bosteder. Systemet kan også nemt videreudvikles med samme koncept, da afgrænsningen er på plads, og der kan let tilføjes og fjernes faner efter behov. Systemet indeholder kun funktionalitet omkring sager, både oprettelse og redigering. Der er allerede lavet faner til planlægning og dagbog, men funktionalitet er ikke blevet implementeret. Disse er der bare for at vise konceptet omkring at let tilføje ekstra faner.

Projektet havde også nogle delmål omhandlende persondataforordningen. Herunder hvilke aspekter der er relevante for systemet, og hvilke der kan implementeres inden for tidsrammen med gruppemedlemmernes evner. Systemet overholder to af persondataforordningens syv principper; formålsbegrænsning og dataminimering. Systemet indsamler kun data, som er essentielle for at sagsbehandlere og bosteder kan give en fyldestgørende service, hverken mere eller mindre. Dataene bliver også kun brugt til dette formål og ikke til andre formål. Systemet overholder også dele af lovlighed, rimelighed og gennemsigtighedsprincippet. Data bliver behandlet lovligt i og med, at de ikke bliver solgt videre. De bliver også til en vis grad behandlet rimeligt, da man kun kan se data, som man på en eller anden måde er tilknyttet. Alle principper skal selvfølgelig opfyldes, hvis systemet på et tidspunkt skulle tages i brug.

Systemet kunne have overholdt flere af persondataforordningens principper. Der er flere årsager til, at systemet ikke gør dette. Da gruppen først begyndte at udvikle systemet, var der mest fokus på at få lavet noget funktionelt, der kunne vises til midtvejsseminaret. Der blev derfor ikke tænkt så meget på at overholde persondataforordningen. Dette har haft en indflydelse på den senere implementering af principperne, og hvilke der er blevet udeladt. Udover de forbedringer, som skulle laves i koden, så ville dette også kræve ændringer til databasen. Dette ville for eksempel være en kryptering af data og automatisk backup. Begge dele er uden for semestrets pensum, og er derfor ikke blevet gjort.

12 Konklusion

For at skabe et afgrænset system som samlede de to systemer, var det nødvendigt med en god opsætning af selve projektet. Her blev UP og Scrum brugt. Disse to sammen gjorde det muligt at lave en inkrementel og iterativ udvikling af det samlede system. Heri var det kun de første to faser i UP, inception og elaboration, som blev brugt grundet tid og ressourcer. I inceptionsfasen blev der vurderet om gruppen skulle arbejde videre med projektet eller ej. Da der ikke var tid til hele system, blev projektet afgrænset til sagsudredning som et proof of concept, for at sammenlægningen var muligt at udføre. Dette vil sige, at der var fokus på opsætningen af system, altså krav, analyse og design efter inceptionsfasen. Elaborationsfasen indebar også den reelle implementering af systemet.

Afgrænsningen betød, at en sagsbehandler skulle kunne bruge systemet og have nogle tilliggende funktioner. Resultatet af projektet er en iterativ prototype, som et forslag til sammenlægningen af de to systemer. Her var det relevant at vise, at der kunne kobles funktionalitet til forskellige brugere.
UP og Scrum gjorde det muligt at lave dette system med en tilpas afgrænsning. Brugen af UP og Scrum tvang gruppen til at have fokus på de større problemer og risici, hvilket betød at det hele tiden var nemt at finde fokus igen. Ved også at skabe systemet inkrementelt betød det, at det blev en flydende overgang fra et ellers småt og meget abstrakt overblik over problemdomænet til et fuldt design og løsningsforslag. Dokumentationen gjorde det også nemt at holde gruppen på samme spor i forhold til udviklingen.

Da det skal være muligt at viderebygge på prototypen, var det nødvendigt at lave et system, som var nemmere at udbygge og arbejde videre på. således at det kunne udvikles til et system som gjorde mere end at indeholde funktionalitet til en sagsbehandler og en login-funktion. Derfor er system sat op i en 3-lags arkitektur med fokus på at sende kun nødvendige data igennem lagene ved hjælp af interfaces. Dertil er det lavet således, at et af lagene kan skiftes ud med at andet lag, som indeholder samme interface-funktionalitet.

Til at afgrænse den data som blev opbevaret, var GDPR-lovgivningen et relevant pejlemærke at gå efter. Da lovgivningen er meget omfattende og derfor svær at inkorporere til punkt og prikke. Dette skyldes både ressource- og tidsbegrænsning. Derfor blev nogle få udvalgt som relevante startsteder til eventuel yderlige arbejde. I projektet er formålsbegrænsning og dataminimering blevet overholdt. Dette betyder at systemet kun har de nødvendige data, hvor alt indsamlet data bliver indsamlet til et specifikt formål.
I programmet er afgrænsningen af funktioner til forskellige brugere opnået ved at vælge, hvilke faner brugeren kan tilgå, lige så snart brugeren logger ind. Det vil sige, at brugeren kun kan se de funktioner, som er tiltænkt dem. Dette gør det også nemt at udskifte disse funktioner eller tilføje nye.

Derfor er det tilnærmelsesvist blevet opnået at lave et system, med dataafgrænsning og en samling af to systemer. Afgrænsningen af selve projektet er lavet for at vise, at der kan kobles forskellig funktionalitet til diverse typer af brugere, som skal bruge systemet. Gruppen har derfor også vist, at denne opsætning ville gøre det muligt at sammenlægge Sensum og Sensum Bosted. Dette betyder at projektets prototype kan ses som et forslag til, hvordan en sammenlægning af to systemer kunne laves. Funktionalitet bliver tildelt gennem gui-laget i form af hvilke knapper der bliver tilgængelige, alt efter hvilken afdeling brugeren ligger under i databasen. Ud fra dette bliver det sikret, at en bruger kun kan se de ting, som er ment til dem.

13 Perspektivering

Hvad skulle der gøres anderledes hvis projektet skulle startes forfra? Den største udfordring har været den grafiske brugerflade, så hvis projektet skulle starte forfra, skulle den grafiske brugerflade designes bedre fra starten af. Den ville skulle designes mere modulært, så det er nemmere at tilføje flere funktioner til systemet, og udvide systemet fremadrettet. Dette havde sparet tid, som er blevet brugt på at omstrukturere meget af GUI laget. Desuden ville systemet blive bygget op med mere fokus på en modulbaseret struktur, grundet at i første omgang hvor meget kode blev lavet på en gang, og der blev taget mange design beslutninger, hver for sig. Dette endte ud i noget rodet kode, som ikke være særlig struktureret. Det er dog blevet lidt forbedret, da GUI laget er blevet refaktoreret en gang. Det har gjort koden mere klar. Dette er for at gøre systemet nemmere at vedligeholde og nemmere at udvide senere hen, hvis der skal tilføjes mere funktionalitet til systemet. Altså i designfasen vil der blive brugt en del mere tid på at designe logik laget i moduler. Hvis disse to ting bliver gjort, er der et solidt fundament til et godt system, som er nemt at vedligeholde og udvide. Sammenlignet med systemet nu, så er det nuværende system lidt svært at udvide, dog er det blevet bedre. Det nuværende system er også sværere at vedligeholde, når det skal opdateres. Det er sværere at vedligeholde, da alt tilgang til logiklaget kun går gennem en klasse, og der er derfor ikke særlig høj samhørighed i klassen, og den er meget blandet og svær at finde rundt i. Det samme i persistenslaget. Denne tilgang til logiklaget, kunne deles mere ud sådan at der i stedet for at der kun er en facade, at der så er en facade til hvert modul, som indeholder logikken til det modul. Så umiddelbart kunne den nuværende facade deles op i en facade til alt der har med bruger og login at gøre, og en facade der har med sagsudredning at gøre. Eventuelt facade til senere tilføjede moduler. Dette vil øge samhørigheden i hver facade, og så er det nemt at vedligeholde og udvide enkelte moduler.

De næste par skridt fremadrettet ville være at udvide systemet med mere funktionalitet og flere funktioner. Det vil være for at understøtte de krav, der er stillet til systemet. Det næste skridt ville være først at få nogle flere af de udeladte hovedfunktioner implementeret. Dernæst ville der være fokus på at få nogle flere af GDPR-principperne implementeret, så systemet også overholder de love og reguleringer, der er for håndtering af persondata. Noget der kunne kigges på fra GDPR er opbevaringsbegrænsning, så dataene bliver slettet efter noget tid, som er bestemt af GDPR. Gennemsigtighed af dataene ville også være et næste skridt indenfor GPDR. Noget der eventuelt også kan kigges på, er at systemet deles op i to, og kører både server-klient og tre lags arkitektur. Dette gør, at der opnås større sikkerhed, for så separeres alt der har med forbindelse med databasen at gøre, og så har brugere ikke direkte adgang til det. Der kræves, at der er meget sikkerhed, når der arbejdes med persondata inden for den offentlige sektor. Der kunne også implementeres et system med adgangstoken, der ville forhøje sikkerheden en ekstra grad, da der hver gang en handling foretages, så skal brugeren verificeres.

14 Litteraturliste

Arlow, J. & Neustadt, I., 2005. *UML 2 and the unified process.* 2. red. London: Addison-Wesley Professional.

Borgersen, L., 2019. *03 Krav til projektaflevering og projektrapport.* [Online]
Tilgængelig på: https://docs.google.com/document/d/1rsWl1OkLQZf8IHPloNqwWPdMzb2rkKo4xVnVwGaAUvU/edit#heading=h.ujvm56mm9lru
[Senest hentet eller vist den 14. maj 2019].

Borgersen, L., 2019. *UML 2 And the Unified Process. Supplements.* [Online]
Tilgængelig på: https://drive.google.com/file/d/0B3ME4awmGS9ac2tRelJCZ0Z0LWM/view
[Senest hentet eller vist den 21 marts 2019].

Garcia-Molina, H., Ullman, J. D. & Widom, J., 2008. *Database Systems: The Complete Book.* 2. red. London: Pearson.

Jones, G. R., 2012. *Organizational Theory, Design, and Change.* 7. red. London: Pearson Education.

Liang, Y. D., 2018. *Introduction to Java Programming, Brief Version, Global Edition.* 11. red. Harlow: Pearson Education.

Nielsen, A. S. & Sørensen , R. C., 2019. *Persondataforordningens 7 principper | GDPR.DK.* [Online]
Tilgængelig på: https://gdpr.dk/persondataforordningen/de-7-principper/
[Senest hentet eller vist den 7. maj 2019].

Samlet Social Organisation (2019) EG Team Online Aps.

Socialstyrelsen, 2013. *Voksenudredningsmetoden (VUM) - metodehåndbog.* [Online]
Tilgængelig på: https://socialstyrelsen.dk/udgivelser/vum
[Senest hentet eller vist den 25. marts 2019].

The European Parliament and the Council of the European Union, 2016. *EUR-Lex 32016R0679 - EN EUR-Lex.* [Online]
Tilgængelig på: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679
[Senest hentet eller vist den 7. maj 2019].

15 Procesrapport

I procesrapporten vil gruppen beskrive og reflektere over deres arbejdsproces, herunder planlægning, arbejdsfordeling og samarbejdet i gruppen.

15.1 Læring og refleksion

Gruppens udgangspunkt for at komme i gang med projektet var lettere begrænset, da erfaring med gruppearbejde over længere periode ikke var den stærkeste side udover sidste semesters erfaringer, og dertil havde nogle fra gruppen ikke den store programmeringserfaring som andre gruppemedlemmer. Efterhånden som andet semester og undervisningen skred frem, fik gruppen flere værktøjer og mere viden til at udføre projektarbejdet, dermed endte det med at få en nogenlunde struktur på projektarbejdet og dets forløb.

Da det foreliggende materiale der var til rådighed, blev udleveret, var det tydeligt at se det problemorienterede arbejde var centralt. Med de nye værktøjer som gruppen nu havde til rådighed, satte det forventningerne op efter netop dette. Der er få i gruppen med større programmeringserfaring, så der var også sat en forventning op efter dette udgangspunkt.

Gennem projektetableringen, har der ikke været så meget individuelt arbejde, da der for eksempel er benyttet samarbejde for at nå frem til den problemstilling, som er blevet sat op for dette projekt. Da implementeringen og rapportskrivningen skulle påbegyndes under første iteration, i elaborations fasen, delte gruppen sig op i mindre hold. Dette var en indsats for at maksimere udnyttelsen af gruppens arbejdskraft. De mindre hold gjorde det nødvendigt at forklare resultater til hinanden. Dermed opnåede gruppen større forståelse for hinandens arbejde.

I starten af projektet under projektetableringen var der ikke den store processtyring. For at nå frem til den problemstilling som endte med at blive opsat, blev der anvendt Hey.space, som blev brugt til koordination af gruppens arbejdsopgaver såsom product backlog og sprint backlog fra Scrum. Det gjorde at gruppen fik en bedre forståelse af hvad der skulle gennemgås, og hvad der skulle laves. I inceptionsfasen kom der styr på processtyringen, hvor der blev lavet et Gantt diagram over tidsplanen. Dette kan også stå til grund for at gruppen blev fokuseret på hvad der skulle gøres, dog da selve kodningen og rapportskrivningen blev igangsat med hensyn til diverse funktioner, blev arbejdsniveau dog sat ned for gruppen. Dette blev rettet op på igen da gruppen påbegyndte anden iteration, hvor der blev lavet et mere detaljeret sprint, som gruppens Scrum Master var god til at få selve gruppen ind på det mest relevante arbejde at udføre, hvilket man kunne have draget fordel af, hvis gruppen havde gjort det samme i første iteration.

I løbet af iterationerne har vores logbog på Github fungeret efter hensigten. Den er blevet udfyldt hver gang gruppen har haft et møde, dog er det primært kun blevet til fremmøde om tirsdagen. Logbogen gav gruppen et klart overblik over vejledermøder, samt hvad det var der blev gennemgået for eksempel sidste gang man mødtes. Ligeledes har det givet overblik over hvad der skulle gennemgås til næste møde. Udover diverse tirsdagsmøder blev der i anden iteration udnyttet de store kompetencer, som gruppemedlemmerne bidrager med til projektarbejdet, for et mere individuelt, koncentreret og selvstændigt arbejde.

15.2 Projektstyring

Projektet blev styret ved brug af Scrum. Scrums planlægning er baseret på sprint. Hvert sprint starter med et planlægningsmøde, hvor der besluttes hvilke opgaver, som skal udføres i løbet af sprintet. Disse opgaver bliver lagt ind i gruppens sprint backlog. Et uddrag af backloggen kan ses herunder. Hver opgave kategoriseres, og det bliver vurderet hvor lang tid, som hver opgave kommer til at tage.

Sprint Backlog 2

29/04 - 29/5

Andet sprint

					Start	
				Dato:	30/04	07/05
Krav	**Kategori**	**Status**	**MoSCow**	Tid tilbage:	150	67
				Hastighed:	42	
Design						
Brugsmønsterrealisering af opret sag	design	I gang	M		8	0
Brugsmønsterrealisering af rediger sag	design	Ikke startet	S		8	0
Brugsmønsterrealisering af se relevante sager	design	Ikke startet	M		8	8
Brugsmønsterrealisering af log ind	design	Ikke startet	C		8	0
Implementering						
Implementering af rediger sag	Implementering	I gang	M		4	0
Opret bruger	Implementering	I gang	M		3	3
Ændre password	Implementering	Færdig	S		3	0
Rediger brugerrettigheder	Implementering	I gang	S		5	0
Slet bruger	Implementering	I gang	S		2	0
5 unit tests	Implementering	I gang	M		4	4
Rapport						
Arkitektur	Design	Færdig	M		3	3

Figur 15.2.1 - Udklip af sprint backlog.

Til at supplere denne backlog, har gruppen brugt hey.space[7] til at koordinere, hvem der laver hvilke opgaver. Hey.space er opsat med fire kategorier; backlog, in progress, testing og done. Hver opgave starter under kategorien backlog og flytter igennem arbejdsprocessen videre til de andre kategorier. Det er også her, at gruppemedlemmer kan reservere en opgave, så der ikke er flere, som laver den samme opgave.

I starten af hver arbejdsdag, holder gruppen et kort møde for at diskutere hvordan arbejdet går. Her kan man blandt andet snakke om udfordringer med specifikke opgaver. Det er også her, at der bliver revurderet hvor lang tid, som skal bruges på hver opgave. Disse møder har været gode til at fange eventuelle problemer tidligt, og sikre at projektet følger tidsplanen.

Efter første sprint, holdt gruppen et retrospektivt møde. Her blev der snakket om, hvordan sprintet er forløbet, og hvordan projektet går. Der blev også diskuteret, hvordan vores tidsvurderinger for opgaverne var, så de kunne forbedres i andet sprint. Det blev vurderet, at sprintet fungerede godt for gruppen. Det har dog været svært at følge, hvad andre gruppemedlemmer har lavet eftersom arbejdet primært har været individuelt.

[7] https://hey.space/

Gruppen udpegede Peter til at være Scrum Master for hele projektet. Scrum Masteren har som opgave at sikre at Scrum ceremonierne bliver afholdt. Han fungerer derfor også som ordstyrer ved møderne. Hvis et gruppemedlem har haft spørgsmål vedrørende Scrum, er det også Scrum Masteren, som skulle svare på dette.

15.3 Identifikation, analyse og bearbejdning af problemer

Der har igennem hele processen af projektstyringen været undervisningslektioner, som har givet en god indsigt i hvad det er for nogle værktøjer, som er væsentlige når man skal arbejde problemorienteret. Gruppen havde mange forskellige store idéer om, hvordan selve den problemorienterede arbejdsproces skulle foregå, men efterhånden som man kom igennem de første faser af projektet, begyndte det hele at indsnævre sig efter, hvad der også var realistisk for gruppen at implementere.

Der er som gruppe blevet arbejdet ud fra undervisningen, som er blevet præsenteret og erfaring fra sidste semesterprojekt, hvor man fra start af fik sat en problemorienteret problemstilling op efter de forskrifter, som var blevet givet gennem aktiviteterne i løbet af projektforløbet samt undervisningen. Projektstyringen i dette semester reflekterer meget godt en arbejdsproces på en given arbejdsplads, hvor man sidder sammen som en større gruppe af specialister for at samarbejde omkring at skabe et større fælles projekt. Gruppens projekt startede med, at der blev udleveret en case, hvor der skulle samles et system ud fra to eksisterende systemer fra EG Team Online ApS. Ud fra den udleverede case og lovgivningen som indebærer VUM, blev der afgrænset mellem planlægning, dagbog og gruppens valg, sagsudredning, som for eksempel indebærer rollefordeling, så det bliver begrænset hvad de forskellige brugere kan se og har rettigheder til at kunne. Gruppen har tilføjet egne features, såsom søg på brugere for administratoren eller søg på en sag for en sagsbehandler.

15.4 Udviklingsprocessen

Der har undervejs i projektet været fokus på forståelse. Det har medført at flere af gruppens medlemmer, er kommet godt igennem de forskellige arbejdsprocesser. Op igennem projektetableringen er der blevet arbejdet med gruppedynamikken og hvad gruppemedlemmerne har af kompetencer samt hvilke svagheder, der er i gruppedynamikken. Under inceptionfasen blev det klart for gruppemedlemmerne, at der skulle fokuseres mere på individuel indlæring og derefter sparring med de andre gruppemedlemmer. I inceptionsfasen, er der blevet arbejdet med den udleverede case opgave, ud fra hvad gruppen har lagt vægt på i selv casen. Der er blevet læst omkring vigtige relationer og lovgivning inden for casen for at få en bedre forståelse af casen og dens indhold. Ligeledes under inceptionsfasen er der blevet arbejdet med en problemformulering, i form af en afgrænsning af caseopgaven, hvor der er blevet udarbejdet en poster med relevante punkter til en fordybelse inden for projektet.

Elaborations fasen er inddelt i to iterationer. Her er der blevet arbejdet med Scrum og UP, som har fungeret yderst optimalt med at organisere og strukturere de to sprints. Et scrum sprint er lig med en iteration fra UP. I første iteration er der arbejdet frem mod at lave en funktionel prototype og at opsætte en relationel database. For at opnå dette, blev brugsmønstre analyseret igennem en brugsmønsterrealisering. Efter første iteration kunne en bruger oprette en sag og se sine sager. Hvor i anden iteration er der fokuseret på udviklingen af systemet i form af at forbedre den allerede implementerede arkitektur og forbedre designet. Brugergrænsefladelaget krævede specielt ekstra fokus for at sikre, at koden er overskuelig og let at arbejde med. Rapporten begyndte også at tage mere form i løbet af anden iteration, hvor rapporten blev sat op efter det udleverede dokument fra Semesterkoordinator Lone Borgersen i form af Krav til projektaflevering[8].

[8] Krav til projektaflevering. Udgivet af Lone Borgersen. Sidst opdateret: 14.05.2017. Internetadresse: https://docs.google.com/document/d/1rsWl1OkLQZf8IHPloNqwWPdMzb2rkKo4xVnVwGaAUvU/edit#

15.5 Formidling og kommunikation

Gruppens udbytte ud fra formidlingen af projektet vil kunne udformes i et enkelt ord, præcision. Det er så vigtigt for både skriftlig, samt mundtlig formidling, at forløbet skal gøres på en klar og tydelig måde som er forstået af alle gruppemedlemmer. Dette har gruppen opnået ud fra udarbejdelsen af en vejlederkontrakt samt en samarbejdsaftale, hvor alle parametre er gjort klare og tydelige fra starten af. Til faglige specifikationer som for eksempel krav, implementering og designvalg, hvor misforståelser nemt kan opstå i blandt projektgruppens medlemmer, fordi gruppen består af flere medlemmer og derved danner en større gruppedynamik hvor det er essentielt at kommunikationen er klar og tydelig.

Derudover erkender gruppen at selve planlægningen af projektprocessen, både implementeringsdelen såvel som selve udformningen af projektet, er en del der skulle klargøres før projektets start. Hele gruppen er klart enige at dette bestemt er et vigtigt punkt som gerne skulle sættes større fokus på til næste projektarbejde. Gruppen har også kigget på uddelegeringen af opgaver, og blev enig om at det er noget der skal gøres mere brug af til næste projektarbejde.

Implementeringen blev hovedsageligt afviklet som individuelt programmeringsarbejde, både under tirsdagsmøderne og som produktivt hjemmearbejde. Dette fungerede godt, da implementeringen kunne diskuteres, og det var muligt at koordinere hvor hvert enkelt gruppemedlem var i processen. Møderne om tirsdagen gav overblik over individuelle features og deres implementering. Det åbnede også op for nemmere kommunikation omkring fejl og mangler i disse features samt generelt ved implementeringen som en helhed.

Efter projektetableringen blev der lavet en postersession i inceptionsfasen, hvor alle grupper skulle fremlægge deres udkast af en ide til det samlede system, samt hele deres projektgrundlag. Her blev der diskuteret på kryds og tværs mellem grupperne, og man samlede op og anvendte det gode samt dårlige feedback de resterende grupper gav. I slutningen af første iteration, blev der afholdt et midtvejsseminar, hvor alle grupper igen skulle fremlægge deres udgaver af et samlet system, gennem en fremvisning af kode, system samt ide.

15.6 Samarbejde i gruppen

Ved starten af projektforløbet kunne der med fordel have blevet brugt mere tid på at snakke om og overveje de forskellige teamroller, som gruppemedlemmerne kunne inddeles i. Dette skete først efter der blev udpeget en Scrum Master. Gruppemedlemmerne gennemgik forskellige styrker og svagheder ud fra Belbins test, der blev vurderet i projektetableringen. Dette kunne sandsynligvis have gavnet på flere måder gennem projektarbejdet. Gruppen kunne have været i bedre stand til at komme med estimeringer omkring, hvor meget tid der skulle sættes af til diverse opgaver, og hvilke områder i projektforløbet ville fremkomme særligt vanskelige på grund af gruppens kompetencer eller manglen deraf. Der skulle havde været tydeligere retningslinjer og et mere veldefineret mål. Dette ville have resulteret i, at arbejdsindsatsen havde været højere og nemmere for alle gruppemedlemmer at bidrage til.

Ambitionsniveauet i gruppen blev forsøgt afstemt ved starten af projektforløbet. Som forløbet skred frem og der skete ændringer i gruppedynamikken, ved at Scrum Master trådte i forgrunden og tog en beslutning, forekom det for gruppen, at det diskuterede ambitionsniveau nok ikke længere var helt realistisk at kunne indfri, eftersom der var store ambitioner til projektstart.

Til næste gang der skal arbejdes i projektgrupper skal der kommunikeres mere klart mellem gruppemedlemmer, og der skal være flere "dumme" spørgsmål, så der ikke er tvivl om det der skal laves, og hvem der laver hvad. Derfor er det vigtigt at diskutere og kommunikere tydeligt med hinanden hvad der fungerer og ikke fungerer. Der skal også sættes større fokus på det sociale, så man kan tale mere flydende med hinanden, inddrage hinanden i samtaler som gælder det faglige, men ligeledes også det sociale aspekt.

15.7 Samarbejde med vejleder

Gruppens vejleder introducerede sig selv i uge 7 i forbindelse med at projektgruppen mødtes for første gang og samlede sig et overblik over hvad projektstyringen indebar. Derfra har gruppen indkaldt til vejledermøde regelmæssigt om tirsdagen i projektetableringen samt inceptionsfasen i forhold til visse aftalte punkter i vejlederkontrakten, hvor tidspunktet ikke er fast og afhænger alt efter afslutningen af Henriks tidligere aftaler. Det har dog varieret lidt fra uge til uge, om en indkaldelse skulle ske, da der i første og anden iteration har været et mere organiseret projektarbejde med hjælpe fra Scrum Masteren, hvilket medførte at gruppen generelt havde færre spørgsmål og spekulationer til vejleder.

Til næste projektforløb kunne der have blevet spurgt om hjælp hurtigere til vejleder, hvilket kunne medføre at problemer kunne have blevet taget stilling til hurtigere. Til vejledermøder fremover, er det vigtigt, som gruppe, at lægge vægt på at agenda og indkaldelse skal ske minimum fredagen før, det gældende tirsdagsmøde, og der skal overvejes eventuelle spørgsmål til vejleder inden indkaldelse, derved kan vejlederen forberede sig til mødet. Denne struktur har gruppen fulgt nogenlunde til punktet gennem projektforløbet som har vist sig at være effektiv til at afdække tvivl og usikkerhed omkring projektarbejdet, både med hensyn til implementeringen og rapport arbejde.

16 Oversigt over projektets kildekode

Der er genereret dokumentation for kildekoden ved hjælp af Doxygen. Filen kan findes i følgende Google Drev mappe: Link[9]
Filen er navngivet: Compassio Source Code Documentation.pdf

16.1 Brugervejledning

Ved systemets start bliver brugeren præsenteret for systemets startside. Her har man mulighed for at starte systemet op ved at trykke på log ind[10]. Denne side indeholder en log ind formel med mulighed for at trykke på log ind, hvilket vil give en fejlmeddelelse hvis intet brugernavn eller adgangskode er skrevet ind, eller hvis disse er forkert. Hvis man skriver rigtigt brugernavn samt adgangskode og trykker på log ind startes systemet herved.

Ved start af systemet præsenteres man for en oversigt over information om brugeren samt brugerens rettigheder, dette ligger under menupunktet som bliver repræsenteret af en burgermenu som ligger i toppen af venstre side. Menuen viser rolle og navn for brugeren samt menupunkter med de relevante rettigheder som brugeren har. Disse menupunkter kan være opret sag, som viser en formel for tilføjelse af en sag med en knap i bunden af siden til at gemme sagen i gruppens database, se sager, som viser en oversigt over alle relevante sager for brugeren samt en søgefunktion der gennemgår sagerne alt efter borgerens navn, cpr-nummer, sagsnummer og tjekker efter sager alt efter om det er en handicapsag, ældre sag eller en misbrugssag.

Alle disse menupunkter vil dog kun være synlige for en sagsbehandler hvor det vil være relevant, såsom hvis brugeren havde været en administrator, havde rettighederne også set anderledes ud. Menupunkterne havde derimod været, opret bruger, rediger bruger samt meget mere. Nogle af menupunkterne har også genvejstaster for øget tilgængelighed. Det er muligt at lukke systemet ned øverst i højre hjørne på krydset, samt at minimere spillet på minus tegnet ved siden af krydset.

17 Projektlog

[10] Log ind information kan findes i internt bilag B.

18 Interne bilag

18.1 Bilag A - Projektforslag, Inceptionsdokument, Vejlederaftale og Samarbejdsaftale

18.2 Bilag B - System Log Ind

Brugernavn	Adgangskode
admin	password
user	password
case	password
social	password

18.3 Bilag C – Rapportkontrolskema

17.3.1 Produktrapport		
Kapitel	Krav	Opfyldt +/-
Omslag	Indeholder omslaget projekttitel, uddannelsesinstitution, fakultet, institut, uddannelse, semester, kursuskode, projektperiode, vejleder, projektgruppe og projektdeltagere (fornavn, efternavn, sdu-email)?	+
Titelblad	(Som omslag ekskl. evt. illustration + evt. kildehenvisning til evt. omslagsillustration. Omslaget kan udgøre både omslag og titelblad. Hvis der medtages selvstændigt titelblad, så er titelbladet rapportens første højre side)	-
Resumé	Omfatter resuméet: ● Den behandlede problemstilling - hvad blev der arbejdet med og hvorfor? ● Fremgangsmåden - anvendte metoder - hvordan blev der arbejdet med det? (hvordan angreb I problemet og hvordan realiserede I løsningen (hvem, hvad, hvornår og hvorfor) ● Hovedresultater og konklusioner – hvad kom der ud af arbejdet? (max 1 side)	+
Forord	Indeholder forord hensigten med rapporten, målgruppe, forhistorie, anerkendelser, afleveringsdato samt underskrifter af alle projektdeltagere? Bemærk: Projektdeltagernes aktive deltagelse i projektforløbet anerkendes gensidigt ved projektdeltagernes underskrifter i rapporten.	+
Indholdsfortegnelse	Er der en samlet indholdsfortegnelse for hele projektrapporten med to eller tre niveauer? Er afsnittene nummererede?	+
Læsevejledning	Er der en vejledning i, hvordan rapporten kan læses, eksempelvis i form af hvilken rækkefølge afsnittene kan læses i og hvordan sammenhængen er mellem de forskellige dele af rapporten, fx mellem produktrapport og bilag?	+
	Er rapportens målgruppe beskrevet?	+

<table>
<tr><td>Redaktionelt</td><td>Beskriver redaktionelt skriveprocessen og ansvarsområder i skriveprocessen?

Ansvarsområder kan fx beskrives på fx følgende form:<table><tr><td>Afsnit</td><td>Ansvarlig</td><td>Bidrag fra</td><td>Kontrolleret af</td></tr><tr><td>Afsnit a</td><td>Person a</td><td>Person b</td><td>Person a, b, c</td></tr><tr><td>Afsnit b</td><td>Person b</td><td>Person a</td><td>Person a, b, c</td></tr><tr><td>Afsnit c</td><td>Person c</td><td>Person b</td><td>Person a, b, c</td></tr></table></td><td>+</td></tr>
<tr><td>Ordliste</td><td>Er der en kort beskrivelse af de fagtermer der bruges gennem rapporten?</td><td>+</td></tr>
<tr><td>Indledning</td><td>Giver indledningen et overblik over projektet og baggrunden for det?</td><td>+</td></tr>
<tr><td></td><td>Giver indledningen et resume af den udleverede case?</td><td>+</td></tr>
<tr><td></td><td>Indeholder indledningen problemformuleringen, jfr. inceptiondokumentet?</td><td>+</td></tr>
<tr><td></td><td>Beskriver indledningen formålet med projektet? Er formålet i overensstemmelse med hensigten med 2. semester?</td><td>+</td></tr>
<tr><td></td><td>Beskriver indledningen målene med projektet? Udtrykker målene specifikke, målbare resultater, jfr. inceptionsdokumentet? Er målene i overensstemmelse med de overordnede mål for 2. semester som udtrykt i studieordningens kap. 9 og de mere specifikke mål for projektet, som udtrykt i fagbeskrivelsen for SI2-PRO?</td><td>+</td></tr>
<tr><td colspan="3">17.3.2 Metode</td></tr>
<tr><td>1.Indledning</td><td>Giver indledningen en introduktion til afsnittet?</td><td>+</td></tr>
<tr><td>2.Metode</td><td>Er metoden i det samlede projekt beskrevet?
Er det beskrevet hvordan UP og Scrum kombineres i projektet, samt hvilke fordele og ulemper der er ved det?</td><td>+</td></tr>
<tr><td>3.Planlægning</td><td>Er planlægningen af elaborationsfasen og de enkelte iterationer beskrevet.
Er backlogs beskrevet?</td><td>+</td></tr>
</table>

	Er rollefordelingen i projektgruppen beskrevet? Er ceremonierne beskrevet? Er scrum-buts beskrevet? Bygger planen på prioriteringen af kravene efter inceptionsfasen.	
Hovedtekst	Hvis der indgår et teoretisk problem: Omfatter hovedteksten et teori-afsnit?	- Ikke teoretisk problem
1.Indledning	Indeholder indledningen en overordnet introduktion til afsnittet?	+
2.Overordnet kravspecifikation (resume, opdateret)	Indeholder afsnittet et opdateret resumé af systemafgrænsningen fra inceptionsdokumentet.	+
3.Detaljeret kravspecifikation	Omfatter den detaljerede kravspecifikation ● Detaljeret brugsmønsterdiagram (hvis relevant) ● Detaljerede brugsmønsterbeskrivelser ● Detaljerede beskrivelser af supplerende krav Fx organiseret efter FURPS+	+
4.Analyse	Omfatter afsnittet overvejelser, beslutninger og resultater vedr. ● Den statiske side af analysemodel ● Den dynamiske side af analysemodel	+
5.Design	Omfatter afsnittet overvejelser, beslutninger og resultater vedr. ● Softwarearkitektur ● Subsystemdesign ● Den statiske side af designmodel ● Den dynamiske side af designmodel ● Design af persistens ● Databasedesign	+
6.Implementering	Omfatter afsnittet overvejelser, beslutninger og resultater vedr. konvertering fra design til kode illustreret gennem	+

	udvalgte centrale eksempler, samt andre vigtige implementeringsbeslutninger Omfatter afsnittet implementering af database.	
7.Test	Omfatter afsnittet en beskrivelse af de udførte test samt resultatet af dem.	+
	Er der medtaget resultater både fra iteration #1 og fra iteration #2?	+
Diskussion	Omfatter diskussionen hvad der er opnået, og hvad der ikke er opnået i projektet i forhold til det forventede som beskrevet i indledningen. Hvad er styrkerne og svaghederne ved jeres resultater? Kunne I have opnået bedre resultater?	+
Konklusion	Opsummerer konklusionen resultaterne og diskussionen af dem og giver det på problemformuleringen?	+
Perspektivering	Fremtidigt arbejde: Hvad ville de næste skridt i projektet være, hvis der var mere tid? Refleksion: Hvordan ville I gribe projektet an, hvis I skulle starte forfra?	+
Litteraturliste	Er litteratur angivet på en anerkendt form? Er alle former for litteratur som bøger, artikler og hjemmesider medtaget? Er der kildehenvisninger i teksten? Materiale som gruppen ikke selv har fremstillet i dette projekt skal være angivet med kilde! Er alle kildehenvisninger i teksten anført på samme måde? Er der kildeangivelser på figurer, grafer etc. som projektgruppen ikke selv har frembragt?	+

17.3.3 Procesrapport		
Kapitel	Krav	Opfyldt +/-
Læring og refleksion	Er der en redegørelse for læring og refleksion?	+
Projektstyring	Er der en redegørelse for projektplanlægning og styring? Er der en beskrivelse af den faktiske projektplanlægning og styring, med inddragelse af væsentlige artefakter, som fx	+

	sprint backlogs, væsentlige ceremonier, som fx daily scrum, rollevaretagelse mm?	
Identifikation, analyse og bearbejdning af problemer	Er der en redegørelse for identifikation, analyse og bearbejdning af problemer	+
Udviklingsprocessen	Er der en redegørelse for det faktiske udviklingsarbejde? Er der en beskrivelse af faserne, iterationerne og det faktiske arbejde i dem?	+
Formidling og kommunikation	Er der en redegørelse for formidling og kommunikation	+
Samarbejde i gruppen	Er der en redegørelse for samarbejde i gruppen	+
Samarbejde med vejleder	Er der en redegørelse for samarbejde med vejleder	+

17.3.4 Kildekode		
Kapitel	Krav	Opfyldt +/-
Oversigt over projektets kildekode	Er der en oversigt over projektets kildekode, fx filstruktur eller javadoc?	+

17.3.5 Brugervejledning		
Kapitel	Krav	Opfyldt +/-
Brugervejledning	Er der en kortfattet brugervejledning?	+

17.3.6 Projektlog		
Kapitel	Krav	Opfyldt +/-
Projektlog	Er der en adresse på og et link til projektloggen i Github	+

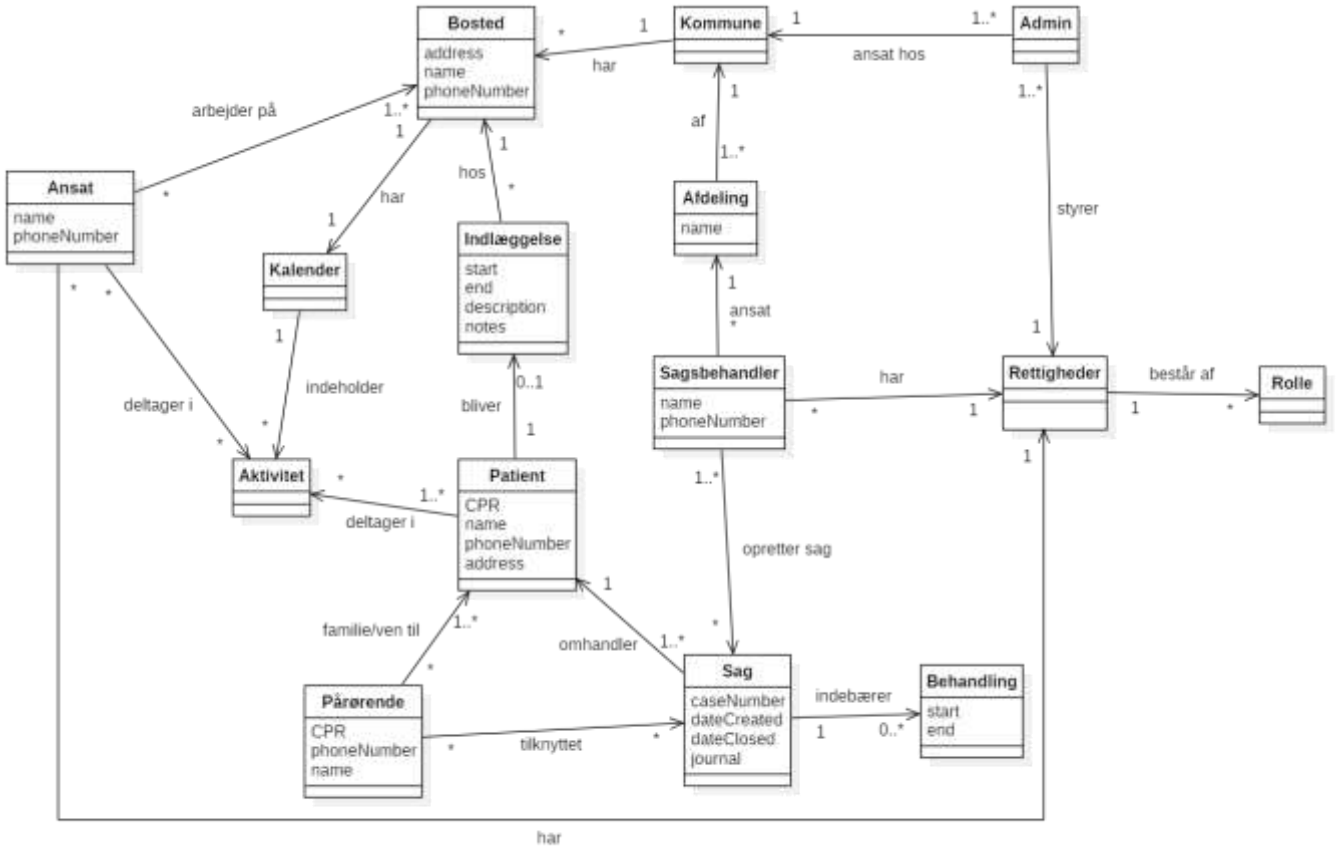
17.3.7 Interne bilag		
Kapitel	Krav	Opfyldt +/-

Projektgrundlag	Er projektgrundlaget medtaget? Inceptionsdokumentet skal afleveres i den udgave det blev bedømt.	+
Rapportkontrolskema	Er der et udfyldt rapportkontrolskema	+
Andet	Er der andre relevante interne bilag, dvs. materialer produceret af gruppen selv?	+

17.3.8 Eksterne bilag		
Kapitel	Krav	Opfyldt +/-
Eksterne bilag	Er der medtaget relevante eksterne bilag, dvs. materialer som gruppen ikke selv har produceret med som er nødvendige for at kunne læse rapporten?	- Ingen billag

17.3.9 Rapporttekniske elementer		Opfyldt +/-
Layout	Er der anvendt samme layout i alle kapitler? Er layout overskueligt/harmonisk?	+
Sprog	Er rapporten skrevet i en neutral sprogtone? Er sproget let læseligt og flydende? Er der udført stavekontrol og kontrol af tegnsætning?	+
Sidenummerering	Er der korrekt og konsistent sidenummerering i rapporten?	+
Figurer/diagrammer	Er alle figurer konsekvent nummererede? Er der figurtitel og figurtekst til alle figurer? Er figurtitler og figurtekster dækkende og afklarende? Er figurerne tydelige og læsbare? Er figurerne informationsgivende og i den rette sammenhæng?	+
Tabeller	Er alle tabeller konsekvent nummererede? Er der en forklarende tabeltekst til alle tabeller? Er alle søjler og rækker forsynet med parametre? Er der enheder på alle relevante rækker og søjler?	+
Sporbarhed af begreber	Er der en konsekvent brug af samme betegnelse for et givet begreb igennem rapporten?	+

18.4 Bilag D - Domænemodel



19.1 Bilag E - Detaljeret brugsmønsterbeskrivelser for “Redigering af sag”, “Log ind” og “Se relevante sager”

Brugsmønster: Redigering af sag	
ID:	2
Primære aktører:	Sagsbehandler.
Sekundære aktører:	Ingen.
Kort beskrivelse:	Sagsbehandleren skal kunne ændre og tilføje oplysninger til en eksisterende sag.
Prækonditioner:	• Sagsbehandleren skal være logget ind. • Sagsbehandleren skal have rettighed til at ændre sagen.
Hovedhændelsesforløb:	1. Brugsmønstret starter når sagsbehandleren vælger en sag. 2. Sagsbehandleren vælger “rediger sag”. 3. Sagsbehandleren tilføjer nye eller opdaterer eksisterende oplysninger til sagen. 4. Hvis sagsbehandleren vælger “gem ændringer” 4.1. Systemet gemmer ændringerne i databasen. 5. Hvis sagsbehandleren vælger “annuller” 5.1. Systemet kasserer ændringerne.
Postkonditioner:	Ingen.
Alternative hændelsesforløb:	Ingen.

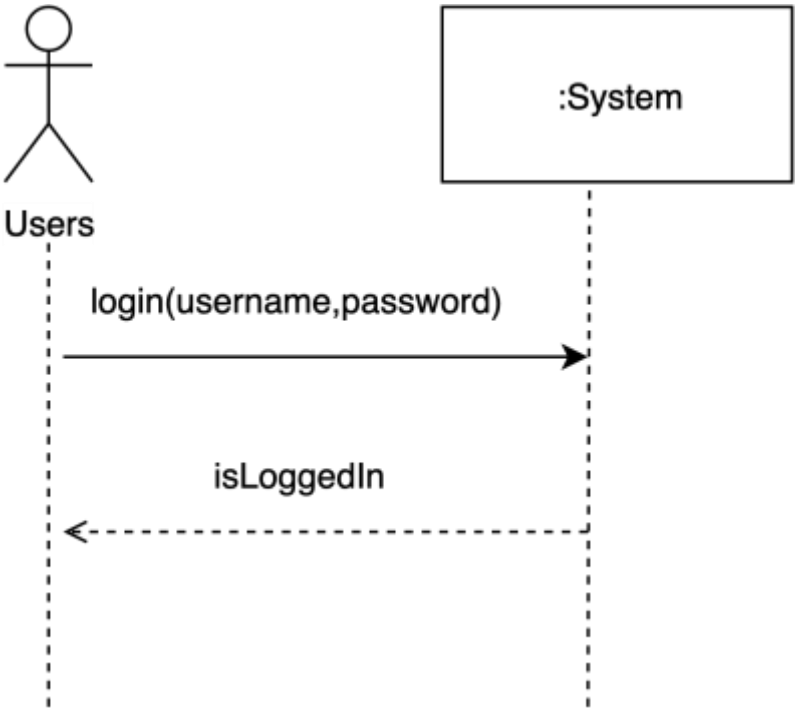
Brugsmønster: Log ind	
ID:	3
Primære aktører:	Sagsbehandler, Admin, Borger/Patient og Ansat.
Sekundære aktører:	Ingen.
Kort beskrivelse:	Aktøren indtaster brugernavn og password. Herefter logges der ind med en vis rolle på den givne side.
Prækonditioner:	Ingen.
Hovedhændelsesforløb:	1. Brugsmønstret starter når aktøren indtaster log ind oplysninger i programmet. 2. Aktøren logger ind. 3. Hvis log ind oplysningerne er forkerte 3.1. Systemet viser en fejlmeddelelse til aktøren. 4. Ellers 4.1. Aktøren logges ind i system.
Postkonditioner:	• Aktøren er logget ind.
Alternative hændelsesforløb:	Ingen.

Brugsmønster: Se relevante sager	
ID:	4
Primære aktører:	Sagsbehandler.
Sekundære aktører:	Ingen.
Kort beskrivelse:	Sagsbehandleren skal kunne se sine egne sager.
Prækonditioner:	• Sagsbehandler skal være logget ind. • Sagsbehandler skal have rettighed til at se sagerne.
Hovedhændelsesforløb:	1. Sagsbehandler vælger vis sager. 2. Systemet viser en liste over sager, hvor sagsbehandleren er tilknyttet. 3. Hvis sagsbehandleren vælger en sag 3.1. Systemet viser sagen på skærmen.
Postkonditioner:	• Sagsbehandler kan se sine sager.
Alternative hændelsesforløb:	SøgPåEnSag (SearchForACase)

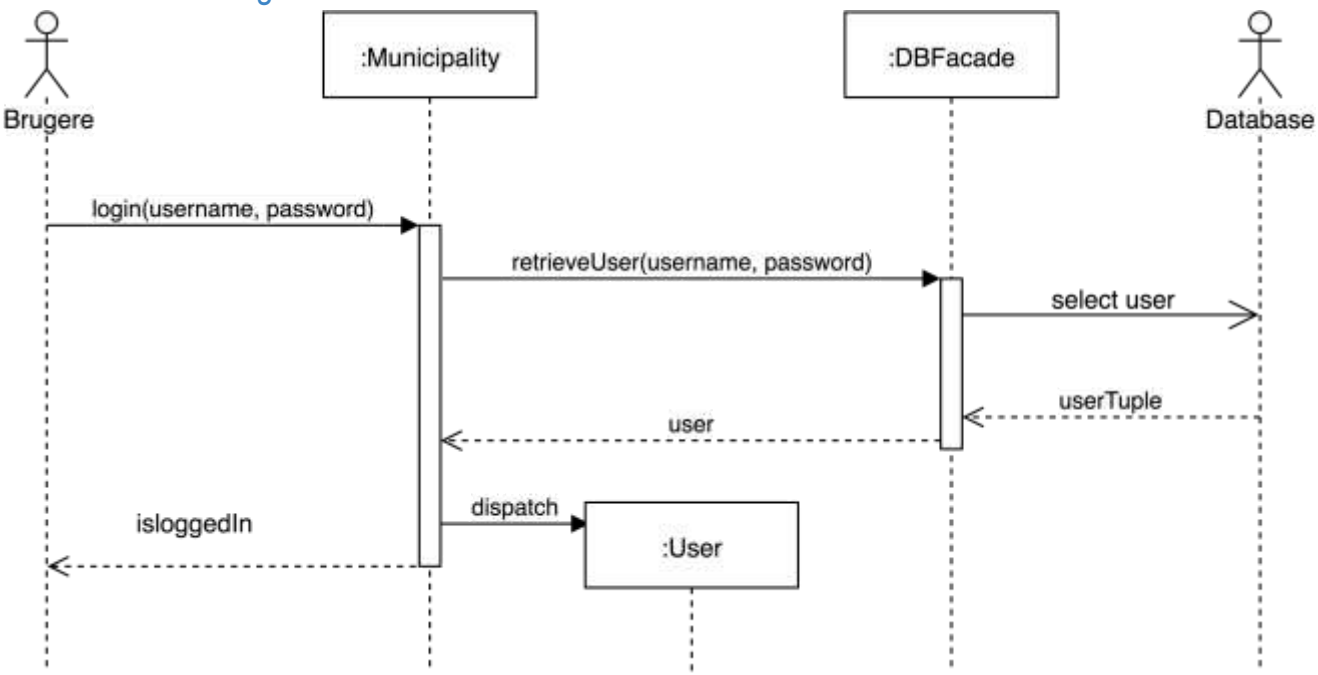
Alternativt flow: Se relevante sager:SøgPåEnSag (SearchForACase)	
ID:	4.1
Primære aktører:	Sagsbehandler.
Sekundære aktører:	Ingen.
Kort beskrivelse:	Sagsbehandleren skal kunne søge på sager.
Prækonditioner:	• Sagsbehandler skal være logget ind.
Alternativt hændelsesforløb:	1. Det alternative flow starter efter step 2 i hovedflowet. 2. Sagsbehandleren indtaster et navn eller CPR-nummer i søgefeltet. 3. Systemet fjerner alle sager fra listen på skærmen. 4. Systemet søger efter sager med det givne navn eller CPR-nummer. 4.1. For hver sag, der passer på søgningen 4.1.1. Tilføj sagen til listen, som vises på skærmen.
Postkonditioner:	• Sagsbehandler kan se de sager, som passer på søgningen.

19.3 Bilag F - Brugsmønsterrealisering for login

19.3.1 System interaktionsdiagram



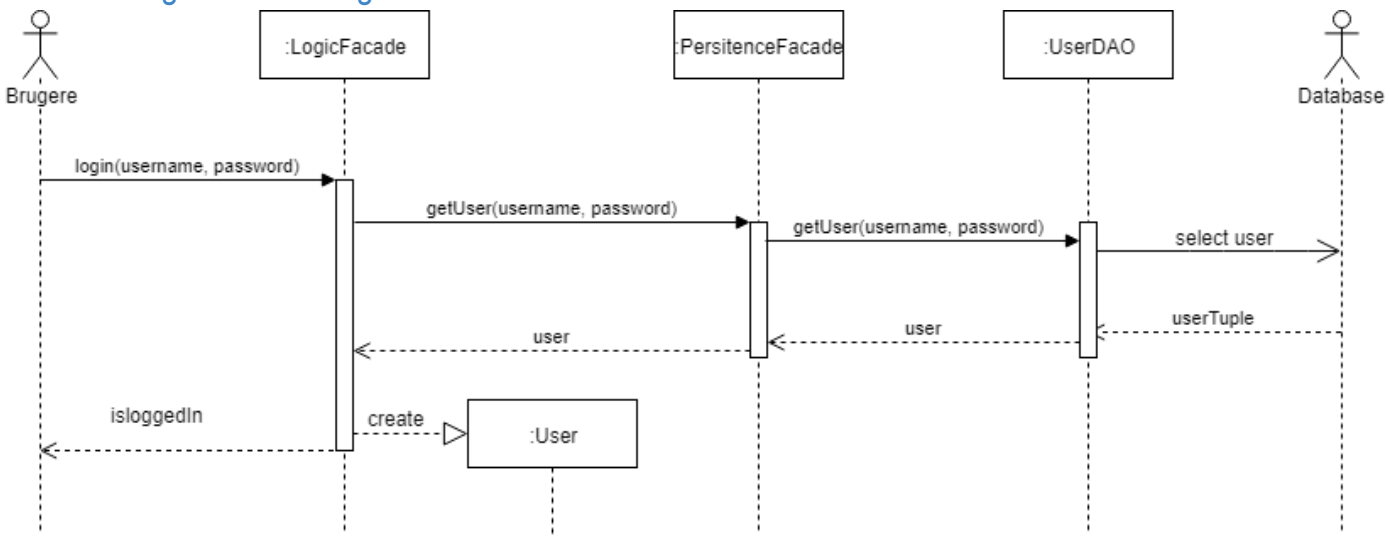
19.3.2 Sekvensdiagram



19.3.3 Kontrakt for operation

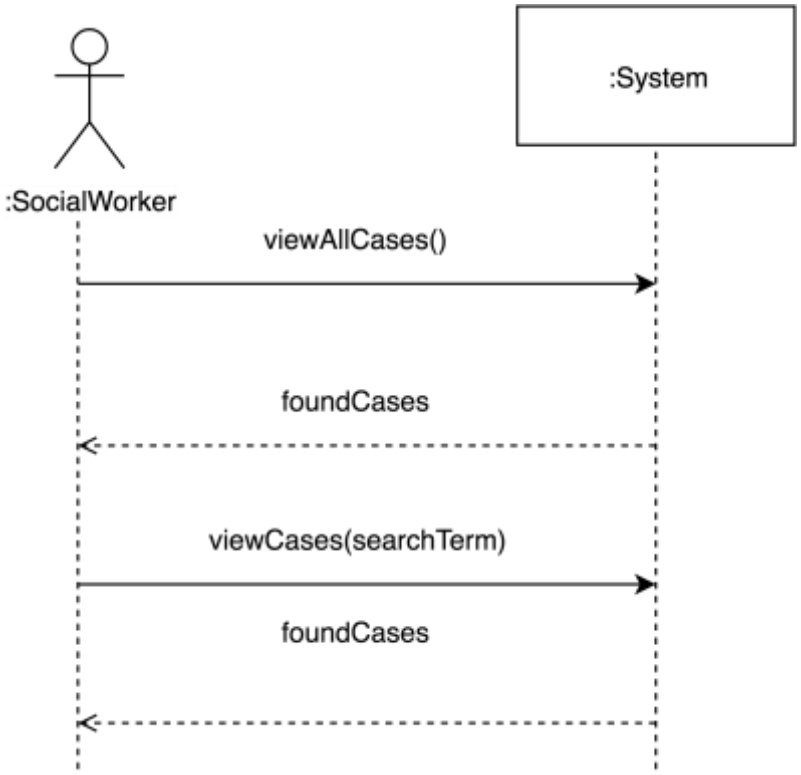
Kontrakt	
Operation	login(username, password)
Krydsreference	Log ind
Ansvar	er at logge en bruger ind, så den kobles til den rigtige bruger-instans. En bruger kan kun logge ind hvis brugernavn og password passer overens med en gemt bruger i systemet. Hvis login er korrekt, så gemmes den tilsvarende bruger-instans ellers vises en fejlmeddelelse.
Output	
Prækondition	• Brugeren er ikke logget ind
Postkondition	• Brugeren blev logget ind • Der blev oprettet en bruger instans • Brugeren blev sendt til den rigtige del af systemet.

19.3.4 Design sekvensdiagram

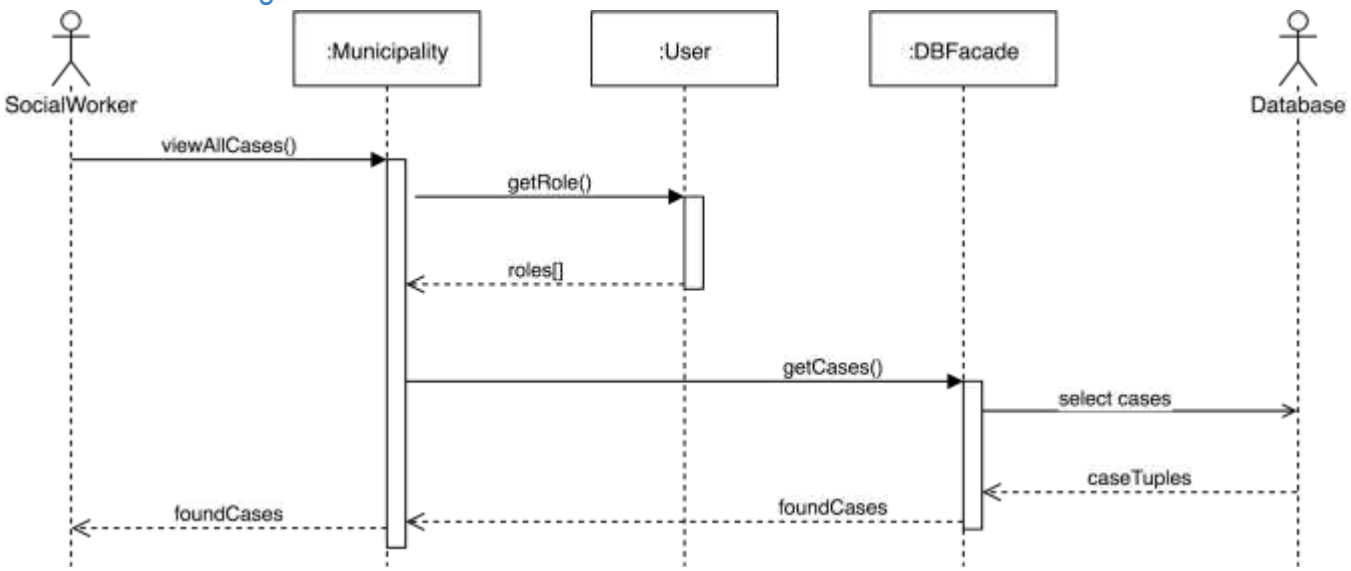


19.4 Bilag G - Brugsmønsterrealisering for SeRelevanteSager

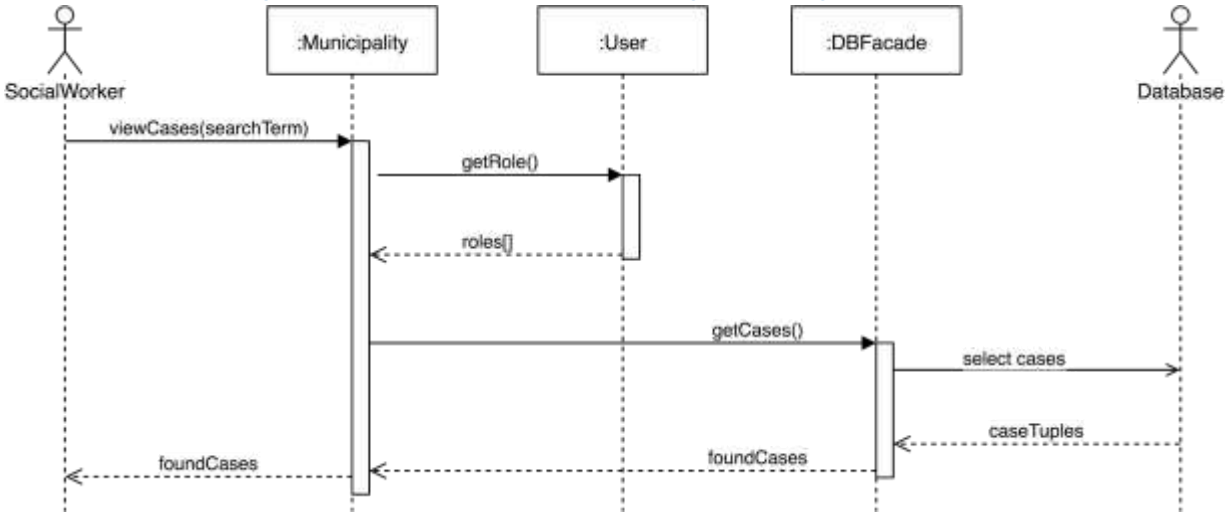
19.4.1 System interaktionsdiagram



19.4.2 Sekvensdiagram



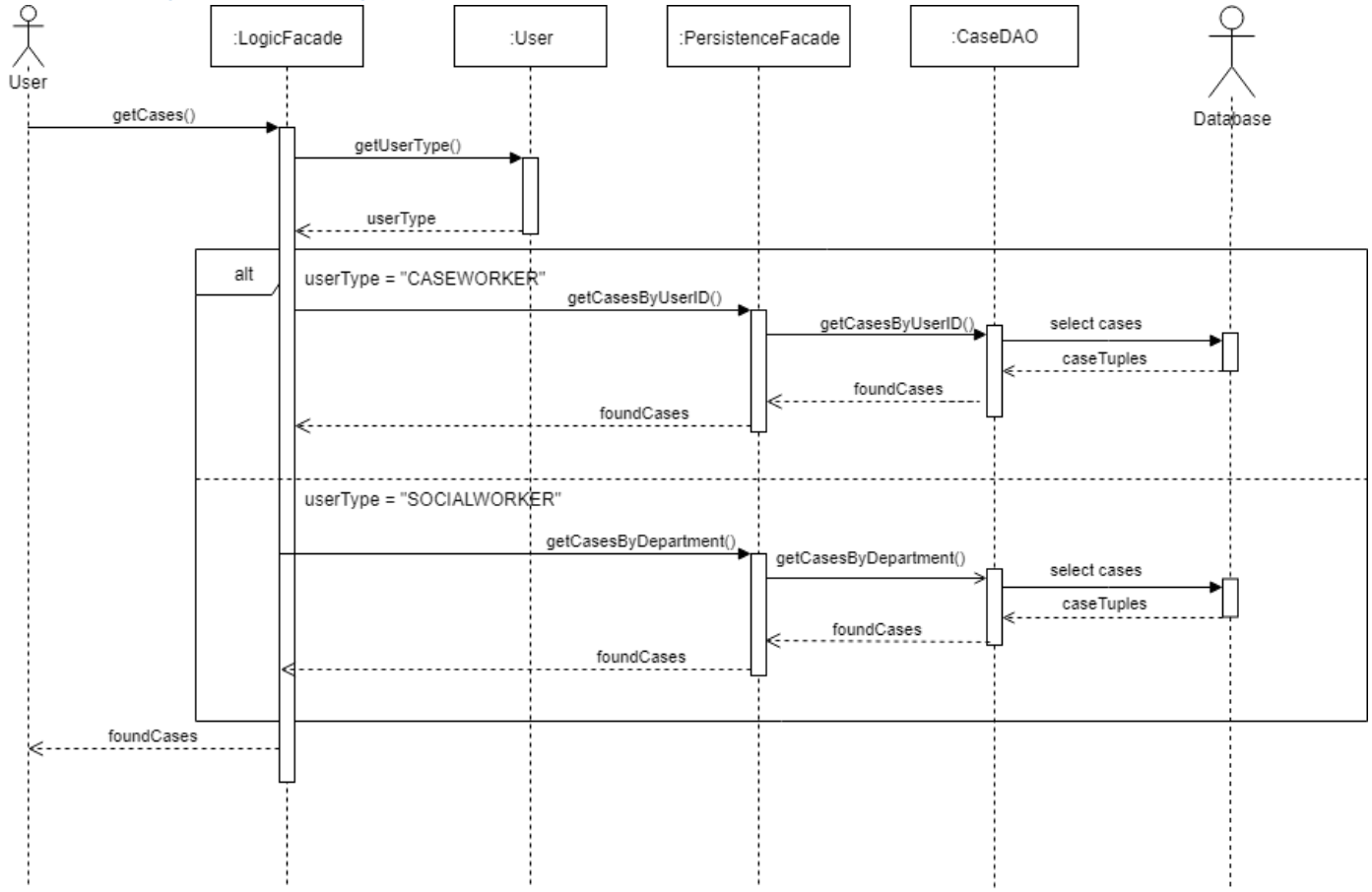
19.4.3 Sekvensdiagram for udvidelsesmønster SøgPåEnSag



19.4.4 Kontrakt for operation

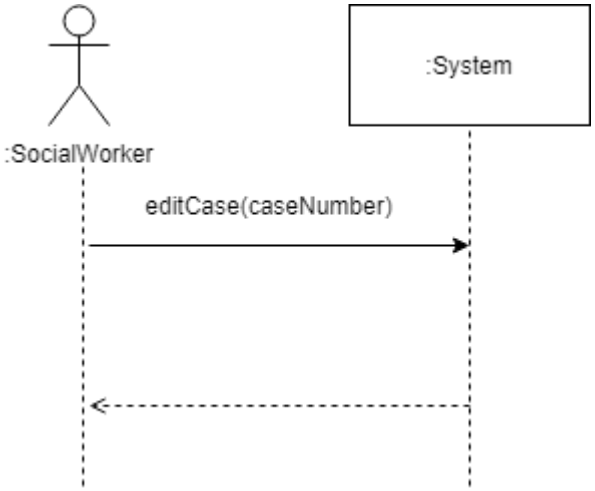
Kontrakt	
Operation	viewCases()
Krydsreference	se relevante sager
Ansvar	er at returnerer en list med sager, som brugeren har rettighederne til at se. giv mulighed for at vælge en sag og åbne den.
Output	
Prækondition	• Brugeren er logget ind.
Postkondition	• at sagsbehandleren kan se liste over tilknyttede sager.

19.4.5 Design sekvensdiagram

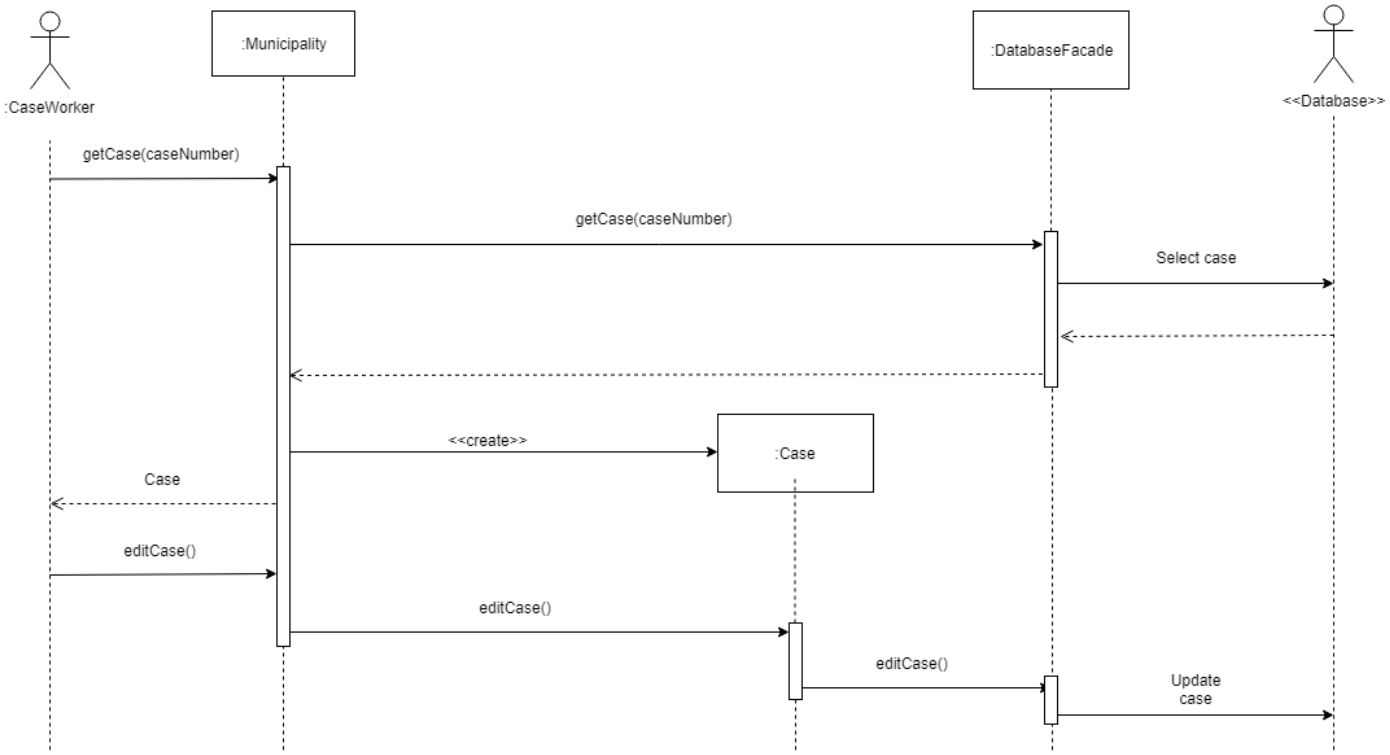


19.5 Bilag H - Brugsmønsterrealisering af rediger sag

19.5.1 System interaktionsdiagram



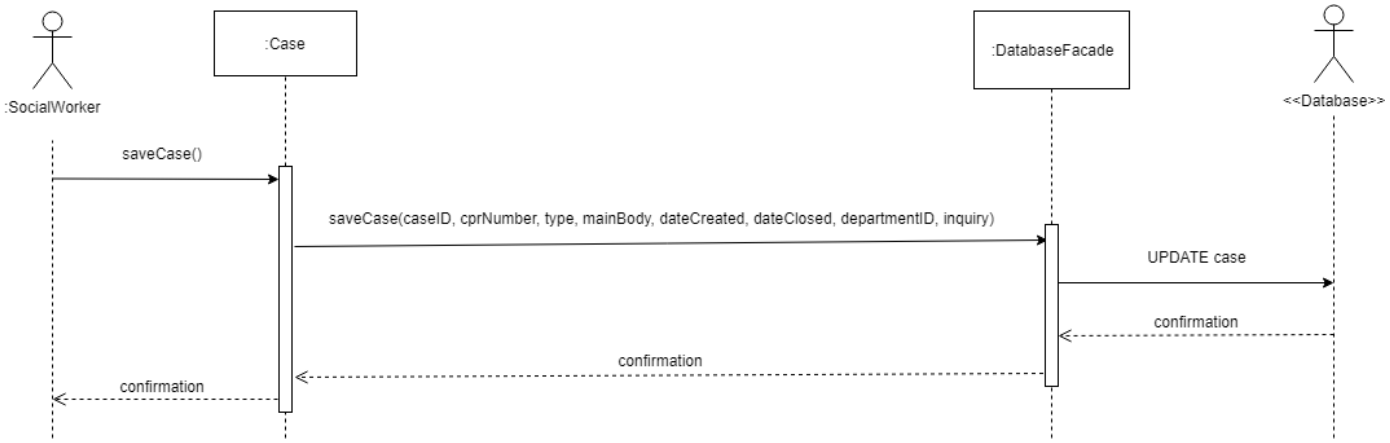
19.5.2 Sekvensdiagram



19.5.3 Kontrakt for operation

Kontrakt	
Operation	editCase()
Krydsreference	rediger sag
Ansvar	at ændre sagen med nye eller ændrede oplysninger fra sagsbehandleren. at gemme sagen efter, at der er blevet lavet ændringer.
Output	editCase()
Prækonditioner	• Brugeren har valgt en Case.
Postkonditioner	• Case blev opdateret med de nye ændringer eller tilføjelser.

19.5.4 Design sekvensdiagram



19.6 Bilag I - Design Class Diagram