EUC NORD - OPGAVESTYRING

Dmab0917 – Gruppe 2

Information

Antal normal sider: 38,25 (91,801 tegn m. mellemrum)
SVN path: https://kraka.ucn.dk/svn/dmab0917_1Sem_Projekt_2
Revisions nr. 300
Kraka.ucn.dk SQL server database: dmab0917_1067544

Andreas Larsen(1026423), Jacob Thomsen(1067472), Katrine Madsen(1067361) & Matias Kragh(1067544)

I. Læsevejledning

II. Referencer

Referencer bliver indikeret med '[]' og et tal der korresponderer med kildelisten (Eksempel: [1]).

Bilag bliver refereret med bogstaver (Eksempel: [A]).

III. Kodebeskrivelser

I systemudvikling og implementationsafsnittet i denne rapport er der en bestemt måde, at kodeudsnit og modeller bliver beskrevet. Herunder bliver klasser indikeret med **Fed** skrift og med 'klasse' stående foran sig (Eksempel: … klassen **TaskDB**). Variabler bliver indikeret med **`Courier New`** skrifttype (Eksempel: **`int weight`**). Metoder bliver indikeret med *kursiv* og efterfulgt af en tom parentes og med "metoden" stående foran sig (Eksempel: … metoden *buildByID()*).

IV. Kode standarder

Konventionerne der er beskrevet herunder, er formuleret i et tidligere projekt med samme medlemmer.

Som udgangspunkt følges Sun Microsystems Inc Java kode konventioner [4]. Undtagelser findes herunder.

- Metode parametre af for eksempel String typen vil parameteren skrives som metodeNavn(String string), hvis det er en primitiv type gives der et beskrivende navn, eksempelvis name.
- Variabler skal være så korte som muligt imens de stadigvæk er beskrivende, det samme er gældende for metoder.
- I tilfælde af for-each loops skal objektet der ligger i kollektionen navngives 'element'.
- I tilfælde af at en metode skal returnere, kaldes return kun en gang og derfor laves en lokal variabel med navnet "res" af returnerings typen.

V. SQL Standarder

- Attributter der bliver angivet som *Primary key* bliver angivet som en *constraint* i slutningen af tabel oprettelsen og ikke når attributen bliver oprettet. Navnet bliver bestemt ud fra tabel navnet og variablen, der er nøglen.

  Eksempel: PKLocation_id
- Attributter der bliver angivet som *Foreign key* bliver angivet som en *constraint* på samme måde som den primære nøgle, hvor navnet bliver bestemt ud fra tabel navnet og den tabel

hvor nøglen går til.

Eksempel: FKLocation_Zipcode

- På tabeller hvor der er en *composite key* bliver den angivet med CPK i stedet for PK.
- Tabel navn er angivet med stort forbogstav og camelCase.
- Attribut navne er angivet med småt.

VI. Terminologi

- Grundlæggende antagelser – Ikke beviste handlinger og tanker iforhold til menneskelige reaktioner og behov.
- Værdier – begrundet strategier, mål og filosofier der enighed om i et kulturelt omfang.
- Artefakterne – Anlæg der bliver brugt til struktureing og processer i kulturen
- Styret missil – Personer i en gruppe der har et fælles mål der ikke nødvendigvis givet til dem.
- Mock-up – En prototype af et system eller maskine, kan være lavet som en samling af papirer.
- Kontinuert – Bliver brugt til at beskrive noget der er vedvarende.

Forord

I dette projekt har vi haft stor nytte af en række personer og dem vil vi gerne takke her. Tak til Henrik Larsen der er en ansat i Bygnings- og Serviceafdelingen af EUC Nord. Han stod til rådighed og gav god information og svar på vores spørgsmål. Tak til vores vejleder Torben Larsen for hans hjælp til udformningen af projektet og vores andre undervisere når vi havde nogle andre spørgsmål. Tak til UCN for undervisningsmulighederne tilbudt og at der var lokaler til rådighed under projektarbejdet.

Indholdsfortegnelse

1. Indledning

EUC Nord Bygnings- og serviceafdeling vil gerne have et system, der kan gøre vedligeholdelsesarbejdet mere overskueligt og effektivt. Det kunne være en forbedring at digitalisere opgavestyringen og give medarbejderne nemme adgangsmuligheder til systemet. Systemet skal primært bruges af de ansatte i Service- og Bygningsafdelingen, dog skal andre ansatte også delvist kunne tilgå systemet. Det er tiltænkt at gøre kommunikationen mellem afdelingerne lettere.

2. Problem område

I Service- og Bygningsafdelingen er der lige nu ikke nogen måde, hvor de nemt og overskueligt kan se hvilke opgaver, som de ansatte har. Opgaverne de ansatte har bliver ofte kun verbalt oprettet. Lederen kan derfor have svært ved at holde overblik over sine ansatte og deres opgaver.

Indberetning af informationer sker gennem filer og kan derfor forholdsvis nemt gå tabt, hvis der ikke er kontrol over deres spredning til medarbejderne.

På nuværende tidspunkt har EUC Nord benyttet et andet system til at kommunikere mellem afdelingerne. Dog er dette system meget besværligt at bruge og derfor er der nogle afdelinger, der vælger ikke at bruge det til fordel for simpel e-mail kommunikation, heraf det verbale som lederne ikke har styr på.

Derfra ligger det centrale problemområde og dette formuleres i vores problemformulering herunder.

3. Problemformulering

Hvordan kan vi lave et brugervenligt opgavestyringssystem, som de ansatte på EUC Nord vil foretrække frem for deres nuværende system igennem digitalisering af deres indberetnings- og organiseringsmetoder?

- Hvad kan der gøres for at standardisere kommunikationsmulighederne, de ansatte har imellem hinanden og lederen for afdelingen?
- Hvordan kan opgavefordelinger til ansatte systematiseres i et IT-system?

4. Metode

Unified Process, også kaldet UP, er en meget fleksibel og åben software development process (SDP), som også opfordrer, at der inkluderes andre gode iterative metoder fra andre SDP'er som for eksempel Scrum eller XP.

UP er en moderne iterativ udviklingsmetode, som er organiseret i korte, men med fast længde af iterationer. Resultatet af hver iteration er testet, integreret og er et eksekverbart delsystem. Den iterative metode er

baseret på, at der gennem udvidelse og udvikling af et system, ender ud med en cyklisk tilpasning og tilbagekobling. Dette medfører, at systemet gradvist over tid vokser systematisk, dette er også kaldet for iterativ og inkrementel udvikling.

Alle iterationer indebærer, at der bliver valgt en lille delmængde af krav og design at lave, som ikke nødvendigvis er det, som ender ud med i det færdige projekt. Det positive ved det er, at design og implementation kan komme hurtigt i gang i små skridt, som der kan gives feedback på, hvorefter der kan ændres en smule i den rigtige retning, i stedet for at have en langsom start grundet mange krav. Denne feedback er meget værd for denne proces, da der i de tidlige iterationer ikke behøves at skulle spekulere over hele projektet. Samtidig kan slutbrugeren få en chance undervejs til at se en partiel del af systemet virke, hvorefter de kunne komme med ændringer til systemet, hvis dette skulle være nødvendigt. I de tidlige stadier af udviklingen af systemet kan dette virke meget ude af balance.

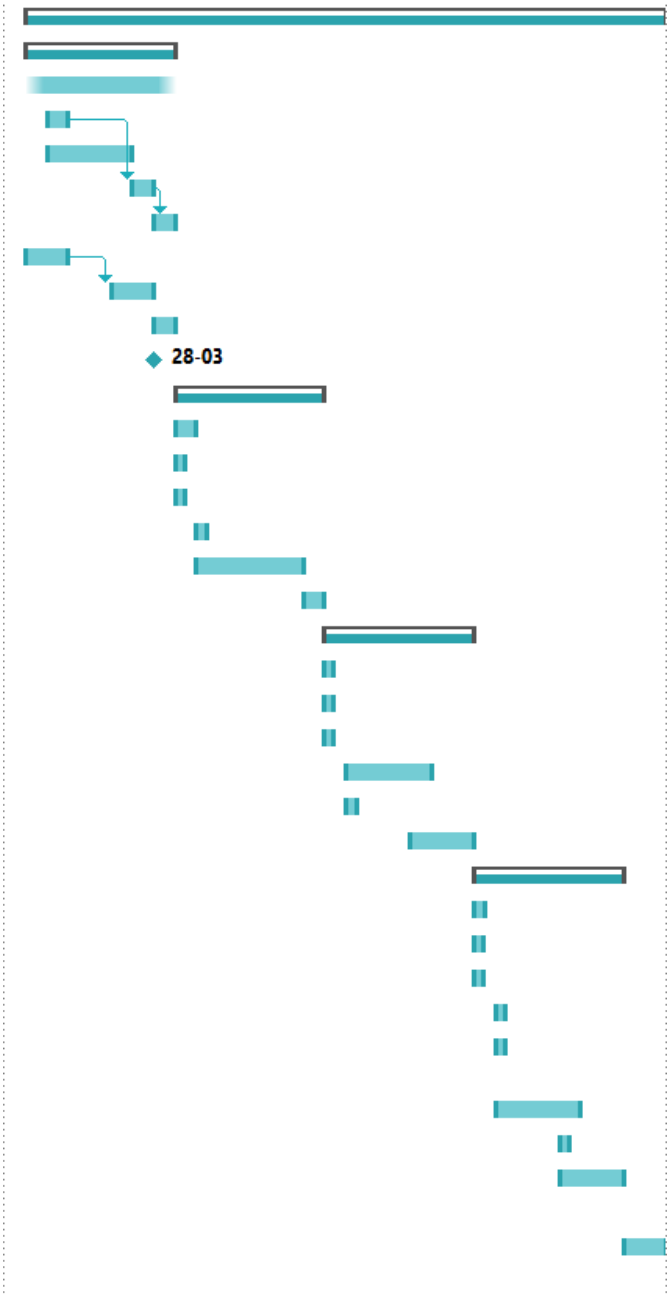
Denne måde at udvikle på giver nogle gode fordele:

- Færre projekt fejl
- Bedre produktivitet
- Lavere fejlfrekvenser
- Tidligt sænkning af høje risici
- Tidligt synlig fremdrift
- Tidlig feedback, som fører til et mere raffineret system, som også opfylder behov og krav langt bedre

Projektet er bygget op efter UP som består af 4 store faser, som kan deles op i flere. De fire faser er:

1. **Inception**, som giver en omtrentlig vision, forretningsmæssig sag, projektets omfang og vage estimater.
2. **Elaboration**, udarbejdes der en mere raffineret vision, en iterativ implementering af kernearkitekturen, løsning af høje risici, en identifikation af de fleste krav samt omfanget og mere realistiske estimater.
3. **Construction**, er en iterativ implementering af de resterende lavere risikofyldte og lettere elementer samt forberedelse til implementering.
4. **Transition**, er beta-teste samt implementering og langsom overdragelse af program til virksomhed

Herunder ses det Gantt kort over de iterationer, som kommer efterfølgende.



Figur 1: Gant kort over tidsplanen

	Task Name	Duration	Start	Finish
1	**Semester Projekt**	**22 days**	**Thu 22-03-18**	**Fri 20-04-18**
2	**Inception**	**5 days**	**Thu 22-03-18**	**Wed 28-03-18**
3	Organisationsbeskrivelse	5 days		
4	Workflow	1 day	Fri 23-03-18	Fri 23-03-18
5	Mock-up og interview	2 days	Fri 23-03-18	Mon 26-03-18
6	Usecase identifikation	1 day	Tue 27-03-18	Tue 27-03-18
7	Usecase beskrivelser	1 day	Wed 28-03-18	Wed 28-03-18
8	Domænemodel	2 days	Thu 22-03-18	Fri 23-03-18
9	Relationel Model	2 days	Mon 26-03-18	Tue 27-03-18
10	System vision	1 day	Wed 28-03-18	Wed 28-03-18
11	Organisations analyse done	0 days	Wed 28-03-18	Wed 28-03-18
12	**Elaboration**	**5 days**	**Thu 29-03-18**	**Wed 04-04-18**
13	Fullydressed for opret opgave	1 day	Thu 29-03-18	Thu 29-03-18
14	SSD for opret opgave	0.5 days	Thu 29-03-18	Thu 29-03-18
15	Interations diagram for opret opgave	0.5 days	Thu 29-03-18	Thu 29-03-18
16	Design klasse diagram for opret opgave	0.5 days	Fri 30-03-18	Fri 30-03-18
17	Opfølgning på virksomheds analyse	3 days	Fri 30-03-18	Tue 03-04-18
18	Undervisning	1 day	Wed 04-04-18	Wed 04-04-18
19	**Construction**	**5 days**	**Thu 05-04-18**	**Wed 11-04-18**
20	Database opsætning	0.33 days	Thu 05-04-18	Thu 05-04-18
21	Database forbindelse	0.33 days	Thu 05-04-18	Thu 05-04-18
22	System test scenarier til opret opgave	0.33 days	Thu 05-04-18	Thu 05-04-18
23	implementation af opret opgave	2 days	Fri 06-04-18	Mon 09-04-18
24	Test af opret opgave implementation	0.5 days	Fri 06-04-18	Fri 06-04-18
25	GUI opret opgave	3 days	Mon 09-04-18	Wed 11-04-18
26	**construction 2**	**5 days**	**Thu 12-04-18**	**Wed 18-04-18**
27	fullydressed for vælg opgave	0.5 days	Thu 12-04-18	Thu 12-04-18
28	System test scenarier til vælg opgave	0.33 days	Thu 12-04-18	Thu 12-04-18
29	SSD for vælg opgave	0.33 days	Thu 12-04-18	Thu 12-04-18
30	Interations diagram for vælg opgave	0.33 days	Fri 13-04-18	Fri 13-04-18
31	Opdatering af designklasse diagram ifh. Vælg opgave	0.33 days	Fri 13-04-18	Fri 13-04-18
32	implementation af vælg opgave	2 days	Fri 13-04-18	Mon 16-04-18
33	Test af vælg opgave implementation	0.33 days	Mon 16-04-18	Mon 16-04-18
34	Gui vælg opgave	3 days	Mon 16-04-18	Wed 18-04-18
35				
36	Rapport skrivning	2 days	Thu 19-04-18	Fri 20-04-18

Der blev så vidt muligt forsøgt at opsætte en tidsplan over projektet, men da dagene der blev brugt på projektet, har været forskudt, så har datoerne ikke kunne give mening for selve opgavens forløb, men disse er mere set som et antal af dage, der er blevet brugt på de forskellige dele.

I perspektiveringsafsnittet vil der blive reflekteret over den måde tidsplanen og Gantt kortet er blev brugt på. Helheden af tidsplanen og Gantt kortet kan ses i bilag[A].

5. Virksomheds analyse

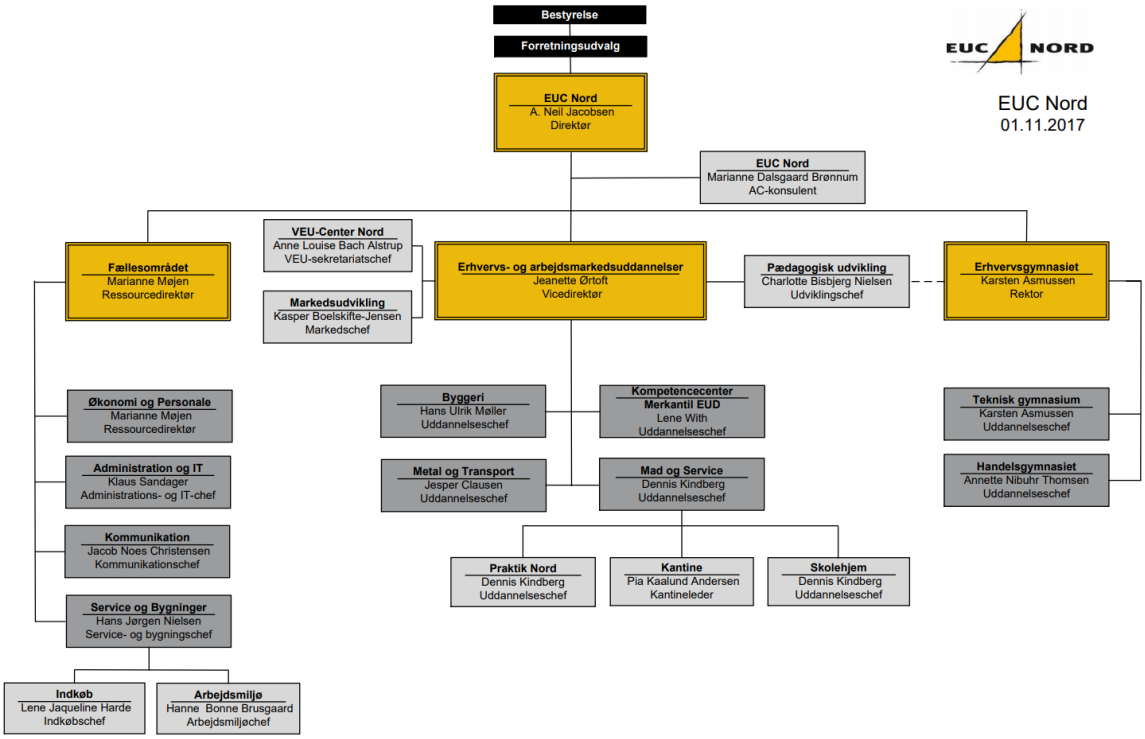
I virksomhedsanalysen bliver virksomhedens karakteristik gennemgået og beskrevet for at få et bedre overblik over hvem systemet skal udvikles til og derved give mulighed for at fokusere på det relevante for virksomheden. EUC Nord er en virksomhed, der er en model for strukturen på systemet til fordel for at kunne lave opgavestyringssystemer til lignende virksomheder. Dette uddybes i Iværksætteri afsnittet senere i rapporten.

5.1. EUC Nord som virksomhed

EUC Nord er en selvejende institution under Ministeriet for børn, undervisning og ligestilling som derfor har egen bestyrelse [3]. Organisationen er spredt på i alt ni adresser i Nordjylland hvor hoveddelen af dem ligger i Hjørring og Frederikshavn. I 2014 havde institutionen en omsætning på 268 millioner kroner. Institutionen kan opdeles i forskellige hovedområder som er delt i tre søjler, en for Erhvervs- og Arbejdsmarkedsuddannelser der har med tekniske og merkantile erhvervsuddannelser samt efteruddannelser, en for Erhvervsgymnasiet der består af en HTX- og en HHX-uddannelse og en søjle for fællesområdet som indeholder forskellige stabsfunktioner. Der er i alt 410 ansatte fordelt på virksomheden hvor 17 af dem er ansat i Bygnings- og serviceafdelingen, hvilket er den del af EUC Nord dette projekt beskæftiger sig med. Institutionen har interesse i at tilbyde uddannelser og kurser i høj kvalitet med fokus på erhvervslivet.

5.2. Organisations analyse

Gennem organisationsdiagrammet, på figur 2, kan det ses at organisationen har maks. tre horisontale arbejdsniveauer hvis bestyrelse og forretningsudvalget undtages. Det maximale kontrolspænd der kan forekomme, er fire. Dette medfører at virksomhedens ledere skal kunne holde overblikket når der skal koordineres. Som nævnt tidligere i afsnittet 5.1 har Fællesområdet stabsfunktionalitet for hele institutionen, hvilket resulterer i at dele af organisationen er decentraliseret og medarbejdere kan kommunikere på tværs af linjer, dog tager direktørerne, bestyrelsen og rektor stadigvæk beslutningerne der har relevans for institutionens helhed. Større opgaver i de forskellige afdelinger skal igennem en leder eller direktør, dette gennemgås i afsnit 6.3. Dette har den fordel at opgaver op til en vis størrelse bliver løst fleksibelt, mens at der er en stærk helhedsorientering når større beslutninger bliver taget. Der kan derfor argumenteres for at organisationsformen ifølge Mintzberg, er fagbureaukratisk [1]. Dette uddybes i afsnit 5.3 omkring virksomhedskultur og ledelse.



Figur 2: Viser EUC Nords organisations struktur [3]

5.3. Kultur & ledelse

Idet der skal indføres et system, er det relevant at se hvilket miljø som systemet skal indføres i og det er det, som vil blive uddybet herunder.

5.3.1. Virksomhedskultur

EUC Nord har lagt en klar deklarering af hvad deres mål for organisationen er på deres hjemmeside [3]. Disse er givet herunder.

- Udvikle unge og voksne fagligt, alment, personligt og socialt for derved at ruste dem til job og videreuddannelse på det private og det offentlige arbejdsmarked i både Danmark og i udlandet.
- Bidrage væsentligt til udvikling af lokalsamfundet og erhvervslivet.
- Sikre et bredt og relevant uddannelsestilbud i landsdelen.

Her tales der om de mål de har for deres studerende, men dette har også en betydning for deres ansatte og den kultur der ligger bag.

Baseret på Scheins kulturforståelse kan der gives en beskrivelse af den kultur der underbygger EUC Nord [1].

De værdier der underlægger organisationens kultur, kan være svære at gennemskue fra afstand. Dog kan der laves en vurdering baseret på de underliggende samfundsbundne **grundlæggende antagelser** der kendetegner uddannelsesinstitutioner i Danmark.

Disse antagelser kunne blandt andet være forståelsen for nutidens studiemiljø og hvordan dette organiseres både nationalt og internationalt. Hvordan sociale aktiviteter er en indforstået del af dette miljø og hvordan strukturen på institutionen angiver det ansvar ansatte har, overfor både ledere såvel studerende.

Baseret på disse antagelser kan **værdierne** for institutionen underbygges og sammenlignes med den vision og de kulturværdier EUC Nord selv reklamerer på deres hjemmeside [3].

Visionpunkter

- Et Spændende og udfordrende uddannelsesmiljø.
- En strategisk ressource for erhvervslivet.
- En attraktiv arbejdsplads.

Kulturpunkter

- Hvor vi alle udvikler sig.
- Hvor vi altid udviser respekt.
- Hvor vi kan være stolte af at være EUC Nord.

Her udviser de en vilje til at anerkendes som en sammenhængende virksomhed i samfundet med nogle klare mål både for ansatte og studerende samt deres selvopfattede kultur. Som Scheins model også beskriver er værdierne mere bevidste end de grundlæggende antagelser.

Alle medlemmer af institutionen har nogle klare opgaver der skal løses for, at denne vision kan realiseres, hvilket vil sige, at der er lagt en vis personlig og kollektiv ansvarsgrad på dem. Værdierne kan derfor tænkes som en form for krav, der er blevet indforstået når man arbejder og studerer.

Til sidst i Scheins model er der **artefakterne**, der er institutionens ydre udseende. Dette kan for eksempel være reklamation for at få nye studerende eller ansætte nye medarbejdere, skolens symbol der indgår i officielt materiale såsom eksamenspapirer og en tradition for at have skolefester eller lignende aktiviteter.

Med ovenstående beskrivelser kan der vurderes en kulturtype der passer ind i hvad både EUC Nord mener om deres kultur og hvad der er afgjort ud fra interviewet med repræsentanten. Der er her tale om en **Styret Missil** kultur ud fra Trompenaars og Hampden-Turners kulturtyper [1].

Kulturen er beskrevet ud fra en relativ lighed mellem de ansatte hierarkisk med et fokus på at løse en opgave. Dette kan genkendes i institutionens beskrivelse af dem selv og hvad de vil gøre for deres studerende og ansatte. Kulturtypen er dog primært beskrevet som at have tæt på intet hierarki, hvilket ikke gør sig helt gældende for EUC Nord. Her er der stadig et hierarki mellem ledere og ansatte og fra ansatte til studerende.

Det styrede missil er beskrevet som at hvert medlem har noget fagligt at byde ind med og der er derfor et fokus på projekter, som kan fremstilles som mindre eller større opgaver. Dette kunne eksempelvis være et samarbejde om et forsøg der kræver forskellige kompetencer. Loyaliteten er rettet mod det faglige og ikke for EUC Nord i sig selv. Dette giver mening da de beskriver, at de vil udvikle folk inde for fagområder og til at skabe nye jobs og hjælpe lokalsamfundet.

5.3.2. Organisation og Kultur

Som set i afsnit 5.2 er organisationsstrukturen bedømt fagbureaukratisk. Denne organisationsstruktur kan opstilles i forhold til den kulturtype EUC Nord er argumenteret til at have i forrige afsnit. Når de sammenlignes, kan der dannes et overblik over, hvordan denne opgaves løsningsforslag på problemformuleringen bedst passer ind i institutionen, der er det udviklede system.

I dette projekt beskæftiges der som sagt med Bygnings- og serviceafdelingen hos EUC Nord. Denne afdeling skal kommunikere med undervisere og ledere i andre afdelinger. Dette vil sige, at systemet skal tage højde for den individualistiske og opgaveorienterede kultur der omkringligger institutionen.

5.3.3. Ledelse

Ud fra forudgående samtale med repræsentanten fra Bygnings- og serviceafdelingen af EUC Nord blev det gjort klart, at lederen for afdelingen var mest interesseret i, at opgaverne blev løst effektivt og ikke nødvendigvis hvordan de blev løst. Dette stemmer overens med den individualistiske opgaveorienterede kultur gennemgået i afsnit 5.3.1. Dette er baseret på, at individer kan bruge midlerne de finder nødvendige for at løse opgaverne, så længe opgaven løses.

Lederne har dog stadig en forventning om, at opgaverne bliver løst og tjekker derfor også om det forløber ordentligt hos deres ansatte. Dette gør de ikke så ofte, men dette betyder så også, at det arbejde der laves dokumenteres ordentligt, så lederne lettest kan følge med. Dette er derfor også en nødvendig del af det udviklede løsningsforslag.

Derudover kan der nævnes magtdistancen mellem ledere og andre ansatte i forhold til organisationsstrukturen og kulturen. I dette tilfælde er magtdistancen kort, selvom der er en klar forståelse for hvem der er ledere og hvem der er pedeller eller andre ansættelses stillinger. Dette eksemplificeres i den

løse tilgang som lederne i Bygnings- og serviceafdelingen har til deres medarbejdere i forhold til opgaveløsning.

5.4. Forandringsledelse

Idet der skal indføres et system, er det vigtigt at kigge på hvordan det kommer til at påvirke virksomheden og undgå en stor omvæltning. Det er også nødvendigt at overveje virksomhedskulturen og ledelsesformen gennemgået i afsnit 5.3. Hvis forandringen strider imod organisationens **grundlæggende antagelser**, **værdie**r eller **artefakter** kan det have konsekvenser for hele virksomheden. Dog i dette tilfælde kan forandringen, som er et opgavestyringssystem have mindre konsekvenser for nogle medarbejdere og større konsekvenser for andre. Eksempelvis undervisere der nu skal omskifte fra kommunikation gennem e-mail til en applikation designet til formålet.

Når en virksomhed skal indføre en forandring som en af de ovennævnte, er der fire forskellige typer, som forandringen kan kategoriseres som. I EUC Nords tilfælde ville dette være en type 1 forandring [1], da det handler om, hvordan det nye system kommer til at forandre virksomheden. Der er ikke tale om omlægning af hele virksomheden, men bare en optimering af arbejdsprocessen i service og bygningsafdelingen, derfor kan det også antages, at risikoen er lille da det ikke direkte påvirker andre dele af virksomheden. Systemet vil dog påvirke kommunikationen mellem denne afdeling og resten af virksomheden. Her kan der opstå konflikter, hvis systemet bliver besværlig eller tidskrævende at bruge, derfor bør ligge stor fokus på, at det bliver så brugervenligt som muligt for de ansatte.

Ansatte kan have mange forskellige reaktioner på forandringer, som går fra engagement og støtte, der er i den positive ende af reaktionerne. I den anden ende kan der være aktiv modstand over for forandringerne, dette kan medføre bevidste negative handlinger fra medarbejderne og derfor vil det helst undgås.

Én måde at undgå de negative reaktioner er at være opmærksom på hvilket delsystem, der bliver ændret på og hvordan det påvirker de andre systemer ud fra Leavitts forandringsmodel [1].

Når der skal vælges en forandringsstrategi, er der fire forskellige hovedstrategier at vælge imellem. Det blev tidligere i afsnittet bestemt, at EUC Nord skal i gang med en type 1 forandring, dette bruges derfor til at vælge strategien. Ved en type 1 forandring vælges repræsentationsstrategien, hvor der er en person, der er repræsentant for brugerne af systemet. Der er en forventning om, at denne person har samme mål og værdier som resten af brugerne, idet repræsentanten står for at acceptere de tiltag, som der måtte være i forbindelse med systemet. Dette kan gøre, at systemet der bliver indført, passer bedre til brugerens brugsmønster og derfor undgås der en større omvæltning.

5.5. Delkonklusion og anbefaling

EUC Nord er en veletableret uddannelsesorganisation i Nordjylland, der bygger på at skabe et godt miljø for de studerende og samtidigt de ansatte. Dette er afbilledet i de ansattes fælles mål om faglighed, som kommer i første række. Der opleves en kort magt distance, men klar forståelse for hierarkiet.

Når et potentielt system skal indføres, anbefales det, at virksomheden er opmærksom på, hvad systemet skal ind og ramme og derfor hvilke dele af organisationen det kan påvirke. Især negative påvirkninger skal så vidst muligt undgås, idet de kan være skadelige for indførelsen af systemet, hvis de ikke bliver håndteret ordentligt. Her bliver det anbefalet, at systemet bliver indført via en repræsentant for brugergruppen og ikke strider imod virksomhedskulturens principper.

Systemet bør indføres, hvis virksomheden vil have bedre overblik, vedligeholdelse og støtte til undervisningen.

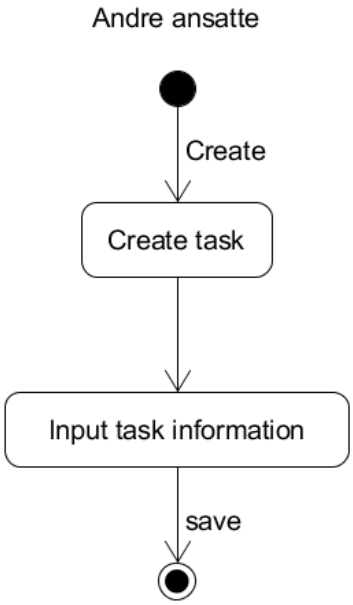
6. Krav specificering

I dette afsnit vil der udarbejdes hvilke krav brugere af systemet ville have til dets funktionalitet. Dette formuleres ud fra aktivitetsdiagrammer for deres arbejdsforløb. Disse aktivitetsdiagrammer er konsolideret information fra korrespondance med en ansat fra EUC Nord.

6.1. Aktivitets diagrammer

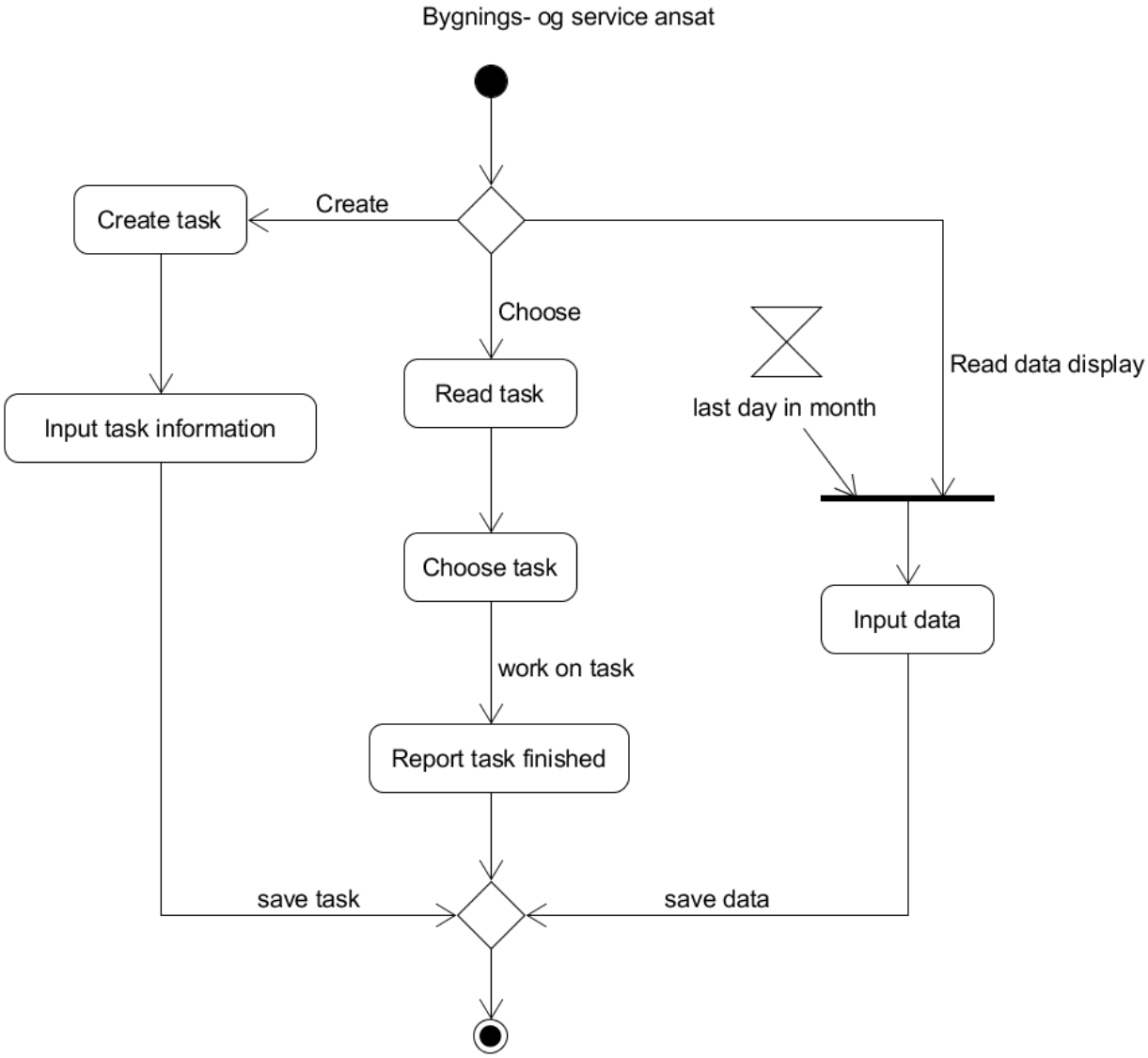
Aktivitetsdiagrammerne herunder er lavet ud fra en forespørgsels, hvor der blev spurgt til den overordnet arbejdsgang og blev lavet til at danne et overblik over problemstillingerne som de ansatte har.

På figur 3 modelleres arbejdsforløbet, når en opgave skal oprettes i systemet for en af de ansatte, der ikke arbejder i Bygnings- og serviceafdelingen. Den ansatte vælger at oprette en opgave og fylder derefter den påkrævede information ud og til sidst gemmer den ansatte og forløbet afsluttes. Aktivitetsdiagrammet viser også måden undervisere (herunder andre ansatte) kommunikerer med Bygnings- og serviceafdelingen da disse opgaver skal løses af afdelingens medarbejdere.



Figur 3: Viser et aktivitetsdiagram over de ansattes kommunikation med Bygnings- og serviceafdelingen.

I figur 4 er aktiviteterne, som en Bygnings- og serviceansat har. De ansatte her har også mulighed for at oprette en opgave, men noget de specifikt kan gøre, er at vælge at løse opgaver, som dem selv og andre har oprettet. De har også en tilbagevendende opgave som er tidsbestemt i forhold til en specifik dag i måneden.



Figur 4: Viser et workflow over en ansat i bygnings- og serviceafdelingen i forhold til opgaveløsning.

Aktivitetsdiagrammerne overskueliggører, hvilke opgaver de ansatte har og kan bruges til at lave medarbejder-opgave-mål tabeller, der går mere i dybden om, hvordan opgaverne bliver løst.

6.2. Medarbejder opgave mål tabel

Her ses medarbejder-opgave-mål tabeller opstillet for aktivitetsdiagrammerne.

Tabellerne er opdelt i fire kolonner. Første kolonne er medarbejder, hvor det kan ses, hvilken medarbejder der kan lave opgaven. Anden kolonne er opgave, den beskriver, hvad opgaven indebærer. Tredje kolonne er mål, hvor det er beskrevet, hvad opgaven skal løse, når den er færdig. Fjerde og sidste kolonne er Step i opgaven, som beskriver, hvordan den ansatte når målet for opgaven.

Medarbejder	Opgave	Mål	Step i opgaven
Service ansat og andre ansatte.	Opret opgave	Opgaven er oprettet	Brugeren vælger at oprette en opgave
	Input information til opgaven	Opgaven har den information den skal have og kan gemmes	Bruger vælger lokation og sted for opgaven. Angiver en prioritet og skriver en beskrivelse af opgavens indhold og gemmer.

Tabel 1: Viser hvilke opgaver andre ansatte har, samt ansatte i bygnings- og service afdelingen har.

Medarbejder	Opgave	Mål	Step i opgaven
Service ansatte	Vælger en opgave	Opgaven er valgt	Går ind i systemet og vælger en opgave, så den kommer i kalenderen
	Starter en opgave	Er startet på en opgave	Starter på den valgte opgave og går hen til det sted, hvor opgaven skal laves
	Færdiggør en opgave	Opgaven er blevet færdiggjort	Når opgaven er færdig går man ind på systemet og sætter opgaven som færdig
	Skrive data ind	Data er blevet skrevet ind	Aflæs måler der angiver data Skriver data ind i systemet og gem.

Tabel 2: Viser hvilke opgaver de ansatte i bygnings- og service afdelingen har.

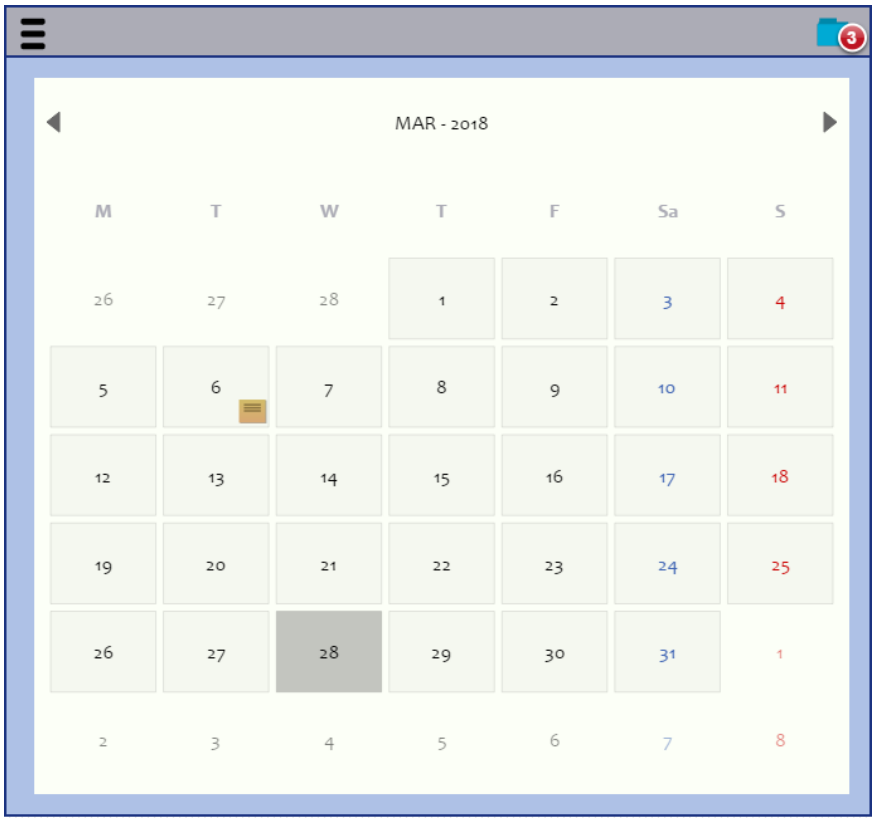
Disse tabeller giver et overblik over, hvordan opgaven bliver løst og hvornår den bliver løst. De er også blevet brugt til at lave en mock-up, som bliver uddybet i afsnit 6.3.

6.3. Mock-up & Interview

For at danne en bedre forståelse for problemdomænet er der blevet formuleret og foretaget et interview med en repræsentant (Pedel) fra Bygnings- og serviceafdelingen. Til at dirigere repræsentanten ind på hvad gruppen havde tænkt, i forhold til et systemforslag, blev der konstrueret et mock-up.

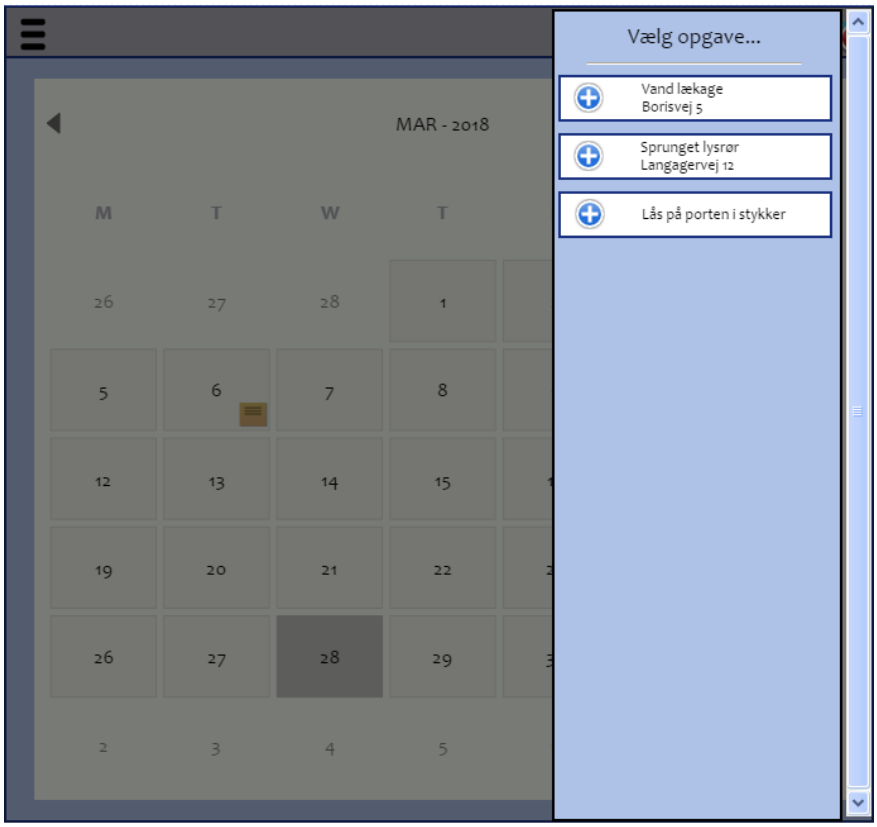
6.3.1. Mock-up

På figur 5 kan der ses det design, der blev tænkt som startsiden på brugerens program: et skema der viser hvilken måned det er, hvilken dag det er (fremhævet med grå) og med markering på om der er opgaver sat på en dag (fremhævet med en notesblok). Programmet er tænkt som en smartphone applikation, da det er yderst vigtigt, at det ikke skal være svært at benytte og dermed besværliggør processen til at organisere arbejdsdagen modsat en computerapplikation der er mindre mobil.



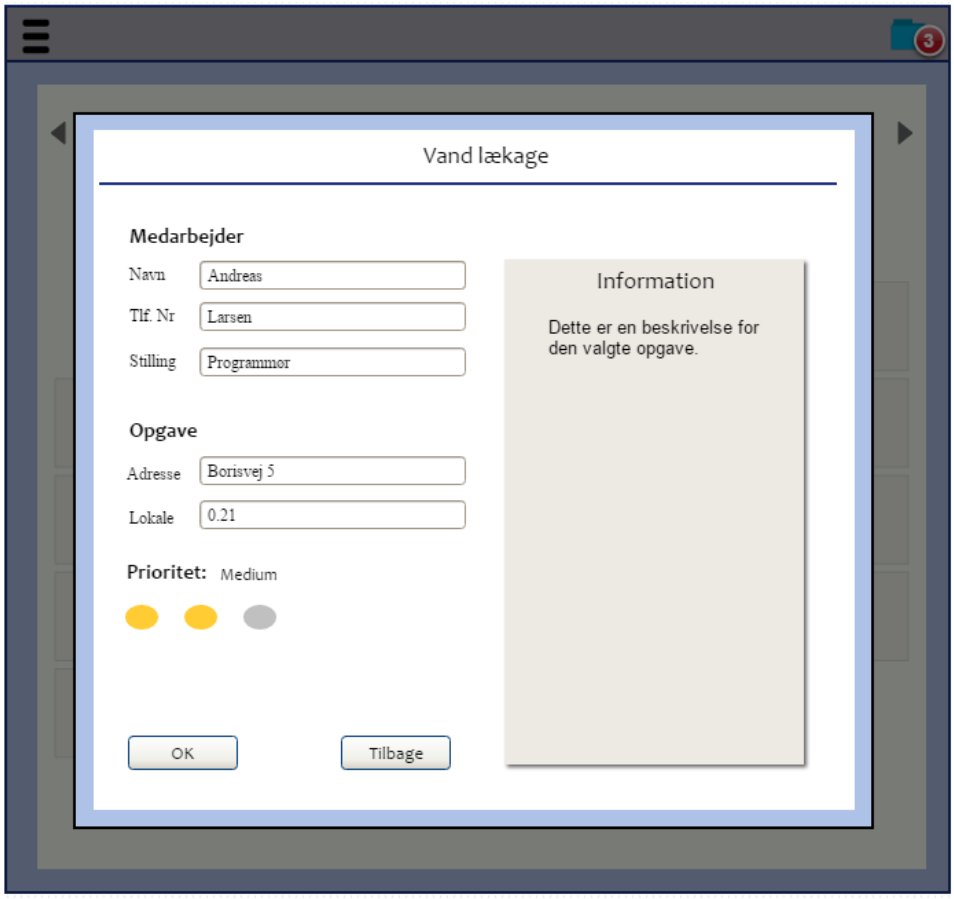
Figur 5: Viser hvordan startsiden kunne se ud når man åbner applikationen

Oppe i højre hjørne kan der ses en mappe med et rødt tal ovenpå. Tallet angiver, hvor mange nye opgaver er klar til at blive valgt af en pedel. Pedellen ville kunne trykke på dette og en side-menu ville lægge sig ovenpå skemaet, hvilket kan ses på figur 6.



Figur 6: Viser vinduet hvor det er muligt at vælge en opgave ud fra en liste af tilgængelige opgaver.

Menuen viser en række af opgaver, der er oprettet enten af brugeren selv eller af en anden bruger, såsom en lærer på EUC Nord. Pedellen ville kunne trykke på en opgave her og se yderligere information omkring opgaven eller få den sat på sit skema. Hver opgave på menuen har en titel og hvis bestemt, en adresse. Et vindue bliver åbnet, hvis brugeren trykker på en opgave, hvori informationer omkring opgaven vises, dette kan ses på figur 7.

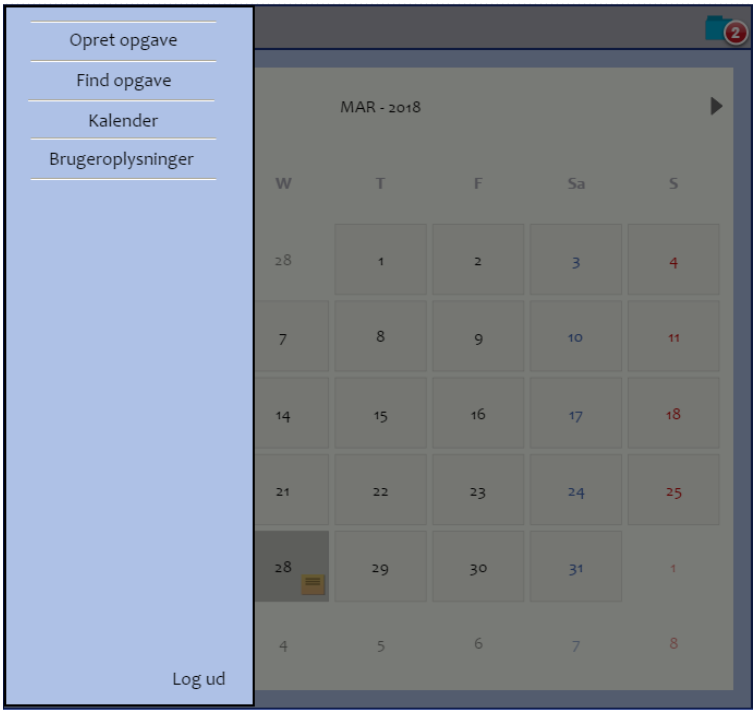


Figur 7: Viser en oversigt over en valgt opgave

Informationerne inkluderer hvem der oprettede opgaven, hvor det finder sted og hvor højt prioriteret opgaven er alt efter hvor kritisk det er at løse den.

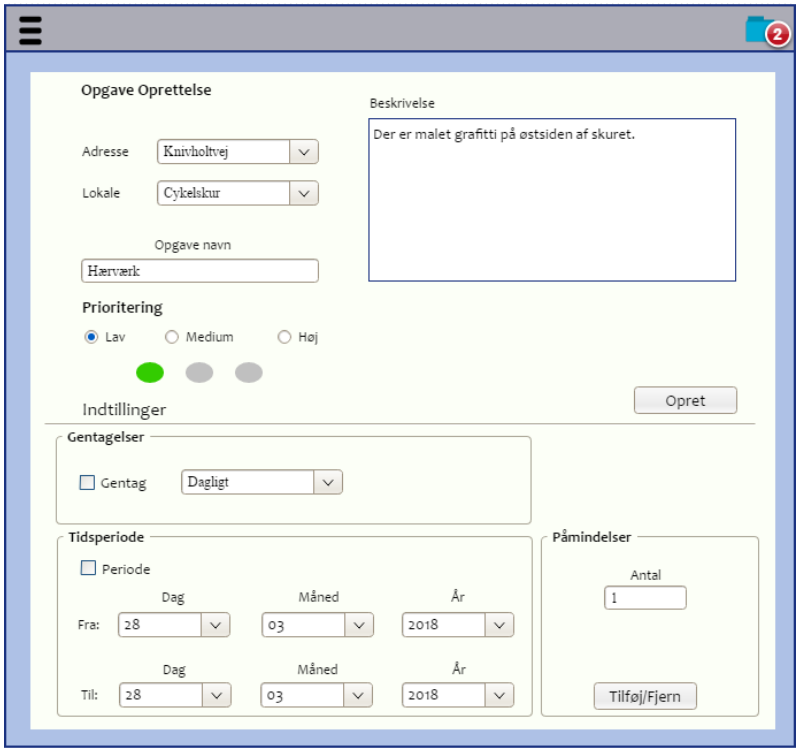
Brugeren kan også vælge at trykke på plustegnet i opgave-menuen for at vælge en opgave de vil udføre. Når en opgave er valgt, bliver skemaet igen vist og antallet af opgaver falder fra tre til to.

De tre bjælker øverst til venste på figur 5 er også en side-menu. Herfra får brugeren en række muligheder, hvori en af dem er oprettelsen af en opgave, set på figur 8.



Figur 8: Viser en menu som den ansatte kan vælge funktioner igennem.

Når de trykker på 'Opret opgave' feltet åbnes en ny side med en formular brugeren kan udfylde. Vinduet kan ses på figur 9.



Figur 9: Viser hvordan en bruger potentielt kunne oprette en opgave.

Her udfyldes informationerne omkring opgaven såsom adressen, titel og hvor højt prioriteret den skal være. Hvis brugeren vil, kan de angive om opgaven skal gentages, om den skal løses inden for en bestemt tidsperiode og om der skal sættes påmindelser på en opgave der udløses på nogle angivne tidspunkter til brugeren, der er sat på opgaven.

Det er kun nødvendigt at udfylde den øverste del af vinduet ovenover 'Indstillinger' overskriften for at oprette en opgave. Hvis de trykker 'Opret' bliver opgaven indsat i systemet og kan tages af personen, der oprettede den eller andre brugere af systemet der har mulighed for at løse den, hvilket et Login system ville sørge for. For at kunne skelne mellem brugertyper.

6.3.2. Interview

Med det ovenstående beskrevne mock-up blev der foretaget et interview med en repræsentant fra bygnings- og serviceafdelingen af EUC Nord. Formatet af interviewet var **semistruktureret** og derfor var der opstillet nogle generelle spørgsmål omkring virksomheden, hvilke systemer de benyttede ellers og hvordan arbejdsdagen normalt forløb for pedellen. Interview spørgsmål og svar kan findes i bilag [C].

I dette afsnit vil der blive gennemgået de mest væsentlige dele der blev afklaret ud fra interviewet og hvilken indflydelse det ville have på design og system.

Først blev der gennemgået hvordan en normal arbejdsdag forløb og hvilke særtilfælde der kunne have indflydelse på dem. Som start på dagen var der nogle generelle rutiner de skulle igennem såsom at tænde lys, hvorefter de tjekkede om der var kommet nye e-mails fra dagen før og om der var opgaver de manglede at løse. Ud fra dette planlagde de resten af dagen. De havde også mulighed for at udskyde en opgave, hvis en anden mere kritisk opgave opstod.

Efter de har været igennem deres sædvanlige rutiner, aftaler de med en gruppeleder og andre pedeller, hvem der løser hvad fra morgenen af. Her er pedellernes specielle kompetencer taget i betragtning, såsom elektricitet- og vand opgaver. Herefter udfører de opgaverne og tager hjem når de er færdige. De giver også besked igennem e-mail eller telefon når opgaven er løst, til personen der oprettede opgaven, hvis det er nødvendigt. Dette er vigtigt for det udviklede system, da denne proces skal automatiseres så en besked om færdiggørelse bliver sendt når opgaven fuldføres.

Et specielt ønske som repræsentanten klargjorde, var et notifikationssystem. Når en opgave var kritisk at få løst, havde en tidsperiode den skulle løses inden for eller bare en generel deadline, for eksempel en årlig græsslåning, ville de gerne have en notifikation på om datoen nærmede sig.

Disse var hovedproblematikkerne der var fundet ud fra interviewet, hvorefter de fik en gennemgang af mock-up.

6.3.3. Mock-up Gennemgang

Efter det generelle interview blev repræsentanten introduceret til det konstruerede mock-up, der blev gennemgået i afsnit 6.3.1. Her vil blive refereret til afsnittets figurer når nødvendigt for at fremvise eventuelle udviklinger repræsentanten foreslog.

Først var der skemaet som startsiden. Dette var en god ide, men brugeren ville have mere gavn af at have et skema der kiggede 5-7 arbejdsdage fremad fra den nuværende dag, da meget af arbejdet foregår ugentligt eller dag til dag.

Som set på figur 5 var der en notifikation på en dag, hvis der var opgaver. Her foreslog repræsentanten at benytte farverne for prioriteringen som set på figur 7 på dagene. For eksempel hvis der var en kritisk opgave, ville der være en rød tone på skemadagen. Repræsentanten mente dette kunne hjælpe til at hurtigt få et overblik over arbejdsdagen og minde dem om vigtige opgaver.

Når det kom til netop disse vigtige opgaver, foreslog repræsentanten også, at systemet gav en notifikationsmeddelelse når en opgave nærmede sig. Dette kunne for eksempel være en opgave der kun skal udføres når det er sommersæson. Her ville brugeren modtage en meddelelse fra morgenen af når de møder ind.

Ud over at kunne opsætte sine egne notifikationer kan brugeren også blokere dage de har afsat til andre vigtigere opgaver eller hvis de er på kursus, på ferie eller ikke er normalt på arbejde af andre grunde. Dette medfører, at en leder der uddelegerer opgaver ikke kan sætte opgaver på blokerede dage.

Medarbejderne har forskellige lokationer de arbejder på normalt, men hvis der skulle være problemer med for eksempel sygdom og lignende kan medarbejdere på én lokation tage opgaver fra en anden, hvis det skulle være nødvendigt. Dette betyder at en pedel skal kunne tage parate opgaver i systemet uden at skulle igennem en leder. Dette stemmer desuden overens med den individualistiske virksomhedskultur gennemgået i afsnit 5.3.

7. System Specificering

Systemspecificeringen vil gennemgå systemets specifikationer på et forholdsvis abstrakt niveau, hvor de generelle områder er dækket, såsom bruger, teknologien som systemet gør brug af, samt de funktionelle- og ikke funktionelle krav.

Dette afsnit er beskrevet ud fra de informationer der er fundet ud fra afsnit 5 og afsnit 6.

7.1. System vision

Systemets formål er overordnet at hjælpe de ansatte i Bygning- og serviceafdelingen på EUC Nord med at organisere deres hverdag og de arbejdsopgaver, der kan være i den. De vigtigste områder er at kunne organisere deres opgaver, som på nuværende tidspunkt ofte bliver organiseret verbalt. Systemet skal for underviserne, i de andre afdelinger af institutionen, give mulighed for at kunne kontakte Bygning- og serviceafdelingen uden, at det forstyrrer og er mere tidskrævende, end det nuværende er.

7.2. Interessenter og brugere

Ud fra afsnit 5 kan der argumenteres for, at interessenterne er EUC Nords ledelse, som består af en bestyrelse og en række direktører. De har alle en interesse i, at systemet kommer til at effektivisere organiseringen af arbejdsrutinerne.

Brugerne af systemet er først og fremmest pedellerne, der skal klare opgaver for de andre brugere af systemet nemlig andre ansatte. De andre ansatte vil nemmere kunne kommunikere med pedellerne og undgå tidsspilde på opgaveløsninger, der er afhængig af begge parter. Da den primære grund til at indføre et system som dette er som sagt at gøre arbejdsgangen mere effektiv. Derfor har de andre ansatte end pedellerne også interesse for systemet.

Systemet har potentiale for videre salg og der kan argumenteres for, at systemet er en generel skabelon, der kan benyttes af mange forskellige firmaer. Derfor kan systemet også være interessant for private firmaer og ikke kun offentlige organisationer lignende EUC Nord.

7.3. Systemets teknologi

Systemet bliver udviklet i sproget Java og som udgangspunkt til computer, dog er der en forhåbning om, at det også skal på smarttelefoner i fremtiden, idet dette er noget repræsentanten fra Bygning- og service afdelingen har efterspurgt. En anden grund til at få det på smarttelefoner er for at øge brugervenligheden i forhold til tilgang til systemet.

7.4. Krav

7.4.1. Funktionelle krav

De funktionelle krav er krav for, hvad systemet skal kunne gøre for kunden og er formuleret ud fra afsnit 6.1, 6.2 og 6.3.

- Opret opgave
    - Opretning af opgaver med relevant information der gemmes kontinuert både for Bygnings- og serviceafdelingen og andre ansatte.
- Vælg opgave
    - Få opgaver i systemet vist og muligheden for at vælge de opgaver der er oprettet i systemet.
- Afslut opgave
    - Valgte opgaver skal kunne afsluttes, når den ansatte har færdiggjort den hvorefter den ikke er let tilgængelig mere.
- Slet opgave
    - Mulighed for at slette opgaver der ikke er relevante eller fejloprettet.
- Indberetning af data
    - Indberetning af data fra de forskellige aflæsninger der er, såsom vand og varme.
- Visning af data
    - En visning af dataene der er blevet indberettet.
- Notifikationer
    - Beskeder til brugeren når der er kritiske opgaver eller informationer.
- Håndtering af ansatte
    - Opret, vis, opdater og slet ansatte i systemet.
- Håndtering af lokationerne
    - Opret, vis, opdater og slet lokationer i systemet.

7.4.2. Ikke-funktionelle krav

De ikke-funktionelle krav er krav som brugere og interessenter forventer, men ikke nødvendigvis specifikt har informeret om. De handler om, hvordan systemet opfattes gennem brugsmønstre.

- Systemet skal være brugervenligt og intuitivt.
    - Systemet skal informere brugeren på en klar måde, hvis der sker noget uregelmæssigt.
- Systemet skal være let læsbart.
- Systemet skal være overskueligt.
- Systemet skal kunne vedligeholdes.

- Systemet skal være sikker
    - Systemet skal ikke lægge informationer om virksomheden og burde ikke nemt kunne tilgås af andre end systemets brugere.
- Systemet skal have en lav responstid.

8. System analyse og design

I dette afsnit påbegyndes analysedelen af systemudviklingen. Der konstrueres et use case diagram, hvor relationen mellem arbejdsopgaver og ansatte opstilles. Herefter laves der brief og casual beskrivelser af use casene og en prioritering af dem baseret på tre kriterier. Herefter gennemgås kandidatklasse diagrammet for domænet og hver iteration for modellen. Det samme gøres for den relationelle model og herefter gennemgås de to use cases der er valgt og implementeret i det færdige system. Henholdsvis Create Task og Choose Task. Kapitlet afsluttes med Design Klasse diagrammet.

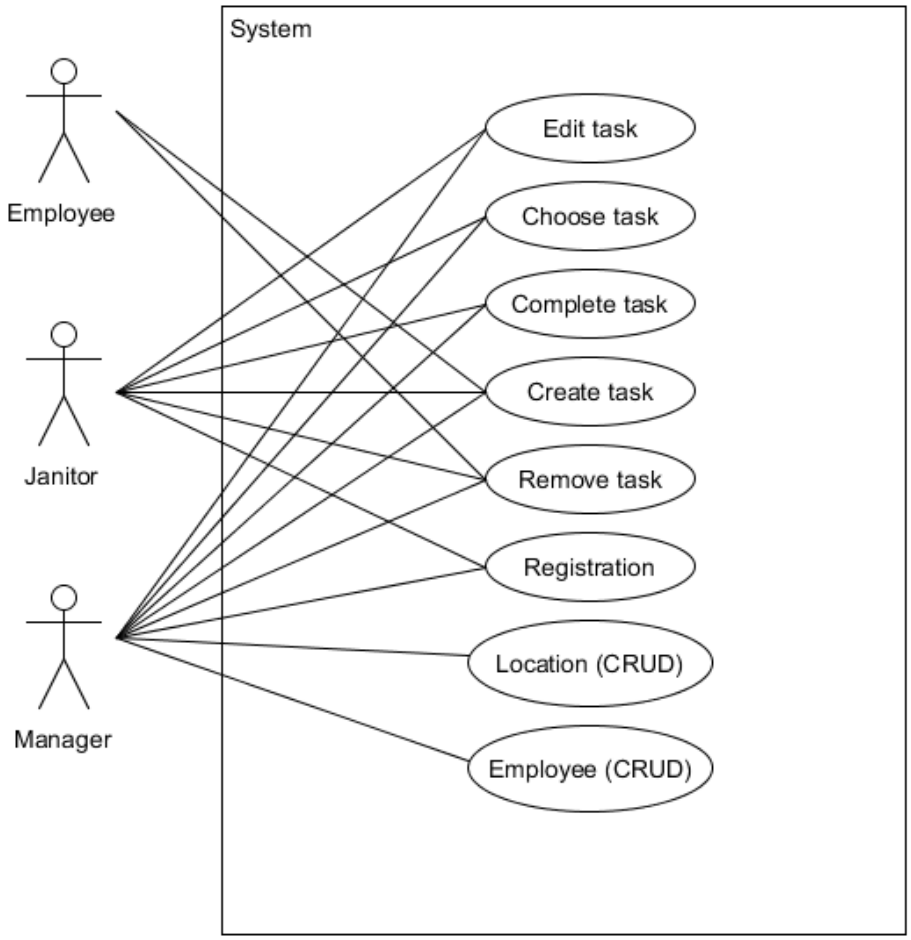
8.1. Use case diagram

Her blev use case diagrammet opstillet for at overskueliggøre hvilke aktører, der gennemgår hvilke use cases.

Use casene er blevet udvalgt fra medarbejder-opgave-mål tabellen samt informationen, der blev indsamlet fra interviewet med repræsentanten gennemgået i afsnit 6.3, kan der opstilles use cases.

Der er tre forskellige aktører der kommer til at gøre brug af systemet.

- Employee
    - Ansatte hos EUC Nord der ikke er ansat i Bygnings- og serviceafdelingen.
- Janitor
    - Pedeller der er ansat i Bygnings- og serviceafdelingen.
- Manager
    - Ledere der er ansat i Bygnings- og serviceafdelingen.



Figur 10: Viser et overblik over de use cases som aktørene kan gennemgå.

Beskrivelser af de forskellige use cases kan findes i næste afsnit.

8.2. Brief og casual use case beskrivelser

Use case beskrivelserne i dette afsnit er af to typer. Brief som består af en beskrivelse af use casen der foregår som forventet. Den anden type er casual som er når opgaven forløber som forventet med alternative retninger [2].

***Edit task*:**

En bruger vil gerne ændre i en opgave og vælger derfor opgaven i systemet, retter den med de nye oplysninger og gemmer den i systemet.

***Choose task*:**

En bruger vil gerne arbejde på en opgave og går derfor ind i systemet og vælger en opgave fra en liste af ikke-påbegyndte opgaver og opgaven er nu påbegyndt af den givne bruger.

***Complete task*:**

En bruger har færdiggjort en påbegyndt opgave og går derfor ind i systemet for at sætte opgaven som færdiggjort.

***Delete task*:**

En bruger vil gerne slette en opgave, der er blevet oprettet. Brugeren finder opgaven og sletter den.

***Create task*:**

En bruger vil oprette en opgave i systemet. Brugeren indtaster oplysningerne på opgaven og vælger hvilke ting der er relevante for opgaven. Brugeren trykker derefter på opret. Opgaven er nu oprettet og kan findes i listen over ikke-påbegyndte opgaver.

- Hvis brugeren er af en bestemt type, kan de sætte en gentagelse, en tidsperiode og en påmindelse på opgaven. Hvor andre type af brugere ikke kan.

***Registration*:**

En bruger får besked om, at der skal registreres nogle oplysninger og bruger derfor systemet til at indtaste dem. Bruger kan nu finde oplysningerne i systemet.

***Location*(*CRUD*):**

Brugeren vil oprette en lokation. Brugeren opretter en lokation med de relevante oplysninger.

Brugeren vil se en specifik lokation. Brugeren søger efter den lokation, med dennes lokationssøgekriterier.

Brugeren vil opdatere en lokation. Bruger søger først efter lokationen og efter at have fundet den, vælger den og ændre den med de nye oplysninger.

Brugeren vil slette en lokation. Brugeren finder lokationen og vælger at den skal slettes.

***Employee*(*CRUD*):**

Brugeren vil oprette en medarbejder. Brugeren opretter en medarbejder med de relevante oplysninger.

Brugeren vil se en specifik medarbejder. Brugeren søger efter den medarbejder, med de medarbejder søgekriterier.

Brugeren vil opdatere en medarbejder. Bruger søger først efter medarbejder og efter at have fundet den, vælger den og ændre den med de nye oplysninger.

Brugeren vil slette en medarbejder. Brugeren finder medarbejderen og vælger at denne skal slettes.

Med alle usecases beskrevet er der en bedre forståelse af hvad de indebære og de kan nu prioriteres.

8.3. Prioritering af use cases og vigtige funktioner

Inden en iteration startes er det vigtigt at vælge den mest relevante use case eller vigtige funktion at arbejde med. Dette kræver en form for prioritering af de mulige use cases. Her bruges tre kriterier; risk, coverage og criticality [2] og deres beskrivelse kan findes i figur 11.

- **Risk** includes both technical complexity and other factors, such as uncertainty of effort or usability.
- **Coverage** implies that all major parts of the system are at least touched on in early iterationsperhaps a "wide and shallow" implementation across many components.
- **Criticality** refers to functions the client considers of high business value.

Figur 11: Larmans beskrivelse af de tre prioterings kriterier.

I tabel 3 har de forskellige use cases fået et tal mellem 1-10 ud fra, hvor vigtig de er i forhold til hvert kriterie. Tallene bliver multipliceret sammen og får en samlet værdi. Med mindre at der er en speciel grund til, at en use case skal laves før andre bliver den med den højeste samlede vurdering lavet først. Dette afgør derfor, at det er Opret opgave, der skal laves som den første.

Use case navn	Risk	Coverage	Criticality	Samlet vurdering
Opret opgave	7	10	10	700
Vælg opgave	6	4	10	240
Færdiggør opgave	2	5	10	100
Login (Hjælpefunktion)	4	2	7	56
Opdatere opgave	4	3	4	48
Registrering	3	2	6	36
Slet opgave	2	3	4	24
Medarbejder(CRUD)	2	2	2	8
Lokation(CRUD)	2	2	2	8

Tabel 3: Viser prioteringer af use cases.

8.4. Kandidatklasser

I dette afsnit vil der blive lavet kandidat klasser, da dette giver et overblik over, hvilke mulige klasser og de tilhørende attributter der kunne være i domænemodellen.

Candidate	Evaluation	In/Out
Employee	A person employed and can use the system.	In
Login	Login system to determine access in the system.	In
Task	A task that an employee can complete. Can be very specific.	In
Calendar	The setup of a calendar that can accept tasks and other information related to them.	In
Access	Information on who has access to what location(s).	Out
Notification	A message that notifies an employee about a task that has to be completed.	In
Information	A collection of relevant information for the employees such as manuals and location information.	Out

Location	Information on each location.	In
Equipment	A piece of equipment assigned with a location and what tasks it can be used for.	Out
PlaceOfWork	The different work locations for each employee. Regional.	Out
Inventory	Information on furniture and appliances separate from Equipment.	Out
Speciality	Information on what an employee is specifically proficient in.	Out
Booking	Booking a worker on a location for a task.	In
Delivery	The transportation of an item or items to a location from another.	Out
Freight	Information on wares that need to be transported to a location.	In
Vehicle	A vehicle that the employees can use for tasks. Only one can be used at a time.	Out
TimePeriod	Specification on when a task can/has to be completed. Intervals and periods.	In
Agreement	A new agreement with an institution that needs the company using the system for new tasks.	Out
Request	The call in of an expert in a field for a difficult task.	In
Room	Room information that can be used for a location with a task.	In
TaskType	Specification of a task. Here other classes can be applied depending on the type.	In
Order	Ordering of a piece of equipment or a spare part(s).	Out
Budget	The Budget. An employees maximum funds for parts. If more is needed they need permission.	Out
Reporting	Report on a completed task.	Out
RecordUtility	Reads on apparatus on a location. Will conjure statistics compared to previous data collected. Stats could be electricity use.	Out
Priority	Priorities for tasks.	In
Schedule	To organise the tasks and employees time.	Out

Tabel 4: Kandidatklasser til systemet ud fra use casene

Kandidat Klasserne er lavet ud fra de oplysninger, som blev fundet frem til i de tidlige udvekslinger med repræsentanten, den samme person der blev interviewet som gennemgået i afsnit 6.3. Ud fra de oplysninger blev der fundet frem til en del forskellige klasser der muligvis kunne findes i domænemodellen. Der er en del af disse klasser som står som 'In', men grundet senere interview og flere oplysninger, så blev disse klasser

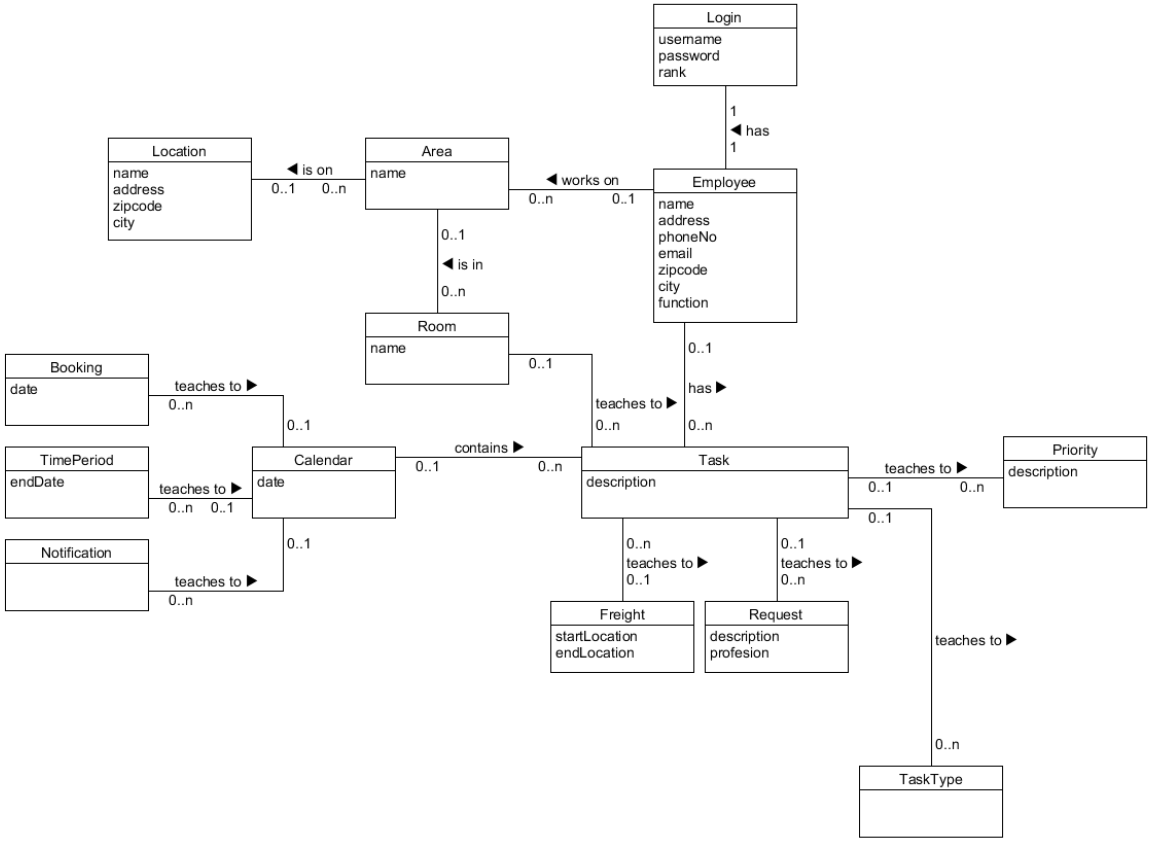
fjernet igen og en enkelt blev tilføjet, som der blev overset i første omgang. Udviklingen af domænemodellen fremgår i næste afsnit.

8.5. Domæne model

Domænemodellen er opbygget hovedsageligt i tre iterationer, dog med små ændringer undervejs for at rette op på fejl eller mangler. Modellen kortlægger virksomhedens domæne i systemrelaterede sammenhænge og giver et visuelt overblik over kandidatklasser og deres relationer til hinanden.

8.5.1. Iteration 1

Den første iteration af domænemodellen er opbygget efter kandidat klasse tabellen, så der kunne skitseres en model ud fra de tidlige oplysninger, hvilket også gør, at denne ser meget anderledes ud end de senere modeller.

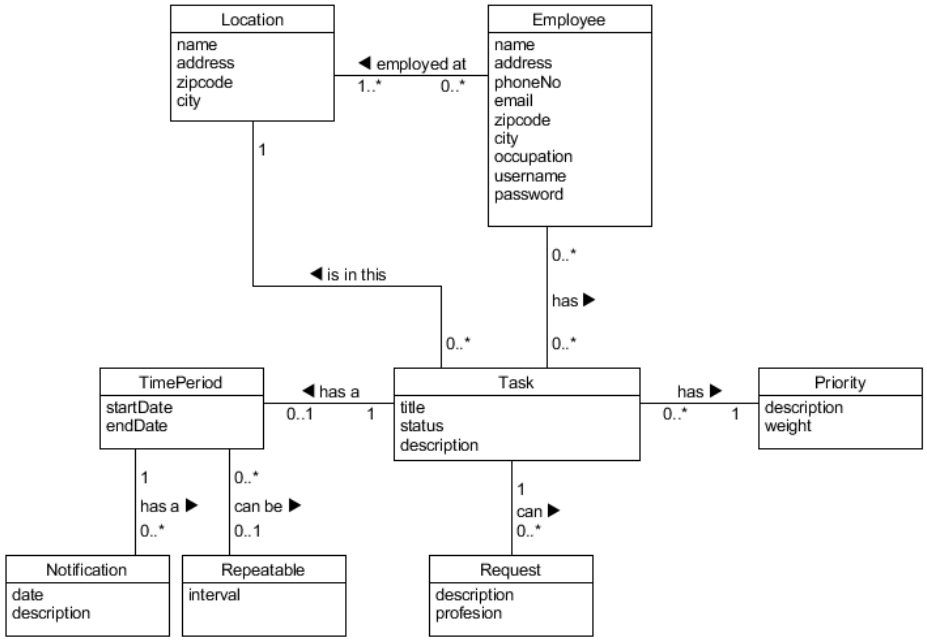


Figur 12: Første iteration af domænemodellen

Figur 12 har mange klasser med kun en lille mængde af nødvendige attributter, for at der blev et bedre overblik over systemet, samt have en mulighed for bedre at kunne visualisere systemet og lave et klart udgangspunkt for udvikling af dette system.

8.5.2. Iteration 2

Efter interviewet gennemgået i afsnit 6.3 med repræsentanten, var informationerne klargjort yderligere i forhold til en mulig implementering, samt hvad der ville være vigtigt for EUC Nord. I starten af anden iteration blev domænemodellen derfor omstruktureret en del, til hvad kan ses på figur 13. Denne iteration bragte domænemodellen tættere på en mulig implementering.

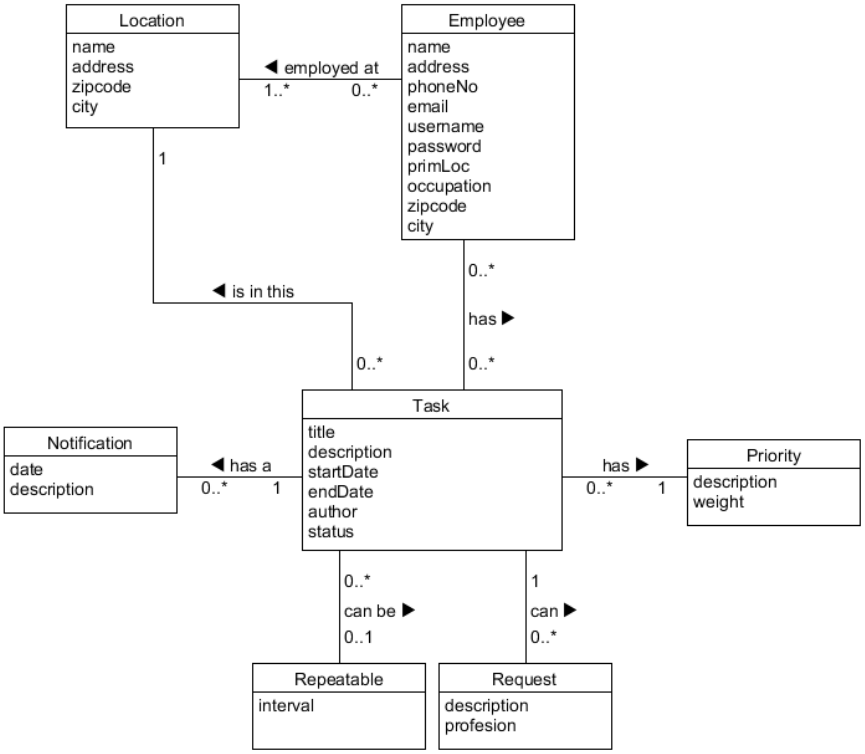


Figur 13: Anden iteration af domænemodellen

På figuren er de største ændringer siden første iteration at 'Location' har overtaget alt form for information vedrørende lokation. 'TimePeriod' har overtaget 'Calender', da der blev fundet ud af, at en kalender egentlig kun er en visuel repræsentation af noget givent information, som en GUI ville kunne tage sig af at vise.

8.5.3. Iteration 3

I tredje iteration af domænemodellen, blev 'TimePeriod' klassen fjernet, da den efter systemmæssige overvejelser ikke længere var relevant som en klasse. Det førte til, at denne blev fjernet fra diagrammet og programmet, samt blev alt sammenhæng og information lagt over på 'Task' i stedet. Der er ikke blevet fjernet information, disse er kun blevet flyttet rundt på. Der blev samtidig også fundet ud af, at der var et par få attributter der manglede, så disse blev også tilføjet. Modellen kan ses på figur 14.

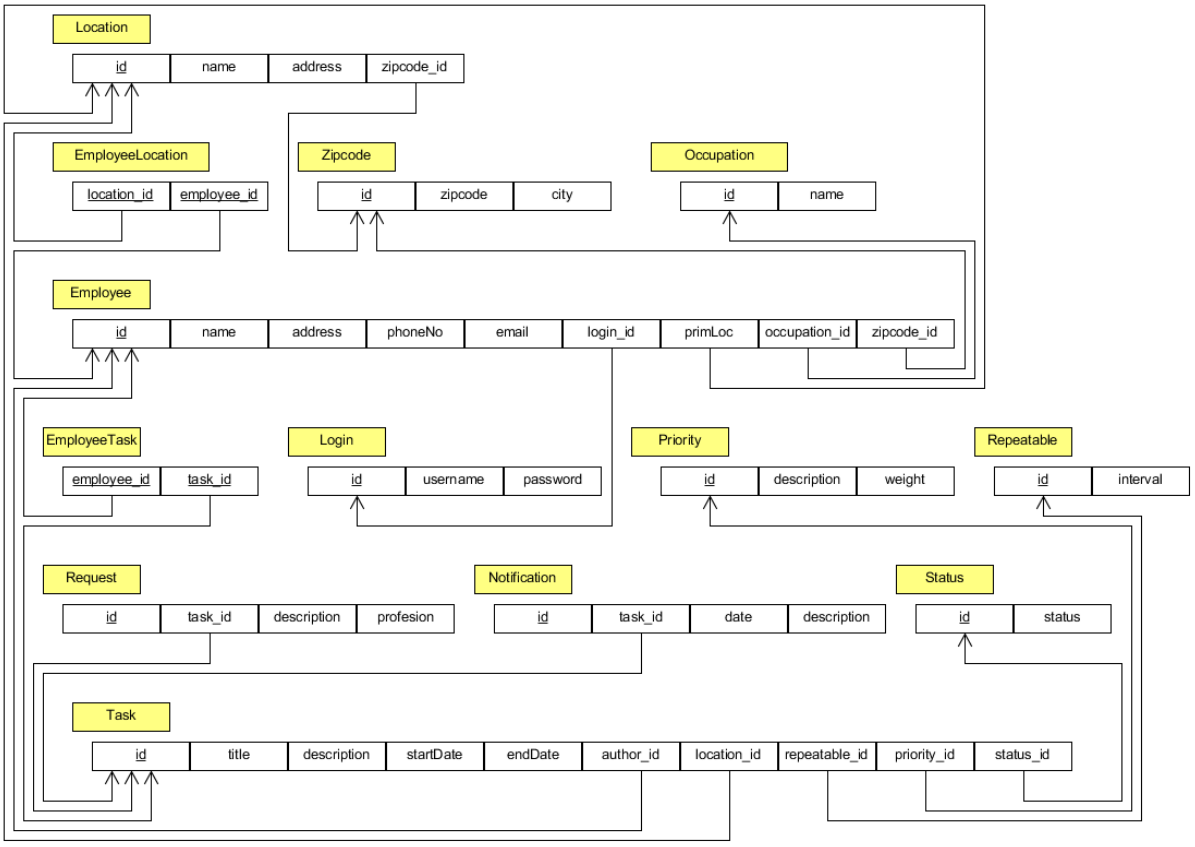


Figur 14: Tredje iteration af domænemodellen

8.6. Relationelle model

Den relationelle model er en modellering af domænet, som det skal opstilles i databasen. Den bliver brugt til at overskueliggøre opsætningen af databasen.

Den relationelle model er bygget op i iterationer parallelt sammen med domænemodellen. Den blev først udarbejdet i anden iteration som en skitsering, men idet domænemodellen ændrede sig blev den først færdiggjort i tredje iteration, da der her blev gået i gang med Create Task use casen og det var i forbindelse med den, at databasen skulle opsættes i Microsoft SQL. Modellen kan ses på figur 15.



Figur 15: Den færdige udgave af den relationelle model

Figur 15 viser den færdige og matchende relationelle model til den færdige domænemodel. I den tidligere iteration var den relationelle model lavet så den overholdte Boyce-Codd Normal Form (BCNF), hvilket betyder at alle unikke attributter skulle være primære nøgler, men der blev taget et valg, hvor der vil kunne blive ændret på alle former for data, hvis det blev nødvendigt. Alle tabeller der ikke havde en 'composite key' har et id som bliver autogenereret af databasen med identity(1,1) som betyder, at alle tabellernes første tuple starter deres id med 1 og stiger med en per tuple, der bliver tilføjet.

For at man kan kalde databasen relationel skal den overholde en normalform.

Der kan argumenteres for, at 'Zipcode' tabellen ikke overholder N3. Idet 'city' attributten er afhængig af 'zipcode' attributten og derfor brydes N3. Det samme kan siges om Login tabellen som har password attributten, som er afhængig af 'username'. Dette blev først opdaget efter projekt iteration fire og er derfor ikke blevet ændret. Resten af modellen overholder dog N3.

8.7. Create Task - iteration 1

Create Task er use casen der blev implementeret i første iteration.

8.7.1. Fully dressed

Her vil der blive lavet en fully dressed beskrivelse over den højest prioriteret use case, som her er Create Task. Denne fully dressed beskrivelse giver et samlet overblik over en kompliceret use case og hvad der bliver gennemgået i denne use cases flow. Fully dressed beskrivelsen kan ses i tabel 5.

Use case navn	**Create Task**	
Aktører	Bruger	
Præbetingelse	En bruger er oprettet i systemet. Brugeren er allerede logget ind i systemet. Lokationer er oprettet i systemet	
Postbetingelse	En opgave er oprettet i systemet	
Frekvens	Max 20 / Dag	
Flow of events	**Aktør**	**System**
	Bruger starter en opgave oprettelse	2. Systemet opretter en opgave, samt tilføjer brugeroplysninger til opgaven
	3. Bruger angiver primær opgave oplysninger	
	4. Bruger gemmer opgaven	5. Opgaven bliver gemt med de angivne oplysninger. Og viser at opgaven er gemt.
Alternative flow	*. Under hele flowet skal opgaveoprettelsen kunne afbrydes	
	2a. Der kan ikke oprettes forbindelse til databasen, brugeren bliver informeret og flowet stopper.	
	3a. Den ønskede lokation findes ikke. Bruger kontakter administration for at få denne oprettet, flowet stopper.	
	3b. I tilfælde af at brugeren er af en bestemt type og ønsker at opgaven skal gentages, sættes en given gentagelse på opgaven. flowet fortsætter fra punkt 3.	

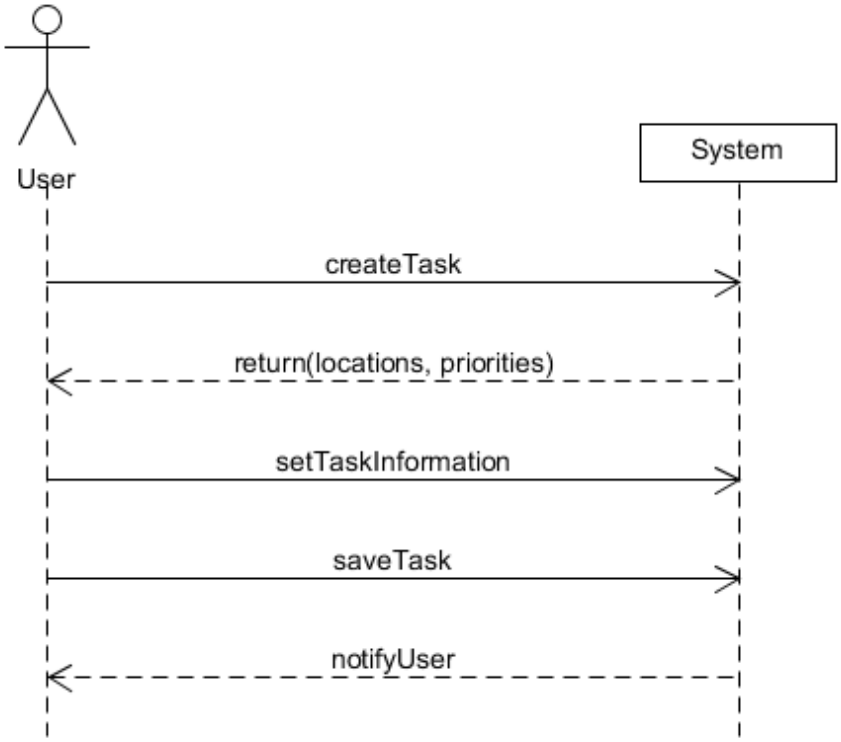
	3c. I tilfælde af at brugeren er af en bestemt type og ønsker, at opgaven skal have en tidsperiode, sættes en periode. flowet fortsætter fra punkt 3.	
	3d. I tilfælde af at brugeren er af en bestemt type og ønsker, at opgaven skal have en påmindelse, sættes en påmindelse. 3d. kan gentages, hvis flere påmindelser ønskes flowet fortsætter fra punkt 3.	
	5a. Brugeren har ikke opgivet et sted for opgaven og/eller ikke angivet en titel, så opgaven kan ikke oprettes før disse er udfyldt flowet går tilbage til punkt 3	
Specielle krav	Alt efter prioritet skal systemet give besked til en bestemt type bruger.	En bestemt type bruger har adgang til flere oprettelses oplysninger.
	Der skal gives titel, lokation og adresse samt adresse for at kunne oprette en opgave.	Systemet skal have nogle prædefineret prioriteter, som skal kunne sættes på en given opgave, så disse kan sorteres efter behov for udførelse.
	En bruger kan være en lærer, pedel eller ledere.	Alle brugere har et login som er tilknyttet dem, som bliver brugt til oprettelsen.

Tabel 5: Fully dressed af Opret opgave

8.7.2. SSD og Operationskontrakt

Her vil blive beskrevet system sekvens diagram for Create Task. Dette system sekvens diagram er blevet lavet ud fra fully dressed beskrivelsen for Create Task.

Systemet opretter en opgave og sætter informationer omkring brugeren, som er informationerne omkring den bruger som har oprettet opgaven. Efter dette bliver der sat informationer på opgaven, disse informationer er titel, prioritet og en beskrivelse af hvad opgaven omhandler. Opgaven bliver herefter gemt med de angivne oplysninger.



Figur 16: SSD for Opret opgave use casen.

Ud fra systemsekvensdiagrammet for Create Task, set på figur 16, er der blevet lavet en operations kontrakt for saveTask. Denne kontrakt er blevet lavet fordi den udfører flere ændringer i systemet ved det givne systemkald. Den kan også gøre det nemmere at lave interaktionsdiagrammet for use casen. Operationskontrakten kan ses på figur 17.

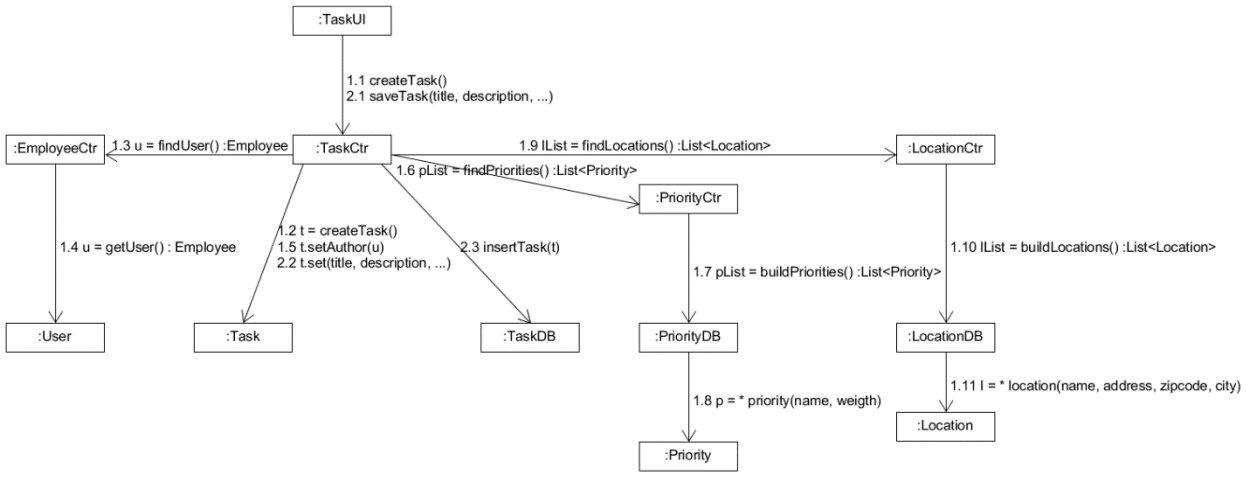
Operation: saveTask Use case: Create task
Precondition: - All information about a task must be included Postcondition: - A task with the information that the user has given such as title, priority and description is saved in the database

Figur 17: Operationskontrakt over opret opgave

8.7.3. Kommunikations diagram

Kommunikationsdiagrammet i figur 18 viser metodekald i systemet for use casen Opret Opgave. Der bliver taget udgangspunkt i SSD'en fra figur 16 og Operations kontrakten fra figur 17 for den her use case og der bliver ikke vist interne metodekald i klasserne.

Når en bruger vil oprette en opgave, bliver metoden *createTask()* kaldt på klassen **TaskCtr** som derefter kalder constructoren på modelklassen **Task**. Derefter returneres brugeren, der opretter opgaven fra klassen **User** samt de informationer, som brugeren kan vælge imellem. Når brugeren vælger at gemme bliver metoden *saveTask()* kaldt på klassen **TaskCtr** med alle informationerne som GUI'en har indsamlet og til sidst bliver opgaven indsat i databasen ved hjælp af metoden *insertTask(),* som bliver kaldt på klassen **TaskDB**.



Figur 18: Kommunikations diagram af Opret opgave

8.8. Choose Task – Iteration

Choose task er use casen der bliver lavet i anden iteration. Igen er denne use case blevet valgt ud fra prioriteringslisten fra tabel 3.

8.8.1. Fully Dressed

I anden iteration blev der lavet en fully dressed beskrivelse af den næst højeste prioriteret use case, som her er Choose Task. Use case beskrivelsen kan ses på tabel 6. Denne fully dressed beskrivelse blev lavet så der kunne ses, hvad der bliver gennemgået i dens main- og alternative flows, samt for at få et samlet overblik over en, hvad der blev bedømt til at være en kompliceret use case.

Use case navn	**Choose Task**	
Aktører	Bruger (Pedel, Leder)	
Præbetingelse	En bruger er oprettet i systemet. Brugeren er allerede logget ind i systemet. Lokationer er oprettet i systemet. Der er en opgave i systemet.	
Postbetingelse	En bruger har valgt en opgave som nu er tilknyttet dem. Status ændrer sig for opgaven.	
Frekvens	Max 20 / Dag	
Flow of events	**Aktør**	**System**
	En bruger vil vælge en opgave	2. Systemet fremviser aktive opgaver i prioriteret rækkefølge
	3. Brugeren vælger en opgave fra listen	4. Systemet fremviser udvidet information om opgaven
	5. Brugeren vælger at udføre opgaven og tilknytter den til sig selv	6. Systemet angiver starttidspunkt på opgaven, brugeren bliver tilføjet til opgaven. Opgavens status bliver opdateret til igangværende.
Alternative flow	1a. Brugeren vil vælge en allerede igangværende opgave Fortsætter fra punkt 2. 6a. Systemet tilføjer brugeren til opgaven flowet stopper	
	5b. Brugeren fortryder den valgte opgave og går tilbage til opgavelisten flowet fortsætter fra punkt 3.	

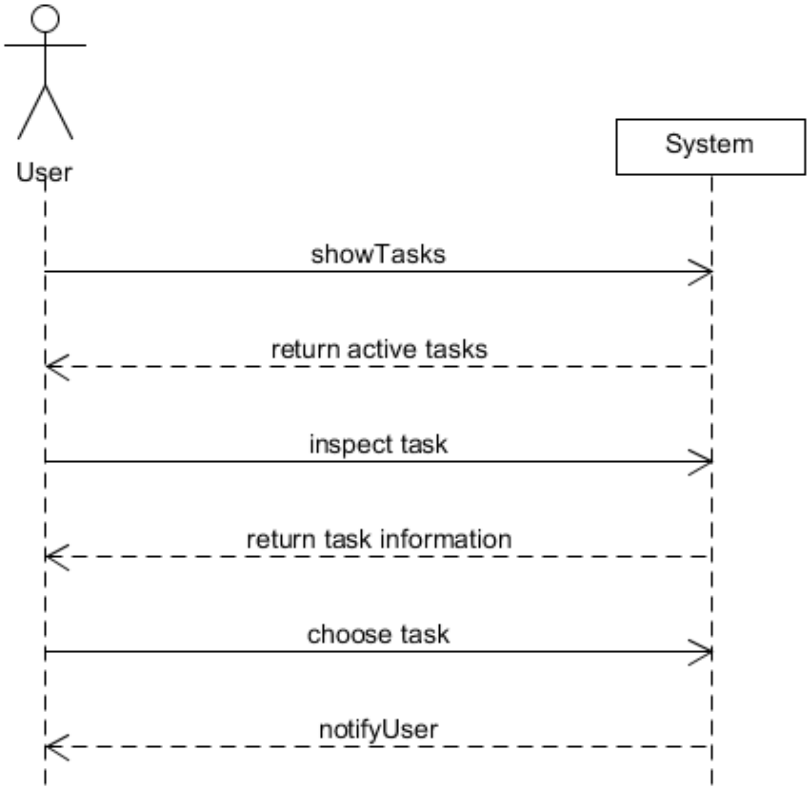
	2c. Der er ingen opgaver i systemet. Systemet meddeler dette til brugeren. flowet stopper	
	2d. Der kan ikke oprettes forbindelse til databasen flowet stopper	
	6e. Brugeren tilknyttes ikke opgaven og antallet af brugere på en opgave ændrer sig ikke flowet stopper	
Specielle krav	Brugeren skal være en pedel eller leder	

Tabel 6: Fully dressed af vælg opgave.

8.8.2. SSD

Her vil blive beskrevet system sekvens diagram for Choose Task. Dette system sekvens diagram er blevet lavet ud fra fully dressed beskrivelse for Choose Task.

Systemet viser en liste af opgaver i en prioriteret rækkefølge. Efter en opgave er valgt, fremvises de informationer, som er på opgaven. Herefter vælger en bruger opgaven, hvor opgaven for tildelt et starttidspunkt. Statussen for den valgte opgave ændrer sig fra at være aktiv til at være igangværende. System sekvens diagrammet kan ses på figur 19.

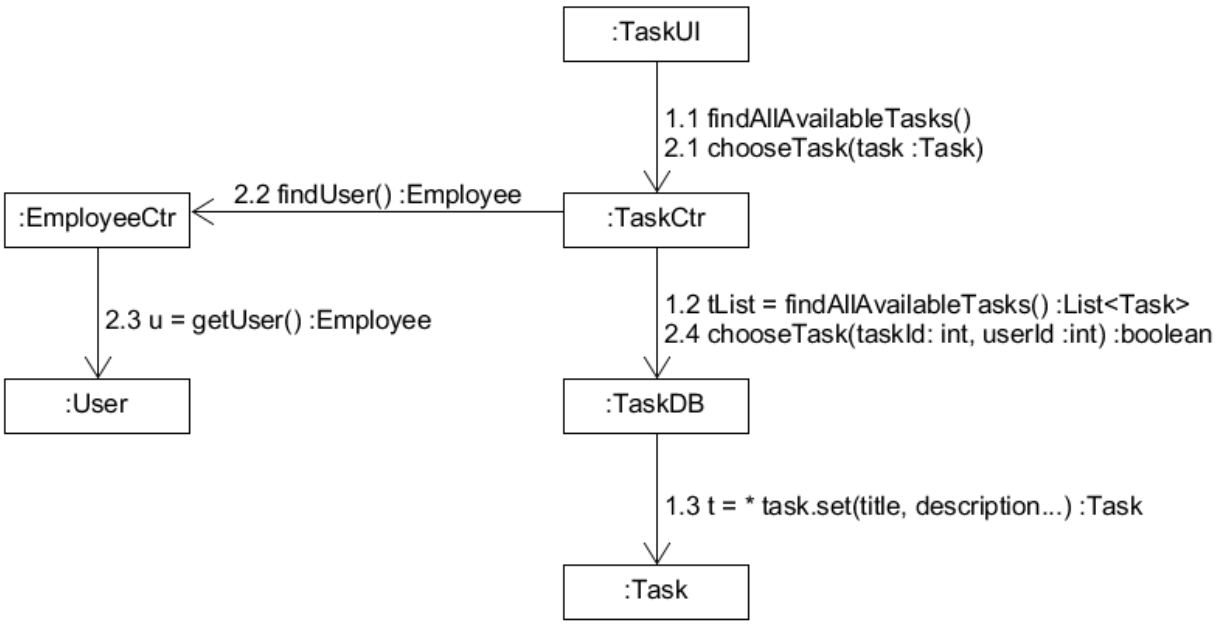


Figur 19: SSD over vælg opgave.

8.8.3. Kommunikations diagram

Kommunikationsdiagrammet viser metodekald i systemet for use casen Choose Task. Der bliver taget udgangspunkt i SSD'en fra figur 19 for den her use case og der bliver ikke vist interne metode kald i klasserne. Kommunikationsdiagrammet kan ses på figur 20.

Når brugeren vil vælge en opgave at arbejde på, returner systemet en liste af opgaver. Listen bliver fundet ved at kalde metoden *findAllAvailableTasks()* på klassen **TaskCtr**, som derefter kalder en metode ved samme navn i klassen **TaskDB** og bygger alle relevante Task objekter ud fra de rå datasæt i databasen. Når brugeren vælger en opgave, bliver metoden *chooseTask()* kaldt på klassen **TaskCtr**, som så returnerer brugeren fra klassen **User**. Herefter kaldes metoden *chooseTask(),* som har brugerens og den valgte opgaves database id som parameter, den bliver kaldt i klassen **TaskDB**, som opdaterer databasen med de nye oplysninger.



Figur 20: Kommunikationsdiagram over vælg opgave.

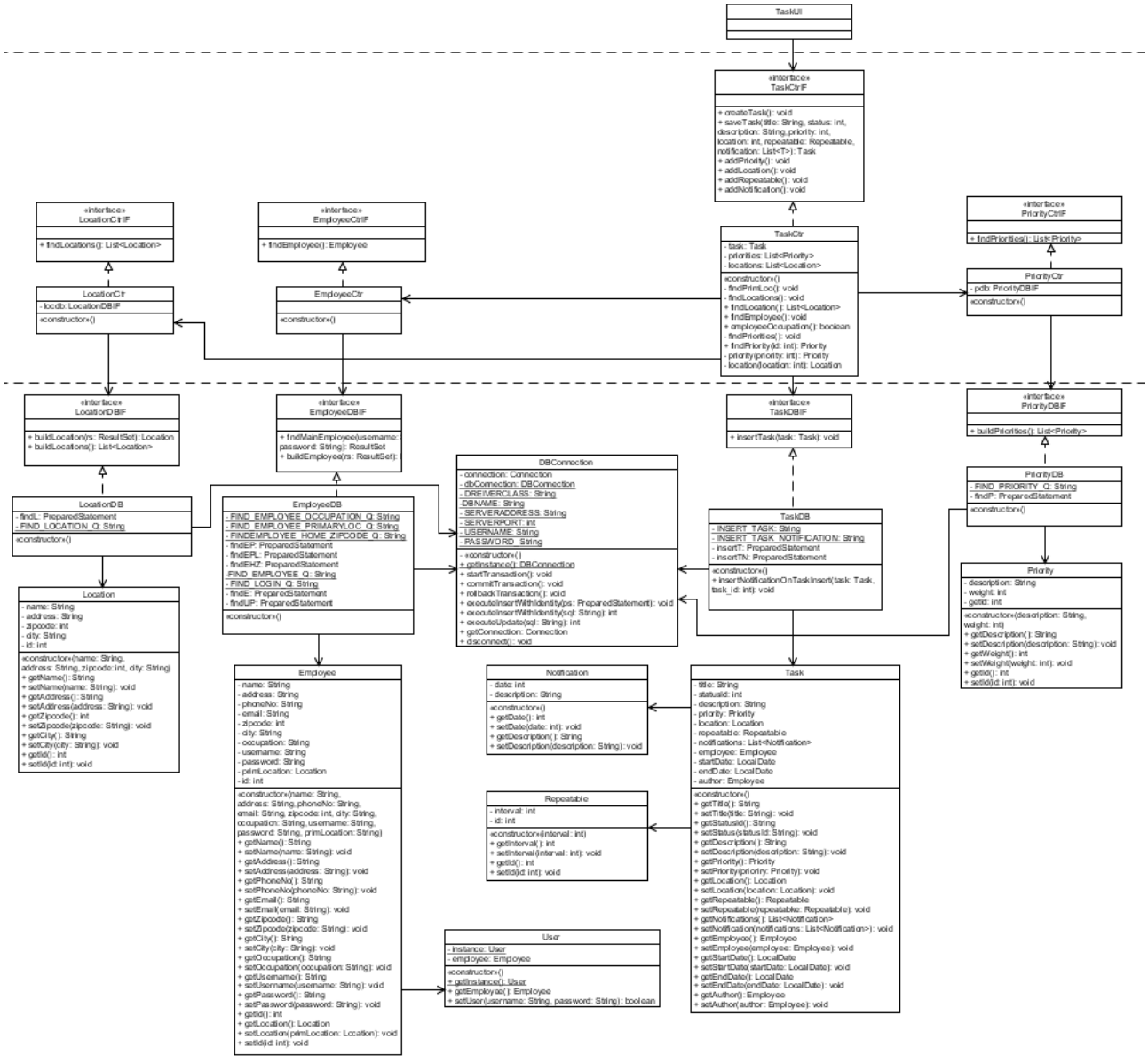
8.9. Design Klasse Diagram

Design Klasse diagrammet har alle informationer som klasserne i systemet indeholder samt hvilke klasser de tilhører. Diagrammet er en opstilling af programstrukturen og komplimenterer interaktions diagrammerne nå det kommer til implementationen. Diagrammet benyttes som udviklingsskabelon og giver muligheden for at få overblik over systemet for udviklere, der skal vedligeholde systemet.

8.9.1. Iteration 1

I figur 21 bliver første iteration af Design Klasse diagrammet vist. Det er indsat her for at give et overblik over, hvordan systemet så ud efter første use case. Den er ikke tiltænkt, som en figur der skal nærstuderes, da de mest komplekse dele af den færdige udgave af modellen bliver gennemgået i dybden i afsnit 9 der omhandler implementationen.

Her kan det også ses, at det er struktureret ud fra en 3-lags arkitektur som bliver beskrevet i afsnit 9.1 om Arkitektur.



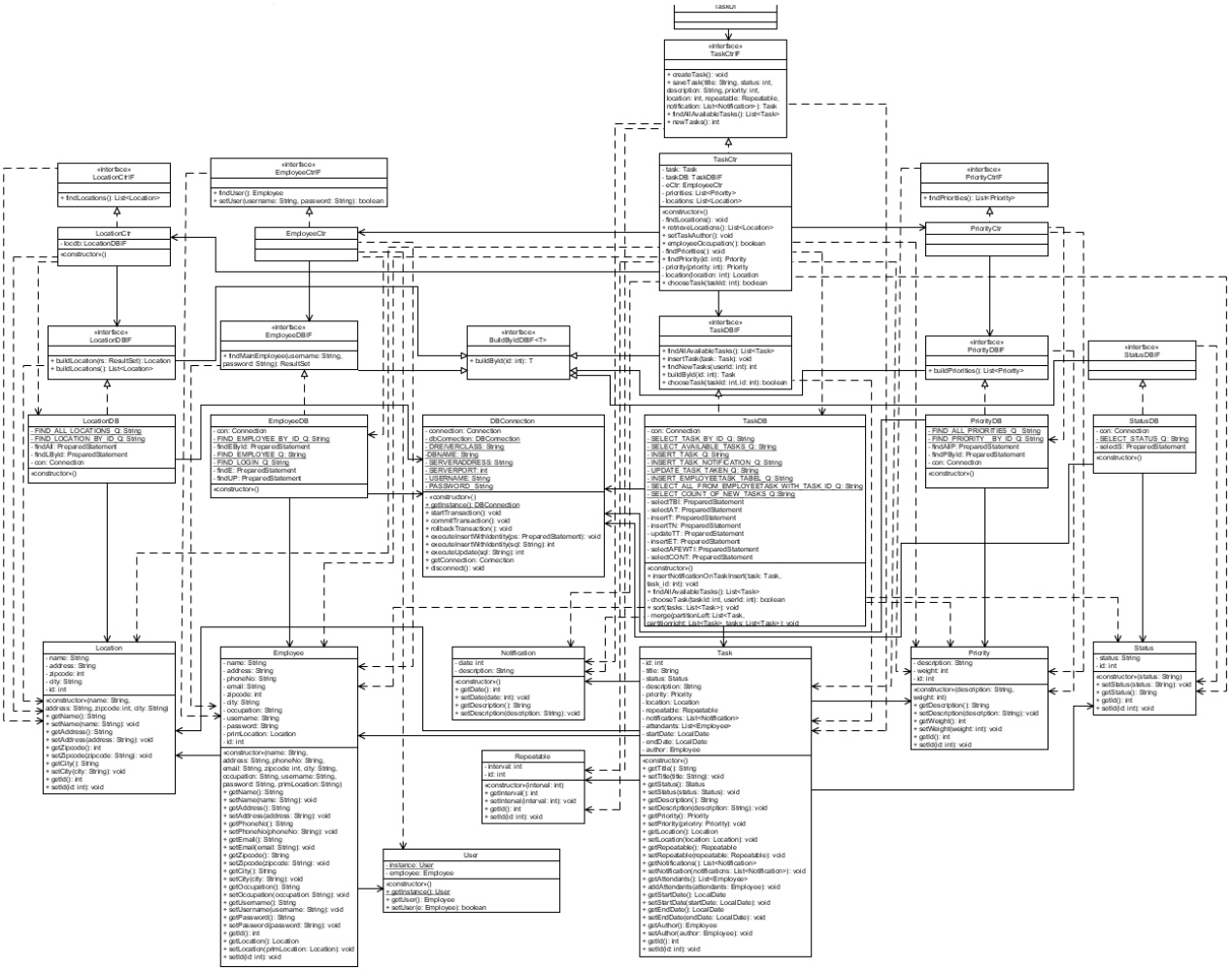
Figur 21: Viser 1. iteration af Design Klasse diagrammet.

Efter første iteration blev der implementeret ud fra diagrammet i figur 21.

8.9.2. Iteration 2

I figur 22 bliver anden iteration af design klassediagrammet vist. Denne udgave er den færdige udgave efter anden use case. Her blev der tilføjet imports, hvilket gør det lidt mere uoverskueligt at kigge på. Derudover er det måske heller ikke nødvendigt for at kunne implementere systemet. Med nuværende udviklingsværktøjer, såsom en IDE (Integrated Development Environment), kan de informere om hvilke imports der er nødvendigt for at kunne kompilere da størstedelen af koblingerne med pilene er for at vise hvilke klasser kender til hvilke andre klasser.

Ved at kigge på udviklingen fra første Iteration til anden Iteration kan det ses at første use cases samtlige klasser, på nær Status funktionalitet.



Figur 22: Viser 2. iteration af design klasse diagrammet

Efter denne iteration af Design Klasse diagrammet blev use casen Choose Task implementet. Den færdige implementation af Design Klasse diagrammet kan som nævnt tidligere ses uddybet i afsnit 9.

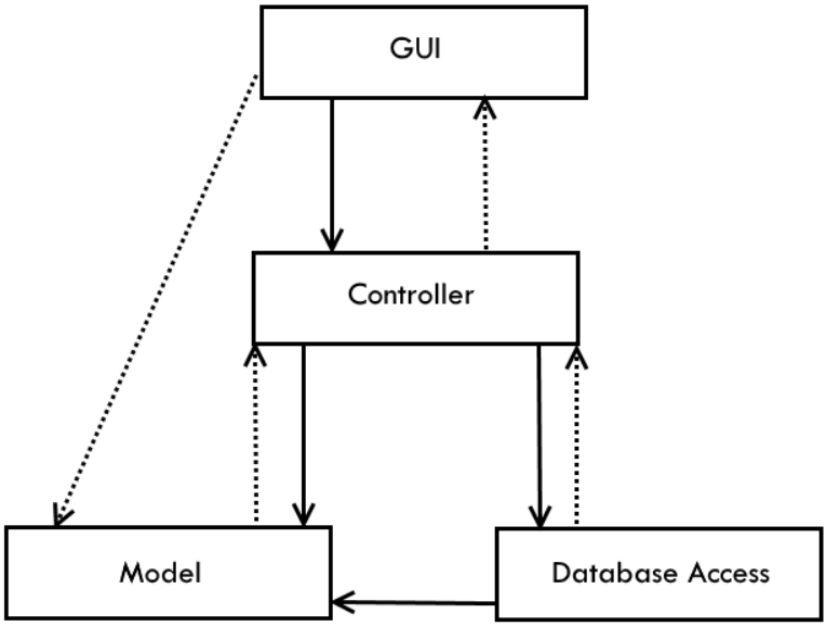
9. Implementation

I dette afsnit vil der blive gennemgået arkitekturen for det implementerede system, klasserne der udgør hver del af programmet og interessante metoder og programdele. I arkitekturafsnittet vil design klassediagrammet uddybes i forhold til den implementerede funktionalitet som en fortsættelse af det tidligere afsnit for diagrammet. Sidst i afsnittet vil test blive kort beskrevet med tilhørende testcases og scenarier.

9.1. Arkitektur

Som gennemgået i tidligere afsnit er softwareudviklingsprocessen UP. I UP er der en proces til at etablere, hvilken arkitektur der giver mest nytte for en eventuel implementation af et system. Det er i Inception fasen, at en kandidat arkitektur opstilles, dog var der i dette projekt allerede en forestilling om, hvilken arkitektur der gjorde sig mest gældende. I Elaboration fasen blev den logiske arkitektur bestemt og hvilke patterns, der eventuelt skulle benyttes til implementationen.

Arkitekturen er af en åben lagret form. Det vil sige, at programmet er opdelt i lag, der hver har en specifik funktionalitet, de har ansvar for. Arkitekturen er åben fordi selvom der kun kaldes ned i hierarkiet kan øvre lag godt læse informationer fra andre lag under dem. Dette kan eksempelvis gøres i **GUI** laget, hvor det kan blive nødvendigt at aflæse informationer fra **Task** objekter, der er bygget i **DB** laget og sendt tilbage. Åben arkitektur reducerer uafhængighed mellem lagene, men kan give bedre vedligeholdelses muligheder.



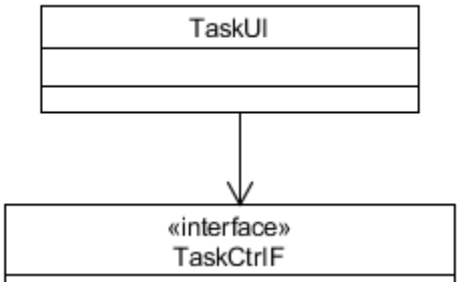
Figur 23: Viser den åbne 3-lags arkitektur af systemet.

Figur 23 viser den arkitektur dette program er implementeret efter. GUI laget er brugergrænsefladen, her vises informationer om objekter og måden brugeren kommunikerer med database data og starter flowet på use cases, der finder sted i laget under.

Dette lag er Controller laget, der styrer funktionaliteten i resten af programmet. Dette betyder metoder, der udfører logik der direkte konsoliderer og transformerer data i de nedre lag til at kunne repræsenteres i GUI. I det nederste lag er databaseklasserne og modelklasserne. Databaseklasserne har adgang til modelklasserne for at kunne bygge objekter, der kan returneres til de øvre lag. I de næste afsnit vil der blive gennemgået hvert lag i arkitekturen fra top til bund og vises kodeudsnit fra de mest dækkende og interessante programdele i hvert respektive lag.

9.2. GUI

I dette afsnit vil der være en kort gennemgang af brugergrænsefladelaget. En af de eneste interessante klasser heri er **DatabasePing**. Denne klasse vil blive beskrevet i afsnittet omkring samtidighed, da denne klasse extender **Thread** og fungerer som en tråd på brugergrænsefladen. På figur 24 kan der ses et udsnit fra Design Klasse diagrammet, hvor GUI laget er forkortet til **TaskUI**. Grunden til dette er, at hovedmængden af funktionalitet i det implementerede program handler om opgavehåndtering. Det kan derefter ses, at GUI laget går til Controller laget.



Figur 24: Viser et udsnit af design klasse diagrammet over hvordan GUIen inteagere med TaskCtrlF

Ud over **DatabasePing** kan der kigges i klassen **MainMenuSelectedTaskJPanel**, hvilket er et vindue, der kommer frem når der vælges at inspicere en opgave, der er hentet tilbage fra databasen, med det formål at kunne vælge den som gennemgået i use casen Choose Task. Der er her en metode som er vist i kodeudsnit 1, der kalder *get()* metoder på det **Task** objekt der er blevet valgt fra listen for at få udfyldt tekstbokse. Metoden er interessant fordi det viser, at GUI laget kan læse informationer direkte fra Model lags klasser, som vist i figur 23.

```
306     private void showInformation(Task task) {
307         txtName.setText(task.getAuthor().getName());
308         txtPhone.setText(task.getAuthor().getPhoneNo());
309         txtOccupation.setText(task.getAuthor().getOccupation());
310         txtTitle.setText(task.getTitle());
311         txtLocation.setText(task.getLocation().getName());
312         txtAddress.setText(task.getLocation().getAddress());
313         lblShowPriority.setText(task.getPriority().getDescription());
314         dtrpnDescription.setText(task.getDescription());
315     }
```

Kodeudsnit 1: Viser kildekoden for metoden showInformation

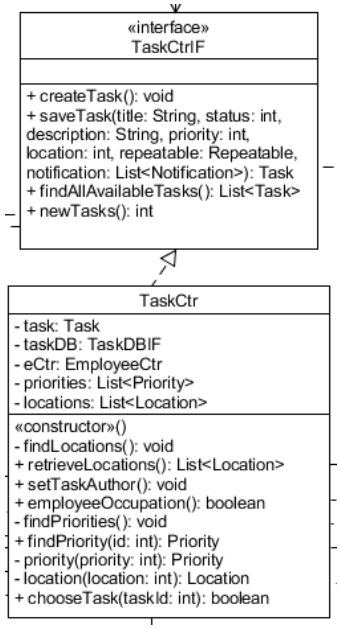
9.2.1. Exception Håndtering

Metoder igennem programmet kaster **NullPointerException** og **SQLException** ofte. Disse er næsten alle håndteret i GUI laget, da det er nødvendigt for brugeren at vide, hvad der eventuelt er gået galt for at have en bedre brugeroplevelse. Det gør det også lettere at finde fejlen for en udvikler. Når en exception bliver kastet sendes den til det originale kaldested, som er GUI. Det skal dog pointeres, at der kun er de største og mest seriøse problemer, der sendes til GUI, ikke alle exceptions bliver håndteret der.

9.3. Controller

I Controller laget ligger der en række klasser, der styrer Model og Database laget. GUI laget kalder metoder fra dette lag til transformering af data for at følge arkitekturreglerne. Et interessant sted at starte er i klassen **TaskCtr** da begge use cases håndterer opgaver. Use casen Opret opgave benytter for eksempel de andre Controller klasser, da disse håndterer deres respektive Model klasser, såsom **EmployeeCtr** til **Employee**.

Controller laget for **TaskCtr** fra Design Klasse diagrammet kan ses på figur 25.



Figur 25: Viser metoder og instance variabler i TaskCtrIF og TaskCtr.

Nogle interessante metoder i **TaskCtr** er *findLocations(), setTaskAuthor(), employeeOccupation(), createTask() og saveTask()*. Disse dækker også størstedelen af de to implementerede use cases, da begge behandler opgaver.

Et godt sted at starte ville være i metoden *createTask() som er vist i kodeudsnit 2*. Metoden bliver kaldt i brugergrænsefladen, når vinduet til at oprette en opgave bliver åbnet. Metoden sætter den nuværende bruger til at være forfatteren af opgaven i metoden *setTaskAuthor()* og laver to metodekald (*findPriorities()* og *findLocations()*), der hver kalder en metode i henholdsvis **PriorityCtr** og **LocationCtr**.

```
87      @Override
88      public void createTask() throws SQLException {
89          Task t = new Task();
90          this.task = t;
91          setTaskAuthor();
92          findPriorities();
93          findLocations();
94      }
```

Kodeudsnit 2: Viser kildekoden for metoden createTask i TaskCtr.

Metoden *findPriorities()* udfylder en liste List<Priority> priorities på hvert **Task** objekt. Metoden får udfyldt listen igennem et metodekald i **PriorityCtr**, der kalder en metode *buildPriorities()* i **PriorityDB**. Der er altid kun tre prioriteter i databasen, grundet databasens oprettelses script. Det samme gør metoden *findLocations(),* men for **Location** objekter.

Dette nu oprettede, næsten tomme, **Task** objekt kan nu gemmes til databasen alt efter respons fra brugeren i GUI laget. Dette gøres ved at kalde *saveTask()* metoden set i kodeudsnit 3.

```
96      @Override
97      public void saveTask(String title, String description, int priority, int location, Repeatable repeatable,
98              List<Notification> notification) throws SQLException {
99          task.setTitle(title);
100         Status temp = new Status("Temp");
101         temp.setId(1); // Just created task
102         task.setStatus(temp);
103         task.setDescription(description);
104         task.setPriority(priority(priority));
105         task.setLocaion(location(location));
106         task.setRepeatable(repeatable);
107         task.setNotification(notification);
108         taskDB.insertTask(task);
109     }
```

Kodeudsnit 3: Viser kildekoden for metoden saveTask i TaskCtr

Metoden tager en række forskellige primitive typer og objekter i dens parametre. Herefter bliver det nu oprettede **Task** objekt i Controlleren fyldt ud med nødvendig information til at kunne gemme den i

databasen. Det kan ses på linje 100, at der oprettes og instansieres et **Status** objekt som herefter sættes til at have et **id** på '1' fordi det er statussen "Oprettet" i databasen. Herefter udfyldes resten af informationerne ud fra inputtet på brugergrænsefladen på opgaven og **TaskDB** metoden *insertTask()* bliver kaldt med denne **Task**.

Til sidst kan metoden *employeeOccupation()* gennemgås. Metoden kan ses i kodeudsnit 4. Denne metode er muligvis en malplacering, da metoden kigger på, hvad forfatteren af **Task** objektet havde som stilling.

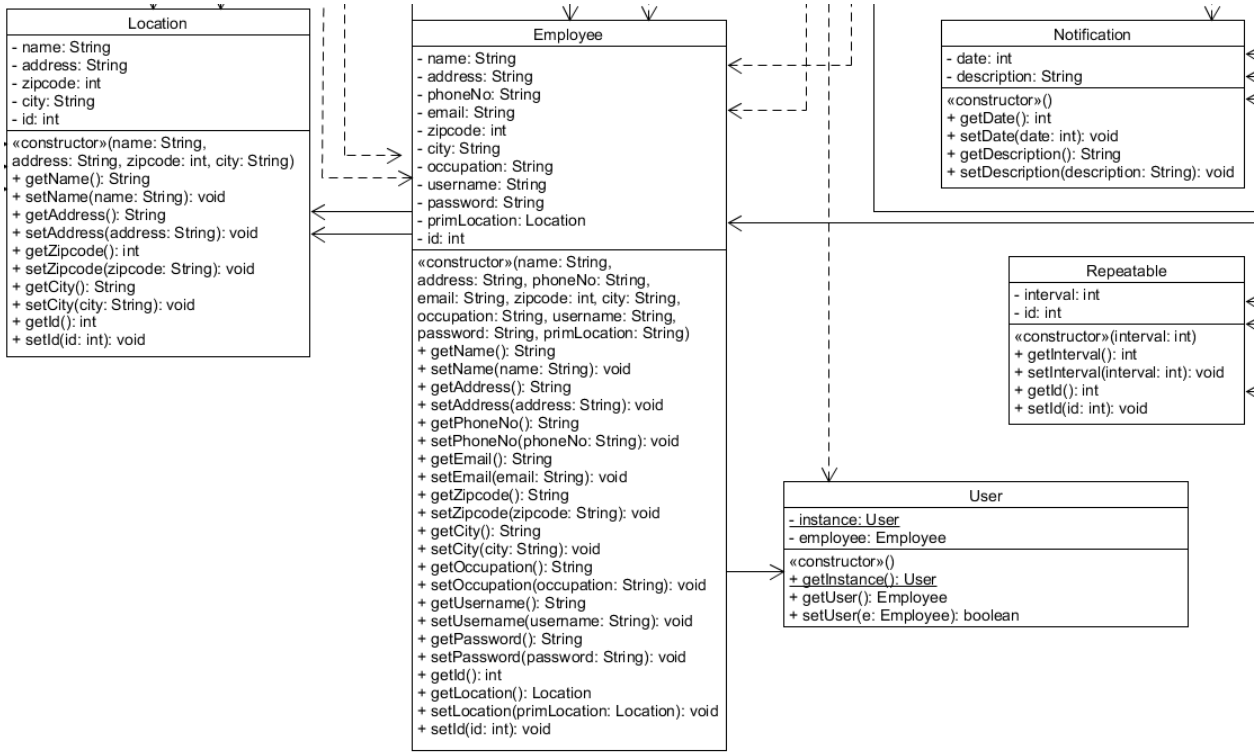
```
69      public boolean employeeOccupation() {
70          Boolean temp = false;
71          String occupation = task.getAuthor().getOccupation();
72          if (occupation.equals("Pedel") || occupation.equals("Leder")) {
73              temp = true;
74          }
75          return temp;
76      }
```

Kodeudsnit 4: Viser kildekoden for employeeOccupation metoden i TaskCtr.

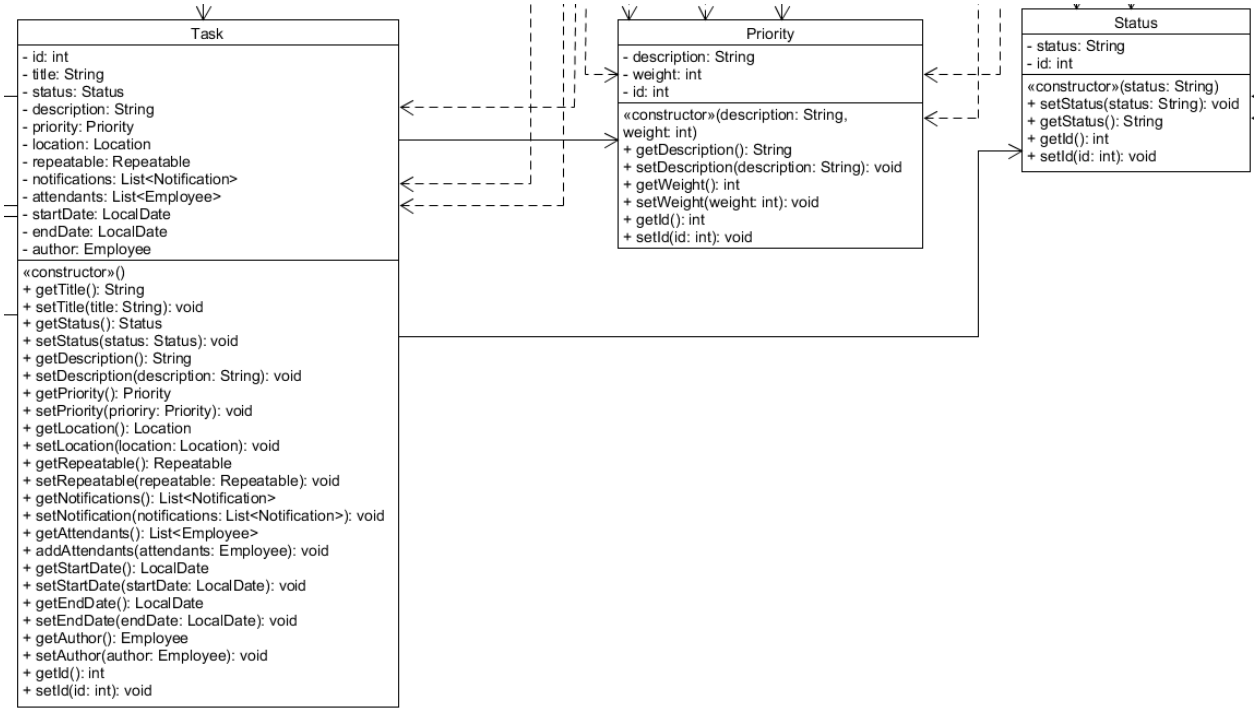
Dette er kun for at kunne se, hvilken type af brugergrænsefladebrugeren skal se, da Undervisere ikke har samme privilegier som Pedeller og Ledere. De kan for eksempel ikke vælge opgaver i systemet. Derfor kunne det argumenteres, at metoden er direkte GUI funktionalitet.

9.4. Model

Model klasserne er i samme lag som databaseklasserne i den lagrede arkitektur. Der vil blive gennemgået hver Model klasse herunder i figur 26 og 27. Den eneste interessante klasse her er User, da en bruger der er logget ind gemmes her. Et udsnit af Design Klasse diagrammet kan ses herunder for Model laget.



Figur 26: Viser nogle af model klasserne i design klasse diagrammet.



Figur 27: Viser resten af model klasserne der er i design klasse diagrammet.

Employee - Employees i programmet er personer, der kan blive brugere af systemet. Der er ingen funktionalitet til at oprette en Employee til databasen, derfor gøres dette i oprettelse scriptet. Employees bruges til at bygge et Employee objekt fra databasen i en builder.

Task - Tasks er generelle opgaver og grundlægger størstedelen af funktionaliteten bag andre klasser. Til at oprette en Task benyttes ikke dens constructor, hvori en anden metode i programmet benytter *set()* metoderne, efter en tom Task er oprettet for at kunne gemme den i databasen.

Location - Location er en lokation der benyttes til oprettelsen af opgaver så det kan ses, hvor opgaven finder sted.

Priority - Prioritet er vigtigt for brugervenligheden af systemet og står for at rangere opgaverne under oprettelse til en vis orden alt efter, hvor vigtige de er at løse under et tidsrum. Tre af disse er sat op i databasen i oprettelsesscriptet.

Status - Status er hvilken tilstand en opgave er i. Den kan være "Oprettet", "Igangværende" og "Færdiggjort". I dette program kan der ikke endnu færdiggøres en opgave, derfor benyttes kun de to første. Disse er også oprettet i databasen i scriptet.

User - User er den nuværende bruger, der er logget ind efter et godkendt login. User er en Employee og en singleton. Der kan kun være en User på én kørende klient. Dette er en måde at hurtigt have adgang til brugerens information, da kun denne klasse skal bruges efter login og brugeren skal ikke findes ydere.

Notification - Notifikationer var ment til at kunne vælges at sættes på en opgave når den skulle oprettes. Det var en form for besked system til at minde en Pedel om, at en opgave var ved at nå sin deadline for udførelse. Til fordel for at implementere mere vigtig funktionalitet blev denne klasse forbigået.

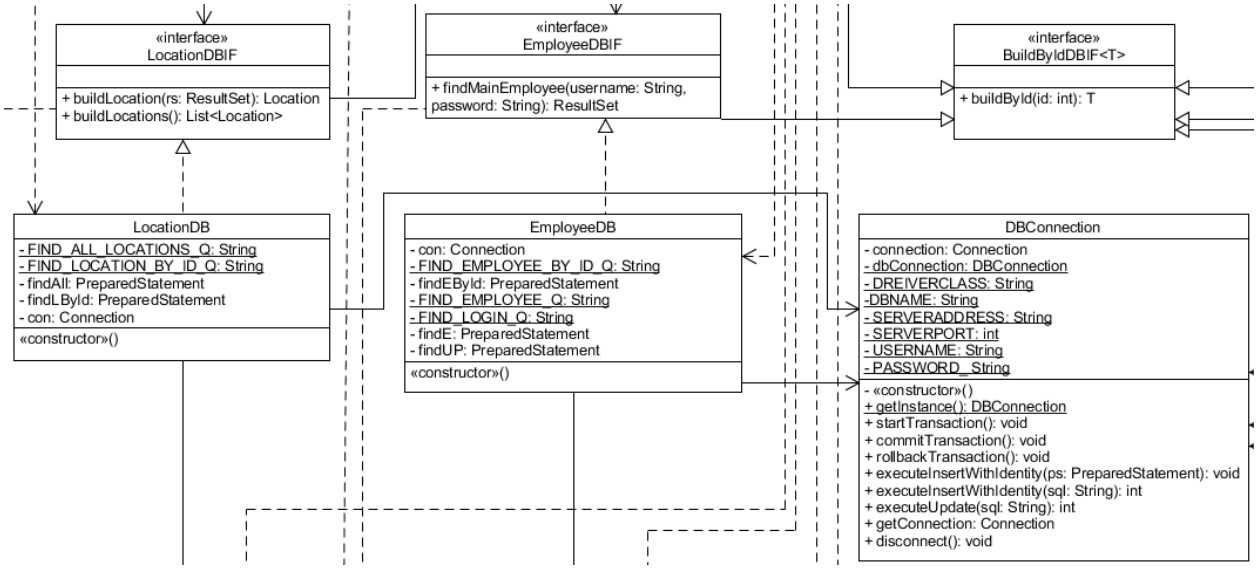
Repeatable - Repeatable var tænkt til at være en mulighed for at sætte et interval på en opgave så den ville blive oprettet hver gang den blev specificeret til. Denne klasse blev også forbigået lige **Notification** til fordel for mere vigtig funktionalitet.

9.5. Database

Database klasserne ligger i samme lag som Model klasserne. Det primære formål med klasserne er at modtage objekter til at indsætte i databasen eller 'build' objekter med det rå data i databasen. Alle database klasser har derfor et **Connection** objekt der instantieres i deres constructors. **Connection** objektet instantieres til at være en **DBConnection,** hvilket er en klasse i samme lag, der er en singleton, derfor er der altid kun en forbindelse til databasen der hentes, hvis den allerede er etableret.

Databaseklasserne kan ses i Design Klasse diagrammet herunder. På figur 28 kan der ses et Funktionelt Interface **BuildByIdDBIF**. Alle database interfaces extender dette interface, da måden at objekter bliver

bygget i database klasserne er ud fra et specifikt id. Alle database klasser implementerer derfor dette interface.



Figur 28: Viser en del af model klasser der kommunikere med databasen.

Databaseklassen **EmployeeDB** håndterer brugeren af systemet idét den sammenligner login informationerne fra brugergrænsefladen og en bruger i databasen. Hvis brugeren findes og informationerne matcher kan de logge ind. Dette gøres igennem metoden *findMainEmployee()* der ses i kodeudsnit 6. Metoden eksekverer to **PreparedStatement** der kører queries på databasen de er vist i kodeudsnit 5.

```
23    private static final String FIND_EMPLOYEE_Q = "SELECT employee.id FROM Employee, Login WHERE ? = Employee.login_id AND Login.id = Employee.login_id;";
24    private static final String FIND_LOGIN_Q = "SELECT * FROM Login WHERE ? = Login.username AND ? = Login.password;";
```

Kodeudsnit 5: Viser to String objekter der bliver brugt i EmployeeDB klassen

```
42      @Override
43      public int findMainEmployee(String username, String password) throws SQLException {
44
45          ResultSet lRs = null;
46          ResultSet eRs = null;
47          int eId = 0;
48
49          findUP.setString(1, username);
50          findUP.setString(2, password);
51          lRs = findUP.executeQuery();
52          if (lRs.next()) {
53              if (lRs.getString(2).equals(username) && lRs.getString(3).equals(password)) {
54                  findE.setInt(1, lRs.getInt(1));
55                  eRs = findE.executeQuery();
56                  if (eRs.next()) {
57                      eId = eRs.getInt(1);
58                  }
59              } else {
60                  eId = -1;
61              }
62          }
63
64          return eId;
65      }
```

Kodeudsnit 6: Viser kildekoden for findMainEmployee metoden fra klassen EmployeeDB.

findUP kører querien **FIND_EMPLOYEE_Q** for først at finde personen, der matcher logininformationerne. Herefter hvis personen findes i databasen bliver deres ID hentet ud ved hjælp af **findE** eksekvering. Dette Id returneres herefter og kan bruges i denne klasses implementation af *buildByID()* og den kan ses i kodeudsnit *7*.

```
67      @Override
68      public Employee buildByID(int id) throws SQLException {
69
70          Employee empl = null;
71          ResultSet rs;
72
73          LocationDBIF locDB = new LocationDB();
74          findEById.setInt(1, id);
75          rs = findEById.executeQuery();
76          if (rs.next()) {
77              empl = new Employee(rs.getString(2), rs.getString(3), rs.getString(4), rs.getString(5), rs.getInt(11),
78                      rs.getString(12), rs.getString(14), rs.getString(16), rs.getString(17),
79                      locDB.buildByID(rs.getInt(7)));
80              empl.setId(rs.getInt(1));
81          }
82
83          return empl;
84      }
```

Kodeudsnit 7: Viser kildekode for metoden buildByID fra klassen EmployeeDB.

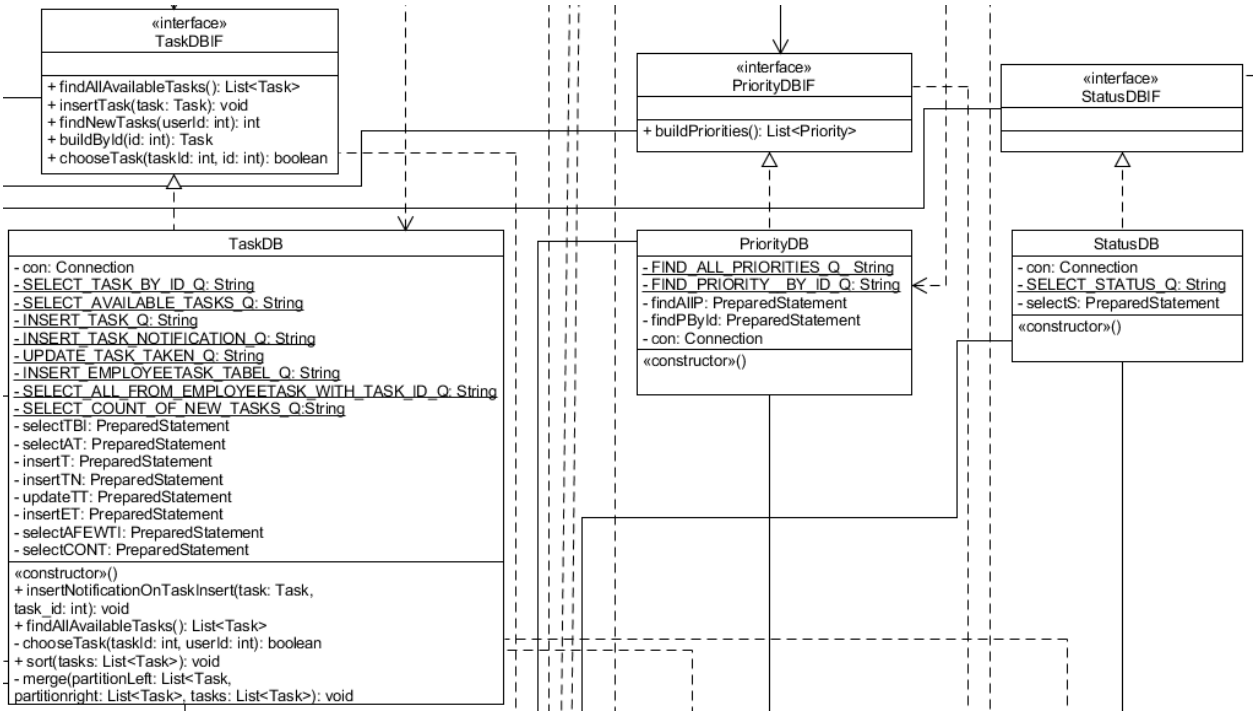
Denne klasses implementation af *buildByID()* kan ses i kodeudsnit 7. Metoden i dette program har kun formålet at oprette en bruger, der kan gemmes i **User** klassen, da det ikke er muligt at oprette **Employee** objekter til andet brug. Denne metode kaldes i **EmployeeCtr** klassen, med information fra brugergrænsefladen. Denne metode kan ses i kodeudsnit 8 og hedder *setUser()*.

```
37      @Override
38      public boolean setUser(String username, String password) throws SQLException, NullPointerException {
39          EmployeeDBIF edb = new EmployeeDB();
40          boolean res = false;
41          res = User.getInstance().setUser(edb.buildByID(edb.findMainEmployee(username, password)));
42          return res;
43      }
```

Kodeudsnit 8: Viser kildekoden for setUser metoden fra klassen EmployeeCtr.

I kodeudsnit 8 på linje 41 bliver instansen af singletonklassen **User** tilknyttet den bruger som **EmployeeDB** metoderne returnerer tilsammen, baseret på login informationer fra brugergrænsefladen.

De andre database klasser kan ses i det andet udsnit af Design Klasse diagrammet i figur 29.



Figur 29: Viser resten af model klasserne der kommunikere med databasen.

Den næste interessante klasse er **TaskDB**. Der blev gennemgået en smule af dens funktionalitet i Controller afsnittet, men der er nogle specifikke dele der er vigtige for use casene.

```
25    private static final String SELECT_AVAILABLE_TASKS_Q = "SELECT Task.id AS taskId, Task.title, Task.description, Task.startDate, Task.endDate, Task.author_id, \r\n"
26            + " Author.name AS AuthorName, Author.address, Author.phoneNo, Author.email, empZip.zipcode as employeeZip, empZip.city as employeeCity,\r\n"
27            + " Occupation.name, Login.username, Login.password, Author.primLoc,\r\n"
28            + " Task.location_id as taskLoc, \r\n"
29            + " --Location.name, Location.address, taskZip.zipcode, taskZip.city, \r\n"
30            + " Repeatable_id, Task.priority_id, Priority.description, Priority.weight, Task.status_id, Status.status\r\n"
31            + "     FROM Task \r\n" + "     --INNER JOIN location ON (task.location_id = location.id)\r\n"
32            + "     --INNER JOIN Zipcode as taskZip ON (Location.zipcode_id = taskZip.id)\r\n"
33            + "     INNER JOIN Priority ON (Priority.id = task.priority_id)\r\n"
34            + "     INNER JOIN Status ON (Status.id = Task.status_id)\r\n"
35            + "     INNER JOIN Employee AS Author ON (Author.id = Task.author_id)\r\n"
36            + "     INNer JOIN Zipcode AS empZip ON (Author.zipcode_id = empZip.id)\r\n"
37            + "     INNER JOIN Occupation ON (Author.occupation_id = Occupation.id) \r\n"
38            + "     INNER JOIN Login ON (Author.login_id = Login.id)\r\n"
39            + "     WHERE Task.status_id <> 3 AND Task.author_id = Author.id;";
```

Kodeudsnit 9: Viser kildekoden for SQL query SELECT_AVAILABLE_TASK_Q

Den første del af klassen er en query der kan ses på kodeudsnit 9. Den henter en række informationer ud fra flere tabeller i databasen, der alle er nødvendige for at bygge **Task** objekter, der kan sættes i en liste og vises i brugergrænsefladen. Metoden *findAllAvailableTasks()* er vist i kodeudsnit 10, den returnerer den liste, som en bruger kan kigge på og vælge en opgave de gerne vil tilknyttes. Den ovenstående query kan eksekveres igennem **PreparedStatement selectAT**.

```
114     public List<Task> findAllAvailableTasks() throws SQLException {
115         List<Task> tList = new ArrayList<>();
116         try {
117             LocationDB ldb = new LocationDB();
118             ResultSet rs = selectAT.executeQuery();
119             while (rs.next()) {
120                 selectAFEWTI.setInt(1, rs.getInt(1));
121                 //ResultSet rsET = selectAFEWTI.executeQuery();
122                 Task t = new Task();
123                 t.setId(rs.getInt(1));
124                 t.setTitle(rs.getString(2));
125                 t.setDescription(rs.getString(3));
126                 if (rs.getDate(4) != null) {
127                     t.setStartDate(rs.getDate(4).toLocalDate());
128                 }
129                 if (rs.getDate(5) != null) {
130                     t.setEndDate(rs.getDate(5).toLocalDate());
131                 }
132                 Employee e = new Employee(rs.getString(7), rs.getString(8), rs.getString(9), rs.getString(10),
133                         rs.getInt(11), rs.getString(12), rs.getString(13), rs.getString(14), rs.getString(15),
134                         ldb.buildByID(rs.getInt(16)));
135                 e.setId(rs.getInt(6));
136                 t.setAuthor(e);
137                 t.setLocaion(ldb.buildByID(rs.getInt(17)));
138                 Priority p = new Priority(rs.getString(20), rs.getInt(21));
139                 p.setId(rs.getInt(19));
140                 t.setPriority(p);
141                 Status s = new Status(rs.getString(23));
142                 s.setId(rs.getInt(22));
143                 t.setStatus(s);
144                 t.setRepeatable(null);
145                 // Is not used and adds time complexity
146                 // while (rsET.next()) {
147                 // t.addAttendants(edb.buildByID(rsET.getInt(1)));
148                 // }
149                 tList.add(t);
150             }
151         } catch (NullPointerException | SQLException e) {
152             e.printStackTrace();
153         }
154 
155         sort(tList);
156         return tList;
157     }
```

Kodeudsnit 10: Viser kildekoden for findAllAvailableTasks metoden i klassen TaskDB

Alt det rå data som querien henter gemmes i et **ResultSet** og benyttes til at bygge en **Task**. Når alle opgaver, der kan vælges, er sat ind i listen, sorteres den med en metode i samme klasse *sort()*, der sorterer efter vægten af prioriteten på hver opgave fra tabellen, dette kan ses på linje 155 i kodeudsnit 10.

Metoden *sort()* er den ene halvdel af en MergeSort sorteringsalgoritme, der er implementeret for at sortere prioriteternes vægt fra højest prioritet til lavest, så de vigtigste opgaver vises først i brugergrænsefladen, da disse er kritiske. Metoden *sort()* og *merge()* kan ses på henholdsvis kodeudsnit 11 og 12.

```
233     public void sort(List<Task> tasks) {
234
235         if (tasks.size() < 2) {
236             return;
237         }
238
239         List<Task> partitionLeft = new ArrayList<Task>(tasks.subList(0, tasks.size() / 2));
240         List<Task> partitionRight = new ArrayList<Task>(tasks.subList(tasks.size() / 2, tasks.size()));
241
242         sort(partitionLeft);
243         sort(partitionRight);
244
245         merge(partitionLeft, partitionRight, tasks);
246
247     }
```

Kodeudsnit 11: Viser kildekoden for sort metoden der er i klassen TaskDB

```
249     private void merge(List<Task> partitionLeft, List<Task> partitionRight, List<Task> tasks) {
250         int cursorLeft = 0;
251         int cursorRight = 0;
252         int cursorMerged = 0;
253
254         while (cursorLeft < partitionLeft.size() && cursorRight < partitionRight.size()) {
255             if (partitionLeft.get(cursorLeft).getPriority().getWeight() > partitionRight.get(cursorRight).getPriority().getWeight()) {
256                 tasks.set(cursorMerged++, partitionLeft.get(cursorLeft++));
257             } else {
258                 tasks.set(cursorMerged++, partitionRight.get(cursorRight++));
259             }
260         }
261
262         while (cursorLeft < partitionLeft.size()) {
263             tasks.set(cursorMerged++, partitionLeft.get(cursorLeft++));
264         }
265         while (cursorRight < partitionRight.size()) {
266             tasks.set(cursorMerged++, partitionRight.get(cursorRight++));
267         }
268
269     }
```

Kodeudsnit 12: Viser merge metoden der bliver brugt i sort metoden. Den ligger i TaskDB.

Metoden *sort()* modtager en liste af **Task** objekter og splitter den først ned i midten i to nye lister; **partitionLeft** og **partitionRight.** Herefter kaldes sorteringen rekursivt på disse partitioner indtil listerne er '1' i størrelse. Når de når dette mål returnerer *sort()* metoden på den vilkårlige liste og *sort()* køres på den anden partition indtil den også returnerer.

På linje 245 kaldes så *merge()*, der i et while-loop, på linje 254 itererer igennem listerne fra parametrene med et index og hvis den ene værdi er større end den anden sættes den ind på første pladsen i den sammensatte liste **tasks**. Dette fortsætter indtil den endelige sammensatte muterede liste, hvor alle de originale integer værdier er sat ind kan returneres fra kald stedet *findAllAvailableTasks()*.

Den sidste metode der er interessant i **TaskDB,** er metoden *chooseTask()*. Denne metode kan ses i kodeudsnit 13. Metoden er hovedmetoden, der kommunikerer med databasen, når en bruger vælger en opgave. Først kaldes denne klasses implementation af *buildByID()* på det id for opgaven i parameteren. Herefter sættes start datoen fordi nu er opgaven sat i gang. Så køres en update query på tabellen EmployeeTask, der er en tabel, der kobler en brugers id til en opgaves id. Dette er databasens repræsentering af alle tilknyttede Pedeller til en opgave.

```
194     public boolean chooseTask(int taskId, int userId) throws SQLException {
195         Task t = null;
196         boolean res = false;
197 
198         try {
199             t = buildByID(taskId);
200             if (t.getStartDate() != null) {
201                 updateTT.setDate(1, Date.valueOf(t.getStartDate()));
202             } else {
203                 updateTT.setDate(1, Date.valueOf(LocalDate.now()));
204             }
205             updateTT.setInt(2, t.getId());
206             insertET.setInt(1, userId);
207             insertET.setInt(2, t.getId());
208 
209             DBConnection.getInstance().startTransaction();
210             updateTT.execute();
211             insertET.execute();
212             DBConnection.getInstance().commitTransaction();
213             res = true;
214 
215         } catch (SQLException e) {
216             DBConnection.getInstance().rollbackTransaction();
217             System.out.println("roll back");
218         }
219         return res;
220     }
```

Kodeudsnit 13: Viser kildekoden af chooseTask metoden i klassen TaskDB..

Måden at datasættet bliver indsat i databasen er lidt speciel da der explicit deklareres et commit og der foretages et rollback hvis en bruger allerede er sat på den opgave.

9.6. Samtidighed

I dette afsnit vil der blive gennemgået, hvordan der er implementeret samtidighed i programmet. Som nævnt i GUI afsnittet står klassen **DatabasePing,** som en klasse der extender **Thread** og fungerer som en tråd. Denne tråd har ansvar for at notificere i en pop-up for brugeren af systemet om, at en ny kritisk opgave er blevet oprettet af en anden bruger.

```
 8 public class DatabasePing implements Runnable {
 9
10     private int count;
11     private TaskCtrIF tCtr;
12
13     public DatabasePing() {
14         setCount(0);
15         try {
16             tCtr = new TaskCtr();
17         } catch (SQLException e) {
18             new MessagePopUpJDialog("Databasen blev ikke fundet").setVisible(true);
19             e.printStackTrace();
20         }
21     }
```

Kodeudsnit 14: Viser kildekoden for klassen DatabasePing's constructor og instance vaiabler.

Klasseheaderen for **DatabasePing** kan ses i kodeudsnit 14. Klassen har variablerne **int count** og **TaskCtrIF tCtr**. De bliver initialiseret i constructoren til klassen, hvor **count** sættes til at være 0 ved brug af en *setCount()* metode. **TaskCtr** initialiserer et **TaskDB** objekt, der opretter en forbindelse til databasen, hvor der fanges en **SQLException** og brugeren for en pop-up, hvis den ikke kunne etableres. Grunden til at **count** sættes til 0 i constructoren er for at sætte mængden af nye højt prioriterede til nul, så første gang at brugeren logger ind og tråden kører opdateres **count** til mængden af opgaver i databasen, hvilket den så husker indtil, at dette antal ændrer sig. Dette gør den i klassens *run()* metode set i kodeudsnit 15.

```
27     @Override
28     public void run() {
29         while (true) {
30             try {
31                 int temp = tCtr.newTasks();
32                 if (count < temp) {
33                     new MessagePopUpJDialog("Der er en ny kritisk opgave.").setVisible(true);
34                     setCount(temp);
35                 } else {
36                     setCount(temp);
37                 }
38                 Thread.sleep(10000);
39             } catch (InterruptedException e) {
40                 e.printStackTrace();
41             } catch (SQLException e) {
42                 e.printStackTrace();
43             }
44         }
45     }
```

Kodeudsnit 15: Viser kildekoden for DatabasePing's run metoden.

På linje 31 kan det ses, at variablen **temp** sættes til at være returværdien fra **TaskCtr** objektet, der blev initialiseret i constructoren. Hvis dette antal er større end, hvad brugeren havde da de loggede ind, bliver der lavet en pop-up, der informerer brugeren om nye kritiske opgaver, der endnu ikke er blevet valgt af en bruger. Hvis returværdien fra *newTasks()* ikke returnerer et større tal sættes **count** til at være det samme uden, at en pop-up vises for brugeren. Herefter prøver den igen efter minimum 10 sekunder som sat i *sleep()* metoden på linje 38, hvorefter operativsystemets scheduler bestemmer hvornår den skal køre trådens *run()* metoden igen.

Metoden *newTasks()* i **TaskCtr** klassen returnerer en værdi fra **TaskDB** metoden *findNewTasks()*. Denne metode kører en SQL query på Task tabellen for at få antallet af opgaver der har statussen "Oprettet", har en prioritet på tre og ikke er oprettet af den samme bruger, der er logget ind. Dette antal returneres som sagt tilbage til **DatabasePing,** der opdaterer antallet af opgaver. Querien kan ses i kodeudsnit 16.

```
45      private static final String SELECT_COUNT_OF_NEW_TASKS_Q = "SELECT COUNT(*) FROM Task WHERE task.status_id = 1"
46                                                                + " AND task.priority_id = 3 AND task.author_id <> ?;";
```

Kodeudsnit 16: Viser SQL query SELECT_COUNT_OF_NEW_TASKS_Q.

Dette er programmets implementation af samtidighed. Tråden bliver oprettet og startet i GUI klassen **MainMenuJDialog**. Det er en hovedmenu, hvor brugeren kan trykke på andre menuer og vinduer. Brugeren kan dog få en pop-up andre steder i brugergrænsefladen også. Metoden *init()* i **MainMenuJDialog** er metoden, der instantierer tråden og starter den henholdsvist på linje 120 og 121 i kodeudsnit 17.

```
116     private void init() {
117         getCombinedTaskJPanel();
118         EmployeeCtrIF eCtr = new EmployeeCtr();
119         if (eCtr.findUser().getOccupation().equals("Pedel")) {
120             t = new DatabasePing();
121             t.start();
122         }
123 
124         if (!eCtr.findUser().getOccupation().equals("Underviser")) {
125             btnMessageSetEnabled();
126         }
127 
128     }
```

Kodeudsnit 17: VIser kildekoden hvor DatabasePing bliver instantieret i GUI.

Metoden tjekker om den nuværende bruger er en Pedel og starter ikke hvis de er en underviser, da undervisere ikke kan udføre opgaver, er der ingen grund til at starte tråden.

10. Test

For at teste implementationen af databasen og de use cases, der interagerer med de data, der er i den, er der blevet opsat en række systemtests, der bearbejder dem. Dette gøres ved at benytte klasserne **TaskDB, TaskCtr, EmployeeCtr, EmployeeDB, Employee** og **User**. Disse klasser er benyttet af begge use cases for at fuldende deres flow. De to use cases der bliver testet er Create Task og Choose Task. Ved at udføre systemtest på disse use cases bliver der samtidigt testet på klassen **DBConnection,** der er programmets forbindelse til databasen og også test på programmets sorteringsalgoritme i klassen **TaskDB**. Alle tests er implementeret ved hjælp af jUnit4.

Test klasserne gør brug af en opsat database, hvilket betyder at for, at testene kan fungere korrekt skal oprettelses-scriptet blive kørt mindst en gang ellers fejler testmetoderne.

Udover use case tests er der også lavet nogle metode tests, som GUI afhænger af. Disse er testet i klasserne **TaskDBInsertTest** og **PriorityCtrFindPrioritysTest**. Henholdsvis testes der på om, der kan hentes en liste af alle tasks med “Oprettet” eller “Igangværende” status og om der er oprettet de tre prioriteter, der benyttes i GUI under oprettelse af en opgave.
I dette afsnit vil der blive gennemgået de scenarier og testcases, der forekommer efter analyse af deres fully dressed beskrivelser og derefter nogle eksempler på, hvordan er blevet testet i programmet.

10.1. Testscenarier og Testcases

10.1.1. Opret opgave

Scenarie navn	**Start flow**	**Alternativt**
Scenarie 1 - Succesfuld opgave oprettelse	Main flow	
Scenarie 2 - Annullering af opgave oprettelse	Main flow	*Kan forekomme i alle flows på et vilkårligt tidspunkt.
Scenarie 3 - Kan ikke oprette forbindelse til databasen	Main flow	2a.
Scenarie 4 - Brugerne har ikke udfyldt alle påkrævede informationer	Main flow	5a.

Tabel 7: Indeholder test sernaier for implementationen af opret opgave.

I Create Task use casen skal en opgave kunne oprettes af en bruger, der er logget ind i systemet. Dette indebærer derfor, at en bruger skal være logget ind før, use casen kan starte ligesom gennemgået i fully dressed afsnittet til Create Task. Casen er opdelt i fire scenarier, hvorfra de henholdsvist er en succesfuld udførelse and use casen, en annullering der foregår igennem GUI, en tabt connection og hvis brugeren ikke har udfyldt alle nødvendige informationer i GUI til, at den kan oprettes.

Scenarie 2 er speciel da en annullering foregår udelukkende gennem GUI og data, der er indtastet bliver ikke gemt undervejs i en opgave. Dette betyder så, at der ikke er lavet systemtests på dette scenarie, da der ikke er data at teste på.

Test case ID	**Scenarie**	**Informationsmængde**	**Bemærkning**	**Forventet resultat**	**Faktisk resultat**

1a	**Scenarie 1** - Succesfuld opgave oprettelse	Påkrævet information*		Opgaven bliver oprettet i databasen	Opgaven blev oprettet i databasen
2a	**Scenarie 2** - Annullering af opgave oprettelse	ingen information		Opgaven bliver ikke oprettet i databasen	Kan ikke testes i en systemtest
2b	**Scenarie 2** - Annullering af opgave oprettelse	Påkrævet information*		Opgaven bliver ikke oprettet og alle informationer der er blevet indtastet bliver glemt	Kan ikke testes i en systemtest
2c	**Scenarie 2** - Annullering af opgave oprettelse	Påkrævet inklusive ekstra information	Dette gælder alternative flows: 3b, 3c, 3d,	Opgaven bliver ikke oprettet og alle informationer der er blevet indtastet bliver glemt	Kan ikke testes i en systemtest
3a	**Scenarie 3** - Kan ikke oprette forbindelse til databasen	In gen information	Test forbindelsen inden start på opgave oprettelse	Brugeren får en fejlmeddelelse hvis der ikke kan oprettes en forbindelse og får ikke lov til at fortsætte	

4a	**Scenarie 4** - Brugerne har ikke udfyldt alle påkrævet informationer	Mangler én af de påkrævet informationer		Brugeren bliver informeret om hvilken information der mangler og opgaven bliver ikke oprettet. Dog bliver ingen information glemt	Brugeren får besked om at de mangler informationer og kan ikke oprette opgaven.

Tabel 8: Viser test caes for de opstillede test sernarier.

Testcases for denne use case kan ses i tabel 8. Disse er implementeret i klassen **CreateTaskTest** ud over Scenarie 2 og Scenarie 3 testcases. Scenarie 3 beskæftiger sig med forbindelsen til databasen, hvilket er blevet testet i **DBConnectionTest** klassen. Den mest interessante testcase her er Scenarie 4, den er vist i kodeudsnit 18.

```
92      @Test
93      public void testScenario4() {
94
95          try {
96              TaskCtrIF tCtr = new TaskCtr();
97
98              tCtr.createTask();
99
100             tCtr.saveTask("testScenario4", null, 1, 3, null, null);
101
102             fail("Expected an SQLException to be thrown");
103
104         } catch (SQLException e) {
105             assertThat(e.getMessage(), is("Cannot insert the value NULL into column 'description',"
106                     + " table 'dmab0917_1067544.dbo.Task'; column does not allow nulls. INSERT fails."));
107         }
108     }
```

Kodeudsnit 18: Kildekode af test sernarie 4.

Testcasen opretter en tom opgave med metodekaldet *createTask()* på en instans af **TaskCtr**. Herefter køres metoden *saveTask()* på linje 100 på den **Task,** der nu er oprettet i controller klassen. Her indsættes en række data i parametrene, der er nødvendige for tabellen i databasen; en titel efterfulgt af en description der begge to ikke må være **null**. Her sættes den så til at være **null,** hvilket vil kaste en **SQLException**. Testen sættes til at fejle, hvis den ikke fanger en exception.

10.1.2. Choose Task

Scenarie navn	Start flow	Alternativt
Scenarie 1 - Succesfuld udførelse af en bruger der vælger en gyldig opgave	Main flow	
Scenarie 2 - Kan ikke oprette forbindelse til databasen	Main flow	2d.
Scenarie 3 - En bruger kan ikke tilføjes til den valgte igangværende opgave	Main flow	6a.
Scenarie 4 - Ingen opgaver findes i systemet	Main flow	2c.
Scenarie 5 - En opgave der er valgt ændrer status til igangværende	Main flow	
Scenarie 6 - En bruger bliver ikke tilknyttet opgaven da han allerede er sat på den	Main flow	6e.

Tabel 9: Test scenarier for vælg opgave

I Choose Task use casen skal en bruger, der er logget ind kunne vælge en opgave fra databasen, hvorefter de bliver tilknyttet de og status for opgaven ændrer sig når det gør sig gældende. Der er her seks scenarier, de kan ses i tabel 9, hvorfra scenarie 2 også er om, der er en forbindelse til databasen. Her kræver det også, at brugeren er logget ind, dog er det umuligt for en bruger at kunne vælge en **Task** da metoden *chooseTask()* afhænger af **User** ikke er **null**. Dette gennemgås i afsnittet for fully dressed af Choose Task.

Choose Task tests er oprettet i klassen **ChooseTaskTest**. Scenarie 2, 5 og 6 har ikke sine egne testmetoder. Scenarie 2 testes i **DBConnection** klassen og Scenarie 5 og 6 bliver testet i andre testmetoder.

11. Iværksætteri

Dette afsnit vil beskæftige sig med en reel hypotetisk virksomhed der producerer opgavestyringssystemer ligesom denne rapport har beskrevet indtil nu. Ideen om denne type system er ikke en ny ide, hvilket betyder at systemet skal skelne sig ud fra resten for at have en chance på markedet. Få justeringer kan derudover for til at andre lignende virksomheder kan gøre brug af det. Afsnittet vil beskrive de økonomiske overvejelser der er taget på denne eventuelle virksomhed og en gennemgang af Alexander Osterwalders forretningsmodel på denne type virksomhed. Iværksætteri afsnittet er placeret efter implementationen af systemet er gennemgået for at give en forestilling om hvad det ville kræve at lave et komplet system.

11.1. Økonomiske overvejelser

For at dette projekt kan lade sig gøre skal der være en økonomisk værdi i for både udviklerne og kunden. Herunder vil der blive lavet et overblik over de store udgifter til systemet og hvad det vil koste at producere og hvilken form for indkomst der skal være for, at det er muligt.

Det antages, at kunden vil kunne have systemet efter 3 måneders udvikling med 4 datamatiker udviklere til en gennemsnitlig løn på 33.000kr/ måneden samlet set, løber det op til 396.000kr i lønninger, som vil være den største udgift til systemet, hertil vil der komme mindre udgifter som for eksempel nyt udstyr, lokaleleje med mere som i sidste ende kan ligge et sted mellem 20.000kr for hele forløbet.

Det bør antages, at der er en opsætnings periode, hvor systemet bliver indført i virksomheden, her vil fejl og mangler i systemet blive løst. Her kan man tage en efterbetaling for at løse de problemer, der må være eller man kan i salgskontrakten beslutte, at de første 6 måneder er service og support gratis, dette vil dog forhøje prisen for systemet fordi der bliver givet en form for sikkerhed. Hvis der er mange indførelses problematikker, vil der være en udvikler sat på opgaven i alle 6 måneder til en samlet pris på 198.000kr i lønninger, dette vil dog være skuffende, hvis man tager udviklingsprocessen i perspektiv og forhåbentligt vil indførelsen være omkostningsfri.

Det vi vil kalde værste udfald for omkostninger, kan være helt op til 614.000kr, men beløbet kan komme ned på omkring 416.000kr.

Herunder er der en tabel, set på Tabel 10, der kan give lidt overblik over udgifterne ud fra det værste udfald.

Udgifts oprindelse.	Beløb i danske kroner.
Lønninger x 4	396.000,-
Lokale leje	10.000,-
Goder til ansatte/udstyr	10.000,-
Indførelse og efter service af system	198.000,-
I alt:	614.000,-

Tabel 10: Indeholder informationer om udgifter for udviklingen af systemet

Dette kan virke som en høj pris i forhold til hvor meget økonomisk gevinst der kan komme ud af at indføre systemet, derfor ville det være relevant at se på, hvordan man måske kan sælge systemet til flere kunder og dermed behøver man ikke at sælge det til den høje pris. Dette vil blive overvejet i forretningsmodellen.

11.2. Forretningsmodel

Forretningsmodellen er et redskab til at få et overskueligt overblik over hvilke elementer, som kan have en indflydelse på virksomhed og derfor hvilke områder der er vigtige at tage højde for. Modellen kan ses på figur 30. Alle beskrivelser i dette afsnit er baseret på denne model.

<table>
<tr><th>Key Partners</th><th>Key Activivities</th><th colspan="2">Value Proposistion</th><th>Customer Relationships</th><th>Customer Segments</th></tr>
<tr><td rowspan="3">* Server udlejnings firma

* Bank</td><td>* Videre udvikling af system

* Udvikling af nye systemer

* Løse support opgaver

* Vedligeholdelse

* Forbedringer</td><td colspan="2" rowspan="3">* Tidsregistrering

* Ledelsesværktøj

* Support

* Logistik

* Praktisk</td><td>* Kunde support via telefon og mail</td><td rowspan="3">* Virksomheder eller personer, som skal håndtere givne opgave</td></tr>
<tr><td>Key Resources</td><td>Channels</td></tr>
<tr><td>* Internet

* Computere

* Telefon

* App Store

* Server

* Kompetent medarbejdere</td><td>* Hjemmeside

* App

* Sociale medier</td></tr>
<tr><td colspan="3">Cost Structure</td><td colspan="3">Revenue Streams</td></tr>
<tr><td colspan="3">* Server leje

* Lønninger</td><td colspan="3">* Salg af system

* Abonement

* Service og overbygning af allerede solgt systemer</td></tr>
</table>

Figur 30: Alexander Osterwalders forretningsmodel på den eventuelle virksomhed.

11.2.1. Value Propositions

Value Propositions består af hvilke værdier som virksomheden kan give til dens kunder.

Denne virksomhed har som udgangspunkt til formål at give kunder et system, der kan hjælpe med at få overblik over medarbejders opgaver og tiden der bliver brugt på de opgaver. Derfor består værdierne af tidsregistrering som i tidsregistrering af arbejdsopgaver, logistik som i hvordan opgaverne bliver fordelt og ledelsesværktøj som giver ledere mulighed for at se informationer omkring de opgaver, der bliver løst. Disse værdier har systemets kvalitet en direkte indflydelse på.

Andre værdier består af support og brugervenlighed. Supporten skal være en værdi for at fastholde kunder og hvis supporten er god, forventes der også at kunderne vender tilbage, hvis de skal bruge andre systemer i fremtiden. Brugervenlighed er forventet, men i dette tilfælde betyder det, at systemet skal være nemt at indføre for virksomhederne selv, hvis de ansatte har mangel på IT-erfaring.

11.2.2. Customer Segments

Customer Segments består af nuværende kunder eller mulige kunde segmenter.

For denne virksomhed er segmentet bredt, idet mange virksomheder har en form for opgaver, der skal løses og dermed skal styres i en form eller anden. For at være lidt mere specifik kan der lidt i opstartsfasen fokusere

på virksomheder, der ligner virksomheden, som systemet bliver udviklet til, nemlig EUC Nord. Som beskrevet i virksomhedsbeskrivelsen er EUC Nord en uddannelsesinstitution og derfor kan der startes med den type kunder, men softwareudviklere kunne i nogle tilfælde bruge et lignede system.

En institutions liste fra 2015 [5] viser, at der er omkring 300 gymnasiale institutioner, som kunne være potentielle kunder. Det betyder dog ikke, at der ikke er mere end 300 potentielle kunder, da systemet burde kunne udvides til at fungere sammen med bolig firmaer og andre virksomheder, der har pedeller eller viceværter ansat.

11.2.3. Customer Relationships

Customer Relationships består af hvilken form for forhold, virksomheden vil have til deres kunder. Her afhænger det meget af hvilken form for produkt, der sælges til sine kunder.

Fordi denne virksomhed sælger et IT-system, forventer kunderne, at de kan få hjælp med systemet, hvis der skulle opstå problemer og at de får hjælp hurtigt især, hvis det er et system som virksomheden abonnerer på da det ellers kan koste dem penge.

Denne virksomhed vil prøve at opnå et godt forhold til kunderne, men især med supporten når komplikationer opstår, dette vil gøre, at de forhåbentligt vil vælge denne virksomhed igen, hvis de skal bruge et nyt system.

11.2.4. Channels

Channels består af, hvilken måde der kan opnås sit kunde segment for eksempelvis at sælge dem systemet.

Her vil virksomheden have en hjemmeside, som kunder kan finde gennem sociale medier eller direkte kontakt. Systemet vil blive delt på de forskellige mobilapp markedspladser som for eksempel GooglePlay eller Apples App Store hvor kunder kan hente softwaren.

11.2.5. Revenue stream

Revenue Stream består af hvilke former for indkomst, virksomheden kommer til at have.

Her er der tre punkter; salg af system, abonnement og service. Salg af system indeholder salg af hele systemer og det kommer til at være, der den største del af indkomsten vil komme fra og hvor de største kunder vil komme fra.

Abonnement vil give en månedlig indkomst fra mindre grupper eller personer, der kan bruge standardfunktionerne og vil betale månedligt baseret på antal bruger af systemet. Disse grupper består af studiegrupper og projektgrupper med mere.

Sidst men ikke mindst vil der være service og overbygninger til allerede solgte systemer. Her er der chance for, at kunden vil have videreudviklet systemet eller små justeringer, som ikke er dækket af salgskontrakten.

11.2.6. Key Partners

Key Partners består af samarbejdspartnere, som er nødvendig for at kunne levere value propositions.

For at have mulighed for at udvikle systemet skal der være en form for startkapital og dette vil en bank kunne tilbyde hvis de tror på virksomheds ideen.

Når systemet er oppe at køre, er det nødvendigt at have et 24/7 database system, her er udviklerne ikke eksperter, derfor vil der kunne outsources dette til et serverudlejningsfirma, hvis kunden ikke selv vil have det på deres lokation.

11.2.7. Key Activities

Key Activities består af daglige aktiviteter I virksomheden for at kunne levere value propositions.

I denne virksomhed er det udvikling af nye systemer, videreudvikling af eksisterende systemer og vedligeholdelse af disse systemer vigtigt for at have et godt produkt, der er moderne og kan bruges i mange forskellige arbejdsmiljøer.

En anden vigtig opgave er at give en god support til kunderne, for at have et godt forhold til kunderne er det altafgørende.

11.2.8. Key Resources

Key Resources består af hvilke midler, der skal til for at levere Value Propositions.

Her er det meget materialistiske midler der skære igennem, såsom internet, computere, telefoner, app markedsplads med mere.

Det er dog også vigtigt at have kompetente medarbejdere, der kan levere systemer i høj kvalitet.

11.2.9. Cost Structure

Cost Structure består af hvilke udgifter, der er for at levere Value Propositions.

Her er lønningerne den helt store udgift, men der er også omkostninger ved server leje og markedsføring, der dog mange andre udgifter, disse er dog mere relateret til daglig drift såsom leje af lokaler med mere.

11.3. Mission

Missionen kan udledes af Value Propositions i forretningsmodellen og måden virksomheden har tænkt sig at løse disse værdier på.

At udvikle konkurrencedygtige opgave systemer som er skræddersyet til den specifikke kunde. Det skal ske igennem god kundekontakt og relevant informationsindsamling i samarbejde med kunderne.

11.4. Vision

Visionen kan ses som et mål for virksomheden at stræbe efter uden at være demotiverende, derfor skal den være realistisk. Man kan se det som hvordan man gerne have at virksomheden skal se ud i fremtiden. Vores vision er: At udvikle systemer der kan effektivisere arbejdsprocesser nu og i fremtiden.

11.5. SWOT

I dette afsnit vil der blive analyseret den eventuelle virksomhed, dette vil blive gjort ud fra en SWOT analyse. De interne og de eksterne forhold er fundet ud for forretningsmodellen. Denne SWOT analyse har til formål at danne et overblik på hvilke styrker og svagheder den eventuelle virksomhed kunne have. SWOT analysen kan ses på tabel 12.

Interne forhold	
Styrker	Svagheder
God kontakt med første mulige kunde Softwareudviklingen foregår som en iterativ proces De ansatte har samarbejdet før	Manglende erfaring Mangel på økonomisk sikkerhed
Eksterne forhold	
Muligheder	Trusler
Stort marked for opgavestyringssystemer Arbejder med klienter ved udviklingen af systemer i et tæt samarbejde Mulighed for at udvikle nye systemer til andre markeder end opgavestyring	Konkurrerende virksomheder Lovgivning der kan påvirke systemudvikling eksempelvis en privatlivslov

Tabel 11: SWOT analyse af den potentielle virksomhed.

Den eventuelle virksomheds interne styrker ligger i, at der kunne være god kontakt med den første kunde, som den eventuelle virksomhed kunne arbejde sammen med. Dette har gjort at virksomheden kan fra starten af vide hvad kunden gerne ville have. De ansatte har arbejde sammen, hvilket gør at der er kan forekomme en virksomhedskultur der passer på alles temperament. Derudover foregår softwareudviklingsprocessen iterativ, da det var sådan dette projekts system blev udviklet.

Virksomhedens interne svagheder er manglende erfaring. Disse manglende erfaringer er samarbejde med en kunde og at udarbejde et helt system på den måde at kunden kunne ønske sig at systemet var. En anden svaghed er mangel på den økonomiske sikkerhed dette indebærer, at den potentielle virksomhed ikke får fuld betaling for det første system der bliver udviklet da der ikke er en aftale med nogen bank.

Dens muligheder er, at der er et stort marked for opgavestyringssystemer, hvilket giver en mulighed for mange nye kunder som gerne vil have et lignede system. Udover opgavestyringssystemer er der også muligheden for at udvikle nye systemer, som gør at der er mulighed for et bredere marked. Samarbejde med en klient er en klar fordel da der er direkte adgang til hvad de vil have modsat hvis der blev forsøgt et salg af et system der eventuelt ikke kunne bruges af nogen virksomhed, fordi den ville være for langt fra hvad de skulle bruge.

Dens trusler er at der allerede er mange virksomhed som også arbejder med opgave styringssystemer eller lignende systemer som gør at der vil være færre kunder eller at kunderne vælger nogle andre virksomheder. Et eksempel på en lovgivning der kunne være en trussel er EU's nye privatlovgivning.

12. Konklusion

Der er udviklet et system der haft formål at løse den originale problemformulering rapporten startede med i afsnit 3. Der vil først blive lavet en besvarelse og vurdering af om problemformuleringen er blevet løst.

Til første spørgsmål i problemformuleringen om det ville være muligt at udvikle et brugervenligt opgavesystem for benyttelse på EUC Nord kan der konkluderes at det er formået at udvikle et system der kan være en god model for hvordan sådan et system kunne se ud. Det kan f.eks. ses på programstrukturen ud fra Design Klasse diagrammet, hvori arkitekturen giver rig mulighed for videreudvikling og implementering af resten af de foreslåede use cases.

Brugervenligheden kan betragtes som en ikke løst opgave, da der ikke var mulighed for at lave en observation for den samme repræsentant på det implementerede system. Dog kan der foretages en antagelse baseret på mock-up interviewet.

I forhold til standardiseringen af kommunikationsmulighederne, kan det betragtes som en løst opgave da, ved hjælp af samtidig programmering, kan informere alle brugere om en nyoprettet opgave. Hvilket er en af de største problematikker der blev formueret ud fra domæneanalysen.

I forhold til opgavefordeling, kan dette betragtes som en delvis løst opgave, da alle brugere af korrekt type login acceptere nye opgaver der oprettes i databasen, dog kan de ikke færdiggøre opgaverne hvilket resultere i at der ikke kan laves statistikker over de ansattes opgaver i arbejdstiden.

13. Perspektivering

Som perspektivering kan det diskuteres at der er mange forskellige udgaver af denne type opgavestyringssystem, men der altid vil være en nytte i at udvikle nye smarte udgaver af systemtypen. Derfor et system der kan være værd at udvikle, dog med et marked under så stor konkurrence er det risikabelt rent finansielt. Derfor ville den eventuelle virksomhed der er beskrevet i afsnit 11 have det svært med at etablere et samarbejde med en virksomhed.

13.1. Proces beskrivelse

Som nævnt i afsnit 4 der omhandler metoden der er blevet anvendt i processen til at udvikle systemet, har vi brugt UP.

Processen der startede med at blive lagt en tidsplan for projektet i Microsoft Projekt den kan findes i afsnit 4 og i bilag [A]. Hele tidsplanen blev ikke lavet til at starte med, da det var besværligt at lave før construction faserne. Derudover var projektperioden delt op over semesteret, hvilket der også skulle tages hensyn til. Dette gjorde det besværligt at overskue, hvor lang tid der var til hver opgave og iteration.

Arbejdsprocessen var positiv det meste af forløbet da alle gruppemedlemmer mødte ind som de skulle og blev til det der var planlagt. I forhold til programmeringsopgaverne i construction faserne, kunne der have været bedre fordelingen af dem. Heldigvis arbejdede vi ofte i par og alle fik lov til at programmere.

Litteraturliste

[1] Staunstrup E, Skriver H.J, Storm-Henningsen P. *Organisation 6. udgave.* Trojka 2017.

[2] Larman C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development 3. edition*. 2004.

[3] EUC Nord. *http://www.eucnord.dk/om-euc-nord/ . 2018.* Sidst besøgt 1/6-2018.

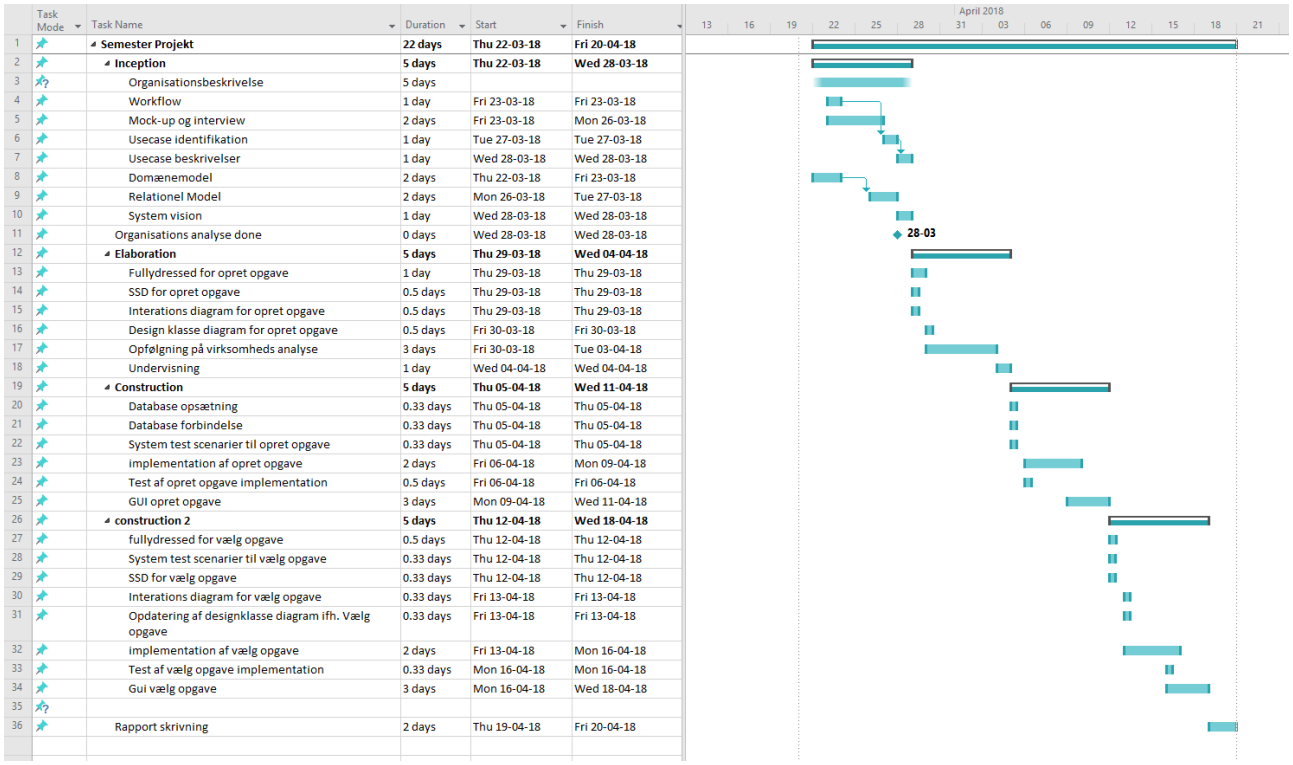
[4] Sun Microsystems Inc. *Java Code Coventions.*
http://www.oracle.com/technetwork/java/codeconventions-150003.pdf, 1997. Sidst besøgt 1/6-2018

[5] Undervisningsministeriets Statistik over gymnasielle institutioner i Danmark. Sidst besøgt 1/6-2018

https://uvm.dk/statistik/gymnasiale-uddannelser/personale-og-institutioner/antal-institutioner

Bilag

A. Tidsplan med Gantt kort.



	Task Mode	Task Name	Duration	Start	Finish
1		**Semester Projekt**	**22 days**	**Thu 22-03-18**	**Fri 20-04-18**
2		**Inception**	**5 days**	**Thu 22-03-18**	**Wed 28-03-18**
3		Organisationsbeskrivelse	5 days		
4		Workflow	1 day	Fri 23-03-18	Fri 23-03-18
5		Mock-up og interview	2 days	Fri 23-03-18	Mon 26-03-18
6		Usecase identifikation	1 day	Tue 27-03-18	Tue 27-03-18
7		Usecase beskrivelser	1 day	Wed 28-03-18	Wed 28-03-18
8		Domænemodel	2 days	Thu 22-03-18	Fri 23-03-18
9		Relationel Model	2 days	Mon 26-03-18	Tue 27-03-18
10		System vision	1 day	Wed 28-03-18	Wed 28-03-18
11		Organisations analyse done	0 days	Wed 28-03-18	Wed 28-03-18
12		**Elaboration**	**5 days**	**Thu 29-03-18**	**Wed 04-04-18**
13		Fullydressed for opret opgave	1 day	Thu 29-03-18	Thu 29-03-18
14		SSD for opret opgave	0.5 days	Thu 29-03-18	Thu 29-03-18
15		Interations diagram for opret opgave	0.5 days	Thu 29-03-18	Thu 29-03-18
16		Design klasse diagram for opret opgave	0.5 days	Fri 30-03-18	Fri 30-03-18
17		Opfølgning på virksomheds analyse	3 days	Fri 30-03-18	Tue 03-04-18
18		Undervisning	1 day	Wed 04-04-18	Wed 04-04-18
19		**Construction**	**5 days**	**Thu 05-04-18**	**Wed 11-04-18**
20		Database opsætning	0.33 days	Thu 05-04-18	Thu 05-04-18
21		Database forbindelse	0.33 days	Thu 05-04-18	Thu 05-04-18
22		System test scenarier til opret opgave	0.33 days	Thu 05-04-18	Thu 05-04-18
23		implementation af opret opgave	2 days	Fri 06-04-18	Mon 09-04-18
24		Test af opret opgave implementation	0.5 days	Fri 06-04-18	Fri 06-04-18
25		GUI opret opgave	3 days	Mon 09-04-18	Wed 11-04-18
26		**construction 2**	**5 days**	**Thu 12-04-18**	**Wed 18-04-18**
27		fullydressed for vælg opgave	0.5 days	Thu 12-04-18	Thu 12-04-18
28		System test scenarier til vælg opgave	0.33 days	Thu 12-04-18	Thu 12-04-18
29		SSD for vælg opgave	0.33 days	Thu 12-04-18	Thu 12-04-18
30		Interations diagram for vælg opgave	0.33 days	Fri 13-04-18	Fri 13-04-18
31		Opdatering af designklasse diagram ifh. Vælg opgave	0.33 days	Fri 13-04-18	Fri 13-04-18
32		implementation af vælg opgave	2 days	Fri 13-04-18	Mon 16-04-18
33		Test af vælg opgave implementation	0.33 days	Mon 16-04-18	Mon 16-04-18
34		Gui vælg opgave	3 days	Mon 16-04-18	Wed 18-04-18
35					
36		Rapport skrivning	2 days	Thu 19-04-18	Fri 20-04-18

B. Tidlige udvekslinger med repræsentanten.

Hvordan kontakter han sine medarbejdere og har han kontakt med andre end dem der er på hans lokation?

- Han har en chef, en makker på stedet og kontakten til andre foregår per telefon eller mail

Hvem laver hvad?

De byder ind på opgaver og deler indbyrdes imellem (typisk også hvem kan bedst/ har mest lyst) Ole tager typisk det meste af morgen runden og generalle all-around opgaver samt fakturering. Henrik tager typisk madkørsel og mere tekniske opgaver. H tager græslåning på Håndbækvej og O tager knivholtvej. Opgaver om alarmer er primært H. Opgaver på skolehjemmet skal begge bookes til, da de ikke går ind på et værelse alene. O tager rundderinger i kælderen. Tage og kontrol (arbejde med stiger eller lign.) skal begge med til. Ellers forefaldne opgaver påtages af den ledige.

Hvordan holder de styr på hvem der laver hvad?

Primært mundtlige aftaler og efter hvem der kan hvad. (Morgen runden afgør opgaverne, de uddelegeres typisk over en kop kaffe)

Hvordan ved de hvilke opgaver de skal løse?

De vurdere selv vigtigheden, så al der omhandler undervisning giver første prioritet. De tekniske ting samt gange mm. Kommer i anden række (meget kommer per intuition, med hvad der er vigtigt og hvad der ikke er)

Hvor får de informationerne fra?

Rengøring lægger et stykke skriftlig besked til dem på skrivebord, ellers mundtlig, telefon eller mail

Hvordan skemalægges der?

Der er faste arbejdstider og opgaverne skal løses inden for dem. 7,5 timer de 4 første dage og 7 timer om fredagen med pauser i arbejdstiden. (Ikke bookning med mindre det er alvorligt og så skal der lægges pauser ind) Mødetid kl 7

Hvilke opgaver er der?

Skolehjemmet har de udvendig og indvendig

Håndbækvej kun udvendig renovation

Knivholtvej både inde og ude

Åbner skolen (alarm og døre slåes fra og åbner selv. Runde af tænding af lys, nogle enkelte døre der skal låses op. Eventuel oprydning)

General vedligehold med bygningerne og eventuelle fejlmeldinger (gået neonrør, lys, varme mm.)

Månedlige aflæsninger af forbrug

Daglig kørsel med mad

Ugentlig udendørs affaldsrunder (tømmer papir, skraldespande askekummer mm.)

Græsslåning ca en gang om ugen i sommeren - begge lokationer

Godkendelse af indkomne faktura - Købte reservedele mm.

Vedligehold af firmabil (vaskes/serviceres efter behov)

Ugentlig runddering af kælder(visuel kontrol) - Tilsyn med installationer (utætte vandrør/ vand i kælder)

Holder øje med grundvandspumper ugentligt

Løbene opgaver så som ændringer af alarm og CTS anlag (varme og ventilation)

Årlig eftersyn af filtre i ventilation

Årlig opsmøring af døre og vinduer

Bestiller tømning af container mm. (efter behov)

Eventuelle andre opgaver (kan de selv klare opgaverne eller skal der bestilles professionelle til det, elektriker eller andre)

Ændringer af invetar (bord og stole) flyttes rundt på efter behov

Meget general opsyn, rundderinger, kontrol og kaffepauser

Bestilles til generalle opgaver

Hejse flag efter behov

Transport af diverse ting (papir til borgerservice mm)

Inspektion af tage og afløb 2 gange årligt (forår og efterår)

Salt ved sne og eventuelt feje efter behov

Diverse andre ting, så de ikke kan bookes (så som møder, kurser og andet)

Har i et EUC system i bruger?

Topdesk som brugere kan melde en opgave generalt på hele skolen. De er kun en en del af dette system, dog bruger de det ikke ret meget, da deres "kunder" ikke bruger det så tit, da det er nemmere at give en opgave mundtligt frem for at skulle logge ind og fejlmelde.

Hvordan bliver det brugt i hverdagen?

Det bliver brugt sideløbende med mail og telefon, for at holde øje med tidsregistrering.

Bruger i noget fra EUC (Canvas/intranet)?

Intranet til nyheder og kan søge på hinanden, da de er så mange ansatte (kartotek). Alle oplysninger ligger også på intranettet(kemikalier, brugsanvisninger mm)

Er der nogle aftaler med andre firmaer, som har specific adgang?

Vvs firma og egne interne håndværkere kan komme ind som de har lyst til (elektriker, maler og tømre)

Bruges der PDA'er / tablets?

Nej, men det kunne være en mulighed(der er snak om det i fremtiden)

Er der nogle information de dagligt skal indberette?

Ikke noget. Kun hvis han henvender sig eller til større opgaver som koster mere end en 5-10.000 (de er selvstyrende)

Har du nogle specifikke krav til et system?

Opgaver ude i fremtiden (lange intervaller imellem) må meget gerne give en indikation for hvornår de skal laves(mulighed for at sætte manuel varsling på en opgave)

C. Interview og muck-up

Gennemgang af mock-up

Forløb over en gennemsnitlig dag

- **Møder ind til de går hjem**

Man har faste rutiner, som at tænde lys osv. Tjekker om der er opgaver fra dagen før og om folk har sendt en mail til dem som hvor de skal lave en opgave. Man kan godt udskyde noget hvis der sker noget andet end det man plejer, f.eks hvis der er et kursus der skal afholdes. Om sommeren, slå græs, tom affald, rydde op. Hjørring og Frederikshavn har ikke særlig meget med hinanden at gøre. Den måde man melder opgaverne ind kan være netbaseret. Det vil tage for lang tid hvis folk skal melde det ind hvis de skal bruge det nuværende system i stedet for bare at ringe. Det er dagligt eller ugentligt skal ikke komme for ofte som en notification, men måske bare om morgen på den dag det skal laves.

Daglige rutiner kunne så med en farve og ugentligt rutiner kunne stå med en anden farver. Dem der ikke sker særlig tit kunne være rart hvis de blev vist.
Akutte opgaver skal komme ind lige med det samme. Det giver mest mening med et uge skema. Man kan ikke bruge noget som er for gamle. Man skal kunne se 7 dage frem fra den dag man er på. Man kan også bruge det til at se hvornår man har tid. Man kan blokere en dag hvis man ikke kan den dag eller om man har fri eller på kursus.
Man snakker sammen om morgen for at aftale hvem der skal tage hvad

Er der ting i ved i skal lave?

- **Dagligt?**
- **Ugentligt?**
- **Månedligt?**
- **Årligt?**
- **Lang tids planlagt?**
- **Udskiftning eller eftersyn?**

Er der ting i ikke ved i skal lave?

- **Er der noget der ofte forekommer som ikke er planlagt?**

Områder og lokaler:

- **Bruger i områder på adresser til fysisk at afgrænse dem? (udendørs? Indendørs? Mere specifikt?)**

Ja, man har områder til at afgrænse hvor man er henne. Indenfor bruger man tal for at sige helt præcist hvor man er henne.

- **Kan der være områder i områder? (Kravlegår-område i et udenfor område)**
- **Har i lokale-navne til alle lokaler?**

Alle rum har et nummer eller bogstaver

Informationer omkring opgaver:

- **Når man skal vælge en opgave (Detail task vindue i pencil) er det så nok informationer til at kunne påbegynde opgaven? Er der måske for meget information?**

Der skal være en adresse og der skal være en tekst

- **Når i opretter en opgave, hvilke informationer er så nødvendige?**

Opretning af opgaver:

- **(opret opgave vindue i pencil) Er der nok informationer, mangler der nogen? Er der for mange? Noget der er unødvendigt?**

Det kan være aktuelt hvis man har en periode for længere opgaver.

Påmindelse:

- **Hvilke valgmuligheder vil du gerne have når du skal have påmindelser? (intervaller? Antal dage før? Specifik dag?)**

Det gør ikke noget hvis man før en påmindelse for en årlig opgave en uge før. Man kan få tre muligheder f.eks 1 -2 dage før eller en uge.
Påmindelser er bedst hvis de kommer lige når arbejdet begynder, så om morgen.

- **Hvad med en beskrivelse?**
- **Dagen påmindelsen vil forekomme?**

Gentagelser:

- **Hvilke muligheder skal der være når det skal være en gentagelig opgave? (daglig, ugenligt, måndeligt og årligt? Andre intervaller?)**

Andre medarbejderes indflydelse:

- **Hvor meget indflydelse har lederen på hverdagen?**
  - **Hvad har de indflydelse på?**
- **Hvor meget indflydelse har underviserne på hverdagen?**
  - **Hvad har de indflydelse på?**
- **Er der andre der har indflydelse på det i laver?**
  - **Hvad kan de have af indflydelse?**

Det giver mening med mappen med 3 talet.
Når man laver en opgave skal man selv kunne vælge hvor det er henne, det behøver ikke at være noget der står i en database. Det er en god ide med en adresse. Det er bedre hvis man selv kan skrive ind for opgaven skal laves.
Man kan selv vælge om man vil se det ugentlig eller månedtligt.

Der er fint med farvelægning for at se prioriteten på opgaven, hvis der er en opgave der ikke er blevet taget i et stykke tid skal skal den blive en høj prioritet. Hvis begge er syge eller er på ferie så kan man tage opgaver fra en anden lokation end den man normalt er på.

Pedeller kan bedømme hvad opgaven handler om f.eks el eller vand og så smide det hen til elektrikeren så han kan lave opgaven.
Gruppe lederen kan uddele opgaver, hvis man f. eks at en person er et sted hvor der er en opgave der skal laves. Andre en gruppelederen skal ikke kunne skema lægge en andens opgaver. Skal ikke kunne vælge en person hvis denne er på feire.

Hvis man har opgaver en hel uge frem så skal man ikke kunne få opgaver i løbet af denne uge

Det vil være anvendeligt hvis man kunne vinge af hvornår opgaven er færdig, den der har oprettet opgaven skal kunne se hvornår opgaven er blevet lavet færdig(en notifikation så man kan se at den er blevet lavet).

Lærerens startside kunne være opret opgave, mens at pedellerens er kalenderen. Kan se de tidligere opgaver der er blevet lavet og se hvem der har lavet den og se om der er nogen her taget opgaven og se om der ikke er nogen der har taget den. Farvekoder for ikke taget, igangværende og fuldført. Opret opgaver er primært for undervisere.

Der kunne være navn, afdeling og telefonnummer(se opgave). Man skal kunne skrive ens initialer også fylder den selv resten ud.