

System til vagtplanlægning

Virkelighed og modeller

Gruppe A312, Software

Det Teknisk- Naturvidenskabelige Basisår
Aalborg Universitet
19. december 2005

Titel:

System til vagtplanlægning

Tema:

Virkelighed og modeller

Projektperiode:

P1, efterårssemesteret 2005

Projektgruppe:

A312

Deltagere:

Allan M. Christensen
Anh T. Nguyen Dao
Esben S. Pedersen
Kim J. Nissen
Martin Midtgaard
Peter H. Bøg

Vejledere:

Finn Jeppe Jepsen
Jette E. Holgaard

Oplagstal: 13

Sidetal: 77

Bilagsantal og -art: 2 sider og 1 cdrom

Afsluttet den 19. december 2005

Synopsis:

Dette projekt omhandler tilblivelsen af et system til vagtplanlægning. Der tages udgangspunkt i Nettos behov for et sådan system samt mangler i deres eksisterende system. For at vi kan sætte os ind i den eventuelle problematik med Nettos nuværende system, skal butikschefen for Netto, Danmarksgade i Aalborg, interviewes. Ud fra dette interview kan vi opstille krav til systemet.

Vi ønsker at lave et automatisk vagtplanlægningssystem, der skal kunne ligge en 16 ugers vagtplan. Udover vagtplanlægning skal systemet også kunne holde styr på vagtbyt, osv.

Vi vælger at systemet skal være web-baseret for bl.a. også at nå ud til de ansatte i hjemmet. Desuden skal systemet programmeres i sproget PHP. Vores system skal understøtte flere varianter af klienter. Vi vælger at begrænse os til en web-klient til pc, og en wap-klient til mobile kommunikationsenheder. Yderligere skal email og sms benyttes til notifikation.

At lave en algoritme til automatisk vagtplanlægning er ikke en umulig opgave, men at tage højde for alle faktorer kan blive vanskeligt. Vi har brugt backtracking til at lave vores algoritme. Den tager dog kun højde for de vigtigste faktorer.

Forord

Denne rapport er skrevet af seks Softwareingeniør studerende på Aalborg Universitet som en del af deres P1 projekt. Projektet startede den 10. oktober 2005 og blev afsluttet den 19. december 2005. Gruppen består af samme medlemmer som i P0 projektet. Gruppen har i projektforsøbet besøgt Netto, Danmarksgade, og bl.a. brugt denne som kilde.

Der vil undervejs være referencer til diverse bilag. Disse bilag er vedlagt rapporten, men behøves ikke læses som en del af rapporten. De kan læses for at få yderligere information om et givent emne. Der vil ved referencen stå på hvilken side bilaget kan forefindes. Referencer til litteraturlisten vil blive gjort med [X], hvor X repræsenterer den nummererede kilde på litteraturlisten.

Til denne rapport høre et produkt i form af et stykke software. Dette software er lavet i PHP og bruger MySQL. Produktet er vedlagt på cd-rom.

Undervejs vil der være flere udklip af koden. Grundet sidste-minuts ændringer af denne, kan udklippene være forældet i forhold til den aktuelle kode.

Gruppen vil godt sige tak til følgende personer:

Finn Jeppe Jepsen og Jette E. Holgaard, vores vejleder og bi-vejleder.

Martin B. Olsen, butikchefen i Netto, Danmarksgade.

Gruppe A313, vores test og nabo-gruppe.

God fornøjelse med rapporten.

Allan M. Christensen

Anh Tuan Nguyen Dao

Esben S. Pedersen

Kim Jung Nissen

Martin Midtgaard

Peter Heino Bøg

Indhold

1	Problemanalyse	6
1.1	Målgruppe	7
1.1.1	Virksomhedsanalyse	9
1.2	Behovsanalyse	11
1.3	Krav til systemet	13
1.3.1	Brugerens krav	13
1.3.2	Programmørens krav	18
1.4	Konklusion på analysedelen	20
2	Problemformulering	22
3	Løsningsforslag	23
3.1	Arkitektur	24
3.1.1	Klient-Server model	24
3.1.2	Databasen	25
3.1.3	MySQL	28
3.1.4	Klient	32
3.1.5	Server	32
3.2	Dokumentation	34
3.2.1	Generelt	34
3.2.2	Sikkerhed	35
3.2.3	Administration	40
3.2.4	Brug af systemet	51
3.2.5	Afgrænsning	64
4	Afslutning	66
4.1	Vurdering	67
4.1.1	Test af systemet	67
4.1.2	Vores vurdering	68
4.2	Konklusion	69
4.3	Perspektivering	70
5	Litteratur	72
5.1	Kildekritik	73
6	Bilag	74
6.1	Bilag 1 - Interviewguide	75
6.2	Bilag 2 - Lovgivning omkring arbejdstider	77

Indledning

I dette projekt vil opbygningen af et system til vagtplanlægning blive gennemgået. Mange virksomheder har i forvejen et sådan system, men vi vil prøve at lave nye funktioner der kunne forbedre systemerne. Blandt andet vil vi lave mulighed for automatiske vagtplanlægning og muligheden for at bruge systemet via web og wap.

Systemet skal være IT-baseret, da vi mener, at det vil give større fleksibilitet og kommunikationsmuligheder, end fx gamle seddel-vagtplaner. For at systemet adskiller sig fra andre nuværende systemer, vil vi implementere nye funktioner¹, der kan gøre arbejdet lettere for vagtplanens administrator. De nye funktioner vil forhåbentligt gøre vagtplanlægningen både hurtigere og lettere for administratoren.

Vi vil forsøge at lave vores system fremtidssikret. Det vil kunne spare firmaer penge, hvis det ikke er nødvendigt for dem at omstille til andre systemer, da vores system helst ikke skulle forældes inden for få år.

Systemet vil blive udviklet med udgangspunkt i en specifik virksomhed, men det kan gøres nemt at tilpasse systemet til andre virksomheder.

- Hvem har behov for et sådan system?
- Hvad skal dette system kunne?
- Hvordan kan en vagtplan ligges automatisk?
- Hvilke krav er der til et sådan system?

For at finde svar på nogle af disse spørgsmål, vil vi lave et interview med en dagligvarebutik.

¹Se funktionsafsnittet side 15

Kapitel 1

Problemanalyse

1.1 Målgruppe

Når vi skal udforme et system til vagtplanlægning, er det vigtigt at definere, hvilken målgruppe vi skal udvikle vores system til. Der skal tages højde for vores målgruppes tekniske kendskab til IT. Vi er nødt til at overveje brugervenlighed, nem tilgang og en godt beskrevet vejledning, som er tilpasset vores målgruppe.

Valg af firma

Mange virksomheder har brug for et system til vagtplanlægning. Vi kan forestille os en folkeskole, som skal have skemalagt næste skoleår. Der skal tages højde for lokaler, hvert enkelte læreres forskellige kompetencer, perioder hvor de skal på kurser og meget mere. Sådant et system ville stille en del udfordringer for programmøren.

Ydermere kunne der tages udgangspunkt i en virksomhed, som for eksempel en dagligvarebutik, som skal have planlagt næste måneds vagter. I et sådant system skal der tages højde for blandt andet ungarbejdere, medarbejdernes kompetencer, feriedage med videre.

Vi skal finde en butik, som er passende til at tage udgangspunkt i. I en dagligvarebutik skal der tages højde for en række arbejdsmarkedsregler, ungarbejdere, der ikke må være alene på arbejde, nøgleansvarlige medarbejdere og meget mere.

Fokus på Netto

Vi har valgt at bruge butikskæden Netto som udgangspunkt til vores system til vagtplanlægning. Da en af gruppemedlemmerne er ansat i Netto, Danmarksgade, har vi let tilgang til denne Nettobutik. Ligesom andre dagligvarekæder har Netto også brug for et antal medarbejdere, der kan dække en arbejdsdag, dermed har de brug for en vagtplan. Denne vagtplan kan enten laves manuelt eller ved hjælp af et færdiglavet IT-system. Netto, Danmarksgade, bruger et computersystem, hvor butikschefen planlægger sine såkaldte basisplaner manuelt¹.

Når en vagtplan planlægges spiller økonomien en rolle. Nettos system udregner samtidig med planglægning også omkostningen for de enkelte arbejdsdage. Altså sikrer systemet sig, at en butikschef er klar over, at lønomkostningen enten er overstøjet, eller at der er penge til gode.

Ved manuel planlægning kan der let opstå menneskelige fejl, såsom forkerte udregninger eller at glemme de ansattes ønsker omkring arbejdstider. En medarbejder kan også få for mange timer den enkelte dag eller i den enkelte uge, hvilket er i modstrid med overenskomsten².

Betydning for den enkelte medarbejder

Den enkelte medarbejder kunne få en oversigt over, hvornår denne skal på arbejde og mulighed for vagtbytte. Ved vagtbytte kunne der automatisk blive sendt en forespørgsel til den person, der byttes med.

Deres nuværende system kræver, at medarbejderne enten selv henvender sig til hinanden eller til butikschefen. Under alle omstændigheder skal butikschefen underrettes. Den automatiske opdatering af vagtplanen, ved et eventuelt vagtbytte, kunne lette arbejdet for butikschefen.

Med det automatiserede system vil medarbejderne ikke opleve de medarbejdermangler, der kunne fremkomme når en kollega ikke tidsnok er blevet informeret om at komme på arbejde.

¹Se 1.2 behovsanalysen side 11

²Se 1.2 behovsanalysen side 11 og bilag 2 side 77

Med Nettos nuværende system har medarbejderne ingen mulighed for at tjekke deres vagtplan hjemmefra. De kan dog vælge at få et udprint af planen, men hvis der sker ændringer, er det ikke sikkert, at medarbejderen bliver underrettet³. Med det automatiserede system, kunne det være muligt for hver enkelt medarbejder at tjekke deres vagtplan enten via internettet eller ved hjælp af deres mobiltelefon (wap). Hvis der sker en ændring i vagtplanen, kunne systemet underrette den pågældende medarbejder via sms eller email.

Systemet skal kunne vedligeholdes af butikschefen

Brugervenligheden omkring et system er meget vigtigt. I en butik er chefen nødvendigvis ikke altid til stede, og de butikansvarlige må overtage administrationen af butikken. Butikschefen er ikke nødvendigvis ekspert i computere, og derfor er det vigtigt, at vi sætter brugervenligheden så højt oppe på vores prioreringsliste som muligt. Dermed betyder det ikke, at vi skal lave vores program så enkelt som muligt. Det kan sagtens være avanceret, men funktionerne skal sættes op således, at de ansatte kan betjene dem hurtigt og enkelt.

Automatisering af opgaver

I en butik som Netto er det kunderne, der er det vigtigste. Vagtplanen er blot en oversigt for, hvornår de enkelte ansatte skal betjene kunderne. Den tankegang benytter vi os af. Vores system skal således være automatiseret sådan, at administrationen ikke kræver mere tid end nødvendig, fx til vagtplanlægning. Dermed er der mere tid til kunderne.

Der vil som oftest være små fejl, når en vagtplan skal lægges. Disse fejl er tidskrævende at rette, og ofte overses enkelte fejl. Her vil vores automatiserede system være en fordel. Butikschefen indtaster oplysninger om de forskellige ansatte, og systemet vil derefter automatisk generere en vagtplan. Dette vil spare en del tid og dermed også penge. Oplysningerne består af, hvornår medarbejderen gerne vil på arbejde, og hvornår medarbejderen ikke kan møde på arbejde (fx ferie).

En anden mulig automatiseret funktion i vores system omhandler videregivningen af informationer omkring vagtbytning eller vagtændring. Med et automatiseret system kunne chefen i en butik slippe for at kontakte de ansatte, når der er en vagtændring. I stedet for at skulle ringe rundt til de ansatte, kunne de få en besked via email eller sms. Dette vil spare endnu mere tid, og chefen kan bruge tiden på noget andet.

Delkonklusion

Vi har valgt at tage udgangspunkt i dagligvarekæden Netto. Vi vil lave et system til vagtplanlægning, som kan lette arbejdet for såvel butikschefen som de ansatte, når der skal lægges vagtplan for de næste 16 uger⁴.

Systemet skal, om muligt, automatisk kunne lægge en vagtplan ud fra givne oplysninger om medarbejderens arbejdstider og kompetencer. Vi vil lade systemet vælge mulige kandidater til et eventuelt vagtbytte ud fra oplysningerne om kompetencer, lovgivningen, og om medarbejderen overhovedet kan arbejde på dette tidspunkt. Ved vagtbytte skal systemet selv tage kontakt til byttekandidaterne.

³Se 1.2 behovsanalysen side 11

⁴Se 1.2 behovsanalysen side 11

1.1.1 Virksomhedsanalyse

I Netto er der mange forskellige opgaver, som skal løses hver dag. For at løse disse opgaver er der forskellige slags medarbejdere i Netto. Alle medarbejdere kan lave andet end det, de er ansat til, dog kan en servicemedarbejder ikke være kasseassistent, eller det der kræver et større ansvar. Netto's daglige opgaver er som følgende:

- Åbne/lukke butikken
- Bestille varer
- Sidde ved kassen
- Betjene kunderne
- Lægge varer på hylderne, samt rydde op på hylderne
- Gøre rent

Medarbejderne i Netto er alle ansat med en stilling. Herunder er en liste over stillingerne i Netto.

- Butikschef
- 1. mand/dame
- 1. assistent
- Butiksassistent
- Elev
- Kasseassistent
- Servicemedarbejdere (flaskedreng/pige)

Butikschefen er den øverste i en Nettobutik. Han tager de vigtigste beslutninger og tager sig af det administrative arbejde. Det er også ham, der har ansvaret for sine medarbejdere såvel som butikken. Hans job går ud på at holde butikken kørende og sørge for, at butikken giver overskud. Han bestemmer, hvem der skal ansættes eller fyres. Derudover kan han også fungere som vikar, der skal afløse en medarbejder i tilfælde af den pågældende er fraværende på grund af sygdom, og det ikke er muligt at finde en anden medarbejder som afløser. Når det administrative er færdiggjort, fungerer en butikschef som 1.mand/dame, hvor han åbner og lukker butikken.

En 1.mand/dame har til opgave at hjælpe butikschefen med det administrative arbejde, når butikschefen ikke er til stede. Derudover sørger de for, at butikken hele tiden har de annoncerede varer (varepåfyldning). Det er også 1.mand/dame, der skal åbne butikken og modtage varer om morgenen, tælle pengeskabet op om aftenen og derefter lukke butikken. Generelt fungerer en 1.mand/dame som butikschefen, når selve butikschefen ikke er til stede.

1.assistentens job er noget lignende en 1.mand/dames. Han skal åbne og lukke butikken og fungerer som den administrerende, når både butikschefen og 1.mand/dame ikke er tilstede. Deres primære arbejde går ud på at sætte varer op på hylderne, hjælpe kunderne eller ekspedere kunderne ved kassen.

I modsætning til de tre førnævnte stillinger har en butiksassistent ikke meget ansvar. Han skal først og fremmest sørge for at hjælpe kunderne og butikken ser ordentlig ud. Derefter hjælper han med at ekspedere kunderne ved kassen. I nødstilfælde kan en butiksassistent blive 1.assistent over en kortere periode.

Netto kan vælge at ansætte en person som elev. Eleven fungerer som en butiksassistent, dog med den undtagelse, at de ikke kan fungere som 1.assistent.

En kasseassistent er ansat til primært at sidde ved kassen og ekspedere kunderne. For at få give en kasseassistent lidt afveksling, kan han sætte varer på hylderne eller rydde op.

Som servicemedarbejder er man ungarbejder. Her vil det sige, at man intet administrative ansvar har. En servicemedarbejder er et andet ord for en flaskedreng/pige. Deres job går ud på at rydde op i butikken. De har ansvaret for, at lageret ser ordentligt ud. Hvis flaskeautomaten bliver fuld, skal servicemedarbejderen tømme flaskeautomaten, og flaskerne fra flaskeautomaten skal sorteres, pakkes og klargøres til afsendelse. Ved lukketid skal servicemedarbejderen feje og vaske gulvet. Hvis kasseassistenterne har brug for hjælp - sætte kurverne på plads, sætte en vare på plads eller tørre noget op - tilkalder de en servicemedarbejder med en ringeklokke ved kassen.

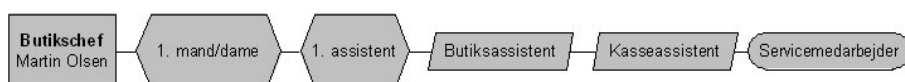
1.2 Behovsanalyse

Det følgende er resultatet af vores interview med butikschefen, Martin B. Olsen, fra Netto Danmarksgade den 19. oktober 2005. Svarene fra Martin B. Olsen har vi samlet til et referat. Udover at interviewe Martin B. Olsen, fik vi en lille demonstration af Nettos nuværende vagtplansystem, som vi har beskrevet under "Nettos system". I afsnittet om Nettos system har vi kun beskrevet den del af system, vi havde tilladelse til at se.

Regler for medarbejdere

I en butik som Netto står chefen for det administrative arbejde. Han kan nødvendigvis ikke være til stede hele tiden, derfor har han nogle 1. mænd/damer til at hjælpe sig, som også står for det administrative arbejde. Da en Netto butik er stor, er det ikke nok med en 1. mand/dame, Netto bliver nødt til at have assistenter. Her har Netto både en 1. assistent og en butiksassistent. Kun 1. assistenten har adgang til de administrative funktioner i fx butikkens computer. En butiksassistenten står for at holde butikken pæn og betjene kunderne. Alle disse ansatte hører under kategorien voksenarbejdere. En anden arbejdskategori er ungarbejdere. Ungarbejdere er medarbejdere under 18 år. De kan ansættes som servicemedarbejder, der sørger for at holde butikken pæn og ren.

Alle 1. mand/dame og 1. assistenter er de såkaldte "nøgleansvarlige" og styrer butikken, når chefen ikke er til stede. Det er dem, der står for åbningen eller lukningen af butikken. De nøgleansvarlige kommer alle igennem et internt kursus, hvor de lærer lukkereglene og håndtering af penge. Arbejdsloven omkring medarbejdere i en butik gælder for alle, med nogle få undta-



Figur 1.1: Oversigt over netto's ansættelsesforhold, opstillet efter rang.

gelses. Loven siger, at en fastansat højst må arbejde to dage om ugen efter kl. 17.45. Alle ansatte skal have mindst 11 timers hvile før den næste arbejdsdag. Desuden har de ret til at kende deres vagtplan 16 uger frem.

En ungarbejdere kan ansættes, når de enten er fyldt 16 år eller afsluttet 9. klasse. De kan dog ikke være kasseassistenter, før de er fyldt 17. Ungarbejdere må heller ikke arbejde efter kl. 20.15, men de enkelte butikker kan lave deres interne aftaler, så ungarbejdere kan arbejde til kl. 20.30.

En fuldtidsansat har 37 arbejdstimer om ugen, som ikke må overskrides, mens en deltidsarbejder kan arbejde ligeså meget, vedkommende ønsker. Der skelnes mellem 3 typer af ansatte, en fastansat, en funktionær og en deltidsarbejder (ikke-funktionær). En funktionær defineres som en ansat, der har over 8,45 timer om ugen, men arbejdsplanen for en funktionær er flydende, altså at arbejdstiderne ikke nødvendigvis ligger fast i de 16 uger, som en arbejdsplan skal strække sig over. En fast ansat derimod har et fast antal arbejdstimer og en fast arbejdsplan, altså at arbejdstiderne er ens fra uge til uge.

Ud over de ansatte i Netto, kan Netto vælge at optage en elev. Disse elever hører under gruppen funktionære. Her gælder reglerne for en funktionær også for en elev.

Når to ansatte skal bytte vagter, skal de ifølge loven, møde op i butikken og underskrive en vagtbytningskontrakt. Herefter vil en af administratorerne godkende kontrakten. I visse butikker er det nok med, at chefen styrer godkendelse af bytninger. Det vil sige, at de ansatte ikke behøver at møde op i butikken, hvis de skal bytte en vagt, men kan nøjes med at give chefen beskeden.

Nettos system til vagtplanlægning

Chefen og 1. mand/dame kan lave en vagtplan. Ifølge loven har medarbejderne ret til at kende deres arbejdsplan 16 uger frem. Når en sådan plan skal laves, laver chefen en til fire basisplaner, én for hver uge. Herefter ganger han basisplanerne op således, at han får en vagtplan for 16 uger. Herefter kan 1. mand/dame gå ind og rette, hvis dette skulle være nødvendigt. Inden den pågældende uge udløber, skal chefen eller 1. mand/dame godkende vagtplanen for den efterfølgende uge for at sikre sig, at de eventuelle nye ændringer er registreret. I løbet af en arbejdsdag kan der være nye ændringer til vagtplanen (en medarbejder kan blive syg). Derfor skal den lukkeansvarlig godkende eller tilføje de nye ændringer til vagtplanen om aftenen.

Betjeningen af systemet kræver et endagskursus. Dette kursus er kun for butikschefen, herefter instruerer butikschefen de nøgleansvarlige i brug af systemet.

Når en vagtplan er færdig, udprintes den og hænges op på en tavle. Dog har hver enkelt ansat ret til at få en udprintning af deres vagtplan over 16 uger. Her får de nøgleansvarlige tilsendt vagtplanen pr. email. Denne emailadresse har Netto tilstedt de nøgleansvarlige. Under planlægningen af vagtplanen holder Nettos system styr på arbejdslovene (se "Regler for medarbejdere" oven over). Disse arbejdslove er indbygget i systemet og vil udmelde fejl, hvis de overtrædes. Dog kan chefen vælge at godkende vagtplanen, selvom systemet melder fejl. Chefen laver en vagtplan ved at finde ud af, hvem og hvor mange ansatte han skal bruge for mandag, tirsdag osv. Herefter vælger han en arbejdstid for hver enkelte medarbejder, det vil sige, manuelt indtaster arbejdstiderne for de enkelte medarbejdere ind i systemet.

Martin B. Olsen er tilfreds med den måde, som systemet er opbygget på. Han synes, at det er brugervenligt og nemt at navigerer rundt i, når man skal bruge en bestemt funktion eller finde en menu. Selve opbygningen er meget simpelt, funktioner, der ledes efter, kan nemt findes, og der er ikke alt for mange menuer som kan forvirre brugeren. Under hver enkelt menu er der en masse funktioner, der ikke er overvældende for brugeren.

Nettos system fungerer ved hjælp af et internt, lukket system. Der skal bruges et program, som kun Netto har, for at kunne bruge de administrative funktioner som fx vagtplanlægningen. Oplysningerne mellem klienten (butikkens computer) og serverne sendes ved hjælp af internetforbindelse. To store servere sørger for at gemme data fra alle Netto-butikker i Danmark. Når vagtplanen skal laves, skal Martin B. Olsen logge på de to store servere. En internetafbrydelse vil resultere i, at Martin B. Olsen ikke kan lave, rette, tilføje eller godkende vagtplanerne.

Forbedringer

Ifølge Martin B. Olsen har Nettos system nogle få mangler. Hvis hele vagtplanen kan generes automatisk, ville der spares tid. Problemet ved at planlægge en vagtplan manuelt er, at der er større risiko for fejl. Her vil et automatisk system reducere menneskelige fejl. Ved menneskelige fejl, mener Martin B. Olsen, at han kan overse en medarbejders ønske omkring arbejdstider eller sætte flere på arbejde, end der er behov for. En anden fejl kan være at sætte en medarbejder til at arbejde mere end 37 timer om ugen og derefter glemme at give vedkommende mindre timer den efterfølgende uge eller senere. Et andet problem med Nettos nuværende system er, at hver medarbejder - udover de nøgleansvarlige - kun kan få deres vagtplan på papir. Her er problemet, at hvis der sker en ændring i vagtplanen, kan det ske, at Martin B. Olsen eller de nøgleansvarlige glemmer at underrette den pågældende medarbejder. Nettos system er som nævnt et lukket system, hvor brugeren kun kan bruge systemet i butikkerne. Der er altså ingen mulighed for, som chef, at planlægge en vagtplan hjemme. Denne mulighed er ønsket blandt butikschefer, mener Martin B. Olsen.

1.3 Krav til systemet

1.3.1 Brugerens krav

Da det er brugerne, der skal benytte systemet, er det vigtigt at systemet også opfylder deres krav. Hver enkelt bruger har forskellige krav, men vi skal ikke nødvendigvis tage højde for alle - kun de vigtigste. Kravene kan bl.a. være, at systemet skal være nemt at gå til, eller at der skal være bestemte funktioner, integreret i systemet. I de følgende afsnit vil vi bestemme hvilke funktioner, der skal være i systemet.

Sikkerhed

Firmaer søger altid efter, at få den bedste sikkerhed der findes, for at beskytte sig imod uvedkommende, skadeligt software eller vira. Grunden til dette er, at stort set alle firmaer har en eller anden form for fortrolige oplysninger, som fx medarbejdernes private oplysninger eller arbejds-papirer for et nyt produkt.

Kravene til sikkerhed varierer fra firma til firma, for ikke alle firmaer har de samme oplysninger og filer, som de vil beskytte. I vores system har vi også en eller anden form for prioritering af vores sikkerhed, så der er nogle af delene af systemet, som vil have større sikkerhed end resten.

Når der logges på systemet, skal der være en sikker forbindelse mellem klienten og serveren. Dette er for at forhindre andre i at opsnappe kodeordet, og derefter logge ind på systemet. Når brugeren beder om at få sin vagtplan frem, behøver sikkerheden ikke være så stor for den forbindelse, da de oplysninger der bliver sendt frem og tilbage, ikke har den større værdi for andre end brugeren. Derimod skal sikkerheden være meget stor, når brugeren prøver at bytte sine vagter. Det nytter ikke noget, at andre kan få lov til at gå ind og bytte vagt på brugerens vegne. Administratoren skal også have en meget stor sikkerhed, for vedkommende har alle rettigheder til at gå ind og ændre i vagtplanen efter smag og behag. Hvis sikkerheden ikke er god nok her, så kan en uvedkommende få lov til at logge ind som administrator og begynde at ændre på vagtplanen.

Vigtigheden i at sikkerheden er i orden, er meget stor, da andre ikke skal have lov til at lave ændringer inde i systemet. Der kan skabes eller laves sikre forbindelser mellem klienten og serveren, ved hjælp af kryptering. Der findes mange forskellige krypteringer, men det er ikke alle, der er lige sikre. Når et system er webbaseret, er det vigtigt at krypteringen er meget stærk, så den ikke bliver brudt. Derfor kan man fx bruge SSL standarten, der benytter anerkendte krypterings algoritmer, og er en standart der stort set understøttes af alle webbrowsere i dag. Ved hjælp af fx denne standart, kan der laves en sikker forbindelse mellem klienten og serveren, og derved formindskes muligheden for opsnapping af kodeord eller anden vigtig information.

Udover at lave en kryptering, er der også mulighed for at bruge en lukket linje. På en lukket linje er sikkerheden helt i top, da det kun er én computer kan få adgang til serveren. Men for at bruge computeren, skal der også indtastes et kodeord, så dette gør systemet endnu mere sikkert.

De data, som findes på serverne, må ikke gå tabt, da dette kan være katastrofalt for firmaet. Det er derfor vigtigt, at der bliver taget backup minimum hver dag for at sikre, at der er mulighed for at gendanne de tabte data, hvis systemet skulle gå ned.

Interface/brugervenlighed

Brugerne af systemet har altid krav til, hvordan brugervenligheden og interfacet skal være. Disse krav kunne muligvis være, at et system skal være brugervenligt og nemt at gå til, og interfacet skal være stilrent og flot.

For at gøre systemet brugervenligt, nytter det ikke, at have en masse ubrugelige funktioner som blot forvirrer brugeren. For at forhindre dette, skal funktionerne være begrænset til de funktioner, som den givne bruger får brug for. Ved hjælp af dette, bliver det nemmere for brugeren, at finde rundt i systemet og bruge det.

Systemet skal dog ikke miste sin funktionalitet pga. at det skal være brugervenligt og simpelt, dette gælder især for administratoren. Administrator-klienten skal have en del flere funktioner end bruger-klienten, da han skal vedligeholde og kunne ændre data i systemet. Han skal også have mulighed for at kunne ændre oplysningerne i databasen, hvis dette bliver nødvendigt. Selv om administratoren har flere funktioner end en normal bruger, må disse ekstra funktioner ikke gå ind og forvirre administratoren. Vedkommende skal stadig have overblik over systemet. Her kan henvises til interviewet af Martin B. Olsen, Netto's butikchef⁵.

Der er blevet udarbejdet en teori - gestalt loven - der fortæller om, hvordan vi opfatter billeder og farver. Der er flere forskellige regler under denne gestalt teori, og den mest basale af dem er prägnanz loven. Denne lov siger, at vi forsøger at se objekter på en god gestalt måde. Dvs. at det fx kan være begreber som orden, simpelthed, symmetri osv. I grove træk handler gestalt teorien om, hvordan vi opfatter ting i dele, som vi mentalt sætter sammen til en helhed. Der er flere love, som kan læses på wikipedia's side om emnet[1].

Pga. at vi ser ting som dele, og sætter dem sammen til en helhed, skal opbygningen være på sådan en måde at hver enkelt del er flot at se på. Derved ser brugeren, at systemet er flot, simpelt og behageligt at kigge på som en helhed. Farverne bliver indblandet, da vi også ser hver enkelt farve som en del af en helhed. Farverne skal derfor passe sammen og være behagelige for øjet.

Mulighederne for designet er store, her kan en af de åbenlyse muligheder være, at designet skal være stilrent og flot. Dette er en mulighed, men da designet ikke er det vigtigste, men det funktionelle er, behøver designet nødvendigvis ikke være stilrent og flot.

Klient/server/database

Der findes mange forskellige måder, at køre et vagtplanlægningssystem på. Der er mulighed for at lave et "stand-alone" program, hvor det skal hentes ned, og køres fra den pågældende computer for at få adgang til vagtplanen. Dette kan fx laves i C++ eller Delphi, som er programmeringssprog. Netto bruger et "stand-alone" program, for at kunne logge på deres hovedserver. Her kræves en bruger og et kodeord, for at kunne logge ind på systemet. Når brugeren er blevet godkendt af systemet, henter programmet al nødvendig data, så brugeren har mulighed for at ændre og gemme nye informationer på serveren.

Der er dog også andre alternativer såsom wap. Med wap kan man hente sin vagtplan ned, og se hvordan vagtplanen ser ud. Dette kræver dog, at medarbejdernes mobiltelefoner understøtter wap, for at de kan bruge denne funktion. Når wap skal bruges er der dog en pris at betale. Prisen bliver sat enten ved forbrugsafregning eller tidsafregning. Dette kan blive dyrt især, når man tænker på, hvor billig en internetforbindelse er i dag, og hvor mange muligheder man får i forhold til wap.

⁵Se 1.2 behovsanalysen side 11

Hvis programmet bliver web-baseret, behøver man ikke tænke på, at skulle hente et eksternt program ned for at kunne lave om på vagtplanen. I stedet kan man gøre det med sin web-browser. For at lave sådan en database, hvor brugeren kan logge ind og tjekke sin vagtplan, skal der være en server til rådighed, hvor informationerne er gemt på. Serveren, der indeholder disse informationer, skal have et højt sikkerhedsniveau for at forhindre, at uvedkommende kan rette i informationerne og derved tilføre firmaet skade.

At lave systemet i HTML er ikke nok. HTML er nemlig kun et markup sprog og kan derfor ikke gøre andet end at præsentere tekst på en speciel måde. Vi skal derfor bruge et serverscriptet såsom PHP eller ASP. Brugeren sender en forespørgsel til serveren som så ”compiler” serverscriptet og sender en HTML side tilbage. PHP har en stor fordel fremfor ASP, og det er at det er gratis. Desuden understøtter det en del forskellige databaseformer som fx MySQL, Informix og Oracle.

Når der laves webprogrammering, skal der tages højde for, at ikke alle bruger Internet Explorer, flere og flere bruger alternativer som Firefox og Opera. Det ville derfor være en god ide at kode i W3C⁶ standart, som er standarden for bl.a. HTML. Derved bliver vagtplanlægningssystemet kompatibelt med flere forskellige browsere.

Hvis systemet laves på wap eller web, bliver det nemt at se sin vagtplan lige meget hvor henne i verden, man befinder sig, da det hele ligger online på internettet og derfor er nemt at få adgang til.

Brugeren af et vagtplanlægningssystem, vil oftest gerne have, at systemet er nemt at komme i gang med og nemt at få adgang til. Derfor kan en af ulemperne ved et ”stand-alone” program være, at hvis brugeren ikke har en særlig stor viden om computere. Det ville måske være mere brugervenligt, hvis brugerne bruge systemet over web.

Næsten alle har idag adgang til internettet (se figur 1.2), og det ville derfor være nemmere for brugeren, hvis de blot skal ind på en internet side, og logge på systemet via sin browser. Her skal brugeren ikke installere noget på sin computer, for at få systemet til at virke. Brugeren skal blot logge ind, og gå i gang med at arbejde med vagterne.

At næsten alle i dag har adgang til internettet, kan ses ud fra følgende tabel fra Danmarks Statistik:

Befolkningens adgang til internet efter adgang, type og tid	
	2004
Adgang til internet i alt	
16-19 år	96
20-39 år	90
40-59 år	89

Enhed : Procent

Figur 1.2: Tabel fra Danmarks Statistik

Mulige funktioner

Et vagtplanlægningssystem kan indeholde en enorm mængde muligheder. Da vores tid er knap, må vi begrænse os til kun at implementere nogle af disse funktioner. I de efterfølgende afsnit vil vi nævne forskellige funktioner, diskutere gode og dårlige sider ved dem, og derefter undersøge om det er funktioner, vi vil forsøge at implementere i vores system.

⁶The World Wide Web Consortium - www.w3.org

Vagtbytte

Vagtbytte er en meget vigtig del af et vagtplanlægningsystem. Muligheden for at der kan byttes vagter. Spørgsmålet er så bare hvem der skal kunne gøre det og i hvor stort et omfang. Det ville ikke give meget mening at lave et system uden mulighed for vagtbytte.

Denne funktion kan udvides på mange måder. De fleste af disse mulige udvidelser, inkluderer muligheden for at brugere/medarbejderne kan bytte deres vagter indbyrdes. Dette kunne gøres ved, at en bruger fortæller systemet, at han ønsker at få byttet en vagt. Andre brugere kan så se dette, og de kan så sige at de vil hjælpe vedkommende og bytte med ham. Der kan så være mulighed for, at de enten direkte bytter eller, at den første bruger først skal godkende byttet. Det kunne også laves sådan, at hvis man fandt en vagt, man ville have, kunne man få systemet til at sende en forespørgelse til ejeren.

Vi skal også overveje, om det skal være muligt at afgive en vagt uden at få en vagt tilbage. En bruger slipper derved af med en vagt, mens en anden bruger får en. Til sidst skal vi overveje, hvor meget administratoren har at sige ved et vagtbytte. Skal han godkende alle sammen? Skal han informeres hver gang, der sker vagtbytte? Eller skal han blive overrasket over, hvem der egentlig møder op til de forskellige vagter?

Af lidt mere alternative løsninger kan man indføre et aktionssystem, så man kan byde på vagter der ønskes byttet. Det ville dog måske være en smule for useriøst, og hvad skulle man byde med?

Systemet skal, lige meget hvilken af ovenstående der bliver lavet, gøre brugeren opmærksom på, om den ansatte overhovedet må have denne vagt. Det kan være en idé at gøre det muligt for brugere at overtage vagter, de egentlig ikke burde kunne tage ved medarbejdermangel. Men som minimum skal administratoren underrettes om dette og måske endda godkende det, før det kan lade sig gøre.

Notifikation

Systemet skal kunne fortælle brugere og administratoren, når der sker noget specielt. Fx når der sker et vagtbytte, som nævnt ovenfor. Der er også her mange måder, hvormed dette kan foregå. Det kan skrives i en simpel logfil. Den vil det højst sandsynligt kun være administratoren, der har adgang til. Logfiler bruges som regel også kun til system specifikke begivenheder, såsom hvornår brugere logger ind, hvornår en bruger er oprettet, eller hvornår den starter og slutter på automatisk vagtplanlægning osv.

Ved andre begivenheder, som når et vagtbytte bliver godkendt eller aktiveres, skal de implicerede parter informeres. Det kan de blive på flere måder. Hvis systemet har et indbygget besked-system, kunne det gøres her. Brugere kunne også få en e-mail tilsendt med diverse informationer og måske et link til yderligere information. Til sidst kan brugere også få tilsendt en sms, som fortæller det mest nødvendige. Den sidste mulighed vil dog koste penge.

Automatisk vagtplanlægning

At få systemet til automatisk at generere en vagtplan, er en meget omfattende, men også en utrolig hjælpsom funktion, hvis den altså virker efter hensigten. Vores undersøgelse har vist, at en sådan funktion er efterspurgt⁷. Den vil kræve nogle inputs fra administratoren, fx hvilke vagter der skal udfyldes, og hvilke medarbejder der kan udfylde dem. Systemet skal så uddele vagterne til alle medarbejdere, så alle får det antal vagter de skal have. Vagter skal kun tildeles en medarbejder, der har de påkrævede evner til vagten. Desuden skal systemet tage højde for

⁷Se 1.2 behovsanalysen side 11

lovgivningen såsom arbejdstider og hviletider. En funktion som automatisk vagtplanlægning vil indeholde en omfattende algoritme, og det kan derfor tage en del tid at gennemføre.

Bruger info

Medarbejderen er en vigtig del af et vagtplanlægningssystem. Vi skal derfor have en database indeholdende en masse informationer om hver medarbejder. Hvor mange oplysninger kommer helt an på, hvilke funktioner vi ønsker at lave. Der er dog nogle meget basale informationer, som bør være der. Fx medarbejderens navn og adresse. Notifikation funktionen kræver bl.a. medarbejderens e-mail og mobiltelefonnummer. Automatisk vagtplanlægning kræver en del informationer om hver medarbejder - fx hvilke vagter de er i stand til at tage, og hvilke de ikke kan. De skal desuden være udformet på en sådan måde, at de er så nemme at tilgå som muligt. Ved automatiseret vagtplanlægning er det en kompliceret algoritme, der skal laves, og hvis man kan gøre lettere ved at designe databasen ordentligt, er det at foretrække.

Andre informationer man kunne have var medarbejdernes fødselsdage. Dette kan bruges til at specificere medarbejderens alder. Det kan også bruges til at gøre alle opmærksom på, at det snart er en medarbejders fødselsdag. Man kunne også samle data om, hvor mange timer hver medarbejder har, hvor mange timer de vil komme til at arbejde fremover, hvor meget de har fået i løn, og hvor meget de vil få i løn fremover.

Se vagtplan

Hver medarbejder skal have mulighed for at se sin egen vagtplan. Det skal også være muligt for administratoren at se alle medarbejders vagtplaner, ellers kan det være svært at lave en vagtplan. For at lave en god vagtbytte-funktion bør det også være muligt for medarbejderne at se hinandens vagtplaner. Så kan de nemlig finde lige nøjagtig den vagt, de ønsker at bytte med. Med dette i tankerne kunne det være en smart idé at lave en samlet vagtplan for alle medarbejdere, hvor man kan se hvilke medarbejdere, der har hvilke vagter og hvornår. Dertil kan man tilføje en personlig vagtplan, så hver medarbejder kan få en mere overskuelig liste over ens egne vagter.

Digital vagtplanstavle

Denne idé kom frem under interviewet med Martin B. Olsen. Det er en digital tavle, som er opsat på selve arbejdspladsen. På denne kan vagtplaner blive vist. På denne måde kan administratoren opdatere vagtplanen uden at skulle udprinte nye vagtplaner og hænge dem op. Det vil spare en masse papir og tid. Det vil dog være en udvidelse, der både vil være dyr og gå ud over vores evner, og det vil derfor være bedst ikke at gøre mere ved det foreløbig.

Timer og lønninger

For at gøre arbejdet endnu nemmere for administratoren, kunne man tilføje statistik over, hvor mange timer hver medarbejder har haft, og hvor meget i løn de skal have. Dette kræver indsigt i flere love og overenskomster. Systemet skal på dette punkt gøres endnu mere sikker, da der ikke må kunne ske fejl med beregningerne. I første omgang er det nok en dårlig idé, at få systemet til at holde styr på timer og lønninger. Men derimod vente til resten af systemet er på plads og fungerer.

Priser

Ved valget af løsningsmetoder til et sådan system, er det vigtigt at tage højde for brugerens krav til pris. Brugeren vil naturligvis altid have det så billigt som muligt i forhold til kvalitet, men en virksomhed kan måske have et ønske om en funktion, som de vil betale ekstra meget for. For

at holde omkostningerne nede for os, og dermed også brugeren, kan vi vælge billige eller gratis udviklingsteknologier. Fx vil open-source⁸ software opfylde disse krav.

Delkonklusion

Igennem dette afsnit, har vi blandt andet diskuteret, hvordan sikkerheden omkring systemet skal være. Sikkerheden er meget vigtigt, da data der bliver sendt, ikke må kunne opsnappes og bruges imod virksomheden, som bruger vores system. Servernes data må heller ikke gå tabt, så der skal tages backup af systemet hver dag, som et minimum, for at forhindre datatab, hvis en af serverne skulle gå ned.

Brugervenligheden og interfacet skal være nemt og simpelt, for at brugeren kan finde rundt i det, og ikke bliver forvirret.

Når man laver sådan et system, så er der altid mange forskellige mulige funktioner, som kan tilføjes vores system. En af de funktioner, som vi har diskuteret, er muligheden for at vores system selv genererer vagtplanen. For at systemet skal kunne gøre dette, så skal systemet have nogle input fra administratoren, samt oplysninger om hver enkelt medarbejder. Systemet skal bruge disse oplysninger for at kunne fordele vagterne på sådan en måde, så vagtplanen overholder overenskomsten samt lovgivningen for dette område.

Vurderingerne i dette afsnit, er en vigtig del af udviklingsarbejdet af systemet. Det hele kan være en del af det færdige system og derved øge funktionaliteten i systemet. Derfor skal der tages højde for, hvad vi vil have med i systemet, og hvad vi synes der er relevant for projektet.

1.3.2 Programmørens krav

Som programmører af et system har vi også krav, vi selv ønsker at opfylde. I dette afsnit vil vi komme ind på udviklingsteknologier og de forskellige funktioner, vi kunne implementere i vores system, og hvilke fordele og ulemper der ville være ved dem.

Klient/server/database

Da vi har begrænset tid til dette projekt, er det vigtigt, at vi benytter et programmeringssprog, vi nogenlunde har styr på i forvejen. Det er også vigtigt, at det valgte programmeringssprog kan opfylde vores krav til systemet. Det nytter ikke noget, at vi ikke benytter et sprog, som ikke kan det, der er behov for.

Brugeren afgør de fleste krav til klienterne, men som programmører har vi også nogle få krav til dem. De skal laves i programmeringssprog, der er nemme at vedligeholde og kan vise dynamisk indhold⁹ som bliver genereret af serveren. Yderligere er det vigtigt, at klienterne er kompatible med så mange tiltænkte operativsystemer (fx Windows, Linux) som muligt.

Serveren er et vigtigt led i vores system, da den formidler og behandler information mellem vores database og brugernes klienter. Det er vigtigt, at serveren kan håndtere det server-system, som vi laver. Den skal også være hurtig og skal kunne klare flere forbindelser mellem servere og klienter.

Databasen skal være midtpunktet i hele vores system. Det er her alle oplysningerne om bl.a. de ansatte og deres vagter ligger gemt. Pga. af økonomiske årsager skal databasesystemet helst være billigt eller gratis at benytte. Endnu et krav til databasen vil være, at den skal være hurtig og kunne håndtere store mængder data.

⁸Software hvor alle har adgang til kildekoden.

⁹Dynamisk indhold betyder, at indholdet kan ændre sig.

Moduler

Hvis vores system til vagtplanlægning skal kunne bruges i andre sammenhænge end hos Netto, er det vigtigt, at der er rig mulighed for at tilpasse systemet. Ikke alle virksomheder har de samme behov, og har derfor ikke altid brug for de samme funktioner.

Store overordnede funktionaliteter kan inddeles i såkaldte moduler¹⁰. Systemets administrator skal have muligheden for at slå disse moduler til eller fra, måske endda installere nye eller fjerne gamle. Et eksempel på et modul til dette system kunne være inddeling i grupper eller lokaler, hvilket der er behov for på fx en skole, men ikke i en butik.

For at gøre systemet fleksibelt ville det være en ide, at gøre det muligt for kunden selv at programmere et modul. Det kunne også være en ide at indføre en flersprogsfunktion, så systemet kan oversættes til flere forskellige fremmedsprog, såsom engelsk, fransk eller tysk. I systemet skal der dog stadig være inkluderet en række standard moduler med de funktioner, der ofte anvendes.

Foruden modulerne skal der også være en fast kerne i systemet, der udover håndteringen af modulerne, står for den generelle drift. Med den generelle drift menes funktioner som modulstyring, database-kommunikation, bruger-oprettelse, administration, sikkerhed etc.

Ved at give vores system denne fleksibilitet, gør vi det dermed mere fremtidssikret. Når brugeren har mulighed for at tilpasse systemet efter behov, vil det give mulighed for at udskifte forældede funktioner med nye mere tidssvarende. På den måde vil det også give systemet mulighed for, at leve videre uden vores direkte indblanding.

Delkonklusion

Hvad angår udviklingsteknologier på klient og server, så er det vigtigt at vælge programmeringssprog, der kan arbejde sammen og er nemme at vedligeholde. At vedligeholdelse af koden er nemt for programmøren er vigtig, da det ellers kunne kræve unødigt meget tid at lave en mindre rettelser. Yderligere skal der kunne kommunikeres hurtigt og med store mængder data til databasen.

For at fremtidssikre vores system gælder det om at udvikle et fleksibelt system. Ved at gøre systemet fleksibelt, er det bl.a. nemmere at implementere systemet i andre virksomheder. Vores system kunne virke som samlingspunkt for en række underliggende moduler. Vi kunne derfor udvikle en egentlig programkerne samt de underliggende moduler. Udvikling og videreudvikling ville derved blive så simpel som mulig for begge parter.

¹⁰Moduler vil sige dele af et system der kan tilføjes eller fjernes via en simpel brugergrænseflade

1.4 Konklusion på analysedelen

Til dette projekt har vi fravalgt visse funktioner til systemet, og vil koncentrere os om de funktioner, vi mener er vigtige for vores produkt.

Målgruppe

Til dette projekt vil vi koncentrere os om Netto som målgruppe. Vi vil forsøge at begrænse funktionsudvalget og skræddersy det til Netto's behov. I første omgang vil det nok ikke blive et færdigt funktionsdygtigt system, men vi vil bestræbe os efter at færdiggøre de enkelte dele, og ikke nødvendigvis samle det til et færdigt produkt klar til Netto. Der skal dog bemærkes, at selvom vi laver produktet til Netto, så kan muligheden for at det nemt kan implementeres i andre virksomheder stadig eksistere.

I systemet vil vi ikke gemme oplysninger om de ansattes løn i vores database, da vi vil koncentrere os om at ligge vagtplanen. Det kunne være smart at tage højde for de ansattes løn i vagtplanlægningen, men det ville også gøre vores vagtplanlægningsalgoritme meget mere avanceret og besværlig at lave.

Brugerens krav

Når vi skal lave systemet, har vi flere forskellige muligheder og ideer, som kan indføres i systemet. Vi bliver dog nød til at begrænse os, i hvilke funktioner og hvad systemet kan, da det ellers vil blive for tidskrævende at lave.

Selve systemet vil vi lave web-baseret, da det på den måde vil være nemt tilgængeligt for den enkelte bruger at logge ind i systemet og se fx. deres vagtplan. De enkelte brugere slipper også for, at skulle installere et separat program, for at kunne se deres vagtplan. De skal kun bruge deres web-browser og internetet. Kryptering mellem klient og server er vigtig, men det er ikke noget vi vil beskæftige os med i dette projekt.

Vi vil forsøge at lave den automatisk vagtplanlægnings algoritme. Det er en utrolig smart funktion, men også en svær opgave. Det er vigtigt at medarbejderne og administratoren kan bytte vagterne. Medarbejderne skal kunne bytte med hinanden og administratoren skal kunne overskrive alt og fuldstændig bestemme hvordan vagterne skal være placeret.

Vi skal have et log-system, så vi altid kan se helt præcist hvad systemet fortager sig. Vi skal også have lavet mulighed for notifikation via email og sms, når der sker vigtige begivenheder såsom vagtbytte eller at en helt ny vagtplan er blevet færdig.

Sikkerhed er en vigtig faktor i vores system, da det data, der bliver sendt frem og tilbage, skal være sikret imod uvedkommende, der vil opsnappe og bruge dataene. Vi har valgt, at vi ikke tager sikkerhed med som en stor del af systemet, da dette ikke er centralt for vores projekt. Vi vil koncentrere os om andre dele af systemet.

Interfacet og brugervenligheden er en vigtig del af vores projekt, men det kommer dog ikke i første række. Grunden til dette er, at det er vigtigere, at systemet i første omgang bliver funktionelt, end at brugerfladen bliver flot og vel designet.

Programmørens krav

Til systemet kan der være forskellige former for klienter. I dette projekt vil vi fokusere på klienter der understøtter wap, web, sms og email, dvs. mobiltelefoner og computere.

Når systemet skal være webbaseret med dynamisk indhold, vil det være nødvendigt at vælge et server script sprog beregnet til formålet. Af eksempler kan nævnes CGI, ASP og PHP.

Vores valg falder på PHP, da det har alle de funktioner vi skal bruge, og tilmed er gratis at benytte og hoste.

Hvad angår database, så har vi kun erfaringer med MySQL og Microsoft Access. Et naturligt valg for os ville derfor være MySQL, da dette databasesystem er gratis, og er bedre optimeret til større systemer. Yderligere har vi også den største erfaring med MySQL.

Ideen med moduler vil vi i første omgang fravælge, da det ville kræve for meget fokus på andet end kernen i vores projekt - et system til vagtplanlægning.

Kapitel 2

Problemformulering

Som udgangspunkt til vores vagtplanlægningsystem vil vi bruge Netto. Efter interviewet med butikschefen for Netto, Danmarksgade, fandt vi ud af at det der var behov for bl.a. en automatiseret vagtplanlægning. Et sådan system ville lette butikschefens arbejde. Udover den automatiske vagtplanlægning vil vi prøve at konstruere funktioner til systemet, der ikke kun letter de administrerendes arbejde, men også de ansattes.

Udfra de, i behovsanalysen¹, nævnte behov, skal systemet derved indeholde følgende funktioner:

- Vagtbytte
- Automatisk vagtplanlægning
- Administration
- Notifikation

Da systemet, som nævnt, også skal være til fordel for de ansatte, vil vi gøre det muligt at bruge vores system overalt. For at gøre dette skal systemet understøtte en række klienter. Server-softwaren, der skal laves i PHP, og benytte MySQL som database, skal kunne kommunikere med følgende klienter:

- Web - For brugere
- Web - For administratorer
- Wap/Mobiltelefon - For brugere
- Sms/Mobiltelefon - Kun notifikation
- Email - Kun notifikation

Hvordan kan vi udvikle et system til vagtplanlægning for Netto, der specialiserer sig i automatiseret vagtplanlægning?

For at løse dette problem vil vi bl.a. benytte metoden "Divide and conquer". Dvs. at vi vil dele problemet op i mindre delproblemer, der er lettere at tage fat på. Når løsningerne på delproblemerne til sidst samles, har vi løsningen til det overordnede problem.

¹Se 1.2 behovsanalysen side 11

Kapitel 3

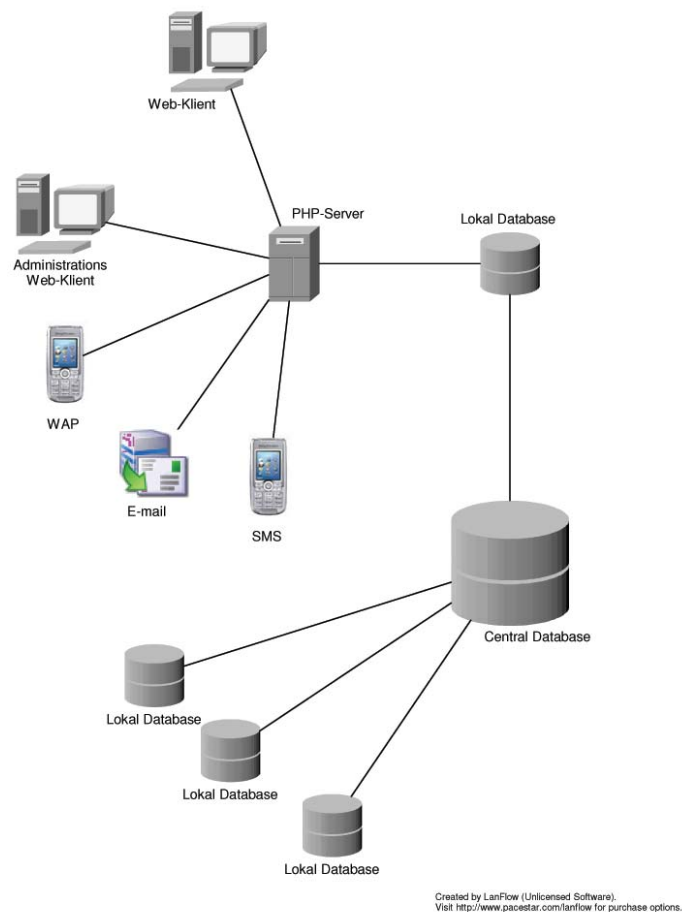
Løsningsforslag

3.1 Arkitektur

3.1.1 Klient-Server model

Inden vi kan udvikle selve systemet, skal vi have et overblik over hvordan selve opbygningen af det kan struktureres.

Serveren i den enkelte butik skal, som tidligere beskrevet, kunne kommunikere med flere forskellige klienter, hvilke kan benyttes af såvel ansatte som administratorer. Klienterne kan inddeles i 2 kategorier: Notifikations- og kommunikations-klienter. Notifikations-klienterne, sms og email, skal bruges til at give korte informerende notifikationer til brugerne. Derimod skal kommunikations-klienterne, wap og web, kunne sende feedback til serveren, og derved evt. ændre data i databasen. Klienterne kommunikerer i princippet kun direkte med butikkens lokale server, og denne står derefter for kommunikationen med databasen. Den lokale database indeholder de nødvendige data for den pågældende butik. Ovenstående struktur vil kunne udbygges til at kommunikere med andre butikker i samme kæde, ved at forbinde de enkelte butikker til en fælles database. Dette ville være en fordel for butikker som fx Netto, der til tider bliver nødsaget til at låne arbejdskraft fra andre butikker. At udbygge systemet med en central database, der forbinder de lokale databaser, vil gøre det muligt for systemet at sørge for bl.a. denne funktion.



Figur 3.1: Diagram over systemets opbygning

3.1.2 Databasen

I dette afsnit vil vi gennemgå hvilke oplysninger der skal gemmes i databasen, om de forskellige medarbejdere med mere. Følgende oplysninger er centrale for funktionaliteten af vores vagtplanlægningssystem, da de danner grundlag for bl.a. den automatiske vagtplanlægning og notifikationsdelen.

Medarbejder

I vores system skal der være en unik identifikation til hver bruger. Det kunne være CPR-nummeret eller et valgfrit brugernavn - brugerens fulde navn er ikke unikt, og kan derfor ikke bruges som unik identifikation. Men vi bliver nød til at have begge oplysninger på hver bruger, da det er smartest at bruge navne når man skal kommunikere med personer, mens det er bedst at bruge CPR-nummer internt i systemet.

Alderen er endnu en vigtig oplysning at have liggende. Da der er visse opgaver hvor man skal være fyldt 18 år, bl.a. hvis man skal sidde ved kassen. Dette bliver imidlertid intet problem, da brugerens alder kan beregnes ud fra CPR-nummeret.

Brugerens stilling i virksomheden er bestemmende for, hvilke opgaver brugeren kan løse, og dermed en vigtig brik i vores system til vagtplanlægning. Vi vil dog gemme oplysninger om hvilke færdigheder hver medarbejder har i stedet for en specifik stillingsbetegnelse. Det giver mulighed for, at medarbejdere der officielt har samme stilling kan have forskellige færdigheder.

Vi skal også vide om hver medarbejder er nøgleansvarlig eller ej, da der altid skal være en nøgleansvarlig tilstede. Dette bliver gemt som en sekundær færdighed.

Vi skal vide hvornår folk vil, kan og ikke kan arbejde. Dette er nødvendig, for at vores system kan lave en prioritering af hvem, der skal arbejde hvornår. Systemet opnår større tilfredshed hos brugerne, ved at den sætter folk i arbejde på tidspunkter hvor de gerne vil frem for tider, hvor de bare kan arbejde.

For at kunne give besked til brugerne når deres vagtplan er blevet ændret, er vi nød til at have deres email adresse eller deres mobil-nummer. Samtidig kan vi gemme oplysninger om hvilke begivenheder hver bruger ønsker at få besked om via email eller sms.

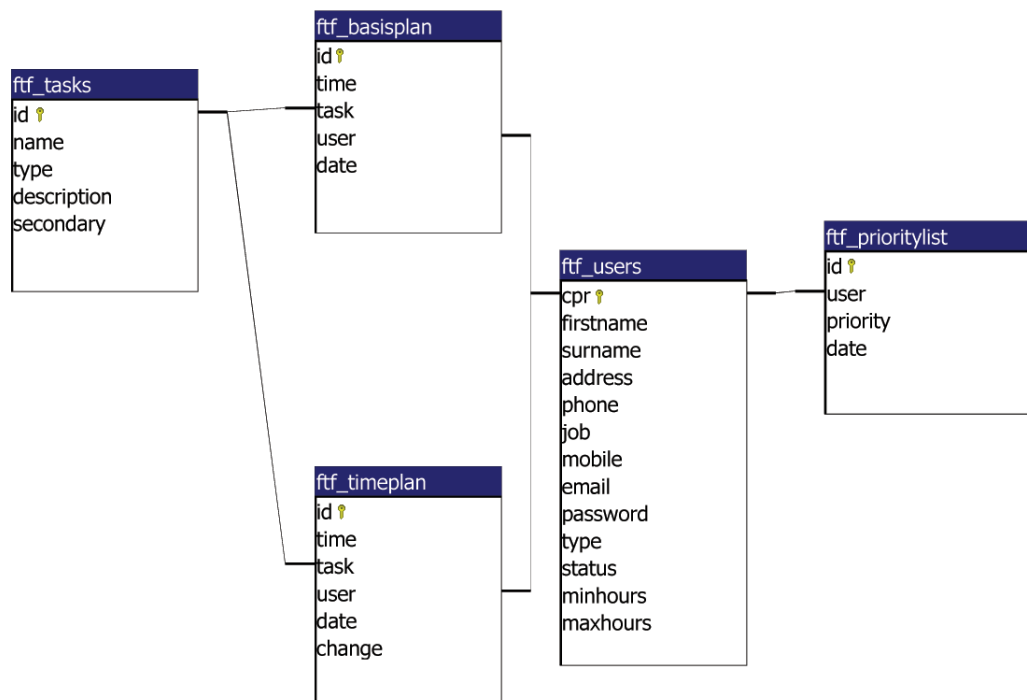
Vagtplan

Der skal være informationer om alle de vagter der skal udfyldes, hvilke kvalifikationer der kræves af medarbejderen, og hvem der i sidste ende har fået tildelt den. Der skal også stå på hvilke tidspunkter vagten er. Til sidst skal der stå om vagten er ønsket byttet.

Da nogle medarbejder kan være fastansatte og derfor skal have faste arbejdstider, ville det være smart at gemme en basisplan med deres arbejdstider. Dette kunne vores automatiske vagtplanlægningsalgoritme gå ud fra.

Database struktur

Selve struktureringen af vores MySQL-database kan ses på figur 3.2. Vi har inddelt databasen i fem forskellige tabeller med hver deres formål. Selvom `ftf_basisplan` her er inkluderet, har vi dog udeladt at tage højde for denne i selve systemet.



Figur 3.2: Diagram over strukturen af MySQL-databasen

På figur 3.2 er der indsat nøgler ud for en kolonne for hver tabel. Nøglen symboliserer, at der er en primary key eller en primær nøgle. Dvs. at disse kolonner indeholder unikke værdier for hver række, i modsætning til de andre kolonner i tabellen, hvor der muligvis kan være værdisammenfald. Hvis vi tager `ftf_users` som et eksempel, har en person et bestemt cpr-nummer, som er unikt. Det samme gælder for de andre tabeller, men her er der ikke nødvendigvis nogle af kolonnerne der indeholder unikke værdier, og derfor er kolonnen "id" indført. Værdien i denne kolonne genereres automatisk, til at være et unikt nummer hver gang en række tilføjes tabellen. Ved at hver tabel har en primær nøgle, kan der altid henvises til én bestemt række i en given tabel.

`ftf_tasks` er tabellen over de opgaver, som den enkelte medarbejder kan udføre. `name` er for at give opgaven et sigende navn, mens `type` bruges til at definere hvilke evner opgaven kræver. Begrebet `type` bruges således, at en bruger kan have evner til at løse flere opgaver listet i `ftf_tasks`. På figur 3.3 er `type` sat således at hver opgave kun kræver en evne, men en opgave kunne sagtens kræve flere evner. En uddybende forklaring på `type` findes i afsnittet om opgaveadministration¹. `description` er en beskrivelse af de forskellige opgaver, vi bruger dem dog ikke, men de kunne bruges. Til sidst er der `secondary`, der definerer om en opgave er sekundær. Hvis den er sat til 0, er der tale om en primær opgave, mens der er tale om en sekundær opgave,

¹Afsnit 3.2.3 side 44

hvis den er sat til 1. En ansat kan have flere opgaver på samme tid, men kun en af dem kan være en primær opgave.

id	name	type	secondary	description
8	Nøgleansvarlig	32	1	Åbne/Lukke
9	Flasker	1	0	Ordne flasker

Figur 3.3: Eksempel på to rækker i `ftf_tasks` tabellen

Meningen med `ftf_basisplan` var, at denne skulle indeholde de fastansattes vagtplan. De ikke fuldtids-ansatte medarbejdere kunne derefter indsættes efter basisplanen, og udfylde hullerne i denne. Vi bruger dog ikke denne tabel i vores produkt, da dette vil kræve noget mere tid og programmering, og da tiden til dette projekt er begrænset, har vi valgt at undlade denne.

`ftf_timeplan` bruges til at gemme de lagte vagtplaner. I denne tabel er alle de vagter som er blevet lagt. Her ligger vagterne for hver enkelt medarbejder, hvor man har mulighed for at se hvem der har vagten, hvad vagten går ud på, hvornår vagterne finder sted og om vagten er sat til byt eller ej. Hver række repræsenterer en opgave et givent tidspunkt på en bestemt dato, og fortæller hvem der har vagten, og hvorvidt denne bruger har sat vagten til byt eller ej. "task" er et id fra `ftf_tasks`, og "user" er et cpr-nummer fra `ftf_users` - bl.a. her ses vigtigheden af at have primære nøgler i hver tabel.

Hver medarbejder har visse informationer, som der skal bruges når vagtplanen skal lægges. Alle disse informationer ligger i `ftf_users`. De mest basale informationer som cpr-nummer, fornavn, efternavn, tlf./mobil osv. bliver lagt ind under hver bruger. Udover disse informationer, så skal systemet også have at vide, hvad medarbejderen er blevet ansat som. Dette defineres i `job`, hvor der fx kan stå 1. mand, elev osv. Udover hvad de er blevet ansat som, skal der også vides, hvilke opgaver den pågældende medarbejder kan løse. Dette gøres under `type`, der angiver hvilke evner den ansatte har. Systemet gemmer alle medarbejderne, men der er kun et vist antal som er ansat. Derfor skal dette også defineres, og dette gøres ved hjælp af `status`.

Der er forskellige former for ansættelse, og hvor meget der må arbejdes under disse ansættelsesforhold. Derfor skal systemet også vide hvad medarbejderen skal have af minimum timer og maksimum timer.

Den sidste tabel er `ftf_prioritylist`, hvor der defineres hvilke dage medarbejderen gerne vil arbejde, og hvor vedkommende ikke har mulighed for det. "user" er den pågældende brugers cpr-nummer, og "priority" er om personen gerne vil, ikke kan eller er ligeglad. Den sidste, date, kan indeholde en dato eller en fx en ugedag. Dette beskrives i afsnittet om prioritetsobjektet.

Ved hjælp af disse tabeller, har vi formet vores database. Alle de informationer, som ligger i vores database, har en relevans, når vi skal programmere og lægge en vagtplan.

3.1.3 MySQL

SQL er et populært sprog, der bruges til hente, gemme og ændre data fra relationelle database-systemer. En relationel database er en database, der er ordnet efter relationer mellem dataene. I SQL består disse relationer i, at alt data bliver gemt i tabeller med rækker og kolonner.

Når man opretter en ny tabel defineres, hvad de forskellige kolonner skal kaldes og indeholde. Hver gang der tilføjes data til en tabel bliver en ny række tilføjet. Den nye række indeholder data, der passer til de tilhørende kolonner

En relationel database er smart, da programmøren ikke behøver at holde styr på, hvordan data er gemt, men blot hvordan deres indbyrdes forhold er. Dvs. hvilke tabeller og hvilke kolonner der skal ledes i.

MySQL er et open-source databasesystem, som bruger SQL²[2]. Kommunikationen til databasen forgår, som før nævnt, via SQL-forespørgsler. Disse forespørgsler bruges hver gang der skal hentes, gemmes eller ændres data på serveren. SQL er et sprog, der er udviklet af IBM i 1970'erne, og er i dag en ISO³-standard.

Der er tre forskellige typer af forespørgsler der bruges i SQL: DML⁴, DDL⁵ og DCL⁶[3]. Vi bruger dog kun de to førstnævnte, hvilke er beskrevet nedenfor med eksempler på anvendelse.

DDL

DDL er de forespørgelser der definerer databaser, tabeller og indekserer tabeller. Her er den eneste DDL-forespørgsel vi bruger i vores system, og en beskrivelse af hvordan den virker:

CREATE TABLE

Når der skal oprettes en tabel, skal der bruges `CREATE TABLE`. Denne består af tre dele:

Først skal der angives hvad tabellen skal hedde.

Eksempel: `CREATE TABLE ftf_tasks`

Herefter definerer man kolonnerne enkeltvis, skrevet op med navnet først, så data-typen og herefter forskellige andre krav.

Eksempel: `('id' INT NOT NULL AUTO_INCREMENT , ... 'secondary' TINYINT(1) NOT NULL ,`

Til sidst bliver der så defineret specielle kolonner - så som primær-kolonne.

Eksempel: `PRIMARY KEY ('id'))`

Nedenstående er et eksempel på en færdig `CREATE TABLE`-forespørgsel:

```
CREATE TABLE ftf_tasks (
  'id' INT NOT NULL AUTO_INCREMENT ,
  'name' VARCHAR( 255 ) NOT NULL ,
  'type' INT NOT NULL ,
  'description' VARCHAR( 255 ) NOT NULL ,
  'secondary' TINYINT( 1 ) NOT NULL ,
  PRIMARY KEY ( 'id' )
)
```

Denne forespørgsel opretter tabellen `ftf_tasks` med kolonnerne: `id`, `name`, `type`, `description` og `secondary`. `id` er primær kolonne og er også automatisk stigende. Nogle af kolonnerne er

²Structured Query Language

³International Standards Organization

⁴Data Manipulation Language

⁵Data Definition Language

⁶Data Control Language

heltal (`INT`), mens andre er tekst med variabel størrelse (`VARCHAR`). Den sidste er et lille heltal (`TINYINT`), som bliver brugt til at gemme den boolske værdi: 0 eller 1.

DML

DML er forespørgsler, der henter, ændrer eller indsætter data i databasen. Her er de DML-forespørgsler vi bruger og en beskrivelse af, hvordan de virker:

SELECT

For at hente information ud af SQL-databasen, bruges kommandoen `SELECT`. En `SELECT`-forespørgsel består af op til fem dele: Den første del er kolonne-delen, hvor der angives hvilke kolonner, fra en tabel, der skal hentes ud af databasen.

Eksempel: `SELECT name, cpr`

Herefter kommer tabel-delen, hvor der angives, fra hvilken tabel, de angivne kolonner skal hentes fra.

Eksempel: `FROM ftf_users`

De følgende dele er ikke påkrævet i `SELECT`-forespørgslen, men kan bruges til at begrænse den information, der hentes ud. `WHERE` er en betingelse, der skal opfyldes, før en række kan blive returneret.

Eksempel: `WHERE status=2`

Hvis resultaterne ønskes returneret i en bestemt rækkefølge, bruges `ORDER BY`.

Eksempel: `ORDER BY firstname ASC`

Der er to muligheder for sortering, enten i stigende (`ASC`) eller aftagende (`DESC`) rækkefølge. Som den sidste mulighed, kan antallet af rækker, der returneres, begrænses. Det gøres med `LIMIT`.

Eksempel: `LIMIT 3`

Et eksempel på en færdig `SELECT`-forespørgsel kunne se således ud:

```
SELECT name,cpr,type FROM ftf_users WHERE status=2 ORDER BY firstname ASC LIMIT 3
```

Denne forespørgelse henter navn, cpr-nummer og type fra tabellen ved navn `ftf_users`, men den henter kun de tre første, der er ansat (`status=2`).

INSERT

For at indsætte data i databasen, bruges `INSERT`. Denne forespørgsel består af op til tre dele: I `INSERT` kommer tabel-delen først, og den vælger tabellen man vil skrive til.

Eksempel: `INSERT INTO ftf_tasks`

Efter man har valgt en tabel, kan man vælge hvilke kolonner man vil indsætte i, men det er ikke påkrævet.

Eksempel: `(name, type)`

Til sidst skrives der hvilke værdier der ønskes indsat i kolonnerne. Dette kan gøres på to måder.

Hvis der er angivet kolonner, skal der bruges `VALUES`.

Eksempel: `VALUES ('Hyldefylder', 8)`

Hvis kolonnerne ikke er angivet, skal disse angives enkeltvis med deres tilhørende værdier.

Eksempel: `SET name='Hyldefylder', type=8`

Her er eksempler på færdige `INSERT`-forespørgsler:

```
INSERT INTO ftf_tasks (name,type) VALUES ('Hyldefylder', 8)
INSERT INTO ftf_tasks SET name='Hyldefylder', type=8
```

Begge forespørgsler gør det samme. De indsætter en opgave i tabellen `ftf_tasks` med navnet “Hyldefylder” og med typen 8.

REPLACE

`REPLACE` fungerer næsten på samme måde som `INSERT`. Der er dog den forskel, at hvis der eksisterer en række i databasen med samme primary-key, så overskrives denne række.

```
REPLACE INTO ftf_tasks (name,type) VALUES('Hyldefylder', 16)
```

Hvis `name` var en primær-nøgle, så ville opgaven “Hyldefylder” få ændret typen til 16., ellers ville der blive indsat en ny.

UPDATE

Når der er en eller flere rækker, der skal rettes skal `UPDATE` bruges. Denne forespørgsel består af op til fem dele:

Der skal som det første altid vælges tabel.

Eksempel: `UPDATE ftf_tasks`

Efter, at det er gjort, angives kolonnerne enkeltvis med den værdi, den skal have.

Eksempel: `SET type=4, description='Noget med hylde'`

Efter disse påkrævede dele, begrænses de påvirkede rækker på samme måde som i `SELECT` - med `WHERE` og `LIMIT`.

En færdig `UPDATE`-forespørgsel kunne se således ud:

```
UPDATE ftf_tasks SET type=4, description='Noget med hylde' WHERE name='Hyldefylder'
LIMIT 1
```

Denne forespørgsel opdaterer rækken, hvor `name` er “Hyldefylder”, således typen og beskrivelsen ændres. Der er `LIMIT` på for, at sikre at der ikke ændres mere end én række.

DELETE

Hvis der skal slettes rækker i en tabel bruges `DELETE`. Det er en meget simpel forespørgsel, der består af op til 4 dele. Den består af de samme dele som de andre forespørgsler:

Først en tabeldel. Eksempel: `DELETE FROM ftf_tasks`

Herefter begrænses de berørte rækker med `WHERE` og `LIMIT`.

En færdig `DELETE`-forespørgsel kunne se således ud:

```
DELETE FROM ftf_tasks WHERE name='Hyldefylder' LIMIT 1
```

Denne forespørgsel sletter rækken i tabellen `ftf_tasks`, hvor navnet er “Hyldefylder”. Den sletter kun den første række, pga. `LIMIT 1`, da det er dumt at slette flere rækker af gangen. Hvis der ikke er angivet en tilstrækkelig begrænsning, kan det ske, at alle rækker i tabellen bliver slettet.

MySQL i PHP

I PHP er der nogle vigtige funktioner, der er skrædersyet til at kommunikere med MySQL[4]:

mysql_connect()

For at få forbindelse til MySQL bruges `mysql_connect()`. Denne funktion har tre parametre: server, brugernavn og kodeord. Her er et eksempel på en PHP-kode, der skaber forbindelse til en lokal MySQL-server:

```
$link = mysql_connect('localhost', 'BrugerNavn', 'Kodeord');  
if (!$link) {  
    die('Kunne ikke forbinde: ' . mysql_error());  
}
```

Som sagt forbinder denne kode til en lokal MySQL-server, og hvis den ikke får forbindelse så stoppes scriptet, og der bliver meldt fejl.

mysql_select_db()

Når der er forbundet til MySQL-serveren, skal der vælges en database. Dette gøres med `mysql_select_db()`:

```
mysql_select_db('database', $link);
```

Her vælges databasen "database". Parameter nummer to (`$link`) er den forbindelse der er oprettet i `mysql_connect()`, men denne parameter er dog valgfri hvis kun en forbindelse bruges.

mysql_query()

For et sende DDL eller DML til MySQL bruges `mysql_query()`:

```
$result = mysql_query('SELECT * FROM ftf_users', $link);
```

Her bliver der sendt en SELECT-query til MySQL. Parameter nummer to (`$link`) er den forbindelse der er oprettet i `mysql_connect()`, og denne er igen valgfri hvis kun en forbindelse bruges. Outputtet bliver gemt i variabelen `$result`.

mysql_close()

Når der ikke skal arbejdes mere med MySQL, skal forbindelsen lukkes med `mysql_close()`.

```
mysql_close($link);
```

Her lukkes forbindelsen, der blev oprettet med `mysql_connect()`.

3.1.4 Klient

HTML er en forkortelse af HyperText Markup Language. HTML er et markup sprog til websider, hvor der kan bestemmes struktur af en side, laves hyperlinks, formaterer teksten osv. Et HTML dokument bliver gemt på en server, hvor man skal skrive en adresse eller URL⁷ for at åbne dokumentet. Med hyperlinks kan der henvises til andre html dokumenter som kan ligge på samme server eller et helt andet sted i verden[5].

Når man skriver HTML er det ved brug af forskellige "tags". Disse tags, også kaldet elementer, bruges til at strukturere indholdet af et HTML-dokument. Selvom det kunne ligne, så er HTML ikke et scriptsprog. Det er ikke muligt at eksekvere noget program med rent HTML.

Der er dog mulighed for at indføre dette, og derved gøre sin webside mere dynamisk ved at implementere kode fra et decideret scriptsprog. Her kan der fx bruges JavaScript, der kan implementeres på ens webside. JavaScript er et klientscript og styres af brugerens web-browser. Ved hjælp af JavaScript, kan der indarbejdes noget eksekverbart, som ellers ikke ville være mulig med almindelig HTML. Et andet scriptsprog som man kan bruges er PHP.

WML ligner meget HTML, men er beregnet til WAP-sider. WML har mere strenge standarder og understøtter bl.a. ikke klientscripts. Vi har valgt at bruge WML 1.1 standarden[5].

Email og sms bliver kun brugt til notifikation, hvilket vil sige, at der kun er tale om envejskommunikation. Dette medfører at vi kan sende begge dele via indbyggede funktioner på serveren, da vi ikke forventer svar. Begge typer beskeder bliver sendt som emails. At sms-beskederne kan sendes som email, skyldes, at vi benytter en sms-gateway.

3.1.5 Server

PHP er et serverscript og er en forkortelse for "Hypertext Preprocessor". At PHP er et serverscript, har den betydning, at serveren, som PHP dokumentet ligger på, først læser hele dokumentet igennem og søger efter PHP kode. Hvis serveren finder PHP kode, bliver denne kode eksekveret. Den eksekverede PHP-kode returneres som et HTML-dokument, og denne kan klienten læse. Klienten vil derfor aldrig se PHP-koden, men kun den færdige HTML-kode[4].

Når man koder i PHP, indgår HTML også, da PHP tager udgangspunkt i HTML. Derfor er det muligt, at blande HTML kode og PHP kode sammen. På den måde er det muligt at lave en webside, som er dynamisk. Fx kan man skrive speciel data ud på baggrund af brugerens indtastninger.

Ved hjælp af PHP, er det muligt at få web-serveren til at udføre diverse kommandoer. Fx er det muligt at tilgå forskellige databaser, som fx MySQL, der er beskrevet i afsnit 3.1.3. Dvs., at det er muligt at tilføje, fjerne, hente eller redigere oplysninger og informationer i sin database ved hjælp af PHP. Samtidig kan disse informationer udskrives som HTML, og derved fås et dynamisk indhold, der ændres afhængig af indholdet i databasen.

I HTML findes der såkaldte forms, som giver brugeren mulighed for at indtaste data og derefter sende dem til serveren. Med PHP får serveren mulighed for at håndtere disse data og fx indsætte dem i en database. Brugeren kan sende disse data på to måder, `get` og `post`. Forskellen på disse to er at ved `get` bliver data sendt i URL'en, mens de med `post` bliver sendt i http-headeren. Der er flere måder at anvende disse data, bl.a.:

```
$get_name = $_GET['name'];  
$post_name = $_POST['name'];
```

⁷Uniform Resource Locator

I PHP er der mulighed for at bruge classes. Med disse kan der oprettes et objekt som indeholder nogle variabler og funktioner. Det kan bruges til fx at indsætte en masse data i et objekt og få den til at bruge disse på en bestemt måde. En class kan se ud på følgende måde:

```
class MyClass {
    var $myVar;

    function MyClass() {
        $this->myVar = $this->myFunction('world!');
    }

    function myFunction($argument) {
        return 'Hello ' . $argument;
    }
}
```

Dette laver en class med navnet MyClass. I den er der en variabel ved navn \$myVar og en funktion kaldet myFunction, samt en såkaldt constructor som hedder det samme som selve classen. En constructor bliver kaldt hver gang classen oprettes, og dette gøres på følgende måde:

```
$myObject = new MyClass;
echo $myObject->myFunction('Europe');
```

For at bruge funktioner og variabler som ligger inde i classen, anvendes operatoren ->. Inde i selve classen skal der anvendes \$this-> for at referere til en intern funktion eller variabel.

Vi bruger PHP4, og derfor har classes ikke helt de samme muligheder som i PHP5. Bl.a. bliver der tilføjet muligheden for at bruge public og private.

I PHP findes flere forskellige funktioner. Fx include(). Denne funktion gør det muligt at inkludere en ekstern fil. På den måde kan der nemt tilføjes samme menu på alle ens sider uden at skulle lave en masse kopier af den samme kode.[4]

3.2 Dokumentation

3.2.1 Generelt

Index.php

Denne fil er kernen af systemet. Alle andre sider skal “køres” gennem denne. Det er derfor ikke muligt for brugeren at gå uden om denne fil. Mere om det i afsnit 3.2.2, der omhandler sikkerhed.

Først i filen hentes variabler fra URL'en samt oprettelse af nogle få grundlæggende variabler. Dette er gjort for at gøre det nemmere at finde de rigtige værdier i de andre filer. Nu inkluderes filen `common.inc.php`. Denne indeholder en masse globale funktioner som bliver dokumenteret senere. Vi bruger funktionen `include_once` for at være sikker på at den kun bliver inkluderet én gang. Dette er hovedsageligt vigtigt i opbygnings- og test-fasen hvor `common.inc.php` kan være inkluderet på andre sider.

Efter at `common.inc.php` er inkluderet inkluderes `page_header.inc.php` og `page_footer.php` som indeholder funktioner til at udskrive toppen og bunden af HTML koden. Dette er gjort således for at vi ikke behøver at skrive den samme mængde HTML kode på alle sider og at vi selv kan vælge hvornår dette skal skrives ud. Der er ingen grund til at udskrive HTML kode hvis vi alligevel sender brugeren videre til en anden side.

Ofte-brugte funktioner

`common.inc.php` indeholder ofte-brugte funktioner, sådan at disse funktioner kun behøver at være defineret i én fil. Derefter kan denne fil inkluderes de steder, hvor der er brug for funktionerne. Først i filen `common.inc.php` defineres nogle globale variabler såsom `$dbLink`, der er en resource, der indeholder forbindelsen til databasen. Herefter oprettes alle funktionerne. Her kommer en liste over de vigtige funktioner i `common.inc.php` og en forklaring til dem.

boolConnectDB()

Denne funktion laver en forbindelse til databasen og gemmer den i `$dbLink` som resten af systemet bruger når data skal hentes eller sendes til databasen. Hvis der sker en fejl undervejs vil funktionen returnere `false`, ellers `true`.

boolCloseDB()

Funktionen lukker forbindelsen til databasen og returnerer `true`, hvis det lykkedes. Ellers returneres `false`. Dette er mest fordi det er god praksis at lukke forbindelsen igen efter, at den er blevet åbnet.

boolTaskCheck()

Denne funktion laver et bitvis tjek på, om en bruger kan udføre en given opgave.

```
$intAnd = $intUserType & $intTaskType;
if ($intAnd == $intTaskType){
    return true;
} else {
    return false;
}
```

Først finder den alle de evner, som brugeren har tilfælles med de, af opgaven, krævede evner. Derefter tjekkes, om de fælles evner er de samme som de evner opgaven kræver. Hvis de ikke er det, har brugeren ikke alle de evner, som opgaven kræver og funktionen returnerer `false`, ellers returnerer den `true`.

strChange()

Når en bruger ønsker at bytte eller overtage en vagt køres denne funktion. Den tjekker først, om vagten allerede er sat til byt. Hvis den er, så skal vagten overtages, ellers skal den sættes til byt. Det hele foregår ved en simpel boolsk værdi i databasen som enten kan være `true` eller `false`.

3.2.2 Sikkerhed

Herunder vil vi gennemgå de sikkerhedstjek vi har implementeret i vores vagtplanlægningssystem. Vi har afgrænset os fra de større sikkerhedstjek som fx en krypteret linje mellem serveren og klienten. De sikkerhedstjek vi har i vores system er listet herunder:

- Login
- Login-niveau
- Session
- `define('FTF')`
- `md5()`
- cpr-tjek

Login

Medarbejderne skal have adgang til vagtplanlægningssystemet, så de kan se og ændre i deres egen vagtplan. Her er det vigtigt, at det ikke er en uvedkommen, der logger ind og laver uønskede ændringer. Vi har derfor lavet et loginsystem, hvor kun medarbejdere og administratorer kan logge sig ind gennem en opstillet `HTML-form`, hvor brugeren skal indtaste sine login-oplysninger.

```
<form action="?p=login" method="post">
<div>
  CPR-nummer<br />
  <input type="text" name="cpr1" maxlength="6" size="6" />
  -<input type="text" name="cpr2" maxlength="4" size="4" />
  <br />
  Password<br />
  <input type="password" name="password" />
  <br />
  <input type="submit" value="Submit" />
</div>
</form>
```

Her kan brugeren indtaste sine oplysninger, og `HTML-formen` sender oplysningerne som `post-værdier`. Herefter genlæses `login.php` med de medsendte `post-værdier`.

```
if($_POST["password"] && $_POST["cpr1"]){
  $query = "SELECT password, type FROM ".getDbPre()."users WHERE cpr
    ='".$_POST["cpr1"]."$_POST["cpr2"]."'";
  boolConnectDB();
  $result = mysql_query($query);
  $row = mysql_fetch_array($result);
  if ($row["password"] == md5($_POST["password"])){
    /* Denne session gemmer pass og bruger til senere brug. */
    $_SESSION["cpr"] = $_POST["cpr1"]."$_POST["cpr2"];
    $_SESSION["password"] = md5($_POST["password"]);
    $_SESSION["type"] = $row["type"];
  }
```

```

        header("location: ?p=start");
    }else{
        $errormsg = "Forkert password.";
    }
    boolCloseDB();
} else {
    session_destroy();
}

```

Login.php tjekker nu om de medsendte post-værdier er korrekte. Ved hjælp af post-værdierne, `$_POST['cpr1']` og `$_POST['cpr2']`, henter den det password, der hører til det indtastede cpr-nummer fra databasen. Herefter sammenlignes det indtastede kodeord med det gemte kodeord fra databasen. Hvis sammenligningen er sand, sættes `$_SESSION` lig med de forskellige `$_POST`-værdier. Hvis de indtastede oplysninger ikke matcher dem i databasen, bliver brugeren sendt tilbage til login-siden, og session slettes med `session_destroy()`. Læg mærke til, at der ikke er sat en `session_start()` i login.php. Vi har valgt at sætte `session_start()` i index.php, for at starte den et fælles sted for alle de sider der bruger sessions.

Login-niveau

```

$intSecurityLevel = 2; // kraftigste securitylevel til at starte med
switch($strPage) {
    case 'login':
        $strFileContent .= 'login';
        $intSecurityLevel = 0;
        break;
    ...
}
...
}

```

Når der åbnes en bestemt side vil der i URL'en stå `?p=x`, hvor `x` er et bestemt side-id. I index.php vil den så tjekke dette id og inkludere den rigtige fil. Samtidig finder den ud af, hvem der må besøge siden, gæst, bruger eller administrator. Disse tre grupper får en integer-værdi fra 0 til 2. Herefter køres funktionen `$intSecurityCheck()` fra `common.inc.php`, som returnerer en tilsvarende værdi for den aktuelle bruger. For at kunne besøge siden skal `$intSecurityCheck()` returnere en værdi der er større end eller lig med sidens påkrævede. Fx kræver det en værdi på 1 eller 2 for at se vagtplanen, mens det ikke er tilladt at blive sendt til vagtplanlægnings-siden med en værdi på 1.

```

$intLoginStatus = 0;
if($intSecurityLevel) { // skal den tjekke sikkerheden?
    $intLoginStatus = intSecurityCheck();
    switch($intLoginStatus){ // tjek sikkerheden
        case 2: // admin
            if (!include_once($strFileContent)) { // som admin må man alt, hent filen
                die('Kan ikke finde filen ("'.$strFileContent.'")!');
            }
            break;
        case 1: // bruger
            if($intSecurityLevel <= $intLoginStatus) {
                if (!include_once($strFileContent)) { // indlæs kun filen hvis man
                    må!
                    die('Kan ikke finde filen ("'.$strFileContent.'")!');
                }
            } else {
                header("Location: ?p=start");
            }
            break;
    }
}

```

```

    case 0: // ikke logget ind!
    default:
        if($intSecurityLevel <= $intLoginStatus) {
            if (!include_once($strFileContent)) { // indlæs kun filen hvis man
                må!
                die('Kan ikke finde filen (&quot;'. $strFileContent. '&quot;)!');
            }
            } else {
                header("Location: ?p=login"); // sikkerheden fejlede, gå til login
                skærmen
            }
            break;
        }
    } else { // sikkerhedstjek behøves ikke
        if (!include_once($strFileContent)) {
            die('Kan ikke finde filen (&quot;'. $strFileContent. '&quot;)!');
        }
    }
}

```

Afhængig af, hvilken bruger, der logger ind i systemet, indlæser `index.php` enten en start-side for admin eller en for en almindelig bruger. Hvis brugeren ikke er logget ind, sendes vedkommende tilbage til login-siden. Funktionen `intSecurityCheck()` beskrives i nedenstående afsnit om session.

Session

For at hindre uvedkommende i at logge ind i vores system, har vi sat en session, som er en global variabel. Med en global variabel er det muligt at lave et tjek, der sikrer, at en medarbejder skal være logget ind for at se/benytte vores system.

Når medarbejderen har indtastet korrekte oplysninger og er logget ind, undersøger `index.php` om session indeholder en værdi, som svarer til værdien i databasen. Altså den laver endnu en login-tjek bare med session-værdierne i stedet for post-værdierne.

```

function intSecurityCheck() {
    // Tjek sessions
    if($_SESSION['cpr']&&$_SESSION['password']) {
        // man er logget ind, test om det er rigtigt
        $query = "SELECT password FROM ftf_users WHERE cpr = '". $_SESSION["cpr"]. "'";
        boolConnectDB();
        $result = mysql_query($query);
        $row = mysql_fetch_array($result);
        boolCloseDB();
        if ($row["password"] == $_SESSION["password"]){
            if($_SESSION["cpr"] == 'admin') {
                return 2;
            } else {
                return 1;
            }
        } else {
            return 0; // NIX! Gå væk med dig!
        }
    } else {
        return 0; // NIX! Gå væk med dig!
    }
}

```

Funktionen `intSecurityCheck()` undersøger om `$_SESSION['password']` har samme værdi som kodeordet i databasen og returnerer værdien 2, 1 eller 0. De returnerede værdier bruges i `index`-siden og giver adgang til bestemte sider i vores system.

define('FTF')

For yderligere at holde uvedkommende ude fra systemet, har vi benyttet funktionen `define()`.

```
// Definer FTF sådan at resten af siderne ikke tror man hacker
define('FTF', true);
```

På index-siden har vi defineret 'FTF' sådan, at uvedkommende ikke kan åbne andre sider end index-siden, hvis de kender til filnavnet. Hvis en bruger fx kender til siden `start.php`, og vedkommende skriver stien til filen, kan vedkommende ikke se siden `start.php`, men bliver istedet sendt tilbage til index-siden.

```
if (!defined('FTF') && !defined('DEBUG')) {
    die("Du har ikke tilladelse til at besøge denne side her.<br />");
}
```

Vi undersøger om FTF er defineret på hver side i vores system, undtagen index-siden. Hvis ikke FTF er defineret, bliver brugeren sendt tilbage til login-siden.

md5()

PHP's `md5()`-funktion bruges til at hashe en streng. Funktionen bruges ofte i forbindelse med hemmelige oplysninger såsom password. Vi har benyttet funktionen i forbindelse med at gemme en medarbejders password. Når en admin opretter en medarbejder i systemet, vil medarbejderen få tildelt et password. Vi hasher passwordet, inden det bliver gemt i databasen.

```
if ($_POST['form_pass'] != '') {
    $db_pass = md5($_POST['form_pass']); //hashed
} else {
    $db_pass = $_POST['form_pass_md5'];
}
```

Funktionen `md5()` laver en forståelige streng til en uforståelige streng bestående af 16 byte som hex-værdier. Fx vil `md5('tekst')` returnere `'70bd86742a05868f74acaf095b3d3fd0'`. Ved at hashe før passwordet bliver gemt, kan administratoren ikke se medarbejdernes password. Uvedkommende kan hellere ikke læse og forstå kodeordet. Derfor kan personen ikke logge ind, selvom vedkommende kender brugerens cpr-nummer.

Cpr-tjek

Cpr-tjekket foregår i funktionen `boolCheckCPR()`. Når der benyttes et system, som bruger cpr-nummer til identifikation, er det vigtigt at tjekke, om et cpr-nummer nu også er validt. Denne funktion tjekker til at begynde med, om de første seks cifre, som udgør fødselsdatoen, nu også er en korrekt dato. Sammen med syv cifre tjekker den også århunrede ud fra et snedigt system. Sidste ciffer er lige hvis personen er kvinde og ulige hvis personen er mand. Man kan når man kalder funktionen sætte personens køn hvis denne kendes. Hvis ikke kønnet kendes på forhånd tjekker den ikke om det sidste ciffer passer til kønnet. Derimod tjekker den altid om kontrolcifferet, sidste ciffer, er korrekt. Der er en bestemt udregning som skal foretages ud fra de første ni cifre og dermed får man tiende ciffer.[6]

Følgende er en illustration af hvordan kontrolcifferet findes på følgende cpr-nummer: 11111-1118 (dette er et validt cpr-nummer):

```
CPR-nummer:  1 1 1 1 1 1 1 1 1 8
              * * * * * * * *
Kontroltal:  4 3 2 7 6 5 4 3 2
              = = = = = = = =
              4+3+2+7+6+5+4+3+2 = 36

36 mod 11 = 3
11 - 3 = 8
```

Hvert ciffer i cpr-nummeret ganges med deres respektive kontroltal og alle resultater lægges sammen. Udregningen giver her 36. Herefter dividere vi det med 11 og finder resten, her 3. Til sidst trækker vi resten fra 11 og får kontrolcifferet, her 8. Dette princip kaldes for “modulo 11”.^[7]

Hvis der ikke sker nogen fejl undervejs vil funktionen returnere `true` og dermed er cpr-nummeret korrekt.

Krypteret linje

Selv med alle vores password-tjek er vores system ikke sikkert nok. Forbindelsen mellem klienten og server er ikke krypteret, altså er oplysningerne, som brugeren indtaster ved login, ikke krypteret. En hacker kan opsnappe de informationer, som en bruger indtaster, inden de sendes til serveren. Selvom passwordet ligger hashet i databasen, kan en hacker opsnappe den uhashede tekststreng på netværket inden den når serveren.

3.2.3 Administration

Installation

Når et firma skal starte med at bruge vores produkt, skal det installeres. Vi ønsker dog ikke at lave en Wise-install eller lignende, da det skal være muligt at oprette produktet på en ekstern server via FTP. Det eneste man skal gøre er at kopiere filerne og derefter køre `install.php`. Den beder én om at skrive databasens navn, adresse, brugernavn og kode, samt et tabel-prefix. `install.php` opretter alle tabeller som systemet bruger og en fil kaldet `config.inc.php` som indeholder nogle globale variabelklæringer:

```
$dbAddr = 'localhost';  
$dbUser = 'user';  
$dbPass = 'pass';  
$dbName = 'name';  
$dbPre = 'ftf_';
```

`index.php` inkluderer `config.inc.php`, og hvis den ikke findes vil den melde fejl. Variablerne bliver brugt i `boolConnectDB()` som skal bruge databasens adresse, brugernavn og kode. Tabel-prefixet bliver brugt i alle de filer der kalder til databasen sådan at den finder den korrekte tabel. Dette er gjort for, at brugeren selv kan vælge navnene på tabellerne, sådan at vores produkt ikke konflikter med andre produkter, der eventuelt bruger databasen. Ud over at oprette tabeller og `config`-filen, indsætter `install.php` også nogle standard værdier i tabellerne. Bl.a. bliver en administrator oprettet med et standard password.

Nu er systemet klar til brug, og administratoren kan begynde på at oprette sine medarbejdere og få lagt en vagtplan.

Administration af medarbejdere

I dette afsnit beskrives funktionen for administration af medarbejdere. Administratoren skal have mulighed for at administrere medarbejderne samt deres oplysninger. Hvis der bliver ansat en ny medarbejder, skal denne oprettes i systemet, og dette kan kun administratoren gøre. Administratoren er også den eneste, der kan ændre nogle bestemte informationer hos en medarbejder som fx, om vedkommende er ansat eller ej eller, hvilken type medarbejder det er osv. Disse informationer kan den almindelige medarbejder ikke ændre. Derfor er det vigtigt, at denne funktion eksisterer for administratoren.

`Adminworker.php` bruges, når administratoren skal oprette en ny medarbejder eller ændre oplysninger for allerede eksisterende medarbejdere. Hele siden er en HTML-form, hvor medarbejdernes oplysninger skal indtastes. Ved siden af HTML-formen er der en liste, hvor alle eksisterende medarbejdere findes. Her kan administratoren hente data om en eksisterende medarbejder og derefter rette disse oplysninger.

Vi vil herunder dokumentere kildekoden bag funktionerne i `adminworker.php`. `Adminworker.php` indeholder flere funktioner, som er listet herunder.

- Tilføjelse af medarbejder
- Gem oplysninger i databasen
- Redigering af medarbejder
- Hent oplysninger i databasen
- Sletning af medarbejder

<form>

Ved oprettelse af en ny medarbejder indtastes de nødvendige oplysninger om den pågældende medarbejder, hvilket afsluttes ved at klikke på [Tilføj]. Derefter bliver de indtastede oplysninger sendt til databasen. Da `adminworker.php` skal bruges til alle funktionerne, er det nødvendigt at vide, hvornår `adminworker.php` skal gøre det ene og det andet.

```
<?php
  if ($id != ''){
    //redigeringsform
    echo "<form name='form' method='post'
        action='?p=adminworker&f=edit&id=".$id."'>";
  }else{
    //tilføjelsesform
    echo "<form name='form' method='post' action='?p=adminworker&f=add'>";
  }
?>
```

For at fortælle `adminworker.php`, at den skal fx tilføje en ny medarbejder, skal den have sendt en værdi med, så `adminworker.php` ved, at den skal tilføje og ikke, fx slette en medarbejder. I starten af formen har vi sat en `if`-sætning, som tjekker om variabelen `id` indeholder en værdi. Hvis den er sand, skal formen bruges til tilføjelse af en ny medarbejder, ellers skal formen bruges til redigering. Vi benytter værdien `post` til attributten `method` for at sende oplysningerne fra vores `input`-boks som `post`-variabler.

```
Fornavn:<br />      <!-- Fornavn -->
<input name="form_firstname" type="text" value="<?php echo $edit_firstname; ?>" />
```

Alle input-bokse har fået tildelt et navn under `name` som `adminworker.php` refererer til, når den henter oplysningerne fra formen og skriver dem ind i databasen. Værdien af en bestemt input-boks bliver defineret af attributten `value`. Ved oprettelse af en medarbejder har `value` ingen værdi, ved redigering vil den få tildelt værdien fra fx `$edit_firstname` som i ovenstående eksempel.

Funktioner

Alle oplysninger om en medarbejder bliver sendt til siden igen med `post`-metoden, hvorefter oplysningerne gemmes i databasen. Øverst på siden undersøges det, om der er sendt oplysninger med fra formen. Hvis det er sandt, så gemmer `adminworker.php` alle `post`-værdierne i nogle variabler.

```
if (($_POST['form_cpr_date'] != '') and ($_POST['form_cpr_con'] != '')){
...
    $db_firstname = $_POST['form_firstname'];
...
}
```

I ovenstående kodeeksempel undersøges der vha. en `if`-sætning, om `post`-værdierne `form_cpr_date` og `form_cpr_con` indeholder noget. Hvis det er sandt, sætter den `$db_firstname` lig med `post`-værdien. Samme procedure gælder for de andre input-bokse.

`Get`-variablen `f` indeholder en værdi, der fortæller, om der skal tilføjes en medarbejder eller ændres oplysninger.

```
if($f == 'add' && $_POST['form_cpr_date'] != '' && $_POST['form_cpr_con'] != ''){
    $write_insert = ...,$db_firstname."','".$db_surname."',...;
    $write_query = "INSERT INTO ftf_users(...,firstname,surname,...) VALUES
        (\".$write_insert.\");
    mysql_query($write_query, $dbLink); //smider værdierne i databasen
    header("Location: index.php?p=adminworker"); //genlæser siden
}
```

Hvis `if`-sætningen er sand, sætter den alle `$db`-værdierne ind i variabelen `$write_insert`. Herefter bruger vi PHP's database-funktion `mysql_query` til at udføre `mysql query`'en. I dette tilfælde gemmer den alle oplysningerne om en medarbejder i databasen.

For at spare på kolonner i databasen, har vi valgt, at oplysninger om medarbejderen som fx adressen, skal sammensættes. Adressen har vi opdelt i flere undervariabler såsom `vej`, `husnr.`, `postnr.` og `by`. Alle underpunkterne bliver sat sammen med PHP-funktionen `implode`.

```
$db_address = ($_POST['form_street'].':'....':".$_POST['form_city']);
```

Her bliver værdierne adskilt med ":" og samlet til variabelen `$db_address`. Når administratoren henter oplysninger fra databasen bruger vi PHP-funktionen `explode` til at adskille variabelen `$db_address` til undervariablerne.

Liste

For at få en oversigt over de eksisterende medarbejdere, har vi lavet en liste ved hjælp af HTML-elementerne `select` og `option`.

```
while ($list_row = mysql_fetch_array($list_result)){
    echo "<option value='".$_list_row['cpr']."'>".$_list_row['firstname']."
        ".$_list_row['surname']."</option>";
}
```

Når administratoren kommer ind på siden, henter den navnene på alle de eksisterende medarbejdere i databasen. `while`-løkken gennemløbes så mange gange, som der er af medarbejdere.

Ved hver medarbejder udskrives `<option>` med attributten `value`, der sættes lig medarbejderens cpr-nummer. `value`-attributten skal bruges, hvis administratoren vil ændre oplysninger om en medarbejder. Her sender formen `cpr`-værdien til siden igen, som så henter den pågældende medarbejders oplysninger.

`adminworker.php` er administratorens værktøj til at ændre, oprette eller fjerne medarbejdere fra systemet. Det er også vigtigt for administratoren at kunne holde styr på sine medarbejdere, hvilket denne fil giver mulighed for.

Opgave-administration

For at kunne lægge en vagtplan, bliver der nød til at være nogle opgaver, der skal løses. Derfor har vi lavet en side til vores system, hvor der kan tilføjes, redigeres og slettes opgaver. Det er kun administratoren, der kan se denne side, da den almindelige medarbejder ikke skal kunne ændre i opgavernes definitioner.

En opgave består af fire dele: et navn, en beskrivelse, en type og en indikator, der fortæller, om opgaven er primær eller sekundær.

Tilføj/Rediger/Fjern

Denne del af systemet foregår på nogenlunde samme måde som bruger-administrationen. Her udfyldes input- og check-bokse. Når der trykkes `Rediger`, genindlæses samme side, hvor felterne blot er udfyldt på forhånd.

Type - hvad er det?

Den del af opgaven der hedder "type", er i databasen gemt som et heltal, der beskriver en binær kombination. Når opgavens type skal bestemmes, afkrydses blot nogle check-bokse. Check-boksene repræsenterer bit-værdierne i heltallet. Scriptet regner herefter selv tallet ud. Disse check-bokse repræsenterer de forskellige evner der er krævet for at løse opgaven.

De muligheder administratoren har for evner, når opgavens type skal bestemmes, er forud programmeret i vores kode, og er derfor ikke dynamisk. Vi kom frem til, at der kunne være brug for følgende evner i Netto:

- Ordne flasker
- Oprydning
- Sidde ved kassen
- Sæt varer på hylder
- Varebestilling
- Åbne/Lukke

En opgave kan derfor kræve en hvilken som helst kombination af ovenstående evner. Dette giver administratoren stor frihed til, hvordan vagtplanlægningen skal styres.

Det vil være muligt at give hver specifik opgave, der skal laves i butikken, hver sin opgave i systemet. Et eksempel på dette kunne være at sidde ved kasse nr. 1, eller at sætte kager på hylder i kageafdelingen.

Der er også mulighed for, at give hver type opgave i butikken hver sin opgave i systemet. Et eksempel på dette kunne være: At sidde ved kassen eller at ordne flasker.

En helt tredje mulighed er: At give hver ansættelses vilkår sin egen opgave. Det kunne være kasseassistent eller flaskedreng. Her skal hver opgave bruge flere evner, fordi at en flaskedreng også skal rydde op, osv.

Denne valgfrihed gør at administratoren ikke behøver at lave den store omstilling, hvis der skiftes til vores system, da administratoren selv bestemmer, hvordan opgaverne skal defineres.

Type - Hvordan det?

Hver opgaves type er som sagt gemt som en binær kombination. Typen består af seks bit, en for hver evne. Hvis opgaven kræver evnen er bitten 1, ellers er den 0. Hvis en opgave kun kræver, at medarbejderen kan ordne flasker ville typen være 100000 (binær) \rightarrow 32 (decimal). På denne måde får enhver kombination af evner, en unik type.

Hver bruger i systemet har tilsvarende en type, der udtrykker, hvilke evner vedkommende har. Dermed bestemmer brugeres type, hvilke opgaver brugeren kan løse. Hvis der skal checkes om en bruger kan løse en opgave, bliver funktionen `boolTaskCheck()`⁸ brugt. Denne funktion sammenligner de to bit kombinationer og ser, om brugeren har tilstrækkelige evner til at løse opgaven.

⁸Funktionen er beskrevet i afsnit 3.2.1

Vagtplanlægning

Når administratoren er logget ind på systemet, er der en række muligheder. En af disse muligheder er at lægge en vagtplan. Denne vagtplanlægning foregår gennem flere trin. Administratoren mærker dog kun til det ene, da resten af trinene foregår pr. automatik.

Det første delproblem i vagtplanlægningen er, for administratoren, at finde ud, hvilke opgaver der skal løses de pågældende dage. Administratoren indsætter de opgaver, der skal løses i et skema. Når vedkommende er tilfreds med dette, kan systemet sættes i gang med at lægge vagtplanen.

Angive input til en vagtplan

Vi vil nu se nærmere på den programstump, der skal hjælpe administratoren med at give det rette input til vagtplanen.

Administratoren angiver input ved hjælp af filen `createplan.php`. I `createplan.php` vises en tabel over én uge. Der er celler i hver ugedag (morgen og aften). Administratoren tilføjer de opgaver, der er defineret i opgaveoprettelsen.

For at administratoren grafisk kan følge med i, hvad der sker under opgavetilrettelæggelsen, og undgå at opdatere siden, hver gang der sker en ændring. Vi har valgt at bruge et klient-script til at løse denne opgave. Klient-scriptet bruges til dynamisk at vise et grafisk overblik, hvorefter alle data kan sendes videre til den egentlige vagtplanlægningsalgoritme.

```
function writeToCell(){
    if(!selectedId){
        alert("Du skal vælge et tidspunkt");
        return ;
    }
    target = document.getElementById(selectedId);
    hiddenFormValue = "";
    target.innerHTML = "";
    isDataToWrite = false;
    for(i=0;i< <?php echo $counter; ?> ;i++){
        from = document.getElementById("opgave"+i);
        if(!isNaN(from.value) && from.value && from.value.indexOf(".") == -1 &&
            from.value != 0){
            isDataToWrite = true;
            target.innerHTML += from.value + " x " + from.name + "<br />";
            for(j=1;j<=from.value;j++){
                hiddenFormValue += ':' + aJTask[i];
            }
        }else{
            from.value = "";
        }
        target.innerHTML += '<input type="hidden" name="'+selectedId+'"'
            value="'+hiddenFormValue.substring(1)+'" />';
    }
    if(!isDataToWrite){
        target.innerHTML = "<br />";
    }
}
```

Funktionen `writeToCell()` skriver de inputs, som administratoren har angivet, ind i de rigtige celler i uge-tabellen. Først opstilles de valgte opgaver i en grafisk liste i cellen. Derefter kommer det mere interessante: Et input af typen `hidden` oprettes i et layer i dokumentet, med de data, der er valgt. Dataene listes som:

`data1:data2: ... :datan-1:datan`. Dette sendes videre til filen `timeplan.gui.php`, som er i stand til at oversætte disse data til opgaver. Administratoren angiver også en startuge for vagtplanen, og hvor mange uger vagtplanen skal vare.

Behandling af administratorens input

Efter, at administratoren har sendt dataene videre til næste side (`timeplan.gui.php`), vil dataene blive behandlet, og en vagtplan vil blive forsøgt lagt.

```
for ($i=0; $i<$_POST["numOfWeeks"]; $i++) { // looper uger igennem
    for ($j=0; $j<7; $j++) { // looper ugedage igennem
        for ($k=1; $k<=2; $k++) { // looper morgen og aften igennem
            if ($_POST["time".$j."-".$k]) {
                $aTask[$i][$j][$k] = explode(":", $_POST["time".$j."-".$k]); //
                indsætter input som array i $aTask
            }
        }
    }
}
```

Ovenstående kode opretter et array med administratorens input. Inputtet opdeles ved “:” og indsættes i et array. Dette sker ved funktionen `explode(":", $_POST["time".$j."-".$k])`.

```
$objTimeplan = new Timeplan($aTask, $intStartDate);
$objTimeplan->solveTimePlan();
```

Et nyt objekt, af typen `Timeplan`, bliver oprettet med administratorens input og startdatoen som argumenter. Derefter kaldes funktionen `solveTimePlan()`, som er en intern funktion i class'en `Timeplan`. Funktionen `solveTimePlan()` eksekveres derefter. Hvordan `solveTimePlan` fungerer, vil vi komme nærmere ind på senere.

Hvis `solveTimePlan()` lykkes, vil vagtplanen blive gemt i databasen, hvorefter den kan vises vha. `showtimeplan.php`.

Generering af vagtplan

At lægge en vagtplan kan virke som en uoverskuelig opgave for de fleste personer. Jo flere faktorer, der skal tages højde for, jo mere kompliceret bliver det at lægge vagtplanen.

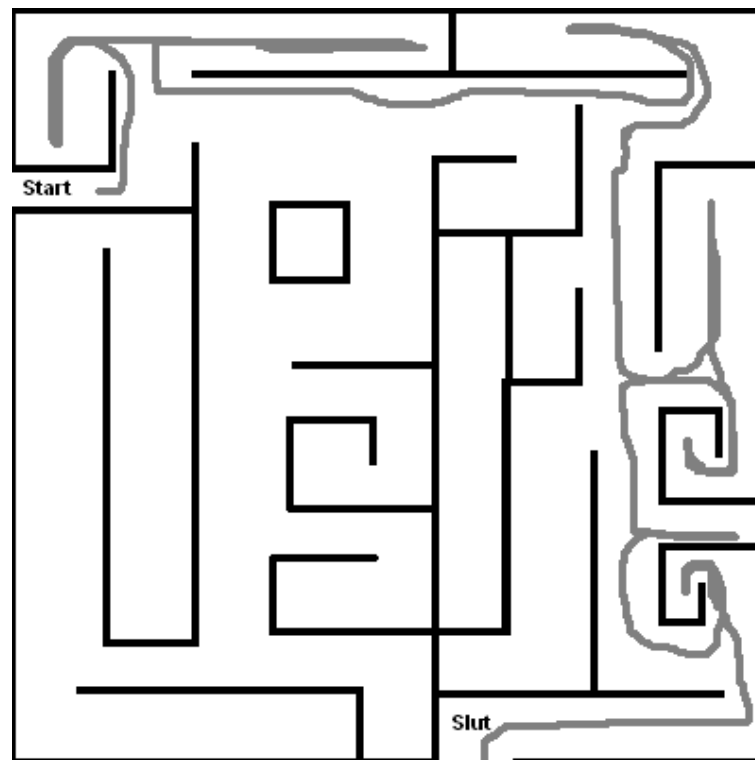
Vi ønsker at få en computer til at lægge vagtplanen for os. Vi skal have fremstillet en algoritme, der gør dette for os. Inden vi begynder med at beskrive vores algoritme til vagtplanlægning, vil vi beskrive et andet problem, som kan give indsigt i de principper, vi bruger til vores algoritme.

Labyrintproblemet

Vi kan forestille os, at vi har en labyrint foran os. Vi skal finde udgangen. Dette kan som sagt også virke uoverskueligt, men griber vi det systematisk an, kan vi sætte nogle regler op, således vi når en løsning, hvis sådan en da findes.

- Labyrinten er løst, når vi når udgangen.
- Hver gang vi skal træffe et valg, træffer vi det valg, der ligger længst til venstre.
- Hvis vi går ind i en blindgyde, går vi tilbage til det sidste sted vi traf et valg, og vælger næste mulighed.
- Der findes ingen løsning på labyrinten, hvis vi bliver nødt til at gå tilbage til før vi traf det første valg.

Figur 3.4 er en illustration af, hvordan labyrintproblemet bliver løst ved hjælp af de ovenstående regler.



Figur 3.4: Illustration af labyrintproblemet

Den opskrevne løsningsstrategi for labyrintproblemet kan udføres af en computer ved hjælp af en backtracking-algoritme. En backtracking-algoritme begynder systematisk at prøve sig frem for at finde en løsning. Der er opstillet et kriterie for, hvornår en løsning er sand, og computeren vil blive ved, indtil løsningen er sand - i dette tilfælde når udgangen af labyrinten er nået.

Principperne for at løse labyrintproblemet kan overføres til problemet med vagtplanlægning. Vi vil få computeren til systematisk at prøve på at lægge vagtplanen, indtil en løsning er nået.

Vagtplanlægnings-algoritmen

Med labyrintproblemet i baghovedet, kan vi nu begynde at udarbejde en algoritme til løsning af vagtplanlægningsproblemet.

Administratorens input er, som beskrevet tidligere i dette afsnit, et 4-dimensionalt array. Vores output har vi derfor valgt at designe på samme måde. Dette array er sideløbende med inputtets.

Det første vores algoritme skal gøre er, at finde det første tomme felt i dette array.

```

$intWeekNum = 0;
$intWeekDay = 0;
$intTask = 0;
$intTime = 0;
$boolReturn = false;
$boolFoundTask = false;
for($i=0 ; $i < count($this->aTaskPlan) && !$boolFoundTask ; $i++){//loop uger
    for($j=0 ; $j < 7 && !$boolFoundTask; $j++){//loop ugedagene
        for($k=1 ; $k <= 2 && !$boolFoundTask; $k++){//loop morgen/aften
            for($l=0 ; $l < count($this->aTaskPlan[$i][$j][$k]) &&
                !$boolFoundTask; $l++){//loop tasks (chefens input)

```

```

        if(!$this->aTimePlan[$i][$j][$k][$l]){
            $boolFoundTask = true;
            $intWeekNum = $i;
            $intWeekDay = $j;
            $intTime = $k;
            $intTask = $l;
        }
    }
}

```

Den ovenstående kode-stump gennemløber arrayet `$aTimePlan` til den finder en vagt, der ikke er besat. Finder den en vagt, der ikke er besat, vil `$boolFoundTask` få værdien `true`, og løkken vil blive brudt, da `!$boolFoundTask` vil returnere `false`. `$intWeekNum`, `$intWeekDay`, `$intTime` og `$intTask` får hver deres værdi, således vi kan lokalisere det rigtige punkt i arrayet senere i koden.

Når der er fundet en vagt, der ikke er besat, vil programmet derpå begynde at indsætte en medarbejder ind på vagten.

```

if($boolFoundTask){
    $this->aUsers = $this->aSortUsers($this->aUsers ... $intTime);
    for($i=0;$i < count($this->aUsers) && !$boolReturn;$i++){
        if(
            $this->typeCheck( ... ) &&
            $this->idleCheck( ... ) &&
            $this->priorityCheck( ... ) &&
            $this->lawCheck( ... )
        ){
            $this->aTimePlan[$intWeekNum] ... [$intTask] = $this->aUsers[$i]->getCPR();
            $boolReturn = $this->solveTimePlan();
            if(!$boolReturn){
                $this->aTimePlan[$intWeekNum]...[$intTask] = false;
            }
        }
    }
}else{
    $boolReturn = true;
}

return $boolReturn;

```

Hvis der er fundet en “tom” plads i vagtplanen (`$aTaskPlan`), når vi ind i `if`-sætningen `if($boolFoundTask)`. Her køres først funktionen `aSortUsers()`. Det er, som navnet antyder, en sorteringsalgoritme. Funktionen får et array af brugere ind som argument, samt tiden til vagtplanen. `aSortUsers()` sorterer derefter arrayet, således at de medarbejdere, der helst vil på arbejde den pågældende periode bliver placeret øverst.

```

function aSortUsers($aUsers, $intDate, $intTime){
    $k=0;
    $aPri = array(0 => 1, 1=>2,2=>0);
    for($i=0;$i<3;$i++){
        for($j=0;$j<count($aUsers);$j++){
            if ($aUsers[$j]->getPriority($intDate, $intTime) == $aPri[$i]){
                $aTemp[$k++] = $aUsers[$j];
            }
        }
    }
    return $aTemp;
} //end aSortUsers

```

Når arrayet `$aUsers` er blevet sorteret efter prioriteringer, skal hver bruger igennem fire tjek, for

at finde ud af, om vedkommende er kvalificeret til at arbejde den pågældende vagt. De fire tjek er hhv.:

- `typeCheck()`, der validerer, om medarbejderen er kvalificeret til at løse opgaven for den pågældende vagt.
- `idleCheck()`, undersøger, om medarbejderen er i gang med at løse en opgave i forvejen
- `priorityCheck()`, undersøger, om medarbejderen i det hele taget kan arbejde den pågældende dag.
- `lawCheck()`, tager højde for 11-timersreglen, se bilag 2⁹.

De ovenstående funktioner returnerer hver især enten `true` eller `false`. Hvis alle funktioner returnerer `true`, er medarbejderen kvalificeret til at arbejde på den pågældende vagt.

```
function idleCheck($intUserCPR,$aTimes,$aTasks){
    for($i=0 ; $i < count($aTimes) ; $i++){
        if($intUserCPR == $aTimes[$i]){
            if(!$this->aTasksTranslate[ $aTasks[$i] ]["secondary"]){
                return false;
            }
        }
    }
    return true;
} //end idleCheck
```

`idleCheck()` er en af de funktioner, der tjekker, om medarbejderen er kvalificeret til at påtage sig den pågældende vagt. `idleCheck()` får cpr-nummeret for den pågældende medarbejder som argument. Alle vagter den pågældende periode gennemløbes i `for`-løkken. Derefter tjekker `idleCheck()`, om der findes en post med det samme cpr-nummer den dag. Hvis det er tilfældet, tjekkes der, om denne vagt er sekundær. En sekundær vagt er en vagt, der ikke optager medarbejderen hele dagen (fx nøgleansvarlig). Det vil sige, at medarbejderen kan tage en anden vagt sideløbende med denne vagt. `$aTasksTranslate` er et array indeholdende alle informationer om opgaverne. Funktionen returnerer `true`, hvis medarbejderen kan arbejde den pågældende vagt.

Hvis alle tjek returnerer `true`, vil brugerens cpr-nummer blive tilført vagtplanen på det rette sted. `solveTimePlan()` vil blive kørt igen, til der ikke er flere ubesatte vagter, og hele funktionen returnerer `true`.

Når `solveTimePlan()` er kørt til ende, vil `$aTimePlan` indeholde en gyldig vagtplan, som er klar til at blive gemt i databasen. Dette sker med filen `timeplan.gui.php`.

⁹Bilag 2: Side 77

3.2.4 Brug af systemet

Vis vagtplan

Da vi har bestemt os for, at der også skal være mulighed for at kunne se sin vagtplan, skal denne funktion også implementeres. Denne funktionalitet bliver til i filen `showtimeplan.php`. I dette afsnit vil der blive beskrevet, hvordan denne er opbygget, og hvordan den fungerer.

Der er en PHP kode før HTML-formen. Her i bliver der sat en `session` for, hvilken visningstype der er valgt for vagtplanen . Dette er for, at scriptet vise vagtplanen på samme måde, som da brugeren forlod siden.

```

if ($_POST['schedule']) {
    $_SESSION['showtimeplan'] = $_POST['schedule'];
} elseif (!$_SESSION['showtimeplan']) {
    $_SESSION['showtimeplan'] = "month";
}
...
<input type="text" name="date" value="<?php echo date("d/m/Y", $intDate); ?>" />
<br />
Vis som:
<br />
<input type="radio" name="schedule" value="month" <?php echo ($_SESSION['showtimeplan']
    == 'month' ? 'checked="checked" ' : '') ?>/>Månedspan<br />
<input type="radio" name="schedule" value="week" <?php echo ($_SESSION['showtimeplan']
    == 'week' ? 'checked="checked" ' : '') ?>/>Ugeplan<br />
...

```

I formen vælges en bestemt dato, samt hvordan resultatet skal vises, enten som uge- eller månedskema. Dette gøres ved hjælp af `input`-elementer med attributten `type="radio"`. For at den kan finde ud af, om det er en måned eller en uge vedkommende leder efter, sættes attributten `value` til enten `"month"` eller `"week"`. Der er kun lavet et enkelt indtastningsfelt, da der kun skal indtastes en dato for at finde enten ugen eller måneden. Indtastningsfeltet er sat til, at når man kommer ind på vagtplanen, så starter den med at vise dags dato, ellers viser den den dato, som er blevet søgt efter, hvis en søgning har fundet sted.

Sætningen `echo ($_POST['showtimeplan']=='month' ? 'checked="checked" ':'')` samt den for `week`, der er nedenunder, gør, at man kun skal vælge at søge efter en måned én gang. Derefter husker `radio`-knappen, om der er ønsket om at blive ved med at søge efter måneder. Ønsker brugeren at søge efter en uge i stedet, skal denne vælge uge. Det samme princip gælder for ugeplan. Hvis søgningen foretages igen, behøves der ikke at være valgt ønsket om visningen igen.

```

if($_POST['date']){
    list($intDay, $intMonth, $intYear) = split('/[-.]', $_POST['date']);
    if (checkdate($intMonth, $intDay, $intYear)) {
        $intWeek=date("W", mktime(0,0,0,$intMonth, $intDay, $intYear));
        $strGET = "&year=".$intYear."&".($_SESSION['showtimeplan']=='week' ?
            "week=".$intWeek : "month=".$intMonth);
    } else {
        $intMonth = date("n");
        $intWeek = date("W");
        $intYear = date("Y");
        $intDay = date("j");
        ...
    }
}

```

Herefter kommer der en `if/else`-sætning. `split()`-funktionen bruges her til at dele `$_POST['date']`, hvor der er `/` eller `-` og gemmes derefter i de listede variable, `$intMonth` osv. Når `$_POST['date']` er behandlet, tjekkes der om den fundne dato er valid, med funktionen `checkdate()`. Hvis datoen er valid, beregnes ugenummeret (`$intWeek`) ud fra datoen. Hvis datoen ikke er valid, sættes dato-variablene til dags dato. `$strGET` gemmer `$intYear` samt enten

`$intWeek` eller `$intMonth` således at de kan blive suffiks til de links der senere bliver genereret ved "Sæt til byt" og "Fjern fra byt". Hvis det ikke bliver gjort, vil brugeren blive sendt tilbage til kalenderen for dags dato, efter at have trykket på et af disse links.

```

if ($_GET['year'] && $_GET['month']){
    $intMonth = $_GET['month'];
    $intYear = $_GET['year'];
    $intDay = 1;
    $strGET = "&year=".$intYear."&month=".$intMonth;
} elseif ($_GET['year'] && $_GET['week']) {
    $intWeek = $_GET['week'];
    $intYear = $_GET['year'];
    $intMonth = date("n", WeekToDate($intWeek, $intYear));
    $intDay = date("j", WeekToDate($intWeek, $intYear));
    $strGET = "&year=".$intYear."&week=".$intWeek;
}

```

Når brugeren vælger at bladre månedvis eller ugevis, vil `$_GET['year']` altid blive sat sammen med enten `$_GET['month']` eller `$_GET['week']` - afhængigt af om brugeren bladrer i hhv. måneder eller uger. Derefter sættes de involverede dato-variabler. Yderligere sættes `$strGET`, som tidligere er beskrevet, igen.

```

if($_GET['changedate'] && $_GET['changetime']){
    $strQuery = "SELECT id FROM ftf_timeplan WHERE date='".$$_GET['changedate']."' AND
        time='".$$_GET['changetime']."' AND user='".$$_SESSION['cpr']."'";
    $sqlResult = mysql_query($strQuery);
    while ($row = mysql_fetch_assoc($sqlResult)){
        strChange($row['id'], $_SESSION["cpr"]);
    }
}
$query = "SELECT ftf_timeplan.id, ... ftf_tasks WHERE ftf_timeplan.user=ftf_users. ...
    .task=ftf_tasks.id ORDER BY ftf_timeplan.date, ... ftf_timeplan.user";
$result = mysql_query($query);

```

Ovenstående `if`-sætning holder styr på, når vagter bliver fjernet eller sat til byt. Betingelserne i `codeif`-sætningen er, at datoen og tiden for vagtbyttet, skal være sat som `get`-variabler. Dette sker når brugeren, på vagtplanen, klikker på [Sæt til byt] eller [Fjern fra byt]. Den del af koden der sørger for byttet, fungerer ved først at finde de involverede opgaver i `ftf_timeplan`, og derefter køre `strChange()` funktionen med dem som argumenter. Forespørgslen til databasen ser således ud: `$strQuery = "SELECT id FROM ftf_timeplan WHERE date='.$_GET['changedate'].' AND time='.$_GET['changetime'].' AND user='.$_SESSION['cpr'].'";` Der bliver bedt om `id` fra tabellen `ftf_timeplan`. `Id`'et bliver hentet der, hvor `time` og `date` er de nævnte `get`-variabler, og hvor brugeren's `cpr`-nummer svarer til den bruger der er logget ind. Resultat af forespørgelsen, bliver sat ind i en `while`-løkke, hvor `$row` bliver sat til en `mysql_fetch_assoc`. Dvs. at når `strChange($row['id'], $_SESSION["cpr"]);` bruges, så kan `id`'et for den pågældende række hentes via `$row["id"]`. Alt dette gør, at når der trykkes på [Sæt til byt] eller [Fjern fra byt], så bliver `change` for alle involverede opgaver i `ftf_timeplan` sat til det modsatte af hvad de er i forvejen - 0 eller 1.

Efter denne funktion er sat på plads, bliver der sendt en forespørgelse. I denne forespørgelse bliver der sendt bud efter alle informationerne fra vores tabel `ftf_timeplan`, desuden bliver der også sendt bud efter fornavn og efternavn fra brugerne samt navnene på opgaverne fra henholdsvis `ftf_users` og `ftf_tasks`. Alle disse informationer bliver gemt i `$result`.

for den uge, når der klikkes.

Det er det samme princip, når det gælder en månedsplan, hvor det blot er for måneder ad gangen. Eneste store forskel er at der her uden videre bare kan subtraheres eller adderes en måned, fra den i forvejen viste måned. Dette skyldes fleksibiliteten i funktionen `mktime()`, der gør at den med en dato som fx. 1/13/2005 godt ved at der menes 1/1/2006.

```
if ($_SESSION['showtimeplan'] == 'week') {  
    echo $taskplan->drawWeekTable($intWeek, $intYear);  
} else {  
    echo $taskplan->drawMonthTable($intMonth, $intYear);  
}
```

Den sidste `if/else`-sætning afgør, hvorvidt der skal tegnes ugekalender eller månedskalender, afhængigt af hvad brugeren har valgt.

Person-søgning

Formålet med søgefunktionen er, at brugerne skal kunne søge efter en eller flere medarbejdere i systemet.

I formen skal der angives de kriterier, som systemet skal lede efter, fx navn, adresse, email osv. Når der trykkes på [Send], bliver der kørt noget PHP-kode, som ser således ud:

```
$query = "SELECT firstname, surname, cpr, address, phone, mobile, email, type, status
FROM ftf_users WHERE ";
if ($_POST['cpr']){
    $query .= "cpr = '" . $_POST["cpr"] . "'";
    $tegn = 1;
}elseif($_POST['firstname']){
    $query .= "firstname LIKE '%" . $_POST["firstname"] . "%'";
    $tegn = 1;
}
...
elseif($_POST['status']){
    $query .= "status = '" . $_POST["status"] . "'";
    $tegn = 1;
}

boolConnectDB();
...
while($row = mysql_fetch_array($result)){
    if ($tegn == 1){
        $dansk = array("Navn", "Efternavn", "Adresse", "Tlf.", "Mobil", "Email",
            "Type", "Status");
        echo "<table>";
        for ($i=0; $i<=7; $i++){
            echo "<tr><td>" . $dansk[$i] . "</td><td>" . $row[$i] . "</td></tr>";
        }
        echo "</table><br />";
    }elseif ($tegn == 1) {
        echo "Personen eksisterer ikke";
    }
}
boolCloseDB();
?>
```

Der er ni forskellige indtastingsfelter, hvor der kan skrives et søge kriterie i. I if/elseif-sætning tjekkes der, hvilket af felterne, der er blevet skrevet i. Derefter bliver den pågældende kode så eksekveret.

Når der er fundet ud af, hvilket et af felterne, der er udfyldt, sendes en forespørgelse til databasen. I denne forespørgelse bliver der bedt om, at få alt, undtaget cpr-nummeret, fra databasen, hvor fx det indtastede cpr er det samme som et af de cpr-numre, der ligger i databasen. I if/elseif-sætningen sættes også en variabel, \$tegn = 1. Denne variabel sørger for, at den efterfølgende kode bliver eksekveret. Ved cpr, address, phone, mobile, email og status, vil der blive sendt en forespørgelse, hvor der kun returneres én person. Ellers bliver der fortaget en anden forespørgelse (LIKE), hvor der bliver søgt efter alle strenge, der ligner det søgte. Søges der fx efter "Jens" som fornavnet, vil forespørgslen returnere de medarbejdere, hvor fornavnet ligner "Jens".

Derefter bliver der forbundet til databasen ved hjælp af boolConnectDB(). Efterfølgende bliver de forskellige variabler sat for at holde styr på det, der skal sendes. Først bliver forespørgelsen sendt til databasen, hvor resultat bliver gemt i variabelen \$result. Siden det er én bestemt række eller et array, som bliver trukket ud af tabellen i databasen, bliver variabelen \$row sat til mysql_fetch_array(\$result). For at kunne finde flere personer, fx med navnet "Jens", bliver vores mysql_fetch_array(\$result) sat ind i en while-løkke. Dette ser således ud:

```
while($row = mysql_fetch_array($result))
```

Der vil blive ved med, at blive trukket rækker ud af databasen, indtil der ikke er flere rækker med den pågældende tekst eller værdi, som man har søgt efter.

For at de informationer, som bliver trukket ud af databasen, kan listes så nemt som muligt, laves der et nyt array, hvori der står passende navne til værdierne i `firstname`, `surname`, `cpr` osv. Inden de forskellige informationer bliver lagt ud i en tabel, skal der laves en `for`-løkke. Dette gøres, så der kan sættes en begrænsning på, hvor mange felter arrayet og tabellen skal være på. `For`-løkkens argumenter er: `$i=0; $i<=7; $i++`. Dette betyder, at den starter ved 0 og tæller op efter, så længe `&i` er mindre eller lig med 7. Løkken køres altså otte gange i alt. Dette er fordi, der kun er otte værdier, som skal tages ud af databasen.

I `else`-sætning fortæller brugeren, at det søgte ikke kunne findes i databasen. Til sidst, bliver forbindelsen til databasen lukket.

Dette script vil kunne hjælpe de ansatte og butikschefen, da de nu har mulighed for at søge efter de oplysninger, som de skal bruge om en medarbejder.

Generering af kalender

For at brugeren kan få overblik over sin vagtplan, har vi lavet en class, der kan tegne uge- og månedskalendere. Det er lavet som en class, således det er let at bruge i programmeringen. For at bruge class'en tilføjer man blot tekst til kalenderen på forskellige datoer og tidspunkter. Når alle begivenheder er tilføjet, beder man det oprettede objekt om at tegne enten en ugeplan eller en månedsplan. Vi vil nu gennemgå de forskellige funktioner der er indeholdt i objektet `Calendar`:

addEvent()

Denne funktion nulstiller alt tekst for det angivne tidspunkt og kalder derefter en ny funktion `appendEvent()`. Funktionen kaldes med tre parametre, `addEvent([dato], [tid], [handling])`.

appendEvent()

Funktionen indsætter tekst for det angivne tidspunkt og kaldes med tre parametre `appendEvent([dato], [tid], [handling])`.

drawMonthTable()

Denne funktion tegner en månedskalender i en tabel. Funktionen kaldes med to parametre, `drawMonthTable([måned], [år])`.

```
$strOutput = '<table class="'. $strClassTable. '" cellpadding="0" cellspacing="0"><tr><td
  class="'. $strClassWeekCell. '"><i>' .ucfirst(strDanishDate($intMonth)) . '<br
  />' . $intYear. '</i></td>' ;
...
$numdays = date("t", $intStartDate); // Antal dage i den givne måned.
...
while ($intCurDate<=$numdays) { // Lav rækker sålænge $intCurDate er mindre eller
  ligmed antal dage i måneden
  $week = date("W", $intStartDate + 24*60*60*($intCurDate-1));
  ...
  for ($i=1; $i<=7; $i++) {
    ...
  }
}
...
$strOutput .= "</table>";
return $strOutput;
```

Når funktionen kaldes, returnerer den `$strOutput`, som er den HTML-kode, der tegner en tabel over den angivne måned. Ved hjælp af `$numdays = date("t", $intStartDate)` henter den det antal dage, der er i den angivne måned. `while`-løkken gennemløber det samme antal gange som antallet af dage. `for`-løkken løber alle ugens syv dage igennem, således at månedskalenderen bliver indelt i uger. Uge-nummeret findes med `date`-funktionen.

drawWeekTable()

Denne funktion fungerer på næsten samme måde som `drawMonthTable()`, men i dette tilfælde genereres en tabel, der viser en ugekalender i stedet for en månedskalender. Den skal kaldes med to parametre, `drawWeekTable([måned], [år])`.

Genereringen af de forskellige kalendere er simpel, da de kan laves med få funktioner blot, der er oprettet dette objekt. `Calendar` bliver brugt både når man skal sætte sine prioriteter, lægge vagtplanen, og hvis der skal ses en vagtplan. `Calendar` er en central class i vores grafiske del af systemet.

Brugerens prioriteter for arbejdstider

Hver ansat skal have mulighed for at angive prioriteter for, hvor vidt de vil arbejde eller ikke kan arbejde på forskellige tidspunkter. Vi kunne forestille os, at en ansat hver eneste uge fx. meget gerne vil arbejde tirsdag aften, men samtidigt ønsker vedkommende aldrig at arbejde fredag aften. Yderligere kan det også være, at vedkommende har en prioritet for en specifik dato længere ude i fremtiden. Hvis vedkommende fx skal til et planlagt arrangement på en specifik dato, skal vedkommende også have mulighed for på forhånd at sætte prioriteten for denne dato til "Kan ikke" - uanset om det så er en tirsdag aften eller ej.

Ovenstående case giver to forskellige definitioner på prioriteter. Den ene er den specifikke dato, og den anden er den ugentlige dag. Den bestemte dato skal internt være prioriteret højere end den ugentlige dag, da en bestemt dato er mere specifik end en ugentlig dag.

`Priority` er et class i vores system, der holder styr på de forskellige ansattes prioriteter, for givne tidspunkter, på forskellige datoer.

Internt i `Priority` arbejdes der med to måder at angive datoer på. Den ene er den specifikke dato, hvor både dag, måned og år er bestemt. Den anden er navnet på en ugedag. Prioriteten for en bestemt ugedag vil derfor gælde ca. 52 gange om året.

En specifik dato angives som et Unix timestamp¹⁰ til den pågældende dato kl. 00:00:00. En ugedag angives som de første tre bogstaver af det engelske ugedagsnavn, fx: "Fri" - Fredag. Ved at bestemme datoangivelserne på denne måde, vil der i PHP ganske simpelt kunne skelnes mellem de to formater ved brug af funktionen `is_numeric()`. Denne funktion returnerer `true`, hvis inputtet er et tal (Unix timestamp) og hvis ikke, så returnerer den `false` (ugedag).

Brugeren kan have forskellige prioriteter på en bestemt dato for de to perioder. Vi har valgt at dele en dag op i: Morgen og Aften. For, at hver bruger kun har en post i databasen pr. dato, og prioriteterne kan gemmes i en celle, har vi valgt at gemme disse data bitvis. Ved at gemme på bitvis fylder dataene mindre, og muligheden for nem udbygning af dagsinddelingen er til stede. Måden prioriteringerne bliver gemt i databasen, er kun relevant for `Priority`, da alt prioritets-kommunikation i systemet vil foregå gennem denne.

Vi har defineret måden prioriteter gemmes på således:

4. bit	3. bit	2. bit	1. bit
2. periode angivet?	2. periodes prioritet	1. periode angivet?	1. periodes prioritet

Denne inddeling gør, at vi har tre prioriteter at arbejde med pr. periode: "Vil gerne"=11, "Neutral"=00, samt "Kan ikke"=10. Alle de dage, hvor den ansatte ikke har angivet prioriteter, er som standard "Neutral". Hvis en ansat for en given dato ikke har valgt en prioritet for morgenen, men har valgt gerne at ville arbejde om aftenen, vil prioriteten for datoen således blive gemt som: 1100 (binær) → 12 (decimal).

Selve class'en indeholder følgende vigtige funktioner:

Priority - constructor

Selve costructoren er ikke en funktion, der skal kaldes manuelt. Funktionen bliver kaldt automatisk, så snart objektet oprettes. For, at objektet ved hvilken brugers data der skal behandles kræves det, når objektet oprettes, at brugerens cpr-nummer angives som parameter.

Funktionen henter alle relevante oplysninger fra databasen, hvilket vil sige de datoer og tilsvarende prioriteter, den pågældende bruger har gemt tidligere. Disse data gemmes så i to

¹⁰Unix timestamp er et givent tidspunkt i sekunder siden 1/1 1970 klokken 00:00:00

forskellige arrays - det ene array indeholder prioriteterne for de specifikke datoer (`$intDates`), hvorimod det andet array indeholder prioriteterne for de ugentlige dage (`$intWeekdays`).

Da constructoren læser alt relevant data fra databasen, er det ikke senere nødvendigt at lave forespørgsler til databasen, før data igen skal gemmes. De ændringer, der foretages i prioriteterne, gemmes midlertidigt kun internt i `Priority`'s to arrays. For, at det senere er muligt at opdatere databasen med ændringer, gemmes posternes id'er fra databasen sammen med prioriteterne i de to arrays.

updateDB()

Hvis de ændringer, der er foretaget i de midlertidige arrays, ønskes gemt i databasen, skal denne funktion kaldes. Funktionen sørger for at opdatere eksisterende prioriteter i databasen og oprette nye, hvis dette er tilfældet.

setPriority()

Denne funktion kaldes, når en prioritet skal angives. Hvis der i forvejen eksisterer en prioritet for det pågældende tidspunkt, opdateres denne, og hvis ikke, oprettes den. Funktionen kræver tre parametre:

`$strDate` = Enten specifik dato, eller ugedag.

`$intTimeOfDay` = Perioden prioriteteten gælder for (1 = Morgen, 2 = Aften)

`$intPriority` = Selve prioriteten (0=Kan ikke, 1=Vil gerne, 2=Neutral)

Hvis prioriteten angives til neutral (2), viderebehandles inputs i funktionen `setNeutral()`, der beskrives nedenunder.

De to forskellige prioriteter, der nu bliver behandlet i denne funktion skal, som defineret, angives i to bit som "10" (Kan ikke) eller "11" (Vil gerne).

Da de to prioriteter er hhv. 0 og 1, er deres decimal værdi ækvivalent med deres binære værdi. Dette medvirker, at den angivende prioritet kan shiftes direkte ind i den, eventuelt, eksisterende prioritets værdi. I PHP shiftes der med operatorene `<<` og `>>` - det betyder at skubbe de binære værdier et givent antal pladser mod enten venstre eller højre. Udover disse to bitvise operatører, anvendes i denne funktion også OR (`|`) samt XOR/"Exclusive OR" (`^`).

setNeutral()

Funktionen sørger for, at de to bit, der gælder for prioriteten for et givent tidspunkt, bliver sat til neutral. Internt i class'en er dette, som beskrevet i tidligere i afsnittet, defineret til "00". Som den førbeskrevne funktion, anvender denne funktion også bitvise operatører for at opnå den ønskede effekt.

getPriority()

Denne funktion skal kaldes bl.a. under vagtplanlægningen, når en brugers prioritet for et bestemt tidspunkt på en bestemt dato ønskes. Først kontrolleres der, om brugeren har angivet en prioritet for en specifik dato, hvis det er tilfældet, returneres denne prioritet. Hvis ikke, kontrolleres det om brugeren har angivet en prioritet for den ugedag, som den specifikke dato falder på - er dette tilfældet, returneres denne. Hvis der ikke er angivet nogen prioritet, returneres 2, der som beskrevet betyder "neutral".

getDayPriority()

Meget i stil med ovenstående funktion, returnerer denne funktion også en brugers prioritet. `getDayPriority()` returnerer dog kun prioriteten, der gælder for en ugedag. Funktionen er be-

regnet til siden, hvor brugeren angiver ugedags prioriteter. Her er det kun ugedags-prioriteterne, der skal vises og ikke prioriteter for specifikke datoer.

getDatePriority()

Denne funktion skal anvendes til samme formål som ovenstående funktion. `getDatePriority()` returnerer dog, i modsætning til ovenstående, kun prioriteter for specifikke datoer.

Som nævnt, skal de to sidstnævnte funktioner anvendes, når brugeren skal præsenteres for sine prioriteter. Den ansatte kan angive og se sine prioriteter under vedkommendes "Brugerprofil". Her kan brugeren vælge mellem prioritets-kalenderen for specifikke datoer og for ugentlige dage. Begge kalendere anvender class'en `Calendar` til formålet.

Notifikation

Ved fx vagtovertagning skal de involverede parter notificeres om, at der er foretaget ændringer i vagtplanen. Som tidligere nævnt, har vi valgt, at denne notifikation skal foregå både via emails og sms-beskeder. Til formålet har vi i PHP oprettet et objekt, der håndterer dette: `Notification`. Denne kan ses i filen `notification.inc.php`.

Når objektet er oprettet, kan der tilføjes modtagere en af gangen via funktionen `addReceipient()`, der tager brugerens cpr-nummer som argument. Når de ønskede modtagere er tilføjet, sendes notifikationen via funktionen `sendNotification()`. Denne funktion kan kaldes med op til fire parametre.

```
$strMessage : Selve beskeden i notifikationen.  
$strSubject : Emnet på notifikationen, der bliver vist både i email og sms-beskeder.  
$boolEmail  : Boolsk variabel der angiver om der skal sendes en email. Er som standard  
              ``true``.  
$boolSMS    : Boolsk variabel der angiver om der skal sendes en sms-besked. Er som  
              standard ``false``.
```

Hvis notifikationen skal sendes til alle brugere af systemet, kan funktionen `sendNotificationToAll()` med fordel anvendes. I modsætning til `sendNotification`, sender denne funktion notifikationen til alle registrerede brugere i systemet - uafhængigt af, hvem der er angivet som modtagere med `addReceipient()`.

For at sende såvel emails som sms-beskeder, benyttes den indbyggede PHP-funktion `mail()`. For at sende email, anvendes funktionen på standard vis, med hhv. email-adresse, emne og besked som parametre. Når en sms-besked skal sendes, så kan det stort set gøres på samme måde, hvis der er adgang til en såkaldt sms-gateway. En sms-gateway er en gateway, der modtager emails og videresender dem som sms-beskeder til de angivne mobil-numre. Sådanne sms-gateways var tidligere gratis stillet til rådighed af teleudbyderne, men dette stoppede de for et par år siden, højst sandsynligt grundet misbrug og spild af en mulig indtægtskilde.

De reklamefri sms-gateways, der findes nu, kræver derfor brugerbetaling. For at teste mulighederne for afsendelse af sms-beskeder, valgte vi den, i forhold til andre, billige sms-gateway stillet til rådighed af Mutax A/S: SureSMS Hosted. Dette var dog ikke videre nogen stor succes, da vore test-beskeder led under flere timers forsinkelse.

WAP

Hele wap-funktionaliteten forgår i `wap.php`. De muligheder den ansatte har via wap-klienten har vi valgt at gøre mere eller mindre begrænset i forhold til web-klienten. Dette skyldes, at de PHP-dokumenter, der genererer XHTML dokumenterne, ikke direkte kan bruges til at generere WML. Selve layout-opsætningen af WML-siderne skal ændres i forhold til XHTML-siderne, da wap-sider er beregnet til mindre displays. Udover, at selve layoutet skal ændres, skal visse XHTML-elementer også erstattes med andre tilsvarende WML-elementer.

Da det ville være en tidskrævende process at konvertere hele systemet til WML, har vi valgt kun at konvertere de vigtigste funktionaliteter: "Login", "Vagtbytte" og "Se vagtplan". Et eksempel på, hvor kompakt data skal opstilles, for at de bliver overskuelige på et lille display, kan ses på figur 3.5. Figuren viser hhv., hvordan et ugeskema vises, og hvordan listen over de vagter, der er sat til byt vises. Ved hver vagt i ugeskemaet er der efter periodens navn ("Morgen" eller "Aften") mulighed for at klikke på et link, der hedder enten [B] eller [F]. De står for hhv. "Sæt til byt" og "Fjern fra byt".

Uge 50:		- Vagtbytte -	
12/12-05 til 18/12-05		- Hovedside -	
Mandag:	Morgen[B] Aften[B]	13/12-05	Aften
Tirsdag:	Aften[B]	Kasse	Overtag
Onsdag:	Aften[B]	14/12-05	Aften
Torsdag:	Fri	Kasse	Fjern
Fredag:	Morgen[B] Aften[B]	18/12-05	Aften
Lørdag:	Aften[B]	Oprydning	Overtag
Søndag:	Aften[B]		

Figur 3.5: Eksempel på to af wap-siderne

På logind-siden skal brugeren indtaste cpr-nummer og kodeord - ligesom på XHTML-siden. Da WAP-klienten oftest er en mobiltelefon, hvor fx tal skal skrives på en speciel, oftest langsommelig måde, er der i WML-standarden angivet en måde at angive, hvilket input en input-boks skal forvente. Dette angives til attributten `format` i input-elementet. Da cpr-nummeret udelukkende består af tal, kan dette angives således: `format="*N"`. Stjernene fortæller at der er tale om et vilkåreligt antal tegn, men kunne erstattes med antal tegn fra 0 til 9. Da vi i cpr-feltet skal have et input på ti tal, kan den maksimale længde ikke defineres der. Her skal vi istedet angive attributten `maxlength: maxlength="10"`. Når disse begrænsninger er defineret for input-feltet, vil tal være valgt som standard input på fx mobiltelefonen, og der vil ikke kunne indtastes mere end ti tegn.[5]

I input-boksen til kodeordet ved vi derimod ikke, hvorvidt der skal indtastes store eller små bogstaver, tal eller antallet af tegn. Derfor kan der ikke sættes en begrænsning på dette input-felt.

WML-dokumenter er inddelt i såkaldte “cards”. Browseren indlæser alle “cards” på samme tid, hvorefter der hurtigt kan skiftes mellem dem, da der ikke skal hentes data for at vise dem. Der skiftes mellem de forskellige “cards” ved at lave hyperlinks, hvor `href`-attributten er sat til `#id`. Id’et henviser til id’et for det “card”, der skal linkes til. To “cards” i et WML-dokument kunne se således ud:

```
<card id="nr1" title="Side 1">
<p>Dette er side 1<br/>
<a href="#nr2">Gå til side 2</a>
</p>
</card>

<card id="nr2" title="Side 2">
<p>Dette er side 2<br/>
<a href="#nr1">Gå til side 1</a>
</p>
</card>
```

I eksemplet er, der på begge “cards”, indsat hyperlinks til det modsatte “card”. Denne egenskab anvender vi i vores system de steder, hvor det er muligt. Da wap oftest benyttes af mobile kommunikationsenheder, skal der tages højde for, at det at oprette en forbindelse og hente data tager lang tid. Hvis brugeren skal hente et dokument med mange “cards”, kan det i første omgang tage lang tid at hente, men efterfølgende går det hurtigt at bladere rundt i dokumentet. Det er vigtigt at finde en passende balance mellem antal “cards”, og hvor meget dokumentet fylder.

3.2.5 Afgrænsning

På grund af tidsmangel, har vi valgt at undlade nogle funktioner i vores system.

- Fuldtidsansatte
- Basisplan
- Godkendelse af vagtplan
- Redigering af vagtplan
- Valg af notifikation

Fuldtidsansatte

I en butik er der nogle ansatte, der er ansat som fuldtidsansat. De skal have 37 timer om ugen ifølge overenskomsten[8]. Det vil sige, når vi laver vores vagtplanlægningsystem, skal den sørge for, at de fuldtidsansatte får de timer de skal have. Vores vagtplanlægningsalgoritme skal altså udtage de fuldtidsansatte først og derefter udtage funktionærer og deltidsarbejdere. Vi har undladt at tage hensyn til denne regel i vores algoritme. Tiden er løbet fra os, og vi vælger derfor at sige, at alle medarbejdere i vores system er ansat på samme vilkår med hensyn til arbejdstimer.

Skulle muligheden for en basisplan implementeres, vil det helt konkret betyde, at det array, hvor alle vagter ligger i, skal fyldes ud inden vagtplanlægningen begyndes.

Hvis vi tager udgangspunkt i vores allerede fremstillede vagtplanlægningsalgoritme¹¹, vil de medarbejdere, der skal i vagtplanen, inden den automatiske vagtplanlægning begyndes bare fyldes på i oprettelsen af arrayet `$aTimePlan`. Arrayet `$aTaskPlan` skal også indeholde en post, der fortæller systemet, at denne plads er besat.

Basisplan

Udover, at vores system automatisk skulle lægge en vagtplan, skulle det gemme en basisplan. Med en basisplan kan systemet gemme de fuldtidsansattes arbejdstider sådan, at butikschefen kan hente planerne frem igen næste gang, der skal lægges vagtplan.

For at gemme en basisplan kan vi tilføje en funktion, der før vagtplanen bliver gemt i databasen, gemmer vagtplanen i en variabel. Butikschefen kan dernæst enten godkende eller rette den vagtplanen, systemet foreslår. Godkender butikschefen den, gemmes den først da i databasen.

Godkendelse

Ifølge vores behovsanalyse, skal en butikschef godkende en vagtplan hver uge, og de nøgleansvarlige skal godkende en vagtplan ved lukketid. Denne funktion har vi ikke med i vores system. Målet var at lave en algoritme, der automatisk kunne lægge en vagtplan. Men at kunne godkende vagtplanen er også en vigtig funktionen, som vi kunne udvide systemet med senere.

Godkendelsesfunktionen kan vi tilføje ved at tilføje en bolsk variabel, der fx hver dag automatisk sættes til `false`. Ved godkendelse ændrer variabelen sig til `true`, som så betyder godkendt.

Den ugentlige godkendelse kan ske på samme måde, hvor en anden variabel holder styr på, om vagtplanen er blevet godkendt for den pågældende uge.

¹¹Se side 48

Redigering

I vores system er der mulighed for at bytte vagter med andre brugere. Administratoren har dog ikke mulighed for at flytte rundt på vagterne. Dette kunne være smart, da vedkommende dermed kunne ændre på vagtplanen, hvis den automatisk genererede vagtplan ikke forekom tilfredsstillende.

Redigeringen kan tilføjes ved at hente de gemte data for en vagtplan fra databasen, derefter kan butikschefen lave sine ændringer til vagtplanen. Denne funktionen er ikke så svær at tilføje, da vi allerede har lavet sådan en funktionen, som bliver brugt, når en medarbejder vil ændre sine personlige oplysninger.

Notifikation

Medarbejderne i vores system kan sætte en vagt til byt. Når de gør dette, vil vagten sættes på en oversigtsliste over alle de vagter, der er sat til byt. Hvis en anden medarbejder overtager en vagt, får den medarbejder, der har sat vagten til byt, en besked via email og sms-besked om, at vagten nu er blevet overtaget af en anden. Her skulle medarbejderen have muligheden for at vælge, om vedkommende vil have besked pr. email og/eller sms-besked. Tiden tillader ikke at denne mulighed bliver implementeret i vores system. Vi mener heller ikke, at det er så relevant, da målet bare var at give en medarbejder besked, når vedkommendes vagt er blevet overtaget.

Vores nuværende system sender en email til medarbejderen som standard. Før der også bliver sendt sms-beskeder, skal der ændres en variabel i PHP-koden. Dette er gjort, da det ellers vil koste penge hver gang, der skal sendes notifikation.

I fremtiden kunne vi implementere funktionen ved at tilføje en ekstra variabel, der holder styr på, om medarbejderen vil have besked via email og/eller sms. Fx kan variablen blive sat til 1 for sms, 2 for email, 3 for begge og 0 for ingen notifikation.

Ethvert system kan udvides på en eller anden måde. Vi har selv en del ting, vi gerne vil udvide vores system med, men tiden tillader os ikke at arbejde mere ved det. Vi har formået at udvikle et system, der virker efter hensigten. Et system, der automatisk kan genere en vagtplan, og tillade medarbejdere at bytte og overtage vagter via en web- eller wap-klient. Alt i alt fungerer vores system udemærket, men der skal flere funktioner og justeringer til systemet, før det kan bruges af en virksomhed.

Kapitel 4

Afslutning

4.1 Vurdering

Da, vores system er færdigt, er det vigtigt at lave nogle test på systemet for at se, om systemet opfylder de krav vi stillede til det. Vi ville teste følgende:

- Om vores `install.php` virkede på andre servere end vores egen.
- Om web-delen virkede.
- Om wap-delen virkede.
- Om systemet var brugervenligt (kunne de finde ud af det?)

Vi havde testet opponentgruppens system, så de gik med til også at teste vores. Vi syntes det var oplagt, da de dermed kunne få indblik i systemet før eksamen. Det vil gøre det lettere at forklare, hvordan de forskellige dele virker, hvis de allerede kender funktioner i systemet.

4.1.1 Test af systemet

Vi bragte vores system til testgruppen på en USB-nøglering. Det var en zip-fil, som blev pakket ud på deres server. Herefter fik de at vide at de skulle gå til mappen via en web-browesr. Her stod der at system skulle installeres (trykkes på [Install]), hvilket de gjorde. Her stødte vi på vores første fejl: Systemet kunne ikke skrive til en af filerne. Dette medførte at systemet ikke kunne definere præfix til tabellerne i databasen.

Efter denne fejl var rettet, fortsatte de med testen. De fik udleveret administrator login. Herefter loggede de på og fik af vide, at de blev nød til at oprette nogle brugere. Det gjorde de så, og det voldte ikke nogle problemer. De forsøgte herefter at lægge en vagtplan, men fik dog nogle intetsigende fejl-meddelelser ud. Testgruppen fik fortalt, hvad de betød - der manglede flere brugere før at der kunne lægges en vagtplan. De oprettede nu flere brugere, og fik succesfuldt lagt en vagtplan. Den intetsigende fejl-meddelelse blev efterfølgende rettet.

Efter vagtplanen var lagt prøvede de at bytte nogle vagter frem og tilbage. Det gik fint, og fejl-meddelelserne var mere sigende her.

Til sidst var de inde via wap, hvor de så de forskellige muligheder der'. Alle funktioner fungerede efter hensigten, og de havde ingen klager over funktionaliteten. Det eneste problem var at de to testede mobil SE k700 og k750, viste tabellen, der indeholder ugevagtplanen, lidt forskelligt. Problemet opstod når en celle i en række indeholdt to linjer, mens en anden celle i samme række kun indeholdt en linje. Dette opstår når fx brugeren har vagter både morgen og aften en bestemt dag, fordi "Morgen" og "Aften" står på hver deres linje, mens ugedagsnavnet kun fylder en linje. Dette medfører, at cellen med en linje får samme størrelse som cellen med to linjer. På den først nævnte mobiltelefon blev resultatet vist som ønsket, da ugedagsnavnet blev placeret i toppen af den udvidede celle. Den anden mobiltelefon viste ugedagsnavnet nederst i cellen, hvilket betød, at det var svært at se, hvilke vagter, der var gældende for, hvilke dage (Se figur 3.5, fredag). I WML-standarden, er der dog desværre, hverken mulighed for at sætte `valign`-attributten, eller `border`-attributten i tabellen. En af disse løsninger ville ellers have været en nem udbedring af problemet, hvis de var understøttet i WML - hvilket de er i XHTML.

4.1.2 Vores vurdering

I dette afsnit vil vi beskrive vores vurdering af systemet. Dette vil vi gøre i forhold til, hvad vi havde sat af mål, og hvad vi selv havde forventet af vores produkt.

Vi har nået at lave en web-klient, hvorfra medarbejderne kan få adgang til systemet, og hvor de kan se deres vagtplan, bytte dem med andre samt ændre deres brugerprofil. Med brugerprofil menes personlige oplysninger samt prioriteter for arbejdstider.

Vi har også fået lavet en wap-klient, hvor medarbejderne kan se deres vagtplan, sætte en vagt til byt eller overtage en andens vagt.

Det sværeste var at lave en algoritme, der automatisk kunne lægge en vagtplan. Vi fik konstrueret algoritmen, men vi har ikke taget højde for fuldtidsansatte¹. Butikschefen skal først oprette nogle opgaver til vores vagtplanlægningssystem. Ud fra opgaverne generer vores algoritme så en vagtplan.

Så ud fra dette, har vi næsten nået alle vores mål. Løsningerne til disse mål fungerer stort set som de skal. Vi synes selv, at vi har fået systemet til at virke, selvom der kunne være nogle enkelte ændringer eller optimeringer, som måske kunne få det sidste til at virke endnu bedre. Dette havde vi desværre ikke tid til.

¹Læs mere under 3.2.5 afgrænsning, side 64

4.2 Konklusion

Ud fra vores interview med butikschefen i Netto, Danmarksgade, fik vi klargjort, hvilke mangler de havde i deres nuværende system til vagtplanlægning. Vi valgte at koncentrere os om hans ønske om at kunne lægge en vagtplan automatisk. Udover dette måtte der laves både små og større funktioner rundt om, før det blev muligt at ligge selve vagtplanen automatisk.

Til selve den automatiske vagtplanlægning valgte vi en kombination af backtracking og vores egne ideer til, hvorledes algoritmen skulle opbygges. Omend det tog længere tid end beregnet, fik vi løst de vigtigste af vores krav og fik udviklet et fungerende system. Efter vi mente udviklingen var færdig, bad vi vores opponentgruppe om at teste systemet for at se, om systemet var brugbart og funktionelt nok. Desuden får vores opponentgruppe en bedre indsigt i vores produkt, så den evt. får nemmere ved at forstå opbygningen og funktionaliteten. Ud fra testen fik vi rettet nogle småproblemer, og vi fik en vurdering af systemet fra andre end os selv.

Når vi ser tilbage på vores mål, og hvordan vores produkt er blevet, kan vi se, at vi har fået løst de fleste opgaver, som vi havde sat os for. Der er dog opgaver vi ikke kunne nå at løse, som vi var blevet nødt til at afgrænse os fra. Her har vi fx funktionen for en basisplan, og hvordan vi skulle indleje de fuldtidsansatte ind i vores automatiske vagtplanlægning.

Vi fik opstillet vores mål i vores problemformulering, samt nogle krav og funktioner der skulle med i vores system.

Vores første problem var at få byttet vagter imellem medarbejderne. Vi konstruerede først nogle class'es og funktioner, som vi skulle bruge for at kunne se en vagtplan. For hver enkelt medarbejder, skulle der være en værdi angivet for deres vagter, der repræsenterede om vagten var sat til byt eller ej.

Der var forskellige former for notifikation, som vi ville have med i vores system. Her var der email og sms notifikation. Det lykkedes at implementere begge, dog med nogle eksterne problemer med sms.

Klienterne for både administratoren og den almindelige bruger er stort set ens. Den almindelige bruger har mulighed for at sætte sine vagter til byt og rette sine personlige oplysninger, hvis disse har ændret sig efter oprettelsen; og selvfølgelig har brugerne mulighed for at se sin vagtplan.

Den administrerende bruger har flere muligheder og funktioner, da denne bruger fx skal kunne oprette eller slette en medarbejder. Administratoren kan logge ind og lægge en vagtplan ved hjælp af autogenereringsfunktionen. Vedkommende har også mulighed for at ændre hver enkelt medarbejders informationer, herunder hvilke evner medarbejderen har, om vedkommende er ansat eller ej, og hvilken form for ansættelse vedkommende har.

Systemet tjekker om det er en almindelig bruger eller en administrerende bruger, som logger ind, hvorved klienten bliver tilpasset den pågældende bruger.

Idéen om at lave systemet modulbaseret er hurtigt blevet forkastet grundet tidsmangel. Vores system er bl.a. derfor ikke så fremtidssikret som vi ønskede.

Vores valg af XHTML, PHP og MySQL gav ikke problemer. Det fungerede udemærket og dem af os, der ikke kendte til førnævnte lærte det hurtigt.

4.3 Perspektivering

Realisering

I dette afsnit vil vi kigge på, hvorvidt vores system vil kunne anvendes i et virkeligt firma.

Vi opfyldte størstedelen af vores opstillede krav, men da vi måtte afgrænse nogle af de mindre funktionaliteter, kan systemet nok ikke direkte bruges i en rigtigt virksomhed. Da systemet som udgangspunkt er lavet til Netto, vil det heller ikke kunne bruges i andre virksomheder før der bliver ændret en mindre del i koden.

Udover sms-notifikationsdelen, har produktet til dette projekt været gratis. Men hvis de afgrænsede elementer blev implementeret, og det samlede system blev gjort klar til distribution, ville der muligvis kunne tages betaling for det. De manglende dele, der skal laves før systemet kunne distribueres, ville dog umiddelbart ikke tage mere end et par måneder at færdiggøre.

Hvis systemet skulle distribueres er der to forskellige måder der skal overvejes. Vi kan enten vælge at distribuere det som et meget fleksibelt system der kan anvendes til mange forskellige formål og virksomheder. Ellers kunne vi vælge at udgive det som et specifikt produkt pr. virksomhed. Første mulighed ville ikke kræve videre programmering fra vores side efter, at systemet er udgivet, men et for fleksibelt system kunne skræmme virksomheder, da der muligvis skal indstilles for meget før det opfylder deres behov. Den anden mulighed ville løbende kræve programmering fra vores side, men til gengæld ville vi kunne levere et mere attraktivt produkt til virksomhederne. Vi ville så også kunne tillade os at tage højere betaling for produktet i sidst nævnte tilfælde.

Metodekritik

Vi har som før nævnt brugt interview som en af vores metoder til indsamling af oplysninger om vores målgruppe. Denne metode er god til at få specifikke indtryk fra en bestemt bruger. Mens et interview ikke kan bruges til at finde en generel tendens. Resultatet af interviewet opfyldte vores hypotese om mangler i Netto's eksisterende system til vagtplanlægning. Vi fik virkelig god respons fra interviewet med Netto-chefen, og det var først efter interviewet, at vi fik vores krav til systemet helt på plads. Vi mener, at vores interview var en succes i og med, at vi opnåede vores mål med den, nemlig at få vores krav på plads.

Gennem hele projektløbet har vi brugt "divide and conquer"², dvs. at vi har delt alle vores problemer op i mindre delproblemer. Dette har vi gjort for lettere at kunne tage fat på problemerne. Det har hjulpet os, når vi sad fast og ikke rigtig kunne få noget skrevet. Vi satte os ned og snakkede om, hvad problemet bestod af, derefter delte vi det op mellem os for til sidst at skrive det sammen. Problemet med "divide and conquer" kan være at få den røde tråd i gennem rapporten, når den bliver delt op i mange dele. Men vi prøver hele tiden at skrive det i en sammenhæng og tænke på, hvad der kommer før og efter det afsnit, vi skriver. Vi mener, at det var en succes at bruge "divide and conquer", da det som sagt har hjulpet os meget, og vi mener stadigvæk, at vi har kunnet holde en rød tråd gennem projektet.

²Del og hersk

Hvis vi havde 3 måneder mere...

Som vi har beskrevet i vores afgrænsning side 64, tillader tiden os ikke at tilføje flere funktioner til vores system. Hvis vi nu havde 3 måneder mere til at arbejde på vores projekt, ville vi tilføje de nævnte funktioner og muligheder, der er beskrevet i afgrænsningsafsnittet. Derudover ville vi arbejde lidt mere på vores design, for nogle af siderne i vores system er ikke så fulde. Informationer omkring oversteget arbejdstimer for den enkelte medarbejder skal også med i designet. Generelt skal der fremgå flere informationer omkring fejl og overtrædelser af overenskomsten.

En større udvidelse vi kunne tilføje systemet, er et lønningssystem som kan håndtere lønninger. Udvidelsen skulle være et modul, som kan slås til i det eksisterende system. Lønningssystemet skal holde styr på hvor mange penge, der er blevet brugt for den aktuelle dag, samt give en større oversigt over forventet lønforbrug og det aktuelle lønforbrug. En ekstra feature til dette udvidelsen kunne være at implementere en funktion, der giver administratoren muligheden for at sende en liste over arbejdstider til et lønkontor, som viderebehandler dataene. Hvis lønkontoret ikke har modtaget dataene ved en bestemt dato, sender deres system en meddelelse om det.

Vores notifikationer bruges i øjeblikket kun til at notificere medarbejderne, hvis deres vagt er blevet byttet. Her kunne der fx udvides til, at alle fik besked, når vagtplanen var lagt. De kan herefter vælge at logge på systemet og tjekke den nye vagtplan. Hvis administratoren nu ændrer i vagtplanen, så kan der udvides til, at de involveret får en notifikation om, at deres vagt er blevet ændret.

En krypteret forbindelse mellem server og klienten ville også være noget, vi ville tage op. Dette ville bl.a. medføre, at vigtige data ikke udenvidere kan opsnappes på netværket som simpelt tekst.

Til sidst, ville vi optimere vores kode. Vi kunne omskrive noget af vores kode, så den bliver mere fleksibel, og måske ville være mere anvendelig end i den er i øjeblikket.

Kapitel 5

Litteratur

- [1] Gestalt loven på wikipedia.
http://en.wikipedia.org/wiki/Gestalt_effect.
- [2] MySQL AB. MySQL 3.23, 4.0, 4.1 reference manual, december 2005.
<http://dev.mysql.com/doc/refman/4.1/en/index.html>.
- [3] The oracle FAQ, december 2005.
<http://www.orafaq.com/glossary/faqglosm.htm>.
- [4] Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Philip Olson, Georg Richter, Damien Seguy og Jakub Vrana. *PHP Manual*. the PHP Documentation Group, November 2005.
<http://www.php.net/docs.php>.
- [5] Refsnes Data. W3 schools, December 2005.
<http://www.w3schools.com>.
- [6] Det Centrale Personregister. Personnummeret i CPR-systemet. Marts 2003.
<http://www.cpr.dk/Index/dokumenter.asp?o=2&n=0&d=397&s=4>.
- [7] Tom Vilmer Paamands. CPR-generator.
<http://tom.paamand.dk/cpr.htm>.
- [8] HK/Handel. *Landsoverenskomst for Butikker 2004 Mellem Dansk Handel Og HK/Handel*. TrykBureauet Grafisk produktion A/S, 2730 Herlev, 2004.
- [9] Jim Giles. Internet encyclopaedias go head to head. December 2005.
<http://www.nature.com/news/2005/051212/full/438900a.html>.
- [10] HK. *Funktionærloven*. DK/Danmark, Marts 2005.

5.1 Kildekritik

Når der bruges information fundet i bøger og specielt på internettet, er det vigtigt, at man er kritisk overfor den information man får. Det nytter ikke at tro på alt, hvad man læser. Det er ofte nødvendigt at sammenligne med andre kilder, for at se om der er overensstemmelse. En kilde er ikke bedre end kildens forfatter.

Vi har bl.a. brugt Wikipedia som kilde til gestaltloven. Wikipedia er et online leksikon der er åben for alle. Her kan alle skrive og rette i andre folks indlæg. Det er vores erfaring, at Wikipedia er et godt sted at finde information. Denne erfaring bakkes bl.a. op af en nylig undersøgelse foretaget af det naturvidenskabelige magasin Nature[9]. Vi er dog også klar over, at vi ikke altid kan regne med det, der står der. Da vi ikke kender forfatteren, kan det være svært at vide, om vedkommende er troværdig. Der er mange, der bruger wikipedia, så der er også stor sandsynlighed for, at fejl hurtigt bliver rettet af andre, der læser der. Alt i alt mener vi, at det er en troværdig kilde til visse formål, da det der står stemmer overens med andre kilder om samme emne.

Vi har hovedsageligt anvendt hjemmesider som kilder til dette projekt. Sagligheden af hjemmesider kan være meget svingende i forhold til fx publicerede bøger. Men da vi til de fleste kilder, har haft mulighed for at tjekke informationerne op mod informationer på emnernes officielle hjemmesider, mener vi, at de valgte kilder er tilstrækkeligt troværdige.

Kapitel 6

Bilag

6.1 Bilag 1 - Interviewguide

Interviewguiden

Inden vi aftalte en tid med Netto's butikschef Martin B. Olsen, skulle vi forberede de spørgsmål, vi ville stille ham. Vi formulerede spørgsmålene således, at vi fik de oplysninger og informationer, vi skulle bruge til at lave et system til vagtplanlægning.

Spørgsmålene blev formuleret således:

- Regler omkring arbejdstider, ungarbejdere, lukkelov?
 - Hvem må lukke?
 - Arbejdstider?
 - Bytning
- Hvordan fungerer jeres nuværende system – hvad kunne gøres bedre?
 - Administration
 - Udmelding til ansatte
 - Vagtplansændringer
 - Brugervenlighed omkring administration
 - Manglende funktioner
 - Tilgang for de ansatte
- Hvem har tilgang til programmet?
- Eventuelle krav/nye ideer til et nyt system?
- Hvilke funktioner ville du stille til et autogenereringssystem?
 - Hurtigt?
 - Funktionelt
 - Valgmuligheder

Vores første spørgsmål, omhandlede reglerne i Netto, samt lidt lovgivning med hensyn til, hvad medarbejderne må og ikke må. Dette skal vi vide, for at finde ud af, hvilke love vi skal tage højde for, når vi skal lave vores system til vagtplanlægning. Her er det vigtigt at vide, hvilke ting de forskellige medarbejdere kan og må, og hvor længe de må arbejde på en dag eller en uge. Vi skal også vide, om medarbejderne må bytte vagter med hinanden, og hvordan dette foregår.

Vi ville også gerne vide, hvordan deres system virker, og hvad det kan. Derfor spurgte vi om, hvordan det fungerer, og hvad der evt. kunne gøres bedre. Når vi skal konstruere vores eget system, vil vi gerne vide, hvad et vagtplansprogram kan, og hvad der evt. er af mangler, så vi enten kan videreudvikle systemet eller lavet et nyt fra bunden.

Vi fik lov til at se systemet, og se hvordan det fungerede. Så kunne vi få en ide om, hvordan sådan et vagtplanslægningsystem kan se ud. Udover hvordan systemet fungere, og hvad der er godt og dårligt ved det og evt. manglende funktioner, ville vi gerne vide, hvem der havde tilgang til systemet. Det er vigtigt at vide, om det kun er butikschefen der har adgang til systemet, eller om andre også havde det.

Når en vagt er blevet byttet, eller når vagtplanen er færdig, vil vi gerne vide, hvordan dette bliver formidlet til medarbejderne. Brugervenligheden af systemet er også vigtig, da systemet ikke skal være for svært, at finde rundt i. Derfor vil vi spørge ham om, hvordan han synes systemet er at navigere rundt i.

Vi er meget interesseret i at vide, om Martin B. Olsen har nogle idéer eller krav til vores system, når vi skal igang med at lave det. Da vi har en ide om, at vores system skal have en autogenereringsfunktion, vil vi gerne vide, hvilke krav eller funktioner som Martin B. Olsen vil have. Vi vil også meget gerne vide, hvad han siger til idéen. Vi vil gerne vide, hvor hurtigt han synes, at autogenereringsfunktionen skal være, hvilke muligheder der skal være, og hvilke funktioner denne funktion skal have.

6.2 Bilag 2 - Lovgivning omkring arbejdstider

Når man lægger en vagtplan, skal der tages højde for, hvilke regler der gælder for arbejdstider, ferie og sygedage, og enkelte særregler, som fx regler for elever.

Vi arbejder ud fra fire stillingsbetegnelser: Fuldtidsansatte, elever, deltidsansatte og funktionærer. Udover stillingsbetegnelserne, tager vi også hensyn til, om medarbejderen er under 18 år.

Fuldtidsansatte

Fuldtidsansatte skal, ifølge Landsoverenskomst for butikker 2004 mellem Dansk Handel & Service og HK/Handel, arbejde 37 timer om ugen, eller 592 timer, i løbet af en vagtplan på 16 uger. Arbejdsplanen skal printes ud for hver medarbejder. Hvis den ansatte har mere end 45 timer i en enkelt uge, skal der udbetales overarbejds løn for disse ekstra timer. Fuldtidsansattes arbejdstid skal så vidt muligt placeres på fem af ugens dage. Det vil sige, at en fuldtidsansat så vidt muligt skal have en fast arbejdstid.

Deltidsansatte

Antal arbejdstider for deltidsansatte bestemmes i hvert tilfælde ved ansættelse. Dvs, at der ikke er nogle regler for, hvor mange timer en deltidsansat skal arbejde over de 16 uger, en vagtplan strækker sig over. Det antal timer, som er aftalt ved ansættelse, skal den deltidsansatte arbejde.

Funktionærer

Alle funktionærer, der arbejder mere end otte timer ugentligt i gennemsnit, er omfattet af funktionærloven[10]. Når en funktionær er på arbejde, er mindste arbejdstid fire timer. Funktionæren har en fleksibel arbejdsplan, som aftales mellem arbejdsgiveren og den ansatte[8].

Elever

Under elevens arbejdstid skal der være en oplæringsansvarlig til stede[8]. Elevens arbejdstid bør ikke overstige otte timer om dagen¹. Det ugentlige antal timer en elev arbejder, bør følge samme princip som arbejdstiderne for fuldtidsansatte.

Delkonklusion

Fuldtidsansatte skal, så vidt muligt arbejde 37 timer om ugen (592 timer på 16 uger). Timerne skal så vidt muligt være fordelt på alle ugens fem arbejdsdage.

Deltidsansattes- og funktionærers arbejdstid er efter aftale med butikschefen.

Elever bør være på arbejde i samme tidsrum, som en fuldtidsansat, der er i stand til at lære eleven op.

¹Se bilag om elever

