

DATAMATIKER - UCN AALBORG

---

# Webbaseret CRM-system

---

---

Peter L. Thomsen

---

Gregers Boye-Jacobsen

15. Januar 2012

# Indhold

<b>Indhold</b>	<b>1</b>
<b>1 Indledning</b>	<b>3</b>
1.1 Forord . . . . .	3
1.2 Problemformulering . . . . .	4
1.3 Virksomhedsbeskrivelse . . . . .	5
<b>2 Metode</b>	<b>6</b>
2.1 Udviklingsmetode . . . . .	6
2.1.1 Anbefalet metode valg . . . . .	7
2.1.2 Scrum . . . . .	8
2.1.3 Tests . . . . .	9
2.2 Teknologier . . . . .	10
2.2.1 MVC . . . . .	10
2.2.2 Database . . . . .	10
2.2.3 Frontend . . . . .	13
2.2.4 Versionstyring - SVN . . . . .	16
<b>3 Arkitektur</b>	<b>17</b>
3.1 Arkitektur . . . . .	17
<b>4 Teori</b>	<b>20</b>
4.1 Patterns . . . . .	20
4.1.1 Formål . . . . .	20
4.1.2 Forskellige patterns . . . . .	20
<b>5 Produkt</b>	<b>27</b>
5.1 Kravspecifikation . . . . .	27
5.2 Design . . . . .	30
5.2.1 Domænebeskrivelse . . . . .	30
5.2.2 Domænemodel . . . . .	31
5.2.3 Databasestruktur . . . . .	33
<b>6 Projektforløb</b>	<b>35</b>

6.1	Product Backlog . . . . .	35
6.2	Sprint 1 . . . . .	36
6.2.1	Sprint backlog . . . . .	36
6.2.2	Burndown Chart . . . . .	37
6.2.3	Udførelse af Sprint 1 . . . . .	37
6.2.4	Retrospective . . . . .	51
6.3	Sprint 2 . . . . .	51
6.3.1	Sprint Backlog . . . . .	51
6.3.2	Burndown Chart . . . . .	52
6.3.3	Udførelse af sprint 2 . . . . .	52
6.3.4	Retrospective . . . . .	60
6.4	Sprint 3 . . . . .	60
6.4.1	Sprint Backlog . . . . .	60
6.4.2	Burndown Chart . . . . .	61
6.4.3	Udførelse af sprint 3 . . . . .	61
6.4.4	Retrospective . . . . .	65
6.5	Sprint 4 . . . . .	65
6.5.1	Sprint Backlog . . . . .	65
6.5.2	Burndown Chart . . . . .	65
6.5.3	Udførelse af sprint 4 . . . . .	66
6.6	Accept test . . . . .	66
6.6.1	Filter Leads . . . . .	66
6.6.2	Upload CSV . . . . .	67
6.6.3	Find Lead Details og Kontakt Details . . . . .	67
<b>7</b>	<b>Sikkerhed</b>	<b>68</b>
<b>8</b>	<b>Perspektivering</b>	<b>70</b>
8.1	Roller og samarbejde . . . . .	70
8.2	Arbejdsmetode . . . . .	71
8.3	Produkt . . . . .	72
<b>9</b>	<b>Konklusion</b>	<b>73</b>
<b>A</b>	<b>Diagrammer</b>	<b>I</b>
<b>Litteratur</b>		<b>II</b>

# Kapitel 1

## Indledning

### 1.1 Forord

Allerede på 1. semester stiftede vi bekendtskab med vores første patterns, først var det et søgepattern, men senere lærte vi også til singleton patternet. Samtidig blev vi undervist i brugen af Model-View-Controller (også kendt som 3-lags arkitektur) mønsteret, der var et arkitekturpattern.

Der er altså tale om to slags patterns: Arkitektur-patterns og design-patterns. Arkitektur-patterns er brede løsninger, der dikterer opbygningen af hele programmet. Her kan f.eks. nævnes MVC der opdeler programmet i tre dele; ét der tager sig af domæneobjekterne i programmet (Model), hvordan domæneobjekterne arbejder sammen (Controller) og endelig hvordan programmet præsenteres (View).

**Model,**  
**View,**  
**Controller**

På den anden side finde Design-patterns, der er mere specifikke dele af koden, der via forskellige konstruktioner, løser en række programmeringsmæssige problemer.

Under vores praktiktid på EqualSums blev vi stillet over for en opgave, hvor det var nødvendigt at bruge flere patterns, nogen kendte (singleton og observer) og også nogle, for os, ukendte (f.eks. Repository-pattern [9]). Da vi blev stillet overfor opgaven til vores hovedopgave indså vi, at vi blev nødt til at sætte os ind i flere patterns. Dette ledte frem til en interesse for patterns generelt.

Larman beskriver patterns med følgende ord:

”Experienced OO developers (and other software developers) build up a repertoire of both general principles and idiomatic solutions that guide them in the creation of software.”

[5, p.278]

Med andre ord, over tid samler udviklere en række generelle løsninger der med ganske lette ændringer passer på mange forskellige problemstillinger - det er disse løsninger der er kendt som patterns.

Vi tænkte derfor, at det kunne være interessant at se nærmere på patterns og hvordan de kan bruges i et stykke software. I denne rapport vil vi dels dokumentere vores arbejde med at udvikle et webbaseret CRM-system til EqualSums. Samtidig vil vi, ved at tage udgangspunkt i dette software, kigge nærmere på de patterns vi bruger, og se på patterns generelt. Vores håb er, derved at få en større forståelse for patterns, og hvordan de kan benyttes som "byggeklodser" i et stykke software.

Customer  
Relationship  
Management

Vi har fra EqualSums' side fået til opgave at lave en webbaseret løsning, der kan håndtere kontakter fra før første kontakt, over mulige kunder til de er etablerede kunder. Der skal holdes styr på kontaktoplysninger, hvad man har foretaget sig af opkald, mails og møder og endelig skal man kunne planlægge aftaler med kunden. Kort sagt skal der konstrueres et CRM-system.

Samtidig skal produktet udvikles, så det er nemt at arbejde videre på for et andet team. Vi har indset, at vi på knap 10 uger ikke kan levere et kompetent CRM-system. Derfor vil vi fokusere på at udvikle i moduler der er prioriteret efter kundens (EqualSums') ønsker.

Rapporten vil blive delt op i 3 dele, første del, hvor vi beskriver vores arbejds-metode. Andet afsnit, teori-afsnittet, hvor vi vil gå i dybden med forskellige patterns, hvad de kan, hvad de ikke kan, og hvorfor vi har valgt at bruge dem. til sidst vil vi beskrive og dokumentere processen fra kravsspecifikation til færdigt produkt

Det hele vil blive rundet af med en konklusion, hvor vi opsummerer rapportens indhold, og de svar vi er kommet frem til gennem processen.

## 1.2 Problemformulering

I vores hovedopgave er den overordnede problemstilling:

„At udvikle et webbaseret CRM-system der benytter patterns, og som er let at udvikle for tilkommende udviklere.“

Det er klart, at det er et meget bredt problem, der skal afgrænses. Vi vil fokusere på patterns som det er relevant at anvende til den stillede opgave, gennemgå dem, forklare hvordan de virker generelt, herunder hvilke problemer de løser og endeligt hvordan vi benytter dem.

Når den endelige kravsspecifikation er på plads, vil vi få virksomheden til at prioritere user-storiesn'e. Herefter vil vi lave estimer. Ud fra disse estimer vil vi vurdere hvilke user-stories det er realistisk at implementere i løbet af 4 sprints á to ugers varighed.

## 1.3 Virksomhedsbeskrivelse

EqualSums er en relativt lille virksomhed der blev stiftet af Mikkel Elliott og Martin Skov Kristensen. Virksomheden beskæftiger sig med udvikling af online softwareløsninger til bogholderi og bogføring. Udover Mikkel og Martin er der en fast udviklerstab på 4 backend-udviklere og én frontendudvikler. Udover den faste stab er der pt. en konsulent, tre praktikanter og en studentermedarbejder. Udviklerstaben styres af en projektleder, der sørger for den interne koordination i udviklergruppen.

Af øvrige ansatte er der tre sælgere hvoraf en også fungerer som administrativ medarbejder.

Ledelsen fremmer bevidst en uhøjtidelig tone i virksomheden, hvilket bl.a. ses ved, at hele virksomheden spiser frokost sammen, og oftest går en kortere tur herefter. Medarbejderne har også sækkestole og boksebold til rådighed, ligesom der også er kaffe ad libitum. Herudover er man til enhver tid velkommen til at medbringe kage.

Der er mere eller mindre frie mødetider, så længe man opnår 37 timer ugentligt, dette giver medarbejderne stor frihed til selv at vælge deres mødetider.

Den oprindelige idé var, at virksomheden skulle tilbyde en hel vifte af softwareløsninger. Da man for alvor kom i gang med den første løsning, blev det dog vurderet, at dette produkt i sig selv kunne holde hele virksomheden beskæftiget. Derfor skiftede fokus til dette enkelte produkt, der idag er det EqualSums udvikler.

I salgsafdelingen bliver der i høj grad brugt Googles services til at holde styr på kontakter, kunder og dokumenter. Der bliver benyttet Google docs til dokumenter, kunder ligger i Google contacts, der automatisk synkroniseres med sælgernes mobiltelefoner, og i et regneark bliver der holdt styr på leads. Problemet er, at der ikke er noget der så at sige, binder alle disse steder sammen. Sælgerne bruger en stor del af dagen på at finde ud af hvem der skal kontaktes, opdatere oplysninger osv. De savner kort sagt et system der kan fungere som fælles samlingspunkt for alle disse services.

# Kapitel 2

## Metode

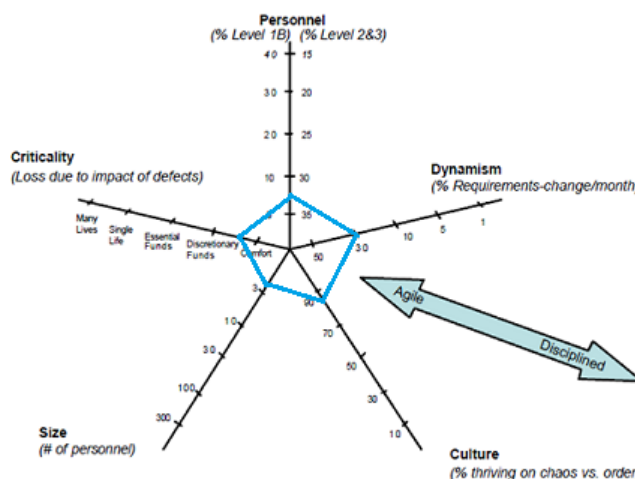
### 2.1 Udviklingsmetode

Her vil vi beskrive den udviklingsmetode vi har valgt at bruge til vores projekt og hvorfor vi har fravalgt andre. Som det første har vi undersøgt hvilken metode der vil passe bedst til vores projekt, til det har vi gjort brug af Berry Boehm's stjernemodel (fig 2.1). Modellen bruges til at vurdere om man bør bruge en agil eller plandreven metode, eller om man skal finde en middelvej. Det gør den ud fra 5 faktorer som man skal svare på i forhold til det projekt man skal til at starte på. Des tættere midten af diagrammet man sætter sine krydser, des mere en agil metode har man brug for. Det samme gør sig gældende den anden vej, hvor, jo længere man kommer ud af akserne, jo mere bør man gøre brug af en mere plandreven metode.

**Personnel:** På denne akse vises hvor erfarent et udviklerteam er. Da ingen i gruppen har nogen særlig stor erhvervsmæssig erfaring på trods af, at vi har lidt kendskab til teknologierne fra skolen, er vores vurdering at den vil ligge mellem 0-35 og 10-30. Hvilket peger hen imod en agil metode.

**Dynamism:** Her bestemmes hvor dynamisk man har behov for at være. Efter de første par møder med EqualSums blev det hurtigt klart for os at der ville komme mange ændringer undervejs. Mange kunder har svært ved at sætte ord på hvad de egentlig gerne vil have til at starte med. Derudover er der flere forskellige vi snakker med som står for hvert deres område. Det betyder vi ligger omkring de 30 som peger på en rimelig agil metode.

**Culture:** Her ser man på hvordan teamet trives bedst. Skal der være regler og procedure for alle, eller har man mere frihed til selv at vælge. Vores team har det godt med et forholdsvist frit arbejdsmiljø. Når vi ser på EqualSums hvor vi laver programmet er der også meget stor frihed og medarbejderne i



Figur 2.1: Stjernemodellen

virksomheden trives herunder. Derfor er det kun naturligt for os at ligge os tæt på de 90 som igen peger på en agil metode.

**Size:** Her kigger man på hvor stort ens team er. Da vi arbejder i en lille to mandsgruppe er den nem at sætte på. Hvilket igen vil sige det ligger sig op af en agil metode. Store grupper kan blandt andet have svært ved at kommunikere og koordinere et stort projekt hvor imod et lille hold nemmere kan styre hvem der laver hvad.

**Criticality:** Criticality afgøres af hvor store konsekvenser fejl i systemet vil have. I vores system kommer fejl ikke til at have den store betydning. De kommer hverken til at koste menneskeliv eller vanvittige økonomiske nedture for firmaet. Derfor lægger vi os ind mod midten igen og dermed en agil metode.

### 2.1.1 Anbefalet metode valg

Ud fra vores analyse i foregående kapitel ved hjælp af stjernemodellen, er det tydeligt at se vi ligger os op af en agil metode. Som metode har vi valgt at bruge Scrum. Vi har allerede erfaring med Scrum fra vores andre projekter samt fra vores praktikophold. Vi syntes derfor scrum er et oplagt metodevalg til vores projekt. Der vil dog i projektet også blive benyttet elementer fra andre metoder, da agil udvikling netop handler om at tilpasse arbejdsformen, alt efter behov. Vi har derfor valgt at tage nogle rutiner med fra nogen af de andre metoder. Fra UP har vi valgt at gøre brug af domænemodellen, samt vores databasediagram. Dette giver et godt fundament, især som mindre erfarne. Derudover har vi også lånt værktøjerne, refactoring og testing som de primære fra XP, men også pair programmering, fælles kode og kodestandarder vil blive brugt.



En af grundene til vi ikke har valgt at arbejde ud fra en UP metode, er blandt andet stjernemodellen, men i høj grad også vores korte projektperiode. Da vi har et meget begrænset tidsrum til at få lavet programmet, er der ganske enkelt ikke tid til at analysere så meget. Derudover ligger det også klart, at der kommer mange udvidelser til systemet da vi igen på grund af den begrænsede tid er blevet nødsaget til at afgrænse problemområdet. Det vil sige, at der er mange opgaver vi slet ikke har med i dette oplæg, som skal laves bagefter.

### 2.1.2 Scrum

Scrum er en agil udviklingsmetode skabt tilbage i 90'eren. Det er et interaktivt og inkrementalt framework som hurtigt er blevet meget populært blandt udviklere, da det giver en stor frihed, men samtidig sætter nogle fast rammer for arbejdsgangen.

#### Roller

I Scrum vil man støde på rollerne Scrum Master, Produkt Owner og Scrum Team. Vores Produkt Owner er Martin Skov Kristensen og er salgschef hos Equalsums. Det er ham der står for projektet sammen med os, så det var naturligt at give ham den rolle, som blandt andet indebærer at prioritere user stories og stå til rådighed når der er spørgsmål omkring produktet. Som Scrum master har vi valgt Peter. Valget faldt på Peter fordi han havde erfaring med programmet ScrumDesk som vi bruger til at styre vores backlog, sprints, osv. Gregers er en del af selve Scrum teamet som udvikler. Peter indgår selvfølgelig også som en del af teamet og udvikler sammen med Gregers.

#### Strukturen

Vi arbejder med 14 dags(10 arbejdsdage) sprints hvor vi har  $2\frac{1}{2}$  arbejdsdag til udvikling på systemet. Grunden til vi ikke har mere er, at der skal være tid til rapportskrivning. Vi har en velocity på 0,7 hvilket betyder vi har 53,2 effektive timer til hvert sprint.

Da vi altid sidder ude i virksomheden når vi udvikler på produktet, indgår vi også i deres Scrum holds daglige Scrum møder, og hvis der ellers er nogen fælles møder. Internt i vi vores projekt holder vi også et dagligt Scrum møde, hvor vi kort fortæller hvad vi har lavet og hvad vi har tænkt os at gå i gang med næste gang. Derudover holder vi også et sprint review møde med Martin (produktowneren) hvor vi viser det frem vi har lavet i sprintet. Det gør vi for at sikre at det vi har lavet lever op til Martins forventninger. Scrum retrospective holder vi løbende dels fordi vi kun er to på holdet og hele tiden snakker sammen under forløbet, og dels fordi man har en tendens til at glemme hvad der gik dårlig og især hvad gik godt.

### 2.1.3 Tests

For at sikre en høj kvalitet i vores program har vi valgt at gøre brug af Unit Test, Integrationstest og til sidst Acceptance Test. Udover at være med til at højne kvaliteten i koden, sikrer disse tests også, at koden virker, og rent faktisk gør det den skal gøre. Der drages også fordel ved at bruge de to først nævnte test når der laves refactoring. Når der er blevet refactored i koden er det nemt og hurtigt at teste om den stadig virker som den skal, ved at køre disse tests. Udover at hjælpe udviklerne, giver det også ejerne et gennemtestet produkt, som vil være i højere kvalitet.

#### Unittests

Der er valgt at unitteste enkelte metoder i systemet fremfor at bruge det som en test-dreven proces. Dette gøres, fordi den nødvendige erfaring med at køre et test-drevet projekt mangler, og derfor vil det blive for stor en tidsrøver.

Da der er gjort meget ud af at designe MVC arkitekturen så systemet udviklet i dette er nemme at teste på, har vi rig mulighed for at teste store dele af vores system. [6]

I systemet testes blandt andet om der modtages de rigtige Views, og om den medsendte ViewModel er korrekt. En ViewModel er data man sender til sit View, ofte vil det være en klasse, eller en IEnumerable. IEnumerable er et Interface der gør det muligt at bruge et for-each loop på vores collection af objekter.

I forbindelse med at teste metoder i vores controller lag, er der implementeret et Repository Pattern, til at hjælpe med at isolere Unit testene. Hvis man bruger den database man sidder og udvikler på kan man ikke længere stole på disse resultater. Dette skyldes, at indholdet af databasen kan blive ændret i forbindelse med tests hvor der skrives til databasen. Derfor mocker man de data så de altid er de samme, så forudsætningen for testen altid er den samme. Repository Pattern og hvordan vi præcist bruger unit test i den forbindelse kan der læses mere om i et senere afsnit.

Når der er blevet refactored en eller flere metoder er det hurtigt og nemt at teste om de forskellige Views og ViewModels stadig er rigtige. Derudover testes metoderne selvfølgelig også, så man er sikker på de stadig gør det de skal, selvom de eventuelt er blevet skrevet om. Det sikrer os imod der pludselig er dele af systemet der ikke længere virker.

#### Integrationstests

Udover unittests er der også valgt at lave Integrationstests. Her testes flere dele af systemet sammen, for at sikre sig at de forskellige lag snakker ordenligt sammen.

I systemet er der lavet integration tests på nogle User Stories, hvor der testes i GUI og hele vejen ned til DB-laget. På den måde bliver alle lagene testet fra top til bund. Her er det igen nemt at teste om en hel userstory stadig virker, hvis der for eksempel skulle komme ændringer fra Produkt Owneren, så man bliver nødt til at refactor en eller flere af metoderne.

### Acceptancetests

Sidst vil der blive lavet en Acceptancetest sammen med Produkt Owner, og eventuelt nogen af brugerne på systemet. Det gøres fordi det er kunden der bedst ved hvordan systemet skal virke og derfor er det en kæmpe hjælp af få dem til at teste systemet. På den måde er man sikker på, at det lever op til deres forventninger.

## 2.2 Teknologier

Når der skal udvikles web-applikationer, er der utroligt mange valg at træffe rent teknologimæssigt. Der skal træffes valg for både backend, frontend, database og alt derimellem. Hvis der skal være fancy animationer skal der så bruges html5 eller flash - eller silverlight? Skal backenden være php eller asp.net? Skal databasen være en postgresql, mssql eller mysql-database? Hvordan snakker backend og frontend sammen? xml, json eller noget helt tredje? Der er mange muligheder og i det følgende kapitel bliver de valg vi har truffet forklaret og begrundet.

### 2.2.1 MVC

Overordnet set har vi valgt at arbejde med *Microsofts ASP.net mvc3 framework*. Dette giver en logisk opbygning i projektet, der bliver delt op i *Model*, *View* og *Controller*. Udover, at det holder sig tæt op af den 3-lags arkitektur som vi har brugt i uddannelsen, giver det også øget overblik. Samtidig giver det mulighed for at bruge *Microsofts Razor View Engine* [4] (Herefter blot Razor) der i forhold til tidligere løsninger som f.eks. web forms giver øget læsbarhed og enklere syntaks.

### 2.2.2 Database

#### EntityFramework

I Equalsums bruges Devarts entity framework (Herefter EF) til at forestå kommunikation mellem database og backend. Da EF ud fra databasen selv konstruerer modeller er det en væsentlig forenkling i forhold til at bruge sql-sætninger.

I al sin enkelthed fortæller man EF, at man f.eks. gerne vil have fat i en kunde-objekt fra databasen, og herefter vil EF selv finde ud af, hvilke relationer der findes i databasen, og sørge for at hente al relevant data ud. Det samme gør sig gældende ved indsætning, hvor man laver et objekt med de ønskede attributter, og herefter fortæller EF at det skal gemmes i databasen, hvorefter de enkelte værdier bliver indsat i de forskellige tabeller med de rette relationer. Med andre ord, EF sørger selv for at foretage mapning mellem databaserelationer, og derudfra dannes et sæt af objekter. På denne måde undgår man, at skulle lave lange, komplicerede join-sætninger. Såfremt fremmednøgler og primærnøgler er korrekt definerede, kan alt dette overlades til EF.

## LINQ

LINQ er som sådan ikke specifikt til brug for databaser, men da det er til at lave vores database-queries vi bruger LINQ, vil det blive nævnt i database-afsnittet.

Language  
INtegrated  
Query

LINQ udmærker sig ved at lave sql-lignende kald. Forskellen er, at man kan bruge LINQ på andre datakilder end databaser, f.eks. arrays. LINQ gør brug af *anonyme typer* og *lambda-expressions* - disse er typer og funktioner der oprettes ad hoc, det vil sige, at de kun eksisterer så længe de bruges, og man undgår derved, at oprette en mængde klasser og metoder. I det følgende eksempel vises et linq-udtryk, hvor man i en graf vil have alle ikke-besøgte noder:

```
1 var unvisitedNodeList = (select from n in nodeList
2                             where n.visited == false
3                             select n);
```

Figur 2.2: LINQ-kald der henter en liste af nodes

En forespørgsel af denne type, ville resultere i en IEnumerable der indeholdt node-objekter (hvor egenskaben visited er falsk).

Man kan også lave sine egne objekter, hvis man ønsker at hente specielle objekter. Dette vil blive yderligere uddybet på side 14 under "JSON".

## Open Authorization (oAuth)

Open Authorization eller oAuth, er en standard for autorisering af f.eks. webiders brug af brugerkonti, uden at brugeren skal udlevere brugernavn og password. oAuth giver ganske enkelt en applikation adgang til en brugers konto uden at applikationen kender brugernavn og password.

Før denne forklaring, vil et par termer blive præciseret:

**Bruger** Brugeren er ejeren af kontoen der skal tilgås, brugeren kender sit brugernavn og password.

**Klient** Klienten er det program eller den side der skal tilgå brugerens konto. Klienten må ikke kende brugerens password.

**Server** Serveren er den entitet der holder styr på brugerens konto, og hvem der har adgang til den, serveren ved hvornår brugerens password er rigtigt<sup>1</sup>.

Grundlæggende findes der to former for OAuth: 2-legged og 3-legged. Disse to former vil ganske kort blive forklaret, hvorefter det valg der er foretaget i forbindelse med dette projekt vil blive begrundet.

**3-legged Authorization** Klienten sender en forespørgsel til serveren og modtager en usigneret access token, samtidig sendes brugeren videre til Serveren. Brugeren identificerer sig overfor serveren, hvorefter klienten modtager en signeret token, som er gældende for den specifikke session. Denne token sendes til klienten. Klienten kan nu bruge denne token til at signere fremtidige requests, og derved identificere sig over for serveren, som godkendt. Der er altså tre parter i denne transaktion, bruger, klient og server [2].

**2-legged Authorization** 2-legged authentication bruges hvor brugeren ikke kan identificere sig over for serveren ved starten af hver session. Det kan f.eks. være ved brug af applikationer på en mobil enhed. Istedet opretter brugeren en hemmelig nøgle, for hver klient. Klienten kender denne hemmelige nøgle og bruger den til at danne en access token, som bruges til at identificere sig som en af brugerens godkendte klienter. I denne transaktion er der kun to parter, klient og server.

Klienten får adgang uden, at brugeren fortæller klienten sit password. Samtidig kan brugeren sætte regler for hvilke rettigheder den enkelte klient har. Dette kendes især fra Facebook, hvor applikationer skal bede om specifikke tilladelser til f.eks. at skrive på brugerens væg, læse vennelisten osv.

Vi vil i projektet benytte 2-legged authentication til at tilgå Googles api på vegne af brugeren. Vi kunne have valgt at bruge 3-legged, da denne er velegnet til brug ved websider, men da man skal identificere sig hver gang man starter en ny session (har haft browseren lukket), blev det vurderet, at det ville blive et irritationsmoment. Samtidig bruger EqualSums Google apps, hvilket muliggør 2-legged authentication. Vi valgte derfor 2-legged authentication, da det for brugeren tilbyder en langt smidigere login-oplevelse.

## Google api

Google tilbyder api til stort set alle services, hvadenten det er maps, docs, calendar, mail eller noget helt andet. Projektet skal autentificere sig via 2-step authentication, og heldigvis har Google lavet et .net library der tilbyder både 2- og 3-step authentication.

---

<sup>1</sup>Det er dårlig skik at opbevare brugerens password i klar-tekst, et hash af brugerens password er at foretrække

### 2.2.3 Frontend

Det har altid været forbundet med stort besvær at udvikle hjemmesider der ser ens ud, og opfører sig ens i alle browsere. Dette skyldes at der findes en lang række standarder, og at alle browsere understøtter disse standarder i større eller mindre grad. Det har derfor været nødvendigt at benytte sig af workarounds og hacks, der udnytter de forskellige browseres svagheder og styrker.

I vores arbejde slipper vi for denne problemstilling, da systemet der udvikles, kun skal bruges internt. Eftersom EqualSums udelukkende bruger Google Chrome, er det kun nødvendigt at udvikle til en enkelt browser. I det efterfølgende vil brugen af forskellige front-end teknologier blive forklaret.

#### Razor View Engine

Som tidligere nævnt bruger vi Microsofts Razor View Engine, der er en ny view engine til asp.net. Fordelen ved at bruge Razor er, at der er en kraftigt forenklet syntaks. Samtidig er Razor smart nok til, selv at vide, hvornår der er tale om et @ der starter en blok, og hvornår det er @ i en e-mail adresse. Dette betyder også, at man ikke behøver markere når en razor-blok afsluttes, til forskel fra asp hvor man bruger %> og php hvor man bruger ?>. Når man starter en Razor kodeblok skriver man ganske enkelt @ efterfulgt af koden. Det er her vigtigt at huske, at der ikke skal semikolon efter blokken.

Dette øger læsbarheden af koden, og hastigheden hvormed man udvikler, da det er langt mere intuitivt.

#### jQuery

jQuery er et framework til javascript der virker i alle moderne browsere, og man skal som udvikler ikke bekymre sig om browser kompatibilitet. Grundideen bag jQuery er "Write less, do more". Som eksempel kan nævnes, at man har mulighed for at bruge css-selectorer (og pseudoselectorer) til at vælge elementer. Man slipper derfor for at bruge document.getElementById og andre mærkelige konstruktioner, da man kan gribe direkte ned i DOM-træet<sup>2</sup>. Man kan f.eks. vælge den første række i tabellen med id "customers" ved at skrive

```
1 $("#customers:first-child")
```

Document  
Object  
Model

Udover det er meget nemmer at vælge et bestemt element, er det også nemt at manipulere elementerne uanset, om man vil ændre, tilføje eller fjerne indhold, tekst, css-egenskaber eller værdier.

Denne fleksibilitet sammen med en lang række indbyggede funktioner til animationer gør jQuery til et effektivt redskab, når der skal bruges Javascript på

<sup>2</sup>Den grundlæggende opbygning af et html-dokument

en side. Desuden gør jQuerys popularitet, at der findes en lang række plugins, som tilføjer næsten enhver funktionalitet man kan tænke sig.

## AJAX

I forbindelse med kommunikation mellem front-end og backend vil vi i høj grad benytte AJAX, som muliggør asynkron kommunikation mellem klient og server. Dette betyder, at der kan hentes og sendes data uden, at siden skal refreshes. På den måde får brugeren en langt bedre og flydende oplevelse, end hvis der skulle refreshes hver gang klienten skulle sende eller forespørge data fra serveren. Dette er med til at højne brugervenligheden.

Asynchronous  
Javascript  
And  
XML

Et sted hvor man bruger AJAX, er ved autofuldførelse, som f.eks. kendt fra Google-søgninger. Har man en debugger der overvåger serverrequests, som f.eks. Firebug mens man skriver sin søgning, vil man se, at der bliver sendt en request for hvert tastetryk. Dette er AJAX-requests der bliver sendt.

Netop brugen af jQuery gør det langt nemmere at benytte sig af asynkrone kald, idet der i jQuery findes en indbygget funktion til at sende asynkrone forespørgsler til serveren, der forenkler processen, som det ses i figur 2.3

```
1 var data = {userId: 4}
2 var url = {"CRMsystem/feedDetails/getUserName"}
3
4 $.getJSON(url, data, function(data){
5   $("#customerName").text(data);
6 })
```

Figur 2.3: Asynkront jQuery-kald

## JSON

JavaScript  
Object  
Notation

Vi har dog valgt at benytte JSON istedet for XML som dataformat til AJAX-requests. Først og fremmest skyldes dette, at JSON er lavet specielt til JavaScript, men samtidig er der langt mindre data der skal sendes frem og tilbage, da data ikke skal markeres på samme måde som XML hvorved der er langt mindre overhead:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persons>
3   <person>
4     <firstName>Thomas</firstName>
5     <lastName>Hansen</lastName>
6     <address>Guldvej 23</address>
7     <city>Aalborg</city>
8     <zipCode>9000</zipCode>
9     <email>thomas@gmail.com</email>
10  </person>
11 </persons>
```

Figur 2.4: XML-eksempel

```
1 person[
2   {
3     firstName: "Thomas",
4     lastName: "Hansen",
5     address: "Guldvej 23",
6     city: "Aalborg",
7     zipCode: 9000,
8     email: "thomas@gmail.com"
9   }
10 ]
```

Figur 2.5: JSON-eksempel

Som det ses af figur 2.4 og 2.5 er JSON-objekter langt enklere end XML-objekter, i det viste eksempel er der tale om et array (omgivet af []) med et enkelt objekt. Et objekt er omkranset af { og }.

At JSON er simpelt, er både en fordel og en ulempe. Når der skal sendes data fra serveren til klienten, kan man ikke bare tage et objekt der er hentet fra databasen og serialisere det som JSON. Der opstår problemer i forbindelse med cirkulære referencer. Dette skyldes en begrænsning i EF som Microsoft er opmærksomme på [8]. Indtil der foreligger en løsning, skal der benyttes en workaround, hvor man laver sine egne, speciallavede objekter via projection selector'en *select* [11, pp. 75-80]. Eksemplet i figur 2.6 tager udgangspunkt i fig. 2.2:

```
1 var unvisitedNoteList = (select from n in nodeList
2                             where n.visited == false
3                             select new {
4                                 name = n.name,
5                                 value = n.value,
6                                 visited = visited
7                             });
```

Figur 2.6: LINQ-kald der henter en liste af nodes, der er klar til JSON



Med dette kald får man et objekt der indeholder egenskaberne name, value og visited. Denne brug af projektion-keyword'et select betyder, at de anonyme objekter serialiseres bagefter.

### CSS3

Brugen af CSS3 giver en lang række muligheder for at lave langt mere elegant design, her kan bl.a. nævnes brug af gradienter og afrundede hjørner. CSS3 er endnu ikke understøttet af alle browsere, og ved visse egenskaber (som f.eks. border-radius), skal der benyttes en egenskab for hver af de 3 browser-engines (Webkit, Mozilla og IE). Som tidligere nævnt bliver der ikke problemer med cross-browser kompatibilitet.

#### 2.2.4 Versionstyring - SVN

Til versionsstyring har benyttet SubVersion (SVN). Det er et open source system som også benyttes hos EqualSums, så det var meget naturligt at gøre brug af det også. Det er et utroligt vigtigt redskab til at styre sin sourcekode.

Hos EqualSums har vi en server hvor vores repository ligger med koden på. Man kan så lave et checkout med TortoiseSVN, hvilket betyder man får en lokal kopi. Når man har lavet nogen ændringer i koden, kan man commit det igen til sit repository, hvor det så vil blive merged med de eksisterende filer. På den måde har alle altid adgang til den nyeste kode. Skulle der bliver uploadet noget kode som man fortryder kan man lave en roll back til en tidligere revision.

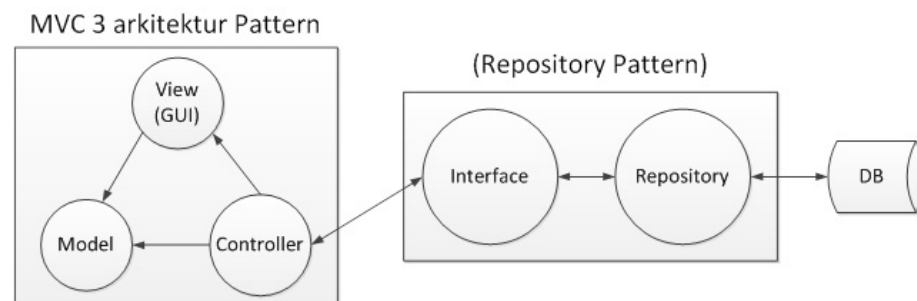
# Kapitel 3

## Arkitektur

For at hjælpe med at give et bedre overblik over systemet og hvordan de forskellige lag snakker sammen i forhold til hinanden, vil arkituren blive beskrevet her.

### 3.1 Arkitektur

Da vi bruger MVC 3 har vi en stor del af vores grundstruktur. MVC som står for Model, View, Controll er også delt op i netop disse tre lag.



Figur 3.1: Arkitektur

Som det kan ses på figur 3.1 er MVC kun en del af arkituren. Der er implementeret et Repository pattern, for at opnå en række fordele, som vil blive beskrevet i et senere afsnit. MVC kontrolleren står for at skrive til de klasser der er implementeret af repository pattern. Controlleren vil sende et objekt ned til sit repository, som skriver det til databasen. Det skal ses som et data access lag, det er her alt kontakt med databasen starter og stopper. Vores repositories

vil hedde `CompanyRepository.cs` mens interfacet til denne vil hedde `ICompanyRepository.cs`. Navnet fortæller selvfølgelig noget omkring hvilket område dette repository bruges.

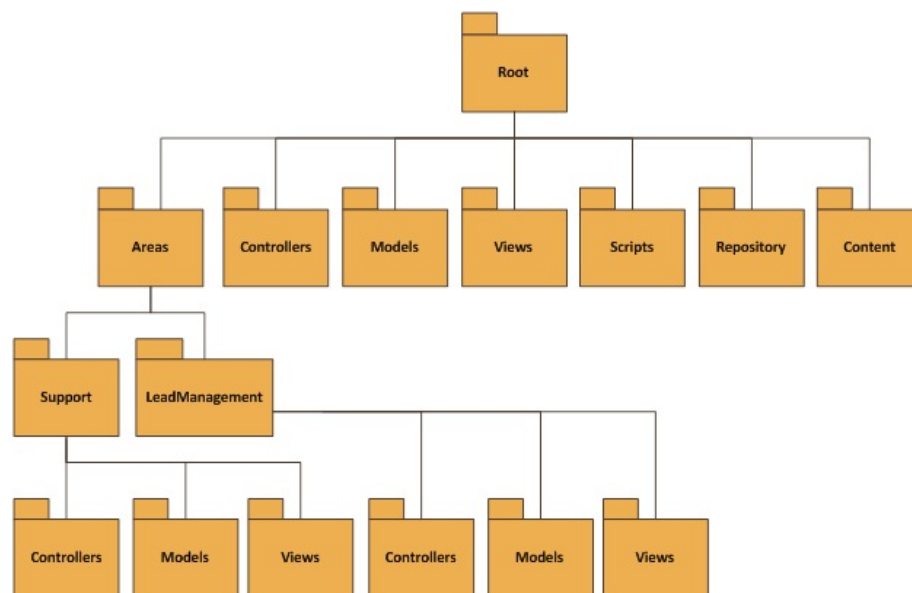
På figur 3.1 ses cirkelen kaldet Model. Det er vores klassiske modellag som man også kender det fra 3-lags arkitekturen. Det er i dette lag domænemodellen bliver realiseret. Da der bruges Entity Framework styrer dette framework laget for os, og det vil derfor virke tomt. Der benyttes Database First tilgangen til at modellere laget ud fra designet af databasen. Derved bruges Entity Frameworket det meste af tiden til at styre dette lag. Vi laver dog nogle modelklasser manuelt så som wrapper-klasser, eller ViewModels. Disse kunne blandt andet bruges hvis man vil sende to entity klasser til samme view. Eller har brug for nogle flere properties, som man ikke er interesseret i bliver gemt i databasen. Der vil komme eksempler på disse senere i rapporten.

View på figur 3.1, er det grafiske interface, som står for at vise den HTML brugeren skal se. Et View benyttes ofte sammen med en model. Det vil sige et objekt med en række properties som man ønsker vist i viewet. Det kunne være et Company objekt der bliver sendt som model, hvor efter viewet har adgang til dens properties. Her er også mulighed for at bruge IEnumerables hvis det er lister man vil skrive ud. Hvis man har en web-form på siden er det muligt at sende samme type objekt tilbage til controlleren, med alle de nye properties man har sat i web-formen. Et view vil ligge en mappe af samme navn som den controller den høre til. De vil typisk have et navn der forklare indholdet af siden, f.eks. `Company.cshtml`.

Normalt er det meningen, at view- og modellaget er skarpt adskilt af controllerlaget, så model og view aldrig kommunikerer direkte. Frameworket er dog skruet sådan sammen, at controlleren sender et model-objekt direkte til viewet.

Controller på figur 3.1 er vores controllerlag. Controlleren styrer hvad der sker i applikationen, og står for at få vist de rigtige views. Den vil minde meget om det controllerlag man også ser i 3-lags arkitekturen. Det er her vi modtager input fra brugeren og opretter objekter som vi sender videre ned til vores data access lag. (Repository pattern). Alle controllere hedder Controller til sidst. Altså en controller til at styre sine companies ville oplagt hedde `CompanyController.cs`. Dette er bestemt af MVC 3.

Selve mappe- og filstrukturen bliver selvfølgelig i høj grad bestemt af MVC, men for at gøre overblikket af systemet totalt vil vi kort beskrive strukturen her.



Figur 3.2: Mappestruktur

I roden af projektet vil man finde nogle config filer med forskellige opsætninger af sitet. Blandt andet SQL forbindelse. Der udover bruges Content til alle CSS filerne, Scripts til JavaScript og jQuery frameworket. I Repository ligger repository klasserne samt deres interface. Controllers, Models og Views er MVCs struktur, og burde være selvforklarende. Mere interessant er Areas [3, pp. 374-381] som er MVCs måde at lave sub-directories på. Hver mappe i et Area får nemlig hver sin Controllers, Models og Views mappe som virker på samme måde som dem i Root mappen. Det gør det nemt at flytte en del af systemet ud i et sub-directory, da alt vil opføre sig på samme måde såfremt det ligger i samme sub-directory.

# Kapitel 4

## Teori

### 4.1 Patterns

Som nævnt i forordet på s. 3, kan patterns beskrives som generelle løsninger, der har vist sig at virke på mange problemer. Patterns er altså en form for ”altmuligværktøjer” som en programmør kan benytte sig af.

#### 4.1.1 Formål

Formålet, eller ideen med patterns er, at det er løsninger, der kan bruges i mange tilfælde. På denne måde undgår programmører at skulle opfinde den dybe tallerken igen og igen. Samtidig er patterns den bevist bedste måde at løse problemerne på, så man undgår fejl, såfremt patterns’ne implementeres korrekt.

I det følgende vil vi forevise en række patterns, som vi har brugt i arbejdet med CRM-systemet. Vi vil gennemgå hvilke former for problemer de løser, hvordan de implementeres, og hvordan og hvorfor vi har benyttet os af dem. Vi har dog valgt at se bort fra helt simple patterns som f.eks. singleton.

#### 4.1.2 Forskellige patterns

##### Repository

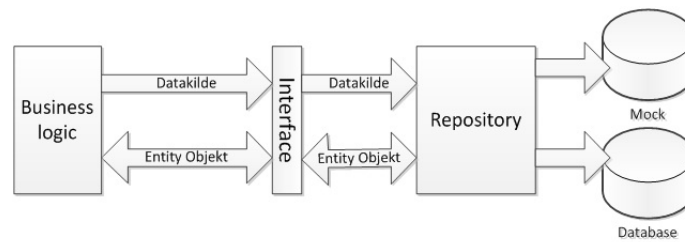
Repository pattern er en måde at adskille kommunikationen mellem databasen og resten af programmet. Det er her vi skriver og læser til og fra databasen. Det skal sammenlignes med et data-accesslag. Når der i dette afsnit tales om repository er det i forhold til Repository pattern, som ikke må forveksles med database repositories.

Ved at implementere repository pattern undgås blandt andet en masse kode-duplikation idet vi kalder ned til vores repository hver gang og genbruger på den måde de forskellige metoder.

Udover mere abstraktion i designet, giver det også mulighed for bedre at kunne unit teste systemet. Det hjælper med at isolere de metoder der skal testes, ved at gøre det langt nemmere at forfalske (Mock) vores data.

### Implementering af Repository

Måden repository'et er implementeret på, er med henblik på at kunne unit teste med et testing framework kaldet Moq [10], dette også giver samtidig den ønskede abstraktion. For at hjælpe med at danne et overblik over vores implementering har vi lavet følgende diagram.



Figur 4.1: Respositorymodel

Som det ses her, har vi vores forrentningslogik først. Her bestemmes hvilken datakilde der skal bruges. Dette vil afhænge af, om vi kommer fra f.eks. en unit test med Mocked data, eller hvis det er en Integrations test som vil bruge de virkelige data til at teste på, eller hvis vi selvfølgelig bare bruger programmet normalt. Dernæst har vi vores interface, som føre os videre ned i vores repository som så henter den ønskede data fra den ønskede datakilde.

Vi startede med at lave et repository lag i vores projekt. Her opretter vi alle vores repository klasser samt deres interface klasser. Se evt. afsnittet om arkitekturen. Næsten alle vores MVC controllere har hvert sit repository da controllerne næsten altid dækker et bestemt område i database. Hvis dette ikke er tilfældet genbruger man selvfølgelig det repository som har de nødvendige metoder. På den måde kan vi også genbruge mange af vores metoder i vores repositories.

Vores repository implementere selvfølgelig det interface der hører sig til. Derudover har vi en constructor hvor vi sætter hvilken datakilde den skal bruge. Da vi bruger Moq kan vi mock dette så den eksempelvis opfatter en simpel tabel som datakilde. Dybere forklaring omkring dette kommer senere i afsnittet.

```
1 public class ContactRepository : IContactRepository
2 {
3     private CRMSystemEntities _db;
4     public ContactRepository(CRMSystemEntities repository)
5     {
6         _db = repository;
7     }
8 }
```

Figur 4.2: Repository Constructor.

I selve respositoriet vil der selvfølgelig være en implementering af hver enkel metode angivet i interfacet, disse er undladt her, da de ikke er relevante.

Da det er vores controllere der står for at styre hvilken datakilde der skal gøres brug af, har vi oprettet en række constructors til netop at styre dette.

```
1 public class LeadDetailsController : Controller
2 {
3     private IContactRepository _repo;
4
5     public LeadDetailsController()
6     {
7         this._repo = new ContactRepository(new
8             CRMSystemEntities());
9     }
10    public LeadDetailsController(IContactRepository repo)
11    {
12        this._repo = repo;
13    }
14
15    public LeadDetailsController(CRMSystemEntities db)
16    {
17        this._repo = new ContactRepository(db);
18    }
19
20    public IContactRepository LeadDetailsRepository
21    {
22        get { return _repo; }
23    }
24 }
```

Figur 4.3: Controller Constructor.

Den første constructor bruges når der eksempelvis skal hentes eller sendes data fra et View som samme controller styrer. Den bruger vores entity framework som standard. Efter den har vi den constructor vi bruger, når vi mocker vores datakilde til Unit Tests. Som det kan ses tager den et interface som parameter. Det er det object vi mocker i vores unit test.

Den sidste bruges hvis dette repository skal bruges fra en anden controller. I Entity frameworket er objekter bundet til en bestemt context, så for at kunne sende objekterne igennem controlleren skal vi have den context med som de er bundet til. Det er det vi gør med denne constructor. Derudover har vi en simpel property for at kunne få fat i repositoryet fra andre controllere.

### Unit Test med repository

(Denne unit test hører til en anden MVC controller og repository end de forgående eksempler) Her vil vi kort beskrive en test metode, for at vise hvordan vi benytter repository pattern i forbindelse med unit testing.

```
1 [TestMethod()]
2 public void BugDetailsTest()
3 {
4     Ticket[] ticketss = new Ticket[] {
5         new Ticket() {Ticketsid = 1, Headline = "test1", Content =
6             "bla",... },
7         new Ticket() {Ticketsid = 2, Headline = "test2", Content =
8             "blabla",... },
9         new Ticket() {Ticketsid = 3, Headline = "test3", Content =
10             "blablabla",... }
11     };
12     int id = 2;
13     Mock<ITicketRepository> mock = new Mock<ITicketRepository>();
14     mock.Setup(m => m.GetDetails(id)).Returns(ticketss[id - 1]);
15
16     TicketController target = new TicketController(mock.Object);
17     var result2 = target.BugDetails(id) as ViewResult;
18     Ticket ticket = (Ticket)result2.ViewData.Model;
19
20     Assert.AreEqual("blabla", ticket.Content);
21 }
```

Figur 4.4: Unit Test.

Testen er en standard unit test der følger den typiske ”arrange, act, assert”. Der startes med at lave et array af Tickets som her vil blive datakilden. Dernæst laves et mock af ITicketRepository. Derudover laves en instans af TicketController hvor der bruges den constructor der tager imod et ITicketRepository objekt som parameter. På den måde ved testen, at vi vil benytte vores array af Tickets som datakilde. Til sidst tjekkes det om det er den forventede data der returneres.



## Decorator

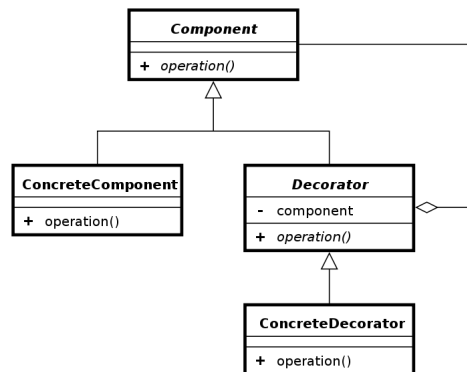
I og med, at der skulle laves et dynamisk filter, hvor det på forhånd ikke var vidst hvilke parametre der skulle filtreres på, var det nødvendigt at finde en løsning der gav en vis grad af fleksibilitet.

Problemet med at bruge Entity framework og LINQ er, at det ikke er muligt at lave dynamiske queries på samme måde som i sql. Dette er nødvendigt i forbindelse med filter-funktionen, da det på forhånd ikke kan siges, hvilke kategorier der skal sorteres på, og hvordan. I SQL kan man f.eks. tilføje en streng til sin query, som er baseret på andre variabler:

```
string completeQuery = query + " AND WHERE userId > 23 " + sortPart;
```

Det er ikke muligt på samme måde at tilføje eksempelvis betingelser i LINQ-kald.

Normalt bruges Decorator pattern til at tilføje funktionalitet til en klasse. Et decorator pattern fungerer ved, at man har en *ConcreteComponent*, som decorator-klassen arver fra. Decorator-klassen er den klasse som skal dekoreres. Den initialisere, hvorefter man giver den videre til en anden klasse som parameter. Denne nye klasse tilføjer så funktionalitet, og den nye klasse bliver så igen givet videre til en anden klasse osv. Denne sende klasser videre, bliver gjort mulig via et factory pattern. Et decorator pattern vil især typisk blive brugt ved grafik, hvor man f.eks. kan tilføje funktionalitet som scrollable, resizable osv til et vindue, hvor vinduet er grundklassen.



Figur 4.5: Decorator Model

På modellen kan ses "grundklassen" (*ConcreteComponent*) og dekoratøren, som er den klasse der skal dekoreres. Øverst ses interfacet *Component*, som er det interface decorator'en arver fra.

## Implementering

Måden vi implementerede decorator-pattern'et på, var ved at først lave en generel abstrakt klasse "ConcreteQuery". Denne klasse indeholder ganske simpelt en liste over companies samt en metode "DoQuery()". Herudover, er der også en constructor, der tager generelle parametre, og lægger dem i de relevante properties. Desuden lavede vi en klasse for hver kolonne i oversigten, som så kan kaldes og filtrere resultaterne.

Hver klasse har en constructor der tager imod det FilterItem der hører til dem, og samtidig tager imod Company-listen som den ser ud pt. Herefter bliver filteret påført listen, og listen returneres.

FilterItem	
Values	List<string>
Criteria	List<int>
CompanyList	List<Company>
Type	string
Category	string

Tabel 4.1: Attributter for FilterItem-klassen

Figur 4.6: city-query klassen, læg mærke til der arves fra concreteQuery og, at den constructor kaldes

```

1 public class CityQuery : ConcreteQuery
2 {
3     public CityQuery(List<CompanyViewModel> companyList, List<
4         string> values, int criteria)
5         : base(companyList, values, criteria)
6     {
7         companyList = (from c in companyList
8             where c.Company.City != null
9             select c).ToList();
10    }
11
12    public override List<CompanyViewModel> DoQuery()
13    {
14        List<CompanyViewModel> returnList = new List<
15            CompanyViewModel>();
16        string value = values[0];
17        if (this.criteria == 1)
18            returnList = (from c in companyList
19                where c.Company.City.Cityname.ToLower().
20                    Contains(value.ToLower())
21                select c).ToList();
22        else
23            returnList = (from c in companyList
24                where !c.Company.City.Cityname.ToLower().
25                    Contains(value.ToLower())
26                select c).ToList();
27
28        return returnList;
29    }
30 }

```

Som det ses af kodeeksemplet i figur 4.6, kan man se, at vi har en fælles constructor der bliver arvet fra ConcreteConstructor, denne constructor bliver kaldt og modtager kriterie og værdi fra det respektive FilterItem-objekt. Disse gemmes under de relevante egenskaber. Herefter tager query-klassen over, og behandler data hvis det er nødvendigt, f.eks. hvis der er tale om tal-data, bliver de konverteret fra en string til integer.

Det er forholdsvis simpelt når vi benytter filteret. Filtercontrolleren får tilsendt en liste med filterItems. Som det ses på figur 4.1 er FilterItem ikke andet end en beholder der holder værdier der er relevante for et enkelt filter.

Herefter løbes listen af filtre igennem, og den dertilhørende query hæftes på concreteQuery-klassen via klassen QueryFactory. Resultatet heraf returneres til controlleren, controlleren serialiserer herefter listen til JSON og sender den tilbage til klienten.

## Kapitel 5

# Produkt

### 5.1 Kravspecifikation

EqualSums har stille en opgave der omhandler udvikling af et CRM system. Det de primært var interesseret i at få udviklet var deres håndtering af kunder. Hvor fokus ville være den måde salgsafdelingen håndtere kunderne på. Da det kun var en lille del af programmet vi skulle lave, blev der stillet krav om at programmet nemt skulle kunne udvides og vedligeholdes.

Selve produktet skulle kunne vise og håndtere alle firmaets kunder, og potentielle kunder. Der skal kunne laves aftaler med de forskellige leads, som også bliver noteret i sælgernes Google-kalender. Derudover skal der være mulighed for at skrive noter på en lang række af de data der er i systemet. Der blev også stillet krav om at kunne sende mails direkte fra programmet, samt hente gamle mails frem.

Det de i bund og grund ønskede, var en effektiviseret kopi af det Google docs regneark de bruger nu, hvor man samtidig har styr over hvad man præcist har aftalt og har haft aftalt med kunderne.

Systemet vil overordnet indeholde tre sider, Min side, Leads Oversigt og leads detaljer. I det følgende vil det kort blive forklaret hvilke opgaver det tænkes der skal løses på hver af de tre sider:

#### ”Min Side”

Min Side vil være en oversigts side, hvor man har vist sin kalender, en liste over dagens gøremål. Det vil være her man starter på når man er logget ind for hurtigt at kunne danne sig et overblik over hvad der skal laves i løbet af dagen.

- Der skal være en oversigt over dagens opgaver, for at gøre det hurtigt og nemt at overskue de opgaver man har for dagen
- Mulighed for at fjerne en opgave når den er løst, så man ikke længere skal se og tænke på den
- Medarbejderens kalender skal vises på siden så man hurtigt og nemt kan se hvilken aftaler man har i løbet af måneden, og dermed også nemmere kan planlægge nye aftaler
- Man skal kunne se detaljer for dagens opgaver så man kan få en mere præcis beskrivelse af hvad der skal laves
- Sortere dagens opgaver så man bedre kan styre hvilken rækkefølge man vil lave dem i

### **”Leads Oversigt”**

Leads Oversigt er en stor tabel over alle de Leads der er i systemet. Her vil være forskellige sorterings muligheder for at kunne finde frem til de leads man vil arbejde med. Tabellen vil vise de vigtigste informationer omkring leadet, så man nemt og hurtigt kan finde den information man er ude efter. Det er også her man kan oprette nye leads, enkeltvis eller via en kommasepareret fil (.CSV).

- Der skal være en oversigt over alle Leads, så man nemt kan finde de leads man har i systemet og læse deres vigtigste data
- Mulighed for at sortere leads efter de viste data så man kan få en liste over præcis de leads man ønsker at arbejde med
- Man skal kunne opdatere data på leads, så de stemmer overens med virkeligheden
- Der skal være mulighed for at afmærke flere leads og opdatere samme felt på dem på én gang, så man slipper for at skulle rette det samme data på mange forskellige leads
- Der skal kunne uploades en CSV fil, med en masse leads så man slipper for at skulle indtaste disse en ad gangen. Der skal samtidig tjekkes for dubletter så man ikke får de samme leads flere gange
- Der skal være en blank plads i bunden så man manuelt kan oprette et enkelt lead
- Skal kunne tilføje en ny kolonne dynamisk i oversigten, så man dynamisk kan tilføje flere data at sortere efter

### ”Lead Detaljer”

Lead Detaljer indeholder en simpel liste af leads til venstre hvor der vil være nogle få sorterings muligheder. Der kan vælges leads ud fra denne liste så man kan få vist alle deres stamdata. Her vil også være mulighed for at se hvilken kontakter man har med leadet.(Kontaktpersoner). Der vil der også være mulighed for at se alle de aftaler man har lavet med dem. Det kan være møder, telefon møder, kurser etc. Her vil der evt. også være vedhæftet en note som man også kan hente frem. Derudover vil der også være noter til både firmaet, kontakter og mails. Der kan også sendes mail fra denne side, samt se tidligere mailkorrespondancer med den valgte kontakt.

- Man skal kunne sortere efter lead status, samt EqualSums-medarbejder så man nemt kan få en præcis liste af leads man vil se dejtalerne på
- Se alle stamdata på et lead, samt en liste af dets kontakter(Kontaktpersoner) så man kan få en dejtaleret information omkring Leadet.
- Man skal kunne se stamdata på en valgt kontakt indenfor det valgte lead, så man nemt kan se de oplysninger der er gemt omkring kontakten
- Man skal kunne rette de stam data der er på både Lead og kontakt, så de stemmer overens med virkeligheden
- Der skal være mulighed for at kunne oprette en ny kontakt på leadet, så man har en bedre mulighed for at kontakte firmaet
- Upload CSV fil med kontakter til et bestemt lead, så man ikke skal indtaste dem manuelt, her tjekkes også for dubletter så man ikke oprette den samme kontakt to gange
- Der skal kunne oprettes noter på leads og kontakter, så man senere kan slå op og læse noten omkring leadet eller kontakten. Derudover skal der også være noter på de Actions(aftaler) man laver med kontakten
- Finde gamle noter frem på både leads og kontakter, så man kan se hvad der er blevet skrevet omkring leadet eller kontakten før i tiden
- Oprette en aftale med et lead eller kontakt(aftale kan være møder, telefon opkald, kurser osv.). aftaleren oprettes også i medarbejderens Google-kalender
- Der skal kunne oprettes noter på aftaler, hvis man skal huske noget specielt omkring aftalen kan man skrive en note omkring det så man er sikker på at huske det
- Find gamle aftaler frem igen samt tilhørende noter, så man hurtigt kan se hvad man egentlig har lovet og aftalt med en kunde, især hvis der opstår uenighed omkring aftaler

- Skal kunne sende en e-mail til den valgte kontakt, samt vedhæft dokumenter i denne. Så slipper man for at skulle åbne et 3. parts program for at skrive en e-mail
- Man skal også kunne vedhæfte filer til sin e-mail
- Der skal være en historik over sendt og modtaget post mellem EqualSums-medarbejderen og kontakten, så man igen kan finde historikken frem og bekræfte hvad der egentlig er blevet aftalt

## 5.2 Design

### 5.2.1 Domænebeskrivelse

Her vil de forskellige klasser og deres associationer kort blive forklaret. Først beskrives klassen, og derefter forklares hvordan associationerne hører til denne klasse.

**Action** Svarer til en aftale der er aftalt mellem en employee og contact. Kunne være et møde.

**Contact:** Der kan være tilknyttet en eller flere kontakter til en action. Der vil altid mindst deltage 1 men nogen gange flere.

**Employee:** Her vil der også være mulighed for at der er flere der deltager. Igen vil der ikke være en action uden en Employee.

**Notes** En Note er en lille besked tilknyttet de klasser den har associationer til. Meget simpel klasse.

**Employee** Indeholder de vigtige informationer der kræves af en medarbejder i virksomheden.

**Company:** Viser hvilken medarbejder der ejer dette firma(Leadet). En ansat kan eje flere virksomheder, men ikke eje et som allerede er ejet af en anden. Han kan også eje 0 virksomheder, da han kan have en anden stilling end Sælger.

**Course:** Viser om denne medarbejder deltager i dette kursus.

**Company** Company indeholder stamdata omkring firmaet(Leadet)

**Contacts:** Viser hvilken kontakter der tilknyttet dette firma. Der kan være flere kontakter men de vil naturligt kun være tilknyttet et firma. Der kan også være 0 kontakter hvis der endnu ikke er skabt kontakt til firmaet.

**Employee:** Viser hvilken ansat der ejer virksomheden. Her vil altid kun være en ejer.

**Contact** Indeholder de vigtige informationer omkring en kontakt.

**Company:** Viser hvilket firma kontakten er en del af.

**Course:** Viser om denne kontakt har forbindelse til et kursus.

**Course** Indeholder data omkring et kursus.

**Contact:** Viser hvilke kontakter der er tilknyttet dette kursus.

**Employee:** Viser hvilke medarbejdere der er tilknyttet dette kursus.

**Participants:** Viser hvilke kursister der er tilknyttet dette kursus.

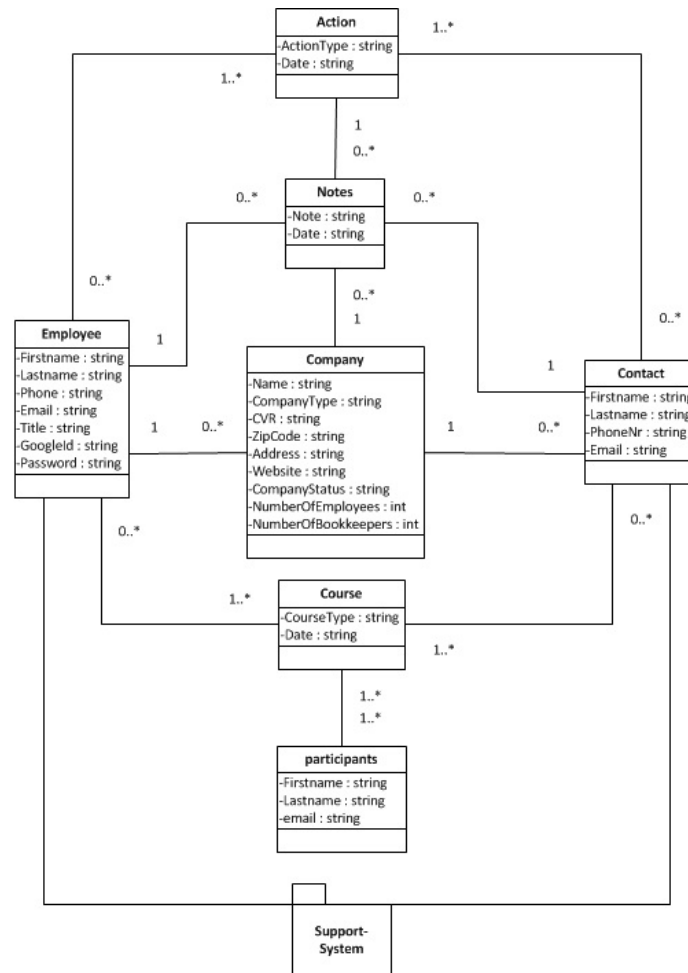
**Participants** Virker som en klasseliste for et kursus. En liste over deltagere som ikke nødvendigvis er kontakter.

**Course:** Viser hvilket kursus listen er tilknyttet. Der vil kun være et kursus pr. liste samt der selvfølgelig kan deltage mange til hvert kursus.

### 5.2.2 Domænemodel

Vi har lavet en domænemodel for at give et bedre overblik over det problemområde vores projekt omhandler. Modellen hjælper os og andre med at forstå hvordan systemet er bygget op. Den viser sammenhængen mellem de forskellige klasser. Navnene på klasserne fortæller hvad vi har med at gøre, mens attributterne fortæller noget om klassens egenskaber.





Figur 5.1: Domænemodel

Det kan ses på modellen at en Employee ikke har direkte forbindelse til Contact, men i stedet har forbindelsen til Company. Dette er fordi de altid vil have alle firmaets kontakter. Action og Course er sat til Contact i stedet for Company for at en ansvarlig kontaktperson til netop den Action eller Kursus. Derudover er Participants sat ind for sig selv da det kan være ansatte i firmaet som man ellers ikke har nogen kontakt til. Support-System er resten af det system der blev lavet under vores praktikophold. Modellen bag det har vi valgt at lave på denne måde for at holde de to projekter adskilt.

### 5.2.3 Databasestruktur

Database designet er lavet ud fra domæne modellen. Vi har forsøgt at holde tabellerne så simple som muligt, det vil sige vi ikke blander data sammen. Derudover forsøger vi at undgå null værdier samt redundans. Sidst vil vi normalisere databasen op til og med Boyce-Codd Normalform for at give os et bedre design, hvor vi minimerer redundans og opdaterer, indsætter og sletter uregelmæssigheder.

Som det ses på domæne modellen er der flere steder mange-til-mange relationer. For at styre dette i database sammenhæng sættes en tabel ind imellem. Her vil de to fremmed nøgler til sammen lave en primær nøgle. Det kan blandt andet ses ved Employee og Action. Her vil der blive sat en ny tabel ind, som indeholder de to fremmed nøgler, som dens primære nøgle.

De steder hvor der er 1-til-mange relationer vil der blive sat en fremmednøgle på den tabel der er mange af. Eksempelvis vil Company have mange kontakter, der sættes en fremmednøgle på kontakten så den altid kan se hvilket firma den hører til. Hvis der havde været 1-til-1 relationer vælges en af tabellerne og vil fremmednøglen ofte blive lagt på den tabel med mindst værdier som peger på den. Man vil også forsøge at ligge nøgler således man undgår nulls.

1. NF kan blandt andet ses i vores Kontakt tabel hvor navnet på kontakten er delt op i fornavn og efternavn, altså atomare data.
2. NF kan ses flere steder i database da vi ikke har nogen attributter der er afhængig af dele af en nøgle.
3. NF ses i det klassiske eksempel med post nummer og by. Da disse er afhængige af hinanden, og den primære nøgle flytter vi ud i en tabel for sig selv. [1, p.348-361]

#### Beskrivelse af databasemodel

Vi vil her beskrive sammenhængen mellem nogen af de tabeller hvor man ikke umiddelbart kan se hvad der menes.

**Company:** Denne tabel har flere tabeller knyttet til sig som kun indeholder en primær nøgle samt en enkelt attribut. (CompanyStatus, CycleState og CompanyTypes) Vi har lagt dem ud i tabeller for sig, fordi vi skal kunne søge og sortere på dem. Det er lister med fastsatte værdier som kunden skal kunne vælge ud fra.

**Titles:** Titles har to mange til mange tabeller, det er fordi en Employee kan have flere titler, samt mange titler kan have adgang til de samme steder i programmet. Det bruges til at styre hvor de forskellige Employees har adgang i programmet.

**Notes:** For at undgå at få en masse null værdier har vi valgt at oprette en

række nye tabeller som ligner det man gør ved mange til mange relationer. Her skal man dog være opmærksom på det faktisk kun er en, en til mange relation.

**ValueList:** Kunden vil være i stand til at oprette dynamiske kolonner af bestemte typer. Derfor har vi lavet denne tabel som indeholder data samt en fremmed nøgle til infoList som fortæller hvilken type data vi skal sortere efter og hvad kolonnens navn er. Disse dynamiske kolonner er ens for alle Companies.

## Kapitel 6

# Projektforløb

Her vil blive forklaret hvordan projektet er planlagt, og gå mere i detaljer omkring de enkelt spændende områder i hvert sprint. Sprintet vil kort blive gennemået hvorefter der vil blive kigget nærmere på nogle kodeeksempler.

I de fire sprints som forløbet strækker sig over, er hvert sprint delt op i 2 uger hvor af ca.  $2\frac{1}{2}$  dag om ugen går med programmering mens de sidste uger af forløbet går over i ren rapportskrivning. Denne struktur er valgt, for at få skrevet lidt ned, mens det man har lavet stadig er nyt og friskt i hukommelsen. Ved ikke at lave for lange sprints får man oftere feedback på produktet hvilket betyder, at man hurtigere kan rette til når eller hvis der kommer ændringer.

### 6.1 Product Backlog

Product backloggen er en liste af alle de user stories der indgår i projektet. Den er med til at give et godt overblik over hvad der mangler at blive lavet. Der kan løbende blive tilføjet nye user stories til product backloggen, som eventuelt kan blive taget med i det næste sprint. 6.2

Story	Subject	Story Type	Description	Importa...	Effort
95	Liste Admin Model	User Story	Som sælger skal jeg kunne modificere mine lister så jeg kan tilføje eller fjerne	100	8
91	Opret hændelse	User Story	Som sælger vil jeg oprette en action så jeg kan huske det når den	100	13
85	Klare gøre arbejdsmiljø	Technical Story	Gøre Visualstudio projektet klar til at begynde arbejdet.	100	8
24	ViewCalendar	User Story	Som Sælger vil jeg kunne Se min kalender for måneden Så jeg kan planlægge	100	4
22	MarkAsComplete	User Story	Som Sælger vil jeg kunne Markere en opgave som løst Så jeg ikke gør	100	8
21	TodayTasksDetails	User Story	Som Sælger vil jeg kunne Se detaljerne for dagens opgaver Så jeg ved	100	8
20	SortTodayTasks	User Story	Som Sælger vil jeg kunne Sortere mine opgaver for i dag Så jeg kan	100	8
14	OpretKontakt	User Story	Som Sælger vil jeg kunne Oprette en ny kontakt under et firma Så Jeg kan gen	100	6
13	CreateNote	User Story	Som Sælger vil jeg kunne gemme en note Så Jeg kan skrive noter jeg skal husk	100	4
12	ShowContactDetails	User Story	Som Sælger vil jeg kunne vælge en kontakt Så jeg kan se alle de oplysninger je	100	9
11	ViewNote	User Story	Som Sælger vil jeg kunne vælge en gammel note Så jeg kan se, hvad jeg har sk	100	8
10	SaveNote	User Story	Som Sælger vil jeg kunne gemme en note Så jeg kan gemme oplysninger om fi	100	8
9	showDetails	User Story	Som Sælger vil jeg kunne klikke på et firma Så Jeg får vist stamdata og kontak	100	8
8	showContacts	User Story	Som Sælger vil jeg kunne se de lead og firmaer der er tilknyttet mig Så jeg kan	100	8
6	Mass update	User Story	Som Sælger vil jeg kunne Markere mange leads på én gang, og opdatere en va	100	8
5	Upload data	User Story	Som Sælger vil jeg kunne uploade hele dokumenter (cvs) med leads, så jeg	100	14
4	Opret nyt lead	User Story	Som Sælger vil jeg kunne Oprette/tilføje firmaer/leads Så Jeg kan få nye leads i	100	8
1	ListeView sorterSubject: ListeView sortering	User Story	Som Sælger vil jeg kunne Sortere og filtrere data i listen over firmaer/leads så	100	20
17	SendMail	User Story	Som Sælger vil jeg kunne Sende mails til kunder Så Kunden får en mail, og jeg	90	13
18	AddFileToMail	User Story	Som Sælger vil jeg kunne Vedhæfte et dokument fra dokumentListen	80	13
25	FindSpot	User Story	Som Sælger vil jeg kunne Både om et ledigt tidsrum, f.eks. 30 minutter	50	40
7	tilføj kolonne	User Story	Som Sælger vil jeg kunne kunne tilføje en kolonne hvor jeg selv bestemmer ty	1	40

Figur 6.1: Backlog

## 6.2 Sprint 1

I det første sprint vil der altid gå tid med at få sat arbejdsmiljø op, men da der skulle arbejdes videre på et projekt som allerede havde sat op, var meget af dette allerede på plads. Der skulle dog laves lidt, da der skulle omstruktureres på opbygningen af projektet ved hjælp af Areas i MVC 3. Derudover blev der også en del refactoring af gamle klasser, fordi der blev lavet en hel del om i database designet, så systemet kunne understøtte al den nye funktionalitet.

### 6.2.1 Sprint backlog

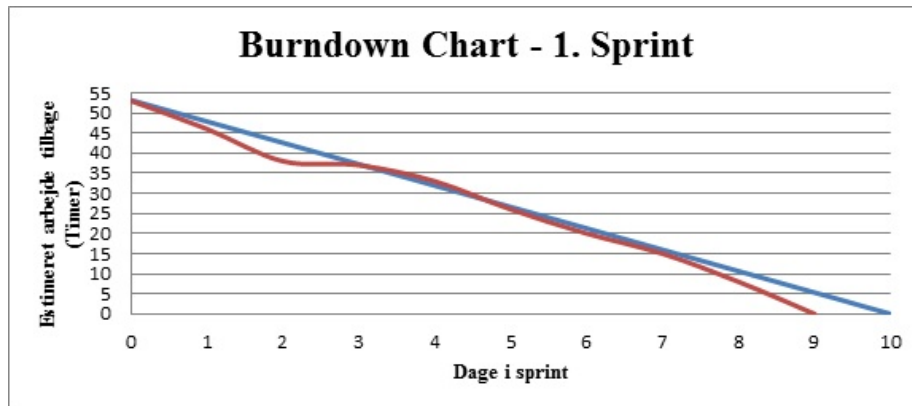
Der blev ikke sat så mange user stories ind i dette sprint for få en god buffer til uventede problemer med arkitekturen og opstart af projektet. Der var også lidt usikkerhed omkring hvor meget der kunne laves på et sprint. Kolonnen "Effort" er det antal timer der er givet hver User Story. Der kan maximalt klares 53,2 i et sprint, og det viste sig at passe ret godt i første sprint.

Subject	Story Type	Description	Importance	Effort
ListeView sorterSubject: ListView sortering	User Story	Som Sælger vil jeg kunne Sortere og filtrere data i listen over firmaer/leads si	100	20
Upload data	User Story	Som Sælger vil jeg kunne uploade hele dokumenter (cvs) med leads, så jeg	100	14
Klare gøre arbejdsmiljø	Technical Story	Gøre Visualstudio projektet klar til at begynde arbejdet.	100	8

Figur 6.2: Sprint 1 backlog

### 6.2.2 Burndown Chart

Som burndown chartet viser, blev de første opgaver løst hurtigere end forventet, men det udlignede sig senere. Bufferen som bevidst var blevet sat ind i dette sprint for at få en god start, blev der heldigvis ikke brug for. Det ses også i form af, at alle opgaver var løst en dag før sprintets deadline.



Figur 6.3: Sprint 1 Burndown Chart

### 6.2.3 Udførelse af Sprint 1

Vi startede med at gøre arbejdsmiljøet klar til kunne håndtere den del af programmet der skulle håndtere salgsafdelingens område. MVC 3 er heldigvis utrolig fleksibelt og vedligeholdelsesvenligt så det gik nemt. Der opstod ikke nogen særlige problemer med hensyn til opsættelse af arbejdsmiljøet.

#### User story - Upload data

*„Som Sælger vil jeg kunne uploade hele dokumenter (cvs) med leads, så jeg kan tilføje mange kontakter på én gang. Så jeg hurtigt kan importere mange kontakter i stedet for at taste dem enkeltvis.“*

Som user story’et beskrive skal der kunne uploades en kommasepareret fil med leads, som efterfølgende skal skrives ind i databasen. Her skal der selvfølgelig tjekkes om der er dubletter så der ikke uploades de samme leads flere gange. En dublet vil være et lead med samme CVR nummer. Skulle der være dubletter i filen vil man bliver ledt over på en siden på figur 6.4 hvor man vil se en liste med unikke leads og en liste med dubletter. Listen øverst er unikke leads, mens den nederste liste er dubletter fundet ud fra CVR-nummer. Her skal man så vælge den dublet man vil bruge, ved at afkrydse checkboxen ved siden af. Grunden til man skal kunne vælge er at der sagtens kan være flere leads med samme CVR nummer i listerne, men med nyere stamdata. Det er så op til sælgeren at vurdere hvilke data er de nyeste.

UploadCSV

Virksomheder

	Navn	Postnr	Adresse	Hjemmeside	CVR	Antal ansatte	Antal Revisorer
<input type="checkbox"/>	Firma2	2000	address2	website2	12346	4	166
<input type="checkbox"/>	Firma7	7000	address7	website7	3456	22	13
<input type="checkbox"/>	Firma8	8000	address8	website8	88888	24	21

Dubletter

	Navn	Postnr	Adresse	Hjemmeside	CVR	Antal ansatte	Antal Revisorer
<input type="checkbox"/>	Firma3	0	address3	website3	123456	12	22
<input type="checkbox"/>	Firma1	1000	address1	website1	123456	2	10
<input type="checkbox"/>	Firma5	5000		website5	123456	1	2
<input type="checkbox"/>	Firma6	6000	address6	website6	12345	22	11
<input type="checkbox"/>	Firma4	4000	address4	website4	12345	2	1

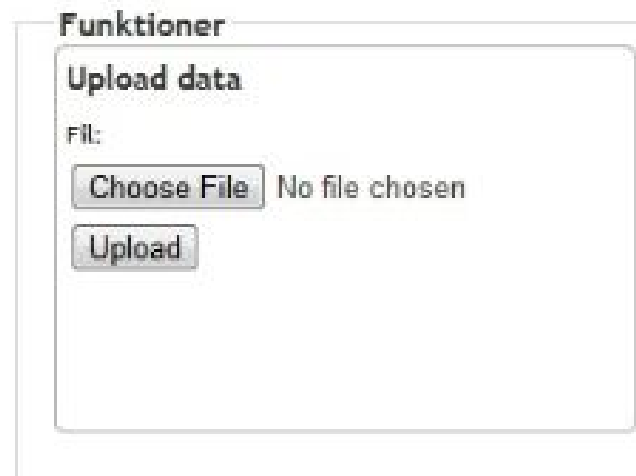
Sorter Dubletter

Figur 6.4: Liste af nye Leads og dubletter i csv-filen.

Undervejs i upload processen tjekkes der for dubletter i CSV filen, og dernæst i databasen inden der skrives hertil.

Det er kun den nederste liste man skal vælge fra. Det er et af de steder vi gerne ville have haft tiden til at lave UI’et lidt om. Istedet for checkboxes skulle der være radiobuttons, da man kun skal kunne vælge ét lead med samme CVR nummer, samt der selvfølgelig ikke skal være nogen valgmulighed ved de unikke leads. Der var problemer med at få grupperet radioknapperne ordenligt via razor. Så alternativet var at lave det om senere, og styre det via JavaScript og jQuery. Det nåede vi desværre aldrig.

Den ønskede CSV fil vælges ved hjælp af en simpel Web-form som man kan finde på Leads-oversigts siden som vist på figur 6.5. Det er vigtigt at et leads stamdata står i den rigtige rækkefølge fordi metoden ikke kan skelne mellem hvilke data det er den læser. Stamdata på et lead vil være oplysninger som CVR-nr, firmanavn, adresse, website osv.



The image shows a web form titled "Funktioner" (Functions) with a sub-section "Upload data". Inside this section, there is a label "Fil:" (File:). Below the label, there is a button labeled "Choose File" and a text label "No file chosen". Below these, there is another button labeled "Upload". The form is enclosed in a light gray border.

Figur 6.5: Webform til upload af CSV-fil

Metoden i figur 6.6 tjekker som det første om filen overhovedet er blevet uploadet korrekt. Såfremt alt går godt, bliver filen kopieret over på serveren, hvorefter stien til filen bliver sendt videre til metoden i figur 6.7 som er metoden der læser filen, mere om den senere.



```

1 [HttpPost]
2 public ActionResult UploadCSV()
3 {
4     List<List<CSVCompanyWrap>> tempList = new List<List<CSVCompanyWrap>>();
5
6     CSVList cvsList = new CSVList();
7     try
8     {
9         HttpPostedFileBase file = Request.Files["
10             uploadCSVContacts"];
11         if (file != null && file.ContentLength > 0)
12         {
13             string fileName = file.FileName;
14             string path = System.IO.Path.Combine(Server.
15                 MapPath("~/Uploads"), fileName);
16             file.SaveAs(path);
17
18             tempList = CSVReader(path);
19
20             cvsList.Companies = tempList[0];
21             cvsList.Doubles = tempList[1];
22         }
23     }
24     catch (Exception e)
25     {
26         ...
27     }
28
29     return View(cvsList);
30 }

```

Figur 6.6: Metode til upload af CSV-fil i ContactListController.cs.

Da man kan sende objekter til vores UI med MVC 3, er der blevet lavet en wrapper klasse til at styre hvilke Company objekter der er blevet valgt på UI'en via checkboxes. Se evt. Arkitektur afsnittet for uddybelse af Models og Views.

CSVCompanyWrap er en wrapper-klasse til Company objekter da der skal være mulighed for at kunne vælge hvilke leads der skal sorteres fra ved hjælp af checkboxes. CSVCompanyWrap indeholder to simple Properties, en bool til at styre om checkboxen er markeret eller ej(true/false) og en property til det givne Company objekt. Ved at lave det på denne måde kan vi bruge MVC 3 til at sende en viewmodel til viewet på figur 6.4. Når man så laver en http-post sendes vores viewmodel tilbage til ContactListController hvor de forskellige checkboxe er valgt til og fra.

CSVList er det vi kalder en ViewModel og indeholder to lister med CSVCompanyWrap objekter, dette er også den model vores view i figur 6.4 forventer at modtage. De to lister svare til en række leads og en række dubletter.

```

1 private List<List<CSVCompanyWrap>> CSVReader(string filePath)
2 {
3     List<List<CSVCompanyWrap>> fullList = new List<List<
4         CSVCompanyWrap>>();
5     List<CSVCompanyWrap> companies = new List<CSVCompanyWrap>();
6     List<CSVCompanyWrap> dublets = new List<CSVCompanyWrap>();
7     try
8     {
9         using (StreamReader reader = new StreamReader(filePath))
10        {
11            string line;
12            while ((line = reader.ReadLine()) != null)
13            {
14                string[] lines = line.Split(';');
15                short number;
16                CSVCompanyWrap cw = new CSVCompanyWrap()
17                {
18                    Company = new Company()
19                    {
20                        ...
21                    },
22                    Choosen = false
23                };
24                if (companies.Where(x => x.Company.Cvr.Equals(cw.Company.
25                    Cvr)).FirstOrDefault() != null)
26                {
27                    dublets.Add(cw);
28                    dublets.Add(companies.Where(x => x.Company.Cvr.Equals(cw.
29                        Company.Cvr)).Single());
30                    companies.Remove(companies.Where(x => x.Company.Cvr.
31                        Equals(cw.Company.Cvr)).Single());
32                }
33                else if (dublets.Where(c => c.Company.Cvr.Equals(cw.Company.
34                    Cvr)).FirstOrDefault() != null)
35                {
36                    dublets.Add(cw);
37                }
38                else
39                {
40                    companies.Add(cw);
41                }
42            }
43        }
44    }
45    catch (Exception e)
46    {
47        ...
48    }
49    fullList.Add(companies);
50    fullList.Add(dublets);
51    return fullList;
52 }

```

Figur 6.7: Metode til læsning af CSV-fil i ContactListController.cs.

Metoden i figur 6.7 læser den uploadet CSV fil og splitter den op hver gang

den støder på et ”;”. For hver linje i filen bliver der lavet et CSVCompanyWarp objekt, med et Company objekt, som stammer fra vores Entity framework, og en false bool.

Derefter tjekkes der så om entity klassen Company eksistere i enten listen af dubletter eller companies. Findes den allerede bliver de flyttet til dubletter listen ellers bliver de lagt i companies listen. Det er ikke kun dubletten der bliver flyttet over i dubletter listen. Det gør begge objekter da der senere skal vælges hvilken der er rigtig, som forklaret tidligere er det op til sælgeren at vurdere hvilket lead har de nyeste data.

Til sidst sendes de to lister tilbage til UploadCSV.cshtml som vist på figur 6.4. Her skal brugeren igen tage stilling til om hvilke data er de rigtige. Det er nemlig ikke nødvendigvis det nyeste eller rigtige data som findes i CSV filen.

```
1 [HttpPost]
2 public ActionResult SortDublets(CSVList csv)
3 {
4     CSVList cvsList = new CSVList();
5
6     foreach(var item in csv.Doubles)
7     {
8         if(item.Choosen)
9         {
10             item.Choosen = false;
11             csv.Companies.Add(item);
12         }
13     }
14     cvsList.Companies = csv.Companies;
15
16     return View("ConfirmCSV", cvsList);
17 }
```

Figur 6.8: Metode til sortering af dubletter i ContactListController.cs.

I figur 6.8 ses en simpel metode der flytter dubletter over i companies listen ud fra hvilken dublet-leads man valgte på hjemmesiden. Herefter bliver man flyttet over til ConfirmCSV.cshtml som ses i figur 6.9. Her skal man bekræfte en sidste gang om man har valgt de rigtige leads.

**Check om data står ordenligt.**

Sæt kryds i dem der skal sorteres FRA!

	Navn	Postnr	Adresse	Hjemmeside	CVR		
<input type="checkbox"/>	Firma2	2000	address2	website2	12346	4	166
<input type="checkbox"/>	Firma7	7000	address7	website7	3456	22	13
<input type="checkbox"/>	Firma8	8000	address8	website8	88888	24	21
<input type="checkbox"/>	Firma1	1000	address1	website1	123456	2	10
<input type="checkbox"/>	Firma6	6000	address6	website6	12345	22	11

Upload Data

Figur 6.9: Godkend stamdata af leads

Det gøres som et simpelt tjek for at man ikke uploader noget data som ikke giver mening. Hvis man skulle vælge et lead som man ikke vil have med pga. forkert data kan man vælge den fra her. Når det er gjort kan man upload sine data til databasen hvor der igen vil blive undersøgt om der er dubletter.

```

1 public ActionResult SubmitCSV(CSVList csv)
2 {
3     List<CSVCompanyWrap> tempList = csv.Companies;
4
5     foreach(var item in tempList.ToList())
6     {
7         if (item.Choosen)
8         {
9             csv.Companies.Remove(item);
10        }
11    }
12    csv.Doubles = new LeadDetailsController(_db).
13        LeadDetailsRepository.AddCompaniesFromCSV(csv);
14    return View("ChooseDbDoubles", csv);
15 }

```

Figur 6.10: Metode til at sortere valgte Leads i ContactListController.cs.

Her løber metoden i figur 6.10 listen igennem for at se om brugeren skulle have fravalgt nogen leads, hvorefter de vil blive fjernet af listen. Til sidst sendes listen til repositoriet hvor de vil blive føjet til databasen.

```
1 public List<CSVCompanyWrap> AddCompaniesFromCSV(CSVList csv)
2 {
3     List<CSVCompanyWrap> companyDoubles = new List<CSVCompanyWrap>
4         >();
5     foreach(var item in csv.Companies.ToList())
6     {
7         CSVCompanyWrap comp = new CSVCompanyWrap()
8         {
9             Company = _db.Companies.Where(x => x.Cvr.Equals(item.
10                 Company.Cvr)).SingleOrDefault(),
11             Chosen = false
12         };
13         if (comp.Company != null)
14         {
15             companyDoubles.Add(item);
16             companyDoubles.Add(comp);
17         }
18         else
19         {
20             _db.Companies.AddObject(item.Company);
21             _db.SaveChanges();
22         }
23     }
24     return companyDoubles;
25 }
26 }
```

Figur 6.11: Metode til at sortere dubletter i database.

Metoden i figur 6.11 ligger i ContactRepository.cs. Derfor skrives og læses der direkte til databasen her. `_db` er vores Entity Framework context.

Inden de endeligt bliver tilføjet til databasen, undersøges listen om der også er dubletter i databasen. Det gøres ved at trække et Company objekt ud fra database via CVR nummeret i den sorteret CSV liste. CVR-nummeret bruges fordi det er unikt og derfor ikke indeholder nulls eller ens værdier. `SingleOrDefault()` returnere enten et objekt eller null. Det vil sige, hvis metoden finder et objekt vil if-sætning være true og så vil de to dubletter blive skrevet i en liste som bliver sendt tilbage til brugeren. Hvis den ikke finder nogen objekter vil leadet fra CSV dokumentet blive skrevet i databasen med det samme.

**Dubletter i Databasen.**

Vælg hvilke Leads du vil beholde eller overskrive.

Dubletter						
	Navn	Postnr	Adresse	Hjemmeside	CVR	
<input type="checkbox"/>	Firma2	2000	address2	website2	12346	4 166
<input type="checkbox"/>	Firma2	2000	address2	website2	12346	155 4
<input type="checkbox"/>	Firma7	7000	address7	website7	3456	22 13
<input type="checkbox"/>	Firma7	7000	address7	website7	3456	111 5
<input type="checkbox"/>	Firma1	1000	address1	website1	123456	2 10
<input type="checkbox"/>	Firma1	1000	address1	website1	123456	
<input type="checkbox"/>	Firma6	6000	address6	website6	12345	22 11
<input type="checkbox"/>	Firma6	6000	address6	website6	12345	

Sorter Dubletter

Figur 6.12: Dubletter af sorteret CSV fil og Database

De dubletter der bliver fundet i CSV dokumentet og databasen skal sorteres på samme måde som tidligere af brugeren. Herefter de bliver sendt ned til repositoriet og overskriver objektet i databasen. Se figur 6.12

### User Story - ListView sortering

*„Som Sælger vil jeg kunne Sortere og filtrere data i listen over firmaer/leads så jeg kan få leads frem i den rækkefølge jeg ønsker“*

Et højt prioriteret punkt på listen over user stories, var muligheden for at sortere leads på forskellige kriterier, så man f.eks. kunne finde alle leads der skulle kontaktes inden for et bestemt datointerval, et bestemt postnummer eller virksomhedstype. Det skulle også være muligt at filtrere på flere kriterier på en gang, så man f.eks. kunne vælge alle virksomheder inden for postnummer 8000-9000 med 20-100 medarbejdere.

Ideen er, at inde på siden med leadsoversigten kan en bruger klikke på en kolonneoverskrift, herefter vises en boks ud for musemarkøren, hvor brugeren kan sætte filteret på denne kolonne op. Når brugeren uden for boksen, bliver filteret gemt og boksen skjules. Herefter kan brugeren enten sætte flere filtre op, eller klikke på "Filtrér" hvorefter filtrene sendes til serveren, der returnerer de leads der falder inden for filternes parametre.

### Vise en Filterboks

Første skridt var, at vise samtlige leads. Dette var en forholdsvis simpel opgave der var hurtigt løst. Næste step var, at lave mulighed for at tilføje filtre. Det blev fastslået, at der grundlæggende var tale om 4 typer filtre: tal, datoer, tekst

og multiple-choice valg. Datatypen for kolonneoverskriften blev specificeret i et array, hvor der var et element for hver kolonne:

I figur 6.13 ses hvordan kolonneoverskrifterne og deres tilsvarende datatyper bindes sammen. Bemærk l. 15-20, her hentes alle mulige employeeStatusser og alle employees ud af databasen. Resultatet af disse lægges i de to arrays defineret i linjerne 8 og 9 - det er kolonner hvor der skal vælges mellem flere fast definerede værdier.

```

1 filter = new Array();
2
3 categories = new Array();
4 categories["Navn"] = "tekst";
5 categories["Adresse"] = "tekst";
6 categories["By"] = "tekst";
7 categories["Postnummer"] = "number";
8 categories["Status"] = new Array();
9 categories["ES-kontakt"] = new Array();
10 categories["Sidste_Kontakt"] = "dato";
11 categories["Kontakt_igen"] = "dato";
12 categories["Web"] = "tekst";
13 categories["Ansatte"] = "number";
14
15     var url = "getEmployeeStatus";
16     $.getJSON(url, function (data) {
17         categories["Status"] = data.states;
18         categories["ES-kontakt"] = data.employees;
19     });
20
21 // $

```

Figur 6.13: Sammenbinding af overskrifter og typer

Der blev lavet fire divs, som blev lagt som det sidste i body-delen af siden, med css-attributten `display: none;` der betyder, at de ikke vises. Disse divs indeholder form-elementer der er baseret på de fire typer af data: dato, tekst, tal eller forudbestemte værdier som f.eks. status og ES-medarbejdere.

Disse 4 formularer ligger i bokse og repræsenterer hver af de fire typer af filtre. Når brugeren klikker på en kolonneoverskrift bliver der lavet en kopi af den relevante boks, såfremt kassen ikke allerede eksisterer.

Figur 6.14: Eksempel på filterboks - her til et tekst-filter

Koden i figur 6.15 håndterer klik på en kolonneoverskrift, og visning af den boks, hvor brugeren bestemmer parametrene for filteret for denne kolonne. Hvis der allerede er sat et filter for denne kolonne vises den gamle filterboks, og hvis det er første gang brugeren klikker på netop denne kolonneoverskrift, oprettes der en ny filterboks som herefter vises.

Der er i denne kodelump en del interessante punkter, det er værd at fremhæve. Først og fremmest i linje 3 og 4, hvor `chosenCategory` ender med at indeholde hvilken slags datatype der er tale om. Herefter bestemmes det i l. 8-26 hvilken formular type der skal vises. Id'et på denne formular (som faktisk er en `div`) gemmes i variabelen `divToOpen`. I linje 26 skjules (ikke fjernes) evt. allerede viste filter-bokse. Efter, i linje 29 at have placeret filterboksen ved musemarkøren, bestemmes det om den valgte filterboks allerede eksisterer.

Er det ikke tilfældet skal den valgte filterboks klones. Den nye boks bliver ikke tilføjet dom-træet umiddelbart, men bliver lagt i variabelen `newElement`. Denne nye boks skal herefter have et id der dannes ud fra kolonneoverskriften og ”\_filter”, radio buttons skal have en `name`-attribut, så de opfører sig korrekt, og kun hører sammen i én specifikke filterboks. Dette betyder, at radioknapper i f.eks. adresse-boksen ikke er kædet sammen med radioknapper i navne-boksen. I linje 36-41 bliver der for dato-felter initialiseret et jQuery plugin, så der vises en dato-vælger ved klik i disse bokse. Endelig appendes den nye boks til DOM-træet på linje 46 og på linje 48 vises den nye boks.

Grunden til, at det bliver undersøgt om filterboksen allerede eksisterer, er for at man kan få lov til at ændre et allerede eksisterende filter.

Hvis den valgte boks allerede eksisterer i DOM-træet bliver det ganske enkelt lagt i `newElement`-variabelen i l. 42.

Som det sidste lægges en `overlay-div` bag den nye filterboks, som, når der bliver klikket på den, vil skjule filterboksen. Denne `overlay-div` eksisterer udelukkende med det formål at registrere klik på et hvilket som helst sted på siden der ikke er filterboks.



```

1  /// Klik på en kolonneoverskrift
2  $('#contactTable th').click(function (e) {
3      var chosenField = $(this).text().replace(" ", "_");
4      var chosenCategory = categories[chosenField];
5
6      var divToOpen;
7
8      if ($.isArray(chosenCategory)) {
9
10         divToOpen = "selectFilter";
11         $('#selectedVal').html('');
12         $.each(chosenCategory, function () {
13             var value = $(this)[0].value;
14             var text = $(this)[0].text;
15             $('#selectedVal').append('<option value="' + value + ' '
16                                     '>' + text + '</option>');
17         });
18     }
19     else if (chosenCategory == "dato") {
20         divToOpen = "dateFilter";
21     }
22     else if (chosenCategory == "number") {
23         divToOpen = "numberFilter";
24     }
25     else if (chosenCategory == "tekst") {
26         divToOpen = "textFilter";
27     }
28
29     $('#.filterDiv').hide();
30     $('##" + divToOpen).css('top', e.pageY + 7).css('left', e.pageX
31         + 7);
32     var suffix = "_Filter";
33
34     if ($('#' + chosenField + suffix).length == 0) {
35         newElement = $('#' + divToOpen).clone();
36         newElement.attr('id', chosenField + suffix);
37         newElement.children('input[type="radio"]').attr('name',
38             chosenField + 'RadioBtn');
39         $(newElement).find('.datePickerField').datepicker({
40             showWeek: true,
41             firstDay: 1,
42             showOtherMonths: true,
43             selectOtherMonths: true
44         });
45         $('#body').append(newElement);
46         //$('#body').after(newElement);
47     }
48     else {
49         newElement = $('#' + chosenField + suffix);
50     }
51     $(newElement).fadeIn("fast");
52     $(newElement).after("<div class='darkClass'></div>");
53 });

```

Figur 6.15: Komplet eventhandler for klik på kolonneoverskrift

### Gemme filter

Når brugeren klikker udenfor filterboksen (dvs. på overlay-div'en) bliver filteret gemt. Det første der sker, er at overlay-div'en fjernes, herefter bliver div'en med filterboksen lagt i en variabel og gemt, så brugeren ikke kan se den længere. Det næste der sker, er at filterets kategori bliver bestemt ud fra id på filterboksen og ud fra denne kategori, filterets type (tekst, tal, dato eller multiple choice). Baseret på typen bliver filteret nu behandlet, så det kan gemmes i et array "filters", der består af en række filter-objekter. Til sidst bliver den relevante kolonneoverskrift markeret med en streg under teksten, så man kan se, at der er oprettet et filter for denne kategori.

I figur 6.16 ses hvordan overlay-div'en fjernes, filterdiven skjules og id'en hentes ud (l. 2-6). I linjerne 14-20 ses et eksempel på, at der gemmes en tekst-værdi. I filter-objektet gemmes egenskaberne values, criteria, type og category. Values indeholder værdien/erne. Ved datoer og tal er values altid et array, da der skal være mulighed for en øvre og nedre grænseværdi. Criteria indeholder kriteriet for filteret (større end, mindre end, må indeholde, må ikke indeholde osv). Type er ganske enkelt type af filteret og category indeholder kategorien.

```

1  $('body').delegate('.darkClass', 'click', function () {
2      $('.darkClass').remove();
3
4      var newlyHiddenElement = $('.filterDiv:visible');
5      newlyHiddenElement.hide();
6      var hiddenElementId = newlyHiddenElement.attr('id').split("_");
7
8      if (hiddenElementId.length == 3)
9          category = hiddenElementId[0] + "_" + hiddenElementId[1];
10     else
11         category = hiddenElementId[0];
12     var categoryType = categories[category];
13     switch (categoryType) {
14         case "tekst":
15
16             var criteria = parseInt($(newlyHiddenElement).children(
17                 'input:radio:checked').val());
18             var value = new Array($(newlyHiddenElement).children(
19                 'input[type="text"]').val());
20             if (value != "")
21                 filter[category] = { values: value, criteria:
22                     criteria, type: "tekst", category: category };
23             break;
24             // $
25             ...

```

Figur 6.16: Uddrag af eventhandler for oprettelse af filter

### Anvend filter

Når bugeren klikker på "Filtrér" blive der oprettet et asynkront kald, der indeholder filter-betingelserne (fig 6.17, l. 2-9). Dette kald sendes til serveren, der

filtrerer. Når klienten modtager resultatet fjernes først den gamle oversigt over leads (l. 11), hvorefter listen med resultater løbes igennem. I kodeeksemplet er det i linje 13, at løkken starter. Dato-felter bliver formatteret til et fornuftigt format (udeladt), herefter samles en ny række (l. 19-26 - dele udeladt). Som det sidste tilføjes den netop oprettede række til DOM-træet (l. 27)

Serverside-delen af filtreringen bliver nærmere forklare på s. 24 under overskriften "Implementering".

```

1  formattedFilter = JSON.stringify(newFilter);
2  $.ajax({
3      url: "ApplyFilter",
4      data: formattedFilter,
5      type: "POST",
6      dataType: "json",
7      contentType: 'application/json; charset= utf-8',
8      processData: false,
9      success: function (data) {
10
11          $('#contactTable tr:nth-child(n+2)').remove();
12
13          for (var company in data.returnCompanies) {
14              var currentCompany = (data.returnCompanies[company]);
15              //console.log(currentCompany);
16
17              ...
18
19              var newTr = "<tr>";
20              newTr += '<td><input id="check_' + currentCompany.Id +
21                  '" type="checkbox" value="' + currentCompany.Id + '
22                  "></td>';
23              newTr += '<td><p>' + currentCompany.Navn + '</p></td>';
24
25              ...
26
27              newTr += '<td><p>' + currentCompany.Ansatte + '</p></td>';
28              newTr += "</tr>";
29              $('#contactTable tbody').append(newTr);
30          }
31      });
32  // $

```

Figur 6.17: Uddrag af selve filtreringen, clientside

Det er et bevidst valg, at filtrene ikke bliver anvendt i samme øjeblik filteret bliver oprettet. Dette skyldes, at der skal være tid til at oprette flere filtre af gangen, uden at skulle vente på, at resultatet af den forrige filtrering bliver loadet og vist.

### 6.2.4 Retrospective

Det første sprint gik utroligt godt, vi fik overraskende få problemer med at klargøre miljøet, hvor vi ellers havde regnet med der kunne opstå flere problemer. Vi fik set en af MVC's stærke sider, ved at man nemt kan vedligeholde det, og tilføje store ændringer til det. Samtidig fik vi også set hvor nemt det bliver at lave selv forholdsvis store ændringer i databasen når man bruger entity frameworket. Der blev selvfølgelig en del refactoring i vores gamle projekt, men på trods af de store ændringer var det hurtigt og nemt at få til at virke igen. Blandt andet skulle vi implementere repository pattern, udover at få adskilt vores database kald fra controllerne fik vi også bedre mulighed for at unit teste. Derudover var der en del ændringer i databasen omkring vores Customer og Company klasser, men da Entity frameworket laver vores modellag var det også forholdsvis nemt at skrive om.

Da vores support del af programmet lå i roden af vores MVC3 projekt skulle der også laves en anden mappe struktur her. Det viste sig at være nemt at gøre med MVCs Areas som betød vi mere eller mindre bare kunne flytte hele projektet ind i en ny mappe, og samtidig bevare al funktionalitet, og mens alle links inden for det Area stadig pegede det rigtige sted hen.

Refactoring blev en nødvendighed da systemet skulle udvides, men også som en del af at forbedre og optimere den kode som allerede var skrevet. Da der bruges mange nye teknologier finder man ofte nye, smartere og mere rigtige måder at gøre tingene på i forhold til de framework man bruger.

## 6.3 Sprint 2

I sprint 2 var der mange små opgaver. Det var mange CRUD opgaver, hvor der blev brugt en hel del JavaScript på at få lavet en GUI uden for mange synkrone serverkald. Det er også derfor noget så simpelt som at vise nogle detaljer tager så "lang" tid som det gjorde her. Det meste af GUI'en kom på plads i dette sprint.

### 6.3.1 Sprint Backlog

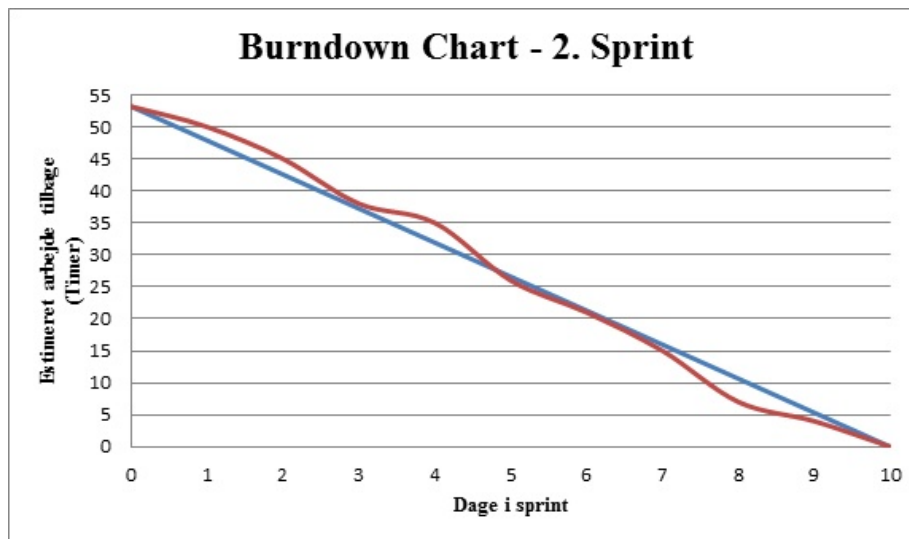
Efter sprint 1 blev vores sprint "Capacity" bekræftet, derfor vi var mere rolige ved at sætte mange user stories i dette sprint og udlade den buffer vi havde sidst. Det er en masse små Stories som samtidig kræver at der blev lavet en del GUI.

Story	Subject	Story Type	Description	Imp...	Effort
4	Opret nyt lead	User Story	Som Sælger vil jeg kunne Oprette/tilføj firmaer/leads Så Jeg kan få nye leads i	100	8
6	Mass update	User Story	Som Sælger vil jeg kunne Markere mange leads på én gang, og opdatere en va	100	8
8	showContacts	User Story	Som Sælger vil jeg kunne se de lead og firmaer der er tilknyttet mig Så jeg kan	100	8
9	showDetails	User Story	Som Sælger vil jeg kunne klikke på et firma Så Jeg får vist stamdata og kontakt	100	8
12	ShowContactDetails	User Story	Som Sælger vil jeg kunne vælge en kontakt Så jeg kan se alle de oplysninger je	100	9
95	Liste Admin Model	User Story	Som sælger skal jeg kunne modificere mine lister så jeg kan tilføje eller fjerne	100	8

Figur 6.18: Sprint 2 backlog

### 6.3.2 Burndown Chart

Vi kom lidt bagud til at starte med, det var hovedsageligt mass-update der tog lidt længere tid end forventet. Da mange af opgaverne mindede om hinanden, indhentede vi den tabte tid igen og nåede i mål til tiden.



Figur 6.19: Sprint 2 Burndown Chart

### 6.3.3 Udførelse af sprint 2

#### User story - Mass update

„Som sælger vil jeg kunne Markere mange leads på én gang, og opdatere en variabel der gælder for dem alle“

Første punkt, var at implementere en måde, så brugeren kunne vælge hvilke leads der skal opdateres. Dette var forholdsvis simpelt, da der ganske enkelt blev tilføjet en checkbox yderst til venstre, der havde `companyId` som value. Næste del var at lave en måde, hvor brugeren kunne fortælle hvilken kategori der skulle opdateres, og hvad den nye værdi skulle være. Der blev tilføjet et element i den øverste bar, hvor man kunne vælge kategori, denne select-liste blev lavet ud fra hvilke kategorier der fandtes i tabellen, som set i fig 6.20

Denne funktion findes på listen over leads, og skal bruges til at opdatere data for mange leads på én gang. F.eks. hvis mange firmaer får ny kontaktperson ved ES.

Første punkt, var at implementere en måde, så brugeren kunne vælge hvilke leads der skal opdateres. Dette var forholdsvis simpelt, da der ganske enkelt blev tilføjet en checkbox yderst til venstre, der havde `companyId` som value. Næste del var, at lave en måde, hvor brugeren kunne fortælle hvilken kategori der skulle opdateres, og hvad den nye værdi skulle være. Der blev tilføjet et element i den øverste bar, hvor man kunne vælge kategori, denne select-liste blev lavet ud fra hvilke kategorier der fandtes i tabellen, som set i fig 6.20

```
1 $.each($('#contactTable th:nth-child(n+2)'), function () {  
2     $('#categoryToUpdate').append('<option value="' + i + '">' + $(  
3         this).text() + '</option>');  
4 });
```

Figur 6.20: Løkke der fylder selectboxen baseret på kolonner i tabellen

### Valg af kategori

I figur 6.21 ses eventhandleren der håndterer valg af kategori der skal opdateres. Når brugeren vælger kategorien, bliver typen bestemt på samme måde som ved oprettelse af et filter (l. 2-3). Hvis der er tale om et array, bliver der lavet en selectbox med de foruddefinerede værdier (l. 5-14), og ellers bliver der oprettet et inputfelt. Hvis der er tale om et datofelt bliver dato-plugin'en også initialiseret (l. 15-23).

```

1 $('body').delegate('#categoryToUpdate option', 'click', function ()
2 {
3     var chosenField = $(this).text().replace(" ", "_");
4     var chosenCategory = categories[chosenField];
5
6     if ($.isArray(chosenCategory)) {
7         $('#massUpdateInput').html('<select id="updateInput"></select>');
8         $.each(chosenCategory, function () {
9             var value = $(this)[0].value;
10            var text = $(this)[0].text;
11
12            $('#updateInput').append('<option value="' + value + '">' +
13                text + '</option>');
14        });
15    }
16    else if (chosenCategory == "dato") {
17        $('#massUpdateInput').html('<input type="text" id="updateInput"
18            class="datePickerField" />');
19        $('#updateInput').datepicker({
20            showWeek: true,
21            firstDay: 1,
22            showOtherMonths: true,
23            selectOtherMonths: true
24        });
25    }
26    else if (chosenCategory == "number") {
27        $('#massUpdateInput').html('<input type="text" id="
28            updateInput" value="nummerfelt" />');
29    }
30    else if (chosenCategory == "tekst") {
31        $('#massUpdateInput').html('<input type="text" id="
32            updateInput" value="tekst" />');
33    }
34 }

```

Figur 6.21: Eventhandleren for valg af kategori

**Gemme værdier** Når værdierne skal gemmes, skal siden finde ud af, hvilket firma-id der er sat hak ved i oversigten, og hvad den nye værdi er. At finde virksomhederne er forholdsvis simpelt, der laves en jQuery-løkke der løber igennem samtlige checkboxe inde i td-tags, der er checked. Herefter er det bare at tage value af disse checkboxes og tilføje dem til et array over companyId's. Når virksomhederne er kendte, skal den nye værdi trækkes ud. Disse værdier skal lægges i to variabler, newVal og newText. Hvis der er tale om tal, datoer eller tekst, er disse to identiske. Hvis der derimod er tale om data fra en selectbox, er der forskel på disse to. Variablen newVal er værdien (value) fra selectboxen, mens newText indeholder teksten.

Når værdierne er hentet ud, er der kun tilbage at sende forespørgslen til serveren,

som vil opdatere værdierne, skrive de nye værdier i tabellen og afmarkere alle selectboxe.

### ShowDetails

*„Som sælger vil jeg kunne klikke på et firma, så jeg får vist stamdata og kontakter“*

Denne userstory hænger tæt sammen med den næste - at se detaljer for kontakter. Begge userstories realiseres på contactDetail-siden.

Først skulle oversigten over virksomheder vises, denne oversigt blev lavet som et partial view, der blev vist. Controlleren for dette view tager en enkelt parameter, der afgør hvordan virksomhederne sorteres. Sorteringsrækkefølgen ændres ved klik i kolonneoversigten.

```
1 public ActionResult _LeadsList(string sortOrder)
2 {
3     List<CompanyList> companies = new List<CompanyList>();
4     companies = repo.GetCompaniesList(companies, sortOrder);
5
6     return PartialView("_LeadsList", companies);
7 }
```

Figur 6.22: Controller for Oversigt over virksomheder

Herefter skulle klik på en virksomhed på listen håndteres. Det blev vurderet, at det bedste ville være, at hente virksomheden via et asynkront kald, der hentede så meget som muligt ind på én gang. Derfor hentes stamdata for virksomheden ind som partial view. Dette partial view returnerer en mængde html, som sættes ind på siden via jQuery. Samtidig foretages der et andet asynkront kald, der henter alle virksomhedens kontakter. Disse kontakter og deres tilhørende data gemmes i en variabel, da de bekvemt nok kommer tilbage, pakket pænt ind i et array. I figur 6.23 ses hvordan listen samles på serveren og herefter returneres som et JSON-objekt.



```

1 public ActionResult getContacts(int companyId)
2 {
3     var contacts = repo.GetContactsByCompany(companyId);
4     string navn = repo.GetCompany(companyId).Name;
5     var contactList = (
6         from c in contacts
7         select new
8         {
9             firstName = c.Firstname,
10            lastName = c.Lastname,
11            contactId = c.Contactsid,
12            phone = c.Phone,
13            cell = c.Mobilephone,
14            actions = (from b in c.Actions
15                      orderby b.Date descending
16                      where b.Date < DateTime.Now
17                      select new
18                      {
19                          actionId = b.Actionsid,
20                          date = b.Date,
21                          desc = b.Description,
22                          type = b.Actiontype.Actiontypesid,
23                          employees = (from d in b.Employees
24                                      select
25                                          d.Firstname + " " + d.
26                                              Lastname)
27                      }),
28            email = c.Email,
29            ...
30    }
31    return Json(new { contactList, navn, notes }, JsonRequestBehavior
32    .AllowGet);
33 }

```

Figur 6.23: Listen over kontakter sættes sammen.

Ud fra dette objekt laves oversigten over kontakter i højre side. Ved at hente al information ud på én gang spares tid når brugeren klikker på en kontakt. Vi vurderede, at den ekstra tid for det ekstra kald, ikke ville betyde noget, i forhold til ventetiden ved at lave et ekstra separat kald ved klik på oversigten. Hvis det senere viser sig, at det er meget store mængder data der skal hentes, som vil forårsage en væsentlig ventetid, kan det let lade sig gøre, at data om en enkelt kontakt først hentes ud ved klik på den specifikke kontakt.

### ShowContactDetails

*„Som sælger vil jeg kunne vælge en kontakt, så jeg kan se alle oplysninger jeg har gemt om kontakten“*

Når en bruger klikker på en kontakt i oversigten, udledes kontaktens id af id'et på den div, der blev klikket på, hvorefter der sættes to cookies, én med contactId'et, og en med nummeret på den kontakt-div der blev klikket på:

```
1
2 $('body').delegate('.contactDiv', 'click', function ()
3 {
4
5     clickedElement = $(this).attr('id').split("_")[1];
6     var clickedContact = contactList[clickedElement];
7     setCookie("actualContactId", clickedContact.contactId);
8     setCookie("contactId", clickedElement);
```

Figur 6.24: Hentning af data om den valgte kontakt.

Herefter fyldes data i de felter hvor de hører til. Siden er i forvejen opbygget med en række div's der har et span-element som child. Dette span-element indeholder de forskellige data og bruges også til at redigere data. Da variabelen clickedContact indeholder alle oplysninger om kontakten er det en forholdsvis simpel ting at udfylde felterne.

```
1 var esContact = clickedContact.ESContact;
2 $('#chosenContact').val(clickedContact.contactId);
3 $('#navn_contact_div').children('span').text(clickedContact.
4     firstName + " " + clickedContact.lastName);
5 $('#mail_contact_div').children('span').text(clickedContact.email
6     );
7 $('#telefon_contact_div').children('span').text(clickedContact.
8     phone);
9 $('#cell_contact_div').children('span').text(clickedContact.cell
10    );
11 $('#medarb_contact_div').children('span').text(esContact.
12    firstName + " " + esContact.lastName);
```

Figur 6.25: Visning af basal data for kontakt

Det næste der sker, er at actions skal vises. Igen ligger alle actions allerede i clickedContact-objektet. Af hensyn til overblikket skal der højst vises de tre sidste actions, længere nede på siden findes en komplet oversigt over alle actions på den aktuelle kontakt. Igen er der allerede en div med et span-element som child, der er klar til at holde informationen:

```

1 var prevActions = ""
2
3 maxActions = (clickedContact.actions.length > 3) ? 3 :
   clickedContact.actions.length;
4
5 for (var actionNo = 0; actionNo < maxActions; actionNo++)
6 {
7   var action = clickedContact.actions[actionNo];
8   var newDate = parseInt(action.date.replace(/[\d.]/g, ""));
9
10  prevActions += " (" + getDateYear(newDate) + ") " + action.desc +
   "<br />";
11 }
12
13 $('#lastContact_contact_div').children('span').html('');
14 $('#lastContact_contact_div').children('span').html(prevActions);

```

Figur 6.26: Tilføjelse af actions i oversigten

```

1 $('#.companyNote').remove();
2
3 //Viser notes for kontakten
4 for (var noteId in clickedContact.notes)
5 {
6   var note = clickedContact.notes[noteId];
7
8   var entryDate = eval("new " + (note.date).replace(/\\/gi, ""));
9   note.dato = (dateFormat(entryDate)).replace(/\\/gi, "-");
10
11   $('#contactNotes').after("<div class='companyAction'><div class='
   companyNote' id='compNote_" + note.notesId + "'><span class=
   ='companyNoteHeader'>" + note.employee + " - " + note.dato +
   "</span><div class='noteContent'>" + note.content + "</div></
   div></div>");
12 }

```

Figur 6.27: Visning af notes for den valgte kontakt

Som det sidste skal der vises opkald og møder med kontakten, hvilket også er forholdsvist simpelt.

```

1  //Viser opkald for kontakten
2  for (var actionId in clickedContact.actions)
3  {
4      var action = clickedContact.actions[actionId];
5      var entryDate = eval("new " + (action.date).replace(/\\/gi, ""))
6      );
7      action.dato = (dateFormat(entryDate)).replace(/\\/gi, "-");
8      var element = "";
9      if (action.type == 1)
10         element = "opkald";
11      if (action.type == 2)
12         element = "meetings";
13
14      if (element != "")
15         $(''#' + element).after("<div class='companyAction'><div class
16         ='companyNote' id='compNote_" + action.actionId + "'><
17         span class='companyNoteHeader'>" + action.dato + "</span
18         ><div class='noteContent'>" + action.desc + "</div></div>");
19
20  }
21  });
22
23  // $

```

Figur 6.28: Eventhandleren for valg af kategori

I disse to user stories blev der lagt en del tanker bag opbygningen af den del, hvor stamdata som telefonnummer, adresse, navn osv. blev vist - både for firma og kontakt. Grunden var, at det skulle være nemt at ændre oplysninger. Product owner ønskede, at det skulle være så simpelt så muligt. Det blev vurderet, at det nemmeste ville være en løsning, hvor brugeren klikkede på en information som f.eks. telefonnummeret, hvorefter teksten ændredes til et tekst-felt. Efter man har ændret oplysningen og klikker udenfor tekstfeltet, ændres det tilbage til tekst og databasen opdateres.

Dette resulterede i en opbygning, hvor hver information havde følgende html:

```

1  <div id="city_firm_div" class="infoDiv">
2      <span>@Model.City.Cityname</span></div>
3  </div>

```

Figur 6.29: Eksempel på datavisning.

Med denne opbygning var det nemt at lave en jQuery eventhandler der reagerede på klik i span i divs med klassen "infoDiv". Teksten i span'en kunne også nemt udskiftes med en infoboks. Denne infoboks har så en eventhandler på, der reagerede når tekstboksen mistede fokus. Via *delegate*-funktionen var det ikke nødvendigt at lægge eventhandleren på hver gang.

Desuden var det også simpelt, for information omkring den enkelte kontakt, at udfylde felterne, da det var span'en i den relevante div der skulle udfyldes via jquery's `.html()`-funktion. Denne funktion giver mulighed for at indsætte html i et bestemt element.

### 6.3.4 Retrospective

Det var et forholdsvis nemt sprint uden de helt store udfordringer, men også et vigtigt sprint da der kom en masse GUI på plads. Det var rart at have noget rigtigt at vise vores product owner.

Vi snakkede også om at vi skal passe på med ikke at kæle for meget ved de helt små detaljer. Derudover skal vi holde os til de user stories/tasks der er skrevet ned og ikke begynde at lave ny funktionalitet, også kaldet "Feature Creep". På trods af det var det et godt sprint der endnu gang endte til tiden, men igen også et sprint uden de store udfordringer.

## 6.4 Sprint 3

I sprint 3 tog vi hul på Note delen af vores system, samt det at kunne oprette en hændelse, hvor den sidst nævnte nok er det mest interessante i dette sprint. Notes delen var en nogenlunde simpel CRUD operation, mens hændelse var noget mere vanskelig da der skulle synkroniseres med Googles kalender.

Derudover skulle der også laves så man kunne se sin kalender samt oprettelse af Kontakter.

### 6.4.1 Sprint Backlog

Der var forholdsvis mange stories i dette sprint, hvor "Opret hændelse" var den klart mest komplekse som kunne skabe lidt problemer hvis vi var uheldige. Der var også en del research involveret med den da vi skulle synkronisere den med Googles Calendar.

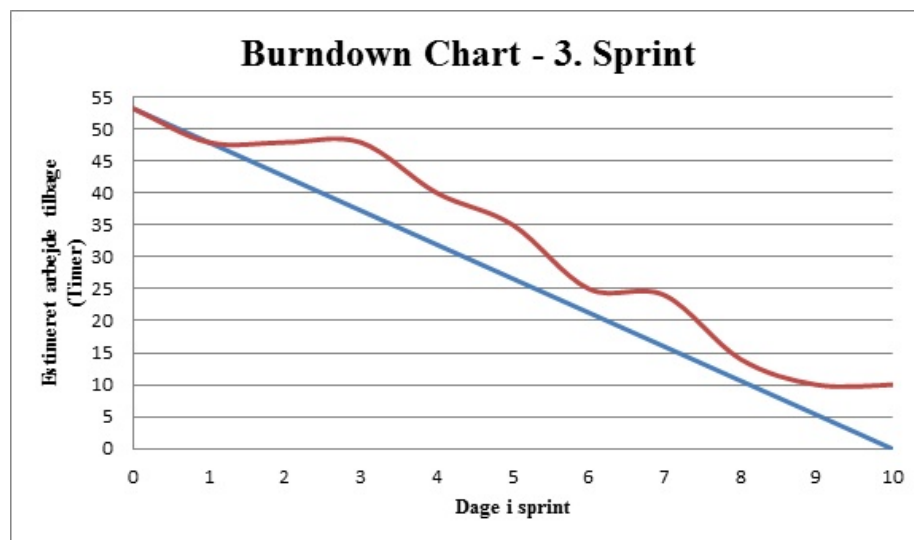
Subject	Story Type	Description	Importa...	Effort
SaveNote	User Story	Som Sælger vil jeg kunne gemme en note Så jeg kan gemme oplysninger om firmaerne	100	8
ViewNote	User Story	Som Sælger vil jeg kunne vælge en gammel note Så jeg kan se, hvad jeg har skrevet om fir	100	8
CreateNote	User Story	Som Sælger vil jeg kunne gemme en note Så jeg kan skrive noter jeg skal huske om denne	100	4
OpretKontakt	User Story	Som Sælger vil jeg kunne Oprette en ny kontakt under et firma Så Jeg kan gemme oplysni	100	6
ViewCalendar	User Story	Som Sælger vil jeg kunne Se min kalender for måneden Så jeg kan planlægge	▼ 100	4
Opret hændelse	User Story	Som sælger vil jeg oprette en action så jeg kan huske det når den	▼ 100	13

Figur 6.30: Sprint 3 backlog

### 6.4.2 Burndown Chart

Som det kan ses på burndown chartet gik det helt galt i dette sprint. Ikke så meget fordi vores estimer ikke holdte, men fordi firmaet vi skrev opgave hos gik konkurs. Det fik selvfølgelig den betydning at vores udviklingsmiljø blev lagt ned og gjorde det umuligt at fortsætte, da vi brugte deres servere. De ville heldigvis gerne hjælpe os færdige med projektet så serveren blev sat op igen på en ny adresse hvor vi så kunne udvikle videre hjemmefra. Det betød dog der var et par dage mens serverne blev flyttet som også kan ses efter dag 1. Da serverne kommer igen fortsætter vi arbejdet og når næsten igennem sprintet på trods af det store slag det var at firmaet gik ned. Da det var en private bredbånds linje serverne blev stillet op på, betød det desværre at forbindelsen var en smule ustabil hvilket kan ses på dag 6.

Det var primært "Opret hændelse" vi ikke nåede at blive færdig med.



Figur 6.31: Sprint 3 Burndown Chart

### 6.4.3 Udførelse af sprint 3

Hvis man ser bort fra problemerne omkring serverne, gik sprintet sådan set udmærket, hvor vi nåede de fleste af vores user stories. Visning og oprettelse af Notes blev lavet som de første, sammen med oprettelse af Kontakt.

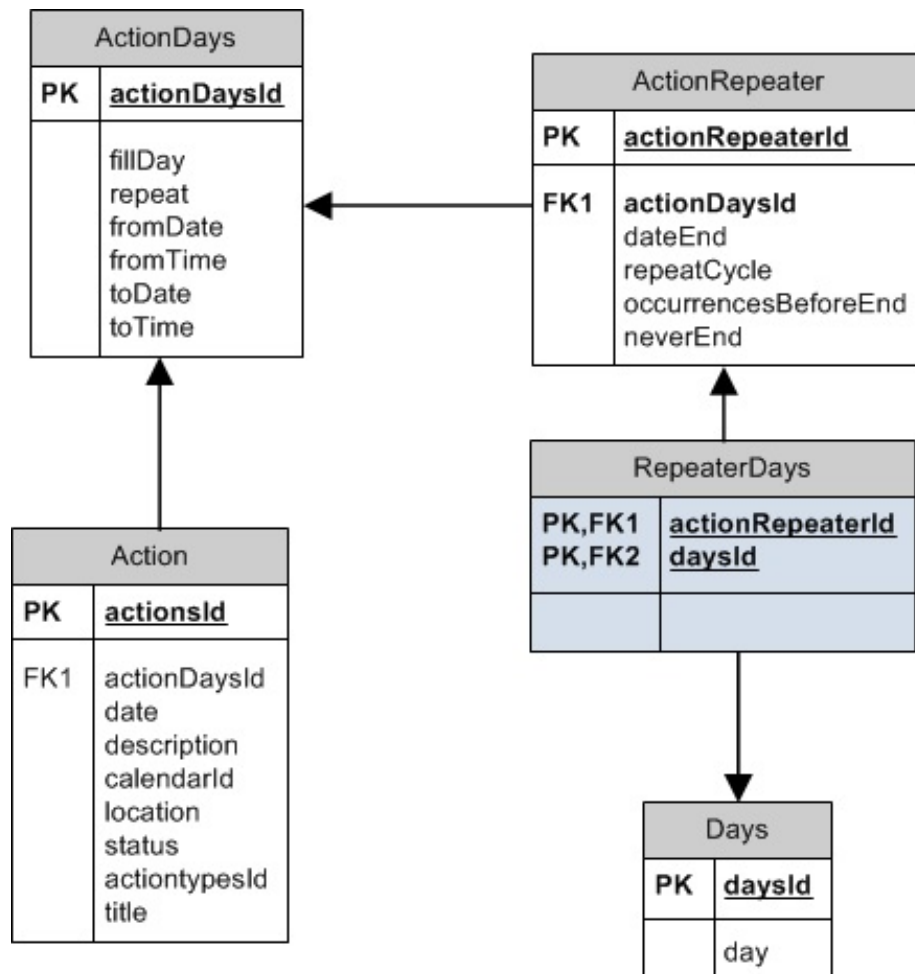
View Calendar havde vi lavet en proof of concept på, så den burde være nem at implementere ordenligt i systemet. Derudover manglede opret hændelse at blive lavet færdig, det lykkedes os dog at lave en stor del af den.

### User Story - Opret Hændelse

*"Som sælger vil jeg oprette en action så jeg kan huske det når den også blive skrevet til min google-kalender"*

Det vores product-owner egentlig ønskede var tæt på en kopi af Googles egen måde at gøre det på. Det betød at udover de mest basale features som overskrift, beskrivelse, til og fra tidspunkter, skulle der også laves gentagelser. Det vil sige der skulle være mulighed for at gentage en hændelse X antal gange hver tirsdag, for eksempel.

Det valgte vi at løse ved at lave en tabel til kun at styre gentagelser, samt en til at styre datoerne.



Figur 6.32: Udsnit af databasediagram.

Som det kan ses på figur 6.32 er der også en date på Action. Dette er noget vi selvfølgelig vil refactor om, så vi ikke har det samme data to steder. Men grundet vores tidsmæssige problemer i dette sprint, nåede vi det aldrig, men vi er dog opmærksomme på dette.

Da oprettelse af en hændelse er en del af et større View med mange Models bruger man et PartialView. Det vil sige man kan sætte et View ind i et View og på den måde dele sine UI op i de forskellige sektioner man nu har på en side. Man kan også sende en anden Model til sit PartialView, det gør det utrolig nemt og smart at bygge sine sider op på den måde.

```

1 <div id="detaljer" class="mainWindow">
2   ...
3       <fieldset>
4           <legend>Hændelser</legend>
5
6           @using (Html.BeginForm())
7           {
8               @Html.Action("Action")
9           }
10      </fieldset>
11  ...
12 </div>

```

Figur 6.33: Importerer et PartialView.

På figur 6.33 ses det View der kalder det PartialView som indeholder den kode der skal til for at vise vores Hændelses sektion. Der kan blandt andet bruges en "helper" kaldet Action. Som parameter tager den her navnet på den metode som skal kaldes i controlleren. At helperen og metoden der bliver kaldt, begge hedder Action er tilfældigt.

```

1 [ChildActionOnly]
2 public ActionResult Action()
3 {
4     ActionViewModel avm = new ActionViewModel();
5     avm.ActionType = repo.GetActionTypes();
6
7     return PartialView("ActionViewModel", avm);
8 }

```

Figur 6.34: ChildActionOnly Metode

På figur 6.34 ses den metode som blev kaldt fra vores View i figur 6.33. Attributten [ChildActionOnly] gør at der kun kan kaldes fra et View. Selve metoden henter en liste af ActionTypes som bliver lagt i vores partialviews model. Listen bruges til at fylde en dropdownbox med data. Til sidst returnerer vi et Parti-



alView(), som her tager to parameter. Den første er navnet på det view vi vil tegne, mens den anden er det objekt vi vil sende med som Model.

I stedet for et postback, laves der et AJAX kald som sender et JSON objekt med til controllerens post-metode. Så længe JSON objektet er bygget op på samme måde som det objekt post-metoden tager som parameter finder MVC selv ud af at lave JSON objektet om til et C# objekt.

```
1 [HttpPost]
2 public JsonResult AddAction(ActionViewModel jsonAction)
3 {
4     ActionViewModel avm = jsonAction;
5
6     CRMSysModel.Action calendarEvent = new CRMSysModel.Action
7     (
8         Title = avm.Title,
9         Description = avm.Description,
10        Location = avm.Location,
11        Actiondays = avm.Actionday,
12        Actiontype = avm.ActionType[0],
13        Calendarid = "en streng",
14        Status = "Igang"
15    );
16
17    repo.CreateAction(calendarEvent);
18
19    return Json(new { status="ok" }, JsonRequestBehavior.AllowGet)
20    ;
21 }
```

Figur 6.35: Hændelse post-metode

Som det ses på figur 6.35 kommer JSON objektet ind som parameter og bliver selv „binded“ om til et ActionViewModel objekt. Det objekt indeholder al den information vi skal bruge til at lave vores Hændelse. Der oprettes en Action med de forskellige properties hvorefter det bliver sendt til Data Access-laget og gemt i databasen.

Her ville vi selvfølgelig gerne have haft sendt informationen videre til vores Google Calendar, så vi kunne oprette en hændelse i brugers kalender.

Metoden her er selvfølgelig ikke helt færdig, da der vil komme nogle tjek på om man har valgt hændelsen til at være hele dagen, eller der skal være gentagelser. Det, at kunne invitere andre til et møde, fik vi heller aldrig implementeret. Det nåede vi desværre ikke som forklaret tidligere i afnisset.

### 6.4.4 Retrospective

På trods af omstændighederne var det egentlig et udmærket sprint. Vi nåede de fleste af vores story points, og der var ikke rigtigt noget at stille op overfor det der skete. Desværre fik vi aldrig rigtigt taget hul på det der kunne have været rigtig spændende at få lavet noget mere på, nemlig Googles API. Vi ville også gerne have haft mere tid til at lave nogle flere tests i det her sprint.

## 6.5 Sprint 4

Sprint 4 blev vi nødt til at springe helt over, da vi simpelthen blev sat for langt bagud i sprint 3 og da vores deadline nærmede sig prioriterede vi rapporten højest. Det var et sprint hvor vi ville være blevet færdig med vores sidste user stories med 100 i vigtighed(Importance). Det ville primært være på "Min Side" der skulle laves features til at styre dagens opgaver. Derudover skulle der også kunne sendes mails fra "Leads Detajler" siden. Derudover skulle den også synkroniseres med de mails der allerede var blevet sendt via Googles interface. Så her skulle der igen gøres brug af Googles API.

### 6.5.1 Sprint Backlog

Det er vores sidste planlagte sprint, og derfor begynder de user stories med mindre vigtighed at dukke op. Det er de sidste af vores vigtigste user stories der er med her.

Subject	Story Type	Description	Importa...	Effort
SendMail	User Story	Som Sælger vil jeg kunne Send mails til kunder Så Kunden får en mail, og jeg kan have	▼ 90	10
AddFileToMail	User Story	Som Sælger vil jeg kunne Vedhæfte et dokument fra dokumentListen	▼ 80	8
SortTodayTasks	User Story	Som Sælger vil jeg kunne Sortere mine opgaver for i dag Så jeg kan	▼ 100	13
TodayTasksDetails	User Story	Som Sælger vil jeg kunne Se detaljerne for dagens opgaver Så jeg ved	▼ 100	13
MarkAsCompleted	User Story	Som Sælger vil jeg kunne Markere en opgave som løst Så jeg ikke gør	▼ 100	8

Figur 6.36: Sprint 4 backlog

### 6.5.2 Burndown Chart

Burndown chartet er blevet undladt da der slet ikke startet op på dette sprint.

### 6.5.3 Udførelse af sprint 4

I praktik perioden blev der lavet en metode til at sende automatisk e-mail til kunder der havde kontaktet kundeservice, så implementeringen af selve afsendelsen af e-mail vil være en overskuelig opgave. Dertil kommer den synkronisering af e-mails som er blevet sendt til en kunde via Googles UI, før de er blevet oprettet i systemet som kunder. Selvfølgelig også efter hvis man skulle sende flere mails fra Google.

Udover sending af e-mails er UI lavet til TodayTask user stories, dog med noget dummy data, blot for at vise produkt owneren hvordan det ville komme til at se ud.

## 6.6 Accept test

Sidst i sprint tre ville vi gerne have haft vores produkt owner og en sælger til at lave en række Accept test som vi har skrevet. Formålet med disse tests er at få produkt owneren til at teste om produktet lever op til hans krav. Der er ofte langt fra beskrivelserne i user stories til det brugerne rent faktisk mente. Ved hele tiden at have vist de nye metoder i vores sprint reviews burde der ikke være så mange overraskelser på den front. Det er mere funktionelle fejl, og test om brugerne kan finde ud af brugergrænsefladen. Ud fra den respons vi ville have fået kunne vi have planlagt vores fjerde sprint med de rettelser og eventuelle fejl der ville blive fundet ville finde. Vi vil starte med at teste nogle af systemets vigtigste funktioner.

Vi nåede desværre aldrig at få sat vores accept test i gang. Vi ville gerne have lavet en bruger til de to testere, og så egentlig bare sætte dem foran systemet og lade dem køre vores accept tests igennem. De skal selvfølgelig tage noter omkring hvordan de oplever at bruge systemet. Samt skrive de fejl ned de oplever, og eventuelle måder tingene kan gøres smartere for dem.

Vi mener det er vigtigt at dem der rent faktisk skal bruge systemet er med til at teste det.

### 6.6.1 Filter Leads

Denne metode bruges til at filtrere leads i oversigten. På den måde kan man finde frem til præcis de leads man ønsker at skulle arbejde med.

- Gå ind på Leads oversigt siden
- Find alle Leads med post nummer mellem 9000 og 9400
- Derefter filtre dem fra

### 6.6.2 Upload CSV

Her kan man uploade en række leads i en CSV fil. På den måde kan man oprette mange leads afgangene frem for et.

- Gå in på Leads oversigt siden
- Upload udleveret CSV fil til systemet
- Vælg de Leads blandt CSV-dubletterne der er rigtige
- Godkend de valgte leads
- Vælg de Leads blandt DB-dubletterne der er rigtige

### 6.6.3 Find Lead Details og Kontakt Details

Her finder vi detaljer omkring et bestemt lead og dens kontakter.

- Sorter firmaerne så kun Leads bliver vist i menuen til venstre
- Find stamdata omkring leadet ved at klikke på navnet
- Find stamdata omkring en Kontakt fra leadet ved at klikke på navnet i højre side
- Find og læs en note ved at kigge på note historie
- Opret en note omkring firmaet ved at klikke på note knappen
- Ret i firmaets stamdata ved at klikke i feltet der skal rettes

## Kapitel 7

# Sikkerhed

Eftersom det er en web-application er der altid nogle sikkerhedsaspekter forbundet med disse. Især hvis det er en siden som bliver lagt ud på Internettet så alle har adgang. Nu er vores kun til internt brug, så vi forventer ikke der er nogen ondsindet brugere af vores system. Med det sagt har vi selvfølgelig stadig gjort os nogle overvejelser omkring de mest basale sikkerhedshuller i webapplikationer. Vi mener det er et område der skal tages seriøst, selvom de data vi arbejder med ikke er personfølsomme bør man stadig beskytte dem hvis man skulle være uheldig at få de forkerte folk ind i virksomheden.

Et af de steder man ofte ser angreb på hjemmesider er via SQL injections. Det er et angreb direkte på de data man har i sin database, og derfor kan det være meget alvorligt hvis man kan få adgang til disse. Man kan trække data ud som man ikke normalt skal kunne se, eller man kan sågar slette hele tabeller.

Man udnytter de steder hvor brugeren skal taste oplysninger ind, da de input vil blive sendt videre til databasen. Dem manipulerer man så database tror det er en SQL kommando som den så udføre i stedet for at sætte de data ind som man egentlig bad den om. Et simpelt eksempel kan være en login form, hvor bruger skal indtaste brugernavn og kode. Hvis det ikke er lavet ordentligt kan man kunne logge ind uden at kende passwordet. Hvis man skriver `fakepassword' OR '1'` i password feltet vil databasen læse det som at endte skal passwordet være "fakepassword eller også skal "1" være true. Dermed har man så fået adgang da 1 vil blive opfattet som true. Alle de steder hvor man sender noget til databasen som brugeren kan manipulerer på en eller anden måde skal man være opmærksom på denne type angreb.

Da vi bruger LINQ er der allerede taget hånd om dette problem for os. LINQ bruger en klasse kaldet `SqlParameter` som sørger for at vores input bliver lavet til parameter værdier, som uskadeliggør denne type angreb. [7] Havde man bygget sine SQL-strenger op med almindelig SQL skulle man selv sørge for disse parameter blev lavet for at undgå injections.

Et andet typisk angreb på web-sites er Cross-site scripting (XSS). Her kan man eksempelvis lægge noget html eller JavaScript kode ind i databasen, hvorefter det så bliver hentet ud igen og fortolket af browseren. Hvis man eksempelvis opretter en support-ticket i vores system, kan man tilføje `<b>` og `</b>` rundt om sin besked. Hvis man så ikke tager højde for den slags vil browseren læse det som at teksten skal stå med fed. Simpelt og harmløst, men det kan bruges til langt værre ting som at opsnappe personlige oplysninger osv. Da vi bruger Razor i vores views slipper vi nemt for den slags angreb da, de metoder vi bruger til at skrive tekst ud med har allerede taget højde for den slags, og skriver alt teksten ud, uden browseren fortolker noget af det. Ønsker man at brugeren skal kunne formatere sine inputs kan man bruge `HTML.Raw()`, men så er man igen sårbar overfor XSS.

For at bruge vores system skal man have en bruger med login og password. Når man har en bruger er en tilknyttet en eller flere titler. Det kan eksempelvis Sælger, Support eller Administrator. Ud fra disse titler og et sitemap over hele sitet kan vi styre hvilken område hver bruger har adgang til. De steder hvor bruger ikke har adgang vil bruger typisk ikke kunne se linket der til, og skulle bruger skrive adressen manuelt vil han bliver flyttet væk med en besked om adgang er nægtet.

For at logge ind har brugeren som sagt et brugernavn og password. Password har vi krypteret med en MD5 kryptering. Det er aldrig smart at gemme passwords i klar tekst. Ville man have yderligere beskyttelse her kunne man vælge at salte sit password. Da vi ved det kun er til internt brug følte vi det var nok blot at kryptere passwordet. Når man salter et password blander man et andet ord sammen med passwordet som så bliver krypteret. På den måde bliver det endnu sværere at regne ud hvad passwordet i virkeligheden er, hvis man skulle få fat i det.

## Kapitel 8

# Perspektivering

I dette afsnit vil vi evaluere på forskellige områder i vores projektforsløb. Vi vil se på hvad der er gået godt og hvad der eventuelt kunne have været gjort bedre og hvorfor.

### 8.1 Roller og samarbejde

Eftersom vi har benyttet os af Scrum som arbejdsmetode indebar det en række roller som skulle besættes. Nogen af rollerne var selvskrevne mens andre er blevet taget ud fra et valg hvor vi mente vi fik mest ud af vores kompetencer. Roller blev inddelt som følgende. Martin Kristensen, Slagchef i EqualSums blev naturligt ProductOwner da han stod for kontakten til os. Peter Thomsen blev ScrumMaster, Gregers Boye-Jacobsen blev udvikler i ScrumTeamet. Scrum Masterens opgaver bestod primært af at holde ScrumDesk opdateret og sørge for vi så godt som muligt fulgte den plan vi havde lagt og fik lavet tingene i den rigtige rækkefølge. De daglige Scrum møder tog vi i fællesskab med de øvrige medarbejdere hos EqualSums, hvilket betød vi på den måde hele tiden blev opdateret omkring hvad vi lavede.

Udover Scrumroller har vi også forsøgt at tage de arbejdsopgaver hvor vores kompetencer var højst. Gregers har blandt andet beskæftiget sig mere med frontend end Peter, hvor Peter derimod har fokuseret mere på backend koden. Det føler vi har virket utrolig godt, da vi begge primært arbejder med det vi er gode til, og hvor vores primære interesse også ligger. Det har også betydet vi kunne nå at lave mere kode end hvis vi skulle sidde og kæmpe med noget den anden kunne klare på langt kortere tid.

Dette var en arbejdsfordeling vi begge fandt helt naturlig under praktikperioden, så det så vi ingen grund til at ændre. Det skal dog siges, at vi har lavet lidt af

det hele, og været rigtig gode til at tage tid til at hjælpe og ikke mindst forklare og lære fra os. Det er noget vi begge har haft stor gavn af.

Derudover har vi haft en rigtig god åben kommunikation, hvor vi hele tiden har fået snakket mange forskellige ting igennem og diskuteret problemstillingerne godt igennem. Det føler vi i høj grad har været med til at højne produktets kvalitet og gjort, at de valg vi har truffet i hvert fald ikke er grebet ud af den blå luft.

Når vi har kodet på programmet har det næsten kun været ude på EqualSums hvor vi har siddet ved siden af hinanden med hver sin computer. Det har derfor været nemt for os hele tiden at sparre med hinanden, og hjælpe hvis vi skulle sidde fast med et problem. Samtidig har vi også haft mulighed for at spørge de andre

Skrivearbejdet har vi forsøgt at dele ud dagen før, og er primært blevet skrevet hjemme. Her har vi igen holdt kontakt med hinanden endte over MSN eller Skype. Vi har hele tiden forsøgt at holde en tæt kontakt, på den måde har vi forsøgt at gøre projektet til ét helt projekt selvom vi selvfølgelig har arbejdet på hver sine afsnit.

## 8.2 Arbejdsmetode

I vores metodeafsnit har vi som nævnt valgt primært at bruge Scrum, et få XP og UP elementer. De scrum praksis vi har brugt har i høj grad hjulpet os med at få et større overblik over projektet. Da man har stor åbenhed i form af produktbacklogs og sprintbacklog har det være nemt for os at se hvem der laver hvad, og havde vi været flere i gruppe er vi sikker på dette ville have været endnu tydeligere. Vi har dog savnet det at vi kunne hænge vores backlogs op på et whiteboard, da vi føler det giver endnu større overblik. Dog har vi til gengæld haft adgang til vores backlogs alle steder, da de ligger online i ScrumDesk. Det har vi især benyttet os af når vi har skrevet hjemme.

Vi har været glade for at arbejde efter en agil arbejdsmetode da vi ofte løbende har fået små ændringer fra kunden. Det ville have været langt værre hvis vi havde lagt os mere fast i et design fra UP.

De diagrammer vi har gjort brug af fra UP har hjulpet os hurtigere i gang, da de hurtigt giver et billede af hvilken retning projekt skal formes. Det giver os et godt fundament som vi kan læne os op af især i starten, for senere at bliver brugt mindre og mindre. Senere i forløbet har vi tjekket op på vores kode og diagram for at sikre at de stadig stemmer overens. Ellers har vi refactoret dem så de passer med virkeligheden.

De elementer vi har brugt fra XP føler vi i høj grad har været med til at øge kvaliteten i programmet. Vi gjorde meget brug af at refactor vores eksisterende



kode, og der er mange steder i koden hvor vi kunne forbedre metoder, og optimere stumper af vores program væsentligt. Det er det af værktøjerne fra XP vi har brugt mest. Derudover kunne vi godt tænke os at lave det til et test drevent projekt, men der er bare en indlæringskurve der gjorde vi ikke havde tid til i dette projekt.

Rent fysisk har vi siddet i samme lokaler som kunden og udviklet. Dette har været en stor hjælp, idet vi altid (op til konkursen) har kunne spørge, hvis der var ting vi ønskede at få afklaret. Samtidig har kunden kunne se vores fremskridt, og stoppe os, hvis der var noget vi havde misforstået.

Det har naturligvis hæmmet os, at virksomheden undervejs gik konkurs, da dette medførte, at vi ikke kunne bruge deres servere og database. Dette har betydet, at vi ikke har nået at færdiggøre produktet. Vi er dog stadig tilfredse med det vi så har nået, og er glade for, at vi ikke valgte at skrive rapporten først, og udvikle til sidst.

Alt i alt har vi været rigtig glade for den arbejdsmetode vi valgte i starten. Den har givet os en passende frihed, og samtidig sat nogle rammer op som hjælper en med at strukturere projektet på fornuftig vis. Vi føler den har været med til at give os et bedre produkt og samtidig muligheden for at nå længere.

## 8.3 Produkt

Som tidligere nævnt, nåede vi ikke at blive færdige med produktet. Dette var heller ikke meningen fra starten af, da vi også skulle have tid til at skrive rapport. Kunden har dog været indforstået med dette, og vi har i udviklingsfasen gjort os store anstrengelser for, at det er nemt for en anden udvikler at overtage produktet.

Med fare for at lyde som en konklusion fra folkeskolen, så har det været et spændende produkt at udvikle. Vi har begge erfaring med webudvikling, men ikke mvc-frameworket. Der har derfor været en del læring, men vi har ikke haft brug for at sætte os ind i html, css og andre grundlæggende web-teknikker. Dette har været til stor gavn, og har betydet, at vi har kunne koncentrere os om alt det nye der skulle læres ifbm. mvc-framework. Derudover var der også en masse andre ting der skulle læres som f.eks. LINQ og entity framework.

## Kapitel 9

# Konklusion

Vi skrev i problemformuleringen, at vi ville fokusere på patterns der kunne hjælpe med at løse opgaven. Vi har i det forløbne kigget på repository, factory og decorator pattern, som vi har forklaret og implementeret. Der er brugt andre patterns i forløbet, men disse er ikke blevet forklaret. Disse patterns er så grundlæggende, at vi vurderede, det ikke ville være relevant at gå i dybden med disse.

Hvad angår de nævnte patterns har de været til stor hjælp, og patterns er helt klart noget man som udvikler skal være opmærksom på, da man ellers risikerer at opfinde den dybe tallerken gang på gang.

Samtidig er en anden konklusion, at en agil tilgang til projektet har været den rigtige, da det har givet mulighed for, at kunden har kunne komme med input undervejs.

Skal der konkluderes på selve produktet, må konklusionen være, at MVC er et meget taknemmeligt framework der i mange henseender gør en udviklers liv nemmere. Vi er overraskede over, hvor meget vi egentlig har fået udviklet i forhold til den begrænsede tid, og at vi kun har været to mand om projektet, og der har været relativt mange nye teknologier at sætte sig ind i.

Mange af de valg vi har foretaget i starten mht. teknologier er kommet os til gode, da disse også har gjort livet nemmere.

## Bilag A

# Diagrammer

Diagrammer kan findes på den vedhæftede cd sammen med kildekode og en elektronisk kopi af denne rapport.

# Litteratur

- [1] Ramez Elmasri. *Fundamentals of database systems*. Pearson Addison Wesley, Boston, 2007. ISBN 032141506X.
- [2] Adam Flaherty. How to use three-legged oauth, April 2009. URL <http://answers.oreilly.com/topic/1381-how-to-use-three-legged-oauth/>.
- [3] Adam Freeman. *Pro Asp.Net Mvc 3 Framework*. APress, Berkeley, 2011. ISBN 9781430234043.
- [4] Scott Guthrie. Introducing 'razor' - a new view engine for asp.net, juli 2010. URL <http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx>.
- [5] Craig Larman. *Applying UML and patterns*. Prentice Hall, Upper Saddle River, NJ, 2005. ISBN 0131489062.
- [6] Microsoft. Unit testing in asp.net mvc applications @ONLINE, . URL [http://msdn.microsoft.com/en-us/library/gg416510\(v=vs.98\).aspx/](http://msdn.microsoft.com/en-us/library/gg416510(v=vs.98).aspx/).
- [7] Microsoft. Frequently asked questions (linq to sql) @ONLINE, . URL <http://msdn.microsoft.com/en-us/library/bb386929.aspx>.
- [8] Microsoft. Entity framework entitycollection and datacontractjsonserializer, Oktober 2008. URL <http://connect.microsoft.com/VisualStudio/feedback/details/354859/entity-framework-entitycollection-and-datacontractjsonserializer>.
- [9] Microsoft. The repository pattern, 12 2011. URL <http://msdn.microsoft.com/en-us/library/ff649690.aspx>.
- [10] Moq. Moq.
- [11] Joseph Rattz. *Pro Linq: Language Integrated Query in C# 2010*. APress, Berkeley, 2010. ISBN 9781430226536.