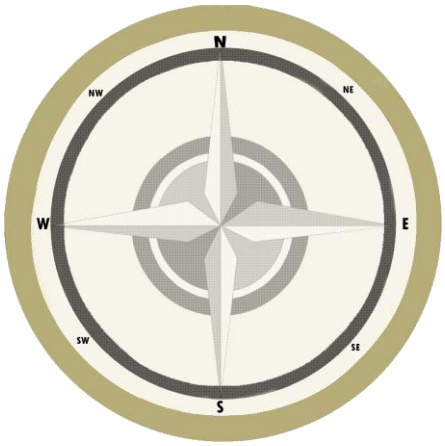
LedighedsKompasset

Fra idé til applikation



En 2. semester-opgave udarbejdet af
Jakob Grosen, Mathias Olsen,
Marc Nielsen og Nicolai Ortiz

INDHOLDSFORTEGNELSE

FORORD

Denne rapport forsøger, efter bedste evne, at opsummere det lange forløb vi har været igennem med udviklingen af vores 2. semester-projekt. Vi har sørget for at komme vidt omkring og lagt fokus på en masse ting, vi selv fandt interessante i forbindelse med projektet.
Selve rapporten er, som udgangspunkt, dikteret af Marc, Mathias og Nicolai, imens Jakob har stået for den skriftlige formulering. Det er angivet i teksten, hvis dette ikke gør sig gældende for specifikke afsnit. God fornøjelse!

1. INCEPTION

1.1 Introduktion til projektet:

I dette projekt har vi fået etableret et samarbejde med Benjamin Kristiansen, fra virksomheden Symbiotic Solutions.
Virksomhedens formål er, groft sagt, at løse én virksomheds problemstilling, med en anden virksomheds kompetencer.
Løsningerne bliver som udgangspunkt realiseret som software, hvor Symbiotic Solutions fungerer som bindeleddet imellem udviklerne af løsningen og modtageren af samme.

1.2 Business canvas

Ud fra et indledende møde med Benjamin, samt de forskellige business-models vi har vedlagt som bilag, har vi fået etableret følgende business canvas for Symbiotic Solutions:

<table>
<tr>
<td>Key partners:

- Jobcentre

- Arbejdernes Landsbank</td>
<td>Key activities:

- Formidling af kontakt og viden mellem udviklere af et produkt og brugerne af samme</td>
<td rowspan="2">Value proposition:

- Overordnet: Virksomheden hjælper med at løse daglige problemstillinger for dennes kunder, vha. software-løsninger

- Specifikt for opgaven: Virksomheden gør hverdagens aktiviteter, i forbindelse med jobsøgning og ledighed, nemmere</td>
<td>Customer relationships:

- Personlig assistance til virksomheder

- Co-creation ved mellem virksomhed såvel som slutbruger</td>
<td>Customer segments:

- Multi-sided platform: Imellem Symbiotic Solutions' kunde og dennes kunder.</td>
</tr>
<tr>
<td colspan="2">Key resources:
- Menneskelige: Roskilde kommune, fagforeninger, præsidenten for Rotary i Roskilde

- Intellektuelle: Stor viden om forretningsverdenen, Roskilde kommune, fagforeninger

- Fysiske: Bil, PC, telefon</td>
<td colspan="2">Channels:

- Digitalt: Hjemmeside, kommunikation pr. mail

- Fysisk: Møder, mund-til-mund, kommunikation pr. telefon</td>
</tr>
<tr>
<td colspan="3">Cost structure:

- Transport

- Udgifter til etablering af nye projekter

- Forbrug</td>
<td colspan="2">Revenue streams:

- Licens og salg af produkter til diverse virksomheder, private såvel som offentlige

- Micro-transactions igennem leverede produkter og ydelser</td>
</tr>
</table>

Derudover har vi fået etableret et samlet Vision Statement for projektet.
Dette er blevet udarbejdet internt i vores gruppe, hvor efter det er blevet tilrettet
og godkendt af Benjamin.

1.3 Vision Statement

Projektet, som præsenteret af Benjamin fra Symbiotic Solutions, er i sin essens en applikation der skal hjælpe arbejdsløse og ledige individer med de daglige og ugentlige gøremål,
der skal udføres i forbindelse med ledighed.
Dette inkluderer påmindelser, generelt såvel som i forbindelse med online ledighedsbekræftelse, bogføring af relevante aktiviteter, som opdatering af CV, jobsøgninger, mv.
Derudover har applikationen også til formål at vejlede og motivere den ledige,
igennem sit ledighedsforløb.

Positioning - Problem Statement

The problem of	Besværligt og demotiverende at være arbejdsløs.
Affects	De arbejdsløse.
the impact of which is	Frataget dagpenge/kontakthjælp og dårligt mentalt helbred.
a successful solution would be	Motiverende og dokumenterende værktøj i form af en softwareapplikation.

Positioning - Product Position Statement

For	De arbejdsløse.
Who	Har brug for motivation og administrativ vejledning.
The (product name)	Mobil application.
That	Motiverer og vejleder.
Unlike	Jobnet og jobcentre.
Our product	Tilbyder en lettere og overskuelig samlet løsning.

Stakeholders - Descriptions

Name	Description	Responsibilities
Investor	Benjamin: Entreprenør og hovedansvarlig for projektet.	- Videreformidle produktet til potentielle købere. - Definere og opretholde krav til projektet. - Formidle relevant information mellem kunde og udvikler.
Investor	Arbejdernes landsbank: Kapitalhaver	- Styrke deres omdømme og brand.

Name	Description	Responsibilities
		- Leverer markedsføringskanaler til Benjamin.

User Environment

- Arbejdsopgaverne i systemet bliver udført af den enkelte bruger.
- I den endelige udgave er det påkrævet at brugeren har internet adgang.
- Systemet udvikles som "Proof of Concept" til Windows™ Phone platformen med planer om realisering af systemet til Android™ og iOS™ platformen.
- Brugeren vil have en gennemsnitlig task-cycle på 4-5 minutter.

Product Overview – Needs and features

Need	Priority	Features	Planned Release
Påmindelse om registrering af ledighed	H	Justerbar timer	Juni 2015
Mulighed for registrering af aktiviteter	H	Kalender logbog	Juni 2015
Personlig motivation	H	Gameification og visuel repræsentation af fremgang	Juni 2015
Vejledning ift. lovgivning	H	Vejledningsopslag: Personligt og generel	Juni 2015
Eksportering af registrerede aktiviteter	M	Afsendelse af registrerede aktiviteter pr. E-mail i form af PDF eller tekst.	Juni 2015

Other Product Requirements

Requirement	Priority	Planned Release
Internet adgang	L	Juni 2015
Mobil telefon med Windows 8.1	H	Juni 2015
Indbygget brugervejledning	H	Juni 2015
Korrekt indstillet system ur	H	Juni 2015

1.4 Problemformulering

Den fremlagte og overordnede problemstilling for projekt er, frit oversat:

Hvordan kan Unified Process (UP), C#-programmering og relationelle databaser bruges,
i udvikling og implementation af et mindre single-user IT system?

Vi har også følgende underproblemstillinger i forbindelse med projektet:

Hvordan kan vi forbedre slutbrugerens oplevelse og motivere igennem Gameification?
- *Hvilke former for Gameification vil slutbrugeren finde acceptabel?*
- *Hvad kan lade sig gøre og hvad har vi kompetencer til at udføre?*

Hvordan kan vi skabe dynamiske interfaces i Windows Apps, med mindst mulig ressource forbrug?

Derudover har vi også nogle specifikke problemstillinger for selve produktet vi udvikler,
men de vil blive gennemgået i det følgende afsnit.

1.5 Projekt etablering:

Udviklingsteamets medlemmer er Jakob, Marc, Mathias og Nicolai.

Jakob ser sig selv, som værende en udmærket programmør, med et øje for problemløsning.
Han er forholdsvis lærenem og har derfor ikke noget imod at udforske nye muligheder,
praktisk såvel som teknisk, i et udviklingsprojekt.
Sidst men ikke mindst, går han meget op i velfungerende og sikre løsninger på problemstillinger.

Nicolai ser ham selv som en der vægter funktionalitet og brugervenlighed højest, men er dog ikke nået til det punkt han gerne ville inden for programmering.
Han er dog kommet længere end regnet med mht. design delen.
Nicolai har altid haft interesse indenfor programmering, og bruger det igennem hans arbejde.

Mathias ser sig selv som en kyndig programmør med speciale i web- og softwareløsninger, baseret på henholdsvis PHP og C#. Hans ambitioner for nye projekter er og vil altid være høje, da han sætter en lid i at lære og udfordre sig selv. Derfor er det essentielt for ham, at arbejde i et sundt og dynamisk miljø, med kollegaer der ligeledes har et ønske om at producere de bedste produkter.

Marc ser sig selv som en løsningsorienteret programmør der stiler mod at lave den mest brugervenlige, men samtidig funktionelle løsning. Erfaringsmæssigt har han arbejdet med Java, Python, Objective-C og senest C#. Hans ambition er at arbejde med mobil- og webudvikling og igennem dette udfordre og udvikle sig selv som programmør.

Vi er et dynamisk team der er i stand til at varetage mange forskellige opgaver,
i forbindelse med udviklingen af en applikation. Vi har et stort fokus på at de tekniske og

funktionelle aspekter af et projekt fungerer optimalt og efter hensigten.
Men vi går også op i at vores projekter bliver præsenteret med pæne og brugervenlige user-interfaces.

Hvordan skal vores team-samarbejde være?

De følgende regler for teamets samarbejde, er blevet nedfældet og vedtaget i fællesskab:

- Vi forventer at interne aftaler bliver overholdt!
- Don't be a dick, a.k.a, vi forventer at medlemmer af teamet kan finde ud af at opføre sig anstændigt over for hinanden!
- Vi forventer at medlemmerne af teamet er engagerede og aktive, uanset hvor uinspirerende opgaven er!
- Der skal altid være plads til at komme med sin mening, og det forventes at man kan diskutere frit og åbent i teamet!

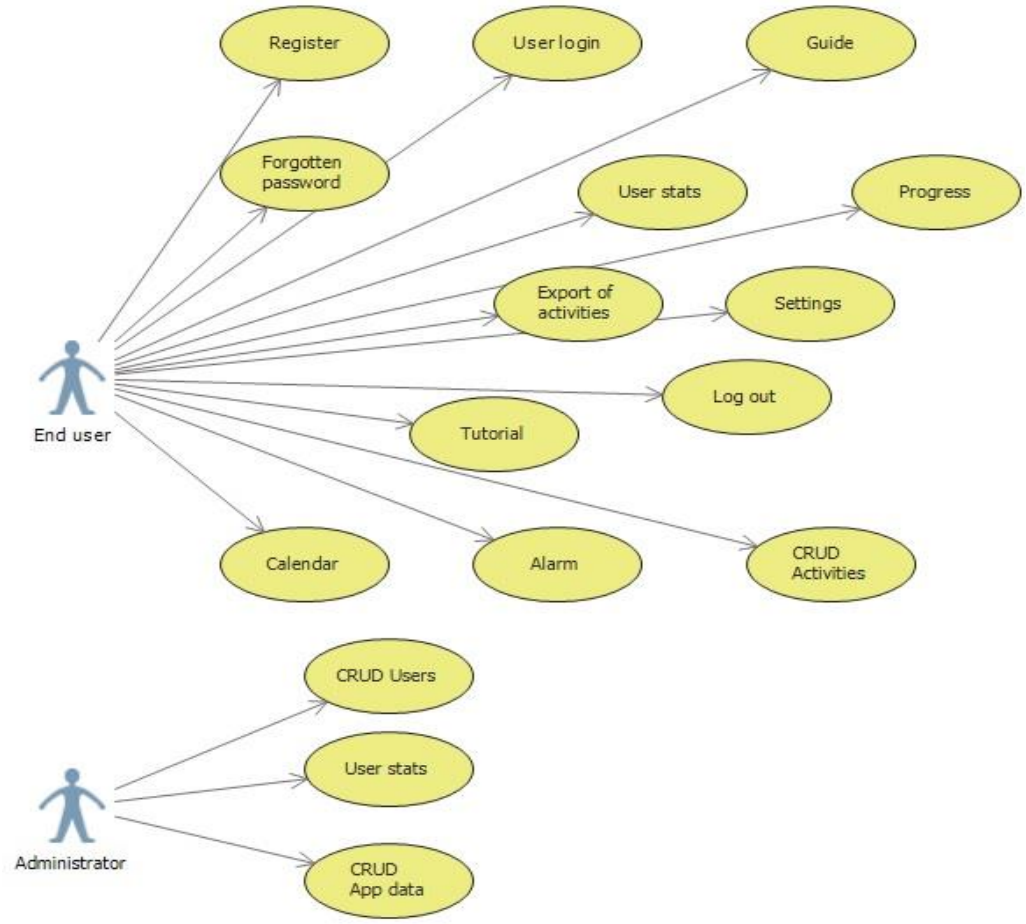
1.6 Risikoanalyse:

Nedenfor har vi forsøgt at kortlægge potentielle risici for projektet og dettes udvikling, for at kunne være forberedt såfremt disse situationer måtte opstå.

Magnitude (1-10)	Description	Impact	Indicator	Plan
1	Risiko for tab af data i forbindelse med ekstern hosting af data	C	Funktionalitet	Sikring af backup data, på endnu en ekstern server, separat fra den primære server, igennem automatiseret spejling af databasen.
3	Fravær baseret på sygdom eller dårlig moral	M	Produktions-hastighed	Uddelegere arbejdet imellem de resterende medlemmer af teamet, eller nedskalere ambitionerne for projektet, alt afhængigt af omstændighederne.
2	Tekniske udfordringer der forhindrer udvikling af features i produktet	H	Produktion	Nedskalere ambitionerne for projektet, alt afhængigt af omstændighederne.

1.7 Use Case Model & Text:

Ud fra de ønskede funktionaliteter fik vi etableret følgende Use Case Model



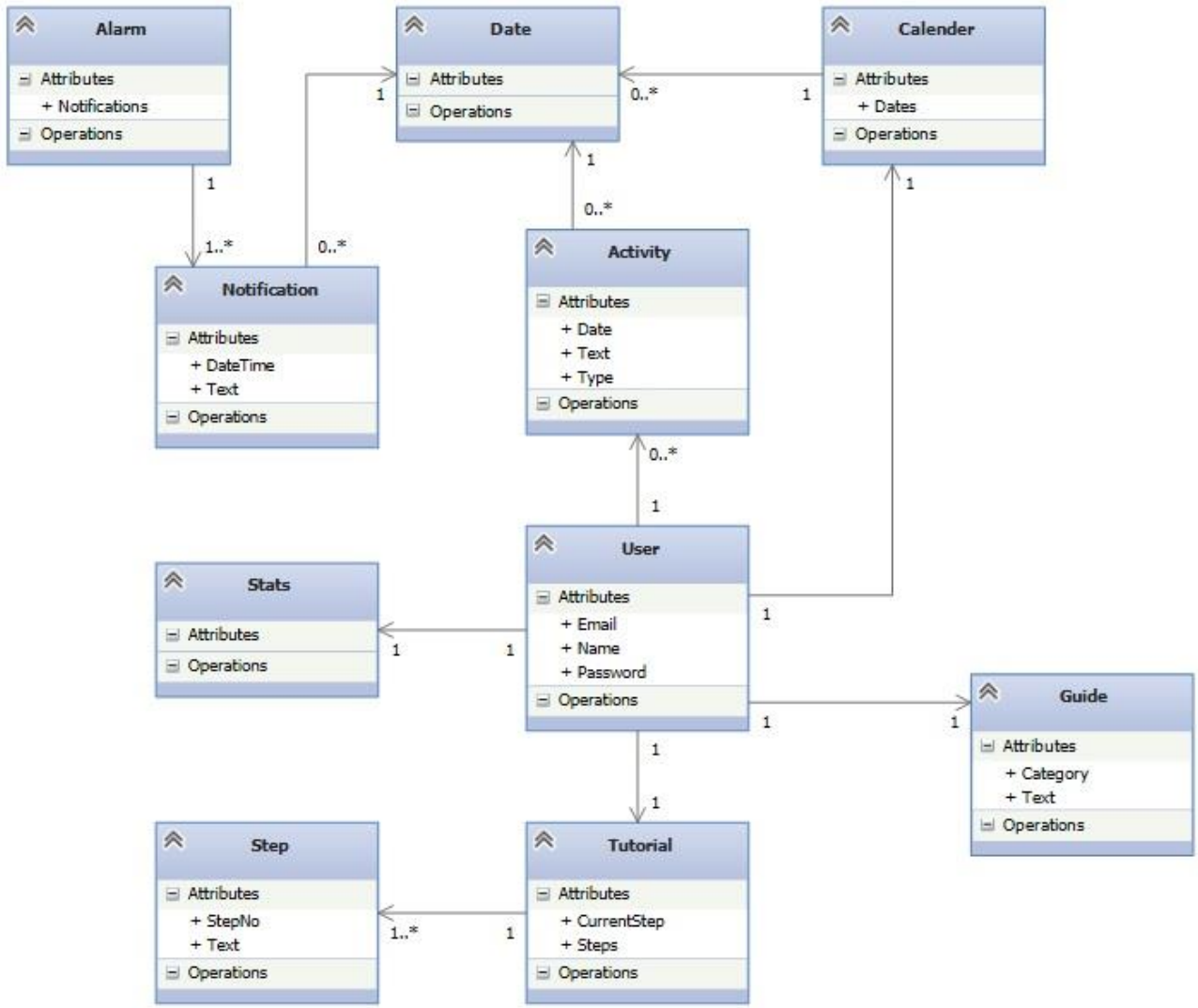
Derudover fik vi også etableret et par Fully Dressed Use Cases, samt nogle få Casual Use Cases. Herunder forefindes den af vores Fully Dressed Use Cases, som vi umiddelbart vurderede kom mest omkring i det ønskede system. Resten af vores Use Cases er vedlagt som bilag.

Use Case Name	Kalender
Scope	Ledighedsapp
Level	User goal
Primary Actor	Slutbruger
Stakeholders And Interests	Slutbruger og Symbiotic Solutions
Preconditions	Brugeren har indstillet uret på sin Windows Phone korrekt.
Success Guarantee	Brugeren får oprettet en kalender-note med eller uden alarm
Main Success Scenario	1. Brugeren er på appens mainpage og trykker på kalender-knappen 2. Brugeren trykker på en dato i kalender-griddet 3. Brugeren bliver præsenteret for en lille pop-up menu, der giver mulighed for at

	1) vise datoens aktiviteter, noter og alarmer 2) oprette note, aktivitet eller alarm 4. Brugeren trykker på ”Opret note” 5. Brugeren bliver navigeret til en ny side med en formular, hvor brugeren kan oprette en note. 6. Brugeren udfylder formularen 7a. Brugeren trykker ”Tilføj note” 1. Brugeren bliver navigeret tilbage til kalender-siden og kan nu se sin note oprettet på den valgte dato 7b. Brugeren trykker ”Tilknyt alarm” 1. Brugeren bliver navigeret til alarm interfacet 2. Brugeren indstiller en alarm for sin note 3. Brugeren trykker ”Tilføj alarm” 4. Brugeren bliver navigeret tilbage til ”Opret Note”-interfacet 5. Brugeren trykker på ”Tilføj note” 6. Brugeren bliver navigeret tilbage til kalender-siden og kan nu se sin note og sin alarm oprettet på den valgte dato
Extensions	Fejlscenariet tager udgangspunkt i udførslen af succes-scenariet punkt 7a eller 7b-5: 1. Brugeren har ikke indtastet noget i noten 2. Brugeren får returneret en fejlbesked 3. Brugeren bliver bedt om at indtaste en note
Special Requirements	UI-tekst skal være synligt ved en afstand på 34 cm
Technology and Data Variations List	
Frequency of Occurrence	Ugentligt, ved bruger-input
Miscellaneous	

1.8 Domæne Model

Her har vi fået defineret de konceptuelle klasser og den foreløbige struktur for projektet



1.9 Afrunding

Inceptionfasen er forløbet efter hensigten; der har været en masse diskussioner internt i gruppen i forhold til applikationens struktur, funktionalitet og udførslen af samme. Gruppen har været god til at holde målet for øje, samt respektere hinandens meninger, selvom stemningen indimellem har været presset.

Vi har, som afslutning på vores inceptionfase, præsenteret vores foreløbige arbejde for vores kunde, som godkendte størstedelen af vores artefakter.
Han var også i stand til at tage nogle beslutninger, på nogle af de punkter hvor vi internt har været uenige og givet os en retning at gå i, hvor vi har været i tvivl.
Derudover kom han her med et ønske om ekstra funktionalitet i vores system, i form af muligheden for at skifte sprog.

2. ELABORATION

Elaboration-fasen bruger man traditionelt til at få etableret arkitekturen i ens software-projekt, samt at få udarbejdet de relevante diagrammer, som f.eks. sekvens diagrammer og klasse diagrammer.

Sekvens diagrammer bruges til at redegøre for diverse aktørers interaktion med systemet, samt systemets interne interaktion med diverse systemklasser og metoder.

Klasse diagrammet har til formål at illustrere sammenhænge og forhold imellem de forskellige typer af klasser man måtte have etableret i et sekvens diagram, samt visuelt at skabe et overblik over systemets grundlæggende klasser.

Derudover arbejders der typisk, i elaboration-fasen, også med prototype-kode af højt prioriterede Use Cases og system-elementer

2.1 Artefakter

Traditionen tro, har vi startet elaboration-fasen med at udarbejde hhv. sekvensdiagrammer og et klassediagram. Vi undlod at lave system sekvens diagrammer, da vi umiddelbart fandt dem overflødige for udviklingen og vores egen forståelse for projektet.
Vi fik udarbejdet to omfattende sekvensdiagrammer ud fra vores fully dressed use cases. Det ene af disse er inkluderet herunder, og det andet er inkluderet som bilag.
Igennem udarbejdelsen af vores sekvensdiagrammer har vi bestræbt os på, så vidt som muligt, at gøre brug af de fem primære GRASP-patterns:

- *Creator*
- *Information expert*
- *Controller*
- *Low coupling*
- *High cohesion*

Creator-mønstret har til formål at uddele ansvaret for instantiering af objekter, så disse bliver oprettet i mest muligt relevante sammenhænge.
Information expert-mønstret går ud på at uddelegere ansvarsområder til klasser, der har den fornødne information, til at opfylde disse ansvar.

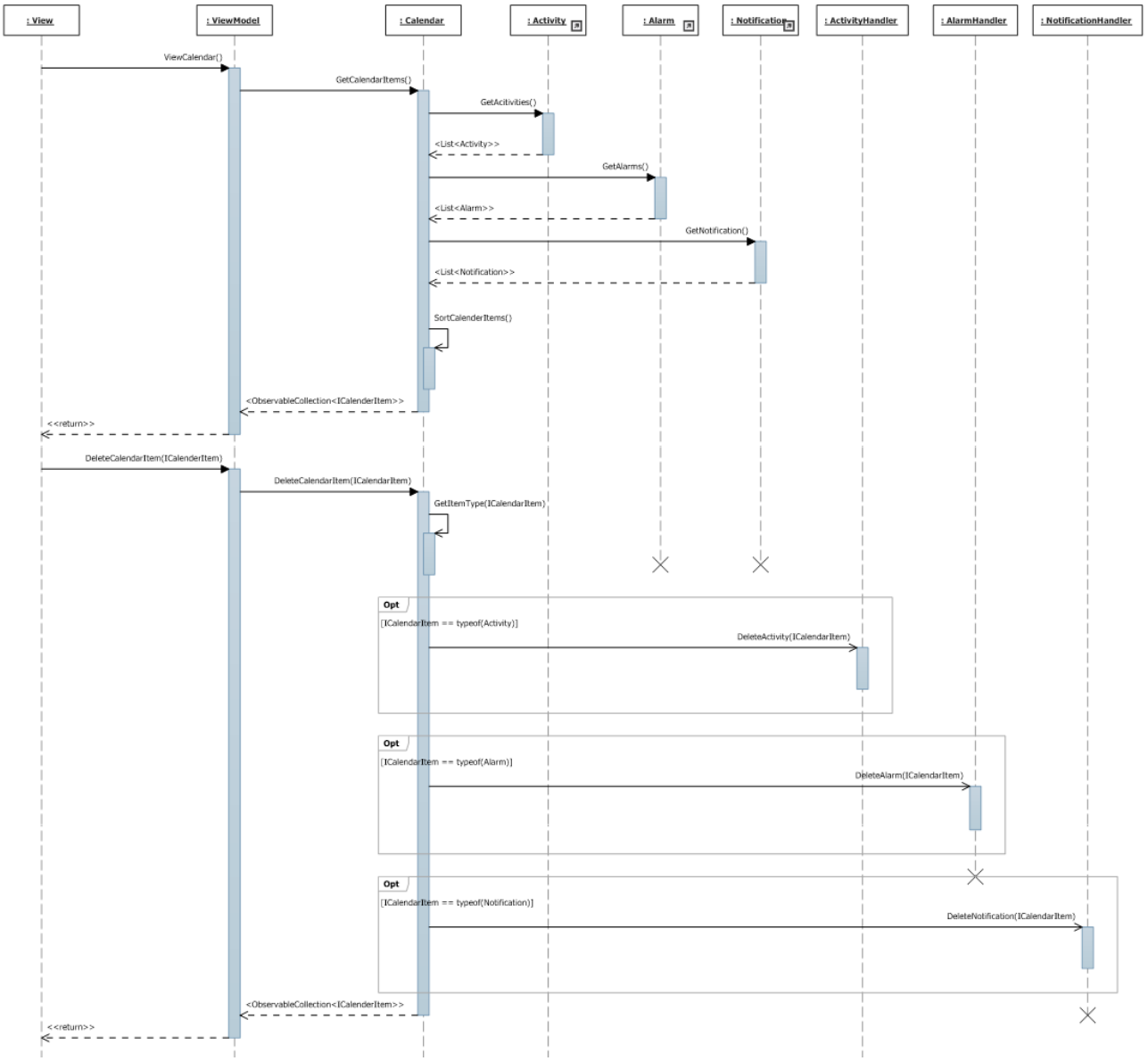
Disse to mønstre manifesterer sig i vores såkaldte ”Handler”-klasser, f.eks. NotificationHandler og ActivityHandler. Disse klasser tager sig af oprettelse og håndtering af specifikke datamodeller som f.eks. Notification og de forskellige typer af Activities i vores projekt.
Controller-mønstret har til formål at etablere et softwaremæssigt lag imellem User Interface og selve systemet som dette skal interagere med, for at kunne etablere mere kontrolleret interaktion

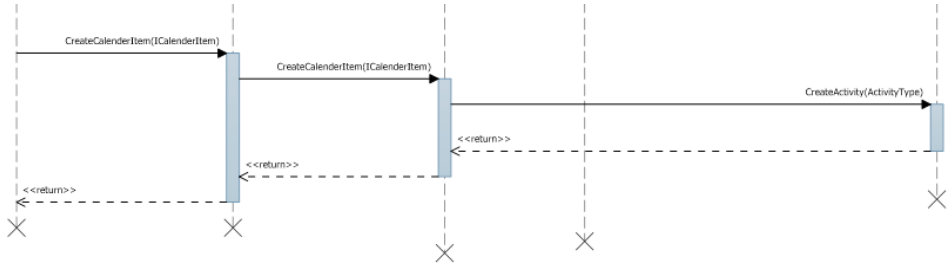
med systemklasser.

Dette mønster bruger vi i kraft af at vi benytter os af MVVM-arkitektur,
som vi vil komme ind på senere.
I MVVM-arkitekturen gør man brug af en såkaldt ViewModel, som netop opfylder betingelserne for *Controller*-mønstret.

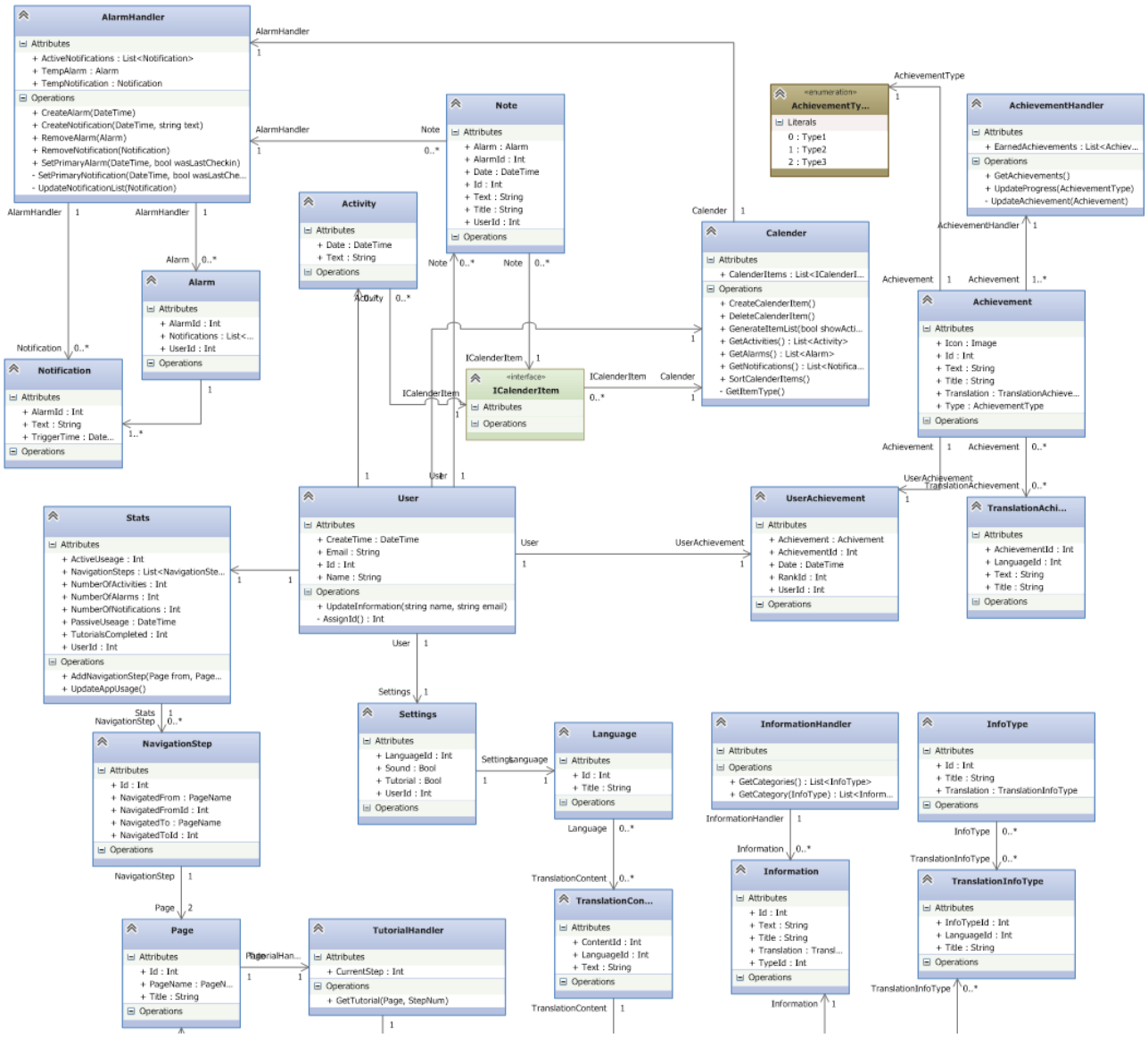
Low Coupling-mønstret sikrer at klasserne i ens system, så vidt muligt, er uafhængige af hinanden.
High Cohesion-mønstret har til formål at specialisere klasser og deres metoder til meget specifikke formål, så de kan genbruges i flest mulige sammenhænge.
Som udgangspunkt prøver vi altid at pålægge disse to mønstre i alt hvad vi laver,
da det sikrer at softwaren bliver nem at modificere på længere sigt.

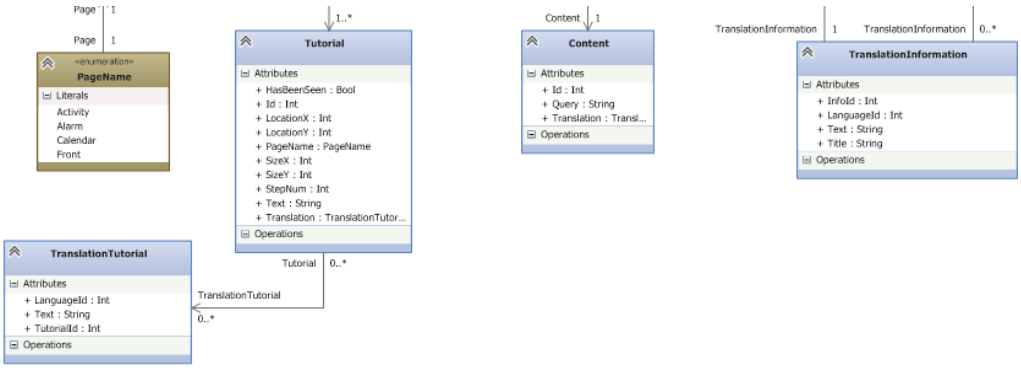
Nedenstående sekvensdiagram relaterer sig til vores ”Kalender” Use Case.





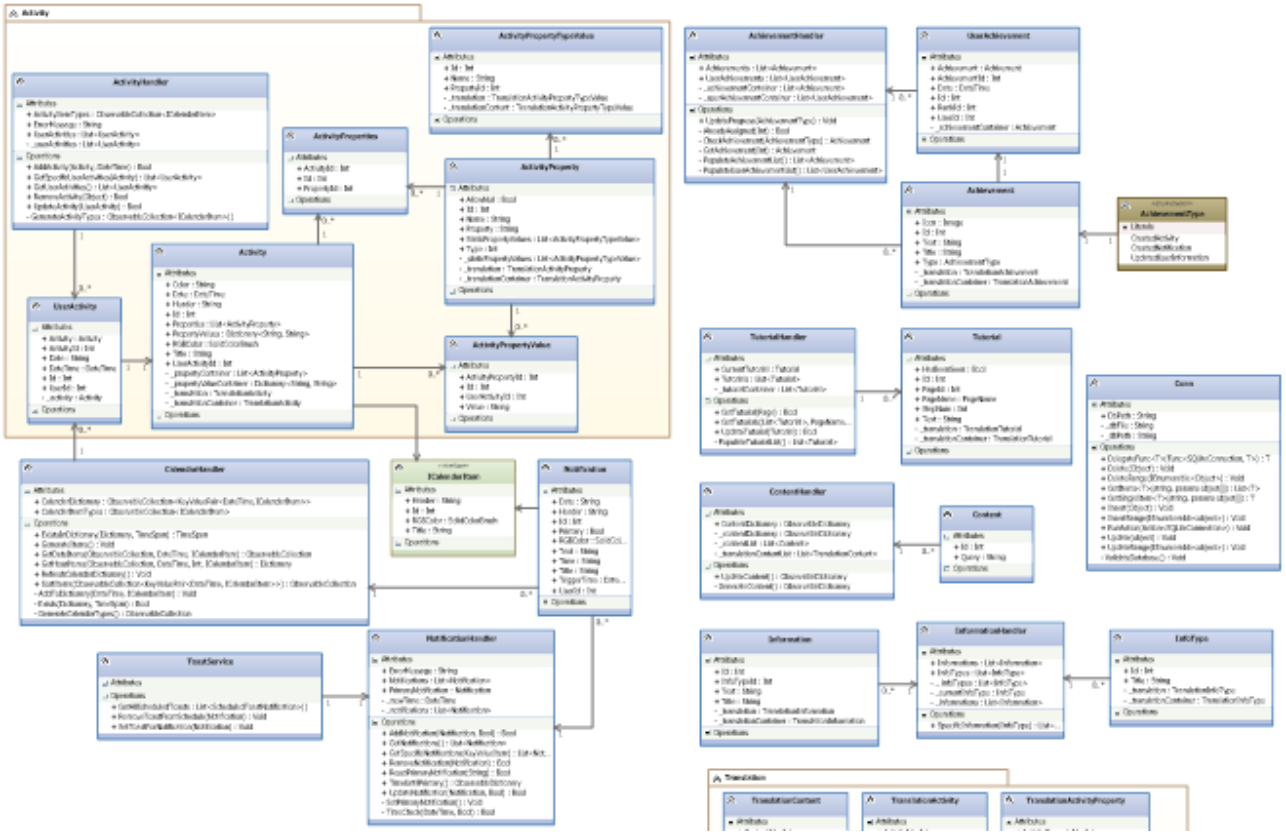
Næste skridt var, ud fra vores sekvensdiagrammer, at få udarbejdet første version af vores klassediagram, for derved at få etableret den grundlæggende arkitektur for vores projekt. Den første version af dette diagram er sat ind herunder.

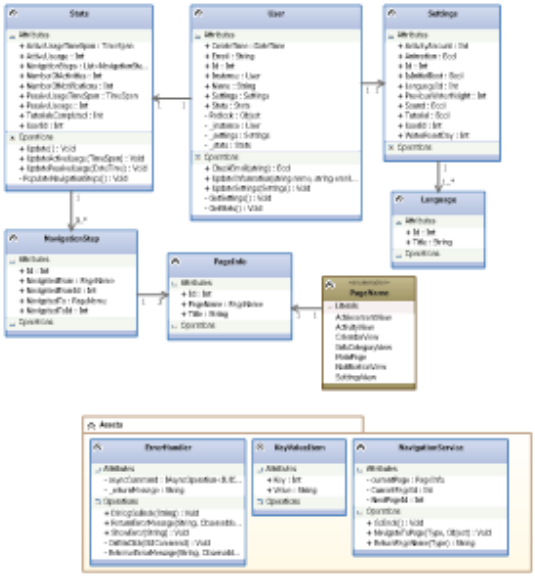


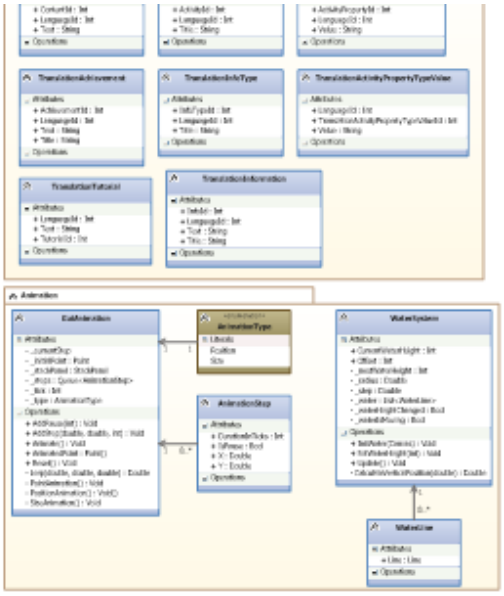


Som det igen kan ses, ud fra denne version af diagrammet, har vi benyttet os af såkaldte Handler-klasser til at håndtere arbejdet med større mængder af objekter. Vi har også benyttet os af et enkelt Interface, da det gav god mening at kunne arbejde med forskellige klasser, som én type, i forbindelse med vores kalender. Derudover har vi valgt at inkludere såkaldte ”Translation”-klasser, som implementerer oversættelser af tekst i applikationen, fra databasen.

Efter arbejdet med udviklingen af vores projekt, er vores klassediagram blevet modificeret og omstruktureret. Vi har for god ordens skyld inkluderet det nye diagram herunder, men eftersom størrelsen på diagrammet gør det svært læseligt i denne kontekst, er det også vedlagt som bilag i en større opløsning.







Størstedelen af ændringerne i diagrammet ligger i form af modifikationer på eksisterende klasser.
Undervejs i udviklingen stødte vi på diverse problemstillinger og mangler i vores oprindelige design, som gjorde at vi måtte udvide eller modificere eksisterende klasser.
F.eks. havde den oprindelige User-klasse blot 4 properties og 2 metoder.
I det opdaterede diagram er denne blevet udvidet, så den nu indeholder 11 properties og 5 metoder.

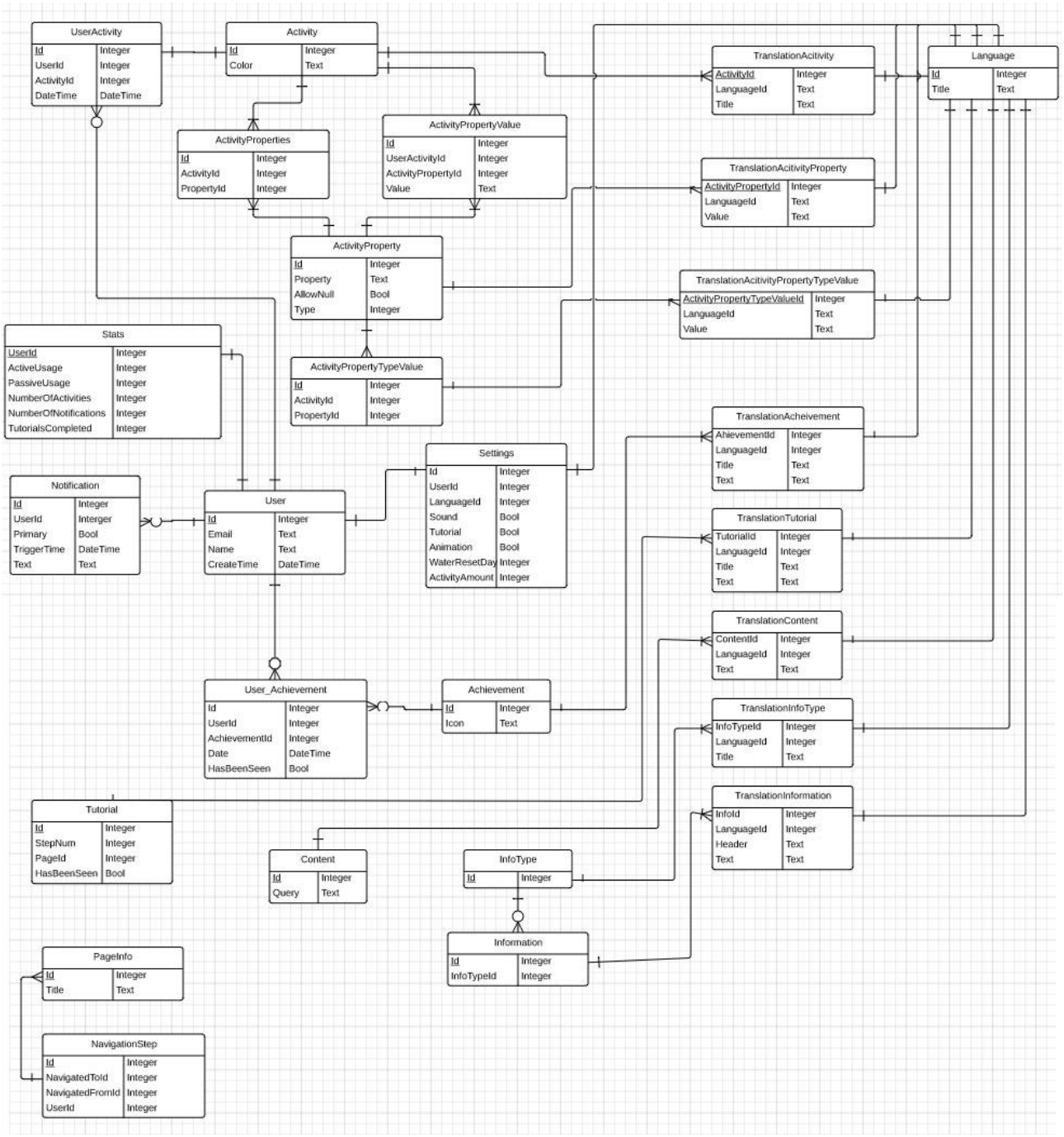
Derudover er der blevet slettet enkelte ting fra diagrammet. Alarm-klassen er blevet fjernet fuldstændigt, og AlarmHandler er blevet omdøbt til NotificationHandler.
Dette skyldes at kombinationen af AlarmHandler, Alarm og Notification gav et uønsket lag af kompleksitet i datastruktur såvel som brugervenlighed. Derved valgte vi at lægge os fast på en forenklet løsning i form af blot NotificationHandler og Notification.

Sidst men ikke mindst er der kommet en større udvidelse i systemet.
Da vi udviklede det oprindelige klasse-diagram, havde vi endnu ikke fået den nødvendige information, for at kunne skrive Activity-delen af diagrammet ind i dens fuldstændige udgave.
Derved bestod den oprindelige Activity-del blot af én enkelt klasse, med navnet Activity, med 2 properties. I det endelige diagram er dette blevet udvidet til hele 6 data-model klasser, med tilhørende Translation-klasser, samt en ActivityHandler-klasse.
Dette er alt sammen for at sikre et mest muligt dynamisk Activity-system, med mulighed for hurtigt og nemt at kunne oprette nye typer af aktiviteter med forskellige attributter.

I forbindelse med etableringen af vores systems arkitektur, har vi også fået optegnet et ER-diagram, for at kortlægge de forskellige relationer i vores, forholdsvis omfattende, database.
Dette diagram er indsat herunder og ligeledes vedlagt som bilag, for at læsere af denne rapport har mulighed for at nærstudere det.

I forhold til database-strukturen har vi forsøgt at opfylde normaliserings-formerne, for at undgå redundant data og isolere gentaget data i mellemtabeller.

Dette er beskrevet nærmere herunder.



Første normaliserings-form

Vi har sørget for at overholde den første normaliserings-form, ved at sørge for at vi aldrig lægger flere værdier, i samme celle i en tabel. For at undgå at dette skulle ske, har vi defineret primary keys i de nødvendige tabeller og oprettet en ekstra tabel som indeholder en værdi, samt en reference til primary key, i form af en foreign key.

Anden normaliserings-form

Her har vi tilfældet med f.eks. Achievement-tabellen, hvor den variable data, såsom ”Date” og ”HasBeenSeen”, er opbevaret i tabellen User_Achievement. Achievement er derfor en særskilt tabel med data, som kan anvendes i flere User_Achievements.
Her er strukturen fokuseret på et multi-user system, som nemt kan give os mulighed for at eksportere rå data og sammenligne denne med en ekstern database,
i form af et potentielt backend-system.

Det eneste sted hvor vi ikke har overholdt anden normaliseringsform, er i forbindelse med Tutorial-tabellen. Eftersom vores database bliver udviklet til såkaldt single-user brug, samt afvikles lokalt på brugerens enhed, har der ikke været noget behov for at skelne mellem flere forskellige instanser af Tutorials. Grundet dette har vi valgt at lagre variabelt data i tabellen selv.

Tredje normaliserings-form

Denne normaliserings-form er en struktur, hvor ingen af en tabels properties, som ikke er definerede som en candidate key, er afhængige af, eller kan relatere til hinanden.
Vi mener ikke at denne normaliserings-form gør sig gældende for vores database,
da vi ganske enkelt ikke har én tabel med non-key attributter, der er afhængige af hinanden.

EAV

Kunden har, i det endelige produkt, ønsket et fleksibelt aktivitets-system, hvor der dynamisk kan tilføjes nye aktiviteter med properties der adskiller sig fra eksisterende aktiviteter.
Dette kunne f.eks. være to aktiviteter i form af jobsøgning og møder, hvor registrering af en jobsøgning typisk ville indeholde langt mere specifik information, end en registrering af et møde. Derved opstår behovet for dynamiske properties.
For at imødekomme dette ønske har vi valgt at strukturere aktivitets/Activity-delen af databasen, efter EAV-mønstret.

EAV står for Entity Attribute Value, og går ud på at lagre properties i database-tabeller, fremfor at oprette specifikke tabeller pr. aktivitet. På denne måde kan man, uden at ændre i kode eller database-struktur, indføre nye typer af aktiviteter. Her følger en nærmere forklaring af de enkelte tabeller og deres rolle:

- User_Activity:

    Denne fungerer som mellemtabel for User og Activity-tabellerne.
    Her lagres information om den enkelte brugers forbindelse til aktiviteter.

- Activity:

Denne fungerer som skabelon for aktiviteter, hvis rækkers primary key bliver brugt til at definere dennes properties.

- ActivityProperty:

Denne definerer alle typer af properties, som en aktivitet kan implementere.

- ActivityProperties:

Denne tager en primary key fra en Activity, og kæder den sammen med en primary key fra ActivityProperty, hvorpå forskellige typer af properties tildeles forskellige aktiviteter.

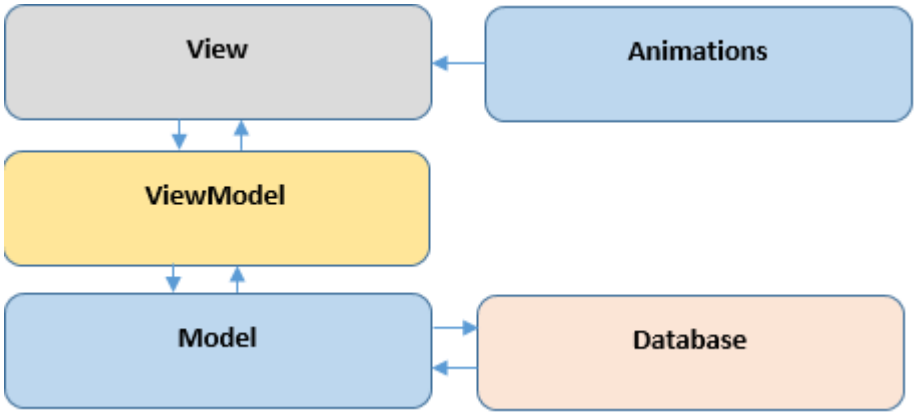
- ActivityPropertyValue:

Denne tager en primary key fra UserActivity, som tilhører en eksisterende aktivitet samt en primary key fra en ActivityProperty, ligeså tilhørende denne aktivitet. Herefter definerer value værdien for den pågældende property, tilhørende aktiviteten. Kort sagt; denne tabel angiver værdierne for brugerens aktiviteter.

I forbindelse med valg af database-teknologi, havde vi snakket om at basere vores database på en blanding af SQLite og MS SQL, for at kunne opbevare mest muligt data lokalt og have mulighed for at opbevare statistisk data på en ekstern server.
Men i den reelle udførsel af vores applikation, har vi valgt at fokusere på den lokale database, altså SQLite, eftersom vores applikation kommer til at fungere som en prototype for vores kunde.
Den eksterne opbevaring af statistisk data vil derfor ikke have relevans for denne prototype.
Valget af SQLite som vores database-teknologi kom af at vi, med Windows Phone-formatet, på daværende tidspunkt ikke kunne benytte os af Entity Framework.

Sidst men ikke mindst, har vi valgt, som den grundlæggende arkitektur, at tage udgangspunkt i MVVM strukturen, som står for Model, View, ViewModel.
Dette sørger for at adskille vores User Interface (View) fra vores systemklasser (Model), ved hjælp af en mellemliggende controller (ViewModel).

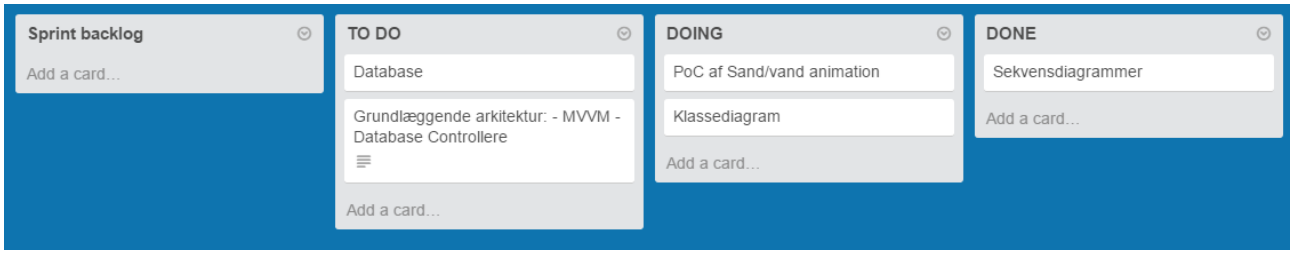
I vores variant, har vi dog taget højde for at vi vil gøre brug af animations-klasser, direkte forbundet med vores User Interface.



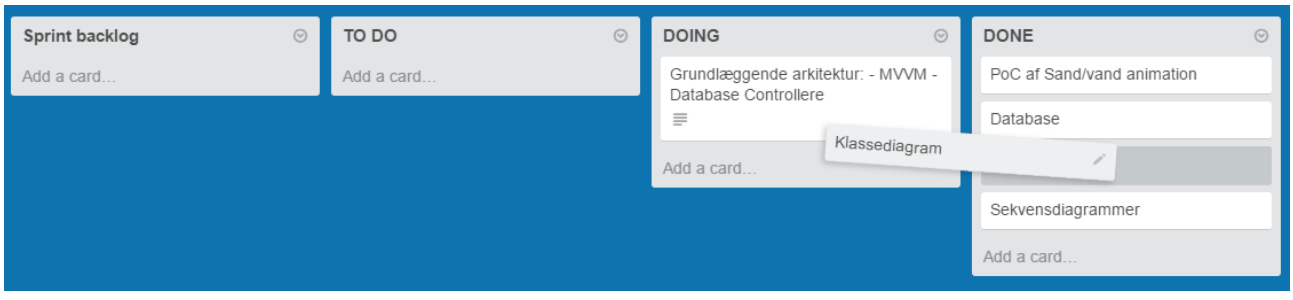
2.2 Brugen af SCRUM

Vi har, i gruppen, sørget for at holde daglige SCRUM-møder, hvor vi får lagt programmet for den pågældende dag og aftaler rækkefølgen for forskellige arbejdsopgaver.
I Inception-fasen var Jakob Grosen gruppens SCRUM-master og i Elaboration-fasen blev det Marc Forsmark der overtog posten.
Nedenfor har vi inkluderet to billeder af vores SCRUM-board.

SCRUM-board fra d. 22 April 2015:



SCRUM-board fra d. 27 April 2015:



Som det kan ses, ud fra vores SCRUM-board, så har vi i denne periode fået udarbejdet en række relevante diagrammer, fået etableret vores database og udarbejdet et proof of concept på vand-animation, vi regner med at skulle inkludere i vores endelige GUI.

2.3 Retroprospective

Fra vores kundes synspunkt har der ikke været nogle større ændringer eller konflikter i projektet, siden det møde der afsluttede vores Inception-fase.
Kunden har på dette tidspunkt været godt tilfreds med projektets forløb og den foreløbige arkitektur, såvel som udvalget af foreslåede features.
Derfor har vi været i stand til, i elaboration fasen, at arbejde meget koncentreret med at få sat detaljerede beskrivelser på vores diagrammer, samt internt at debattere de forskellige måder vi kan gribe projektet an på.

2.4 Sprint review

I denne fase har vi arbejdet rimelig koncentreret med at få etableret de nødvendige artefakter og få lagt grundstenene for vores projekt, så vi bedst muligt kunne komme i gang med construction-fasen. Dette har betydet at vi har haft en forholdsvis stor arbejdsbyrde i forbindelse med udarbejdelsen af vores diagrammer, da disse gerne skulle være så detaljerede som muligt for at give os mulighed for en god og struktureret start på næste fase.

Vi har derfor haft mange diskussioner og debatter undervejs, for at kunne blive enige om hvilken struktur vi umiddelbart så flest fordele ved.

I slutningen af elaboration-fasen, begyndte vi at etablere den grundlæggende arkitektur for vores projekt, samt databasen til samme.

I den forbindelse lærte vi, som tidligere nævnt, at vi ikke kunne benytte os af Entity Framework som vores database-teknologi. Dette har betydet vi har i sidste ende måtte oprette systemklasser for ordentligt at kunne spejle database-tabeller, og derved er projektet blevet modificeret let,
i forhold til hvad vi oprindeligt havde forestillet os.

I forhold til arbejdsindsatser har gruppen været samlet om størstedelen af arbejdet indtil videre.
Vi har udarbejdet diagrammer og rapport i fællesskab, imens etableringen af database struktur er blevet håndteret af Mathias Olsen.
Vi har derudover indgået en aftale omkring det videre arbejde i Construction-fasen,
da vi både kan se fordele i at arbejde selvstændigt og som en samlet gruppe.

Vi besluttede derfor at starten af vores Construction-fase skulle forløbe med at fordele arbejdsopgaver i fællesskab, for derefter at etablere deadlines for disse opgaver.
Herefter har det været op til det enkelte medlem af gruppen at få udført disse opgaver inden for tidsrammen. Vi havde derudover aftalt, såfremt vi arbejdede hjemme, at man så vidt så muligt ville sørge for at være tilgængelig på Skype, så man hurtigt og nemt ville kunne diskutere eventuelle problemstillinger. Derudover sørgede vi for jævnligt at holde Skype-møder, for at holde styr på projektets løbende fremgang.

3. CONSTRUCTION:

Vores første opgave i denne fase, var at få etableret de klasser vi havde tegnet op,
i det oprindelige klassediagram. Som udgangspunkt valgte vi at undlade brug af SCRUM-boards.
I stedet brugte vi en tilgang, hvor vi hver især frit kunne vælge hvilken klasse vi ville starte med at arbejde på. Fra og med den sidste halvdel af uge 18 satte vi dermed gang i udviklingen af Model-laget i vores projekt. Vi havde, i gruppen, aftalt at vi ville skyde efter at have indholdet af det oprindelige klasse-diagram færdigt efter ca. 1 uge, altså midt i uge 19.
Arbejdet endte med at blive strækket hen mod slutningen af uge 19, og omkring søndag d. 10 Maj følte vi os klar til at gå videre til næste fase i vores udvikling;
etablering og opsætning af Views og ViewModels.

I løbet af denne proces har der også foreligget nogle modifikationer af Model-laget,
da vi løbende fandt potentielle konflikter og smartere metoder, hvorpå vi kunne arbejde
med data imellem Model-lag og ViewModel-lag.
Arbejdet med View og ViewModel-lagene blev afrundet fuldstændigt d. 28. Maj.

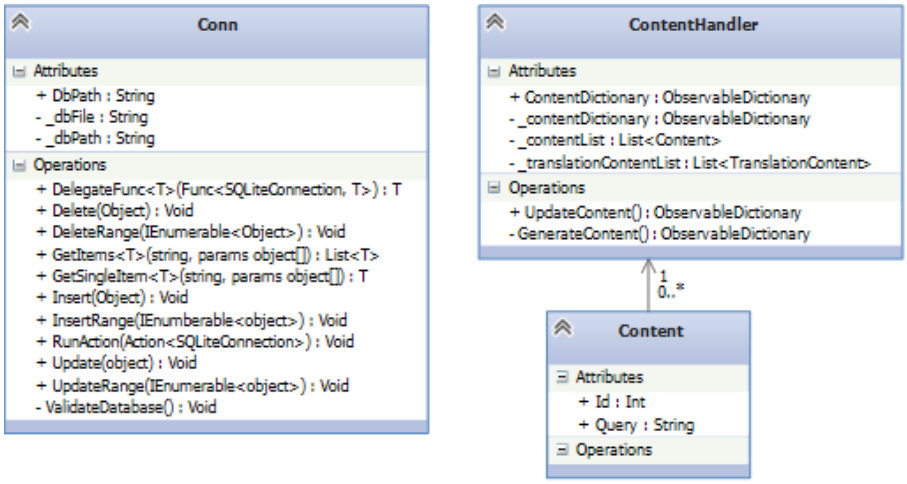
I vores udviklingsarbejde valgte vi at bruge Team Foundation Server til versions-styring af projektet, da dette på forhånd er integreret i Visual Studio og ville give os nogle fordele,
ifht. til hurtigt at kunne integrere ændringer i vores system.

I det følgende afsnit foreligger en række eksempler fra vores Model-lag, i form af mindre udsnit fra vores endelige klassediagram, samt eksempler på kode fra klasser og metoder, som vi i gruppen mener har interessante og/eller vigtige funktioner i vores system.

3.1 Udpluk fra Model-laget

Med undtagelse af Handlers og diverse statiske klasser, er alle klasser i vores Model-lag sat op til at spejle database-strukturen, så vi nemmest muligt har kunnet arbejde objekt-orienteret
med data fra databasen. Derudover indeholder de fleste datamodeller i vores projekt også en property med oversættelser af de statiske værdier, bl.a. i form af Activity-klassen
og dens tilhørende TranslationActivity-klasse. Dette har været essentielt for dynamisk at kunne ændre sproget i vores endelige applikation.

Det første eksempel fra Model-laget vi vil tage fat på, er hhv. Conn, Content og ContentHandler-klasserne, som er inkluderet herunder, i form af et udsnit fra vores klassediagram.



I forbindelse med disse klasser har Mathias stået for etableringen og implementationen af klasserne, som han gennemgår mere detaljeret herunder.

Conn, Content & ContentHandler

For at kunne kommunikere med SQLite frameworket i vores system, har vi valgt at introducere en decorator klasse, som har til ansvar at forenkle vores sql queries, sikre åbning og lukning af database connections, samt at spejle vores table-data til systemobjekter vha. ORM.

Nedenfor ses en dokumentation af vores decorator klasse ”Conn”.

1
```
private static string _dbFile = "Storage.db";
private static string _dbPath = string.Empty;
public static string DbPath
{
    get
    {
        if (string.IsNullOrEmpty(_dbPath))
        {
            ValidateDatabase();
            _dbPath = Path.Combine(ApplicationData.Current.LocalFolder.Path, _dbFile);
        }
        return _dbPath;
    }
}
```

2

```
private static async void ValidateDatabase()
{
    bool isDatabaseExisting;
    try
    {
        StorageFile storageFile = await ApplicationData.Current.LocalFolder.GetFileAsync(_dbFile);
        isDatabaseExisting = true;
    }
    catch (Exception ex)
    {
        isDatabaseExisting = false;
    }

    if (!isDatabaseExisting)
    {
        StorageFile databaseFile = await Package.Current.InstalledLocation.GetFileAsync(_dbFile);
        await databaseFile.CopyAsync(ApplicationData.Current.LocalFolder);
    }
}
```

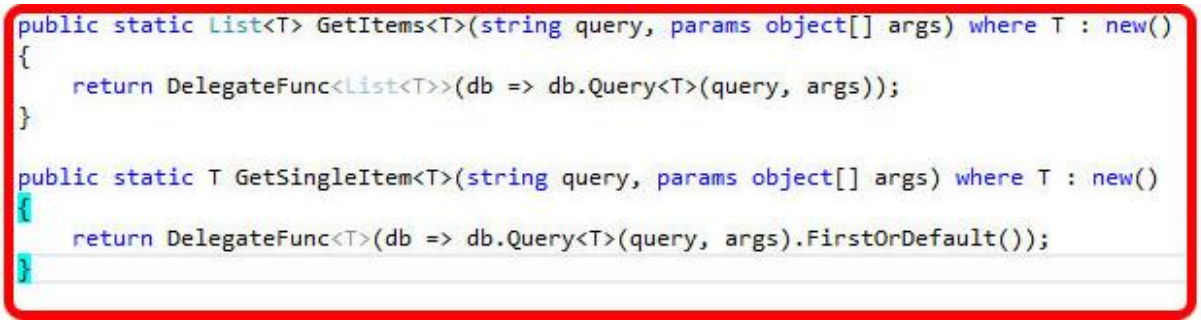
1. String propertien DbPath indeholder stien til databasen, og det vil derfor være den der kaldes, hver gang en forbindelse til databasen skal etableres.
   Dens get metode, tjekker om stien allerede er sat, hvis ikke så kaldes ValidateDatabase metoden.

2. ValidateDatabase metoden tjekker først om sqlite filen eksisterer i telefonens hukommelse (ApplicationData.Current.LocalFolder). Hvis dette ikke er tilfældet, henter den databasen fra system mappen (Package.Current.InstalledLocation) og kopierer den til telefonens hukommelse. Årsagen til dette, er at sqlite filen i system-mappen er skrivebeskyttet, og dens instanser (evt. Med nye ændringer) derfor ikke kan gemmes igen.

3

```
public static T DelegateFunc<T>(Func<SQLiteConnection, T> funcToRun) where T : new()
{
    using (var db = new SQLiteConnection(DbPath))
    {
        db.Trace = true;
        return funcToRun(db);
    }
}

public static void RunAction(Action<SQLiteConnection> actionToRun)
{
    using (var db = new SQLiteConnection(DbPath))
    {
        db.Trace = true;
        db.BeginTransaction();
        actionToRun(db);
        db.Commit();
    }
}
```

4



```
public static List<T> GetItems<T>(string query, params object[] args) where T : new()
{
    return DelegateFunc<List<T>>(db => db.Query<T>(query, args));
}

public static T GetSingleItem<T>(string query, params object[] args) where T : new()
{
    return DelegateFunc<T>(db => db.Query<T>(query, args).FirstOrDefault());
}
```

For ikke at skulle gentage kode i vores decorator klasse og samtidig sikre at vores database forbindelse bliver lukket korrekt, har vi valgt at oprette to klasser, som tager imod henholdsvis en metode (Func) og an Action.

3. DelegeateFunc<T> tager imod en metode med parametret af typen SQLiteConnection og returnerer samme type som den delegerede metode vha. Type T tagget.
   Det interessante ved denne metode, er derfor at den kan håndtere andre metoder med forskellige return types og eksekvere dem igennem sin egen kode.

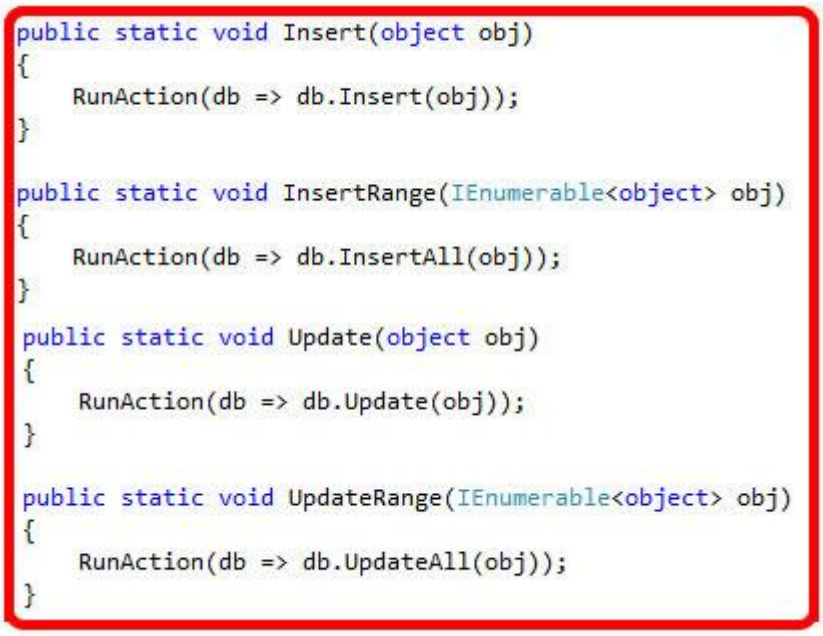
   I eksemplet ovenfor kan man se at DelegateFunc har returntype T, hvilket fortæller os, at den returnerer et objekt af samme type, som den kaldes med. ”where T : new()”, angiver at T skal returnere et nyt objekt af denne type.

4. GetItems metoden bygger på samme princip som DelegateFunc. Dette en af to metoder vi kalder i modellaget, når vi eksekverer SELECT queries igennem vores DBMS.
   Metoden kaldes med en specifik type (T) og returnerer en List af samme type (T), i dens krop kaldes DelegateFunc metoden med argumentet db => db.Query<T>(query, args).
   I argumentet angiver vi en variabel (db) af typen SQLiteConnection, og kalder dens metode Query<T>(query, args). Dette håndteres så efterfølgende af DelegateFunc

   Hvis man kigger tilbage til DelegateFunc, kan man se at dens krop sørger for at åbne en database forbindelse baseret på det første argument (db) og returnerer derefter resultatet af den delegerede metode Query<T>(query, args).

   Et eksempel på brugen af metoden er følgende:
   List<Activity> Aktiviteter =
   Conn.GetItems<Activity>(”SELECT * FROM Activity WHERE UserId = ?”,
   User.Instance.Id);

5



```
public static void Insert(object obj)
{
    RunAction(db => db.Insert(obj));
}

public static void InsertRange(IEnumerable<object> obj)
{
    RunAction(db => db.InsertAll(obj));
}
public static void Update(object obj)
{
    RunAction(db => db.Update(obj));
}

public static void UpdateRange(IEnumerable<object> obj)
{
    RunAction(db => db.UpdateAll(obj));
}
```

5. Et andet eksempel på delegerede metoder er at vi har oprettet ovenstående metoder, til at udføre SQL queries der ikke returnerer en værdi.
   Ovenstående metoder tager imod en liste eller enkelte objekter, som videresendes til SQLite metoderne Insert, Update og Delete. Disse metoder delegeres videre til RunAction, som sørger for at etablere database forbindelsen inden de udføres.

ContentHandlerens formål er at generere en liste af oversættelser af statisk tekst i systemet, baseret på slutbrugerens valgte sprogindstilling. Grundet at klassens indhold skal bruges på alle views i applikationen, er det vigtigt at dataene ikke skal hentes hver eneste gang, de skal bruges.

1

```
public static ObservableDictionary ContentDictionary
{
    get
    {
        if (_contentDictionary == null)
            _contentDictionary = GenerateContent();
        return _contentDictionary;
    }
}

private static ObservableDictionary _contentDictionary = null;
private static List<Content> _contentList;
private static List<TranslationContent> _translationContentList;

private static ObservableDictionary GenerateContent()
{
    _contentList = Conn.GetItems<Content>("SELECT * FROM Content");
     string sql =
                @"SELECT * FROM TranslationContent
                        INNER JOIN Settings ON Settings.LanguageId = TranslationContent.LanguageId";
    _translationContentList = Conn.GetItems<TranslationContent>(sql);
    ObservableDictionary tempDictionary = new ObservableDictionary();
    foreach (Content obj in _contentList)
        tempDictionary.Add(obj.Query, _translationContentList.FirstOrDefault(data => data.ContentId == obj.Id).Text);
    return tempDictionary;
}
```

1. Vi har valgt at placere vores oversættelsesdata fra databasen i en ObservableDictionary så vi på alle views kan binde direkte til den samme property.
   ContentDictionary's get metode tjekker om oversættelserne allerede er hentet, hvis ikke, eksekveres GenerateContent metoden.

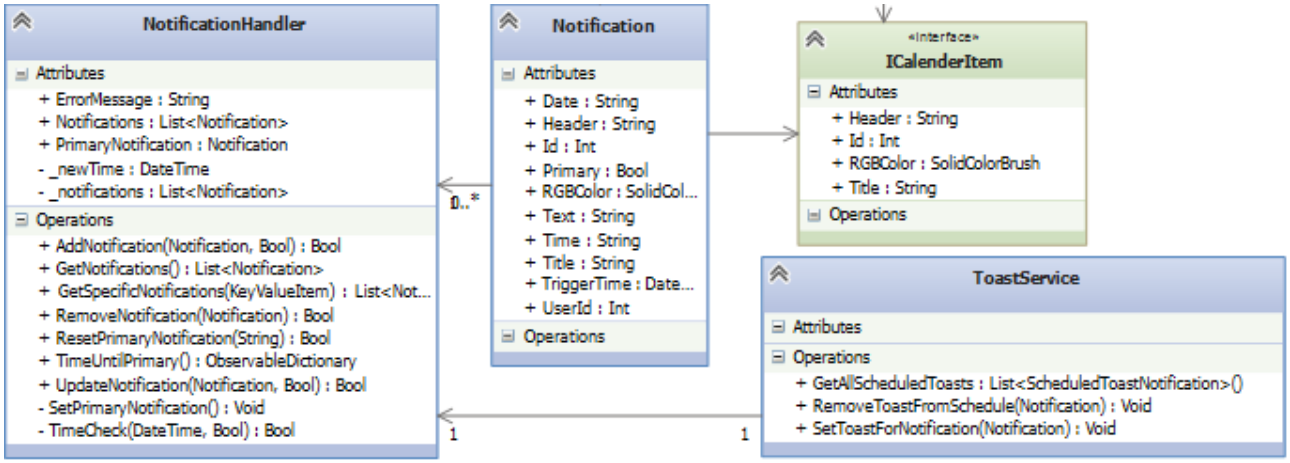
   GenerateContent henter alle Content query navne, som bruges til at definere nøglerne til ContentDictionary.
   Derefter hentes alle oversættelser der passer til brugerens sprogindstilling og lagres som værdier til de pågældende nøgler.

   ContentDictionary bruges i "Contents" viewmodellen, som senere bliver nedarvet til alle viewmodels. Dette er med til at sikre at der er adgang til oversættelserne på alle views hvor der er tilknyttet en viewmodel.

   Et eksempel på brugen af oversættelserne er følgende:
   ViewModel:
   ObservableDictionary Content = ContentHandler.ContentDictionary;

   View:
   <TextBlock Text={Binding Content[HEADER_TEXT]} />

Det næste eksempel fra vores Model-lag er kombination af Notification, NotificationHandler samt ToastService-klasserne.
Disse klasser er vist i et udsnit fra vores klassediagram, som kan ses herunder.
Inkluderet i udsnittet er også vores ICalenderItem-interface der sørger for at gøre vores Notification-klasse kompatibel med vores CalendarHandler-klasse.



I denne forbindelse har Marc stået for udviklingen af ToastService-klassen, og givet en gennemgang af klassen herunder. Da Mathias har stået for etableringen af NotificationHandler har

han herefter lavet en lille gennemgang af essentielle funktioner i NotificationHandler-klassen, samt dennes relation til ToastService.

ToastService

ToastService klassen er en hjælperklasse der har til opgave at fungere som en adapter mellem vores models implementation af Notification og Windows Phones eget notifikation system, kaldet Toast Notifcations.

Toasts findes i flere forskellige udgaver afhængige af hvornår/hvordan de skal vises og hvorfra deres indhold kommer. I vores applikation har vi brug for at bruge den type der kaldes ScheduledToastNotification, da vi skal kunne planlægge hvornår notifikationen skal vises og hvad indholdet er på forhånd og fra telefonen selv.

Den første metode i ToastService har til formål at planlægge en Toast Notification givet en Notification fra vores eget system.

```
public static void SetToastForNotification(Notification notification)
{
    string text = notification.Text;

1   var toastTemplate = ToastTemplateType.ToastText01;
    XmlDocument toastXml = ToastNotificationManager.GetTemplateContent(toastTemplate);
    XmlNodeList toastTextElements = toastXml.GetElementsByTagName("text");

    toastTextElements[0].AppendChild(toastXml.CreateTextNode(text));
2   var scheduledToast = new ScheduledToastNotification(toastXml, notification.TriggerTime)
    {
        Id = notification.Id.ToString()
    };

3   ToastNotificationManager.CreateToastNotifier().AddToSchedule(scheduledToast);
}
```

Indholdet af en Toast Notification er defineret ud fra et XML dokument der indeholder en række linjer tekst. Der er række skabeloner at vælge imellem som varierer hvor mange linjer tekst der er og hvilke der fed tekst. Vi har valgt den simpleste skabelon hvor ingen tekst er fed. I kode del 1 fra ovenstående figur foregår netop dette, da vi henter en template og konstruere et XML dokument ud fra det. Til sidst gemmer vi en reference til den del af XML dokumentet der indeholder teksten.

I del 2 starter vi med at tilføje et element til listen af tekst i XML dokumentet, med tekstindholdet fra den givne Notification fra vores eget system. Nu har vi alt hvad vi skal bruge, så vi instantierer et objekt af typen ScheduledToastNotification og giver den XML dokumentet vi har konstrueret, et tidspunkt den skal vises (taget fra vores givne Notification) og et Id så vi kan referere til den igen på et senere tidspunkt.
I del 3 giver vi vores ScheduledToastNotification til ToastNotificationManager,
som sætter den op i telefonens system.

Den anden metode i ToastService klassen har til formål at slette en ScheduledToastNotification givet en tilsvarende Notification fra vores eget system.

```
public static void RemoveToastFromSchedule(Notification notification)
{
    string toastId = notification.Id.ToString();

1   ScheduledToastNotification toastToBeRemoved = ToastNotificationManager.CreateToastNotifier()
                                                  .GetScheduledToastNotifications().FirstOrDefault(x => x.Id == toastId);

2   if (toastToBeRemoved != null)
    {
        ToastNotificationManager.CreateToastNotifier().RemoveFromSchedule(toastToBeRemoved);
    }
}
```

Kode del 1, i ovenstående kodeeksempel, starter med at finde en reference til ScheduledToastNotification. Dette gøres ved at søge listen af planlagte notifikationer, givet fra ToastNotificationManager, igennem med FirstOrDefault-metoden.

Kode del 2, fjerner så den fundne ScheduledToastNotification fra telefonens system, givet at den blev fundet og derfor ikke er null.

NotificationHandler

NotificationHandlers formål er at håndtere CRUD på sekundære alarmer, såvel som at vedligeholde den primære alarm. Nedenfor vises et eksempel på at oprette en ny alarm i systemet.

```
1   public bool AddNotification(Notification notification, bool lastCheckIn = false)
    {
        try
        {
            if (String.IsNullOrEmpty(notification.Text) && !notification.Primary)
                throw new Exception("ERROR_FILL_ALL_FIELDS");

            if (notification.Primary)
                if (PrimaryNotification != null)
                throw new Exception("ERROR_PRIMARY_ALREADY_EXISTS");

            if (lastCheckIn && notification.TriggerTime > DateTime.Now)
                throw new Exception("ERROR_PAST_TIME");

            if (!lastCheckIn && notification.TriggerTime < DateTime.Now)
                throw new Exception("ERROR_FUTURE_TIME");

            if (!TimeCheck(notification.TriggerTime, lastCheckIn))
                    throw new Exception("ERROR_TIME_INCORRECT");

            if (lastCheckIn)
                notification.TriggerTime = NewTime;
```

```
        Conn.Insert(notification);
        ToastService.SetToastForNotification(notification);

        if (notification.Primary)
            SetPrimaryNotification();
        else
            AchievementHandler.UpdateProgress(AchievementType.CreatedOneNotification);
    }
    catch (Exception ex)
    {
        ErrorMessage = ErrorHandler.ReturnErrorMessage(ex.Message);
        return false;
    }
    return true;
```

AddNotification er en metode med public access modifier, som returnerer en bool og tager imod to argumenter.
Det første argument er af typen notification, hvilket betyder at metoden arbejder ud fra en allerede instantieret notification. Dette foregår i ViewModel laget, hvor en alarm (notification) bindes direkte til et Views tekstbokse, og derfra opretter en midlertidig notification.
Det næste argument af typen bool, har en default værdi ”false”, grundet at det kun bruges, hvis der arbejdes med en primær alarm. (Notification har en bool property kaldet Primary).

Metodens krop køres igennem en try-catch, for at sikre håndtering af eventuelle fejl der måtte opstå fra brugerens såvel som systemets side af.
I den øvre del af try blokken, kastes der manuelt exceptions, hvis notifications properties ikke stemmer overens med systemets krav. Disse exceptions bliver herefter fanget i catch blokken og konverteres igennem ErrorHandler, til den korrekte oversættelse af fejlen.

Hvis ingen af kriterierne for de manuelle exceptions bliver mødt vil følgende blive udført:
- Notification objektet bliver indsat i databasen igennem Conn klassens Insert metode.
- Der oprettes en ”Toast” vha. ToastService klassen, som udløser en alarm i telefonen på det pågældende tidspunkt alarmen er sat til.
- Hvis notification er af typen Primary, vil SetPrimaryNotification blive udført, som sørger for at håndtere den primære alarm korrekt.
- Sidste men ikke mindst, tjekker systemet om den nye oprettelse af en notification er nok til at udløse en achievement. Dette sker igennem AchievementHandler klassen UpdateProgress metode.

2

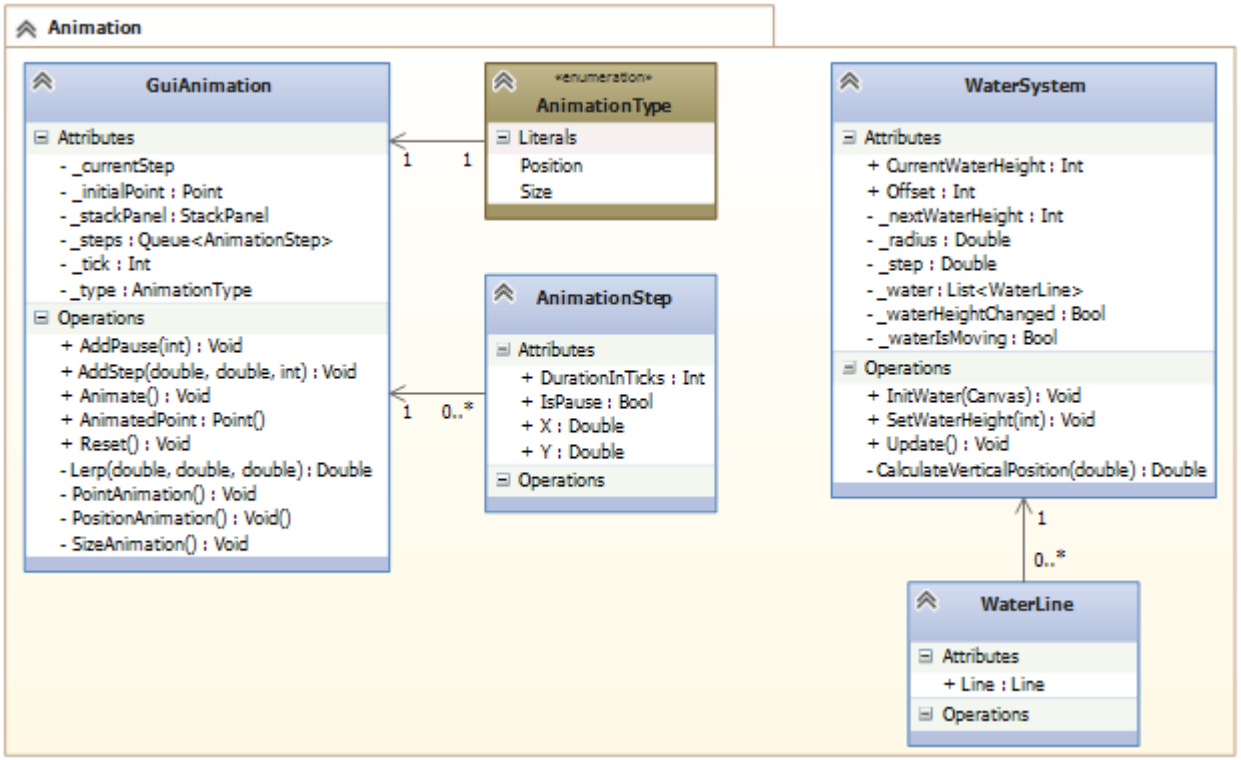
```
private bool TimeCheck(DateTime time, bool lastCheckIn)
{
    if (lastCheckIn)
    {
        if (time <= DateTime.Now)
        {
            bool tempCheck = true;
            while (tempCheck)
            {
                time = time.Add(new TimeSpan(7, 0, 0, 0));
                if (time > DateTime.Now)
                    tempCheck = false;
            }
        }
        NewTime = time;
        return true;
    }

    if (time <= DateTime.Now)
        return false;
    return true;
}
```

Et eksempel på en af de kriterier der skal opfyldes, når en ny notification oprettes, er TimeCheck metoden. Metoden går i alt sin enkelhed ud på at tjekke om tidspunktet der er knyttet til den pågældende notification er af korrekt format og tidspunkt.
På ovenstående billede ses der at metoden tager imod et DateTime argument samt en bool (lastCheckIn bliver overført fra AddNotification metoden).

Metoden sørger for følgende:
- Hvis lastCheckIn er true, vil metoden opdatere DateTime argumentet indtil at dets værdi er sat til et tidspunkt efter nu og derefter returnere true.
- Hvis lastCheckIn er false forespørges der om DateTime argumentet er før nu og returnerer en bool efter det.

Animation

Billedet herunder viser en mindre samling af klasser, udarbejdet af Jakob, der håndterer forskellige animations-relaterede opgaver i vores endelige applikation.



GuiAnimation-klassen er Jakobs forsøg på at lave et forsimplet og skræddersyet alternativ til Storyboard og DoubleAnimation-klasserne, der eksisterer i det oprindelige framework.
Grundlaget for at lave denne klasse, var at vi fandt det meget omstændigt at bruge Storyboard og DoubleAnimation til at håndtere mange af de animerede elementer vi ville have med i projektet.

Brugen af Storyboard og DoubleAnimation kan kræve rigtig mange linjers kode for at opnå forholdsvis simple animationer, hvis disses parametre også skal være variable.
Derfor var målet at lave en klasse der ville kunne bruges i de fleste animations-sammenhænge, uden at kræve for meget kode ved instantieringen af nye animationer.

Fleksibiliteten ift. instantieringen ses bl.a. i form af klassens 2 constructors.
Dette gør at man kan oprette et objekt af GuiAnimation i 2 forskellige tilstande;

1

```
20 references
public GuiAnimation(StackPanel stackPanel, AnimationType type)
{
    _stackpanel = stackPanel;
    _type = type;
    IsDone = false;
    _initialPoint = new Point(Canvas.GetLeft(stackPanel), Canvas.GetTop(stackPanel));
}
```

2

```
4 references
public GuiAnimation(Point initalPoint)
{

    IsDone = false;
    _initialPoint = initalPoint;
}
```

Constructor nr. 1 binder objektet til et specifikt StackPanel, og en specifik animations-type ud fra vores AnimationType-enum. Dermed arbejder animationen på et specifikt StackPanel og modificerer enten størrelsen eller positionen af dette, alt efter valget af animations-type.

Constructor nr. 2 tager blot imod et objekt af typen Point, der indikerer X og Y-værdier for animationens startpunkt. Denne type instantiering binder sig derfor ikke til et specifikt StackPanel eller andet UIControl, men lægger derimod op til at man henter behandlede værdier ud igennem klassens AnimatedPoint() metode, som returnerer et objekt af typen Point.
AnimatedPoint() indeholder ikke logik der skal differentiere imellem at modificere størrelse eller positions-værdier. AnimatedPoint()-metoden er en forenklet udgave af klassens Animate()-metode, som er indsat herunder.

```
public void Animate()
{
```

1

```
    if (_currentStep == null&&_steps.Count>0)
    {
        _currentStep = _steps.Dequeue();
    }

    if (_tick == _currentStep.DurationInTicks && _steps.Count > 0)
    {
        _currentStep = _steps.Dequeue();
        _tick = 0;
    }
    else if (_tick == _currentStep.DurationInTicks && _steps.Count == 0)
    {
        IsDone = true;
    }
```

2

```
        if (_tick < _currentStep.DurationInTicks)
        {
            if (!_currentStep.IsPause)
            {
                switch (_type)
                {
                    case AnimationType.Position:
                        PositionAnimation();
                        break;
                    case AnimationType.Size:
                        SizeAnimation();
                        break;
                }
            }
            _tick++;
        }
    }
```

Som punkt 1 i ovenstående billede viser, har Animate()-metoden, kort forklaret, ansvaret for at hente og afvikle objekter af typen AnimationStep fra den interne Queue i GuiAnimation-klassen. Såfremt klassen ikke er i gang med behandlingen af, eller er nået til sidste ”tick”-værdi i et AnimationStep, vil et nyt AnimationStep blive hentet ud og ”tick”-tælleren vil blive nulstillet.

Hvis et AnimationStep er nået til sit sidste ”tick” og der i øvrigt ikke er flere af disse objekter tilgængelige, vil dette medføre at den public variable bool-værdi ”IsDone” bliver sat til ”True”.

Hvis ellers metoden er kommet så vidt, uden at blive afsluttet af andre grunde, vil den så gå videre til punkt 2 og tjekke om det pågældende AnimationStep fungerer som en pause.
Hvis dette er tilfældet, vil metoden slutte af med blot at inkrementere ”tick”-tælleren.
Ellers ryger metoden ind i en Switch, der aktiverer den relevante metode, for den valgte AnimationType.

Uanset typen af animation man har valgt, og uanset om man benytter sig af AnimatedPoint() eller Animate()-metoden, vil klassen gøre brug af den private Lerp()-metode, som vises her:

```
private double Lerp(double value1, double value2, double t)
{
    return (1 - t) * value1 + t * value2;
}
```

Denne metode er en implementation af lineær interpolation, der sørger for at give animationerne en mere interessant bevægelse, end hvis man blot inkrementerede de pågældende værdier, ud fra en statisk variabel.

WaterSystem-klassen håndterer animationen af bølgerne, som findes i det UserControl der er implementeret på applikationens MainPage.
Disse bølger bliver genereret i form af en liste af WaterLine-objekter, der essentielt blot indeholder et Line-objekt og dennes properties. Disse WaterLine-objekter bliver indsat i et Canvas, med en fast horisontal position, hvor blot den ene vertikale position bliver bundet fast til højden af det

pågældende Canvas. Den anden vertikale position bliver udregnet ud fra metoden i punkt 2:

```
private double CalculateVerticalPosition(double internalOffset)
{
    if (_waterIsMoving && _radius < 5.0)
    {
        _radius += 0.001;
    }
    else if (!_waterIsMoving && _radius > 2.0)
    {
        _radius -= 0.001;
    }
    return Math.Sin(_step + (internalOffset / 5)+Offset) * _radius + CurrentWaterHeight;
}
```

1

2

If-sætningen, markeret i punkt 1, sørger også for at justere radiussen på den udregnede sinuskurve alt afhængigt af om vandhøjden pt. bliver justeret. Derved kan man opleve højere bølger imens vandet stiger eller falder, samt lavere bølger når vandstanden ligger fast.

Ovenstående metode er inkorporeret i WaterSystems Update-metode, som kan ses herunder.

```
public void Update()
{
    for (int i = 0; i < _water.Count; i++)
    {
        _water[i].Line.Y2 = CalculateVerticalPosition(i);
    }
    _step += 0.5;

    if (!_waterHeightChanged) return;

    _waterIsMoving = true;

    if (CurrentWaterHeight > _nextWaterHeight && CurrentWaterHeight > _nextWaterHeight+1)
    {
        CurrentWaterHeight-=2;
    }
    else if (CurrentWaterHeight < _nextWaterHeight && CurrentWaterHeight < _nextWaterHeight + 1)
    {
        CurrentWaterHeight+=2;
    }
    else
    {
        _waterHeightChanged = false;
        _waterIsMoving = false;
    }
}
```

1

2

Denne metode sørger for at køre hver enkelt WaterLine-objekt igennem førnævnte metode (punkt 1), samt at justere vandstanden op eller ned, alt afhængigt af variablerne for nuværende og næste vandstand (punkt 2).

3.2 Udpluk fra View-laget

I dette projekt har vi lagt meget arbejde i at lave pæne og brugervenlige GUIs.
Vi vil ikke som sådan gennemgå specifikke dele af vores View-lag her, men blot henlede opmærksomheden på nogle enkelte af de løsninger og koncepter vi har gjort brug af.
Herunder har Jakob skrevet lidt om UserControls, hvorefter Mathias har skrevet lidt om EventHandlers.

UserControls

For at undgå større mængder af genbrugt XAML-kode på en række af vores Views, valgte vi at gøre brug af UserControls. Det har gjort det muligt for os at konstruere vores egne dynamiske UI-controls, hvor vi i høj grad selv har kunnet definere disses adfærd og udseende.

Dette har manifesteret sig i en række forskellige UserControls, som f.eks. TutorialUC.
TutorialUC bliver brugt i forbindelse med visning af tutorial-popups i vores applikation.

Derudover har vi bl.a. også forskellige typer af BottomBar-UserControls, som er kædet sammen med vores MascotUC. Vores BottomBars er essentielt vores App-bar i bunden af applikationen. Men eftersom vi ønskede at denne skulle have forskellige features, alt afhængigt af siden den lå på, valgte vi at lave flere forskellige udgaver af denne.
Det de dog alle har tilfælles er, at de alle implementerer vores MascotUC.
Dette sørger for at vores maskot, i form af et kompas i nederste højre hjørne, bliver implementeret på præcis samme måde på alle BottomBars.

EventHandlers
En af fordelene ved UserControls, er at de arver direkte fra den side, som de implementeres på. Hvilket betyder at et View med en bundet ViewModel, automatisk vil videregive den samme instans af ViewModellen til UserControllet.

Vores MaskotUC indeholder et ”Click” event, som udføres når der klikkes på billedet af maskotten. Denne event har til formål at vise en tutorial for brugeren, med informationer om det pågældende view.

Problematikken i dette, er at MaskotUC er et UserControl implementeret på ”View”, som skal interagere med et andet UserControl (TutorialUC) implementeret i det samme ”View”.
Det betyder at de i deres egen kontekst, ikke har kendskab til hinanden og dermed kan vi opretholde lav kobling imellem de to UserControls.

For at interagere med TutorialUC fra MaskotUC har vi valgt at gøre brug af EventHandlers.
Disse går i alt sin enkelhed ud på, at delegere et event fra ViewModellen, evt. I form at en metode, videre til et EventHandler objekt i vores UserControl. Dette event kaldes efterfølgende i Click eventet på MaskotUC

Nedenfor vises et eksempel på at kalde en slet metode (Lagret i Viewmodellen) og udføre denne i Click metoden på UserControllet.

1

```
public ActivityVm()
{
    BottomBarActivity.ButtonDeleteEvent += new EventHandler(DeleteActivity);
    InitTutorialUc(PageName.ActivityView);
}

private void DeleteActivity(object sender, EventArgs e)
{
    if (!handler.RemoveActivity(SelectedActivity))
    {
        ErrorHandler.ShowError(handler.ErrorMessage);
        return;
    }
    handler.UserActivities = handler.GetUserActivites();
    _userActivities.Remove(SelectedActivity);
    SelectedActivity = null;
    OnPropertyChanged("UserActivities");
}
```

1. Contructoren i ActivityVm tilføjer en ny event, til BottomBarActivity's ButtonDeleteEvent. Der refereres til metoden DeleteActivity, som så bliver lagret i EventHandleren med korrekte referencer til ViewModellens properties.

2

```
private static EventHandler _buttonDeleteEvent;
public static event EventHandler ButtonDeleteEvent
{
    add
    {
        _buttonDeleteEvent = null;
        _buttonDeleteEvent += value;
    }
    remove
    {
        _buttonDeleteEvent -= value;
    }
}

private void ButtonDelete_Tapped(object sender, TappedRoutedEventArgs e)
{
    if (_buttonDeleteEvent != null)
        _buttonDeleteEvent(this, EventArgs.Empty);
}
```

2. EventHandler propertien sættes med add og remove, for at sikre at der ikke kan tilføjes mere end én event, til EventHandleren.
   Når brugeren klikker på Delete knappen, udføres ButtonDeleteEventet, som på pågældende tidspunkt vil indeholde metoden "DeleteActivity" fra Viewmodellen.

3.3 Udpluk fra ViewModel-laget

Vi vil her ikke gennemgå specifikke ViewModels, da disse som udgangspunkt ikke er specielt interessante, sammenlignet med f.eks. vores Model-lag.
Dog er der en vigtig detalje, ift. implementeringen af gentagende features, som vi gerne vil henlede opmærksomheden på.

Igen, for at undgå større mængder af gentagende kode spredt rundt omkring i vores applikation, valgte vi at oprette en klasse med navnet ”Contents” som fungerer som en form for Information Expert-klasse. Denne klasse blev ikke i sig selv brugt som en ViewModel, men alle andre ViewModels i vores projekt blev sat op til at nedarve fra denne klasse.

Det betød at vi kunne implementere alle features, som vi vidste ville gå igen i alle ViewModels, i denne ene klasse. Derfor kunne vi f.eks. tilknytte INotifyPropertyChanged til klassen, samt opbevare importerede tekst-variabler fra databasen og Tutorial-objekter heri.
I sidste ende var denne opsætning uden tvivl med til at spare en del kode væk i vores ViewModels.

3.4 Review

Onsdag d. 20 Maj var der lagt op til at de forskellige grupper i vores klasse skulle lave reviews af hinandens projekter. Vores projekt blev gennemgået af IClerk-gruppen, på daværende tidspunkt bestående af Stefan, Tobias, Allan og Morten.
De havde på forhånd modtaget et kort oplæg, indeholdende en beskrivelse af vores projekt, samt de features vi ville have lavet review på. Dette dokument er vedlagt som bilag.

De indleverede deres review i form af 4 individuelt udfyldte ark, hvori de havde noteret deres umiddelbare indtryk og kritikpunkter. Vi har herunder forsøgt at opsummere de mest relevante af disse indtryk og kritikpunkter. De individuelle ark er vedlagt som bilag.

Overordnet har IClerk-gruppen været meget positive over for applikationens design og funktionalitet. Med få undtagelser har de fundet vores GUIs rimeligt intuitive og lette at arbejde med.
Den primære kritik har været på størrelsen af enkelte knapper i vores GUI, som potentielt kunne være for små, når applikationen afvikles på en smartphone.

Derudover var der enkelte kritikpunkter baseret på daværende mangler i opsætningen og i vores GUI, som var baseret på det faktum, at vi på dette stadie ikke var fuldstændigt færdige med at implementere vores features i vores Views.

Den endegyldige dom blev, at vores applikation blev godkendt uden noter.

3.5 Tests

I forbindelse med udviklingen af projektet har vi løbende lavet en del tests, i form af debugging, whitebox og blackbox tests. Dette har vi primært gjort i forbindelse med løbende fejlfinding på projektet, for at sikre god og fejlfri performance af vores kode. Dette har været absolut essentielt for vores endegyldige applikation og dennes stabilitet.

Desværre, for os selv og for denne rapport, har vi ikke dokumenteret størstedelen af disse tests. Dette skyldes primært at vi har haft meget travlt i udviklingen af vores applikation, hvilket er et punkt vi vil adressere i forbindelse med vores konklusion. Ergo har intern testning primært været udført i forbindelse med udvikling af de forskellige lag i vores applikation og primært som en sidste handling inden arbejdet fortsatte et andet sted, eller i forbindelse med fejlfinding.

Den eneste dokumenterede test vi har nået at få med, er en Acceptance-test, som Nicolai
har stået for at udføre. Denne er vedlagt som bilag.
Testen blev udført d. 29 Maj, og var overordnet positivt indstillet over for applikationen.

De eneste to kritikpunkter der blev fremhævet, i forbindelse med testen,
var valget af farver i forbindelse med aktivitetstyper i applikationens kalender.
Her blev det påpeget at dette potentielt kunne skabe forvirring hvis brugeren er farveblind
og at man eventuelt kunne ændre dette ved at arbejde mere i retning af former frem for farver.
Derudover forelå der også et ønske om at få vist typen af aktivitet,
når man er i gang med indtastningen af denne.

3.6 Performance

Applikationens performance har været noget vi har givet meget opmærksomhed.
Marc har herunder skrevet en opsummering af de overvejelser vi løbende har gjort os.

Under hele projektet har vi været opmærksomme på at levere en hurtig og flydende oplevelse af vores applikation til slutbrugeren. Der er en række områder hvor vi havde/har udfordringer i forhold til performance, omend det ikke har været kritisk.

Database

I forhold til databasen har vi oplevet nogle performance problemer, både pga. valg af framework og implementation.

I starten af projektet kiggede vi på mulighederne for lokale databaser og frameworks. Valget faldt på SQLite og sqlite-net henholdsvis, da Entity Framework på daværende tidspunkt ikke understøtter Windows Phone. Entity Framework ville sandsynligvis dog kunne tilbyde bedre performance end sqlite-net.

Efterfølgende oplevede vi relativ sløv performance for databasen, især i forbindelse med at slette flere rækker på én gang. Dette skyldes at sqlite-net ikke understøttede cascade delete, hvilket resulterede i et behov for adskillige delete kald. Dette problem blev mere eller mindre løst ved at bruge transactions og først comitte til databasen når alle delete queries var blevet sat op.

Der er en række optimeringer vi stadig ville kunne have implementeret, givet mere tid. I følge dokumentationen på sqlite-net da kan man vinde en væsentlig performance forbedring i forbindelse med DateTime, ved at gemme dem internt som ticks (sekunder siden UNIX tid) i stedet for en string. Denne ændring skulle dog have været implementeret fra starten, og ville kræve en større gennemgang af hele databasen hvis den skulle udføres på nuværende tidspunkt.

Til sidst ville man kunne opnå en bedre brugeroplevelse ved at implementere Conn som asynkron, således at dataene kunne blive hentet mens interfacet stadig ville være responsivt.

Vandanimation

Vandanimationen på forsiden bruger de indbyggede Canvas funktioner til at tegne linjer, hvilket gør den relativ tung. Tidligt i forløbet oplevede vi en væsentlig performance forbedring ved at sænke antallet af linjer der tegnes men gøre dem tykkere. Til trods for denne forbedring valgte vi at inkludere en indstilling for at slå animationen fra, af hensyn til telefoner med ældre/dårlig hardware.

Optimalt set ville vi nok have kunnet få meget bedre performance ved at bruge et framework dedikeret til grafik, såsom Win2D.

3.7 Sprint review

Construction-fasen har, uden tvivl, været den mest stressende og intense del af vores projekt.
Første del, med etablering af Model-laget var som sådan meget ligetil, da vi trods alt havde etableret strukturen for dette i Elaboration-fasen.
Til gengæld var det her, at vi for alvor også begyndte at realisere tankerne om GUI-design.
Dette medførte flere diskussioner om brugervenlighed, intuitivitet mm. for at vi kunne blive enige om en specifik retning, for udseendet af vores applikation.

Løbende har der selvfølgelig ind imellem foreligget problemstillinger og uønskede fejl,
men disse har ikke været så alvorlige, at de har tvunget os til at nedjustere vores ambitionsniveau.
Til gengæld har vi til tider oplevet en skævvridning ift. arbejdsfordelingen,
som er noget vi vil komme ind på i det næste afsnit.

Gruppen har igennem dette forløb primært arbejdet hjemmefra, med nogle få fysiske møder på skolen, i forbindelse med f.eks. diskussioner om GUI-design.
Derudover har vi haft daglige møder via Skype, hvor vi nemt og hurtigt har kunnet diskutere problemstillinger og løsningsforslag, imens de opstod.
Set tilbage på forløbet kunne man godt argumentere for, at vi måske burde have benyttet os af SCRUM, for at planlægge forløbet mere præcist. Men i sidste ende syntes vi at udviklingsforløbet

har været rimelig overskueligt. Det har været forholdsvis nemt at overskue foreliggende arbejdsopgaver, samt prioriteringen af disse og i sidste ende nåede vi også at blive færdige med Construction-fasen hele 6 dage før projektets afslutning.

I det næste afsnit vil snakke om afslutningen på projektet, og konkludere på den samlede proces.

4 RELEASE & KONKLUSION

I dette sidste afsnit, vil vi så vidt som muligt reflektere over det forløb, som denne rapport forsøger at beskrive. Vi vil forholde os til de oprindelige problemstillinger for projektet
og de mål vi havde sat op for projektet. Vi vil også forholde os til de interne regler vi havde stillet op for arbejdet i gruppen og så vil vi, ikke mindst, komme med bud på hvad vi kunne have gjort bedre i vores arbejde. Først vil vi dog lægge ud med den respons vi fik fra vores kunde.

Som en af de sidste brikker i det puslespil der har udgjort vores projekt, ville vi meget gerne have noget feedback fra vores kunde, Benjamin fra Symbiotic Solutions.
Vi fik arrangeret et møde på Elisagårdsvej, torsdag d. 28 Maj,
hvor vi inviterede Benjamin til at komme forbi, så han kunne se og afprøve resultatet.
Nicolai har herefter nedfældet følgende skrivelse om mødet.

4.1 Præsentation for kunden

Vi startede mødet med at præsentere projektet på en computer, for at vise Benjamin hvordan projektet ville fungere med optimal drivkraft, og give ham en god ide om, hvordan projektet ville være i praksis. Vi viste ham rundt i applikationen, samtidig med at vi fortalte ham lidt om tankerne bag nogle af de forskellige funktioner der er i applikationen. For at Benjamin selv kunne få et indtryk af hvordan applikationen ville fungere på en mobiltelefon, havde vi installeret mobil-udgave af applikationen på to forskellige telefoner, hvorefter han kunne udforske applikationen
på egen hånd. Vi havde lånt to telefoner – En nyere model, venligst udlånt af Steen, samt en lidt ældre model, udlånt af Benjamin selv.
Dette gav et godt billede af, hvordan applikationen ville fungere på en
henholdsvis gammel telefon og en af nyere version.

Benjamin gav en detaljeret beskrivelse af hvilke ting han syntes om, samt hvad der, efter hans mening, skulle laves om, for at applikationen blev mere brugervenlig.

De negative bemærkninger var:

- ”Gem”-knappen på sider, hvor brugeren skal indtaste data var gemt væk
  når telefonens tastatur var aktivt. Dette gjorde det lidt besværligt at oprette f.eks.
  en aktivitet, hvor en potentiel bruger formentligt gerne ville gennemføre processen hurtigt.

Løsningen ville være at flytte ”Gem” knappen til toppen, i tilfælde hvor brugeren skal indtaste data.

- Manglende ”Enter”-funktion, for hver indtastning, for at gå videre til næste indtastning automatisk. Altså, at man efter en indtastning i f.eks. en tekstboks kan trykke på ”Enter” på telefonens tastatur, for derefter at blive smidt videre til næste tekstboks i rækken.

De positive bemærkninger var:

- Rigtig godt at Præstationer (Achievements) var kommet ind over uden at blive for prangende.
- Generelt rigtig flot projekt, samt brugbart i praksis
- Flot at en 2. semester gruppe kunne præstere at opbygge et projekt på dette niveau.

I slutningen af mødet blev der talt om mulig videreudvikling af applikationen, men intet blev aftalt, da vi selvfølgelig vil have travlt med eksamen det næste stykke tid.

Dermed kan vi konkludere at projektet har været en umiddelbar succes, da vi selv såvel som Benjamin, er tilfredse med det endelige udfald.

4.2 Refleksioner omkring samarbejde

Vi fik en god start på samarbejdet med Benjamin. I forbindelse med opstarten af projektet har vi haft et rigtigt godt indledende møde, hvor Benjamin tydeligvis har været forberedt og har kunnet give os en umiddelbar retning for projektet som helhed.
Generelt har vi været rigtig tilfreds med de fysiske møder vi har haft med Benjamin, hvor vi som regel har kunnet få taget beslutninger i samarbejde med ham eller få valideret vores foreløbige arbejde.

Dog har der, imellem de fysiske møder, været situationer hvor der har været lang responstid fra Benjamin på e-mails, hvilket har medført at vi indimellem har måttet tage nogle kritiske beslutninger, uden at konferere med ham først.

Vi har en fornemmelse af at dette skyldes flere ting:

Vi har, i gruppen, haft meget høje ambitioner for vores projekt og har derfor forsøgt at være så professionelle som overhovedet muligt omkring udviklingen af projektet.
Dette betyder selvfølgelig også at vi har haft nogle høje forventninger til samarbejdet med Benjamin og de ressourcer han havde at byde på ift. dette.

Dernæst har vi fornemmelsen af at Benjamin ikke selv har prioriteret projektet voldsomt højt, i forhold til andre projekter han sideløbende måtte have haft gang i. Dette er muligvis fordi at han har været bevidst om at dette selvfølgelig er et skoleprojekt.

Til trods for dette, har vi været godt tilfredse med samarbejdet.

Benjamin har været god til at specificere hans ønsker og krav, men stadig givet plads til fleksibilitet i fortolkningen af disse ønsker og krav. Han har derudover været god til at give konstruktiv feedback, i forbindelse med de artefakter vi løbende har produceret,
samt på den endegyldige applikation.

4.3 Overordnet konklusion

Til at starte med, vil vi kigge på de oprindelige problemstillinger for vores projekt
og reflektere over vores resultat i relation til disse.

Første problemstilling var:
Hvordan kan Unified Process (UP), C#-programmering og relationelle databaser bruges,
i udvikling og implementation af et mindre single-user IT system?

Dette har vi i høj grad bevist med vores projekt, da disse 3 virkemidler har været fuldstændigt essentielle for udviklingen af projektet.

Vi havde derefter vores egne problemstillinger:
Hvordan kan vi forbedre slutbrugerens oplevelse og motivere igennem Gameification?
- *Hvilke former for Gameification vil slutbrugeren finde acceptabel?*
- *Hvad kan lade sig gøre og hvad har vi kompetencer til at udføre?*

Gameification koncepter er noget vi diskuterede meget i projekts opstart,
men dette blev lettere nedprioriteret i sidste ende, da vi følte at det var vigtigere at prioritere appens kernefunktionalitet. Derudover følte vi også at der var en risiko for at applikationen kunne blive meget useriøst, hvis det fik for stor en rolle.
De elementer vi, i sidste ende, har valgt at inkludere føler vi er noget en potentiel slutbruger ville finde acceptabelt ved dagligt brug.

Vi er ikke i tvivl om at vi ville kunne have lavet større og mere avancerede implementationer af diverse koncepter, hvis tiden og gruppens prioritering tillod dette.

Vores sidste problemstilling var:
Hvordan kan vi skabe dynamiske interfaces i Windows Apps, med mindst mulig ressource forbrug?

Vi syntes selv at vi har haft meget fokus på at lave eksempelvis UserControls der har kunnet genbruges mange stedet i vores applikation. Derudover har vi også haft fokus på at lave animerede GUIs, som ikke ville bruge for mange ressourcer, om end vi føler der er plads
til forbedring på dette punkt.

Generelt for projektet, har vi i de 2 indledende faser ikke følt og specielt pressede, tidsmæssigt, hvor dette til gengæld har været tilfældet for Construction-fasen,
som har været decideret stressende. Dette har til dels været pga. en stor arbejdsbyrde i forbindelse med udviklingen, samt deadlines der ikke altid blev overholdt og ikke mindst ulige fordeling af

arbejdsbyrden. Vi er selv af den opfattelse, at det har været et spørgsmål om manglende disciplin i løbet af Construction-fasen. Dette ville ikke nødvendigvis afhjælpes med SCRUM, da vi i forvejen arbejdede efter en SCRUM-agtig tilgang, hvor vi løbende holdte styr på arbejdsopgaverne, samt deres fremgang. Vi sørgede også for at uddelegere disse opgaver og opsætte deadlines for deres færdiggørelse og tests. Dette til trods, blev vi færdige med de højest prioriterede features fra vores Vision Statement før vores endelige deadline.

Alt i alt har det været et lærerigt forløb, hvor vi, til trods for periodisk stress,
har fået udviklet en fyldestgørende og stabil løsning, både med henblik på skolens,
såvel som vores kundes ønsker og krav.

4.4 Individuelle konklusioner

Udover den ovenstående og overordnede konklusion, har vi i gruppen valgt at inkludere personlige konklusioner på det forløb vi har været igennem.
Dette har vi valgt at gøre, for at sætte fokus på vores individuelle præstationer
og hvad vi kunne gøre anderledes i fremtidige projekter.

Jakob skriver:
"Personligt er jeg som udgangspunkt godt tilfreds med det resultat vi har fået ud af vores arbejde.
Jeg syntes det har været et spændende projekt at stable på benene og føre ud i livet.
Til gengæld syntes jeg at arbejdsindsatsen i projektet har været skævvredet.
Det har betydet at det i høj grad har været Mathias og undertegnede der har lavet arbejdet, hvilket jeg ikke finder så hensigtsmæssigt.
For Nicolais vedkommende ved jeg at han ikke har så højt et niveau, at han ville kunne varetage nogle af de mere avancerede opgaver, men han har bidraget hvor han har kunnet.
For Marcs vedkommende har jeg desværre indtrykket af, at det har været omvendt.
Det er mit indtryk at han ikke har været voldsomt engageret i projektet, hvilket bl.a. er kommet til udtryk i deadlines der ikke er blevet overholdt og generelt manglende indsats i meget af arbejdet."

Marc skriver:
" Personligt synes jeg det har været et spændende projekt med nye udfordringer. Jeg fik brugt i praksis en række teknikker som jeg enten ikke har prøvet før eller blot i mindre "proof of concept" projekter. I særdeleshed er det mit første større Windows Phone projekt og første gang jeg bruger en database i et sådan omfang.

Det har også været spændende at arbejde med Jakob, Mathias og Nicolai for første gang, og finde ud af hvilken gruppedynamik der var. Det har været interessant at arbejde med en mere struktureret tilgang til deadlines og projekt planlægning generelt."

Mathias skriver:
"Planlægningen samt udviklingen af Ledigheds Kompasset, har været et yderst lærerigt projekt,

med alt fra spændende nye udfordringer til stressende arbejdstimer sent ud på aftnerne.
Jeg forsøger oftest at stille høje krav til mig selv og sætter derfor altid ambitionsniveauet højt; Ikke kun for at udfordre mig selv, men også for at lære og producere et produkt, som jeg i sidste ende, kan stå inde for. Projektet har været den perfekte platform, til at udvikle et omfattende og stabilt stykke software.

Dette er grundet arbejdstilgangen vha. Unified Process, som har givet os rig mulighed for at planlægge og arbejde iterativt over en længere periode.

Overordnet set, føler jeg at vi som gruppe, er kommet i mål, og har udarbejdet en applikation, som vi kan være stolte af.

Når det så er sagt, føler jeg en stor uligevægt i forbindelse med arbejdet under Construction fasen. Jeg er tilbøjelig til at sige, at jeg er skuffet over den manglende arbejdsindsats, og føler at størstedelen af arbejdet i programmeringsfasen har hvilet på mine skuldre. Vi har gennemgående sørget for at uddelegere arbejdsopgaver og i den forbindelse ærgrer jeg mig over, at de klarlagte regler for gruppearbejdet, ikke er blevet overholdt.

Her refererer jeg til Marc, som ikke har udvist den bedste arbejdsdisciplin f.eks. i form af deadlines der ikke er blevet overholdt. Dette har betydet at arbejdsbyrden er blevet pålagt mig, for at sikre, at vi blev færdige med projektet inden vores fælles deadline.

Givet projektets størrelse har det tilføjet ekstra stress og jeg savner derfor,
generelt en større indsats.
Det er noget som jeg personligt, og vi i fællesskab som gruppe, har tænkt meget over.
Derfor mener jeg også, at det er vigtigt at vi i fremtiden, klarlægger strengere regler
og betingelser, for vores fælles arbejdsbyrder."

Nicolai skriver:
"Projektet her på andet semester har været spændende, da det har åbnet øjnene lidt mere for, hvilke muligheder der er inden for kodning. Jeg har lært en del om, hvordan gruppearbejde kan gå, hvis tidsfrister ikke bliver overholdt, eller andre irrelevante aftaler har været prioriteret, hvilket har været tilfældet ved vores nye gruppemedlem.
Mathias og Jakob har som sædvanligt taget byrden på sig, og været yderst produktive.
I forhold til tidligere projekter, har jeg denne gang været med inde under huden på dette projekt, hvilket jeg har set en vis stolthed i, med opgaver imellem, som ikke altid har været det mest simple, og medført at jeg denne gang har følt mig som en del af teamet.
Jeg har desværre ikke altid været i stand til at løfte opgaven pga. manglende erfaring, men har i sidste ende været en god påmindelse om, hvad jeg skal bruge sommerferien på at læse op på."

Hermed afsluttes vores rapport om vores 2. semester-projekt. Vi håber at i har nydt at læse rapporten og at i har fået et godt indtryk af vores forløb og så ser vi frem til at svare på jeres spørgsmål d. 18 Juni.