**I4 PRJ4**

Forår 2015

SmartFridge

Rapport

***Gruppe 5***

**Deltagere:**

|  |
| --- |
| **#1**  Stud.nr.: 201370952 **Navn**: Kristoffer Lerbæk Pedersen |
| **#2**  Stud.nr.: 201270810 **Navn**: Mathias Siig Nørregaard |
| **#3**  Stud.nr.: 201371027 **Navn:** Mathias Schmidt Østergaard |
| **#4**  Stud.nr.: 201371009 **Navn:** Mathis Malte Møller |
| **#5**  Stud.nr.: 201370747 **Navn:** Mikkel Koch Jensen |
| **#6**  Stud.nr.: 201370786 **Navn:** Rasmus Witt Jensen |

**Vejleder:** Lars Mortensen

28. maj 2015

# Resumé

# Abstract

Indholdsfortegnelse

[Resumé I](#_Toc420321209)

[Abstract I](#_Toc420321210)

[Arbejdsfordeling 1](#_Toc420321211)

[Indledning 1](#_Toc420321212)

[Opgaveformulering 1](#_Toc420321213)

[Projektafgræsning 1](#_Toc420321214)

[Systembeskrivelse 2](#_Toc420321215)

[Krav 3](#_Toc420321216)

[Aktørbeskrivelse 3](#_Toc420321217)

[Use case-beskrivelse 3](#_Toc420321218)

[UC1: Se varer 4](#_Toc420321219)

[UC2: Tilføj vare 4](#_Toc420321220)

[UC3: Fjern vare 4](#_Toc420321221)

[UC4: Rediger vare 4](#_Toc420321222)

[UC5: Synkroniser til ekstern database 4](#_Toc420321223)

[Projektgennemførelse 4](#_Toc420321224)

[Metoder 4](#_Toc420321225)

[Specifikation og analyse 4](#_Toc420321226)

[Systemarkitektur 5](#_Toc420321227)

[Design, implementering og test 5](#_Toc420321228)

[Resultater og diskussion 5](#_Toc420321229)

[Udviklingsværktøjer 6](#_Toc420321230)

[Git 6](#_Toc420321231)

[Microsoft Visual Studio 2013 6](#_Toc420321232)

[Microsoft Visio 2013 6](#_Toc420321233)

[LucidChart 6](#_Toc420321234)

[DDS-Lite 6](#_Toc420321235)

[Entity Framework 6](#_Toc420321236)

[Microsoft Sync Framework 6](#_Toc420321237)

[NUnit 6](#_Toc420321238)

[NSubstitute 6](#_Toc420321239)

[Doxygen 6](#_Toc420321240)

[Opnåede erfaringer 7](#_Toc420321241)

[Kristoffer Lerbæk Pedersen 7](#_Toc420321242)

[Mathias Siig Nørregaard 7](#_Toc420321243)

[Mathias Schmidt Østergaard 7](#_Toc420321244)

[Mathis Malte Møller 7](#_Toc420321245)

[Mikkel Koch Jensen 7](#_Toc420321246)

[Rasmus Witt Jensen 7](#_Toc420321247)

[Fremtidigt arbejde 7](#_Toc420321248)

[Konklusion 9](#_Toc420321249)

[Referencer 11](#_Toc420321250)

[Bilag 12](#_Toc420321251)

# Arbejdsfordeling

# Indledning

# Opgaveformulering

Formålet med dette projekt er at udvikle et system, som tillader registrering af varer i et køleskab via en grafisk brugergrænseflade på en lokal skærm. Lagring af disse oplysninger sker i en lokal database, der synkroniseres med en ekstern database, som kan tilgås via et web-interface. Systemet vil også tilbyde vedligeholdelse af en indkøbsseddel, samt en liste over standardvarer, der altid ønskes i køleskabet.

Systemet, som er relativt simpelt, byder desuden på rig mulighed for udvidelse, i form af inkorporering af ekstra funktioner; eksempelvis oplysning om ernæringsværdier, opskrifter baseret på eksisterende ingredienser og eksisterende/fremtidige tilbud på manglende varer.

Systemet skal ses som en ekstern tilføjelse til eksisterende køleskabe, og ikke i første omgang som en indbygget feature i nye modeller. Dette sikrer at systemet kan tilbydes til en bredere målgruppe end køberne af ”high end”-køleskabe.

Visionen er at tilbyde brugeren et hurtigt og effektivt overblik over køleskabets indhold, til lettelse i en hverdag, hvor man ikke altid har en opdateret indkøbsliste inden for rækkevidde.

Brugerne kan være privatpersoner, såvel som industrielle køkkener og catering-virksomheder, som ønsker overblik over indholdet i deres køleskab(e).

# Projektafgræsning

Udførslen af projektet er med fokus på softwareudvikling, så problemer som ikke er direkte software-relaterede, vil ikke blive forsøgt løst. Dette inkluderer:

* Strømforsyning til enheden, der kører Fridge app’en.
  + Kabling
  + Evt. batterilevetid
* Udvikling af enhed til kørsel af Fridge app’en.
  + Lenovo Yoga 2 Pro (**bilag XX**) benyttes som platform for Fridge app.

Ydermere vil *Web app* ikke indeholde de udvidelser, der er blevet tilføjet til *Fridge app*, og altså kun indeholde kernefunktionaliteterne. Dog har det også i kernefunktionaliteterne været nødvendigt med en afgrænsning; *Web app* vil ikke automatisk tilføje varer fra *Standard-beholdning* til *Indkøbsliste* ved mangel i *Køleskab*, ligesom det heller ikke vil være muligt at initiere en synkronisering mellem den lokale og den eksterne database. Den æstetiske fremtoning af *Web app* vil heller ikke blive prioriteret, da fokus ligger på at udstille den implementerede funktionalitet.

# Systembeskrivelse

Systemet, skitseret i **Figur 1**, skal kunne assistere brugeren, ved at tilbyde opretholdelse af en liste over hvilke varer, der er i køleskabet, og en indkøbsseddel til manglende varer. Der kan tilføjes og fjernes varer i takt med indkøb og forbrug, samt opretholdes en liste over ønskede varer i køleskabet, der ved mangel automatisk tilføjes til indkøbssedlen. Systemet vil desuden bestå af en lokal og en ekstern database, hvori de oprettede lister gemmes, så listerne også kan tilgås fra en web-applikation.

**Fridge app**



**Køleskab**

**Lokal DB**

**Ekstern DB**

**Computer**

**Smartphone**

**Tablet**

**Web app**

**Figur 1** Skitsering af systemet SmartFridge

Når brugeren starter systemet, vil den lokale database synkronisere med den eksterne database, for at sikre at begge databaser er opdateret og stemmer overens. Herefter vil den lokale og den eksterne database synkronisere jævnligt, så længe systemet er i brug.

Efter systemet er startet, vil det være muligt for brugeren at tilføje varer til listen over varer i køleskabet, på indkøbslisten eller i standardbeholdningen, hvorpå brugeren får eksisterende varetyper som forslag, eller kan tilføje nye varetyper selv.

Det er muligt at se varerne på de forskellige lister. I denne forbindelse vil det være muligt at redigere mængden af varerne i takt med forbrug.

Systemet vil selv tilføje vare til indkøbssedlen når køleskabet indeholder en mindre kvantitet af en vare, end der er angivet på listen over standard-varer.

|  |
| --- |
| **Figur 2** Use case-diagram over SmartFridge |

# Krav

Ud fra opgaveformuleringen, er der udarbejdet en række *use cases*, som beskriver aktørernes interaktion med systemet. Disse use cases fungerer som kravspecifikation, og bruges i den tidlige del af udviklingsfasen til at bestemme systemets funktionalitet. For *fully dressed use cases*, henvises til projektdokumentationens **side XX**.

## Aktørbeskrivelse

På *use case*-diagrammet på **Figur 2** ses en række aktører. Disse er beskrevet i følgende aktørbeskrivelse. For mere detaljerede beskrivelser, henvises til projektdokumentationens **side XX**.

**Bruger** er en primær aktør, som ønsker at benytte systemet ved at tilføje, fjerne, redigere og aflæse varer.

**Ekstern Database** er en sekundær aktør, som løbende synkroniseres med systemets lokale database. Herudover fungerer den som direkte database for systemets web-applikation.

## Use case-beskrivelse

*Use cases* fra *use case*-diagrammet er beskrevet i følgende afsnit. Hver *use case* beskriver et scenarie, hvor Bruger interagerer med systemet.

### UC1: Se varer

Bruger får frembragt alle de nuværende varer på en ønsket liste.

### UC2: Tilføj vare

Bruger tilføjer en vare til en ønsket liste.

### UC3: Fjern vare

Bruger fjerner en vare fra en ønsket liste.

### UC4: Rediger vare

Bruger redigerer vareinformationerne på en vare i en ønsket liste.

### UC5: Synkroniser til ekstern database

Bruger initierer en øjeblikkelig synkronisering mellem Lokal Database og Ekstern Database.

# Projektgennemførelse

# Metoder

# Specifikation og analyse

Da udviklingen af WPF-applikationen gik i gang, var det meget nyt. Gruppen lærte undervejs hvordan delene skulle programmeres, og der var ikke de store overvejelser omkring hvilken arkitektur programmet skulle bygges op efter. Dette har givet nogle problemer med at bevare overskueligheden, og med at holde koden let at vedligeholde. I slutningen af april blev der foretaget en analyse af *MVC*, *MVP* og *MVVM*, og om det kunne betale sig at refakturere koden, for at overholde disse arkitekturer. Gruppen fandt, at det ville have været en god idé at opbygge WPF-applikationen efter *MVVM*-arkitekturen, helt fra starten af. Hvis det var sket, ville mængden af kode i *code behind* have været reduceret væsentligt. Problemet med kode i *code behind* er, at det gør det vanskeligt at teste koden i form af *unit tests*. *MVC* og *MVP* blev også undersøgt. Grundlæggende har det vist sig at WPF-applikationen til dels følger MVC/MVP-arkitekturen. Der er et *View* i form af XAML, *presenter*/*controller* i form af *code behind*, *Business Logic Layer* (*BLL*)og *Data Access Layer* (*DAL*). Det ville have været en fordel at fjerne koden i *code behind*, og rykke det ud i et lag for sig selv. Fordelene ved at gøre dette kunne dog ikke gøre op for den tid det ville tage at skrive koden om. Gruppen har derfor besluttet at arbejde videre med koden som den var, men at lære af det og bruge den nye viden i forbindelse med udviklingen af *Web app*.



**Figur 3** MVC/MVP til 3-lags-arkitektur

En anden udfordring har været *mapping*’en mellem objekt-udgaven af en *entity* i koden, og dens relationelle database-udgave. Der er brugt lang tid på at få denne *mapping* til at være korrekt, og sørge for at udgaven i koden er den samme som den eksisterende i databasen. Løsningen på dette blev fundet gennem undervisningen i I4DAB. Grundet den fremskredne udvikling i WPF-applikationen blev det ligeledes besluttet at det ikke kunne betale sig at refaktorere koden, men at implementere det som en del af *Web app*.

I *Web app* er der også blevet indført et *GOF-pattern* ved navn *Facade*. *Facade*-mønsteret har i dette tilfælde til opgave at give et enkelt adgangspunkt til *DAL*. I forbindelse med *DAL* er der et *repository* for hver tabel i databasen, samt en databasekontekst. For at overskueliggøre dette er der lavet en facade, så man i koden, der skal bruge disse elementer, blot opretter en *Facade*, og ikke alle de forskellige elementer. Dette gør koden mere vedligeholdelsesvenlig, og overskueliggør koden, set fra et højere abstraktionsniveau.

For yderligere info om de trufne beslutninger, henvises til mødereferaterne (**Bilag XX**).

# Systemarkitektur

# Design, implementering og test

# Resultater og diskussion

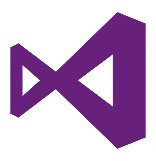
# Udviklingsværktøjer

Herunder gives en kort beskrivelse af relevante udviklingsværktøjer benyttet i projektet.

## Git

Til organisering af filer, både kode og dokumentation, er der blevet benyttet et Git-repository fra GitHub. Dette er et system til fildeling, hvor der samtidig tages højde for filhistorik. Det har været muligt at sidde flere personer på samme fil på en gang, for derefter at ’merge’ indholdet sammen til én fil.

## Microsoft Visual Studio 2013

Udviklingsværktøjet er blevet brugt til at skrive stort set alt kode. Databasen, *Fridge app* og *Web app* er alle blevet udviklet i Visual Studio. Alle *frameworks* nævnt herunder er ’udvidelser’ til dette værktøj.

## Microsoft Visio 2013

Et tegneværktøj, som er benyttet til de fleste UML-diagrammer for systemet.

## LucidChart

Et tegneværktøj, der kan bruges til at lave diverse UML-diagrammer til databasen.

## DDS-Lite

Et værktøj, der gør det muligt at designe en relationel database, og herefter generere de nødvendige scripts til at bygge databasen. Er benyttet til at designe den lokale database.

## Entity Framework

Et bibliotek, der gør det muligt at arbejde i et objektorienteret .NET-miljø, som kan mappes direkte til en relationel database.

## Microsoft Sync Framework

Et bibliotek, der hjælper med synkroniseringsprocessen mellem den lokale og den eksterne database.

## NUnit

Framework benyttet til systematiske tests, for at sikre kodens kvalitet ved ændringer.

## NSubstitute

Framework brugt i forbindelse med tests. Bruges for dynamisk at substituere undermoduler i det modul som testes, frem for manuelt at skrive *fakes*.

## Doxygen

DoxygenVærktøj benyttet til at generere overskuelig dokumentation for programkode.

# Opnåede erfaringer

## Kristoffer Lerbæk Pedersen

## Mathias Siig Nørregaard

## Mathias Schmidt Østergaard

## Mathis Malte Møller

## Mikkel Koch Jensen

## Rasmus Witt Jensen

For mig har dette semesterprojekt været det mest lærerige og succesfulde igennem mit studie. Det her ligeledes været det mest relevante for mine mål som softwareudvikler. Arbejdet med WPF, ASP.NET og JavaScript har været utrolig udfordrende og spændende.

Jeg har lært vigtigheden af at have opbygget sin kode efter en ordentlig arkitektur, og opfylde SOLID principperne. Uden dette har vi haft utroligt svært ved at teste programmet.

På trods af at jeg føler at der har været mangler i forhold til at opfylde disse principper, synes jeg at det endelige resultat er tilfredsstillende. Gruppen har som helhed formået at bære opgaven, og få lavet et godt produkt. Hvert medlem har ydet en god indsats, og leveret et godt stykke arbejde.

Projektstyringen, hvor der har været stor vægt på en agil tilgang, og med en delelementer fra Scrum, har vist sig at være yderst effektiv. Det er sjældent sket at folk har ventet på at andre blev færdige med noget, for at kunne komme videre med deres eget. Hvis dette har været tilfældet, er det der har stået i vejen blevet prioriteret højt, og derved hurtigt blevet løst.

Der har til tider været meget gøgl i gruppen, hvilket har kunnet hæmme arbejdet. Dette blev der dog hurtigt rettet op på, ved at indføre at Kristoffer og jeg selv skulle sørge for at folk holdt fokus under vores gruppearbejde. Dette har effektiviseret arbejdet, og jeg føler ikke et lige så stort tidspres i forhold til projektet her i slutningen, i forhold til hvad jeg har gjort i de forrige semesterprojekter.

Personligt føler jeg gruppen har fungeret godt, og har haft et godt sammenhold. Alle virker tilfredse, og har folk været utilfredse med noget, er det blevet sagt og rettet op på.

# Fremtidigt arbejde

For at finde de første mål for fremtidigt arbejde, ses på MoSCoW-opdelingen i projektdokumentationen, **side XX**, hvor de første ikke-implementerede use cases er at finde under *Should*:

* En påmindelse om manglende vare(r) på en af listerne.
* Muligheden for at tilføjelse flere skabe.
* Et log-in-system, så må kan være flere brugere om samme system, samt af sikkerhedsmæssige årsager.

Hernæst vil det være en idé at foretage en evaluering af eventuelt manglende udvidelser, såsom en mere dynamisk indkøbsliste, med mulighed for at krydse varer af, som brugeren har lagt i kurven, mens der handles.

Implementering af funktionaliteterne under *Could* vil gøre produktet mere gennemført, og vil derfor være næste skridt i udviklingen, hvis det antages at produktet fortsat kan udvikles uden lancering.

Antages der derimod en deadline for produktlancering, vil en højere prioritet være at se på de afgrænsninger, som er sat, i særdeleshed at *Fridge app* udelukkende er udviklet til at køre i opløsningen 1920x1080 pixels.

Til sidst bør der kigges på *Would*/*Won’t*-delen. Disse dele mangler stadig at blive teknologiundersøgt tilstrækkeligt til at der kan foretages en vurdering om hvorvidt, det ville være rentabelt at implementere disse funktioner.

# Konklusion

Projektet er endt ud med en WPF-applikation, en web-applikation og tilhørende databaser. Den ene database kører lokalt, og den anden kører på en Azure-server sammen med web-applikationen. De to databaser synkroniserer automatisk, eller ved at brugeren aktivt beder om det ved et tryk på brugerinterfacet. Der er implementeret flere funktionaliteter på WPF-applikationen end på web-applikationen, da det er her, gruppens største fokus har været. Arbejdet med WPF applikationen startede før introduktionen til *MVP*, *MVC* og *MVVM*. Konsekvensen af dette er manglende afkobling af XAML, *code behind* og *Business Logic Layer* (*BLL*). Dette har gjort det utrolig svært at teste, og derfor er der heller ikke skrevet så mange *unit tests* og integrationstests som ønsket.

Databasetilgangen er i WPF implementeret med ADO.NET, hvilket har givet en del problemer i forhold til *mapping*, og at bevare data-integriteten i selve databasen.

I web-applikationen har vi haft fokus på at få designet og struktureret programmet bedre end vores WPF-applikation. Her har vi brugt *MVC*, og haft fokus på at overholde dette designprincip. Formålet har været at bruge alle de erfaringer, vi har fået under udviklingen af WPF-applikationen, og dermed udvikle en velstruktureret web-applikation.

Databasetilgangen er lavet med *Entity Framework* (*EF*), og mange features og *mapping* sker derfor automatisk. Dette har gjort web-applikationen væsentligt mere læsbar og vedligeholdelsesvenlig end WPF-applikationen.

I projektet er der brugt *continuous integration* i form af et Git-repository. Brugen af Git har gjort det utroligt let at arbejde i det samme *Visual Studio*-projekt på samme tid, og rette de fejl, der kommer, når nye funktionaliteter implementeres. Der har dog været nogle problemer i form af *merging*, når der er blevet ændret i præcis de samme filer, imellem hvert push/pull. Ved disse merges er der nogle få gange gået data tabt, som skulle have været bevaret. På trods af dette opvejer fordelene klart de ulemper, der har måttet være.

Git var ligeledes en nødvendighed, da automatisering af tests i form af *Jenkins* var et ønske, og sammenkoblingen af Git og *Jenkins* var blevet introduceret i I4SWT. Efter at have brugt meget tid på at få det sat op, måtte vi dog konstatere at det ikke var tiden værd, da der uden adgang til selve *Jenkins* serveren ikke kunne laves de nødvendige rettelser for at få *Jenkins* til at kompilere projektet. De første par fejl blev løst over et par uger i samarbejde med Troels Fedder Jensen, men da der blev ved med at dukke nye småting op, gik vi fra brugen af *Jenkins* igen.

Selve gruppearbejdet har været udført med en agil tankegang, og har primært været inspireret af *Scrum*, i form af stand-up møder, retrospektmøder, sprints og backlog. Den agile udviklingsproces har gjort det muligt hele tiden at have et fungerende produkt, som kunne vises til vejlederen, og løbende at tage de vigtigste opgaver. Det har også muliggjort at alle i projektgruppen hele tiden har været klar over hvor langt projektet har været på et givet tidspunkt, og har været i stand til at søge hjælp med det samme, hvis et medlem har været forhindret i at arbejde videre.

Alt dette har resulteret i et enestående slutresultat, med et projekt som opfylder alle de krav, der blev sat i starten af projektet. Det har dog ikke været muligt at implementere mange af de ekstra tilføjelser, grundet tidspres. Dog er der blevet stiftet bekendtskab med nye teknologier, som Azure, hvilket der ikke er blevet undervist i. Ligeledes er der synkronisering mellem to databaser, og en web-applikation, hvilket der ikke har været stort fokus på uden for projektet.

# Referencer

**Der er ingen kilder i dokumentet.**

# Bilag