# Design, implementering og test

## Overordnet design

Deployment-diagrammet i **Figur 1** viser, hvordan de interne dele af Smart Fridge interagerer. Som det fremgår, interagerer Bruger kun med enten en *Fridge app* eller *Web app*, og har altså ingen direkte interaktion med den bagvedliggende logik.

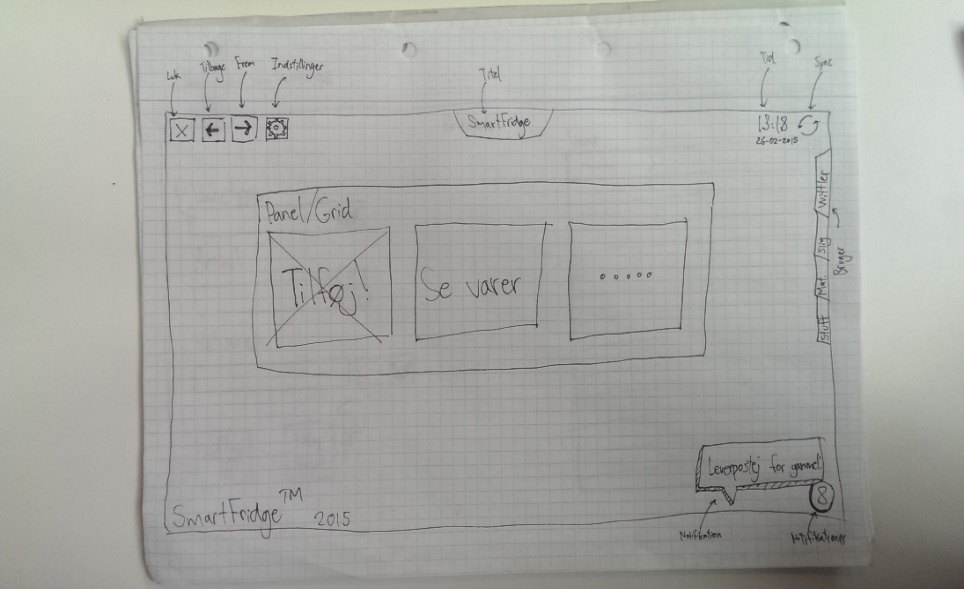


**Figur 1** Deployment Diagram for hele SmartFridge-systemet

For et mere detaljeret *Deployment*-diagram, henvises til **bilag XX**.

## Fridge app

I dette afsnit vil der blive set på processen omkring det visuelle design af *Fridge app*, såvel som den konkrete implementering af systemet, og de mere bemærkelsesværdige funktioner. Ligeledes vil der blive redegjort for overvejelserne omkring systemets testbarhed.



**Figur 2** Skitse af hovedmenu og ramme

### Design

Designet på *Fridge app* er, på baggrund af en forudgående brainstorm, i første omgang skitseret i hånden, hvorefter modelleringen i WPF har taget udgangspunkt i disse skitser.

Dette afsnit vil beskrive de ønsker til designet, som kom frem i brainstormen. For flere detaljer omkring designet, henvises til projektdokumentationens **side XX**.

Helt overordnet består designet, som skitseret på **Figur 2**, af hovedvinduet, som danner en ramme om en *User Control*; et vindue som dynamisk kan skifte indhold.

#### Størrelse

På baggrund af det apparat, systemet i første omgang udvikles til, designes *Fridge app* med en størrelse på 1920x1080 pixels, mens de *User Controls*, som vises i midten af skærmen, har en størrelse på 1280x720 pixels.

### Implementering

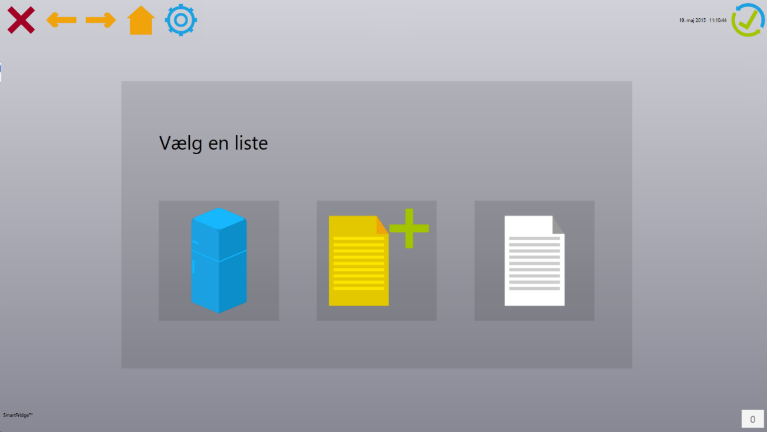
Den overordnede navigation i *Fridge app* illustreres som et *state machine*-diagram i **Figur 3**.



**Figur 3** State machine diagram over Fridge app

Bemærk at rammen, som til enhver tid omkranser de forskellige *User Controls*, har knapperne *Hjem*, *Frem* og *Tilbage*, som øger letheden af navigationen, ved til enhver tid at tilbyde muligheden for at gå til listeoversigten, samt en side frem eller tilbage. Disse knapper er ikke illustreret på **Figur 3**.

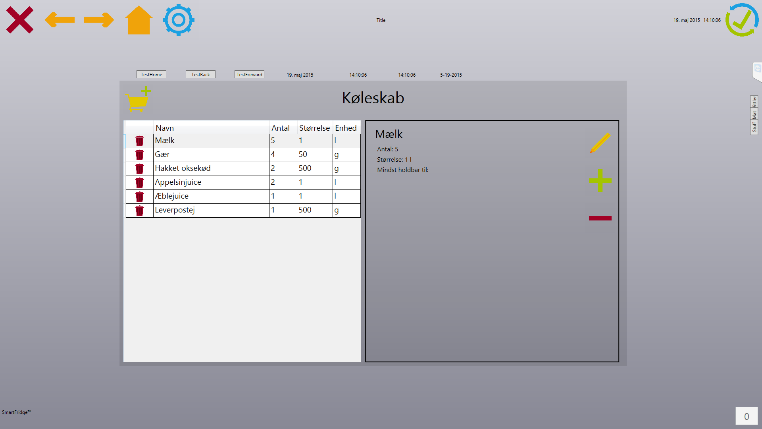
Ved åbning af programmet, indlæses rammen, som indlæser listeoversigten. Ved valg af en liste, udskiftes listeoversigten med den valgte liste; *Køleskab*, *Indkøbsliste* eller *Standard-beholdning*. Ved indlæsning af disse lister, hentes informationerne om deres tilknyttede varer fra databasen, og indlæses i en liste.



**Figur 4** Hovedmenu

Fra listerne er det muligt at redigere varerne, og det er muligt at tilgå den sidste *User Control*, *Tilføj vare*.

I vinduet *Tilføj vare* er det muligt at tilføje varer til den pågældende liste, som har fremkaldt det. Efter de ønskede varer er tilføjet, er det muligt at gemme, hvorved varerne persisteres, og der returneres til *Vis varer*.



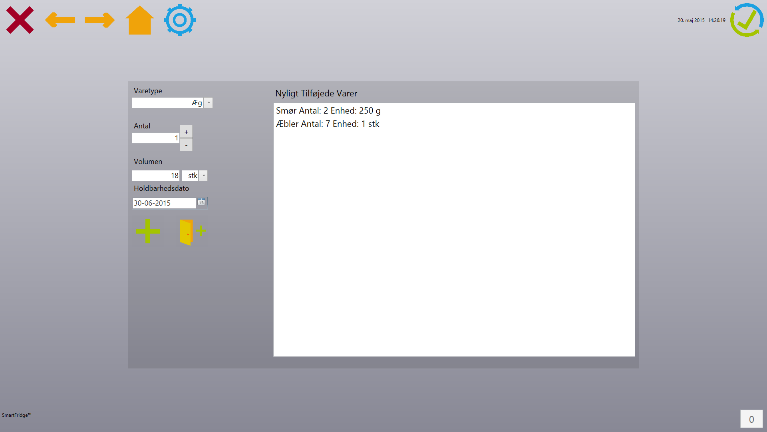
**Figur 5** Listen "Køleskab"

#### View

*Viewet* er blevet implementeret i nogenlunde overensstemmelse med det oprindelige design.

Fra hovedvinduet indlæses listeoversigten, som vist i **Figur 4**. Herfra er det muligt at tilgå de eksisterende lister, som i første omgang består af *Køleskab*, *Indkøbsliste* og *Standard-beholdning*.

Fra den valgte liste er det muligt at se, tilføje, redigere og fjerne varer, som vist i **Figur 5**.



**Figur 6** Tilføj vare(r)

Vælges der at tilføje varer, er det muligt med vinduet vist i **Figur 6**.

For mere detaljerede beskrivelser af *View*-delen af *Fridge app*, henvises til projektdokumentations **side XX**.

#### Business Logic Layer

*Business Logic Layer* er blevet implementeret på en sådan måde, at det står for at facilitere al kommunikation mellem *View* og *Data Access Layer*. Det konverterer *ListItem* og deres tilhørende *items* fra databasen til et samlet objekt i form af et *GUIItem*, som bruges af *viewet*, og omvendt når der skal tilføjes til databasen. Det er også i dette lag, der laves tjek på hvorvidt en vare har overskredet dens holdbarhedsdato, og hvis den mangler fra standard-beholdningen, få den tilføjet til indkøbslisten.

### Test

I begyndelsen og under udviklingen af projektet, var det målet at skrive unit tests til hver enkelt implementeret funktion. Dette blev af flere årsager ikke realiseret.

Mange af funktionerne i projektet er lavet ud fra ”*button click*”-funktioner, som genereres efter behov ud fra knapperne i viewet. I disse funktioner blev der, i starten af projektet, implementeret al logik, som senere overført til projektets *Business Logic Layer* (*BLL*). Konsekvensen af dette blev at ”*button click*”-funktionerne blev svære at teste vha. *NUnit*-frameworket, da det ikke har været muligt at simulere et museklik, når både eventet og funktionen er private. Ydermere blev det besluttet af disse tests ikke var relevante i samme grad som tests af *BLL*-laget, hvor størstedelen af den reelle databehandling fandt sted. Her blev der, så vidt muligt, lavet unit test, dog mest fokuseret på integrationstests.

Integrationstestene blev, pga. manglende unit tests og den høje kobling, et krav til projektet. Mange af disse tests er baseret på at hente, slette og manipulere data fra databasen.

Skulle problematikken med unit testene være undgået, skulle projektet havet været bygget mere op om et mønster/*pattern*, der havde gjort det mere testbart. Dette mønster kunne eksempelvis have været MVC-mønstret, som adskiller modellerne, viewet og controlleren bedre end måden, hvorpå denne applikation er opbygget. Med dette mønster ville koblingen blive lavere, og ingen tests af viewet være nødvendige, da der ikke ville være noget code-behind. For mere information om overvejelserne omkring forskellige mønstre, henvises til projektdokumentations **side XX**.

## Web app

### Design

### Implementering

### Test

## Database

### Design

### Implementering

### Test