# Design, implementering og test

## Web app

### Design

Valget af *ASP.NET* er baseret på den funktionalitet som frameworket giver, frem for hvis *Web app* udelukkende var blevet lavet i HTML. Vha. ASP.Net er der mulighed for at manipulere med det data som brugeren indtaster og det data der ligger i databasen, i C#.

Når der laves en web-applikation i *ASP.NET*, skal der træffes et valg om hvilke frameworks og mønstre, man ønsker at bruge til at udvikle applikationen. Da der skulle besluttes hvordan applikationen skulle udvikles, var to metoder under overvejelse: *ASP.NET Web Forms* og *ASP.NET MVC*. *Web Forms* og *MVC* benytter sig begge af HTML, men ifølge forskellige websites (bl.a. (Sukesh, 2014) og (Hambrick, 2013)), har udvikleren mere kontrol over renderingen af den HTML-kode som brugeren ender med i browseren, hvis der benyttes *MVC* i forhold i *Web Forms*.

De *View*-filer, der benyttes, er .*cshtml*-filer i *Razor*-syntax (W3Schools, u.d.). Dette er valgt på baggrund af hvor nemt det er at bruge. For at lave et *View*, skal der blot benyttes HTML, og hvis der skal tilføjes funktionalitet kan der kaldes en funktion i en controller, eller der kan skrives C#-kode direkte *Viewet* vha. ’@’-annotationen. *Web Forms* bruger *drag’n’drop*-designmetoden i stedet for *HTML*-kodning. Dette gør det mere visuelt at designe en applikation, men har ikke været en nødvendighed under dette projekt.

MVC-mønsteret er primært valgt på baggrund af to ting. Den første har været, at det har skullet være nemt at køre tests på, og den anden har været den faglighed, som er opnået omkring mønsteret i både *I4SWT*- og *I4GUI*-fagene.

Til at lagre applikationens data, er *MSSQL* valgt over *MySQL*. Grunden til dette er muligheden for synkronisering mellem den online database og den lokale. Det er muligt at synkronisere mellem en *MySQL*-og en *MSSQL-*database, men da Windows-applikationen allerede benyttede sig af *MSSQL*, var det nemmere at lave tilsvarende til webapplikationen. Se mere omkring dette i ’Database’-afsnittet, **side XX**.

Det overordnede design for web-applikationen minder meget om designet for *Fridge app*, men der er dog et par små forskelle. Den væsentligste forskel fra designet af *Fridge app* er at *Rediger vare* er kommet ud i et *View* for sig, i stedet for at være en del af en *Item List.* For en mere detaljeret beskrivelse af designet af *Web app* henvises til projektdokumentationen **side XX.** En overvejende del af overvejelserne går på at web-applikationen skal kunne bruges på tværs af platforme. Da ikke alle enheder har lige store skærme, eller lige høj opløsning, har det været vigtigt, ikke at have for meget i hvert *View*. I nogle tilfælde har dette blot betydet en omstrukturering af det i forvejen eksisterende *View*, og i andre tilfælde, som med *Rediger vare*, er der blevet lavet et helt nyt *View*.

### Implementering

Til oprettelsen af *Web app*-projektet er *Visual Studio* benyttet. *Visual Studio* har hjulpet med at oprette et projekt, der benytter sig af *MVC*-mønsteret, og som bruger Razor engine. Ydermere sættes det nemt op til *Azure* (Microsoft, 2010). Mappestrukturen bliver herved givet af *Visual Studio*. Mere om dette ses i projektdokumentations **side XX**.

Når data skal sendes fra et *View* til en *Controller*, benyttes *HTML Helpers*. Med disse hjælpere er det muligt at kalde en funktion i en *Controller* og sende information med. Dette kunne være information om et nyoprettet item. Ydermere findes der knapper og tekstbokse, som bliver renderet på Viewet som HTML-kode. Disse knapper hjælper også med at eksekvere kode i controlleren.

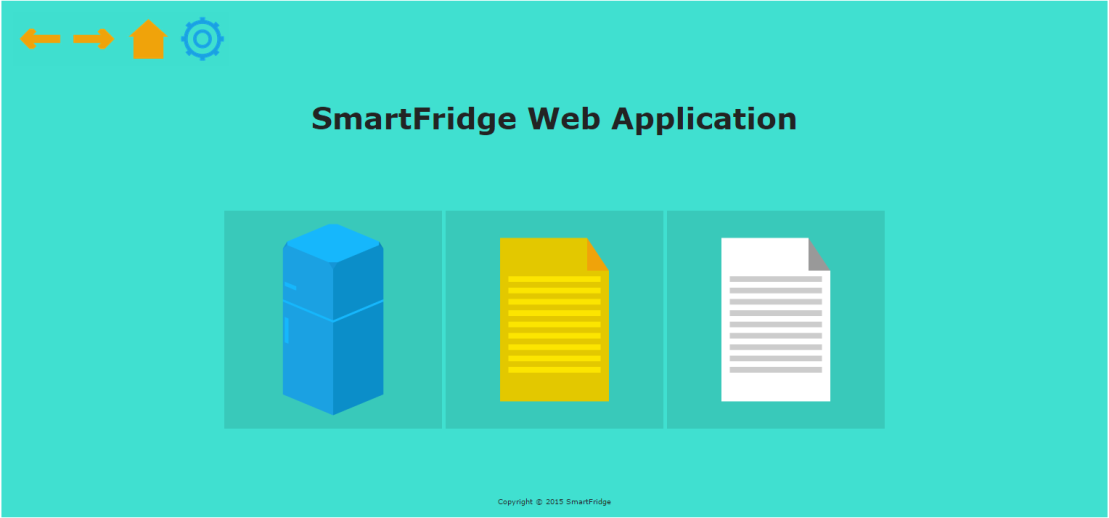
*Razor*-syntax til en *HTML Helper* (W3Schools, 2015):

@Html.ActionLink("About this Website", "About")

#### View

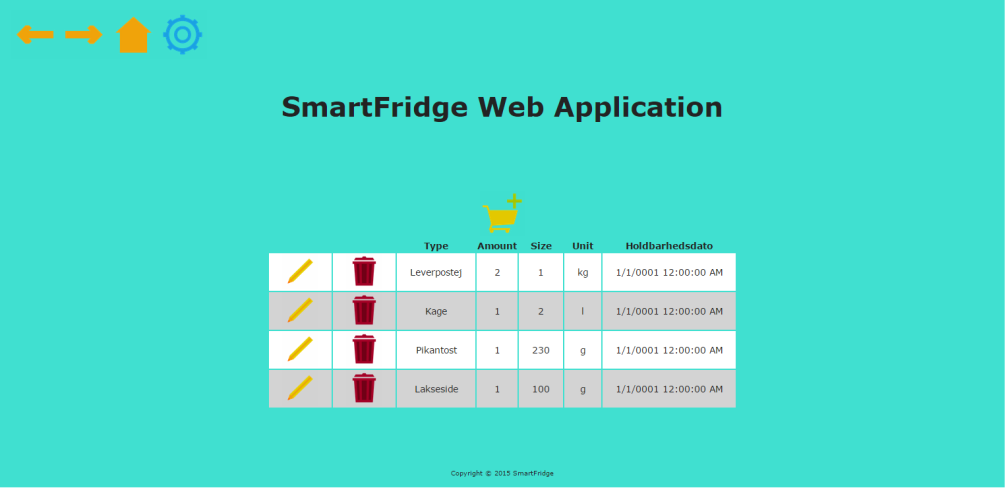
Grundet tidspres er der ikke blevet fokuseret så meget på *View*-delen af web-applikationen. Til trods for dette, er de kommet til nogenlunde at ligne det designede. Der har dog primært været fokus på funktionalitet over udseende.

På **Figur 1** ses *Index*, der svarer til hovedmenuen i *Fridge app*, hvorfra en eksisterende liste vælges.



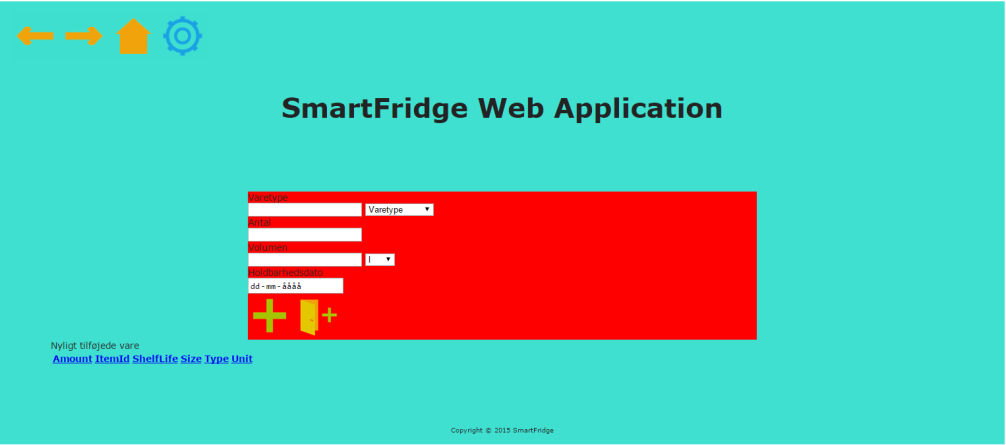
**Figur 1** Index

Når en liste er valgt bliver brugeren præsenteret for et *View*, der minder om det, som kan ses på **Figur 2**. Dette *View* viser indholdet af den valgte liste, og giver mulighed for at navigere videre til enten *Tilføj vare* eller *Rediger vare*.



**Figur 2** Listen ”Køleskab”

Vælges *Tilføj vare* bliver brugeren ført videre til vinduet på **Figur 3**, hvorfra der kan tilføjes varer til den nuværende liste.



**Figur 3** Viewet ”Tilføj vare”

Vælges *Rediger vare*, præsenteres brugeren for vinduet på **Figur 4**, hvor tekstboksene vil være fyldt med den nuværende vareinformation.



**Figur 4** Viewet ”Rediger vare”

Foretages der ændringer, gemmes disse når der trykkes på fluebenet, hvorefter der returneres til den liste, som tidligere blev valgt. Trykkes der på krydset, returneres til listen, og vareinformationen er, som den var før.

#### Controllers

Samlet set udgør *controllerne* det, der svarer til *Business Logic Layer* fra *Fridge app*, og de varetager altså kommunikationen med den eksterne database. Hver *Controller* har sit ansvar. *ListView* står for at indlæse og præsentere alle varer på en liste, og giver mulighed for at slette en udvalgt vare. Den står også for at omdirigere til de to v*iews*, *Additem* og *EditItem*. Inden *controlleren* omdirigerer til *EditItem*, finder den først den vare, der skal sendes med til denne, før der omdirigeres. *AddItem*-*controlleren* har til ansvar at tilføje varer til databasen. Finder den en tilsvarende vare på den nuværende liste, sørger den for blot at øge mængden af den eksisterende vare med det tilføjede antal, i stedet for at indsætte en dublet. *EditItem*-*controlleren* står for redigering og opdatering af en valgt vare. *EditItem* sørger også for at en vare ikke kan opdateres, så den ikke optræder som en dublet på listen; dette håndteres om i *AddItem*.

#### Cache

Til *Web app* er der blevet implementeret en *cache*. Denne *cache* indeholder alle de ting, som er fælles for alle *Controllers*. Klassen består kun af properties, der alle er statiske. Begrundelsen for dette er netop at det er fælles data, som alle controllers skal arbejde på. Det er også i *cachen* at facaden til *Data Access Layer* ligger, da der på alle tidspunkter kun må være én facade.

### Test

Da webapplikationen er bygget op omkring MVC-mønstret skal der kun testes på *Controller-* og *Model*-klasserne i projektet, da det er her, funktionaliteten og logikken i applikationen ligger. Pga. den lave kobling, som dette mønster giver, er *unit tests* nemme at lave. Udfordringen i dette projekt har været at koble den logik, som er lavet til *Fridge app*, over til *Web app*. For at sikre kontinuitet mellem de to, er logikken blevet genbrugt.

Et af problemerne ved den måde, hvorpå funktionerne i applikationen er opbygget, er at mange af dem returnerer et *View* eller et *RedirectToAction*, som set **Figur 5**, efter et item har gennemgået anden logik i samme funktion. Et eksempel på dette kunne være i *EditItemController* hvor funktionen *UpdateItem* er implementeret. Denne funktion tager imod en *FormCollection* hvori de nye værdier, der skal ændres i det gamle item, ligger. Logikken i funktionen ændrer det gamle item til de nye værdier, men returnerer det *View*, som skal vises efter *item*’et er ændret. Dermed er det ikke muligt blot at sammenligne returværdien fra funktionen med hvad der forventes, at det pågældende item er ændret til. I stedet skal der testes på om der eksisterer et *item* i listen med *items*, der har de rette værdier. Dette går igen med alle de funktioner, der har *ActionResult* som returparameter.

return RedirectToAction("ListView", "LisView");

**Figur 5** Udsnit af hvad der returneres fra UpdateItem-funktionen i EditItemController.cs

Tests til webapplikationen har ikke været et af vores store fokus områder. Selv om en af grundene til valget af *MVC*-mønsteret var at det skulle være langt mere testbart, blev funktionaliteten prioriteret højere. Dette ses også på mængden af tests i web-applikationens solution.