# Design og implementering

## Web app

Figur - WebApp mappestruktur. MVC pattern

### Design

Mappestrukturen (Figur 1) i web applikations projektet er generet af Visual Studio under oprettelsen af projektet. Den følger MVC mønstret og har derfor mapperne: Controllers, Views og Models. Models mappen er i dette projekt gjort overflødigt da de klasser der burde ligge i denne mappe ligger i et class library ved navn: ”SmartFride\_WebModels”, som har taget udgangspunkt i modellaget fra den anden applikation.

App\_Data mappen bruges ikke i dette projekt da al data der skal gemmes i denne applikation gemmes i en database.

App\_Start indeholder klasser der henter scripts og bestemmer hvilken side der skal vises under opstart af applikationen. Disse er autogenerede og ikke ændret i.

Content mappen indeholder de .cs stylesheet, som hvert view bruger for at style siderne.

Images, fonts og Scripts mapperne indeholder henholdsvis de billeder, de scripts og de fonts, som bruges i applikationen.

I Web.config (XML fil) indeholder information om hvilke assamblys applikationen skal hente, hvilke connectionsstrings den skal bruge og meget mere.

Global.asax (cs fil) kører de filer der ligger i App\_Start under opstart af applikationen.

Hver controller har et View tilknyttet, hvor controlleren sørger for funktionalitet og Viewet sørger for at rendere html kode, som browseren kan læse og brugeren herved kan se i sin browser.

Dette projekt består af 4 controllers:

* HomeCotroller
* LisViewController
* AddItemController
* EditItemController

Homecontroller’en sørger for applikationens forside og har viewet ’Index.cshtml’ tilknyttet. Controlleren eneste funktionalitet er at returnere Index viewet.

LisViewController’en henter seneste data fra databasen og viser denne data i et grid. I dette view som denne controller returnere, har brugeren mulighed for at tilføje, redigere og slette items. Hvis ”tilføj item” vælges, kaldes AddItemController’en. Vælges ”redigér item” kaldes EditItemController’en og det item der skal redigeres sendes med. ”Slet item” kalder en funktion i LisViewController’en, som sletter det valgte item og returnere LisView’et med den nye opdateret model.

### Design af views

De individuelle views følger stort set samme design som deres modparter i Fridge-app, dog med et par forskelle. Da det er en web applikation, og den derfor vil kører i en browser, er luk knappen overflødig og derfor ikke taget med. En anden følge af at det kører i browser har været at uret fra Fridge-app har virket overflødigt da uanset, hvilken enhed den tilgåes fra har let adgang til et ur på anden vis. Udover disse to ting er "rammen" den samme som på Fridge-app og har altså stadig navigations knapperne. Da rammen er den samme vil denne ikke være med på de wireframes, der er i dette afsnit. Endnu en begrundelsen for de mindre forskelle, der er på udseendet af de to applikationer, er at web applikationen, også skal kunne bruges på mindre enheder, hvor for mange ting i hvert view kunne besværliggøre brug.

#### C:\Users\Mikkel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\AddItem wireframe.pngAdditem

**Figur 2** Wireframe for Additem

Dette view svarer til "Tilføj varer" fra Fridge-app. Der er stort set ikke ændre på udseendet af dette view, den eneste forskel er, at listen over nyligt tilføjede varer er placeret under selve tilføjelses menuen i stedet for ved siden af som det kan ses ud fra det tilhørende wireframe på **Figur 2**.

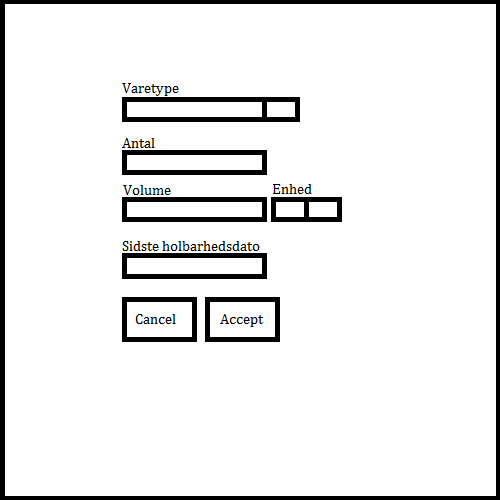
#### Edititem

Det er ved edit item, at der er sket den største forandring. I Fridge-app var denne en del af det vindue, hvor man kunne se alle vare på en given liste, på web applikationen har den fået et vindue for sig selv. Grunden til dette var, at der ved brug af mindre enheder ville gå udover brugervenligheden, hvis der var for mange ting på et view. Som det kan ses på **Figur 3**, minder den rigtig meget om både boksen med vareinformation fra Fridge-apps List view, og om Add items view.

#### Index

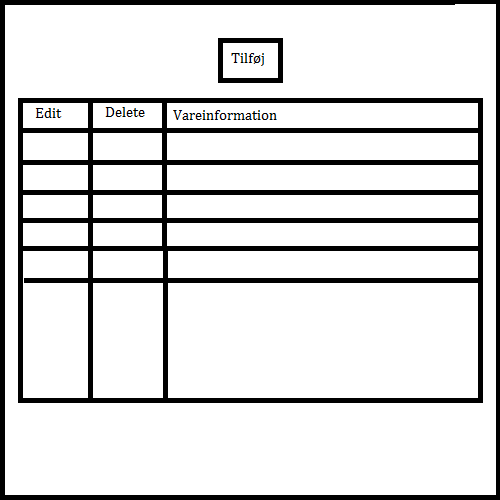
Index svarer til hovedmenuen i Fridge-app, hertil har vi valgt at beholde det design, der var lavet der til.

**Figur 3** Wireframe for EditItem



#### ListView

Som det fremgår af Figur 4 har List viewet været igennem nogle ændringer. Mest iøjefaldene er nok at boksen med vare information er væk. Dette skyldes at al information vedrørende en vare kunne vises i gridded, og at funktionaliteten vedrørende redigering af vareinformation blev flyttet ud i et view for sig. Udover dette er der ikke blevet ændret der store ved viewet udover at der i griddet til vareinformation er kommet en knap, der fører til EditItem viewet, til venstre for slet knappen.



Figur Wireframe for List view

### Implementering

#### Controllers

##### AddItemController

##### EditItemController

EditItem controlleren har 3 forskellige funktioner, den første af disse er EditItem funktionen, der tildels kan kaldes for en constructor for EditItem viewet. Her indlæses alt det data, der skal bruges i viewet, såsom varetyper fra cachen og de forskellige enheder, der kan bruges til at beskrive en varer. Udover dette ligges, den valgte vare i ViewData, og varetyperne og enhederne ligges i ViewBag. Det hele kan ses på **Kodestump 1**.

|  |
| --- |
| public ActionResult EditItem(GUIItem oldItem)  {  \_oldItem = oldItem;  \_types = new List<SelectListItem>();  \_types.Add(new SelectListItem { Text = "Varetype", Value = "Varetype", Selected = true });  foreach (var item in Cache.DbItems)  {  \_types.Add(new SelectListItem { Text = item.ItemName,Value=item.ItemName });  }  \_units = new List<SelectListItem>(){new SelectListItem{Text = "l", Value = "l"},  new SelectListItem{Text = "dl", Value = "dl"},  new SelectListItem{Text = "ml", Value = "ml"},  new SelectListItem{Text = "kg", Value = "kg"},  new SelectListItem{Text = "g", Value = "g"},  new SelectListItem{Text = "stk", Value = "stk"}};  ViewData.Add("oldItem", \_oldItem);  ViewBag.types = \_types;  ViewBag.units = \_units;  return View();  } |

**Kodestump 1** Viewets "constructor"

Den anden funktion, der dækker over den primære opgave for Controlleren er UpdateItem. Denne funktion sørger for at de i viewet indtastede data bliver gemt ned i databasen, ved at opdatere enten det tilhørende ListItem, Item eller begge dele. Funktionen består af en serie af tjek som der kan ses et eksempel på, på **Kodestump 2**. tjekkene går alle sammen på at finde ud af hvad, der er blevet ændret og hvordan det skal håndteres. I tilfælde af en vare opdateres på en sådan måde, at det stemmer overens med allerede eksisterende vare på den liste, håndteres dette ved at tælle den originale vares antal op, og slette duplikatten.

|  |
| --- |
| if (listItem.Item.ItemId == \_oldItem.ItemId)  {  if ((listItem.Item.ItemName == \_updatedGUIItem.Type))  {  listItem.Amount = Convert.ToInt32(\_updatedGUIItem.Amount);  listItem.Volume = Convert.ToInt32(\_updatedGUIItem.Size);  listItem.Unit = \_updatedGUIItem.Unit;  listItem.ShelfLife = \_updatedGUIItem.ShelfLife;  uow.ListItemRepo.Update(listItem);  uow.SaveChanges();  Cache.DalFacade.DisposeUnitOfWork();  return RedirectToAction("ListView", "LisView");  }  [...] |

**Kodestump 2** Eksempel på Check i UpdateItem funktionen

Den tredje funktion i controlleren er selectedUnit funktionen. Denne funktion sørger for at det er den valgte vares enhed, der vises i dropdownmenuen i stedet for det øverste element. Som det kan ses på **Kodestump 3**, gøres dette ved at finde enheden i listen over enheder, og sætte dens Selected property til true.

|  |
| --- |
| public static void selectedUnit(GUIItem guiItem)  {  foreach (var unit in \_units)  {  if (unit.Text == guiItem.Unit)  {  unit.Selected = true;  break;  }  }  } |

**Kodestump 3** Den valgte items enhed findes, og sættes som   
den valgte med Selected propertyen.

##### HomeController

HomeController controlleren er de første controller, der kommer i brug. Når siden tilgås kaldes dens Index metode, som returnere Index viewed, det er også i denne metode, hvor cachens data acces layer facade instantieres, cachen er beskrevet længere nede i dette dokument. Udover denne metode har den også SetCurrentList funktionen, der kaldes når brugeren har trykket på en af de tre mulige knapper i index viewed. Funktionen der kan ses på **Kodestump 1**, gør brug af cachens DAL facade til at finde den liste i databasen, der svarer til den Bruger valgte, og sætte den i cachen, før den returnere det relevante view.

|  |
| --- |
| public ActionResult SetCurrentList(string listToEdit)  {  List currentList = new List();  var uow = Cache.DalFacade.GetUnitOfWork();  currentList = uow.ListRepo.Find(l => l.ListName == listToEdit);  Cache.DalFacade.DisposeUnitOfWork();  Cache.CurrentList = currentList;  return RedirectToAction("ListView","LisView");  } |

**Kodestump 4** Her ses metoden, hvor den nuværende liste sættes i Cachen

##### LisViewController

#### Views

##### AddItem

**Figur 5** EditItem view

##### EditItem

Udover at det har fået sit eget view minder måden EditItem ser ud på rigtig meget om rediger vare fra Fridge-app. Varens information indlæses i tekstbokse, som kan redigeres i, og Bruger har mulighed for at vælge at ente gemme eller annullere de foretagegne ændringer. Viewed er ikke blevet helt som designet, hovedsagligt er det fluebenet og krydset, der ikke er kommet til at sidde som på designet. Dette skyldes bøvl med at få to forskellige Html forms til at være på samme linje. udover dette skulle det egentlig bare have været centreret.

På **Kodestump 2** ses det allerførste, der sker i viewed. Den vare Bruger valgte i List viewed indlæses fra EditItem viewets ViewData, og der kaldes en funktion, der finder ud af, hvilken enhed, der skal være valgt i enheds dropdown menuen. Alle inputfelterne i viewed er lavet med Html helpers, på **Kodestump 3** kan der ses nogle eksempler på disse. Som det kan ses er der hovedsagligt brugt TextBox helperen, men i eksemplerne fra **Kodestump 3** kan det ses at der er manipuleret lidt med nogle af dem. Det er muligt at ændre på typen af TextBox helperen så den f.eks. kun tager imod tal eller datoer. Overholdes input typen ikke, gives der besked om dette, og man kan ikke gemme ændringerne før de alle sammen er overholdt. Alle Html helperne med vareinformationer i er i en Html form, der sender informationen videre til UpdateItem funktionen i EditItem controlleren.

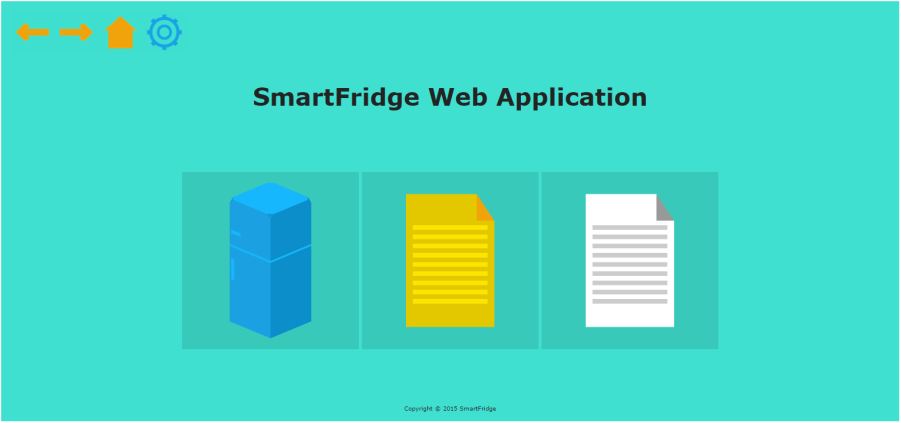
|  |
| --- |
| @{  ViewBag.Title = "EditItem";  Layout = "~/Views/Shared/\_Layout.cshtml";  GUIItem dataSource = ViewData["olditem"] as GUIItem;  EditItemController.selectedUnit(dataSource);  } |

**Kodestump 5** Her ses, hvordan den valgte items data indlæses til brug i viewed

|  |
| --- |
| @Html.TextBox("Type", dataSource.Type, new  {  @id = "VaretypeText"  })  @Html.DropDownList("types", (List<SelectListItem>)ViewBag.types , new  {  @id = "VaretypeDropDown"  })  [...]  @Html.TextBox("Amount", dataSource.Amount, new { @type = "number", @min = 0 })  [...]  @Html.TextBox("Shelflife", dataSource.ShelfLife.ToString("dd-mm-yyyy"), new{@type = "date", @autocomplete = "on"}) |

**Kodestump 6** Eksempler på forskellige typer af Html helpers.

##### Index

Index viewed der svarer til hovedmenuen fra Fridge-app er det simpleste af alle web applikationens views og består kun af tre knapper. Som det ses på **Figur 5**, ligner de to også hinanden rigtigt meget.

**Figur 6** Index view web applikationens svar på en hovedmenu

##### ListView

#### Models

Modellerne brugt i web-appen, er de samme klasser, der blev brugt i Fridge-app med undtagelse af GUIItemList. Der en model for hver type database entitet, der gøres brug af ved al database tilgang, og GUIItem klassen, som er den lokale repræsentation af en vare og derfor bruges i forbindelse med samtlige views. Modellerne er implementeret i et klassebibliotek for sig, da data access layered, der også ligger i et klasse bibliotek for sig, har behov for at kende til disse. Ellers er praksissen med at ligge modeller i et klasse bibliotek normalt kun noget, der gøres brug af i projekter med mange modeller.

#### Cache

Til wep-appen er der blevet implementeret en cache klasse, der indeholder de relevante ting for de forskellige views. Alle cache klassens attributter er static, . Især med facaden til data acces layered er det vigtigt at det er static, da flere instanser af denne kunne resultere i fejl, hvis flere instanser samtidigt forsøger at tilgå databasen. Som det fremgår af **Kodestump 1** CurrentList propertyens set metode, sørger for også at hente alle ListItems, der figurere på den nuværende liste og alle Items ned. Her hentes også samtlige items fra databasen ned. Grunden til at alle items hentes ned er at deres navne bruges som info i dropdown menuer i både EditItem og AddItem viewene, hvor det altså er relevant at få alle varetyper med, og ikke blot dem, der er tilknyttet ListItems på den nuværende liste

|  |
| --- |
| public static List CurrentList { get { return \_currentList; }  set  {  \_currentList = value;  var uow = DalFacade.GetUnitOfWork();  CurrentListItems = new List<ListItem>();  var tempList = uow.ListItemRepo.GetAll().ToList();  if (tempList.Any())  {  foreach (var Listitem in tempList)  {  if (Listitem.ListId == \_currentList.ListId)  {  CurrentListItems.Add(Listitem);  }  }  }  DbItems = uow.ItemRepo.GetAll().ToList();  DalFacade.DisposeUnitOfWork();  }  } |

**Kodestump 7** CurrentList propertyen, hvis set metode også henter alle relevante   
ListItems på listen ned

### Test

Grundet tidspres er der ikke blevet lavet decideret unit tests eller integrations tests af web-applikationen med undtagelsen af dennes DAL-lag. Alle funktioner er dog testet. Da applikationen er bygget op efter MVC mønstret er der relativt lav kobling, der ellers er ideelt i test øjemed.

**Design:**

Her beskrives det mere detaljerede design for hhv. HW og SW.

**Implementering:**

Her beskrives implementeringen i detaljer (kredsløbsberegninger, klassediagrammer, sekvensdiagrammer, state-diagrammer, aktivitetsdiagrammer. sourcekode udsnit m.m.).

**Test:**

Her beskrives de for projektet gennemførte enhedstests. Integrationstests og accepttest kommer i de følgende afsnit.

Evt. testprogrammer vedlægges på CD-ROM.