##### AddItemController

AddItem består af én constructor og 2 funktioner.

Constructoren indlæser items fra databasen vha. Unit Of Work, og tilføjer herefter alle items til ListGuiItems. model bliver sat til newGuiItems, som er de items der skal tilføjes i addNewItem-funktionen. Dette ses på Kodestump 1. Constructoren returnere et View, som sørger for at AddItemViewet bliver vist for brugeren.

|  |
| --- |
| public ActionResult AddItem()  {  var uow = Cache.DalFacade.GetUnitOfWork();  ListGuiItemTypes.Add(new SelectListItem { Text = "Varetype", Value = "Varetype", Selected = true });  foreach (var guiItemTypes in uow.ItemRepo.GetAll())  {  ListGuiItemTypes.Add(new SelectListItem { Text = guiItemTypes.ItemName });  }  model = newGuiItems;  ViewBag.ListNewGuiItems = ListGuiItemTypes;  uow.SaveChanges();  Cache.DalFacade.DisposeUnitOfWork();  return View(model);  } |

Kodestump – AddItemController’s constructor

Funktionen addNewItem tager imod de indtastede data og tilføjer dem til guiItemToAdd. Men først tjekkes der på om den holdbarhedsdato der medsendes er indtastet. Er denne ikke det, sættes det holdbarhedsdatoen til en default værdi. Ellers bliver den konverteret til den rigtige DateTime. Den skal konverteres da den modtager værdien som en string, men for at databasen kan forstå denne værdi skal den være af typen DateTime.

|  |
| --- |
| public ActionResult addNewItem(string Varetype, string Antal, string Volume, string Enhed, string Holdbarhedsdato, string ItemImgClicked)  […]  GUIItem guiItemToAdd = new GUIItem();  guiItemToAdd.ShelfLife = dblistItemDateTime; //AMOUNT READ FROM FIELD  guiItemToAdd.Amount = Convert.ToUInt32(Antal); //Antal READ FROM FIELD  guiItemToAdd.Size = Convert.ToUInt32(Volume); //Volume READ FROM FIELD  guiItemToAdd.Type = Varetype; //Varetype READ FROM FIELD  guiItemToAdd.Unit = Enhed; //unit READ FROM FIELD  foreach (var newGuiItem in newGuiItems)  {  if (newGuiItem.Type.Equals(guiItemToAdd.Type) &&  newGuiItem.Size.Equals(guiItemToAdd.Size) &&  newGuiItem.Unit.Equals(guiItemToAdd.Unit) &&  newGuiItem.ShelfLife.Equals(guiItemToAdd.ShelfLife))  {  newGuiItem.Amount += guiItemToAdd.Amount;  return null;//Viewet skal dog opdateres først  }  }  newGuiItems.Add(guiItemToAdd);  […]  model = newGuiItems;  ViewBag.ListNewGuiItems = ListGuiItemTypes;  return PartialView("~/Views/AddItem/AddItem.cshtml",model); |

Kodestump – udsnit af AdditemController’s addNewItem funktion

På Kodestump 2 ses hvordan der tjekkes om item der skal tilføjes allerede eksistere. Hvis den allerede gør det, ændre der kun på dette items Amount i stedet for at tilføje et nyt. Findes item ikke i listen bliver det tilføjet. Når den nye item er tilføjet lægges det i model et PartialView returneres, så det nye item kan ses i tabellen på siden.

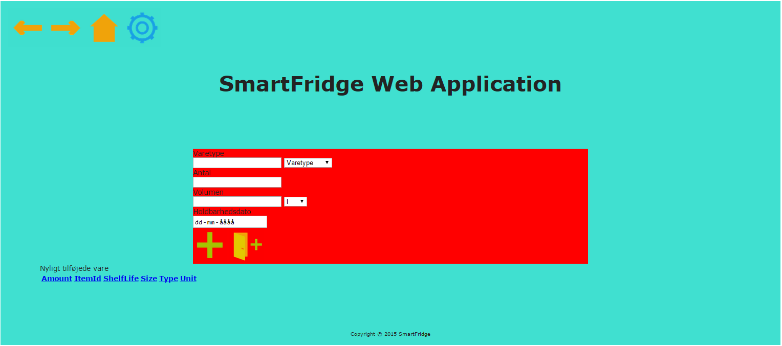
I addNewItem funktionen tjekkes der også på parameteren ’ItemImgClicked’. Hvis denne er lig ”Exit”, betyder det at knappen ’Tilføj og Exit’ er trykket på. Er dette tilfældet bliver funktionen Exit i AddItemControlleren kaldt.

|  |
| --- |
| public ActionResult Exit()  {  if (newGuiItems.Count == 0) {return null;}  var uow = Cache.DalFacade.GetUnitOfWork();  foreach (var newGuiItem in newGuiItems)  {  //Searching for item in DB  Item dbItem = uow.ItemRepo.Find(l => l.ItemName == newGuiItem.Type);  if (dbItem == null)  {  dbItem = new Item(newGuiItem.Type){StdUnit = newGuiItem.Unit, StdVolume = (int)newGuiItem.Size};  uow.ItemRepo.Add(dbItem);  }  //Searching for ListItem in DB  ListItem dbListItem = uow.ListItemRepo.Find(l =>  l.List.ListName == Cache.CurrentList.ListName &&  l.Item.ItemName == newGuiItem.Type &&  l.Unit == newGuiItem.Unit &&  l.Volume == newGuiItem.Size &&  l.ShelfLife == newGuiItem.ShelfLife);  if (dbListItem != null)  {  dbListItem.Amount += (int)newGuiItem.Amount;  }  else if (dbListItem == null)  {  dbListItem = new ListItem();  dbListItem.ShelfLife = newGuiItem.ShelfLife;  dbListItem.Amount = (int)newGuiItem.Amount;  dbListItem.Volume = (int)newGuiItem.Size;  dbListItem.Unit = newGuiItem.Unit;  dbListItem.ListId = Cache.CurrentList.ListId;  //dbListItem.List = Cache.CurrentList;  dbListItem.ItemId = dbItem.ItemId;  dbListItem.Item = dbItem;    uow.ListItemRepo.Add(dbListItem);  }  }  uow.SaveChanges();  Cache.DalFacade.DisposeUnitOfWork();  newGuiItems = new List<GUIItem>();  model = newGuiItems;  return RedirectToAction("ListView", "LisView"); |

Kodestump – Udsnit af AddItemController’s Exit-funktion

Exit funktionen (på Kodestump 3) tjekker først og fremmeste på om der er tilføjet nogen items. Er der ikke det returneres der null. Er dette ikke tilfældet skal der tjekkes på om hvert item der er tilføjet ligger i databasen. Der ledes først i item listen om det nye item eksistere. Herefter ledes der i ListItem for at se om det skulle lægge der. Findes de ikke bliver de tilføjet til databasen. Hvis det nye item ligger i ListItem, skal det nuværende items Amount tælles op. Efter alle nye item er gennemgået gemmes ændringerne i databasen og listerne med item opdateres. Til sidst returneres LisView Viewet.

##### AddItem View

Ligesom i SmartFridge-applikationen har AddItem fået sit eget view. På Figur 1 ses en rød indramning der indeholder de fire inputs: Varetype, Antal, Volumen og Holdbarhedsdato. Ydermere indeholder den ’Tilføj’ knappen symboliseret af en plus og ’Tilføj og Exit’ knappen symboliseret af en dør og et plus. Under den røde indramning er tabellen med de items der er tilføjet uden at afslutte.

Figur – AddItem View

Viewet indeholder en IEnumerable med de GuiItems der er tilføjet uden at afslutte. Ligesom de andre views bruger den samme layout (\_Layout.cshtml) som wrapper til viewet. Disse ting ses på Kodestump 4.

|  |
| --- |
| @model IEnumerable<SmartFridge\_WebModels.GUIItem>  @{  ViewBag.Title = "AddItem";  Layout = "~/Views/Shared/\_Layout.cshtml";  } |

Kodestump – Udsnit af additem.cshtml. Header

Alle inputs er inde i div klassn ”NewItem” som ses på Kodestump 5. For at alle inputs gælder det at det skal være inde i @using (Html.BeginForm()){}. Dette er gjort for at fortælle programmet hvilken information den skal sende med i til addNewItem funktionen. Varetypen har to muligheder for at få et input. Enten kan brugeren vælge at skrive det ind i <Input> objektet eller vælge det på den @html.DropDownList der er ved siden af inputtet.

|  |
| --- |
| <div class="NewItem" style="background-color: red; margin: 0 22em 0 22em">  @using (Html.BeginForm("addNewItem", "AddItem"))  {  <label>Varetype</label>  <br />  <input type="text" name="Varetype" id="VaretypeText" required/>    @Html.DropDownList("ListNewGuiItems", (IEnumerable<SelectListItem>)ViewBag.ListNewGuiItems, new  {  @id = "VaretypeDropDown"  })  <br />  <label>Antal</label>  <br />  <input type="number" name="Antal" min="0" required />  <br />  […]  <input type="image" src="~/Images/Add.png" alt="Submit" class="AddImg" id="addItemImgClicked" name="ItemImgClicked" value="Add" />  <input type="image" src="~/Images/AddAndExit.png" alt="Submit" class="AddImg" name="ItemImgClicked" value="Exit" />  } |

Kodestump – Udsnit af AddItem.cshtml. Inputs

AddItem viewet indeholder også et javascript der sørger for at lave en Varetype dropdown og en kalender der popper op på under Holdbarhedsdato inputtet. På Kodestump 6 ses hvordan kalender funktionen virker. Her ses det at den nuværende måned plus 1, dag plus 1 og år plus 1 bliver lagt ind som standard værdier. Bliver andet valgt sendes det med når funktionen addNewItem bliver kaldt.

|  |
| --- |
| var dNow = new Date();  var utcdatePlusAYear = (dNow.getMonth() + 1) + '-' + dNow.getDate() + '-' + (dNow.getFullYear() + 1);  $('#addItemImgClicked').mouseover(function (event)  {  if ($('#LifeOfItem').text == null) {  $('#LifeOfItem').text(utcdatePlusAYear);  }  $('#LifeOfItem').text(utcdatePlusAYear);  }); |

Kodestump 6 – kodesnut fra addItemScripts.js

##### LisViewController

LisViewController indeholder én constructor, 2 funktioner og en liste med de items der skal vises på ListView’s view. Constructoren henter de items der ligger i Chache klassen ind og tjekker først på om denne liste er tom. På Kodestump 7 tjekkes det både på CurrentListItems og DbItem om id’erne passe samme på hvert item. Hvis ListItem og Item’s id matcher skal navnet fra Item og amount, volumen og unit fra ListItem lægges sammen til ét nyt item og lægges over i tempData listen. Efter der er blevet tjekket på alle items i listerne bliver de lagt ind i model som herefter returneres sammen med viewet.

|  |
| --- |
| Cache.CurrentList = Cache.CurrentList;  List<GUIItem> tempData = new List<GUIItem>();  if(Cache.CurrentListItems.Any())  {  foreach (var listItem in Cache.CurrentListItems)  {  foreach (var item in Cache.DbItems)  {  if(item.ItemId == listItem.ItemId)  {  GUIItem temp = new GUIItem(item.ItemName, (uint)listItem.Amount, (uint)listItem.Volume, listItem.Unit){ItemId = item.ItemId};  tempData.Add(temp);  }    }  }  }  model = tempData;  return View(model); |

Kodestump – Udsnit af ListViewController’s constructor.

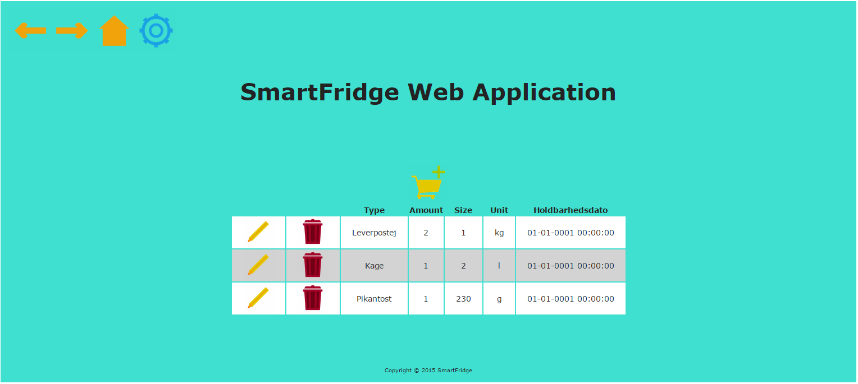
Den første funktion i ListViewController er ToEditItem, hvis opgave er at få sendt det rigtige GUIItem over til EditItemController’en. Da det ikke var muligt at sende Id’et med fra viewet, så tjekker denne funktion på det medsendte GUIItem’s attributter om de stemmer overens med et item på listen. Når dette bliver fundet bliver EditItem funktionen kaldt i EditItemControlleren, og det fundne item sendes med over.

Den anden funktion i ListViewController er DeleteSelectedItem. Ligesom ToEditItem funktionen får den et GUIItem med fra viewet og tjekker på om dette item eksistere i model listen. Når/hvis det medsendte item bliver fundet konverteres IEnumerable model om til en liste så den kan rette i. herefter slettes itemet fra listen og den nye liste, uden det slettede item, lægges i model igen. Herefter skal itemet slettes fra databasen, hvilket gøres ved at finde og slette den i Cache’ns CurrentListItems. Til sidst gemmes ændringerne, chache’ns adgang til databasen bliver disposed og constructoren, der returnere et LisView viewet, bliver kaldt.

|  |
| --- |
| […]  foreach (var item in model)  {  if (item.Type == itemToDelete.Type && item.Amount == itemToDelete.Amount && item.Size == itemToDelete.Size && item.Unit == itemToDelete.Unit)  {  List<GUIItem> tempList = model.ToList();  tempList.Remove(itemToDelete);  model = tempList;  foreach (var dblistitem in Cache.CurrentListItems)  {  if (dblistitem.ItemId == item.ItemId)  {  uow.ListItemRepo.Delete(uow.ListItemRepo.Find(l =>  l.ItemId == dblistitem.ItemId &&  l.ListId == dblistitem.ListId &&  l.Unit == dblistitem.Unit &&  l.Amount == dblistitem.Amount &&  l.ShelfLife == dblistitem.ShelfLife &&  l.Volume == dblistitem.Volume));  }  }  }  }  uow.SaveChanges();  Cache.DalFacade.DisposeUnitOfWork();  return RedirectToAction("ListView", "LisView");  […] |

Kodestump – Udsnit af DeleteSelectedItem funktionen i LisViewController

##### ListView

ListView’et indeholder en ’Tilføj Item’-knap til at navigere brugeren hen AddItemView. Denne knap er symboliseret som en indkøbsvogn og et plus. Ydermere indeholder ListView en tabel med de items der er på den pågældende liste. Hvert item har to knapper ud fra sig. Et ’EditItem’-knap symboliseret af en blyant og en ’DeleteItem’-knap symboliseret af en skraldespand. Disse knapper bliver tilføjet når griddet laves. På Kodestump 9 ses det hvordan knappen er tilføjet til griddet. Cols, som griddet hedder, bliver lavet hver gang viewet bliver renderet. For hvert item i model listen bliver der lavet syv nye columns. Det første to collumns er redigér og slet knapperne. Herefter bliver hver attribut tilføjet, som det ses på Kodestump 9 hvor Type bliver tilføjet.

Figur – ListView’s View

Efter cols er blevet oprettet, sørger @grid.GetHtml for at rendere og style det på viewet.

|  |
| --- |
| […]  IEnumerable<WebGridColumn> cols =  new List<WebGridColumn> {  new WebGridColumn{  Format =  @<text>  <a href="@Url.Action("ToEditItem", "LisView", new GUIItem(item.Type, item.Amount, item.Size, item.Unit))"><img src="~/Images/Edit.png" class="RowImg" /></a>  </text>  […]  new WebGridColumn{  Header="Type",  Format = @<text>  <span class="display">  <label id="lblType">@item.Type</label>  </span>  </text>  },  […]  <div align="center">  @grid.GetHtml(columns:cols,  rowStyle:"oddRow", alternatingRowStyle:"evenRow")  </div> |

Kodestump – Udsnit af ListView.cshtml