Applikation MVC

Erik Gustafsson (a23erigu)

Table of Contents

Intro

Denna rapport är skapad för inlämningsuppgiften Applikation MVC från kursen Databaskonstruktion G1F.
Inlämningen handlar om ett .net program som ska skapa och visa upp en hemsida. Rapporten kommer främst fokusera på lösningar till problem som uppgiften vill at man löser.
Programmet är skrivet i .net så koden kommer bestå av en kombination av C# och html.
VG försöktes nås med detta program så de uppgifterna kommer tas upp i slutet.
Några delar kommer vara väldigt lika Applikation PHP inlämningen då nästan samma funktionaliteter har skapats.

MVC

Till programmet så användes 2 modeller, 2 vyer och 1 controller.
De två vyerna som används är Index.cshtml och Tomteverkstad.cshtml. Index innehåller endast introsidan medan Tomteverkstaden innehåller resten.
Modellerna som finns är TomtenisseModel.cs och LeksakModel.cs. Programmet är skapade så att varje tabell i databasen har sin egen modell och Tomtenissetabellen och Leksaktabellen är det enda som används så det är endast de som får modeller.
Den enda controller som används är HomeController.cs då programmet var så litet att det inte behövdes fler. Mestadels av allt som händer i HomeController är i Tomteverkstad då den är sidan som används.

```
string formID = form["formID"];

if (formID == "searchTomtenissar") {

   // kod för att söka
}

else if (formID == "create") {

   // kod för att skapa
}

else if (formID == "update") {
   // kod för att updatera
}
```

Figur 1: hur HomeController fungerar.

I Tomteverkstad i HomeController så används denna kod för att kunna veta vad det är som ska köras när något på hemsidan händer. Den gör detta genom att kolla formID vilket är ett hidden fält som alla forms har.

Lägga till data i tabeller

I programmet så körs en skapa-funktion för att skapa tomtenissar. För att skapa tomtenissar så behöver användaren skicka med namn, ID, mängden nötter och mängden russin tomtenissen ska få som lön

```
<form method="post" action="@Url.Action("Tomteverkstad", "Home")">
    <p> Create tomtenisse </p>
    Name of tomtenisse
    <input type="text" name="Name" value=""> <br>
    ID of tomtenisse
    <input type="text" name="CId" value=""> <br>
    Amount of nuts earned
    <input type="number" name="Nuts" value=""> <br>
    Amount of raisin earned
    <input type="number" name="Raisin" value=""> <br>
    <input type="hidden" name="formID" value="create">
    <input type="submit" name="submit" value="Submit">
</form>
```

Figur 2: form för skapandet av tomtenissar.

Metoden har inputs för namn, ID, nötter och russin då det behövs för att skapa tomtenissar men de har även ett hidden-fält vid namn formID. Alla form-metoder i programmet har ett sådant hidden-fält då det används för att veta vilken form det är som aktiverats.

```
@ViewBag.name = form["Name"];
String ID = form["CID"];
String nuts = form["Nuts"];
String raisin = form["Raisin"];
if (@ViewBag.name.Count == 0 || ID == null || nuts == null || raisin == null) {
   ViewBag.updateStatus = "create needs all values";
}
else {
   int.TryParse(nuts, out int intNuts);
   int.TryParse(raisin, out int intRaisin);
   tm.CreateTomtenisse(@ViewBag.name, ID, intNuts, intRaisin);
   ViewBag.createStatus = "create might be successful";
}
```

Figur 3: controller för skapandet av tomtenissar.

Efter att programmet kollat att det är skapa-form som används så börjar det med att ta ut värdera och kolla att de alla är satta. Efter att ha kollat att värdera är satta så kör det CreateTomtenisse

```
public DataTable CreateTomtenisse(String Name, String ID, int Nuts, int Raisin) {
   MySqlConnection dbcon = new MySqlConnection(connectionString);
   dbcon.Open();
   MySqlDataAdapter adapter = new MySqlDataAdapter("insert into
   Tomtenisse(Namn, IdNr, Nötter, Russin) values(@inputName , @inputID ,
   @inputNuts , @inputRaisin)", dbcon);
   adapter.SelectCommand.Parameters.AddWithValue("@inputName", Name);
   adapter.SelectCommand.Parameters.AddWithValue("@inputID", ID);
   adapter.SelectCommand.Parameters.AddWithValue("@inputNuts", Nuts);
   adapter.SelectCommand.Parameters.AddWithValue("@inputRaisin", Raisin);
   DataSet ds = new DataSet();
   adapter.Fill(ds, "result");
   DataTable createTable = ds.Tables["result"];
   dbcon.Close();


   return createTable;
}
```
Figur 4: CreateTomtenisse funktionen.

Här så körs sql koden för att skapa tomtenissar med namn, ID, nötter och russin som inputs. Det försöktes att göra så koden skriver ut error-meddelande om något går fel med sql satsen.

Dropdown (Listbox) genererad från data

Dropdown används i programmet för att vissa namnet på alla tomtenissar som existerar.

```
<select name="dropdown">

   @foreach (var Tomtenisse in ViewBag.AllTomtenissar.Rows)

   {

      <option>@Tomtenisse[0]</option>

   }

</select>
```
Figur 5: cshtml för dropdown.

Cshtml koden för dropdown är bara en select med en for loop för att gå igenom alla namn. Denna kod har ingen check för tomtenissar på sig då det alltid ska finnas tomtenissar i databasen.

```
DataTable AllTomtenissarData = tm.GetTomtenissar();
ViewBag.AllTomtenissar = AllTomtenissarData;
```
Figur 6: hämta tomtenissar

I controllern så körs bara en enkel kod för att få tag på alla tomtenissar och skicka den till vyn.

Sql satsen för att hämta tomtenissar tas upp i Exekvera en procedur då den ligger i en procedur.

Sökning i databasen

Sökningen används för att kunna hitta information om någon tomtenisse genom deras namn.

```
<form method="post" action="@Url.Action("Tomteverkstad", "Home")">
   <p> Search for Tomtenissar </p>
   <input type="hidden" name="formID" value="searchTomtenissar">
   <input type="text" name="Name" value= @ViewBag.name>
   <input type="submit" name="submit" value="Submit">
</form>
```
Figur 7: form för att söka.

Denna form tar endast in ett namn då det är det enda som behövs för att kunna söka.

```
@if(ViewBag.Tomtenissar != null &&
((System.Data.DataTable)ViewBag.Tomtenissar).Rows.Count > 0){


  @foreach (var Tomtenisse in ViewBag.Tomtenissar.Rows)
  {
    <pre>
      @for (var i = 0; i < ViewBag.Tomtenissar.Columns.Count; i++)
      {
        <div>@ViewBag.Tomtenissar.Columns[i] : @Tomtenisse[i]</div>
      }
      @Html.ActionLink("Delete", "DeleteTomtenisse", "Home", new{Name =
      @Tomtenisse[0], PNR = @Tomtenisse[1]}, null)
    </pre>
  }
}
else{
  <p>no data found</p>
}
```

Figur 8: skriva ut den sökta tomtenissen.

Denna kod körs efter formen i Tomteverksta.cshtml för att skriva ut resultatet av sökningen. Den skriver även ut kolumnerna till varje element så man vet vilken data det är (anledningen att kolumnerna är på svenska är för att databasen är svensk). Det skrivs även ut en länk till varje användare som kan användas för att ta bort dem men den tas upp i Generera länkar.

```
@ViewBag.name = form["Name"];

DataTable TomtenissarData = tm.GetTomtenissarByName(@ViewBag.name);

ViewBag.Tomtenissar = TomtenissarData;
```

Figur 9: hämta alla tomtenissar i controller.

Här hämtas alla tomtenissar och skickas till vyn. Anledningen @ViewBag.namn används är så namnet sparas i sökfältet.

```
public DataTable GetTomtenissarByName(String Name) {

   MySqlConnection dbcon = new MySqlConnection(connectionString);

   dbcon.Open();

   MySqlDataAdapter adapter = new MySqlDataAdapter("select * from Tomtenisse
   where Namn = @inputName", dbcon);

   adapter.SelectCommand.Parameters.AddWithValue("@inputName", Name);

   DataSet ds = new DataSet();

   adapter.Fill(ds, "result");

   DataTable tomtenisseTable = ds.Tables["result"];

   dbcon.Close();


   return tomtenisseTable;

}
```

Figur 10: sql sats för att hämta tomtenissar beroende på namn.

Här hämtas alla tomtenissar som har de namn som skickas in.

Förändring av innehållet i databasen

Det fins två sätt att ändra i programmet, en update och en delete. Denna del kommer bara prata om updaten medan deleten tas upp i Generera länkar. Updaten finns för att kunna göra tomtenissar till chefnissar och tillbaka.

```
<form method="post" action="@Url.Action("Tomteverkstad", "Home")">
    <p> Make chefnisse </p>
    Name of tomtenisse
    <input type="text" name="Name" value=""> <br>
    ID of tomtenisse
    <input type="text" name="ChefId" value=""> <br>
    Shoe size <br>
    <input type="radio" id="none" name="Shoesize" value="none">
    <label for="none">None</label> <br>
    <input type="radio" id="mini" name="Shoesize" value="mini">
    <label for="mini">Mini</label> <br>
    <input type="radio" id="medium" name="Shoesize" value="medium">
    <label for="medium">Medium</label> <br>
    <input type="radio" id="maxi" name="Shoesize" value="maxi">
    <label for="maxi">Maxi</label> <br>
    <input type="radio" id="ultra" name="Shoesize" value="ultra">
    <label for="ultra">Ultra</label> <br>
    <input type="radio" id="mega" name="Shoesize" value="mega">
    <label for="mega">Mega</label> <br>
    <input type="hidden" name="formID" value="update">
    <input type="submit" name="submit" value="Submit">
</form>
```

Figur 11: hämta data för uppdatering

Detta är den största form som används i programmet då de har radioknappar för att bestämma storleken av skorna.

```
else if (formID == "update") {
    @ViewBag.name = form["Name"];
    String ID = form["ChefID"];
    String ShoeSize = form["ShoeSize"];
    if (@ViewBag.name.Count == 0 || ID == null || ShoeSize == null) {
        ViewBag.updateStatus = "update needs all values";
    }
    else {
        if (ShoeSize == "none")
        {
            tm.MakeShoeSizeNull(@ViewBag.name, ID);
        }
        else
        {
            tm.UpdateShoeSize(ShoeSize, @ViewBag.name, ID);
        }
        ViewBag.updateStatus = "update might be successful";
    }
}
```

Figur 12: update i controllern

Update i controllern har två olika funktioner den kan köra då att göra skostorlek till null kräver sin egen.

```
public DataTable UpdateShoeSize(String ShoeSize, String Name, String ID) {
   MySqlConnection dbcon = new MySqlConnection(connectionString);
   dbcon.Open();
   MySqlDataAdapter adapter = new MySqlDataAdapter("update Tomtenisse set
   Skostorlek = @inputShoeSize where Namn = @inputName and IdNr = @inputID",
   dbcon);
   adapter.SelectCommand.Parameters.AddWithValue("@inputShoeSize", ShoeSize);
   adapter.SelectCommand.Parameters.AddWithValue("@inputName", Name);
   adapter.SelectCommand.Parameters.AddWithValue("@inputID", ID);
   DataSet ds = new DataSet();
   adapter.Fill(ds, "result");
   DataTable updateTable = ds.Tables["result"];
   dbcon.Close();


   return updateTable;
}
```

Figur 13: uppdatera skostorleken

Vanliga uppdatering av skostorlek tar namn, id och vilken storlek de ska uppdatera med.

```
public DataTable MakeShoeSizeNull(String Name, String ID) {

   MySqlConnection dbcon = new MySqlConnection(connectionString);

   dbcon.Open();

   MySqlDataAdapter adapter = new MySqlDataAdapter("update Tomtenisse set
   Skostorlek = NULL where Namn = @inputName and IdNr = @inputID", dbcon);

   adapter.SelectCommand.Parameters.AddWithValue("@inputName", Name);

   adapter.SelectCommand.Parameters.AddWithValue("@inputID", ID);

   DataSet ds = new DataSet();

   adapter.Fill(ds, "result");

   DataTable updateTable = ds.Tables["result"];

   dbcon.Close();


   return updateTable;

}
```
Figur 14: uppdatera skostorleken med null

Null-varianten av uppdatering av skostorlek tar bara namn och id då de alltid ska göra skostorlek till null.

Anledningen det behövs en speciell för null är då html inte kan skick null som ett svar så man får i stället skapas en egen sql-sats för det

Exekvera en procedur

Det finns två procedurer som används i programmet men bara en av dem kommer tas upp här (den andra tas upp i VG Exekvera en procedur med parameter).
En av proceduren som exekveras i programmet används för att hämta alla tomtenissar.

```
public DataTable GetTomtenissar() {

    MySqlConnection dbcon = new MySqlConnection(connectionString);

    dbcon.Open();

    MySqlDataAdapter adapter = new MySqlDataAdapter("CALL getNissar", dbcon);

    DataSet ds = new DataSet();

    adapter.Fill(ds, "result");

    DataTable tomtenisseTable = ds.Tables["result"];

    dbcon.Close();


    return tomtenisseTable;

}
```

Figur 15: procedur för att hämta tomtenissar.

Att göra en procedur är väldigt likt en vanligt sql sats bara att man anropar proceduren i stället för en sql-sats.

Två olika databastabeller

Två tabeller visas i programmet, den första är Tomtenissar (har prats om tidigare i texten) och den andra är leksaker. I programmet går det att söka på leksaker beroende på pris. Hur informationen hämtas för leksaker kommer tas upp i VG Exekvera en procedur med parameter.

```
<form method="post" action="@Url.Action("Tomteverkstad", "Home")">
   <p> Get toys by prise </p>
   Price of toy
   <input type="number" name="Prise" value=""> <br>
   <input type="hidden" name="formID" value="searchLeksaker">
   <input type="submit" name="submit" value="Submit">
</form>
```

Figur 16: form för att söka på leksaker.

Denna form tar endast in pris på leksaker när man ska söka.

```
@if (ViewBag.leksaker != null &&
((System.Data.DataTable)ViewBag.leksaker).Rows.Count > 0) {

   @foreach (var Leksak in ViewBag.leksaker.Rows)
   {
      <pre>
         @for (var i = 0; i < ViewBag.leksaker.Columns.Count; i++)
         {
            <div>@ViewBag.leksaker.Columns[i] : @Leksak[i]</div>
         }
      </pre>
   }
}
else {
   <p>no data found</p>
}
```

Figur 17: för att skriva ut leksakerna.

Denna är nästan identisk till koden för att skriva ut sökta tomtenissar.

```
@ViewBag.Prise = form["Prise"];

int.TryParse(@ViewBag.Prise, out int prise);

LeksakModel Lm = new LeksakModel(_configuration);

DataTable LeksakData = Lm.GetleksakByPrise(prise);

@ViewBag.leksaker = LeksakData;
```
Figur 18: hämta leksakerna i controller.

För att hämta leksaker behövs leksaks-modellen användas.

Generera länkar

Det görs en länk i detta program och den finns när man söker på tomtenissar för att kunna ta bort tomtenissar.

```
@Html.ActionLink("Delete", "DeleteTomtenisse", "Home", new{Name =
@Tomtenisse[0], PNR = @Tomtenisse[1]}, null)
```
Figur 19: för att skapa länken.

Länken innehåller namnet och id på användaren som ska tas bort. Dessa skickas sedan till sin egen del i HomeContoller kallad DeleteTomtenisse.

```
public IActionResult DeleteTomtenisse(String Name, String PNR)

{

   TomtenisseModel tm = new TomtenisseModel(_configuration);

   tm.DeleteTomtenisse(Name, PNR);



   return RedirectToAction("Tomteverkstad", "Home");

}
```
Figur 20: hur DeleteTomtenisse ser ut i controllern.

Controllern skickar namnet och id till modellen för att tas bort och går sedan tillbaka till Tomteverkstad.

VG Hidden-fält

Som nämnt förut i rapporten så används hidden-fält i alla forms för att controllern ska kunna veta vilken from det är som har körts.

```
string formID = form["formID"];

if (formID == "searchTomtenissar") {

   // kod för att söka
}

else if(formID == "create") {

   // kod för att skapa
}

else if (formID == "update") {
   // kod för att updatera
}
```

Figur 1: hur HomeController fungerar.

Alla hidden-fält har namnet av formID.

```
<form method="post" action="@Url.Action("Tomteverkstad", "Home")">

   <p> Search for Tomtenissar </p>

   <input type="hidden" name="formID" value="searchTomtenissar">

   <input type="text" name="Name" value= @ViewBag.name>

   <input type="submit" name="submit" value="Submit">

</form>
```

Figur 7: form för att söka.

Exempel på hur ett av hidden-fälten ser ut.

VG Exekvera en procedur med parameter

Denna del inkluderar den andra proceduren som fanns kvar i programmet.
Denna procedurers används till att hitta leksakerna till den andra tabellen som vissas.

```
public DataTable GetleksakByPrise(int Prise)

{

   MySqlConnection dbcon = new MySqlConnection(connectionString);

   dbcon.Open();

   MySqlDataAdapter adapter = new MySqlDataAdapter("Call
   getLeksakerPåPris(@inputPrise)", dbcon);

   adapter.SelectCommand.Parameters.AddWithValue("@inputPrise", Prise);

   DataSet ds = new DataSet();

   adapter.Fill(ds, "result");

   DataTable LeksakerTable = ds.Tables["result"];

   dbcon.Close();



   return LeksakerTable;

}
```

Figur 21: procedur med parameter

Proceduren är inte så annorlunda från en vanlig proceduren mer än att denna behöver en parameter.