Laborationsrapport

Moment 2 – ASP.NET Core MVC
DT102G, ASP.NET med C#

Författare: Rebecca Jönsson Seiron, rese1800@student.miun.se
Termin, år: VT, 2020

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

Sammanfattning

Syftet är att skapa ett projekt som bygger på MVC-designmönstret och förstå filfördelning och arkitektur, routings och vart logik ska placeras och varför.

Uppgiften bestod i 2 delar, del 1 räcker för godkänt, del 2 ger chans till VG. Krav för del 1 är:

- Att ni verkligen använder models, controller och vyer.
- På något sätt ändra grundroutingen.
- Skicka data: manipulera/uppdatera/beräkna i controller, mata in/interagera med i Views, möjligtvis spara/hämta i Model (utan databaskoppling).
- Ha minst tre separata Views.
- Skicka data från controllern och vyn med tre varianter: ViewBag, ViewData och parameterpassning med en vy-modell i grunden.Förklaring på parameterpassningen: actionmetoden returnerar return view(namnet_på_en_instans_av_en_modell)i stället för enbart en return view().
- Spara data så att det finns tillgängligt då man går från en sida till en annan. Session och/eller Cookie.

Krav för del 2 är:

- Skapa någon slags partial som laddas in och används i minst två Views.
- Implementera en enkel validering för ett eller fler input-fält.
- Använda minst tre olika typer av inmatningskontroller i vyn. Testa något mer än textbox: dropdownlist, radiobuttons, checkboxar eller något som kom med HTML5. (Se bara till att din webbläsare klarar av kontrollern.)

Både del 1 och del 2 är lösta.

Innehållsförteckning

1 Konstruktion

Först kollade jag på Mattias video[1] gällande hur man kopierar en webbplats. En bit in började jag fundera på vilken av de tidigare uppgifter vi gjort som skulle kunna passa in, valet stod mellan projektuppgiften i JavaScripbaserad webbutveckling och projektuppgiften i Webbutveckling II. Jag valde den förstnämnda, framförallt eftersom jag har struktur och kod relativt färskt i minnet.

Ett ”Empty” projekt skapades med Visual Studio. Det första jag gjorde var att lägga in de mappar som behövs (wwwroot, css, js, images, Models, Controllers och Views).

I Startup.cs lades routingen till för MVC, möjlighet att kunna använda staticfiles, addControllerWithViews (och session lades också till efter att ha kollat Pers videos). Samt hur MVC routingen skulle agera.

HomeController skapades, här finns Index och Kontakt och dessa returnerar bara sina egna views, funktionerna högerklickades och en ”add View” gjordes. Från https://guarded-castle-72495.herokuapp.com/ kopierades all HTML kod, detta gjordes via ”Inspektera” eftersom ”Visa sidkälla” returnerar i stor sett ingenting (då webbplatsen är byggd i React). Den enda skillnaden mellan webbplatsen gjord för denna uppgift och projektuppgiften i föregående kurs är att den nu är uppdelad över flera sidor istället för att ligga på en och samma sida.

All JavaScript-kod kopierades och klistrades in i en JavaScript-fil döpt ”main-.js”. Menyn fick hamna i en egen js-fil (”menu.js”) då detta var den enda kodsnutt som gällde genom alla sidor. Css kopierades på samma vis som HTML-koden och lades i mappen css (main.css), bilder kopierades från projektmappen till ”images”.

Sharedmappen skapades och här ligger ”_ViewImports”, ”_ViewStart” samt ”_Layouts”. ”_ViewStart” pekar mot ”_Layouts”, ”_Layouts” är ”grundlayouten” för uppgiften vilket innebär att allt som är konstant gällande struktur och design hamnar här, mao: Header, footer, meny och koden för ”menu.js”. I denna fil finns ”@renderBody()”, denna returnerar innehållet i den aktuella ”Viewn”.

”_ViewImports”har import för Taghelpers.

Menyn fixades till med taghelpers.

```
<div id="header-wrapper">
    <div id="header">
        <div id="logo"><a href="#"><img src="/images/logo.png" alt="Logotyp"></a></div>
        <button class="hamburger">
            <span class="hamburger__box"><span class="hamburger__inner"></span></span>
        </button>
        <div class="navigation">
            <ul class="navigation__list">
                <li class="navigation__item"><a asp-controller="Home" asp-action="Index" asp-fragment="about">Om oss</a></li>
                <li class="navigation__item"><a asp-controller="Home" asp-action="Index" asp-fragment="menu">Menu</a></li>
                <li class="navigation__item "><a id="bokbord" asp-controller="Booking" asp-action="BokaBord">Boka bord</a></li>
                <li class="navigation__item "><a  asp-controller="Booking" asp-action="Seebookings">Dina Bokningar</a></li>
                <li class="navigation__item"><a asp-controller="Home" asp-action="Kontakt">Kontakt</a></li>
            </ul>
        </div>
    </div>
</div>
```

Detta var relativt straight forward, jag stötte på lite problem när jag kom till ankartaggar inom den egna sidan, lösningen efter en del googlande var ”asp-fragment” (som kan ses för sidorna ”Om oss” och ”Meny”).

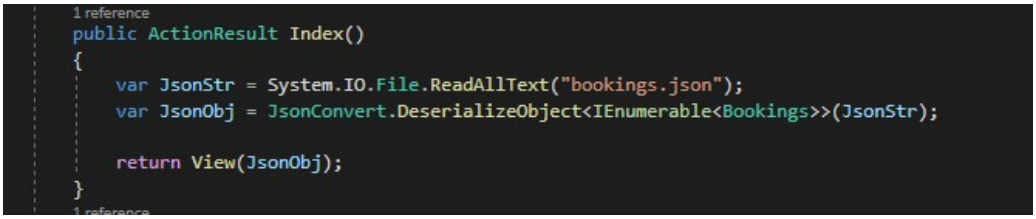
Både Index och Kontakt var enkla att skapa och innehåller inget intressant för denna uppgift så jag väljer att inte ta upp dem närmre. All funktionalitet ligger på sidan ”Boka bord” och ”Dina bokningar”. Jag valde att använda mig av json för lagring av data. En ny Controller skapades (”bookingController”) samt en Model (”Booking”). I modellen gjordes ”sets” och ”gets” för de värden jag vill skicka vidare till jsonfilen. (Email fick en ”required” för formvalidation)

```
namespace moment_2.Models
{

    42 references
    public class Bookings
    {
        1 reference
        public Bookings()
        {
        }

        14 references
        public int Id { get; set; }

        3 references
        public string Name { get; set; }

        2 references
        public int Number { get; set; }

        6 references
        public DateTime Date { get; set; }

        5 references
        public string Time { get; set; }
        [Required]
        3 references
        public string Email { get; set; }

        5 references
        public int Table { get; set; }

    }
}
```

I bookingController skapades (genom add controller och MVC Controller class med medföljande funktionalitet för CRUD). följande funktioner:

- Index()
- Rebook()
- Seebookings()
- Cancel()
- Details()
- BokaBord() + IFromCollection
- BokaBord2() + IFormCollection
- Edit() + IFormCollection
- Delete() med IformCollection

Index() returnerar ”confirmation” vid en bokning. Det är därför Json-filen läses in i denna funktion och konverteras.



I index-viewn hämtas denna in genom modellen genom att deklarera:

@model Ienumerable<moment_2.Models.Bookings>
denna möjliggör att kunna loopa igenom modellen och få ut värden från jsonfilen.

```
@foreach (var booking in Model)
{
    if (@booking.Id == @Model.Last().Id)
    {

        string date = @booking.Date.ToString("yyyy-MM-dd");
        <div id="hide2">
            <h2>Tack för din bokning @booking.Name</h2>
            <div id="bookings2" data-aos="fade-right">
                <div className="block2">
                    <p>
                        Datum: @date, klockan: @booking.Time
                        Bord: @booking.Table

                    </p>
                </div>
            </div>
        </div>
    }
}
```

För att kunna visa enbart den senaste bokning som gjorts görs här en if-sats. Om boknings ID är detsamma som det sista (högsta) idt i arrayn läs ut enbart denna info. För att slippa få ut tiden i min ”Date” så görs denna om till en string som läser ur år, månad, dag (yyyy-MM-dd)

```
{
  "Id": 4,
  "Name": "l",
  "Number": 2,
  "Date": "2020-03-02T00:00:00",
  "Time": "18:00",
  "Email": "r@h.com",
  "Table": 8
},
```

BokaBord() är viewn för ”Boka bord”, det är i denna man kan boka bord, även denna behandlar Jsonfilen och denna läses ut i Viewn genom att hämta det sista Idt. Denna ligger i en hidden input och skickas sedan vidare till BokaBord(IFormCollection collection).

```
<form id="form" asp-action="BokaBord" method="post">
    <label class="hidd">Namn</label><input class="hidd" name="Name" type="text" id="names">
    <label class="hidd">Antal</label><input name="Number" class="hidd" type="number" id="number"><br>
    <label class="hidd">Datum</label><input name="Date" class="hidd" type="date" id="date">
    <label class="hidd">Tid</label><input name="Time" class="hidd" type="time" id="time"><br>
    <label class="hidd">E-post</label><input name="Email" class="hidd" type="email" id="email"><br>
    <input class="hidden" type="number" name="Id" value="@Model.Last().Id" />
    <input class="hidden" type="number" name="Table" id="taab" />
    <div id="cont">
        <i id="italic">*Alla fält måste fyllas i för att kunna gå vidare.</i>

        <div class="hidd" id="conti">Gå vidare</div>
        <input type="submit" name="submit1" id="hideSubmit" value="Boka" />
    </div>

</form>
```

BokaBord(IFormCollection collection) i sin tur sparar bokningen till jsonfilen. Hur man gjorde detta var ju inte helt självklart, i Pers videos tas det inte upp hur man sparar, uppdaterar eller raderar från en jsonfil. Mattias film tog enbart upp hur man läser in jsonfiler. Efter endel googlande hittade jag hur man gjorde på Stackoverflow[2]. Det tog även ett tag innan jag klurade ut hur man omvandlar ett värde till DateTime.

```
0 references
public ActionResult BokaBord(IFormCollection collection)

{

    try
    {
        // Convert date input to correct datatype (DateTime)
        string inDate = collection["Date"];
        DateTime outDate = Convert.ToDateTime(inDate);

        // Read existing json data
        var jsonData = System.IO.File.ReadAllText("bookings.json");
        // De-serialize to object or create new list
        var book = JsonConvert.DeserializeObject<List<Bookings>>(jsonData)?? new List<Bookings>();

        // Add 1 to id.
        int newId = Convert.ToInt32(collection["Id"]) + 1;
        // Add booking
        book.Add(new Bookings()
        {

        Id = newId,
            Name = collection["Name"],
            Table = Convert.ToInt32(collection["Table"]),
            Number = Convert.ToInt32(collection["Number"]),
            Email = collection["Email"],
            Date = outDate,
            Time = collection["Time"]


        }) ;


        // Update json data string
        jsonData = JsonConvert.SerializeObject(book);
        System.IO.File.WriteAllText("bookings.json", jsonData);

        return RedirectToAction(nameof(Index));
    }
    catch
    {
        var JsonStr = System.IO.File.ReadAllText("bookings.json");
        var JsonObj = JsonConvert.DeserializeObject<IEnumerable<Bookings>>(JsonStr);

        return View(JsonObj);

    }
}
```

Kort förklarat:
En string för inDate skapas genom att hämta (Date) värdet från formuläret. Detta omvandlas till en DateTime för att matcha deklarationen i (Bookings)modellen.

Jsonfilen läses igenom och deserialiseras, om denna inte finns skapas en lista av (modellen)Bookings.

För att den nya bokningen ska ha ett id hämtas det senaste idt in från en hidden input i formuläret och till denna adderas 1.

Därefter läggs de efterfrågade värdena till. Värderna serialiseras och skrivs till jsonfilen. Därefter returneras man till index() och kan se den bokning man just gjort.

Skulle något här gå fel returneras man bara tillbaka till ”Boka bord”- sidan.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult BokaBord2(IFormCollection collection, Bookings model)

{
    if (ModelState.IsValid)
    {
        try
        {

            // Create ViewBag from input
            ViewBag.Email = collection["Email"];

            var JsonStr = System.IO.File.ReadAllText("bookings.json");
            var JsonObj = JsonConvert.DeserializeObject<IEnumerable<Bookings>>(JsonStr);

            return View(JsonObj);
        }
        catch
        {

            return RedirectToAction(nameof(Seebookings));
        }
    }
    else
    {
        // SET
        string iKnow = "Du måste fylla i en epostadress!";
        HttpContext.Session.SetString("bb", iKnow);

      // ViewBag.error = "Du måste fylla i en epostadress!";
        return RedirectToAction(nameof(Seebookings));
    }


}
```

För att hämta en bokning baserad på sin epost (som finns i Viewn ”Seebookings”) finns en form som pekar mot BokaBord2(IFormCollection collection), det är även här min validering ligger. Om valideringen går igenom sparas inputfältet för epost i en viewbag. Denna viewbag används som kontroll mot jsonfilen för att hämta enbart bokningar som matchar angiven epostadress. Går valideringen inte igenom returneras man till viewn ”Seebookings”. Här fick jag göra en fuling och la in min sessionsvariabel. Error meddelandet syntes inte när man returnerades på detta vis. Misstänker att det hade fungerat om funktionen hade samma namn som viewn (Seebookings)?

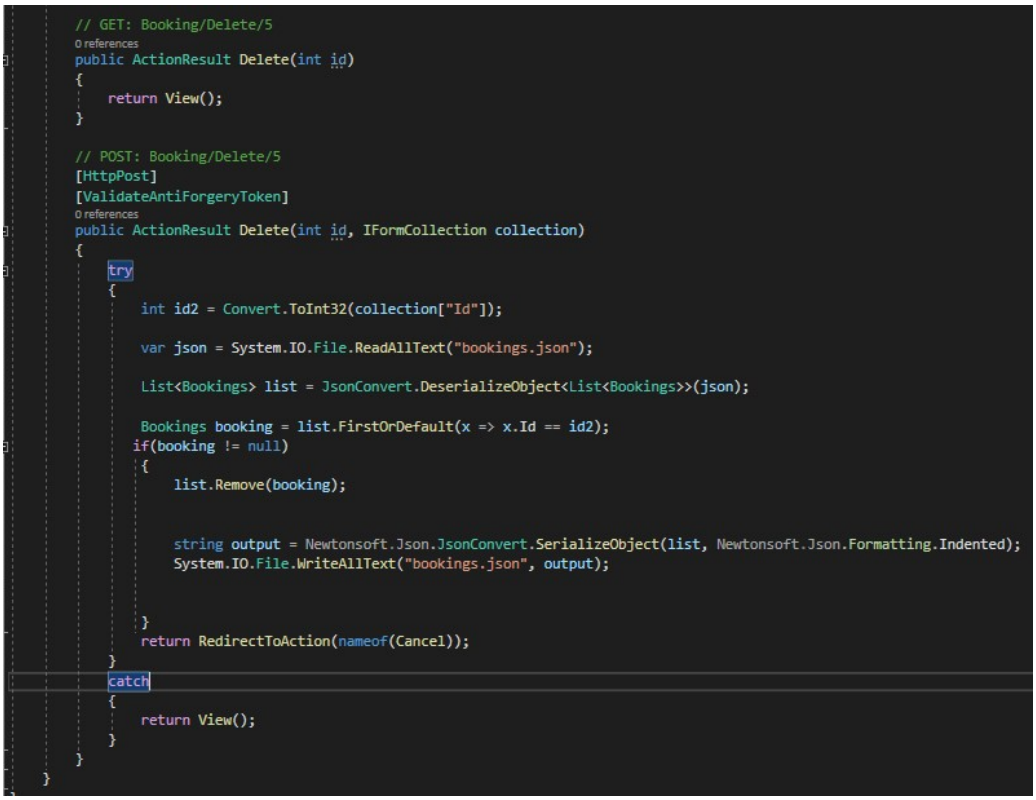
Resten av funktionerna i BookingController som returnerar views följer samma princip så istället för att ta upp alla views tar jag upp funktionerna för update och delete.

```
// POST: Booking/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Edit(int id, IFormCollection collection)
{


    try
    {

        // Get Id from hidden inputfield
        int id2 = Convert.ToInt32(collection["Id"]);
        // Convert date input to correct datatype (DateTime)
        string inDate = collection["Date"];
        DateTime outDate = Convert.ToDateTime(inDate);

        var json = System.IO.File.ReadAllText("bookings.json");

        List<Bookings> list = JsonConvert.DeserializeObject<List<Bookings>>(json);

        Bookings update = list.FirstOrDefault(x => x.Id == id2);

        if (update != null)
        {

            update.Table = Convert.ToInt32(collection["Table"]);
            update.Number = Convert.ToInt32(collection["Number"]);
            update.Date = outDate;
            update.Time = collection["Time"];

            string output = Newtonsoft.Json.JsonConvert.SerializeObject(list, Newtonsoft.Json.Formatting.Indented);
            System.IO.File.WriteAllText("bookings.json", output);

        }

        var JsonStr = System.IO.File.ReadAllText("bookings.json");
        var JsonObj = JsonConvert.DeserializeObject<IEnumerable<Bookings>>(JsonStr);

        return RedirectToAction(nameof(Rebook));
    }
    catch
    {
        var JsonStr = System.IO.File.ReadAllText("bookings.json");
        var JsonObj = JsonConvert.DeserializeObject<IEnumerable<Bookings>>(JsonStr);

        return View(JsonObj);

    }
}
```

När jag skapade funktionalitet för update fick jag en jäkla massa bekymmer, jag testade att försöka returnera både views och viewbags men inget hände, tog mig en bra stund innan jag fattade felet. Jag hade glömt asp-action för formuläret (det kommer nog inte hända igen ;)). För att skapa själva funktionen för update fick jag bra hjälp genom stackoverflow[3]. Första delen är väldigt lik hur man lägger till värden i en jsonfil, den största skillnaden är att man först letar reda på det id som stämmer överens med idt inputfältet, detta värde hämtade jag genom url:en och med javascript placerade jag det i en hidden input.

```
// Take id from url place in hidden input box
var url = window.location.href;
var lastPart = url.split("/").pop();

document.getElementById("id");
id.value = lastPart;
```

Resten är väldigt likt hur man lägger till. Går uppdateringen igenom skickas man vidare till viewn ”Rebook” som visar din ombokning. Annars kommer man bara returneras till aktuell view.



```
// GET: Booking/Delete/5
0 references
public ActionResult Delete(int id)
{
    return View();
}

// POST: Booking/Delete/5
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Delete(int id, IFormCollection collection)
{
    try
    {
        int id2 = Convert.ToInt32(collection["Id"]);

        var json = System.IO.File.ReadAllText("bookings.json");

        List<Bookings> list = JsonConvert.DeserializeObject<List<Bookings>>(json);

        Bookings booking = list.FirstOrDefault(x => x.Id == id2);
        if(booking != null)
        {
            list.Remove(booking);


            string output = Newtonsoft.Json.JsonConvert.SerializeObject(list, Newtonsoft.Json.Formatting.Indented);
            System.IO.File.WriteAllText("bookings.json", output);


        }
        return RedirectToAction(nameof(Cancel));
    }
    catch
    {
        return View();
    }
}
}
```

Delete påminner väldigt mycket om update (funktionen edit()). Tänkte att om ”Add” finns för min ”lägga till” borde motsvarande finnas för ”Remove” och det fungerade fint.

För att göra en patial följde jag en bra video på Youtube[4]. I katalogen ”Home” skapades filen ”_partialView”. Här la jag in koden för formuläret på kontaktsidan. I kontakt.cshtml skrevs sedan @Html.Partial(”_partialView”).

2 Slutsatser

Det var väldigt mycket nytt men det mesta var ganska självklart när man fått lite kläm på att. Tänket påminner ganska mycket om när vi kodade i php och hämtade in filer med ”include”, tänker framförallt på partials. Men det finns ju väldigt mycket kvar att lära här och det känns ändå som om jag förstår grunderna någorlunda. I och med webbplatsen jag valde blev allt ganska stort och ”projektigt”. Det tog en hel del tid att dela upp webbplatsen så den passade deluppgiften och med facit i hand borde jag nog valt något enklare, det är väldigt mycket javascript som styr webbplatsen (hur och när saker ska visas) och att gå igenom all js, ta bort och modifiera gav en hel del extrajobb.

3 Källförteckning

[1] Youtube, ”Skapa MVC-webbplats utifrån färdig HTML/CSS-grund”
https://www.youtube.com/watch?v=hbFp62cmj6k&feature=emb_logo
Publicerad 2020-01-26. Hämtad 2020-02-26

[2] Stackoverflow, ”how to Append a json file without disturbing the formatting”,
https://stackoverflow.com/questions/20626849/how-to-append-a-json-file-without-disturbing-the-formatting
Publicerad 2014. Hämtad 2020-02-27.

[3] Stackoverflow, ”How do I edit a json file in c#”,
https://stackoverflow.com/questions/36784849/how-do-i-edit-a-json-file-in-c
Publicerad 2017. Hämtad 2020-02-27.

[4] Youtube, ”Partial View in MVC”,
https://www.youtube.com/watch?v=MpK9vjGxSkg&t=
Publicerad 2019-08-21. Hämtad 2020-02-27.