Del2: WebService Frontend

I denna del kommer vi att skapa klienten för applikation med hjälp av HTML, javscript, jquery och bootstrap. Följande funktioner kommer att implementeras:

- Lista alla filmer
- Lägga till en ny film

Vi kommer att använda jQuery för att göra anrop till vårt API.

Det första vi gör är att lägga till en javascriptfil för vår applikation. Detta gör vi genom att högerklicka på mappen scripts, välj add och sedan JavaScript File. Döp denna till app!

Vi börjar med att göra en checker för att se så att den laddas korrekt på vår hemsida. I din app.js, skriv:

console.log(”testing”);

och spara filen.

Vi måste nu inkludera den i vårt projekt så att den laddas hem av browsern när vi går in på vår url.

Scrolla upp till mappen App_Start och öppna filen BundleConfig.cs. I denna fil vill vi skapa en ny bundle med vårt skript. Vi gör detta i RegisterBundles-funktionen och lägger till den under Jquerybundlen med följande kod.

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                "~/Scripts/jquery-{version}.js"));

    //App.js
    bundles.Add(new ScriptBundle("~/bundles/app").Include(
          "~/Scripts/app.js"));
```

Vi kan nu gå in i mappen Views, sedan shared och välja _Layout.cshtml. Här länkar vi nu in vår nya scriptbundle!

```
@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/app")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
```

Starta nu MVCapplikationen. När sidan har laddat klart, tryck på f12 på tangentbordet alternativt högerklicka på hemsidan, granska komponent och sedan gå till fliken console. Om texten ”testing” dyker upp fungerar allting, annars testa att ladda om och om det fortfarande inte fungerar se så att du har följt stegen rätt. Det är även viktigt att vi laddar in vår bundle efter jquery då vårt script kommer att använda sig av jquery.

Vi ska nu göra en superduperenkel förfrågan mot vårt api med hjälp av jquery och dess GET-metod. Vi kan använda jquerys ajaxanrop men för GET och POST finns det simplare funktioner.

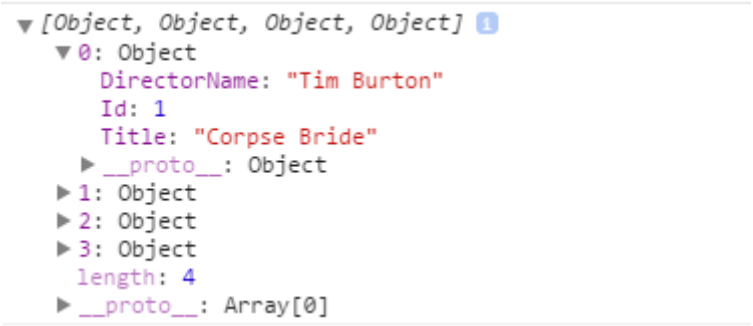
Dokumentation om jquery.get hittar du på http://api.jquery.com/jquery.get/

I din js-fil, ta bort din console.log och byt ut en mot:

```
$.get("/api/Movies", function (movies) {
    console.log(movies);
});
```

Det första argumentet är en URI till ditt API för att göra requests. Du hittar alla sådana under API-fliken på din egen hemsida. Den andra parametern är en anonym funktion som kommer att köras när svaret kommer tillbaka från servern med data.

Kolla i konsolen i din webbläsare och öppna svaret.

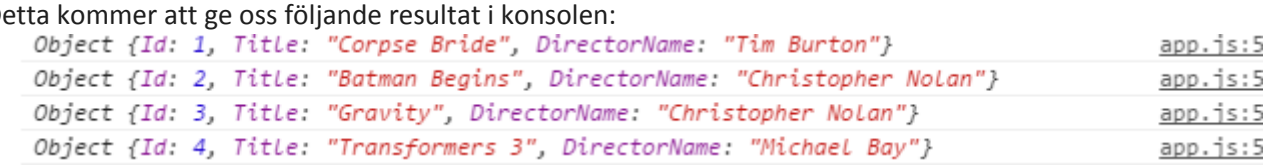


Här ser du det svar du får från servern med din GET-request. Detta är i JSONformat och svaret är ett Javascriptobject.
Vi kan loopa över denna med antingen jquerys .each eller en vanlig for-loop i ren javascript. Jag kommer att använda javascriptloopen eftersom jag anser att det är klarare vad som händer med den. Lägg nedanstående kod i den anonyma funktion som är ett callback till $.get vi skrev innan.

```
for (var movie in movies)
    {
        console.log(movies[movie]);
    }
```

Vi skapar en variabel movie som representerar en nyckel i objektet movies. För varje movienyckel vi hittar (som en foreach i c#) refererar vi dens plats i movies-objektet genom att peka på dess index som i en array.

Detta kommer att ge oss följande resultat i konsolen:



När vi skickar en GET-request till servern med /api/movies som argument gör vi alltså en förfrågan mot vårt API som sedan returnerar det MovieDTO vi skapade i gårdagens uppgift. Denna data kan vi nu använda för att berika vår sida med innehåll från databasen!

Vi kan bryta ner den förenklade get requesten till en ren ajax-request för att se hur den verkligen funkar. Vi kan kommentera ut den kod vi tidigare skrivit i app.js och definiera en funktion som heter GetData.

```
function GetData() {
}
```

I denna funktion använder vi jquerys ajaxmetod för att bygga en GET - request. Nedanför ser en enkel uppbyggnad för ett anrop med ajax och vilken information vi behöver. POST och PUT behöver även en data-nyckel med det objektet vi vill sätta in. Ett exempel på det kommer längre ner.

```
function GetData() {
    $.ajax({
        url: 'url för anropet',
        type: 'typ av request (GET/POST/PUT/DELETE)',
        success: function som körs om vår request fungerade
    })
}
```

För vår GET request vill vi hämta alla filmer vilket innebär att vi vill anropa APIt med URI api/Movies

```
function GetData() {
    $.ajax({
        url: 'api/Movies',
        type: 'typ av request (GET/POST/PUT/DELETE)',
        success: function som körs om vår request fungerade
    })
}
```

Eftersom vi ska HÄMTA alla filmer vill vi att vår type är GET.

```
function GetData() {
    $.ajax({
        url: 'api/Movies',
        type: 'GET',
        success: function som körs om vår request fungerade
    })
}
```

Och om det funkar vill vi köra en funktion. I exemplet endan har jag deklarerat en anonym funktion som kommer att vara ett callback om vårt api returnerar att operationen gick bra!

```
function GetData() {
    $.ajax({
        url: 'api/Movies',
        type: 'GET',
        success: function (data) {
            console.log(data);
        }
    })
}
```

Kallar vi nu på funktionen kommer vi att hämta data från vårt API! Anledningen till att jag går igenom denna och inte nöjer mig med den lättare jquery-varianten $.get är för att PUT och DELETE inte finns i jQuery på samma sätt utan måste ske genom ett regelrätt ajaxanrop.

Vi kan nu göra samma sak, fast för att lägga in data i databasen genom vårt API. Vi anropar samma URI men ändrar type till POST. Om vi tittar i vår postmetod i MoviesController.cs så ser vi att den tar emot ett MovieObjekt. Detta betyder att vi måste lägga till en nyckel som håller ett sådant objekt, vi döper den

```
function AddData() {
    $.ajax({
        url: 'api/Movies',
        type: 'POST',
        data: {
            Vår data ska ligga här
        },
        success: function(data){console.log(data)}
        });

};
```

När vi nu matar in data i detta objekt är det viktigt att den är korrekt, annars kommer vi att få felkoder från servern. Det som ska motsvara ett Movieobjekt vilket kan se ut såhär:

```
function AddData() {
    $.ajax({
        url: 'api/Movies',
        type: 'POST',
        data: {
            Title: "Harry Potter",
            Price: 300,
            Year: 2006,
            Genre: "Fantasy",
            DirectorId: 1
        },
        success: function(data){console.log(data)}
        });

};
```

Alla nycklar och värden motsvarar ett vanligt MovieObjekt.

Testa att kalla på AddData() och sedan GetData() för att se så att vi fått en ny post i databasen. Att ändra data data med PUT och ta bort data med DELETE har samma process. Kolla i era controllers och tänk ut hur du kan skapa dessa funktioner!

För att avluta denna tutorial ska vi
1: Använda den data vi har hämtat
2: Lägga in data från fält

Det första exemplet kommer att vara i ren javascript och den andra gör vi med jQuery.

Vi kan skapa en funktion FillJumbotron som tar emot ett argument som heter data. Denna funktion ska kallas på i den anonyma funktion som skickas när ett GET request går igenom och kör den funktion som ligger i ”success”-nyckeln.

```
function GetData() {
    $.ajax({
        url: 'api/Movies',
        type: 'GET',
        success: function (data) {
            FillJumbotron(data);
        }
    })
}
```

Vår FillJumbotron-funktion tar emot det data som returneras och plockar ut det element som har jumbotronklassen (det stora gråa i toppen av sidan, ligger där som standard i mvc på index.cshtml. Vi loopar sedan över datan och för varje film sparar vi ner den i en variabel movie som vi sedan loopar på för att få ut alla attribut. Vi använder den variabel där vi sparat ner vårt element med jumbotronklassen och säger att dess HTML ska utöka med varje nytt värde vi stöter på + en
 tagg för att bryta raden. Det finns många snyggare sätt att göra detta på men detta är för den grundligaste funktionen!

```
function FillJumbotron(data)
{
    var jumbotron = document.querySelector(".jumbotron");

    for(var object in data)
    {
        var movie = data[object];

        for(var key in movie)
        {
            jumbotron.innerHTML += movie[key] + "<br>";

        }
    }
}
```

För att lägga in data måste vi ha någonstans där användaren kan skriva in den. I detta exempel gör vi en enkel lösning där vi strukturerar upp inputfält i en tabell tillsammans med rubriker för dessa. Vi kommer med hjälp av jQuery att plocka ut dess värden och skicka det med POST till apit. Tabellen ser du nedanför, observera att det ligger en knappt i slutet. Denna kommer trigga AddData-funktionen när den klickas på.

```
<table>
    <tr>
        <td>
            <label for="title">Titel</label>
        </td>
        <td>
            <input type="text" id="title">
        </td>
    </tr>
    <tr>
        <td>
            <label for="year">Year</label>
        </td>
        <td>
            <input type="text" id="year">
        </td>
    </tr>
    <tr>
        <td>
            <label for="price">Price</label>
        </td>
        <td>
            <input type="text" id="price">
        </td>
    </tr>
    <tr>
        <td>
            <label for="genre">Genre</label>
        </td>
        <td>
            <input type="text" id="genre">
        </td>
    </tr>
    <tr>
        <td>
            <label for=" directorId">Director ID</label>
        </td>
        <td>
            <input type="text" id="directorId">
        </td>
    </tr>

</table>
<button id="submit">Add movie</button>
```

Istället för att vi har fördefinierade värden i vår AddData-funktion kan vi nu istället enkelt med hjälp av jQuery välja fälten och sedan plocka ut dess värden med .val() funktionen. Detta är så klart inte idiotsäkert och för bästa användarupplevelse för man titta så att dessa faktiskt innehåller något.

```
function AddData() {
    $.ajax({
        url: 'api/Movies',
        type: 'POST',
        data: {
            Title: $('#title').val(),
            Price: $('#price').val(),
            Year: $('#year').val(),
            Genre: $('#genre').val(),
            DirectorId: $('#directorId').val()
        },
        success: function(data){console.log(data)}
        });

};
```

Sista men inte minst vill vi binda vår knapp så den kör AddData-funktionen när vi klickar på den.

```
$('#submit').on('click', AddData);
```

Du har nu ett dokument med funktioner för GET och POST. Du kan snygga till detta utifrån din egen önskan och sedan påbörja uppgiften för veckan!