Lab 4 – The Entity Framework

# Description

This lab allows you to start putting all of the pieces together to create coherent web applications using ASP.NET MVC Core and the Entity Framework.

# Estimated Time

This lab will take an estimated 8 hours to complete

# Deliverable

Stand by for further instructions.

# Notes

* Be sure Visual Studio is up to date.
* Install SQL Server 2016 Community Edition
* Install SQL Server Management Studio Community Edition
* Follow along closely to the instructions!
* When using the <input> tag be sure to set the type to collect the proper values
* Use the example code ‘ASP.NET MVC and the Entity Framework’ for help and tips.
* Demo can be found at: <http://cst8359lab4.azurewebsites.net/>

# Install SQL Server and the SQL Server Management Studio

1. SQL Server Community Edition can be located at
   1. <https://www.microsoft.com/en-ca/cloud-platform/sql-server-editions-express>
   2. Download and install the application. You can use default settings for the most part.
   3. NOTE: once you set a password be sure you remember it. Also, add your current user as an administrator so Visual Studio can impersonate you for easy access.
2. SQL Server Management Studio can be located at:
   1. <https://msdn.microsoft.com/en-us/library/mt238290.aspx>
   2. Download and install the application. Default settings should also be acceptable
   3. NOTE: Once the server is installed you should either be able to log into your database using this tool and create the database and required table.
3. The script to create the database and its table can be located at the Lab 4 demo site.
4. ***I will be demoing this process in class during Week 6. Please ask questions otherwise.***

# Create a new MVC Core project called ‘Lab4’

1. Open Visual Studio 2015
2. Click: File -> New -> Project
3. Click: Templates -> Visual C# -> Web, select ‘ASP.NET Core Web Application (.NET Core)’
4. Name the application ‘Lab4’, save the project in your desired location and click the button ‘OK’
5. Select the ‘Empty’ ASP.NET Core Template
6. Be sure to uncheck ‘Host in the cloud’
7. Click the ‘OK’ button.

# Configure your new Web Application

1. Modify the file ‘project.json’. Replace the section:

"dependencies": {

"Microsoft.NETCore.App": {

"version": "1.0.1",

"type": "platform"

},

"Microsoft.AspNetCore.Diagnostics": "1.0.0",

"Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",

"Microsoft.AspNetCore.Server.Kestrel": "1.0.1",

"Microsoft.Extensions.Logging.Console": "1.0.0"

},

with:

"dependencies": {

"Microsoft.NETCore.App": {

"version": "1.0.1",

"type": "platform"

},

"Microsoft.AspNetCore.Diagnostics": "1.0.0",

"Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",

"Microsoft.AspNetCore.Server.Kestrel": "1.0.1",

"Microsoft.AspNetCore.Mvc": "1.0.1",

"Microsoft.AspNetCore.Session": "1.0.0",

"Microsoft.AspNetCore.StaticFiles": "1.0.0",

"Microsoft.Extensions.Logging.Console": "1.0.0",

"Microsoft.Extensions.Caching.Memory": "1.0.0",

"Microsoft.EntityFrameworkCore.SqlServer": "1.0.1" },

1. Visual Studio should now update itself with the packages you need for this application to run.
2. Modify ‘Startup.cs’. Add the following lines to the method ‘ConfigureServices(IServiceCollection services)’”

var connection = @"Server=localhost;Database=Lab4;Trusted\_Connection=True;MultipleActiveResultSets=true";

// if that fails try: var connection =

// @"Server=localhost\SQLEXPRESS;Database=Lab4;Trusted\_Connection=True;MultipleActiveResultSets=true";

services.AddDbContext<MovieContext>(options => options.UseSqlServer(connection));

services.AddMvc();

services.AddMemoryCache();

services.AddSession();

1. Modify ‘Startup.cs’. Replace the contents of ‘Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)’ with:

loggerFactory.AddConsole();

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles();

app.UseSession();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

# Create the ‘Controllers’ and ‘Views’ and ‘Models’ folders

1. Create a folder in your project called ‘Controllers’
2. Create a new Controller in a folder called ‘Home’
   1. NOTE: if you use ‘MVC Controller Class’ template from visual studio it will create all the basic code you need to have a valid controller
   2. To do so right click on the ‘Controllers’ folder, click ‘Add’, click ‘New Item’ and select the ‘MVC Controller Class’
3. Create a folder in your project called ‘Views’
4. At the root of this new ‘Views’ folder create a filed called ‘\_ViewImports.cshtml’
5. Add the following lines of code to the file ‘\_ViewImports.cshtml’

@using Lab4

@addTagHelper "\*, Microsoft.AspNetCore.Mvc.TagHelpers"

1. At the root of the ‘Views’ folder created a folder called ‘Home’
2. At the root of the project create a folder called ‘Models’

# Create the Model

1. At the root of the ‘Models’ folder create a file called ‘MoviesContext.cs’
   1. Use the example code located in ‘ASP.NET MVC Core and the Entity Framework’ zip file to create your data context.
   2. Make sure your context you include the Constructor and the DBSet to hold your movie objects.
2. At the root of the ‘Models’ folder create a file called ‘Movie.cs’
3. Add the following Properties to the ‘Movie.cs’ file
   1. int MovieId
   2. string Title
      1. Add the ‘Required’ attribute
      2. Add the ‘StringValue’ attribute, with a value of 1000
   3. string SubTitle
      1. Add the ‘Required’ attribute
      2. Add the ‘StringValue’ attribute, with a value of 1000
   4. string Description
      1. Add the ‘Required’ attribute
      2. Add the ‘StringValue’ attribute, with a value of 1000
   5. DateTime Year
      1. Add the ‘Required’ attribute
      2. Add the ‘DataType’ attribute, with the value DataType.Data
   6. int Rating
      1. Add the ‘Required’ attribute
      2. Add the ‘Range’ attribute, with the values 1 and 5

# Create the Controller’s Actions and Views

1. Copy the code for your database access and constructor from the ‘HomeController.cs’ file in example ‘ASP.NET MVC Core and the Entity Framework’ example code.
2. In the ‘Home’ Controller create an Action and corresponding View named ‘Index’
   1. This action will display the list of movies in your database
   2. The first column of the list of movies will have two options
      1. Edit
      2. Delete
   3. This action will display a message saying ‘There are no movies in your database’ or words to that affect, if no words exist.
3. In the ‘Home’ Controller create an Action and corresponding View named ‘AddMovie’
   1. This action will display a form that will collect a ‘Strongly Typed’ model of type ‘Movie’. Create all necessary form elements to collect this data.
   2. This form with ‘POST’ itself to a ‘CreateMovie’ Action of the ‘Home’ Controller
4. In the ‘Home’ Controller create an Action named ‘CreateMovie’
   1. This action accepts a Model of type Movie as a parameter.
   2. This action verifies that the Model’s data is valid.
   3. This action saves the data to the database.
   4. This action then returns not ‘View()’ but ‘RedirectToView(“Index”)’
5. In the ‘Home’ Controller create an Action and corresponding View named ‘EditMovie’
   1. This Action is called when the user clicks the ‘Edit’ button on the ‘Index’ view.
   2. This action will accept a parameter named ‘id’. This ‘id’ value will be used to query your database and get the appropriate ‘Movie’ object from the database.
   3. The ‘Movie’ object will be passed a parameter to the view.
   4. The View will contain a form necessary to modify the ‘Movie’ object.
   5. This form will ‘POST” itself to a ‘ModifyMovie’ Action of the ‘Home’ Controller
6. In the ‘Home’ Controller create an Action named ‘ModifyMovie’
   1. This action accepts a Model of the type Movie as a parameter.
   2. This action verifies that the Model’s data is valid.
   3. This action saves the modified Movie data to the database.
   4. This action then returns not ‘View()’ but ‘RedirectToView(“Index”)’
7. In the ‘Home’ Controller create an Action named ‘DeleteMovie’
   1. This Action is called when the user clicks the ‘Delete’ button on the ‘Index’ view.
   2. This action will accept a parameter named ‘id’. This ‘id’ value will be used to query your database and get the appropriate ‘Movie’ object from the database.
   3. This ‘Movie’ object will deleted the object from the database.
   4. This action then returns not ‘View()’ but ‘RedirectToView(“Index”)’

# Lastly

1. Be prepared to completely reuse (rip off) the code example provided in ‘ASP.NET MVC Core and the Entity Framework’