Lab 4 – The Entity Framework

# Description

This lab allows you to start putting all 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 7 hours to complete

# Deliverable

Deploy your website to Windows Azure and push your code to GitHub. Submit both links to Brightspace.

See “Brightspace -> Course Content -> Extra Materials -> Azure Usage” for information about deploying Azure Web Apps, Databases and Storage Accounts.

# Notes

* Be sure Visual Studio is up to date.
* 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.
* By properties, we always mean public properties
* Demo can be found at: <http://afrasialab4.azurewebsites.net/>
* Sample code can be found at: <https://github.com/aarad-ac/EFCore>

# Create a new MVC Core project called ‘Lab4’

1. Open Visual Studio 2019
2. Create new ASP.NET Core Empty application called Lab4.
   1. Uncheck the HTTPS box
   2. Choose .NET 5.0 as your target framework
3. Once created, right click on your project in solution explorer
4. From the menu, select Manage NuGet Packages
5. Click on browse tab
6. In the search box, type: Microsoft.EntityFrameworkCore.SqlServer
7. Hit enter
8. Click on Microsoft.EntityFrameworkCore.SqlServer from the list, make sure the selected version is v5.0.6
9. On the right panel, hit Install, click ok after and accept the license information
10. Repeat steps 3-9 for these packages too:
    1. “Microsoft.EntityFrameworkCore" -> v5.0.6
    2. "Microsoft.EntityFrameworkCore.Sqlite” -> v5.0.6
    3. "Microsoft.EntityFrameworkCore.Tools" -> v5.0.6
    4. “Microsoft.VisualStudio.Web.CodeGeneration.Design” -> v5.0.2

# Configure your new Web Application

1. At the root of the project create a folder called ‘Data’
2. Under Data folder, create a class ‘DbInitializer.cs’:
   1. Replace the class with the code below and ignore the syntax errors for now:

public static class DbInitializer

{

public static void Initialize(SchoolCommunityContext context)

{

context.Database.EnsureCreated();

}

}

1. Replace body of Main in ‘Program.cs’ with the code below, ignore the syntax errors around ` SchoolCommunityContext` but fix the rest of them by adding the right using statements (you are on your own):

var host = CreateHostBuilder(args).Build();

using (var scope = host.Services.CreateScope())

{

var services = scope.ServiceProvider;

try

{

var context = services.GetRequiredService<SchoolCommunityContext >();

DbInitializer.Initialize(context);

}

catch (Exception ex)

{

var logger = services.GetRequiredService<ILogger<Program>>();

logger.LogError(ex, "An error occurred while seeding the database.");

}

}

host.Run();

1. See the Azure SQL document in Brightspace -> extra materials to find your database connection string
2. Modify both appsettings.Development.json and appsettings.json and add the following lines right before “Logging”, replacing \*\*\*\*Enter your connection string here\*\*\* with your connection string. Do not remove the quotes.

"ConnectionStrings": {

"DefaultConnection": "\*\*\*\*Enter your connection string here\*\*\*"

},

1. Modify ‘Startup.cs’. Add the following constructor to the class, and fix the issues by adding the right using statements (you are on your own):

public IConfiguration Configuration { get; }

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

1. Modify ‘Startup.cs’. Add the following lines to the method ‘ConfigureServices(IServiceCollection services)’”. Ignore errors around ` SchoolCommunityContext` and fix the rest by adding the right using statements (you are on your own):

var connection = Configuration.GetConnectionString("DefaultConnection");

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

services.AddControllersWithViews();

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

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

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

});

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

1. Create a folder in your project called ‘Controllers’
2. Under ‘Controllers’, create a new empty MVC Controller called ‘Home’. Note that the class name is HomeController.cs
3. Create a folder in your project called ‘Views’
4. At the root of this new ‘Views’ folder, create a Razor View Imports file called ‘\_ViewImports.cshtml’
5. Add the following lines of code to the file ‘\_ViewImports.cshtml’

@using Lab4

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

1. Under views, create your default layout in the shared folder
2. At the root of the ‘Views’ folder created a folder called ‘Home’
3. At the root of the project create a folder called ‘Models’
4. At the root create wwwroot folder and add a css file there. You can use the one I posted with the sample code
5. Create a view called Error.cshtml under Views/Shared. It should contain some error message like this example:

@{

ViewData["Title"] = "Error";

}

<h1 class="text-danger">Error.</h1>

<h2 class="text-danger">An error occurred while processing your request.</h2>

# Create the Model

1. Under the ‘Models’ folder, create a new calss called ‘Student.cs’
   1. Add the following Properties to the ‘Student.cs’ file
      1. int Id
      2. string LastName
         1. Add the ‘Required’ attribute
         2. Add the ‘StringLength’ attribute, with a value of 50
         3. Set the display name to ‘Last Name’
      3. string FirstName
         1. Add appropriate attributes like LastName’s
         2. Display name should be ‘Last Name’
      4. DateTime EnrollmentDate
         1. Set the attributes:
            1. [DataType(DataType.Date)]
            2. [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
            3. [Display(Name = "Enrollment Date")]
      5. string FullName
         1. this is a calculated field from: LastName + “, “ + FirstName
2. At the root of the ‘Models’ folder, create a file called ‘Community.cs’
   1. Add the following Properties to the ‘Community.cs’ file
      1. string Id
         1. make sure this is not database generated by setting the DatabaseGenerated attribute to DatabaseGeneratedOption.None
         2. Display name should read ‘Registration Number’
         3. Add the ‘Required’ attribute
      2. string Title
         1. Add the ‘Required’ attribute
         2. Add the ‘StringLength’ attribute, with a value of max 50 and MinimumLength = 3
      3. decimal Budget
         1. Add attributes to define data type currency and column type money
            1. [DataType(DataType.Currency)]
            2. [Column(TypeName = "money")]
3. At the root of the ‘Models’ folder, create a file called ‘CommunityMembership.cs’
   1. Add the following Properties to the ‘CommunityMembership.cs’ file
      1. int StudentId
      2. string CommunityId
4. Make sure you understand the following Entity Relation Diagram before continuing with the rest of the lab.

N-1

1-N

CommunityMembership

Community

Student

1. Refer to the lecture, and add the proper navigational properties to CommunityMembership, Community and Student models
   1. For Navigational properties you define in Community and Student models, name both Membership, and make sure that they navigate to CommunityMembership
2. At the root of the ‘Data’ folder, create a file called ‘SchoolCommunityContext.cs’
   1. Use the example code posted in the lectures to create your data context.
   2. Make sure in your context you include the Constructor and the DBSet to hold your entity objects. The DBSet variables should have a plural name of entities. Like for Student model:
      1. public DbSet<Student> Students { get; set; }
   3. Make sure the table names are not plural. You can do that by setting the proper names in OnModelCreating method. Like, for the Student model:
      1. modelBuilder.Entity<Student>().ToTable("Student");
   4. CommunityMembership should have a composite key of StudentId and CommunityId:

modelBuilder.Entity<CommunityMembership>()

.HasKey(c => new { c.StudentId, c.CommunityId });

1. Once you reach this step, all your previous Syntax Errors that you ignored should be fixed. If you still have syntax errors, you need to debug your code.

# Initialize DB

1. The code in DbInitializer.cs makes sure that your DB exists. You need to fill it with data. At least 3 students. At least 3 Communities, and at least there should exist one student who is a member of two or more communities. You can do this through Azure SQL connection, or by adding the following code to ‘Initialize’ method in ‘DbInitializer.cs’ after context.Database.EnsureCreated();:

if (context.Students.Any())

{

return; // DB has been seeded

}

var students = new Student[]

{

new Student{FirstName="Carson",LastName="Alexander",EnrollmentDate=DateTime.Parse("2005-09-01")},

new Student{FirstName="Meredith",LastName="Alonso",EnrollmentDate=DateTime.Parse("2002-09-01")},

new Student{FirstName="Arturo",LastName="Anand",EnrollmentDate=DateTime.Parse("2003-09-01")},

new Student{FirstName="Gytis",LastName="Barzdukas",EnrollmentDate=DateTime.Parse("2002-09-01")},

};

foreach (Student s in students)

{

context.Students.Add(s);

}

context.SaveChanges();

var communities = new Community[]

{

new Community{Id="A1",Title="Alpha",Budget=300},

new Community{Id="B1",Title="Beta",Budget=130},

new Community{Id="O1",Title="Omega",Budget=390},

};

foreach (Community c in communities)

{

context.Communities.Add(c);

}

context.SaveChanges();

var memberships = new CommunityMembership[]

{

new CommunityMembership{StudentId=1,CommunityId="A1"},

new CommunityMembership{StudentId=1,CommunityId="B1"},

new CommunityMembership{StudentId=1,CommunityId="O1"},

new CommunityMembership{StudentId=2,CommunityId="A1"},

new CommunityMembership{StudentId=2,CommunityId="B1"},

new CommunityMembership{StudentId=3,CommunityId="A1"},

};

foreach (var m in memberships)

{

context.CommunityMemberships.Add(m);

}

# context.SaveChanges();

# Create the Controller’s Actions and Views

1. Important Note: The following steps, scaffold controllers and view for you. Make sure your models are final, and you will not make any changes to them after the following steps. If you change models after scaffolding, you either need to replicate the changes manually in scaffolded classes or delete all scaffolded classes and redo the following steps. It goes without saying that you need to change the scaffolded classes as you see appropriate. The scaffolding helps you to reduce the amount of manual coding, but you should still be in control and make necessary changes after the scaffolding is done.
2. Right click on ‘Controllers’
3. From the menu, under ‘Add’, click on ‘New Scaffolded Item’
4. From the options, select ‘MVC Controller with views, using Entity Framework’, and click on ‘Add’
5. For model, chose ‘Community’, and for Data context choose ‘SchoolCommunityContext’
6. Make sure all the checkboxes are selected.
7. Click ‘Add’
8. Inspect newly created community views, and CommunityController
9. Repeat steps 1-7 for Student Model
10. In the ‘Home’ Controller create an Action and corresponding View named ‘Index’
    1. This will be your welcome screen.
    2. Introduce yourself
    3. If you have a feedback or comment about the assignments or course, put here (optional)
11. In the ‘Home’ controller, create an action Error:

public IActionResult Error()

{

return View(); // Do you need to modify this line? Use your judgement based on the app you developed so far

}

1. Open your \_Layout, and create action links to
   1. Home controller and Index action
   2. Student controller and Index action
   3. Community controller and Index action
2. Create a folder called ‘ViewModels’ under Models
   1. Under it, create a ViewModel called ‘CommunityViewModel.cs’
   2. Refer to the sample code posted. This should have IEnumerable fields referencing Communities, Students, and CommunityMemberships
      1. Example: public IEnumerable<Student> Students { get; set; }
3. Open Views/Communities/Index.cshtml
   1. Replace the model declaration line with: @model Lab4.Models.ViewModels.CommunityViewModel
   2. Modify the class in a way that once you select the communities, it shows a list of memberships

# Important Note:

1. Do not underestimate this lab. There are many moving pieces and if you miss one piece you might spend hours fixing it. Start early and ask questions if you are stuck.
2. It goes without saying that you need to modify the scaffolded classes to add some of the functionality you need. Scaffolding just creates the structure (stubs) for you and you nee to fill in the gaps.
3. If you change the models after scaffolding is done, you might want to delete all the scaffolded classes and scaffold again. Of course, this should be used as the last resort. Try to learn and modify the code yourself.
4. If you change your models after your first run of the program, you will need to reflect the changes in the database manually, delete the database or simply change the database name in the appsettings so a new database gets created based on the model changes. We will avoid this later by effectively using migrations.
5. Your assignment 1 will build on top of this lab. Therefore, it is very important that this lab is done right.
6. To have an easier time with debugging your application, I encourage that you test your application with a local database (see the sample code for connection string), and once you are satisfied with your app, change the connection to Azure SQL and test it again by running the app locally. Again, once you are happy with the outcome, follow the steps for publishing and publish your app.
7. During the publish process, you need to configure your SQL server
   1. On the publish tab, and under ‘Service Dependencies’ section, find the ‘SQL Server Database, and click ‘Configure’ in front of it.
   2. On the next page, select ‘Azure SQL Database’
   3. Click next on the following page, too
   4. On the ‘Configure Azure SQL Database’,
      1. enter your Azure SQL username and password
      2. On ‘Save Connection String in’, select ‘None’
      3. click next
   5. Uncheck both NuGet Packages and Secrets store
   6. Click finish
8. Click close, and you are back to Publish tab
9. Click MoreActions -> Edit
10. On the new window, click on ‘settings’ tab
11. Under ‘Database’ and ‘Entity Framework Migrations’, uncheck all checkboxes
12. Click save
13. Click publish button
14. Note that if after publishing your app cannot reach the database, follow these steps:
    1. Go to Azure Portal
    2. Find your Azure SQL instance
    3. On the Overview tab, click on ‘Set Server Firewall’
    4. On the Firewall Settings page, change ‘Allow Azure services and resources to access this server’ from ‘No’ to ‘Yes’
    5. Click ‘save’

# Push your code to GitHub

1. Open command prompt, and navigate to your lab4 solution directory
2. If you have not already done so, download and install git: <https://gitforwindows.org/>
   1. Go with all defaults
3. Follow these steps to push your code to GitHub: <https://docs.github.com/en/free-pro-team@latest/github/importing-your-projects-to-github/adding-an-existing-project-to-github-using-the-command-line>
4. Note: No PASSWORDs should be pushed to Git. You must add both your appsettings files to gitignore, so they never get pushed to git.
   1. Just create a file called’.gitignore’
   2. And then add the following lines to it:

appsettings.json

appsettings.Development.json

# Grading Scheme

|  |  |
| --- | --- |
| task | mark |
| program.cs | 2 |
| Startup.cs | 2 |
| folder structures | 1 |
| Error page | 1 |
| css | 1 |
| student.cs | 5 |
| Community.cs | 4 |
| CommunityMembership | 2 |
| dbcontext | 3 |
| viewmodel | 3 |
| layout | 1 |
| communities controller | 4 |
| student controller | 1 |
| homecontroller | 1 |
| dbinitializer | 1 |
| student views | 2 |
| community views | 2 |
| communityIndex | 4 |
| gitignore | 3 |
| publishing | 2 |
| working end-to-end solution | 5 |
|  |  |
|  |  |
| Total | 50 |