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
- 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/EntityFramework

Create a new MVC Core project called 'Lab4'

1. Open Visual Studio 2022
2. Create new ASP.NET Core Empty application called Lab4.
    a. Uncheck the HTTPS box
    b. Choose .NET 6.0 as your target framework
    c. Check Do not use top-level statements
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 v6.0.10 (or newer)
9. On the right panel, hit Install, click ok after and accept the license information
10. Repeat steps 3-9 for these packages too:
    a. “Microsoft.EntityFrameworkCore" -> v6.0.10 (or newer)
    b. "Microsoft.EntityFrameworkCore.Sqlite” -> v6.0.10 (or newer)
    c. "Microsoft.EntityFrameworkCore.Tools" -> v6.0.10 (or newer)
    d. “Microsoft.VisualStudio.Web.CodeGeneration.Design” -> v6.0.10 (or newer)

Configure your new Web Application

1. Open your project file, and comment out (or delete) the line: <Nullable>enable</Nullable>
2. At the root of the project create a folder called ‘Data’
3. Under Data folder, create a class ‘DbInitializer.cs’:
    a. Replace the class with the code below and ignore the syntax errors around `MarketDbContext` for now:

```
public static class DbInitializer
 {
     public static void Initialize(MarketDbContext context)
     {
         context.Database.EnsureCreated();

    }

}
```

4. Clear the content of the Program.cs file and replace it with the code at: https://gist.github.com/aarad-ac/96ca0fdd99497c80d701309f597193da
5. Then replace /*LabName*/ with Lab4, and replace /*DBContext*/ with `MarketDb`Context
6. Ignore the syntax errors around `MarketDbContext` `and` but fix the rest of them by adding the right ‘using’ statements (you are on your own)
7. See the Azure SQL document in Brightspace -> extra materials to find your database connection string
8. 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***"
    },
```

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
```

6. At the root of this new ‘Views’ folder, create a Razor View Start file called ‘_ViewStart.cshtml’
7. Under views, create your default layout in the Shared folder (you are on your own)
8. At the root of the ‘Views’ folder created a folder called ‘Home’
9. At the root of the project create a folder called ‘Models’
10. At the root of the project create wwwroot folder and add a css file there. You can use the one I posted with the sample code
11. Create a view called Error.cshtml under Views/Shared. It should contain some error message like this example:

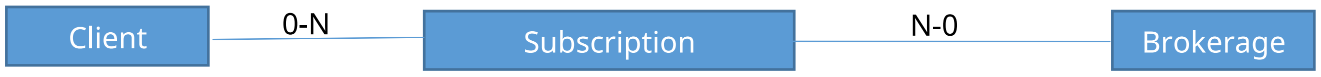
```
@{
    ViewData["Title"] = "Error";
}

<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 class called ‘Client.cs’
    a. Add the following Properties to the ‘Client.cs’ file
        i. int Id
        ii. 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’

iii. string FirstName
   1. Add appropriate attributes like LastName's
   2. Display name should be First Name'
iv. DateTime `BirthDate`
   1. Set the attributes:
      a. `[DataType(DataType.Date)]`
      b. `[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]`
      c. `[Display(Name = "Birth Date")]`
v. string FullName
   1. this is a calculated field from: LastName + ", " + FirstName

2. At the root of the 'Models' folder, create a file called 'Broker.cs'
   a. Add the following Properties to the 'Brokerage.cs' file
      i. 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
      ii. string Title
         1. Add the 'Required' attribute
         2. Add the 'StringLength' attribute, with a value of max 50 and MinimumLength = 3
      iii. decimal Fee
         1. Add attributes to define data type currency and column type money
            a. `[DataType(DataType.Currency)]`
            b. `[Column(TypeName = "money")]`
3. At the root of the 'Models' folder, create a file called 'Subscription.cs'
   a. Add the following Properties to the 'Subscription.cs' file
      i. int ClientId
      ii. string BrokerageId
4. Make sure you understand the following Entity Relation Diagram before continuing with the rest of the lab.



5. Refer to the lecture, and add the proper navigational properties to Subscription, Client and Brokerage models

   a. For Navigational properties you define in Client and Brokerage models, name both Subscriptions, and make sure that they navigate to Subscription model
6. At the root of the 'Data' folder, create a file called 'MarketDbContext.cs'
   a. Use lecture slides to create your data context. Remember, you need the constructor, and pay attention to inheritance.
   b. Make sure in your context you include the Constructor and the DBSet(s) to hold your entity objects. The DBSet variables should have a plural name of entities. Like for Client model:
      i. `public DbSet<Client> Clients { get; set; }`
   c. Make sure the table names are not plural. You can do that by setting the proper names in OnModelCreating method. Like, for the Client model:
      i. `modelBuilder.Entity<Client>().ToTable("Client");`
   d. Subscription should have a composite key of ClientId and BrokerageId, and we need to set that using Fluent API:

```
modelBuilder.Entity<Subscription>()
                  .HasKey(c => new { c.ClientId, c.BrokerageId });
```

7. 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 clients. At least 3 brokerages, and at least there should exist one client who is subscribed to two or more brokerages. 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.Clients.Any())
{
   return;   // DB has been seeded
}

var clients = new Client[]
{
new Client{FirstName="Carson",LastName="Alexander",BirthDate=DateTime.Parse("1995-01-09")},
new Client{FirstName="Meredith",LastName="Alonso",BirthDate=DateTime.Parse("1992-09-05")},
new Client{FirstName="Arturo",LastName="Anand",BirthDate=DateTime.Parse("1993-10-09")},
new Client{FirstName="Gytis",LastName="Barzdukas",BirthDate=DateTime.Parse("1992-01-09")},
};
foreach (Client s in clients)
{
   context.Clients.Add(s);
}
context.SaveChanges();

var brokerages = new Brokerage[]
{
new Brokerage{Id="A1",Title="Alpha",Fee=300},
new Brokerage{Id="B1",Title="Beta",Fee=130},
new Brokerage{Id="O1",Title="Omega",Fee=390},
};
foreach (Brokerage c in brokerages)
{
   context.Brokerages.Add(c);
}
context.SaveChanges();

var subscriptions = new Subscription[]
{
new Subscription{ClientId=1,BrokerageId="A1"},
new Subscription{ClientId=1,BrokerageId="B1"},
new Subscription{ClientId=1,BrokerageId="O1"},
new Subscription{ClientId=2,BrokerageId="A1"},
new Subscription{ClientId=2,BrokerageId="B1"},
new Subscription{ClientId=3,BrokerageId="A1"},
};
foreach (var subscription in subscriptions)
{
   context.Subscriptions.Add(subscription);
}
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 ‘Brokerage’, and for Data context choose ‘MarketDbContext’
6. Make sure all the checkboxes are selected.
7. Click ‘Add’
8. Inspect newly created Brokerage views, and BrokeragesController
9. Repeat steps 1-7 for Client Model
10. In the ‘Home’ Controller create an Action and corresponding View named ‘Index’
    a. This will be your welcome screen.
    b. Introduce yourself
    c. 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
 }
```

12. Open your _Layout, and create action links to
    a. Home controller and Index action
    b. Client controller and Index action
    c. Brokerage controller and Index action
13. Create a folder called ‘ViewModels’ under Models
    a. Under it, create a ViewModel called ‘BrokerageViewModel.cs’
    b. Refer to the sample code posted. This should have IEnumerable fields referencing Brokerages, Clients, and Subscriptions
        i. Example: `public IEnumerable<Client> Clients { get; set; }`
14. `Open Views/Brokerages/Index.cshtml`
    a. `Replace the model declaration line with: @model Lab4.Models.ViewModels.BrokeragesViewModel`
    b. `Modify the class in a way that once you select the Brokerages, it shows a list of subscribed clients`

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 need 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 2 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
    a. On the publish tab, and under 'Service Dependencies' section, find the 'SQL Server Database, and click 'Configure' in front of it.
    b. On the next page, select 'Azure SQL Database'
    c. Click next on the following page, too
    d. On the 'Configure Azure SQL Database',
        i. enter your Azure SQL username and password
        ii. On 'Save Connection String in', select 'None'
        iii. click next
    e. Uncheck both NuGet Packages and Secrets store
    f. 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 the steps to set up firewall from the Azure Database document under extra materials

Push your code to GitHub

1. Note: No PASSWORDs should be pushed to Git. You must add both your appsettings files to gitignore, so they never get pushed to git. Thus the must be in .gitignore

Grading Scheme

task	mark
program.cs	2
folder structures	1
Error page	1
css	1
Client.cs	5
Brokerage.cs	4
Subscription.cs	2
Dbcontext	3
Viewmodel	3
Layout	1
Brokerages controller	4
Client controller	1
Home controller	1
dbinitializer	1
Client views	2
Brokerage views	2
Brokerage Index	4
gitignore	3
publishing	4
working end-to-end solution	5
Total	50