Lab: Developing ASP.NET MVC Core Models

# Scenario

You are planning to create and code an MVC model that implements your plan for photos and comments in the Adventure Works photo sharing application. The model must store data in a SQL database and include properties that describe photos, comments, and their content. The model must enable the application to store uploaded photos, edit their properties, and delete them in response to user requests.

# Objectives

After completing this lab, you will be able to:

* Create a new ASP.NET MVC Core project in Visual Studio.
* Add new models to the ASP.NET MVC Core web application and add properties to the models.
* Use display and edit data annotations in the MVC models to assign property attributes to views and controllers.
* Add Entity Framework code to persist your models into a SQL database.
* Use Tools to generate migrations and seed some initial data.
* Ensure that the database gets created during startup.

**Estimated Time**: 30 minutes

# Exercise 1: Creating an MVC Project and Adding a Model

## Scenario

In this exercise, you will:

* Create a new MVC Core web application in Visual Studio.
* Add model classes to the web application.

The main tasks for this exercise are as follows:

1. Create a new MVC project.
2. Add a new MVC model.

### Task 1: Create a new MVC project.

1. Start Visual Studio and create a new ASP.NET MVC Core (.NET Core) web application by using the following information:

* **Name**: PhotoSharingApplication
* **Location**: Allfiles (C):\LabfilesMVCCore\Mod03
* **Solution name**: PhotoSharingApplication
* **Create directory for solution**: True
* **Templates**: ASP.NET Core 2.1
* **Project template**: Web Application (Model-View-Controller)
* **Authentication**: No Authentication
* **Host in the cloud**: false
* **Enable Docker Support**: false
* **Configure for HTTPS**: true

### Task 2: Add a new MVC model.

This is the schema of the models for your application

PHOTO

+ Id: int { get; set; }

+ Title: string { get; set; } [Required]

+ PhotoFile: byte[] { get; set; } [DisplayName("Picture")]

+ ImageMimeType : string { get; set; }

+ Description: string { get; set; } [DataType(DataType.MultilineText)]

+ CreatedDate : DateTime { get; set; } [DataType(DataType.DateTime), DisplayFormat(DataFormatString ="{0:dd/MM/yy}", ApplyFormatInEditMode =true)]

+ UserName : string { get; set; }

+ Comments : virtual List<Comment> { get; set; }

COMMENT

+ Id : int { get; set; }

+ PhotoId : int { get; set; }

+ UserName : string { get; set; }

+ Subject : string { get; set; } [Required, StringLength(250)]

+ Body : string { get; set; } [DataType(DataType.MultilineText)]

+ Photo : virtual Photo { get; set; }

1. Under the **Models folder** in the **PhotoSharingApplication** project
   1. Add a new model class by using the following information:

* **Class name**: Photo
  1. Add another model class by using the following information:
* **Class name**: Comment

**Results**: After completing this exercise, you will be able to create an MVC Core web application and add model classes to the web application.

# Exercise 2: Adding Properties to MVC Models

## Scenario

In this exercise, you will:

* Add properties to the Photo and the Comment model classes.
* Implement a relationship between model classes.

The main tasks for this exercise are as follows:

1. Add properties to the Photo model class.
2. Add properties to the Comment model class.
3. Implement a relationship between model classes.

### Task 1: Add properties to the Photo model class.

1. Add a primary key property to the Photo model class by using the following information:

* **Scope**: public
* **Property name**: Id
* **Data type**: integer
* **Acces**s: Read and write

1. Add a title property to the Photo model class by using the following information:

* **Scope**: public
* **Property name**: Title
* **Data type**: string
* **Access**: Read and write

1. Add an image property to the Photo model class by using the following information:

* **Scope**: public
* **Property names**: PhotoFile
* **Data type**: byte []
* **Access**: Read and write

1. Add a MIME Type property to the Photo model class by using the following information:

* **Scope**: public
* **Property names**: ImageMimeType
* **Data type**: string
* **Access**: Read and write

1. Add a description property to the Photo model class by using the following information:

* **Scope**: public
* **Property name**: Description
* **Data type**: String
* **Access**: Read and write

1. Add a date property to the Photo model class by using the following information:

* **Scope**: public
* **Property name**: CreatedDate
* **Data type**: DateTime
* **Access**: Read and write

1. Add a user name property to the Photo model class by using the following information:

* **Scope**: public
* **Property name**: UserName
* **Data type**: string
* **Access**: Read and write

### Task 2: Add properties to the Comment model class.

1. Add a primary key to the Comment model class by using the following information:

* **Scope**: public
* **Property name**: Id
* **Access**: Read and write

1. Add a PhotoId property to the Comment model class by using the following information:

* **Scope**: public
* **Property name**: PhotoId
* **Data** type: integer
* **Access**: Read and write

1. Add a user name property to the Comment model class by using the following information:

* **Scope**: public
* **Property name**: UserName
* **Data type**: string
* **Access**: Read and write

1. Add a subject property to the Comment model class by using the following information:

* **Scope**: public
* **Property name**: Subject
* **Data type**: string
* **Access**: Read and write

1. Add a body text property to the Comment model class by using the following information:

* **Scope**: public
* **Property name**: Body
* **Data type**: string
* **Access**: Read and write

### Task 3: Implement a relationship between model classes.

1. Add a new property to the Photo model class to retrieve comments for a given photo by using the following information:

* **Scope**: public virtual
* **Property name**: Comments
* **Data type**: a List of Comments
* **Access**: Read and write

1. Add a new property to the Comment model class to retrieve the photo for a given comment by using the following information:

* **Scope**: public virtual
* **Property name**: Photo
* **Property type**: Photo
* **Access**: Read and write

**Results**: After completing this exercise, you will be able to add properties to classes to describe them to the MVC runtime. You will also implement a one-to-many relationship between classes.

# Exercise 3: Using Data Annotations in MVC Models

## Scenario

In this exercise, you will:

* Add data annotations to the properties to help MVC web application render them in views and validate user input.

The main tasks for this exercise are as follows:

1. Add display and edit data annotations to the model.
2. Add validation data annotations to the model.

### Task 1: Add display and edit data annotations to the model.

1. Add a display data annotation to the Photo model class to ensure that the PhotoFile property is displayed with the name **Picture**.
2. Add an edit data annotation to the Photo model class that ensures the Description property editor is a multiline text box.
3. Add the following data annotations to the Photo model class to describe the CreatedDate property:

* **Data type**: DateTime
* **Display name**: Created Date
* **Display format**: {0:dd/MM/yy} (ApplyFormatInEditMode : true)

1. Add an edit data annotation to the Comment model class that ensures that the Body property editor is a multiline text box.

### Task 2: Add validation data annotations to the model.

1. Add a validation data annotation to the Photo model class to ensure that the users complete the Title field.
2. Add validation data annotations to the Comment model class to ensure that the users complete the Subject field and enter a string with a length shorter than 250 characters.

**Results**: After completing this exercise, you will be able to add property descriptions and data annotations to the two model classes in the MVC web application.

# Exercise 4: Use Entity Framework to persist your data

## Scenario

In this exercise, you will:

* Add Entity Framework code to the Photo Sharing application in code-first mode.
* Setup and configure the Database

The main tasks for this exercise are as follows:

1. Write and configure an Entity Framework Context.
2. Add Migrations.
3. Seed the database with an initial photo and comments.
4. Create and seed database during startup.

### Task 1: Write and configure Entity Framework Context.

1. Add a DbContext
   1. Add a Data folder
   2. In the Data folder, create a new class PhotoSharingApplicationContext
   3. Derive your class from DbContext
   4. Add a read / write property Photos of type DbSet<Photo>
   5. Add a read / write property Comments of type DbSet<Comment>
   6. Add a constructor accepting a DbContextOptions<PhotoSharingApplicationContext> options parameter
      1. Let the constructor pass the parameter to the constructor of the base class
2. Configure the DbContext
   1. Add the connection string to the settings file
      1. Open appsettings.json file
      2. Add a ConnectionStrings json object
      3. Add a PhotoSharingApplicationContext property with the following value:

"Server=(localdb)\\mssqllocaldb;Database=PhotoSharingApplicationContext;Trusted\_Connection=True;MultipleActiveResultSets=true"

* 1. Inject the DbContext as a service
     1. Open the Startup file
     2. Locate the ConfigureServices method
     3. Add a call to the AddDbContext extension method
     4. Use the UseSqlServer options
     5. Read the connection string from the Configuration object

### Task 2: Add an Initial Migration

You need to create the database, and you'll use the EF Core Migrations feature to do that. Migrations lets you create a database that matches your data model and update the database schema when your data model changes.

1. From the Tools menu, select NuGet Package Manager > Package Manager Console.
2. In the PMC, enter the following commands:  
   Add-Migration Initial

The Add-Migration command creates code to create the initial database schema. The schema is based on the model specified in the DbContext(In the Data/PhotoSharingApplicationContext.cs file). The Initial argument is used to name the migrations. You can use any name, but by convention you choose a name that describes the migration.

### Task 2: Seed the database during model creation

The first migration took care of the schema of the two tables. You now also want to seed the database with an initial photo and a couple of comments.

You are going to override the OnModelCreating method of your PhotoSharingApplicationContext to make sure that the Photo entity has one photo and that the Comment entity has two comments. In order to do that you’re going to use a feature introduced in EF Core 2.1: the HasData method.

You will then add a new migration which will generate a class to insert the data in the database. You will need to improve the generated code to optimize it.

Your strategy to insert a picture file in the database is going to be the following: you will embed one picture as a resource into your project, then write a method that finds the picture and returns an array of bytes. You will use this method to fill the PhotoFile property of your Photo model during the OnModelCreating.

The Add-Migration command will generate three .cs files which will contain the serialized byte array, hardcoded into the migration classes. You will replace the array of byte with a call to your method so that your compiled library won’t contain your picture four times (one time as an embedded resource and three times as an array in the code).

The main steps for this task are as follows:

1. Embed a .jpg file as a resource into your project.
2. Write a method that finds the embedded resource, reads its content and returns it as an array of byte.
3. Override the OnModelCreating to make sure that your database has a photo and two related comments.
4. Add a migration and optimize its code

##### Embed a .jpg file as a resource into your project

1. Create a folder named “images” in the root of your project
2. Copy the “flower.jpg” file from LabFiles\Mod03\Resources\images to your images folder
3. In Visual Studio, right click on the flower.jpg file, select Properties. Under the Build Action property, select “Embedded Resource”.

##### Write a method that finds the embedded resource, reads its content and returns it as an array of byte.

1. In your Data folder, create a new class named EmbeddedFileReader
2. Create a public static method that accepts a string path and returns an array of byte. Name the method GetFileBytes
3. Get the executing assembly by invoking the Assembly.GetExecutingAssembly method
4. Create an instance of an EmbeddedFileProvider, passing the executing assembly to the constructor
5. Get an IFileInfo by invoking the GetFileInfo of the file provider, passing the path as a parameter
6. Get a Stream by invoking the CreateReadStream method of the IFileInfo
7. Create an instance of a BinaryReader passing the stream and Encoding.UTF8 as parameters
8. Invoke the ReadBytes method of the BinaryReader passing the Length property of the Stream. Return the resulting byte array.
9. Do not forget to dispose of the BinaryReader

##### Override the OnModelCreating to make sure that your database has a photo and two related comments.

1. Open your PhotoSharingApplication class
2. Override the OnModelCreating method
3. Invoke the OnModelCreating of the base class
4. Get hold of the EntityTypeBuilder of your Photo model by invoking the Entity<Photo> method of the modelBuilder
5. Invoke the HasData method of the EntityTypeBuilder<Photo> passing a new Photo object initialized as follows:

new Photo {

Id = 1,

Title = "Me standing on top of a mountain",

Description = "I was very impressed with myself",

UserName = "Fred",

PhotoFile = EmbeddedFileReader.GetFileBytes("images.flower.JPG"),

ImageMimeType = "image/jpeg",

CreatedDate = DateTime.Today

}

1. Get hold of the EntityTypeBuilder of your Comment model by invoking the Entity<Comment> method of the modelBuilder
2. Invoke the HasData method of the EntityTypeBuilder<Comment> passing two new Comment objects initialized as follows:

new Comment {

Id = 1,

PhotoId = 1,

UserName = "Bert",

Subject = "A Big Mountain",

Body = "That looks like a very high mountain you have climbed"

}

new Comment {

Id = 2,

PhotoId = 1,

UserName = "Sue",

Subject = "So?",

Body = "I climbed a mountain that high before breakfast everyday"

}

##### Add a migration and optimize its code

Open the Package Manager Console and type Add-Migration SeedData

Under the Migrations folder, find and open the SeedData, the SeedData.Designer and the ModelSnapshot files. Should Visual Studio hang or freeze when you try to open the files, try disabling Code Lens or opening them in Visual Studio Code.

If you search for “byte[]” you will find occurrences where the PhotoFile property is set to an array of byte whose content is a long sequence of comma separated numbers: that is the flower.jpg file, written into your source code multiple times. Replace the byte array with a call to PhotoSharingApplication.Data.EmbeddedFileReader.GetFileBytes("images.flower.JPG").

This is not strictly necessary, but at least you avoid having multiple copies of the same file compiled into your library multiple times.

### Task 3: Create the Database and apply each migration during startup

As a last step, you want to make sure that the database gets generated and that all the migrations are applied during startup, after having configured your web host but right before running it.

You need to get hold of your PhotoSharingApplicationContext so that you can call its Migrate method. Because your Startup has injected the PhotoSharingApplicationContext as a Scoped service, you need to create a scope, get the service provider from the scope, then get the required service from the service provider.

https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/intro?view=aspnetcore-2.1

1. Modify the Main method of Program.cs class to look like this:

IWebHost host = CreateWebHostBuilder(args).Build();

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

var services = scope.ServiceProvider;

try {

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

context.Database.Migrate();

} catch (Exception ex) {

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

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

}

}

host.Run();

**Run the application and verify that the database has been created, that it contains the Photos and Comments table and that the tables contain the expected data.**

**Results**: After completing this exercise, you will be able to create an MVC application that uses a SQL Database as its data store. In the next lab you are going to write a Controller that makes use of your PhotoSharingApplicationContext.

**Question**: You are building a site that collects information from customers for their accounts. You want to ensure that customers enter a valid email address in the Email property. How would you do this?

**Question**: You have been asked to create an intranet site that publishes a customer database, created by the sales department, to all employees within your company. How would you create the model with Entity Framework?