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 a new model to the ASP.NET MVC Core web application and add properties to the model.
* Use display and edit data annotations in the MVC model to assign property attributes to views and controllers.
* Use Visual Studio to create a new SQL database and connect to the database.
* Add Entity Framework code to the model classes in the MVC model.
* Use display and edit data annotations in the MVC model to assign property attributes to views and controllers.

**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 web application by using the following information:

* **Name**: PhotoSharingApplication
* **Location**: Allfiles (C):\LabfilesMVCCore\Mod03
* **Solution name**: PhotoSharingApplication
* **Create directory for solution**: True
* **Project template**: MVC
* **Authentication**: No Authentication
* **Host in the cloud**: false

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

1. Create a folder named **Models** in the **PhotoSharingApplication** project
2. Add a new model class to the Models folder by using the following information:

* **Class name**: Photo

1. Add another model class to the Models folder 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
* **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
* **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: Creating a New SQL Database

## Scenario

In this exercise, you will:

* Update all the packages to the latest version
* 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. Add an Entity Framework Context to the model.
2. Add an Initializer.
3. Create a SQL database and obtain a connection string.

### Task 1: Add an Entity Framework Context to the model.

1. Use the NuGet Package Manager to Update all the Packages
2. Use the NuGet Package manager to add the latest version of
   1. Microsoft.EntityFrameworkCore.SqlServer
   2. Microsoft.EntityFrameworkCore.SqlServer.Design
   3. Microsoft.EntityFrameworkCore.Tools
3. Add a new class called PhotoSharingContext to a Data folder and ensure that the new class inherits the Microsoft.EntityFrameworkCore.DbContext class.
4. Add a constructor with a parameter of type DbContextOptions<PhotoSharingContext> that passes the options to the base contructor
5. Override the OnConfiguring method and call the base.OnConfiguring
6. Add public DbSet properties to Photos and Comments to enable Entity Framework to create database tables called Photos and Comments.

### Task 2: Add an Initializer

1. Add a new static class called PhotoSharingInitializer to the Data folder.
2. Open the getFileBytes.txt file from the following location and add all the text of the file as new methods to the PhotoSharingInitializer class:

* File path: Allfiles (C):\LabfilesMVCCore\Mod03\CodeSnippets

1. Create an extension method of an IApplicationBuilder:
   1. public async static Task Seed(this IApplicationBuilder app, string path)
2. Initialize a **context** variable with the result of app.ApplicationServices.GetService<PhotoSharingContext>()
3. Invoke the context.Database.Migrate() method to ensure that the database gets created and all the Migrations are applied
4. Invoke the addPhoto() method passing the context and a newly created Photo objects with the following properties:

* new Photo {
* Title = "Me standing on top of a mountain",
* Description = "I was very impressed with myself",
* UserName = "Fred",
* PhotoFile = getFileBytes($@"{path}\Images\flower.jpg"),
* ImageMimeType = "image/jpeg",
* CreatedDate = DateTime.Today,
* Comments = new List<Comment>(){
* new Comment {
* UserName = "Bert",
* Subject = "A Big Mountain",
* Body = "That looks like a very high mountain you have climbed"
* },
* new Comment {
* UserName = "Sue",
* Subject = "So?",
* Body = "I climbed a mountain that high before breakfast everyday"
* }
* }
* }

1. Invoke the addPhoto() method passing the context and a newly created Photo objects with the following properties:

* new Photo {
* Title = "My New Adventure Works Bike",
* Description = "It's the bees knees!",
* UserName = "Fred",
* PhotoFile = getFileBytes($@"{path}\Images\orchard.jpg"),
* ImageMimeType = "image/jpeg",
* CreatedDate = DateTime.Today,
* Comments = new List<Comment>(){
* new Comment {
* UserName = "Fred",
* Subject = "Jealous",
* Body = "Wow, that new bike looks great!"
* }
* }
* }

1. Invoke the addPhoto() method passing the context and a newly created Photo objects with the following properties:

* new Photo {
* Title = "View from the start line",
* Description = "I took this photo just before we started over my handle bars.",
* UserName = "Sue",
* PhotoFile = getFileBytes($@"{path}\Images\path.jpg"),
* ImageMimeType = "image/jpeg",
* CreatedDate = DateTime.Today
* }

1. Save the changes to the context
2. Open Startup.cs and add a line of code to the Configure method that calls app.SeedData, passing a env.WebRootPath as path. Also add the following namespaces:

* PhotoSharingApplication.Models;
* Microsoft.EntityFrameworkCore;

### Task 3: Configure the SQL database.

1. Modify the appsettings.json:
   1. Add a "ConnectionStrings" property
   2. Set the value to an object
   3. Add a "PhotoSharingContext" property to that object
   4. Set the value to "Server=(localdb)\\mssqllocaldb;Database=PhotoSharingApplication-lab03;Trusted\_Connection=True;"

* {
* "ConnectionStrings": {
* "PhotoSharingContext": "Server=(localdb)\\mssqllocaldb;Database=PhotoSharingApplication-lab03;Trusted\_Connection=True;"
* },
* "Logging": {
* "IncludeScopes": false,
* "LogLevel": {
* "Default": "Debug",
* "System": "Information",
* "Microsoft": "Information"
* }
* }
* }

1. Register the PhotoSharingContext as a service
   1. On the Startup.cs , in the ConfigureServices method, before calling services.AddMvc()
      1. Read the connection string from the Configuration object, using the GetConnectionString method and passing "PhotoSharingContext" as a parameter
      2. Add a DbContext of type PhotoSharingContext to the service collection object
      3. Configure the options to use SqlServer passing the connection string read on step i
   2. Add the following namespaces
      1. PhotoSharingApplication.Data
      2. Microsoft.EntityFrameworkCore
2. Build the Photo Sharing application.
3. Add the initial migration
   1. Go to the Package Manager Console
   2. Type Add-Migration “Initial”
4. Build the Photo Sharing application.

**Results**: After completing this exercise, you will be able to create an MVC application that uses a SQL Database as its data store.

# Exercise 5: Testing the Model and Database

## Scenario

In this exercise, you will:

* Configure your project to enable Visual Studio to automatically generate the Controller with the Views
* Add a controller and views to the MVC web application.
* Run the web application.

The main tasks for this exercise are as follows:

1. Configure the project
2. Add a controller and views.
3. Add an image and run the application.

### Task 1: Configure the project

1. Add the following NuGet packages to your project:
   1. "Microsoft.EntityFrameworkCore.Design"
   2. "Microsoft.EntityFrameworkCore.Tools"
   3. "Microsoft.VisualStudio.Web.CodeGeneration.Tools"
   4. "Microsoft.VisualStudio.Web.CodeGenerators.Mvc"
2. Add the following Tools to your project
   1. "Microsoft.EntityFrameworkCore.Tools"
   2. "Microsoft.VisualStudio.Web.CodeGeneration.Tools"
3. Build your application

### Task 2: Add a controller and views.

1. Add a new controller to the PhotoSharingApplication project by using the following information:

* **Template**: MVC Controller views, using Entity Framework
* **Model class**: Photo
* **Data context class**: PhotoSharingContext
* **Generate Views**: true
* **Reference Script Libraries**: true
* **Use Layout Page**: true
* **Controller Name**: PhotosController

### Task 3: Add an image and run the application.

1. Copy 3 images into the wwwroot/images folder by using the following information:

* **Destination folder name**: wwwroot/images
* **Images to be copied**: flower.jpg, orchard.jpg, path.jpg
* **Source location of the images**: Allfiles (C):\LabfilesMVCCore\Mod03\Images

1. Run the application by debugging, and access the following relative path:

* /photos

**Results**: After completing this exercise, you will be able to add controllers, views, and images to an MVC web application and test the application by displaying data from a SQL database.

**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?