CarLister: Homework #2 – Add Database storage and image upload

Task 1) Adding a database table to store car information

1. In the **Solution** **Explorer**, from the **Models** folder, open the **CarViewModel** class. We will use this class to represent the cars in our database.
2. Add an integer **Id** attribute to the class, a string **Path** attribute, and change the **Value** attribute from a string to a float (we’ll need it to be a float for later things like sorting, etc):  
     
    public int Id { get; set; }  
    public string Path { get; set; }  
    …   
    …  
    …  
    [Display(Name = "Price")]  
    public float Value { get; set; }
3. Add the following class to the CarsViewModels file below the CarViewModel class:

public class CarListerDBContext : DbContext  
 {  
 public DbSet<CarViewModel> Cars { get; set; }  
 }

1. Add the following **using** statement to the top of the file:

using System.Data.Entity;

1. Now we need to add a **Connection** **String** to the **Web.config** file so that your application can create a connection to the database to store your car information. Open the **Web.config** file at the bottom of the **Solution** **Explorer** and locate the **<connectionStrings>** tag.
2. Add the following string to this section below any other connection strings that are currently present:

<add name="CarListerDBContext" connectionString="Data   
 Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\Cars.mdf;  
 Integrated Security=True" providerName="System.Data.SqlClient"

1. This small amount of code and XML is everything you need to write in order to represent and store your cars data in a database.
2. Now we need to build a new CarsController class that you can use to display the car data and allow users of your website to add new car listings and upload image files.

Task 2) Build the new CarsController

1. Before we begin, you need to build your application or you will get an error when you try to add the controller. From the **Debug** menu, select **Build**.
2. Correct any syntax errors that may have been reported and build again. When you have a successful build, continue to step 3.
3. In the **Solution Explorer**, right click on the **Controllers** folder, select **Add**, and choose **Controller**.
4. Select **MVC 5 Controller with views, using Entity Framework**.
5. Enter **CarsController** for the Controller name.
6. Select **CarViewModel** for the Model class.
7. Select **CarListerDbContext** for the Data context class.
8. Click **Add**.
9. Visual Studio has now created a new set of controller methods and their corresponding views. All of these currently work and will result in successful logging of car listings into your database. Test it out by changing the **Car Lister link** on your home page (found on the menu in **Views/Shared/\_Layout.cshtml**) and running the application.
10. Currently your list does not behave in the same manner as the one you built during the previous class session. Fix this by changing the code in the new **Index.cshtml** file under **Views/Cars** to reflect the behavior in the previous list view.
11. Change the display line for the Value attribute in the foreach loop. It needs to be formatted for better viewing. We can do this as follows:

$@item.Value.ToString("N0")

The value will display with a dollar sign in front, followed by the numeric value stored in this attribute. The format specified “N0” will display the number with a comma every three digits and zero decimal places. If you want decimal places to appear, change the 0 to the number of places – i.e. “N2”.

1. When you have a successful build, you may delete the **old CarListerController** and the **old CarLister view** that you were using previously. You will no longer need them.
2. Make sure your main menu references the new car lister (Views\Cars\Index.cshtml) and test your application.

Task 3) Enable uploading of image files with car data

1. We will need to modify our Create.cshtml view, which allows us to add a new car to the database, and then its corresponding controller to facilitate the file upload. Locate and open Views/Cars/**Create.cshtml**.
2. We will make three changes here. First, modify the statement that begins the form ( @using (Html.BeginForm()) ) as follows:

@using (Html.BeginForm("Create", "Cars", FormMethod.Post, new { enctype = "multipart/form-data" }))

This instructs the specified Post method, “Create”, to expect multiple data entities from the view. This is necessary since we need to add a control to the page to handle the uploading of a file.

1. Next, add a hidden attribute (Html.HiddenFor(…)) inside the form for the **Path** attribute of the view model.
2. Finally, locate the form group for the **PicUrl** attribute of your model. Comment out the two lines that create an editor and a validation message for the attribute. In their place, add the following HTML code:

<input type="file" id="fileUpload" name="fileUpload" />

This line creates a new input field of type file; that is, it lets the user browse for a file and then creates a reference to that file associated with the variable name assigned in the tag (**fileUpload**).

1. Now open the **CarsController.cs** file. Add the following using statement at the top of the file to give access to the necessary file IO methods you need to process the uploaded file:

using System.IO;

1. Now find the **Post** version of the **Create** method. Add the **fileUpload** variable to the parameter list of the method. Its type is **HttpPostedFileBase**.

public ActionResult Create(CarViewModel carViewModel, HttpPostedFileBase fileUpload)

1. The rest of your modifications will take place inside the if statement, **if (ModelState.IsValid)**. First, add a new if statement inside the one mentioned that checks the fileUpload variable to make sure it is 1) not null, and 2) has a ContentLength greater than zero:

if (fileUpload != null && fileUpload.ContentLength > 0)   
 {  
  
 }

* 1. Inside this if statement, create a new variable called **fileName** and set it equal to the filename of the uploaded file, referenced by the **fileUpload** variable. You will need to use the Path object and its GetFileName method to retrieve this information:

var fileName = Path.GetFileName(fileUpload.FileName);

* 1. Next, assign the **Path** attribute of the **carViewModel** object to the new server location in which the uploaded file will be stored. Use the **Path** object’s **Combine** method to create this as follows:

carViewModel.Path = Path.Combine(Server.MapPath("~/img/"),  
 fileName);

* 1. Save the uploaded file to the newly specified path location by calling the **SaveAs** method of the **fileUpload** variable and passing it the new **path**:

fileUpload.SaveAs(carViewModel.Path);

* 1. Finally, set the PicUrl attribute of the model to reference the relative path of the new image’s location on the server:

carViewModel.PicUrl = "~/img/" + fileName;

1. Test your application. When you are satisfied that it is functioning correctly, modify the **Edit view and controller** for your cars to allow uploading of image files, just as you have done with **Create**.
2. Test your application.
3. **BONUS 1:** Modify the **Delete** controllerand viewto remove the image associated with the selected car from the server (this is why the Path attribute of the view model is important). When done, verify that the file has been removed by checking the contents of the **img** folder for your project, which is the location we have specified for storing images.
4. **BONUS 2:** Modify the **Edit** controller and viewto allow the user to select a new image to associate with the selected car and upload the image to server.