Building an Automobile Management Web Application using ASP.NET Core MVC and Entity Framework Core

Introduction

Imagine you're an employee of an online car retailer named **Automobile eStore**. Your manager has asked you to develop a Web application for automobile management (CarID, CarName, Manufacturer, Price, and ReleasedYear). The application has to support adding, viewing, modifying, and removing products—a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD).

This lab explores creating an application using ASP.NET MVC with .NET Core, and C#. An **SQL Server Database** will be created to persist the car's data that will be used for reading and managing automobile data by **Entity Framework Core (EF Core)**

Lab Objectives

In this lab, you will:

- Use the Visual Studio.NET to create ASP.NET Core MVC and Class Library (.dll) project.
- Create a SQL Server database named MyStock that has a Cars table.
- Develop a DataProvider class to perform CRUD actions using EF core.
- Apply passing data by ViewBag in ASP.NET Core MVC application.
- Apply Repository pattern and Singleton pattern in a project.
- Add CRUD action methods to ASP.NET Core MVC application.
- Run the project and test the ASP.NET Core MVC actions.

MyStock Database

T470S.MyStock - Diagram_0*

Cars

	Column Name	Condensed Type	Nullable	Identity
🔑	CarID	int	No	☐
	CarName	varchar(50)	No	☐
	Manufacturer	varchar(50)	No	☐
	Price	money	No	☐
	ReleasedYear	int	No	☐
				☐

T470S.MyStock - dbo.Cars

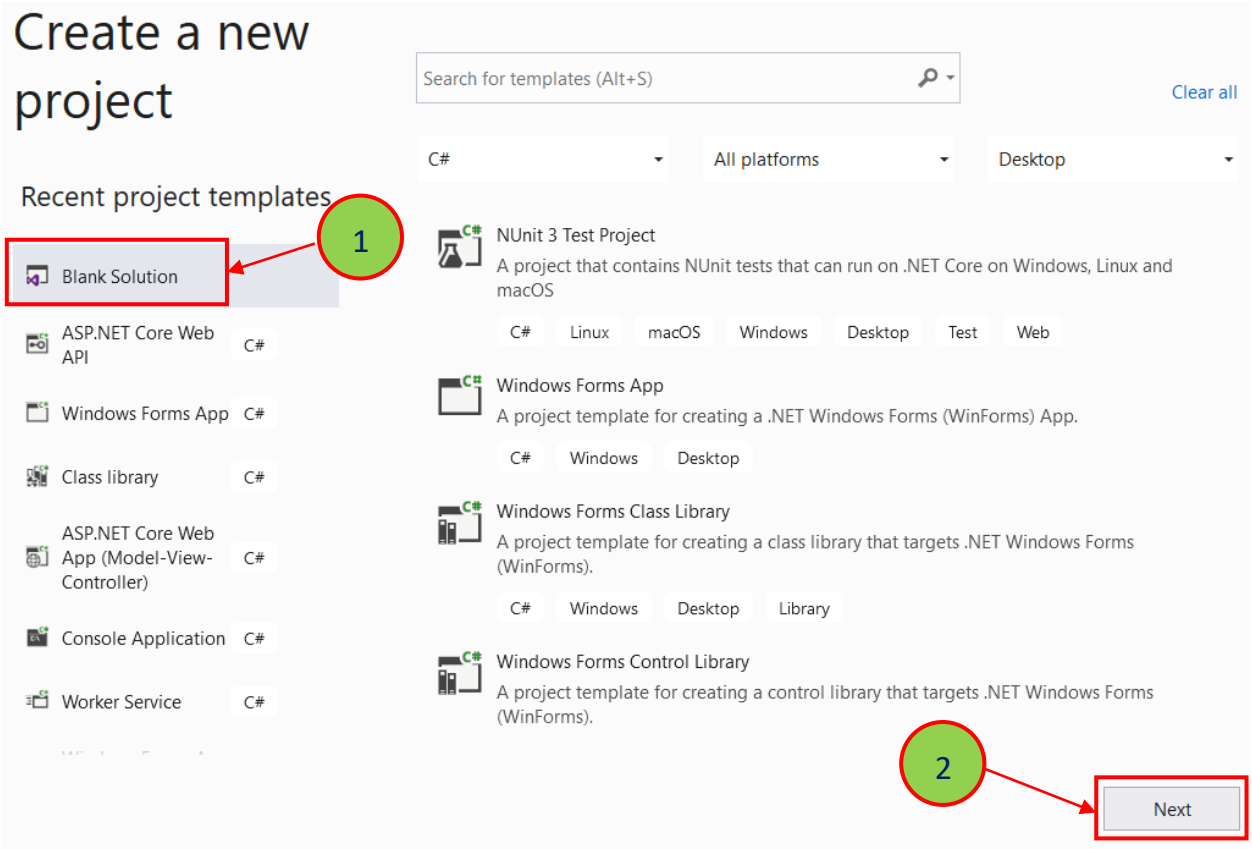
CarID	CarName	Manufacturer	Price	ReleasedYear
1	Accord	Honda Motor Company	24970.0000	2021
3	Clarity	Honda Motor Company	33400.0000	2021
4	BMW 8 Series	BMW	85000.0000	2021
5	Audi A6	Audi	14000.0000	2020

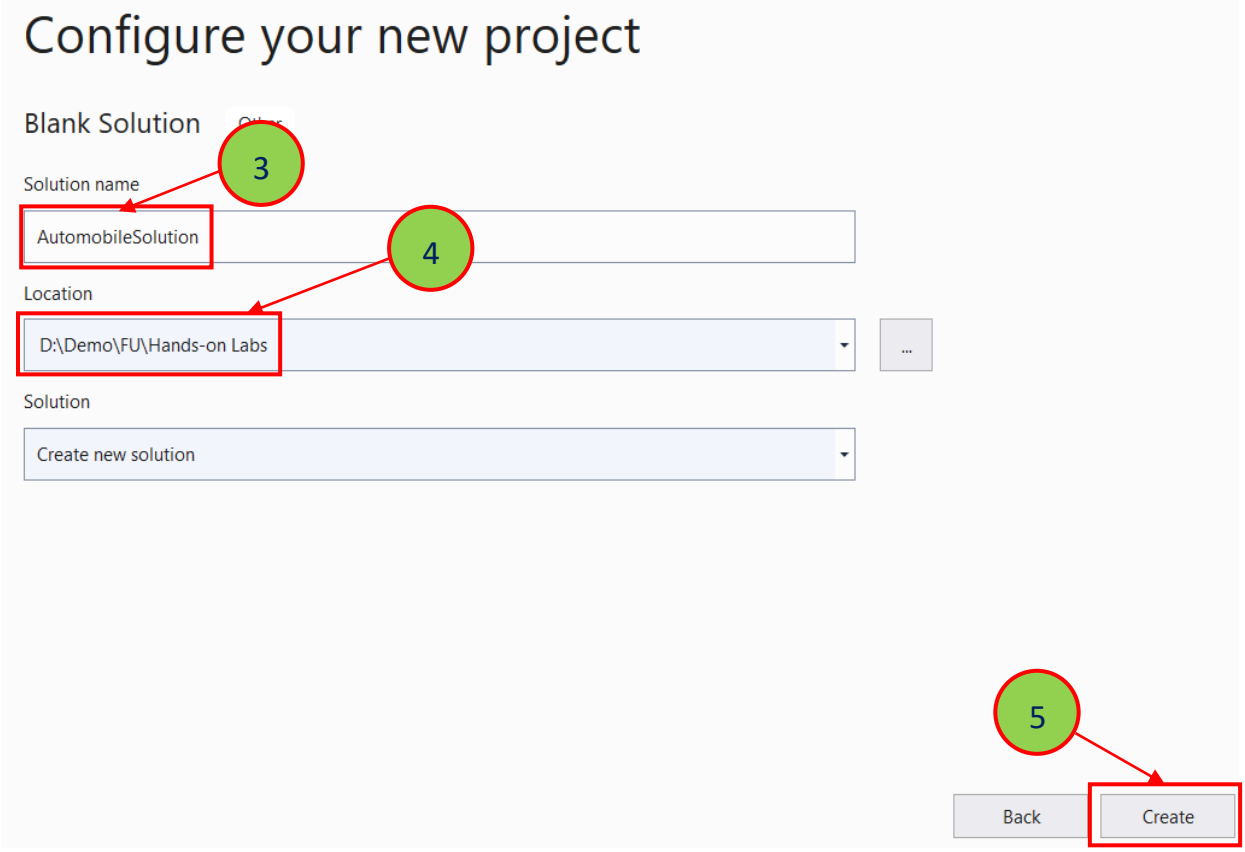
Activity 01: Build a solution by Visual Studio.NET

Create a Blank Solution named **AutomobileSolution** then add new a Class Library Project named **AutomobileLibrary** and a Windows Forms project named **AutomobileWinApp**

Step 01. Create a Blank solution.

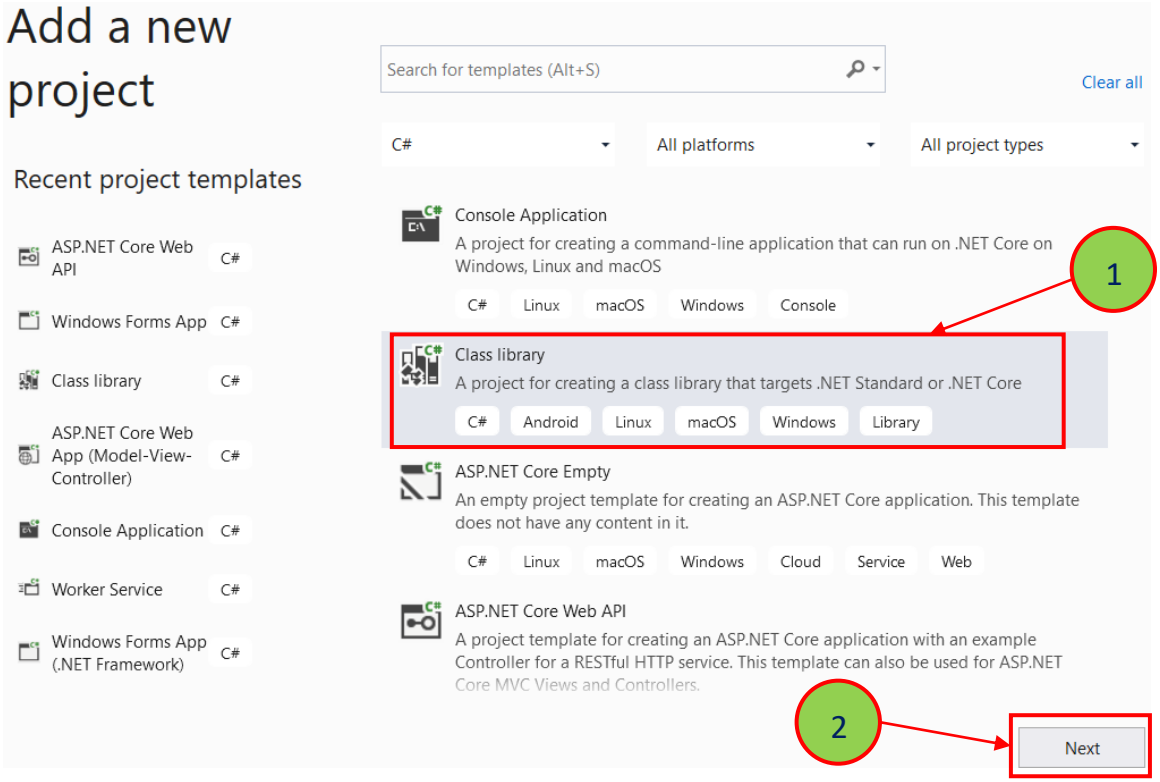
- Open the Visual Studio .NET application and performs steps as follows:

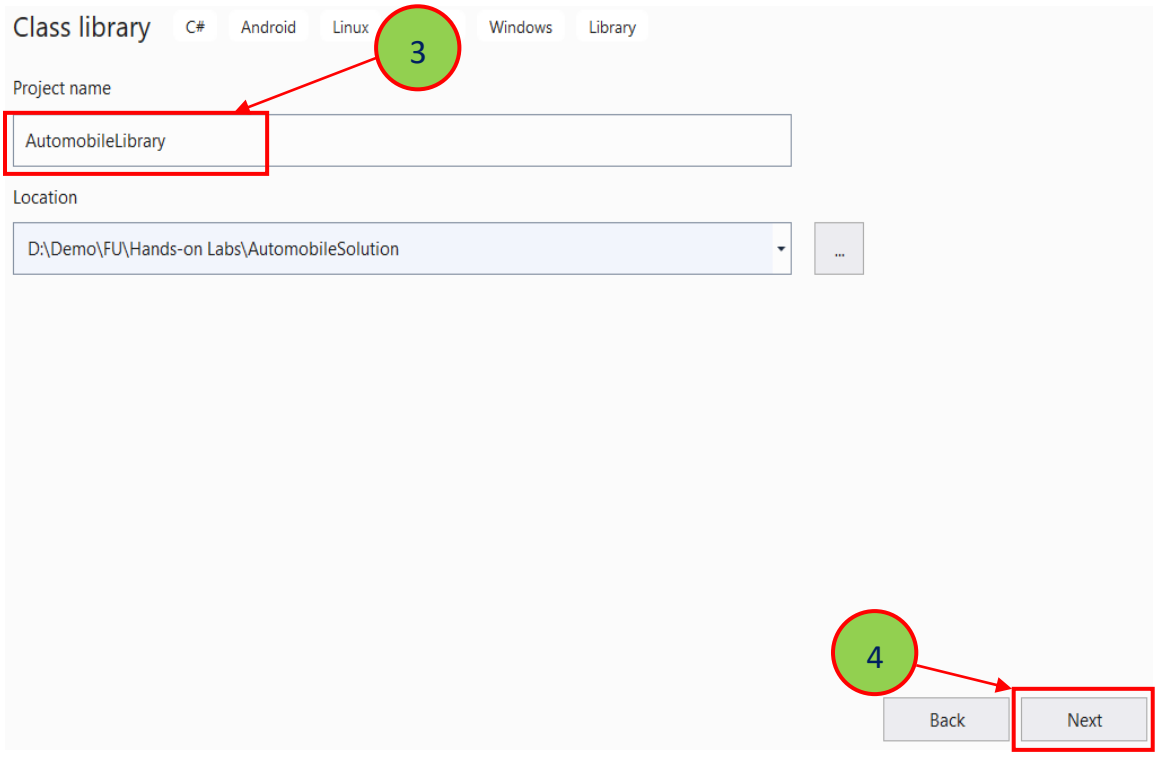

Create a new project
Search for templates (Alt+S)
Clear all
C#
All platforms
Desktop
Recent project templates
1
Blank Solution
ASP.NET Core Web API C#
Windows Forms App C#
Class library C#
ASP.NET Core Web App (Model-View-Controller) C#
Console Application C#
Worker Service C#
NUnit 3 Test Project
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Desktop Test Web
Windows Forms App
A project template for creating a .NET Windows Forms (WinForms) App.
C# Windows Desktop
Windows Forms Class Library
A project template for creating a class library that targets .NET Windows Forms (WinForms).
C# Windows Desktop Library
Windows Forms Control Library
A project template for creating a control library that targets .NET Windows Forms (WinForms).
2
Next

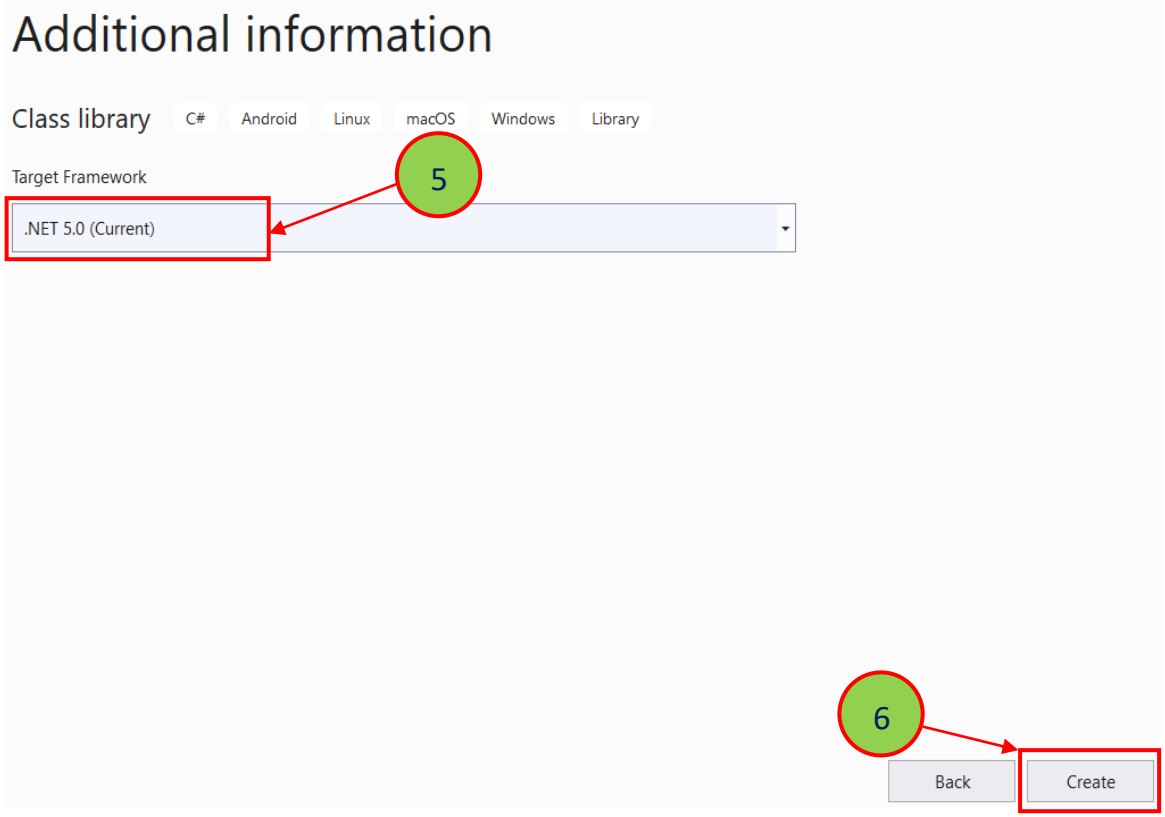

Configure your new project
Blank Solution
3
Solution name
AutomobileSolution
4
Location
D:\Demo\FU\Hands-on Labs
...
Solution
Create new solution
5
Back
Create

Step 02. Create a Class Library project.

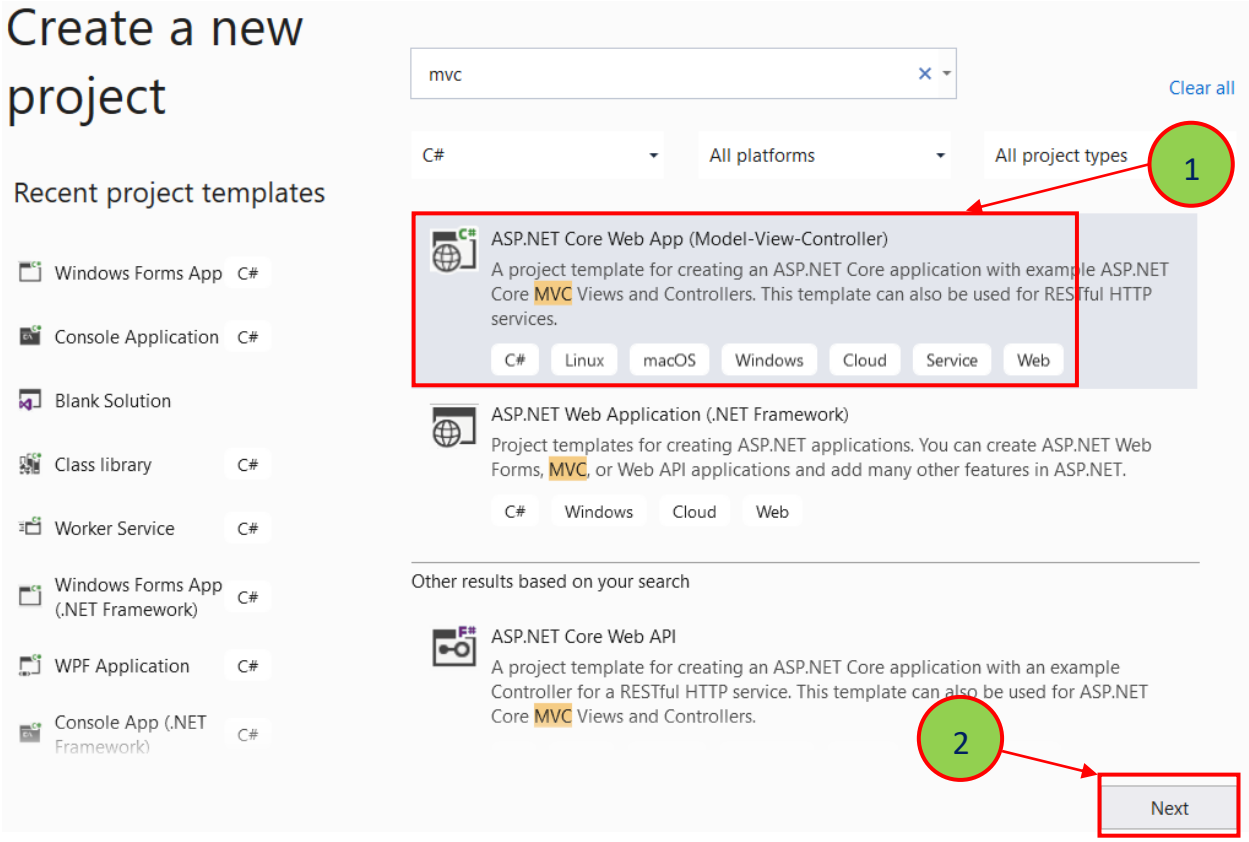
- From the File menu | Add | New Project, on the Add New Project dialog, select “Class Library” and performs steps as follows:

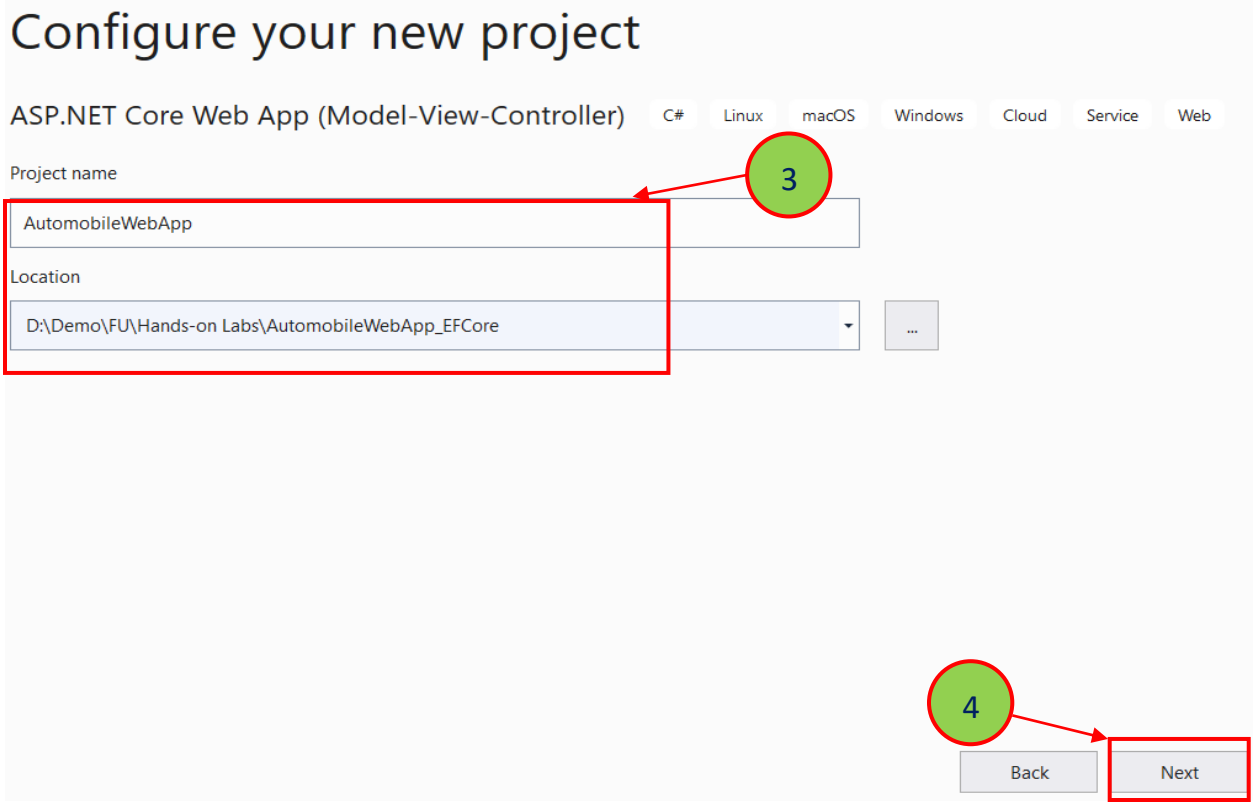


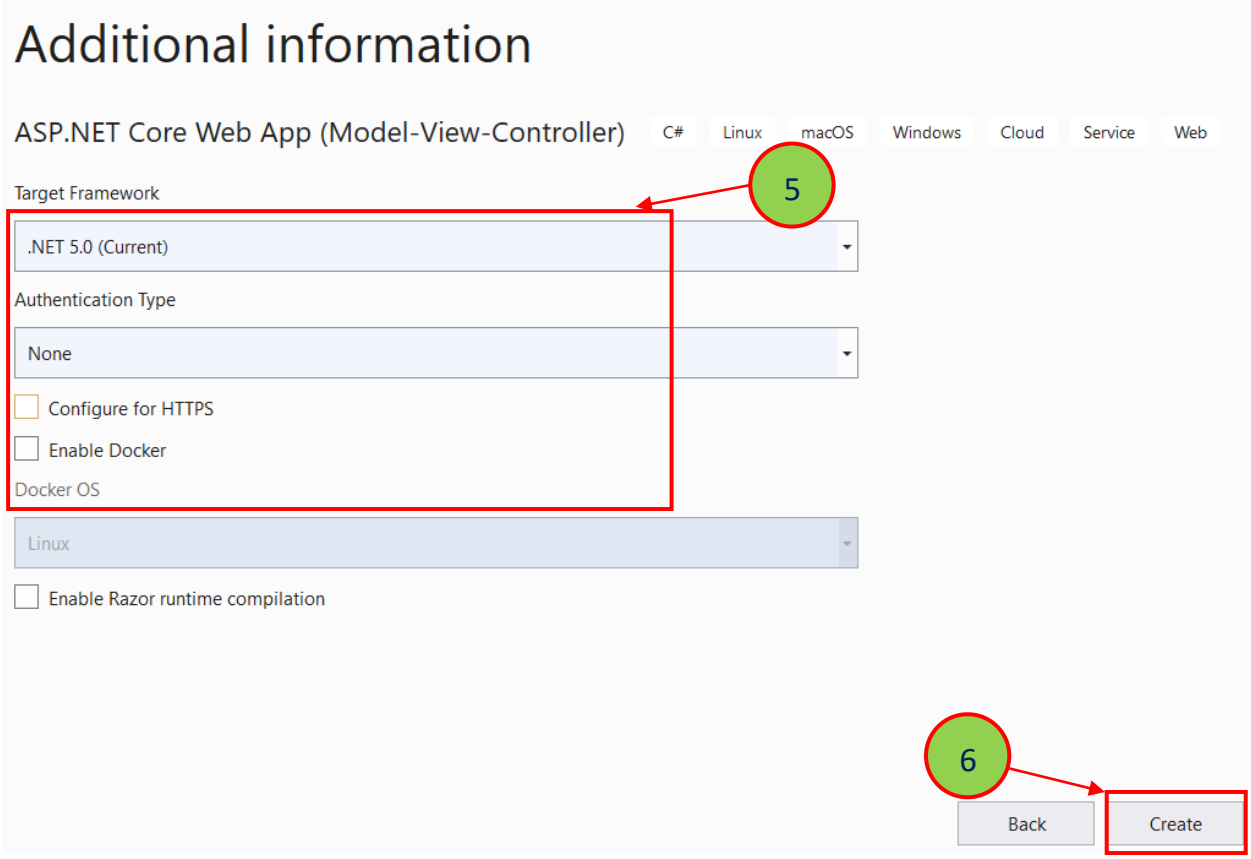




Step 03. Create an ASP.NET Core MVC project.

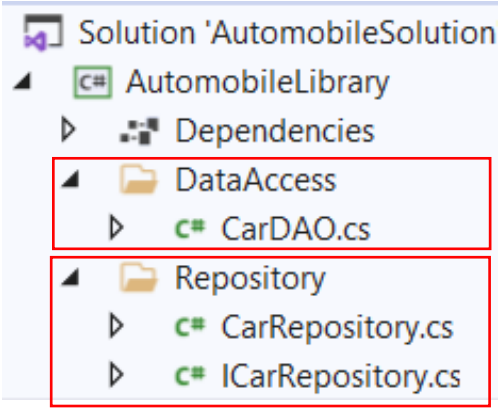



Configure your new project
ASP.NET Core Web App (Model-View-Controller)
C#
Linux
macOS
Windows
Cloud
Service
Web
Project name
3
AutomobileWebApp
Location
D:\Demo\FU\Hands-on Labs\AutomobileWebApp_EFCore
...
4
Back
Next

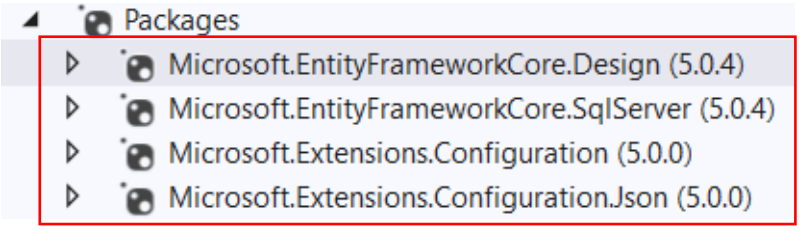

Additional information
ASP.NET Core Web App (Model-View-Controller)
C#
Linux
macOS
Windows
Cloud
Service
Web
Target Framework
5
.NET 5.0 (Current)
Authentication Type
None
Configure for HTTPS
Enable Docker
Docker OS
Linux
Enable Razor runtime compilation
6
Back
Create

Activity 02: Write codes for the AutomobileLibrary project

Step 01. Create folders and add classes to the project as follows:

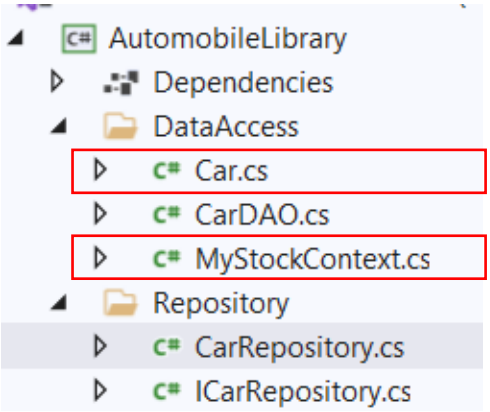


Step 02. Install the following packages from Nuget:



Step 03. Right-click on the project, select **Open in Terminal.** On Developer PowerShell dialog, run **dotnet ef** command to generate database context and Car entity model from **MyStock** database as follows:

```
dotnet ef dbcontext scaffold "Server=(local);uid=sa;pwd=123;database=MyStock"
Microsoft.EntityFrameworkCore.SqlServer --output-dir DataAccess
```



Step 04. Write codes for **CarDAO** class of **CarDAO.cs** as follows:

```
public class CarDAO  {
    //-----------------------------------------------
    //Using Singleton Pattern
    private static CarDAO instance = null;
    private static readonly object instanceLock = new object();
    public static CarDAO Instance {
        get
        {
            lock (instanceLock){
                if (instance == null){
                    instance = new CarDAO();
                }
                return instance;
            }
        }
    }

//-----------------------------------------------------------------
public IEnumerable<Car> GetCarList() {
    var cars = new List<Car>();
    try {
        using var context = new MyStockContext();
        cars = context.Cars.ToList();
    }
    catch (Exception ex){
        throw new Exception(ex.Message);
    }
    return cars;
}
//-----------------------------------------------------------------
public Car GetCarByID(int carID){
    Car car = null;
    try{
        using var context = new MyStockContext();
        car = context.Cars.SingleOrDefault(c => c.CarId == carID);
    }
    catch (Exception ex){
        throw new Exception(ex.Message);
    }
    return car;
}
//-----------------------------------------------------------------
```

```
public void AddNew(Car car){
    try
    {
        Car _car = GetCarByID(car.CarId);
        if (_car == null) {
            using var context = new MyStockContext();
            context.Cars.Add(car);
            context.SaveChanges();
        }
        else {
            throw new Exception("The car is already exist.");
        }
    }
    catch (Exception ex) {
        throw new Exception(ex.Message);
    }
}

//-------------------------------------------------------------
public void Update(Car car){
    try {
        Car _car = GetCarByID(car.CarId);
        if (_car != null) {
            using var context = new MyStockContext();
            context.Cars.Update(car);
            context.SaveChanges();
        }
        else {
            throw new Exception("The car does not already exist.");
        }
    }
    catch (Exception ex){
        throw new Exception(ex.Message);
    }
}
//-------------------------------------------------------------
```

```
    public void Remove(int carID)
    {
        try{
            Car car = GetCarByID(carID);
            if (car != null){
                using var context = new MyStockContext();
                context.Cars.Remove(car);
                context.SaveChanges();
            }
            else{
                throw new Exception("The car does not already exist.");
            }
        }
        catch (Exception ex){
            throw new Exception(ex.Message);
        }
    }//end Remove
}//end class
```

Step 07. Write codes for **ICarRepository.cs** as follows:

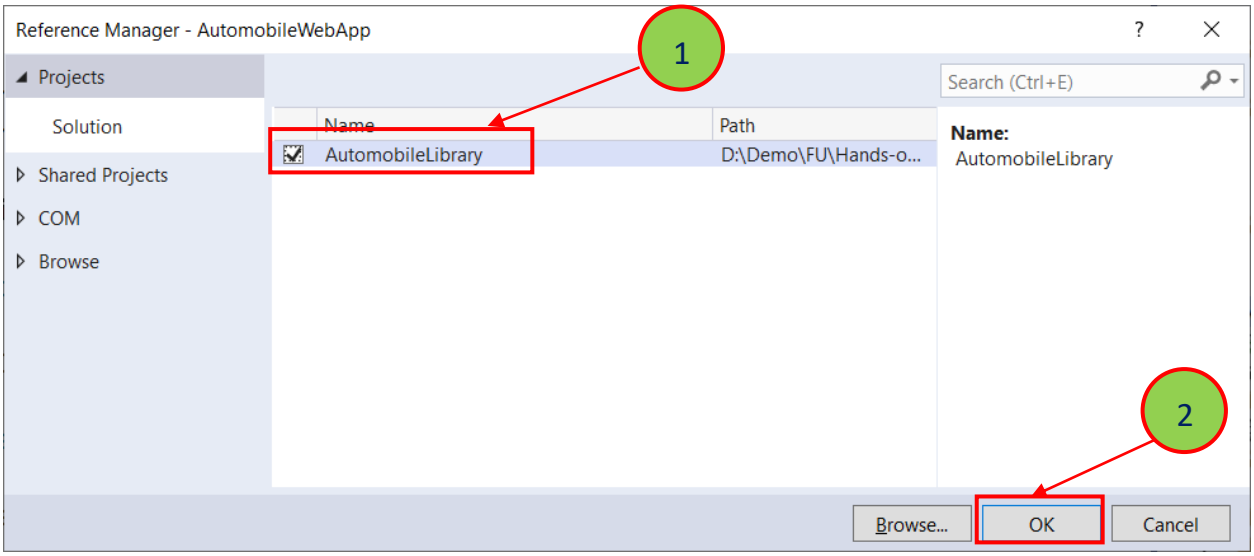
```
using System.Collections;
using AutomobileLibrary.DataAccess;

namespace AutomobileLibrary.Repository{
    public interface ICarRepository{
        IEnumerable<Car> GetCars();
        Car GetCarByID(int carId);
        void InsertCar(Car car);
        void DeleteCar(int carId);
        void UpdateCar(Car car);
    }
}
```

Step 08. Write codes for **CarRepository.cs** as follows:

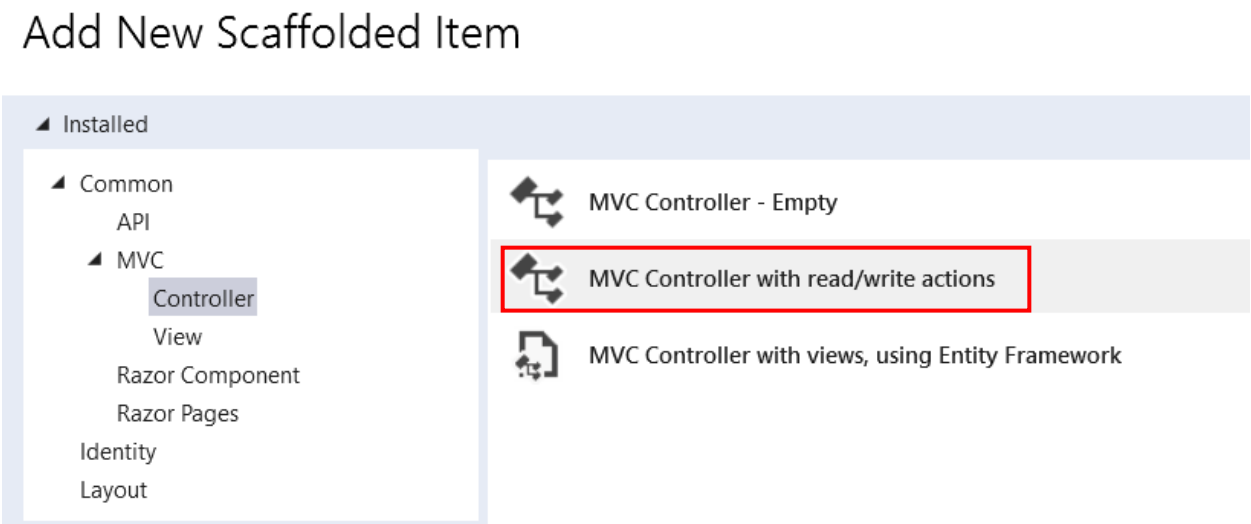
```
using AutomobileLibrary.DataAccess;
namespace AutomobileLibrary.Repository{
    public class CarRepository : ICarRepository {
        public Car GetCarByID(int carId) => CarDAO.Instance.GetCarByID(carId);
        public IEnumerable<Car> GetCars() => CarDAO.Instance.GetCarList();
        public void InsertCar(Car car) => CarDAO.Instance.AddNew(car);
        public void DeleteCar(int carId) => CarDAO.Instance.Remove(carId);
        public void UpdateCar(Car car) => CarDAO.Instance.Update(car);
    }
}
```

Activity 03: Reference to AutomobileLibrary project and write codes for ASP.NET Core MVC project

Step 01. Right-click on **AutomobileWebApp** project, select Add | Project Reference, and perform as the below figure:



Step 02. Right-click on the **AutomobileWebApp** project, select Add | Controller and perform as the below figure then click **Add.**



On the next dialog, enter the controller name is **CarsController.cs**, click **Add** to finish.

Step 03. Write codes for action methods of **CarsController.cs** as follows:

```
using AutomobileLibrary.Repository;
using AutomobileLibrary.DataAccess;

public class CarsController : Controller{
    ICarRepository carRepository = null;
    public CarsController()  =>  carRepository = new CarRepository();
    // GET: CarsController
    public ActionResult Index(){
        var carList = carRepository.GetCars();
        return View(carList);
    }
    // GET: CarsController/Details/5
    public ActionResult Details(int? id){
        if (id == null) {
            return NotFound();
        }
        var car = carRepository.GetCarByID(id.Value);
        if (car == null) {
            return NotFound();
        }
        return View(car);
    }
    // GET: CarsController/Create
    public ActionResult Create() => View();

    // POST: CarsController/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(Car car){
        try {
            if (ModelState.IsValid){
                carRepository.InsertCar(car);
            }
            return RedirectToAction(nameof(Index));
        }
        catch(Exception ex){
            ViewBag.Message = ex.Message;
            return View(car);
        }
    }
```

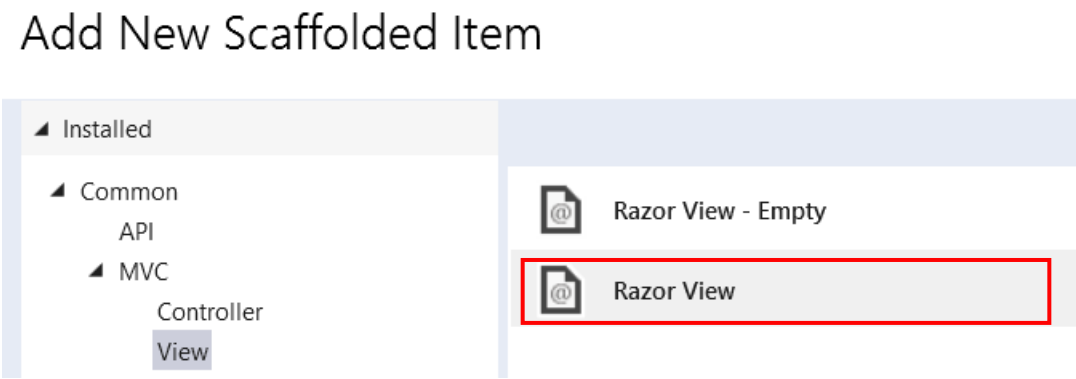
```
// GET: CarsController/Edit/5
public ActionResult Edit(int? id){
    if (id == null){
        return NotFound();
    }
    var car = carRepository.GetCarByID(id.Value);
    if (car == null){
        return NotFound();
    }
    return View(car);
}

// POST: CarsController/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(int id, Car car){
    try {
        if (id != car.CarId) {
            return NotFound();
        }
        if (ModelState.IsValid){
            carRepository.UpdateCar(car);
        }
        return RedirectToAction(nameof(Index));
    }
    catch(Exception ex){
        ViewBag.Message = ex.Message;
        return View();
    }
}

// GET: CarsController/Delete/5
public ActionResult Delete(int? id){
    if (id == null){
        return NotFound();
    }
    var car = carRepository.GetCarByID(id.Value);
    if (car == null){
        return NotFound();
    }
    return View(car);
}
```

```
    // POST: CarsController/Delete/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Delete(int id){
        try {
            carRepository.DeleteCar(id);
            return RedirectToAction(nameof(Index));
        }
        catch (Exception ex)
        {
            ViewBag.Message = ex.Message;
            return View();
        }
    }
}//end class
```

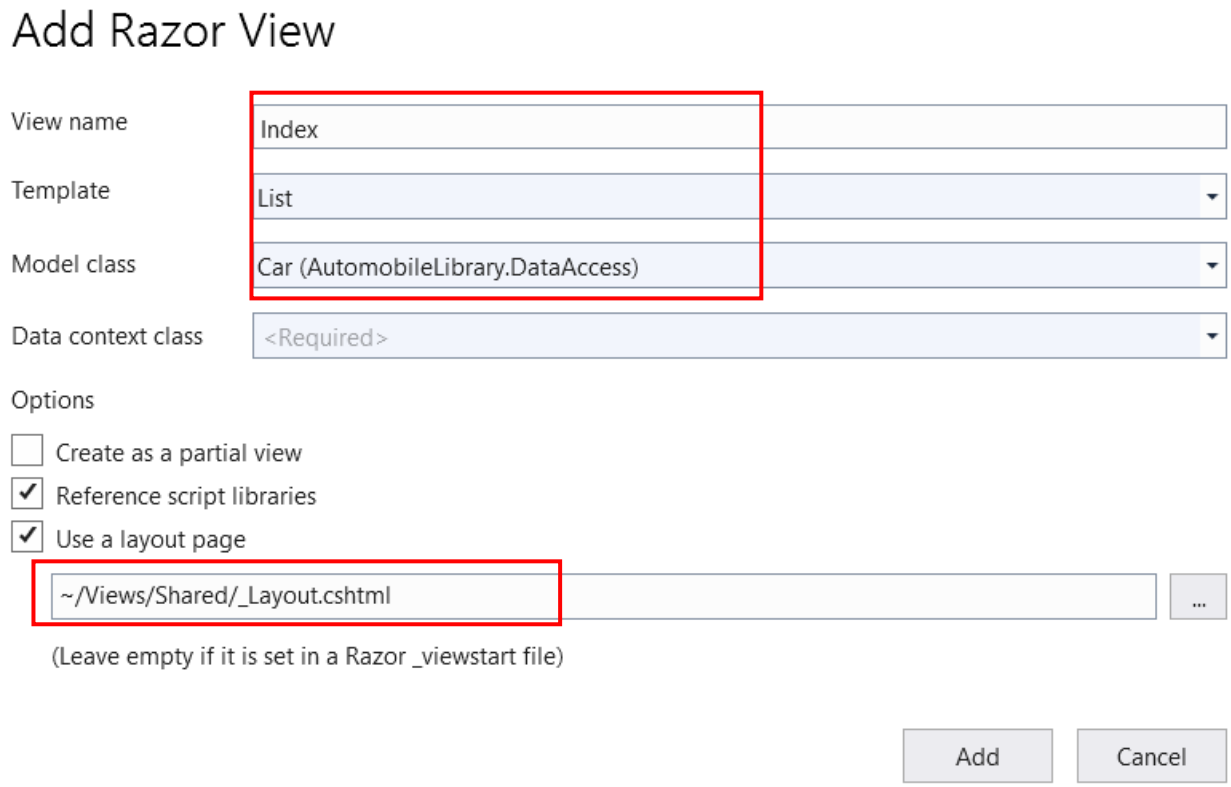
Activity 04: Create views AutomobileWebApp project

Step 01. Right-click on **View** folder | Add | New Folder named **Cars** then right-click on **Cars** folder | Add | View named **Index** as follows:

Step 02. Right-click on **Cars** folder | Add | View named **Index** as follows then click **Add.**



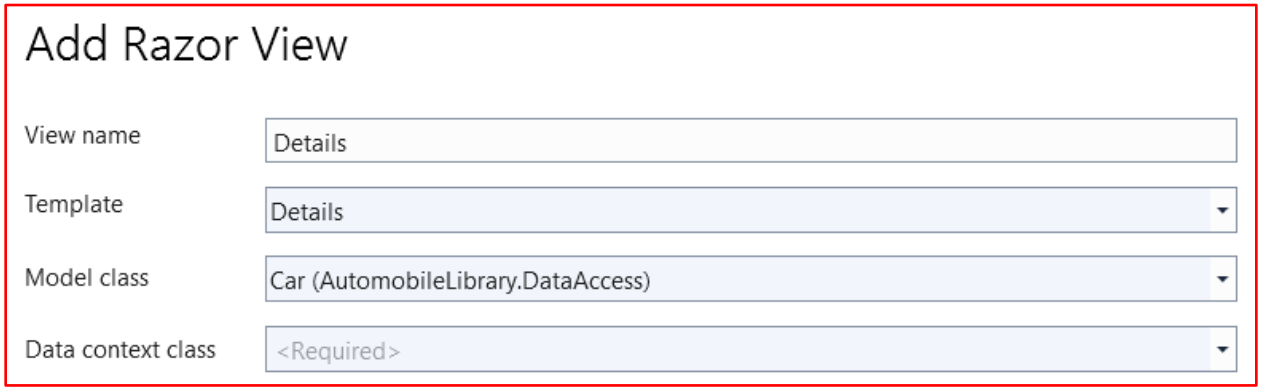
On the next dialog, setup as the below figure then click **Add** to finish.



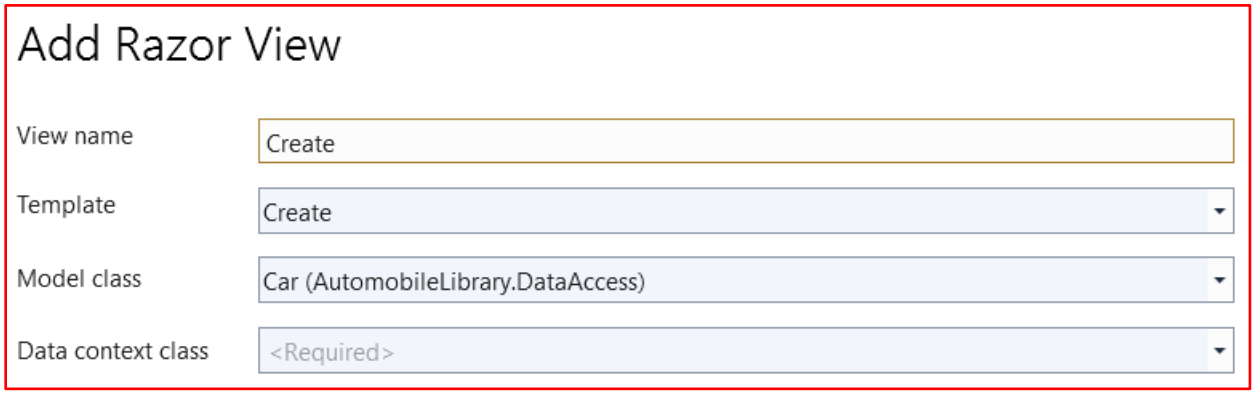
Step 03. Open **Index.cshtm** and update codes for table tag as follows:

```
<table class="table">
    <thead>
        <tr>…</tr>
    </thead>
    <tbody>
    @foreach (var item in Model) {
        <tr>
            <td>…</td>
            <td>…</td>
            <td>…</td>
            <td>…</td>
            <td>…</td>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id=item.CarId }) |
                @Html.ActionLink("Details", "Details", new { id = item.CarId }) |
                @Html.ActionLink("Delete", "Delete", new { id = item.CarId })
            </td>
        </tr>
}
    </tbody>
</table>
```

Step 04. Add **Details** view as the below figure:

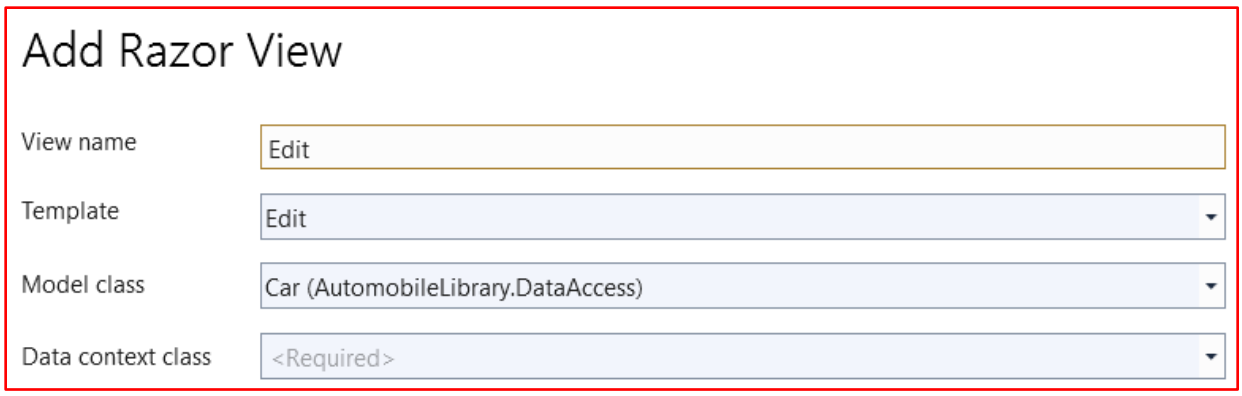


Step 05. Add **Create** view then open this view and update as follows:

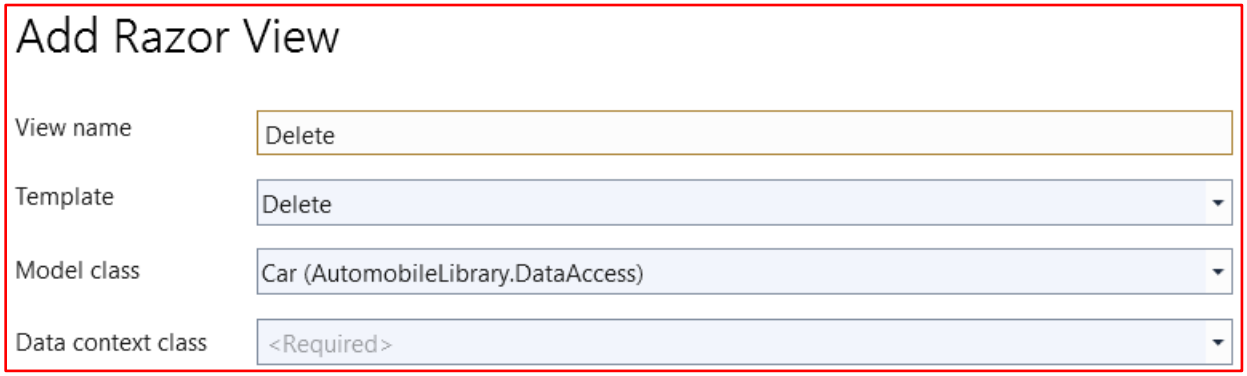



```
@model AutomobileLibrary.DataAccess.Car
@{
    ViewData["Title"] = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h1>Add New Car</h1>
<h4>@ViewBag.Message</h4>
<hr />
<div class="row">…</div>

<div>…</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Step 06. Add **Edit** view as the below figure:



Step 07. Add **Delete** view as the below figure:



Activity 05: Run AutomobileWebApp project and test all actions

Step 01. Press Ctrl+F5 to run the project then edit the link on the web browser as follows:

http://localhost:27136/Cars/Index

The result will show in the below figure:

localhost:27136/Cars/Index

AutomobileWebApp Home Privacy

Car List

Create New

CarId	CarName	Manufacturer	Price	ReleasedYear	
1	Civic 2.0	Honda	34000.00	2021	Edit | Details | Delete
2	BMW 523i	BMW	100000.00	2021	Edit | Details | Delete
3	Clarity	Honda Motor Company	33400.00	2021	Edit | Details | Delete
4	BMW 8 Series	BMW	85000.00	2021	Edit | Details | Delete
5	Audi A8	Audi	16000.00	2020	Edit | Details | Delete

Step 02. Click **Edit** link and display the result as the below figure, enter the values on text fields then click **Save** to update the car.

Update Car

CarId

1

CarName

Civic 2.0

Manufacturer

Honda

Price

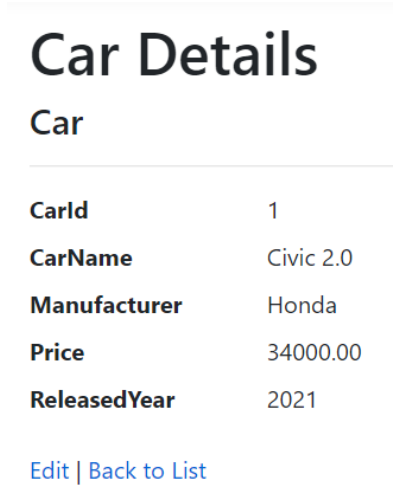
34000.00

ReleasedYear

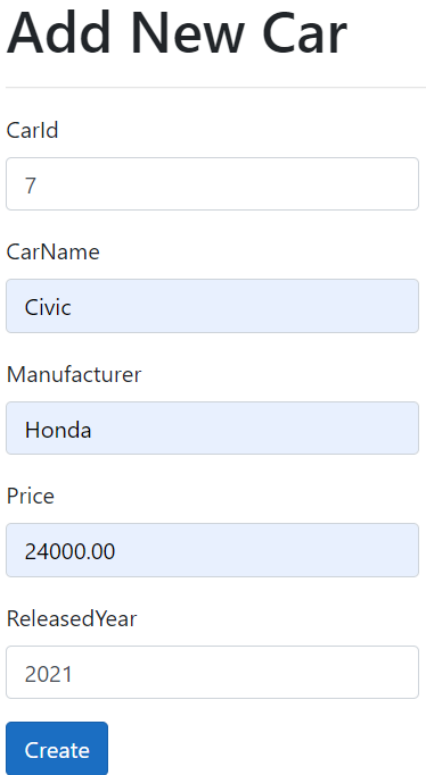
2021

Save

Step 03. Click **Details** link to view Car details.



Step 04. Click **Create New** link and display the result as the below figure, enter the values on text fields then click **Save** to add a new car.



Step 05. Click **Delete** link and display the result as the below figure. Click Delete button to remove Car

Delete

Are you sure you want to delete this?

Car

CarId	5
CarName	Audi A8
Manufacturer	Audi
Price	16000.00
ReleasedYear	2020

Delete | Back to List