LAB 2. BUILDING BASIC WEB APPLICATIONS WITH ASP.NET MVC 5 (1)

1. Introduction

This lab is the second in a series of practical exercises on ASP.NET MVC. It focuses on familiarizing yourself with and building **Model** classes and then develop the **Controller** layer to handle user requests. Students will learn how to define data objects and create simulated data to work with.

2. Objective

Upon completion of this lab, you will be able to:

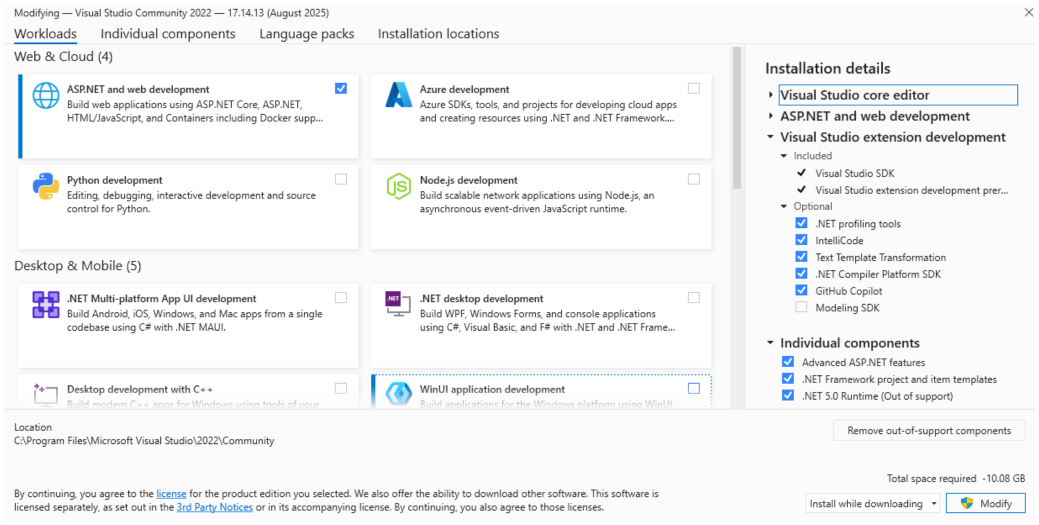
- Build Model classes to represent data objects in an application.
- Create fake data when a real database is not available.
- Create a new Controller to handle user requests.
- Use the Controller to interact with the Model.
- Create and configure action methods to return data.
- Set up the default routing for the application.
- Build basic CRUD functionalities (Create, Read, Update, Delete).

3. Project setup

a. Visual Studio 2022 Installation

- Visit the official Microsoft website to download the Visual Studio 2022 installer.
- Open the installer and select the ASP.NET and web development workload.

- Click Install and wait for the download and installation to complete.


Modifying — Visual Studio Community 2022 — 17.14.13 (August 2025)

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

- ASP.NET and web development ✓ — Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- Azure development — Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....
- Python development — Editing, debugging, interactive development and source control for Python.
- Node.js development — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- .NET Multi-platform App UI development — Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.
- .NET desktop development — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- Desktop development with C++
- WinUI application development

Installation details

- Visual Studio core editor
- ASP.NET and web development
- Visual Studio extension development
  - Included
    - ✓ Visual Studio SDK
    - ✓ Visual Studio extension development prer...
  - Optional
    - ✓ .NET profiling tools
    - ✓ IntelliCode
    - ✓ Text Template Transformation
    - ✓ .NET Compiler Platform SDK
    - ✓ GitHub Copilot
    - ☐ Modeling SDK
- Individual components
  - ✓ Advanced ASP.NET features
  - ✓ .NET Framework project and item templates
  - ✓ .NET 5.0 Runtime (Out of support)

Location
C:\Program Files\Microsoft Visual Studio\2022\Community

Remove out-of-support components

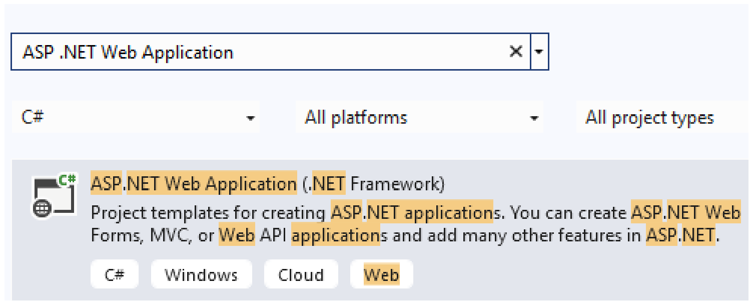
Total space required -10.08 GB

By continuing, you agree to the license for the product edition you selected. We also offer the ability to download other software. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Install while downloading | Modify

b. Project Initialization

- **Open Visual Studio 2022** and select **Create a new project**.
- Search for and select the **ASP.NET Web Application (.NET Framework)** template.

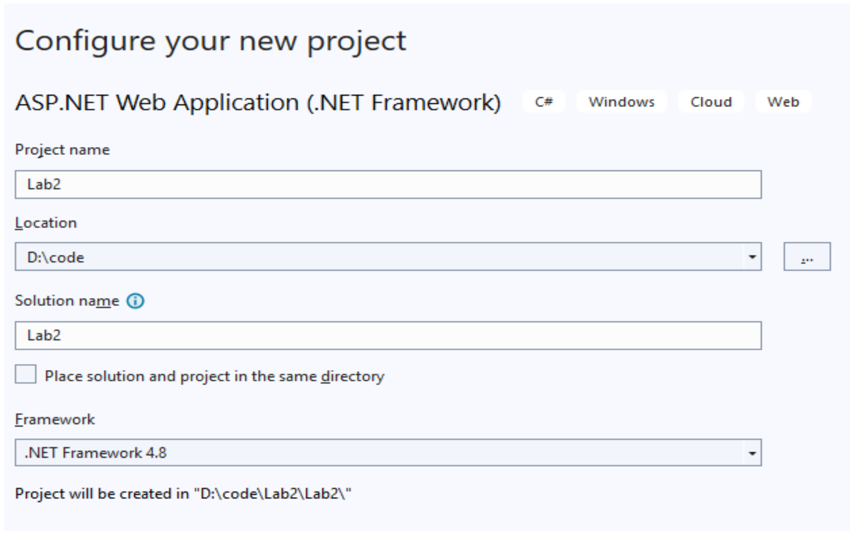

ASP .NET Web Application

C# | All platforms | All project types

ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

C# | Windows | Cloud | Web

- Name the project and choose a location to save it.


Configure your new project

ASP.NET Web Application (.NET Framework) | C# | Windows | Cloud | Web

Project name: Lab2

Location: D:\code

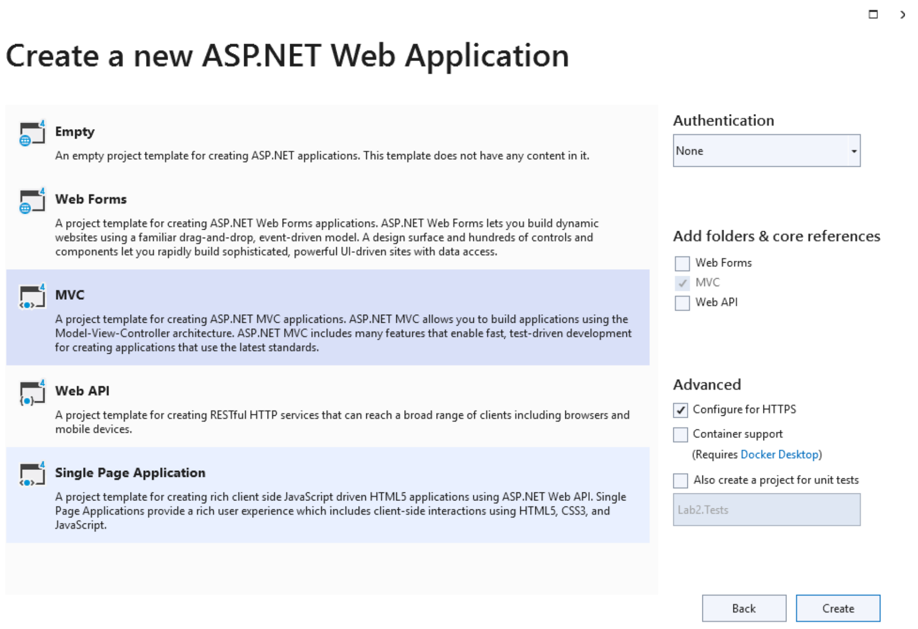
Solution name: Lab2

☐ Place solution and project in the same directory

Framework: .NET Framework 4.8

Project will be created in "D:\code\Lab2\Lab2\"

- In the next window, select the **MVC** template and click **Create**.



Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
None

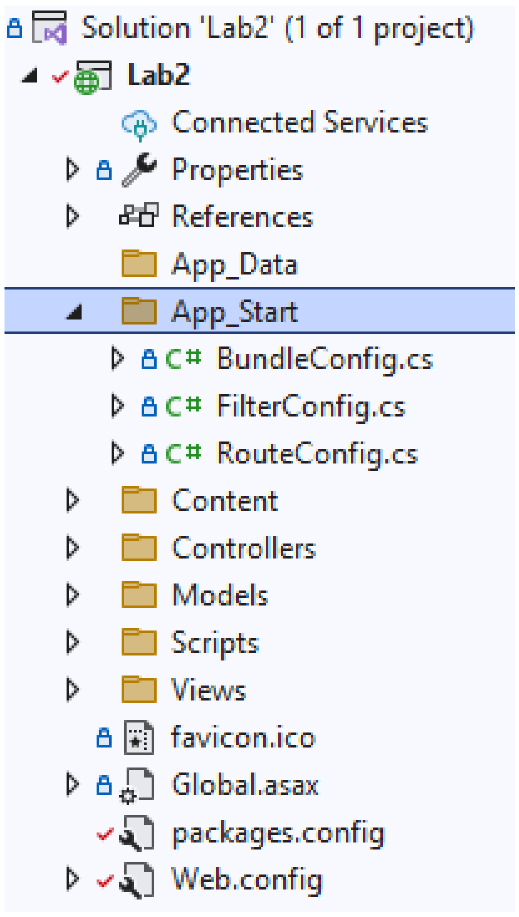
Add folders & core references
- [ ] Web Forms
- [✓] MVC
- [ ] Web API

Advanced
- [✓] Configure for HTTPS
- [ ] Container support
(Requires Docker Desktop)
- [ ] Also create a project for unit tests

Lab2.Tests

4. Complete the following requirements

a. Familiarize yourself with the project structure



```
Solution 'Lab2' (1 of 1 project)
  Lab2
    Connected Services
    Properties
    References
    App_Data
    App_Start
      C# BundleConfig.cs
      C# FilterConfig.cs
      C# RouteConfig.cs
    Content
    Controllers
    Models
    Scripts
    Views
    favicon.ico
    Global.asax
    packages.config
    Web.config
```

An ASP.NET MVC project has a clear folder structure that separates the roles in the MVC pattern.

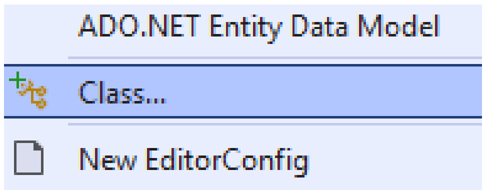
- **App_Data** folder: Contains application data files, such as database files (.mdf) or XML files.
- **App_Start** folder: Holds classes for configuring the application at startup.
- **Content** folder: Stores static files like CSS, images, and fonts.
- **Controllers** folder: Contains all the controllers, which are responsible for handling HTTP requests.
- **Models** folder: This folder contains the model classes that represent data objects and business logic.
- **Scripts** folder: Stores your JavaScript files.
- **Views** folder: Where all the views (.cshtml files) are stored. Child folders often correspond to controllers (e.g., Views/Home), and Views/Shared contains common views.
- **Solution**: This is a logical container in Visual Studio that holds one or more projects. When you create a new project, Visual Studio automatically creates a Solution to manage it.
- **Web.config**: This is a crucial configuration file that contains global settings for your web application, such as database connection strings, framework customizations, and security settings.
- **RouteConfig.cs**: This file is located in the App_Start folder and is responsible for defining the routing rules for the application. It maps URLs to specific action methods in a Controller.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;
```

```
namespace Lab2
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapMvcAttributeRoutes();
            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Product", action = "Index", id =
UrlParameter.Optional }
            );
        }
    }
}
```

b. Build the Model and Fake Data

- **Create the Product class**:
    - In **Solution Explorer**, right-click the **Models** folder and select **Add** -> **Class...**.



    - Name the file **Product.cs**.
    - Add the following properties to the Product class: Id, Name, and Price.

```
public class Product
{
   public int Id { get; set; }
   public string Name { get; set; }
   public decimal Price { get; set; }
}
```

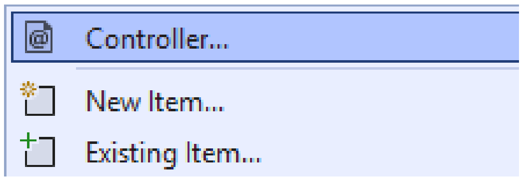
- Create the FakeDatabase class:

- In the same **Product.cs** file or a new file, create a static class named **FakeDatabase**.
- Create a static Products property of type List<Product>.
- Initialize the fake data for the product list inside the static constructor of the FakeDatabase class.

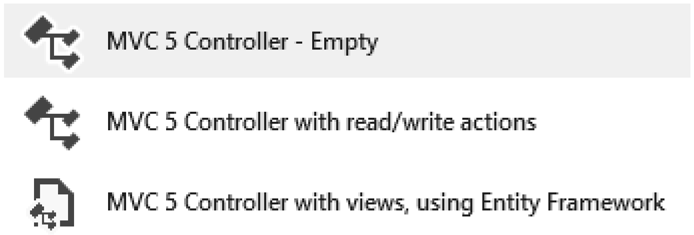
```
public static class FakeDatabase
{
   public static List<Product> Products { get; set; }

   static FakeDatabase()
   {
      Products = new List<Product>
      {
         new Product { Id = 1, Name = "Laptop HP Envy", Price = 28500000 },
         new Product { Id = 2, Name = "Apple Macbook Pro 14 inch", Price = 42000000
},
         new Product { Id = 3, Name = "Màn hình Dell Ultrasharp", Price = 8900000 },
         new Product { Id = 4, Name = "Bàn phím cơ Logitech G Pro", Price = 3800000 },
         new Product { Id = 5, Name = "Chuột không dây Logitech MX Master 3S", Price
= 2650000 },
         new Product { Id = 6, Name = "Tai nghe Sony WH-1000XM5", Price = 6990000
},
        new Product { Id = 7, Name = "Ổ cứng SSD Samsung 1TB", Price = 1750000 },
        new Product { Id = 8, Name = "Webcam Logitech C922", Price = 2100000 },
        new Product { Id = 9, Name = "Loa Bluetooth JBL Flip 6", Price = 2300000 },
        new Product { Id = 10, Name = "Router Wifi Asus RT-AX86U", Price = 5800000
}
      };
   }
}
```

c. Build the Controller and CRUD Functionality

- In **Solution Explorer**, right-click the **Controllers** folder.



- Select **Add** => **Controller....**
- Choose **MVC 5 Controller - Empty** and click **Add**.


MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework

- Name the controller **ProductController.cs**.
- Click **Add**. A new ProductController.cs file will be created.

d. Build the Read functionality (Listing Products)

- Open the **ProductController.cs file**.
- Add a **using** statement at the top of the file to access your model classes:

```
using Lab2.Models;
```

Inside the **ProductController** class, create an **Index** action method. This method will fetch the product data and prepare it for the view.

```
public ActionResult Index()
{
    var products = FakeDatabase.Products;
    return View(products);
}
```

- **Note:** You will create the corresponding **View** for this action in **Lab 3**.

e. Configure the Default Route

- The default route determines which controller and action method are executed when a user visits the root of your website (e.g., http://localhost:5000).
- In the **Solution Explorer**, open the **App_Start** folder and then open **RouteConfig.cs**.
- Locate the **routes.MapRoute** method. The default route is typically configured as follows:

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

- To make your ProductController the default landing page, change the controller default value to "**Product**":

```
defaults: new { controller = "Product", action = "Index", id = UrlParameter.Optional }
```

f. Build the CRUD functionalities

Now, let's create the action methods to handle the **Create**, **Update**, **Delete** operations.

- Open ProductController.cs.
- Add the following action methods to handle the **Create** functionality:

```
[HttpGet]
public ActionResult Create()
{
    return View();
}
[HttpPost]
public ActionResult Create(Product product)
{
    FakeDatabase.Products.Add(product);
    return RedirectToAction("Index");
}
```

- Add the following action methods to handle the **Update** functionality:

```
[HttpGet]
public ActionResult Update(int id)
{
    var product = FakeDatabase.Products.FirstOrDefault(p => p.Id == id);
    return View(product);
}

[HttpPost]
public ActionResult Update(Product product)
{
    var existingProduct = FakeDatabase.Products.FirstOrDefault(p => p.Id == product.Id);
    if (existingProduct != null)
    {
        existingProduct.Name = product.Name;
        existingProduct.Price = product.Price;
    }
    return RedirectToAction("Index");
}
```

- Add the following action method to handle the **Delete** functionality:

```
public ActionResult Delete(int id)
{
   var product = FakeDatabase.Products.FirstOrDefault(p => p.Id == id);
   if (product != null)
   {
      FakeDatabase.Products.Remove(product);
   }
   return RedirectToAction("Index");
}
```

5. Summary

- In this lab, you became familiar with the basic structure of an ASP.NET MVC project and learned how to create Model classes to define data. You also learned how to create simulated data. Furthermore, you learned how to create a controller and build the core CRUD (Create, Read, Update, Delete) functionalities.
- By defining action methods and interacting with the FakeDatabase model, you have laid the foundation for a fully functional web application. In the next lab, you will learn how to create the corresponding Views to display the data and interact with these controller actions.

6. Exercises

a. Exercise 1: Category Management

- Create a **Category** class with the following properties: Id, Name, Description.
- Add a List<Category> named Categories inside FakeDatabase.
- Implement a CategoryController with the following actions:
    - Index() → Return all categories.
    - Details(int id) → Return details of a category by Id.
    - Create(Category category) → Add a new category.

- Edit(int id, Category updatedCategory) → Update an existing category.
- Delete(int id) → Remove a category.

b. Exercise 2: Order Management

- Create an **Order** class with the following properties: Id, CustomerName, OrderDate, ProductIds.

- Add a List<Order> named Orders inside FakeDatabase.

- Implement an OrderController with the following actions:

  - Index() → Return all orders.
  - Details(int id) → Return order details including the product IDs.
  - Create(Order order) → Add a new order (you can assume ProductIds are passed in).
  - Edit(int id, Order updatedOrder) → Update order information.
  - Delete(int id) → Remove an order.