Lab Manual for Cloud Computing

Lab No. 3

Implementing Microsoft core application using ASP.Net MVC Frame work

LAB 03: IMPLEMENTING MICROSOFT CORE APPLICATION USING ASP.NET MVC FRAME WORK

1. INTRODUCTION:

What is .NET Core?

.NET Core is a general-purpose, modular, cross-platform and open-source implementation of the .NET Platform. It contains many of the same APIs as the .NET Framework (but .NET Core is a smaller set) and includes runtime, framework, compiler and tools components that support a variety of operating systems and chip targets.

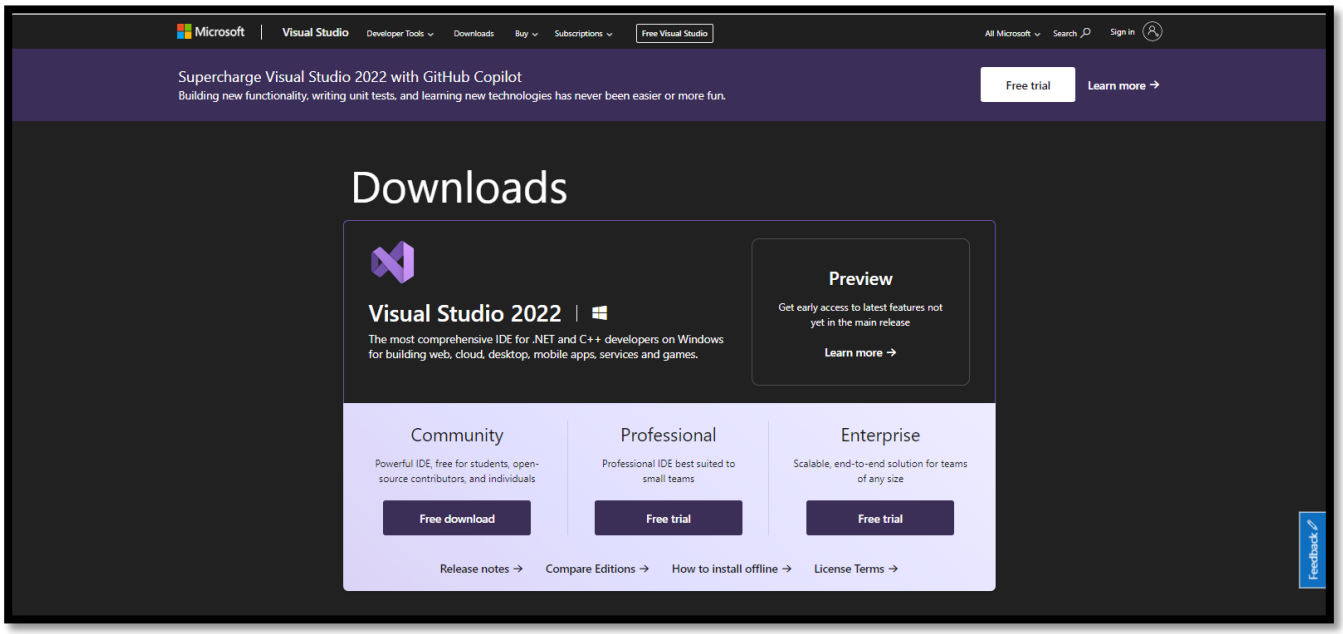
The .NET Core implementation was primarily driven by the ASP.NET Core workloads but also by the need and desire to have a more modern runtime. It can be used in the device, cloud, and embedded/IoT scenarios.

.NET Core is highly relevant to cloud computing as it is a cross-platform, open-source framework that supports microservices architecture, containerization, and seamless integration with cloud services. Its cross-platform compatibility allows developers to deploy applications across various operating systems, essential for the diverse cloud environments. .NET Core's support for microservices enables the creation of modular and scalable cloud-native applications. Additionally, its containerization capabilities, especially with Docker, align well with cloud practices, providing consistency and ease of deployment. The framework's seamless integration with cloud services and APIs facilitates efficient utilization of cloud features. Overall, .NET Core is well-suited for developing and deploying applications in the dynamic and scalable landscape of cloud computing.

Tools Required:

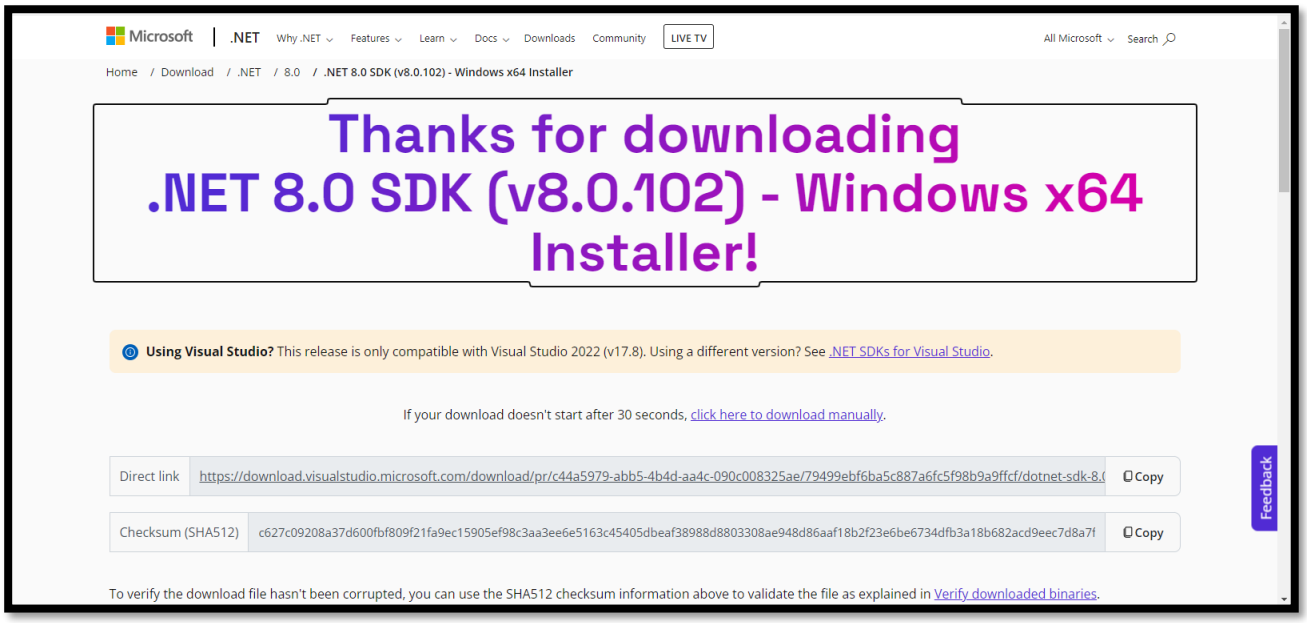
1. **Visual studio 2022:**

URL for downloading Visual studio 2022: https://visualstudio.microsoft.com/downloads/



2. .NET 8.0 SDK (v8.0.102)

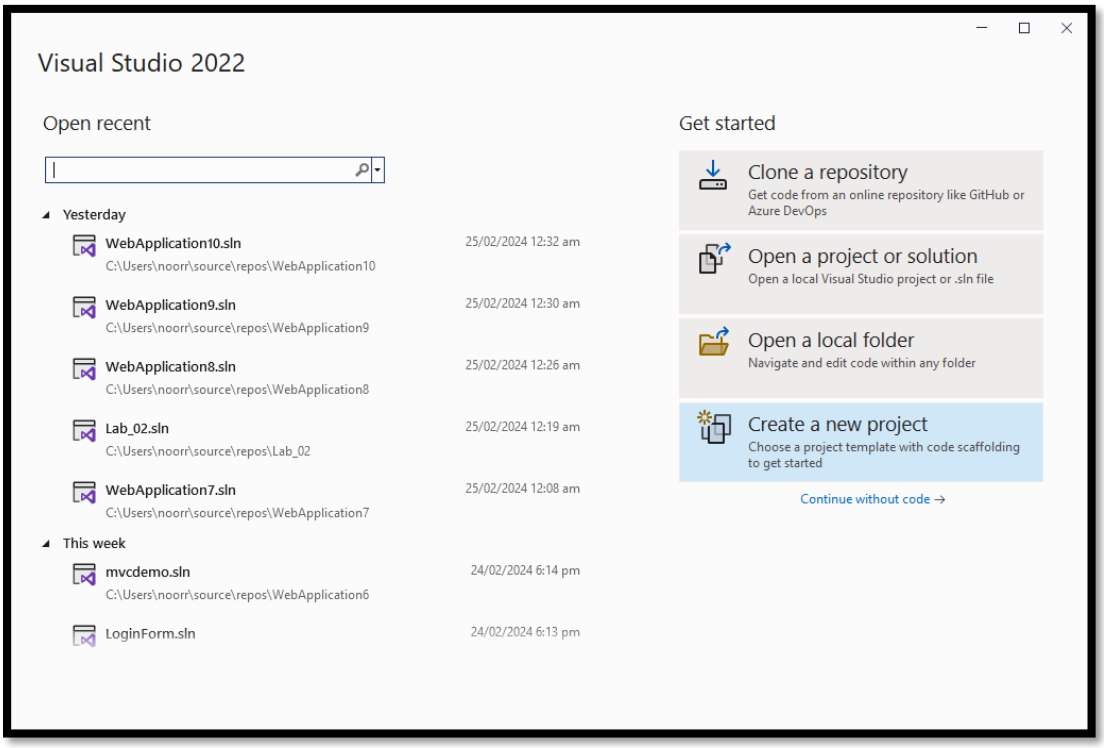
URL for downloading: https://dotnet.microsoft.com/en-us/download/dotnet/thank-you/sdk-8.0.102windows-x64-installer



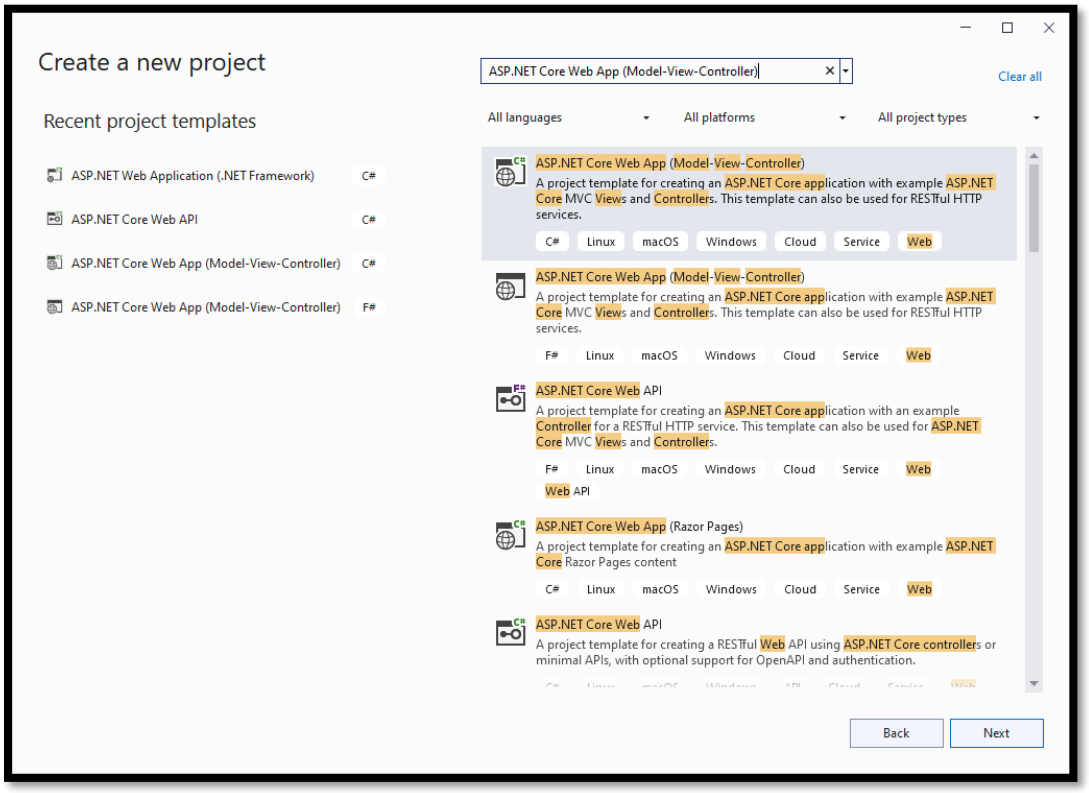
Getting Started Creating .Net Core web application

1. Open visual studio 2022 IDE.

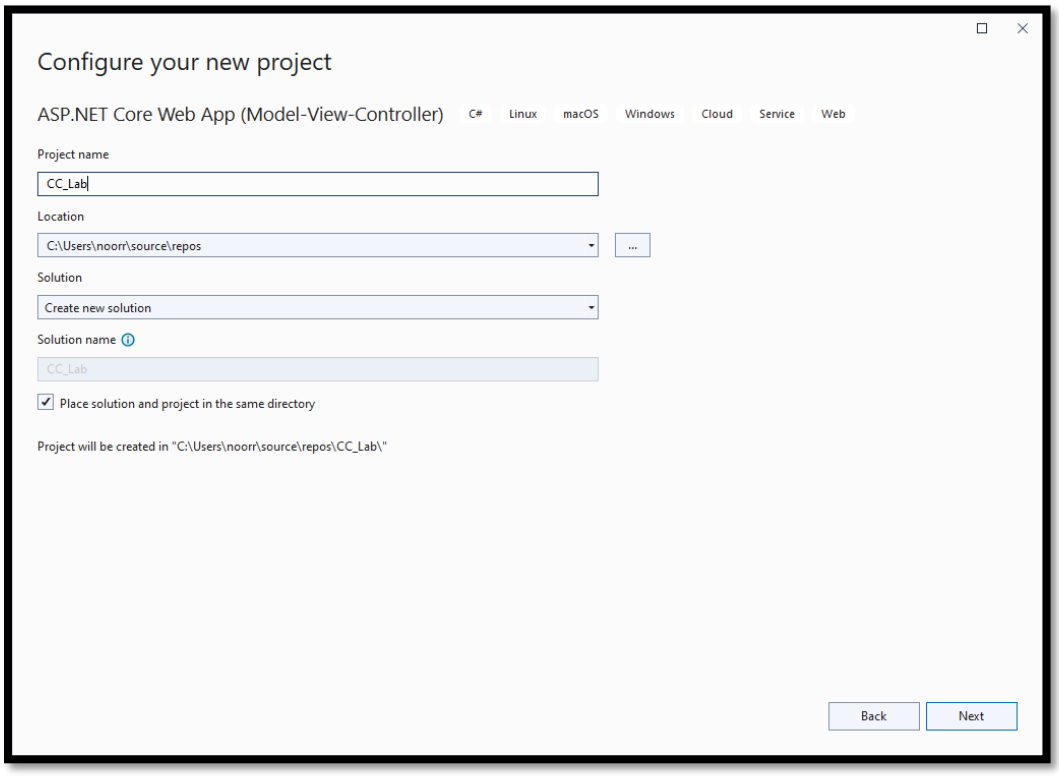
2. Click on "Create a new project" in the start window.



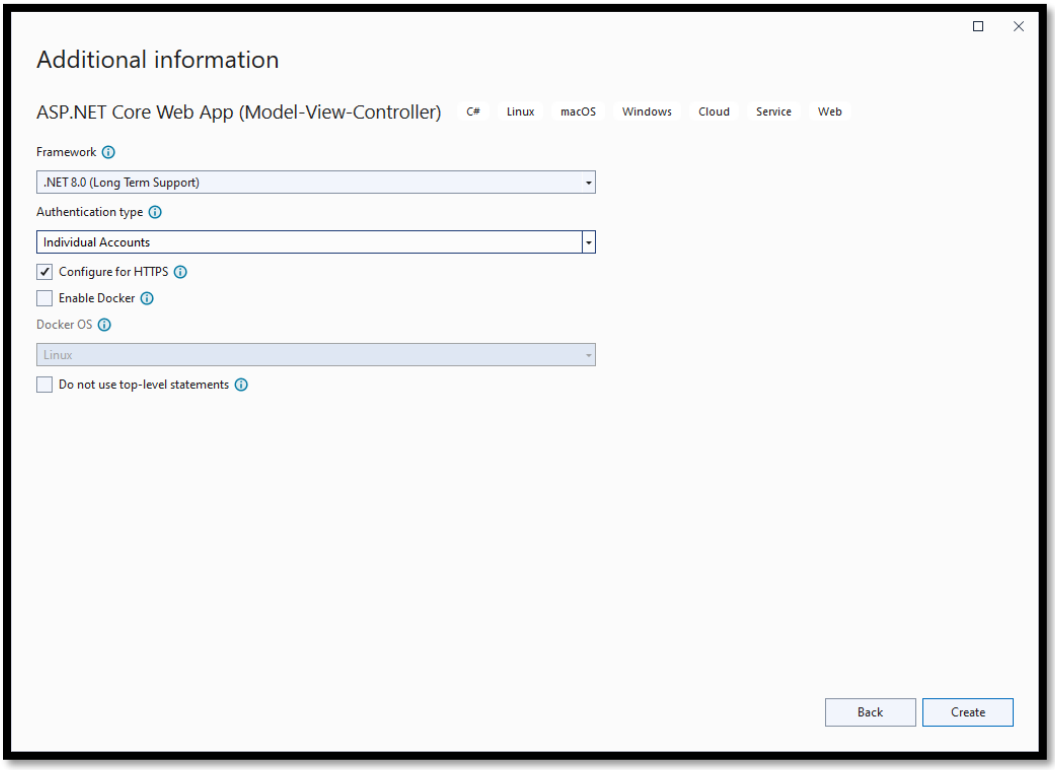
3. In the "Create a new project" window, search for " ASP.NET Core Web App (Model-View-Controller)" in the search bar, and then click NEXT.



4. Provide a name and location for your project and then Click NEXT.

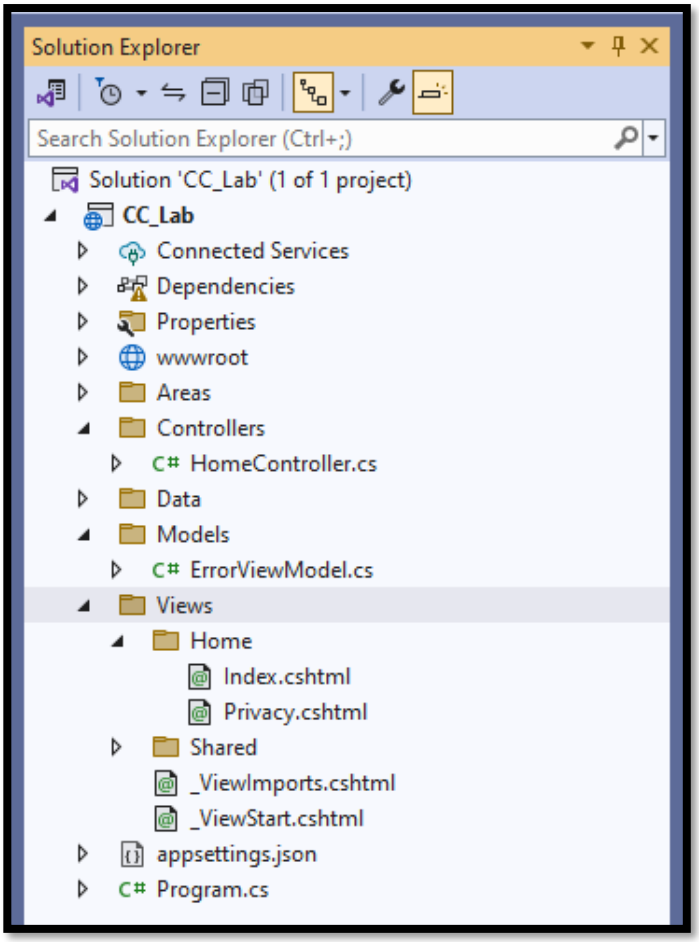


5. Select the desired framework version (.NET Core) and authentication type, and Click CREATE.



Let's have a look at Project structure which is created.

When you create an ASP.NET Core Web App using the "Model-View-Controller (MVC)" template in Visual Studio 2022, the project structure is organized to follow the MVC architectural pattern. Below is an overview of the typical project structure for such an application:



- **Controllers:** The Controllers folder contains the controllers of your application. Controllers handle incoming HTTP requests, process the input, and return an appropriate response. By default, there is a HomeController to handle the main page.

- **Views:** The Views folder contains the visual components of your application. Each controller has its corresponding folder inside Views, and within each controller's folder, there are views (.cshtml files) that define the HTML markup. Views are responsible for presenting data to the user.

- **Models:** The Models folder is where you define the data models for your application. Models represent the entities or data structures that the application works with. These models are often used to interact with a database or other data sources.

- **wwwroot:** The wwwroot folder is the root of your web application and contains static files like stylesheets, JavaScript files, images, and other assets. Files placed here are served directly to clients without going through the MVC routing.

- **appsettings.json:** This JSON configuration file holds various settings for your application, such as database connection strings, API keys, and other configuration parameters. It's used to configure the behavior of your application.

- **Program.cs:** The Program.cs file is the entry point of the application. It contains the Main method, which sets up the web host and starts the application.

- **Dependencies:** The Dependencies node in the solution explorer shows the NuGet packages and dependencies used in your project.

- **Properties:** The Properties folder may contain files like launchSettings.json, which configures how the application is launched and debugged.

6. Now just save the application and run.

Let's create the simple login age using .NET Core Web Application (MVC):

Below is a simple step-by-step guide to creating an ASP.NET Core MVC web application with a login form:

1. Model:

Right-click on the "Models" folder in your project and add a new class, for example, **LoginViewModel.cs**. Define properties for username and password in the class.

```
public class LoginViewModel
   {
       public string Username { get; set; }
       public string Password { get; set; }
   }
```

2. Controller:

Right-click on the "Controllers" folder and add a new controller, for example, **AccountController.cs**. In the controller, add actions for handling login.

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }


        [HttpPost]
        public IActionResult Login(LoginViewModel model)
        {
            // In a real application, you would validate the username and password against a
               database or another source.
            if (model.Username == "user" && model.Password == "password")
            {
                // Successful login
                return RedirectToAction("Welcome", new { username = model.Username });
            }
            // Failed login
            ViewBag.ErrorMessage = "Invalid username or password";
            return View("Index");
        }


        public IActionResult Welcome(string username)
        {
            ViewBag.Username = username;
            return View();
        }

    }
```

3. Views:

Right-click on the "Home" folder present in the Views folder and then add a new view named **Index.cshtml** and **Welcome.cshtml**. Add the following code.

```
@model WebApplication1.Models.ViewsModel.LoginViewModel

@{
    ViewData["Title"] = "Login Page";
}
<h2>Login</h2>

<form method="post" asp-action="Login">
    <div>
        <label asp-for="Username">Username:</label>
        <input asp-for="Username" required />
    </div>
    <div>
        <label asp-for="Password">Password:</label>
        <input asp-for="Password" type="password" required />
    </div>
    <button type="submit">Login</button>
</form>

@if (!string.IsNullOrEmpty(ViewBag.ErrorMessage))
{
    <p style="color: red;">@ViewBag.ErrorMessage</p>
}
```

```
@{
    ViewData["Title"] = "Welcome Page";
}
<h2>Welcome, @ViewBag.Username!</h2>
```

4. Save and then debug your application.

2. Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

3. Objectives

After completing this lab the student should be able to:

a. *Grasp the fundamentals of Model-View-Controller architecture in .NET Core Web Apps for creating modular and organized applications.*
b. *Learn to seamlessly integrate cloud computing principles into .NET Core MVC, emphasizing scalability and efficiency.*
c. *Apply knowledge to develop a basic .NET Core MVC web app, incorporating cloud services for hands-on experience in real-world scenarios.*

4. Lab Tasks/Practical Work

1. Create login and registration form using simple MVC net core.
2. Create feedback form using simple MVC net core.