**Create the ASP.NET Core Web App.**

When creating a new visual studio project, select ASP.NET Core Web App (Model-View-Controller) project template.

For now, set Authentication Type to none and uncheck everything else.

You can choose the latest .net core version but older ones should work as well.

Make sure that the template project runs fine. Index action of the Home Controller will run and produce a welcome view.

Include the following NuGet packages (only versions compatible with your .net core version will install):

Tools -> NuGet Package Manager -> Manage NuGet packages for Solution. Browse for and install the following packages.

* + Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore
  + Microsoft.EntityFrameworkCore.SqlServer

Create a model class Client and the database context class named ClientContext as was done when you followed the ASP.Net Core Web API tutorial in which Entity Framework was used (though only for in-memory database). Refer to the attached MVCSampleApp.zip for reference code. You will need to register the database context (in Program.cs) and provide value for the connection string (in appsettings.json). If you like the database to be created if it does not already exists then you need to specify that by calling dbcontext.Database.EnsureCreated(). Check out the attached MVC project (in particular the Program.cs and appsettings.json) as well as the following link on the above steps.

<https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/intro?view=aspnetcore-7.0>

Also check out the LaunchSettings.json in the properties folder of the project.

After the above steps, build the project and generate controller (and now also the views for the model class(es)) as was done for the Web API Tutorial as follows:

[right click on controllers folder] -> Add -> New Scaffolded Item -> MVC controller with Views using EntityFramework.

For model class choose Client and for db context class choose ClientContext

Note: There is some possibility that you may be asked to download yet another nuget package at this time to facilitate this auto-generation probably if you have older versions of .net core installed on your computer.

If you run the project and append “/Clients” to the url on the automatically launched browser instance you will see the Clients index view which would allow you to manage clients. Alternative, add “Clients” tab to the navigation bar by making changes to the \_layout.cshtml (see for reference the \_layout.cshtml in the attached sample project to make this addition to the navigation bar).

After you run the app, the database (if does not exist) will be recreated that you can check via Sql Server Management Studio. Please note that the database server and the database name are now different (checkout the connection string in the appsettings.json file).

**Creating Selenium based Automated UI Test**

Refer to the following link as a fall back.

<https://www.browserstack.com/guide/selenium-with-c-sharp-for-automated-test>

Right Click MVCSampleApp solution (not the project) ->Add New Project [select NUnit Test Project C#] as another project in the MVCSampleApp solution. You can name it whatever you like.

Add Selenium.WebDriver as well as Selenium.Support NuGet packages to the test project (both packages say “by Selenium” to be sure).

Type chrome::/version on the browser to check the chrome driver version. For browsers such as FireFox or Safari refer to the above link or other online tutorials which will go through similar steps.

Go to the following links to download the corresponding Chrome Driver version.

<https://googlechromelabs.github.io/chrome-for-testing/>

If your chrome is even older then can try finding it here (you can consider upgrading your browser to the latest version but at your own risk 😊 ):

<https://developer.chrome.com/docs/chromedriver/downloads>

create a “drivers” folder in the test project and add the chromedriver.exe (extracted from the downloaded zip file) into that folder.

UnitTest1.cs can look like as follows (url variable in the code should point to the url that shows up on the browser when you run MVCSampleApp in VisualStudio):

**using OpenQA.Selenium;**

**using OpenQA.Selenium.Chrome;**

**namespace TestProject1**

**{**

**public class Tests**

**{**

**[SetUp]**

**public void Setup()**

**{**

**}**

**[Test]**

**public void Test1()**

**{**

**string path = Directory.GetParent(Environment.CurrentDirectory).Parent.Parent.FullName;**

**ChromeDriver driver = new ChromeDriver(path + @"\drivers\");**

**string url = "http://localhost:5120";**

**//ChromeDriver driver = new ChromeDriver();**

**driver.Manage().Window.Maximize();**

**driver.Navigate().GoToUrl(url);**

**driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(5);**

**driver.FindElement(By.LinkText("Clients")).Click();**

**driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(5);**

**driver.FindElement(By.LinkText("Create New")).Click();**

**driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(5);**

**driver.FindElement(By.Id("Name")).SendKeys("Marge Simpson");**

**driver.FindElement(By.Id("Address")).SendKeys("Springfield");**

**driver.FindElement(By.XPath("//Input[@type='submit']")).Click();**

**Assert.Pass();**

**}**

**[TearDownAttribute]**

**public void TearDown()**

**{**

**}**

**}**

**}**

Click on the “Debug” tab on the top navigation bar of Visual Studio (it is between Build and Test tabs) and press Start without Debugging (make sure that there is no other browser window open with welcome page of your MVCSampleApp). Right click on the UnitTest1.cs and press run. You will see the above code automatically launching the Web App and creating a new client.