Module 03 Lab 01 Worksheet

Building an ASP.NET Core Web Application

# Overview

This lab takes our C# development to the next level. You will be creating a Web application using C# and ASP.NET and deploying it to the public Internet. By this time, you should have decided which type of application you will be creating for this course.

*It's important that you complete this lab before doing Lab 02.* You will be using the code you write in this lab for the next one.

|  |  |
| --- | --- |
| * Close any current projects in VS Code. * In the Explorer, click on Open Folder. * Create a new folder named firstInitialLastName\_webapp. For example, Edith Palka would create a folder called epalka\_webapp. * Click on Open. * In the terminal, create an empty MVC web application with the command:   dotnet new mvc  What is the name of your project? (5 pts.)  Program.cs | 5 |
| * Open your project folder in VS Code. * Agree to the prompts about restoring package dependencies and adding resources. * Open the Integrated Terminal and use the command dotnet run to compile and run your application. * Point your Web browser to <http://localhost:5000>.   Paste a screenshot of the resulting Web page (5 pts.) | 5 |
| Click on a link on the Web page and paste a screenshot to verify that the link is 'live'. (5 pts.) | 5 |
| In the terminal, enter <Ctrl>-C to stop the application.  Before we continue, however, let's put our project under *version control*.   * Click the Git icon in the VS Code Side Bar. * Click on initialize git repository.   How many changes are listed? (5 pts.) | 5 |
| * Enter your first commit message (Ex. First commit of my C# web project) * Click on the check mark or select Commit All from the menu.   How many changes are listed? (5 pts.) | 5 |
| Now that we've saved our project, let's take a look at the file structure. | 0 |
| You may have noticed a new file in this project - Startup.cs. This is where we put the setup code for your application. Anything that needs to happen before your application starts goes here.   * Open Startup.cs * Examine the Startup class definition.   How many methods are defined? (5 pts.)  Name the methods in Startup.cs. (5 pts.) | 10 |
| The Startup class *must* define a Configure() method, but the ConfigureServices() method is optional.  The Configure() method tells your application how it will respond to individual HTTP requests (ie, requests from a Web client).  How many arguments does the Configure() method take in our app? (5 pts.)  List the arguments. (5 pts.) | 10 |
| * IApplicationBuilder - This is required and is the interface that configures an application's request pipeline. So app.UseStaticFiles() adds support for static files. app.UseMvc() adds *routing* (this tells the app where to send requests) to the application and sets MVC as the default router. The arguments set the default route to a view named Index in a subfolder of Views called Home. * IHostingEnvironment - This is a service that lets us load in settings from our environment at startup. It also lets us support multiple environments (ex. Development vs. production) * ILoggerFactory - This lets us set up and add logging providers so our app can send us messages.   The UseStaticFiles() method lets our app serve files from the wwwroot folder in our project. We can put images, animations, videos, even HTML in here and our application can access it directly and display it in a browser.  The UseMvc (routes…) method takes a bit of explaining. A *route* is how we tell our application where to send user requests, based on the URL that is included in the request packet.  For example, if a user made a request to <http://mywebapp.example.com/images>, the application might display a gallery of image thumbnails. In this case, we're setting up a method that looks at the URL and determines which controller and action should be invoked. The *default route* will send the request to a controller named Home and have it display a page named Index. (We'll be creating those next.) | 0 |
| Examine the Controllers folder. What does it contain? (5 pts.) | 5 |
| Examine your Views folder. What does it contain? (5 pts.) | 5 |
| The .cshtml extension indicates that the file has both Web markup (HTML) and C# code.  Delete the following files from your project:  \_ViewImports.cshtml  \_ViewStart.cshtml  \_Layout.cshtml  Error.cshtml  Open Index.cshtml (in the Views/Home folder)  Replace the contents of Index.cshtml with  <h1>Hello World!</h1>  <p>The time is @DateTime.Now</p>  (Don't worry about what markup like <h1> and <p> mean for now.)  The Now property of DateTime returns the current time in localized format. Note that we can insert C# code in with HTML by prefacing it with @. We can insert a block of C# code with  @{  ….many lines of code…..  }  We can use this to create dynamic Web page content using C# and take advantage of the .NET and ASP.NET framework libraries.   * Save your changes, then compile and run your code with dotnet run. * Point your Web browser to <http://localhost:5000>.   Copy and paste a screenshot of the Web page. (5 pts.) | 5 |
| * Stop the application with <Ctrl>-C as you did before. * Before continuing, commit your changes to version control with the message   Modified default View Index.cshtml  Now that we have a very basic, but working, Web application, let's modify two more views:   * About (tells visitors about this application) * Contact (displays contact information for those with more questions)   Every other Web page has these, so why not us?   * Replace the contents of About.cshtml with:   <h1>This is an empty About page.</h1>  <p>I'll put more in later.</p>   * Replace the contents of Contact.cshtml with:   <h1>Here is my contact information</h1>  <p>Call me!</p> | 0 |
| Remember from earlier that we use routes to determine what content to serve to users. These routes are defined as methods in the Controller class. For our app, we'll find these in the class file HomeController.cs.   * Open HomeController.cs * Note the following lines in the HomeController class definition:   public IActionResult Index()  {  return View();  }  public IActionResult Contact()  {  return View();  }  public IActionResult About()  {  return View();  }  This tells our controller if we get a URL ending in /home/About, serve up About.cshtml.  Similarly, if the URL ends in /home/Contact, serve up Contact.cshtml.  The default view is Index.cshtml.  Run your app with dotnet run.  What do you see when you point your Web browser to (5 pts. each):  <http://localhost:5000>  <http://localhost:5000/home/About>  <http://localhost:5000/home/Contact> | 15 |
| Quit the server with <Ctrl>-C.  Commit your changes to version control with the message:  Added two more views, Contact and About  Your lab is complete. | 0 |
| **Total** |  |

Complete this worksheet and submit it to your instructor along with a ZIP of your project.