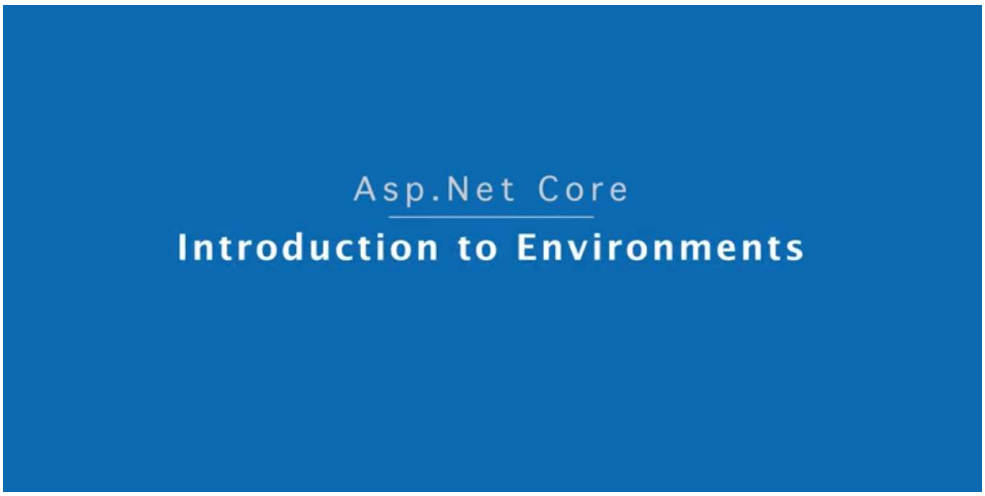
Section 13: Environments [MVC and Web API]

Monday, January 20, 2025 8:10 PM



An environment represents a system in which the application is currently running.
For example, today you have created an ASP.NET Core application, and the machine used for development is called the **development environment**.

While developing the application, the environment should be set to **development** so that you can get code-related error messages, such as details of exceptions.
But tomorrow, if you upload the same application to a server where other developers and quality controllers can access it, that environment should be set as the **staging environment**.

Later, when the same application is uploaded to the production server, which is the real server from which end users access the application, the exception details should be hidden.
The users should not see exception details or code-related errors. If they encounter runtime errors, they can report them to you through other means, such as email.

These are the three built-in environments supported by ASP.NET Core:

1. Development
2. Staging
3. Production

Alternatively, you can configure your own custom environments based on your needs.
For example, you can create environments like:

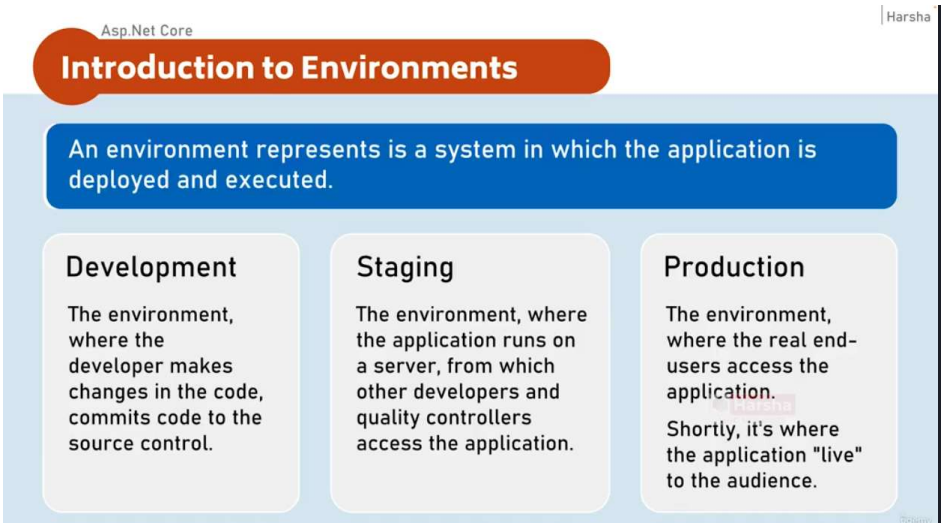
- Unit Testing Environment
- Integration Testing Environment
- Beta Environment

However, for most projects, companies, and teams, the three built-in environments are usually sufficient.
It is essential for every ASP.NET Core developer to know how to configure the environment name in ASP.NET Core.
You should be able to change the environment based on the current working machine:

- Development environment on the developer's machine
- Staging environment on the staging server
- Production environment on the production server

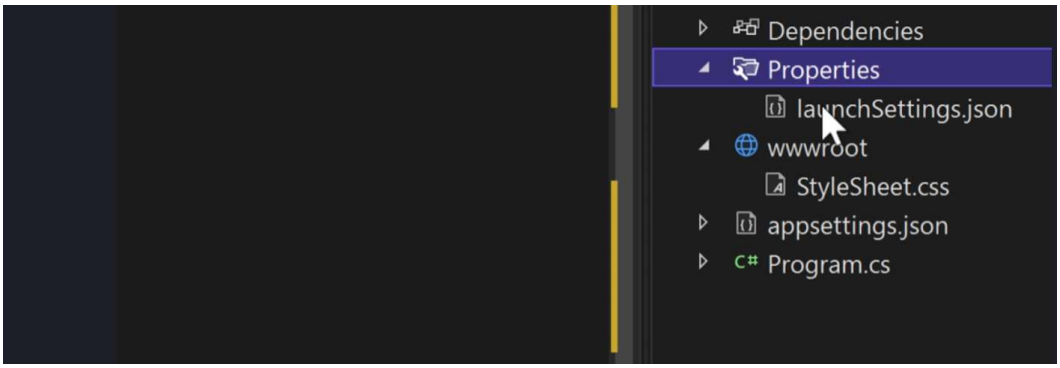
You should also understand the different ways to configure the environment name.
This is exactly what we will focus on in this section. In the next lecture, we will learn how to:

1. Configure the environment name in an ASP.NET Core web application project.
2. Read the current working environment in different files, such as the Program.cs file and the controllers.



In this lecture, we are going to focus on how to configure the environment name in the launchSettings.json file.
By default, when you create a new ASP.NET Core web application, it automatically generates a file called launchSettings.json.
This file contains the runtime profiles of the application, allowing one of them to be used for running the application.
We have already seen these profiles at the beginning of the course.
The key point today is that you can configure the current environment name under the **environmentVariables** property.
Let me show you how.

Now, create a new web application and open launchSettings.json file from 'Properties'

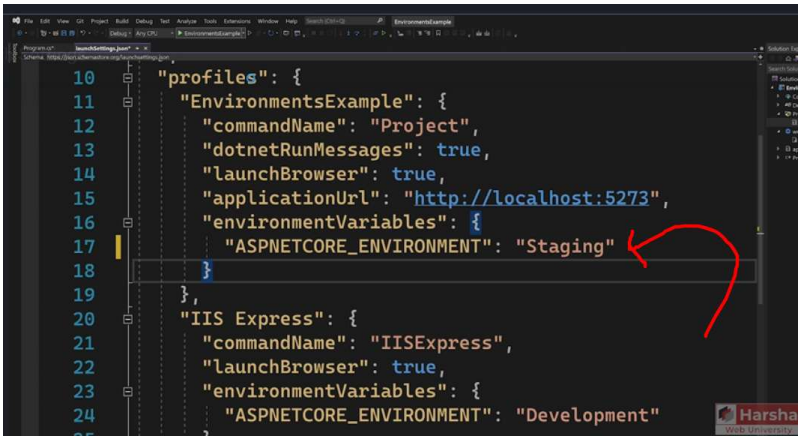


It has 2 profiles. One is for Kestrel Server. Another one is for IIS Express. We have discussed about the differences in the beginning of the lecture.

But the point is, you can add one or more environment variables with a predefined name. you can configure the environment individually for every runtime profile. That is for Kestrel server and IIS Express server.

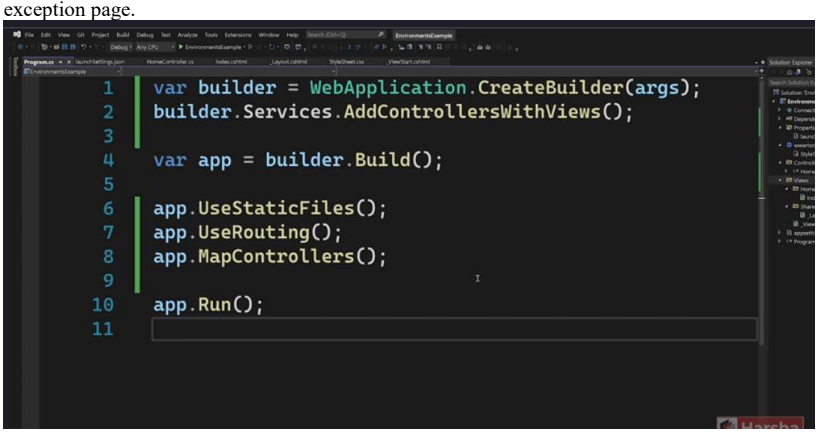
By default, when you create a new project, the environment is set as Development. But you can configure the same as staging / production or any other custom name whatever you want.

For Example, I would like to change the current working environment as staging. That's it



Changing the environment name does not make any differences to the output. The application works exactly same as before, then what will be the difference with this environment name?

Particularly in the program.cs file, if the current environment is the developer, you can enable the developer exception page.



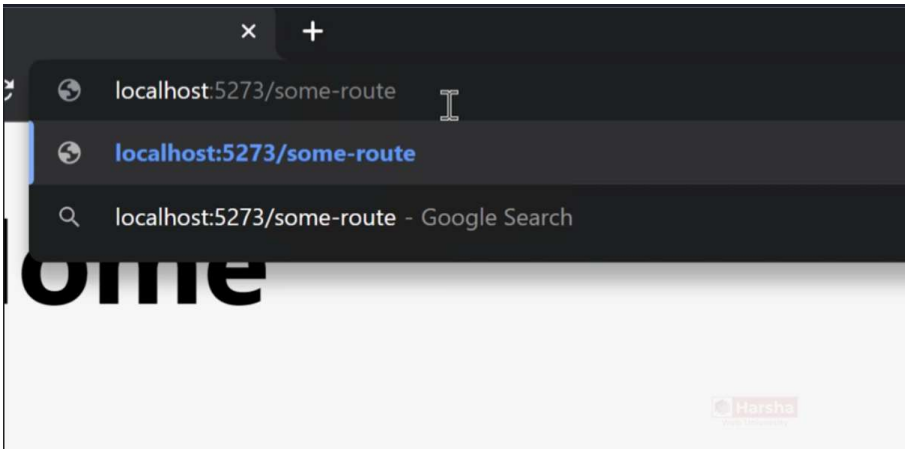
Alternatively, if the current environment is production, you can enable the custom exception page which is user friendly but not developer friendly. Means the developer error page may show the exception details page which is useful for coding purposes.

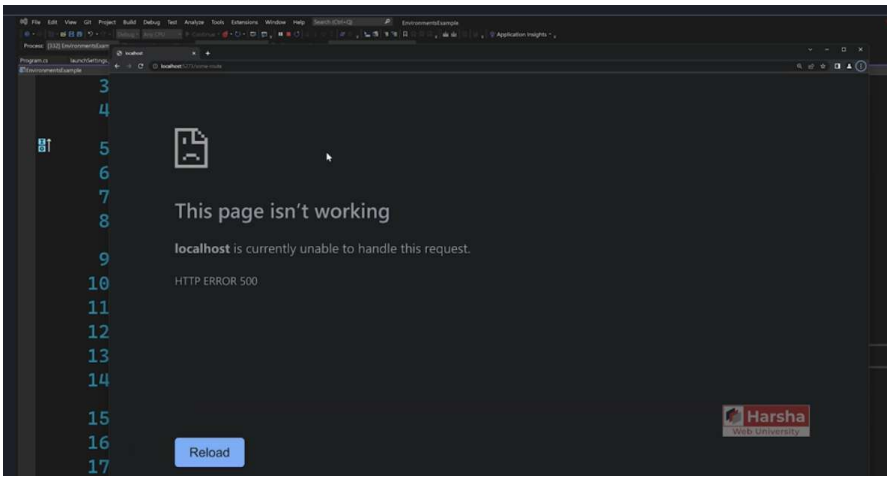
```
1  var builder = WebApplication.CreateBuilder(args);
2  builder.Services.AddControllersWithViews();
3
4  var app = builder.Build();
5
6  if (app.Environment.IsDevelopment())
7  {
8      app.UseDeveloperExceptionPage();
9  }
10
11 app.UseStaticFiles();
12 app.UseRouting();
13 app.MapControllers();
14
15 app.Run();
```

So, whenever an application runs and you got an exception due to some reason it automatically shows the built-in asp.net core error page which contain the complete details of the exception. Let's make some mistake intentionally so that our application can raise exception.

```
3  namespace EnvironmentsExample.Controllers
4  {
       0 references
5      public class HomeController : Controller
6      {
7          [Route("/")]
8          [Route("some-route")]
           0 references
9          public IActionResult Index()
10         {
11             return View();
12         }
13
14         [Route("some-route")]
           0 references
15         public IActionResult Other()
16         {
17             return View();
```

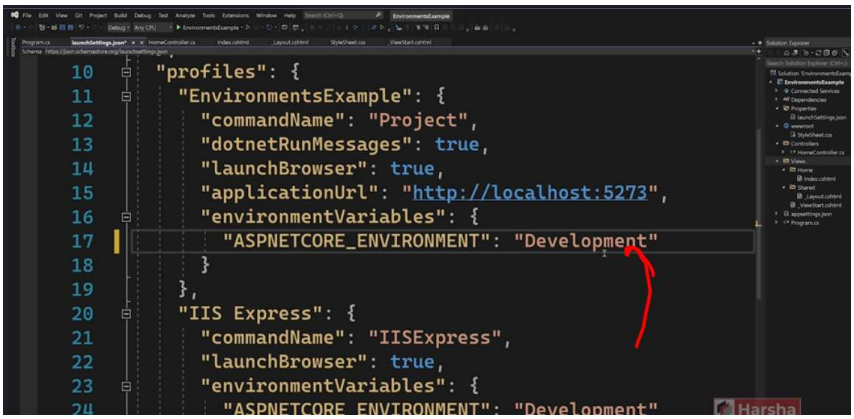
Now run this application.



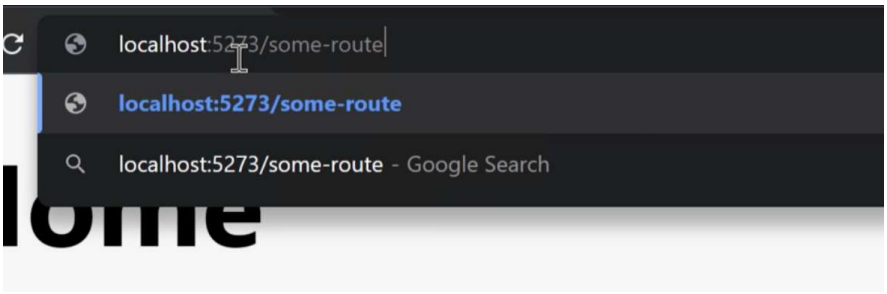


Since we are in staging environment, we have not got developer exception.

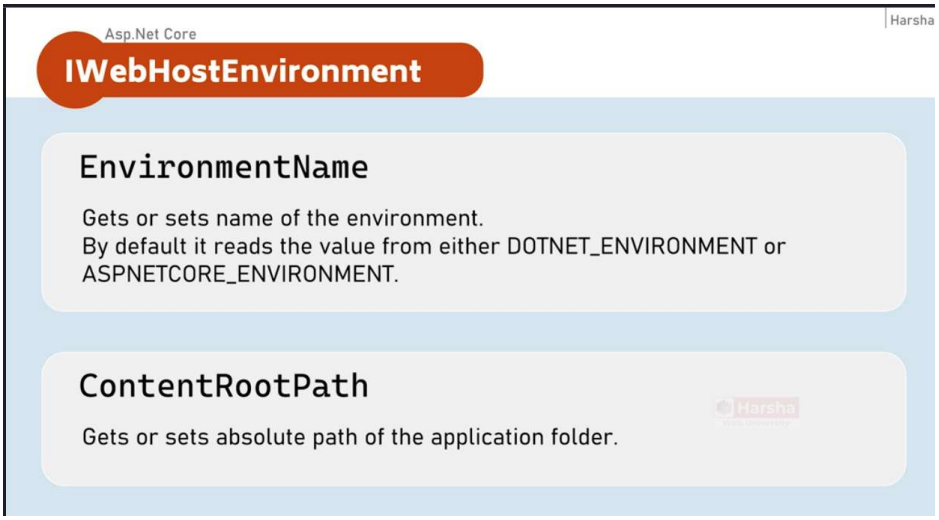
Now let's set the 'environment variables' to 'Development'



We have enabled the exception page. If any exception occurs, asp.net core automatically prepares the error page with exception details.



Optionally, you can also write environment name in the 'DOTNET_ENVIRONMENT' but if you mention both, asp.net core overrides the previous one. From asp.net core it is recommended to set only 'ASPNETCORE_ENVIRONMENT'



Asp.Net Core

IWebHostEnvironment

IsDevelopment()

Returns Boolean true, if the current environment name is "Development".

IsStaging()

Returns Boolean true, if the current environment name is "Staging".

IsProduction()

Returns Boolean true, if the current environment name is "Production".

Asp.Net Core

IWebHostEnvironment

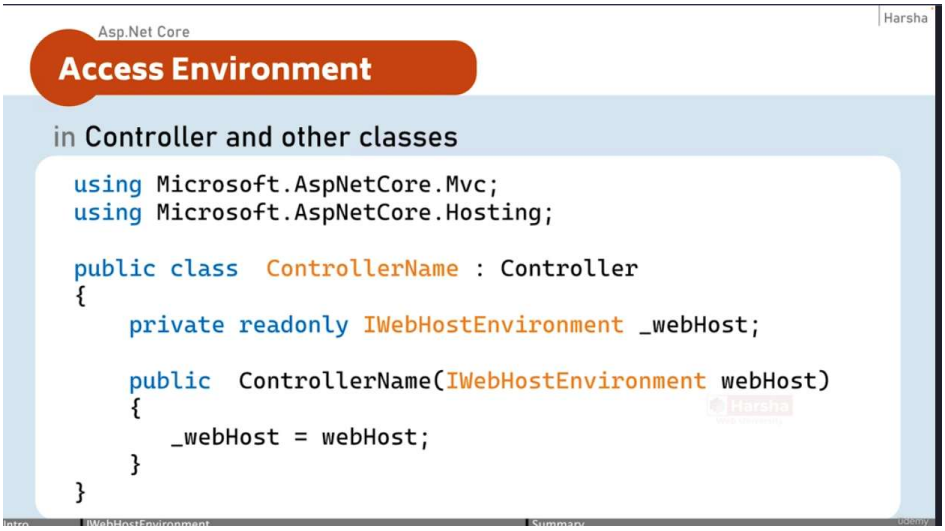
IsEnvironment(string environmentName)

Returns Boolean true, if the current environment name matches with the specified environment.

```
2  builder.Services.AddControllersWithViews();
3
4  var app = builder.Build();
5
6  if (app.Environment.IsDevelopment() ||
   app.Environment.IsStaging() ||
   app.Environment.IsEnvironment("Beta"))
7  {
8      app.UseDeveloperExceptionPage();
9  }
10
11 app.UseStaticFiles();
12 app.UseRouting();
13 app.MapControllers();
14
15 app.Run();
16
```

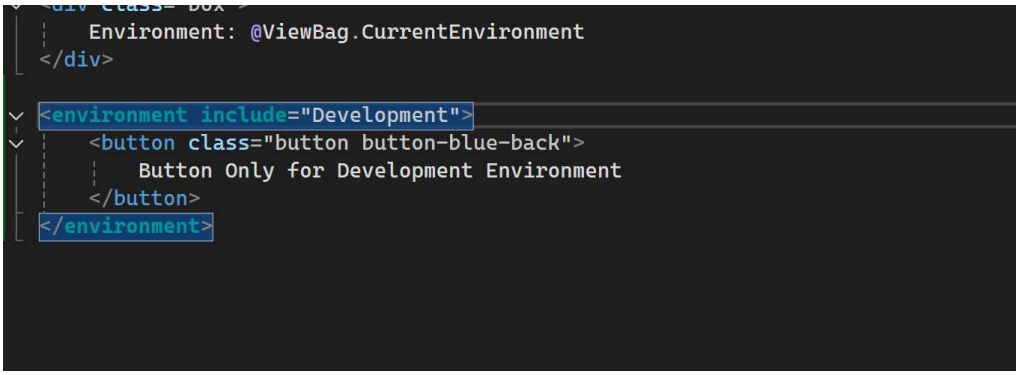
Custom environment name

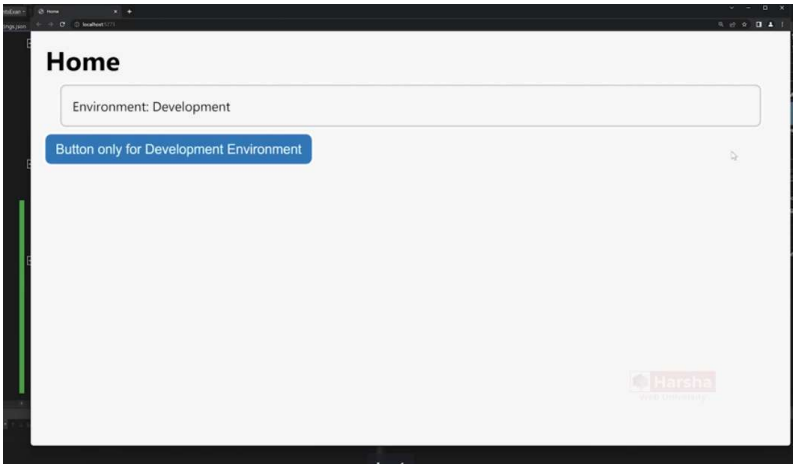
Sometimes in real-world cases you have to access the current working environment name programmatically, either in the controller or in any other services. So how can you do that? To access the current working environment, you can inject a predefined service called IWebHostEnvironment. So that using that service instance, you can access the current working environment, just like app.Environment in Program.cs.



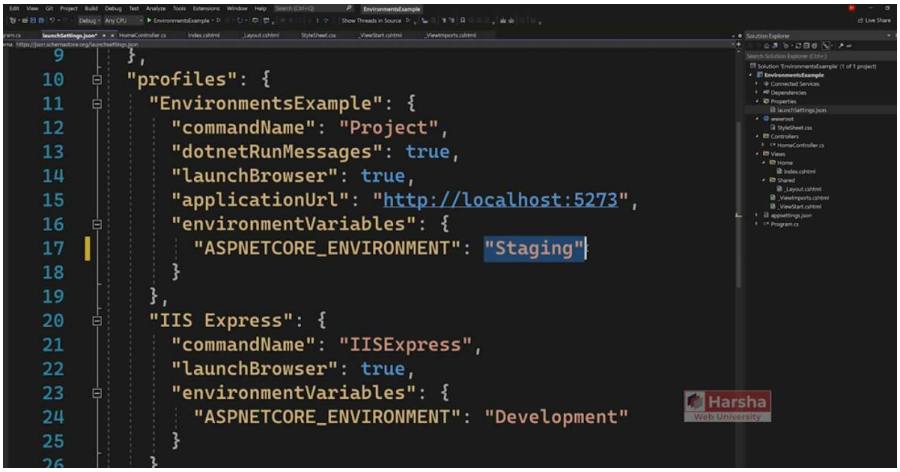
asp.net viewers can show additional content in the specific environment for example in case if you are in the development environment you may show a button for database migrations or opening the log otherwise in case if the current environment is production environment you may show an additional button for end user for the database backup like this you can show additional content or hide the existing content in the particular environment then you can use a predefined tag helper called environment like this with either include or exclude option let me show that practically.

It appears because currently we are in the 'Development' environment.

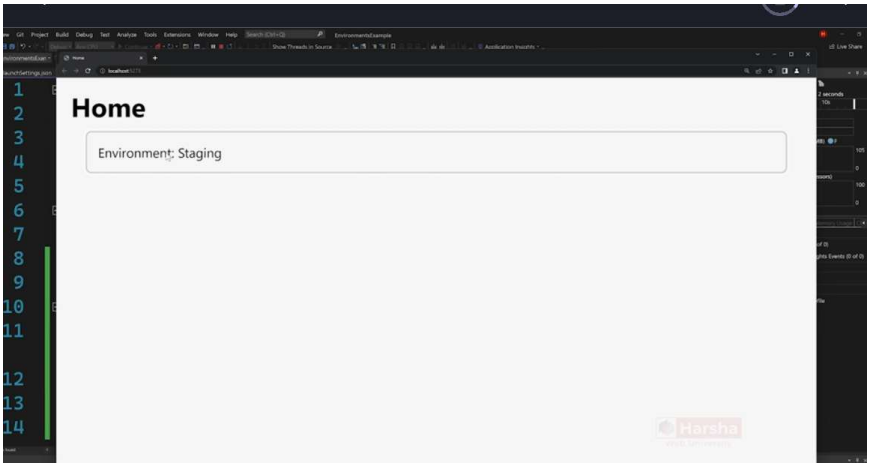


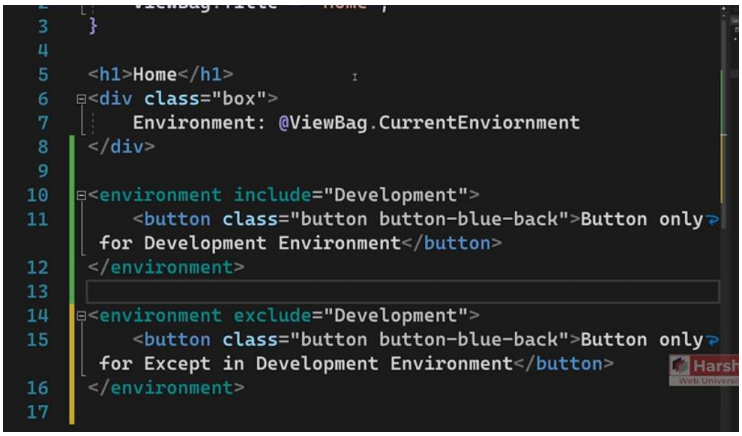


Go to launch.settings.json file and change it to any other.

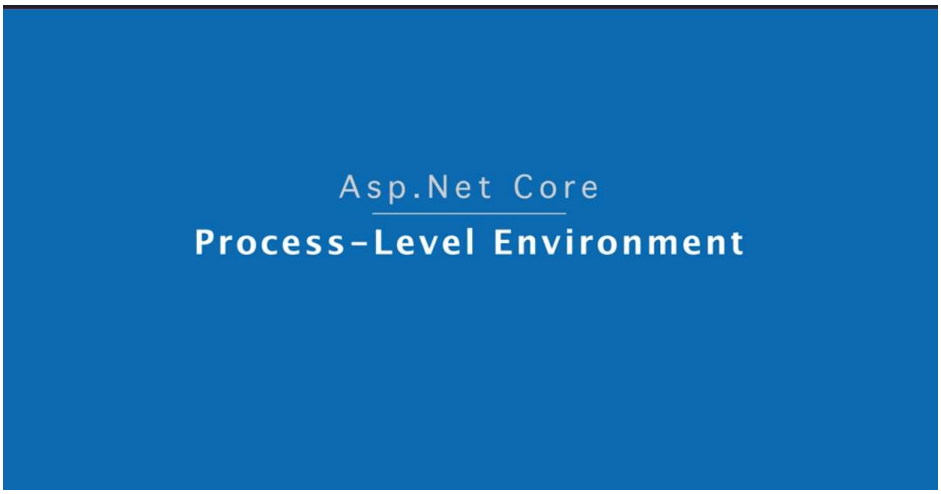


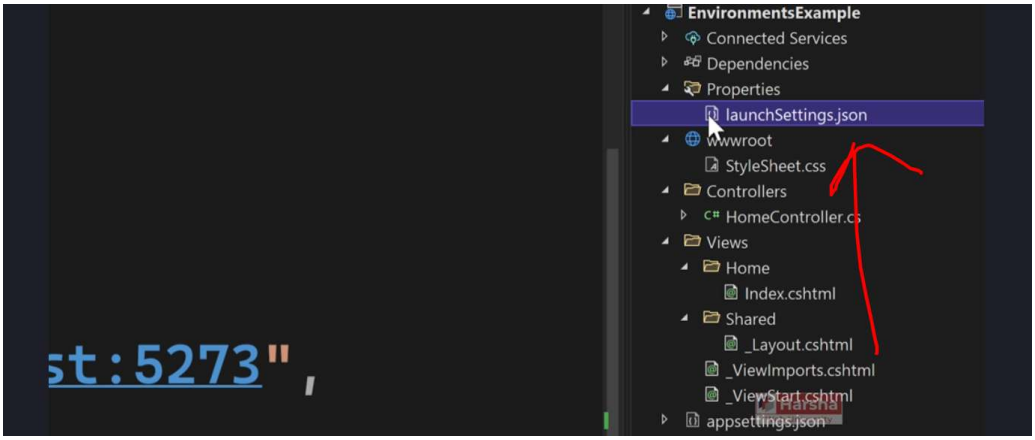
Close the application and re-run it. Yes, the button does not appear.



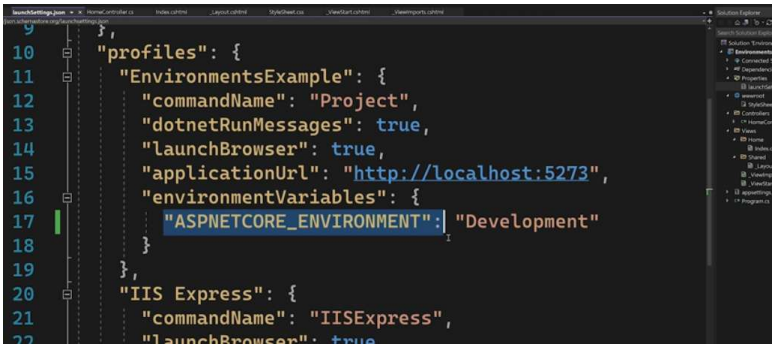


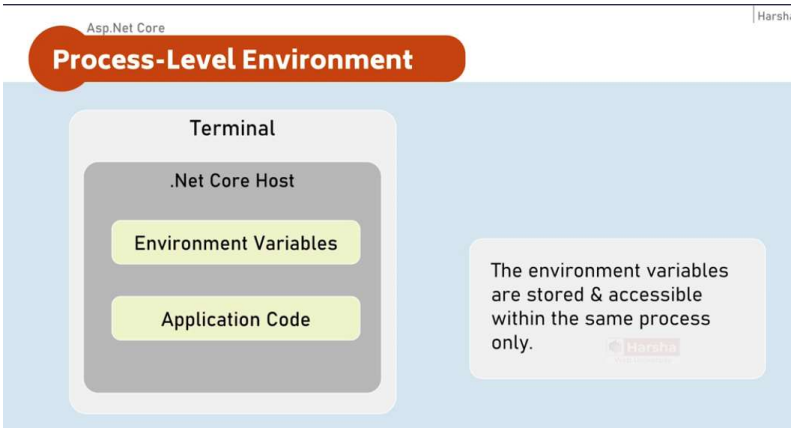
It's very rare to use it in real world applications.





Remember that this launchSettings.json file is only for the developer machine. In the case of a staging server or production server, you will not deploy this launchSettings.json file.
The question is, on machines other than the development machine, such as the staging or production server, how can you set these environment variables? Particularly, how do you set the ASP.NET Core environment on machines other than the developer machine?



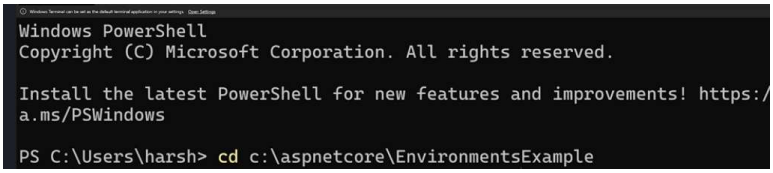


Imagine you are going to run your ASP.NET Core application on a machine other than the development machine.
On this machine, you have already installed .NET, meaning the framework is available on the staging or production server. However, Visual Studio will not be available on these servers.
In this case, you need to deploy the application to a specific folder and start running it through the Command Prompt or Windows PowerShell.
You can set up the environment variables directly through the Command Prompt or PowerShell. To do this, you will write a command to set the environment variable, and then you will start the application using the dotnet run command.

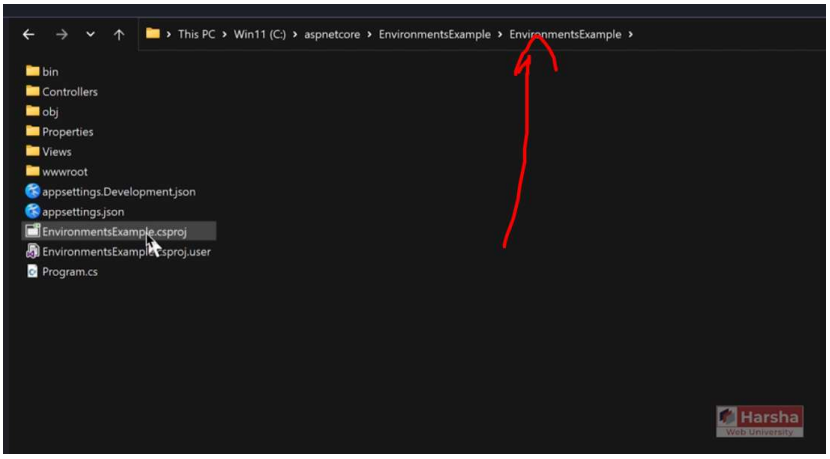
Let me demonstrate this practically.
Assume the current machine is the staging environment or possibly the production environment. You have the compiled source code with you, but the launchSettings.json file is not present.

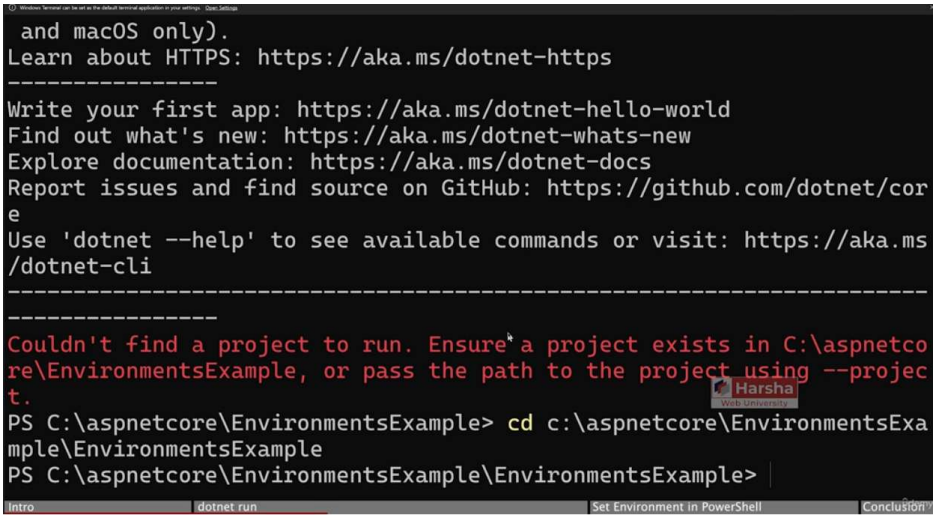
In this case, you need to open an inbuilt Windows application called **Terminal** (or Command Prompt/Windows PowerShell) to proceed.

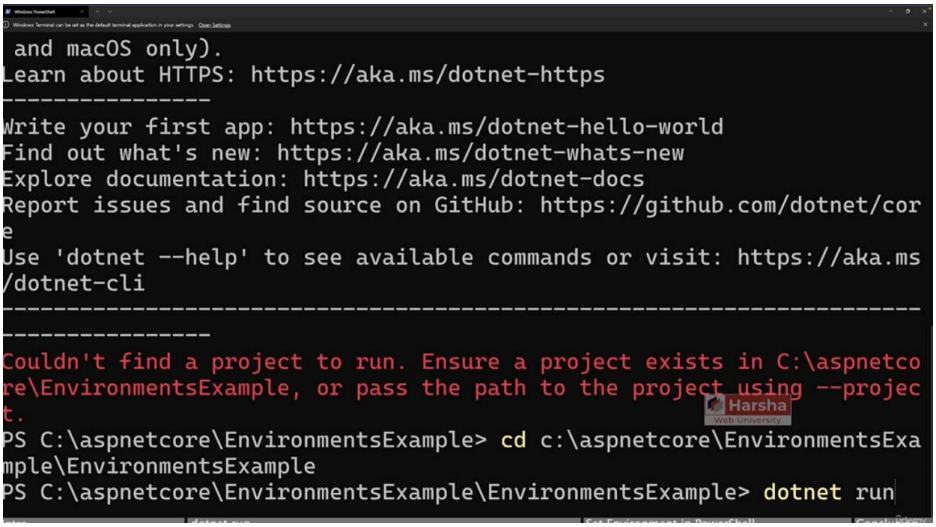
You can go to the search button and search for **Terminal**.
If you cannot find this app on your machine, you can download it from the Windows Store. Once installed, open the Terminal app.
Make sure you are using **Windows PowerShell** and not the regular Command Prompt.
In PowerShell, navigate to the folder where your application is located. For example, on my machine, it is located at:
C:\ASP.NET Core\Environments Example
Ensure you have navigated to the correct folder where your solution file is ready.

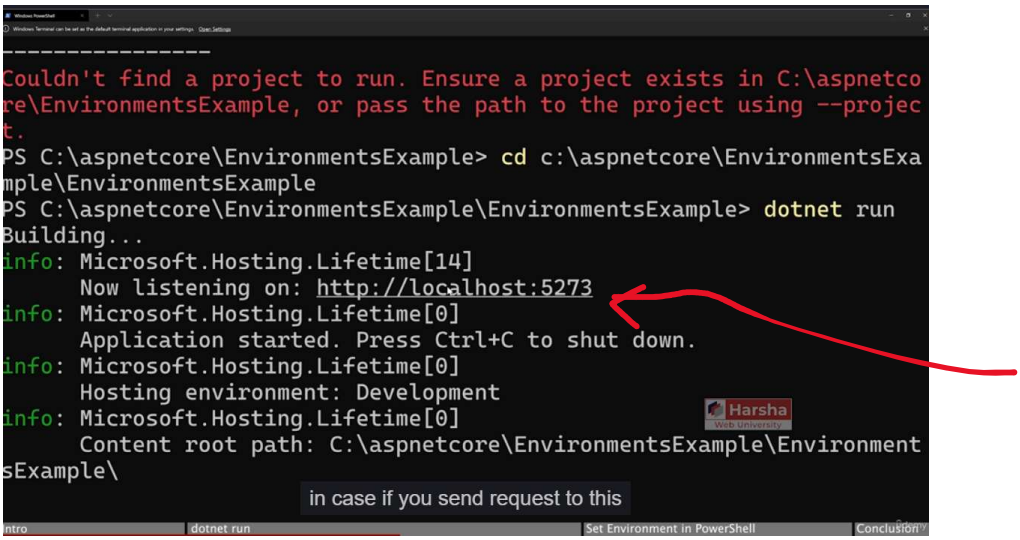


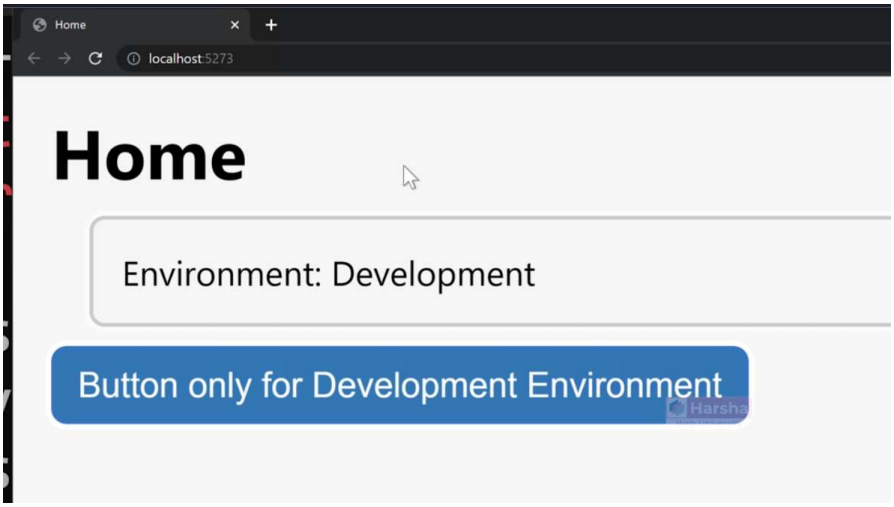
Navigate to:
C:\ASP.NET Core\Environments Example\Environments Example
The first Environments Example folder is the **solution folder**, while the second Environments Example folder is the **project folder**.



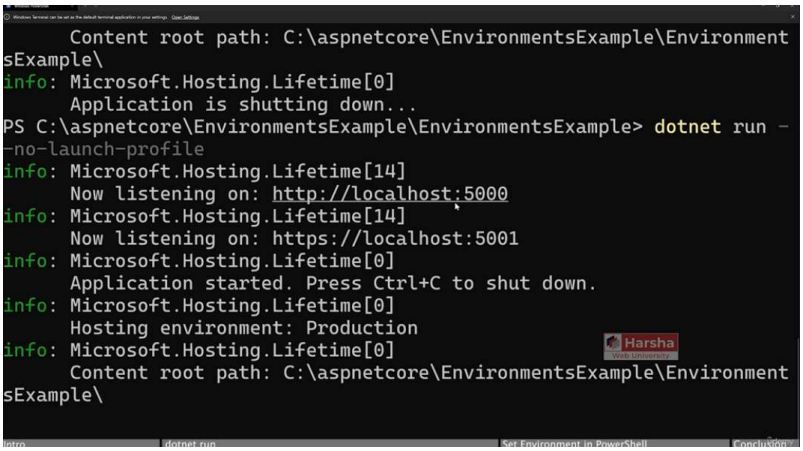








By default, the launchSettings.json file will be read automatically because it is present in that folder.
If you delete that file, it will automatically default to the **production environment**. However, we don't want to delete the file. Instead, you can use an option to disable the launch profile file while launching the application.
To do this, press **Ctrl + C** to stop the application.

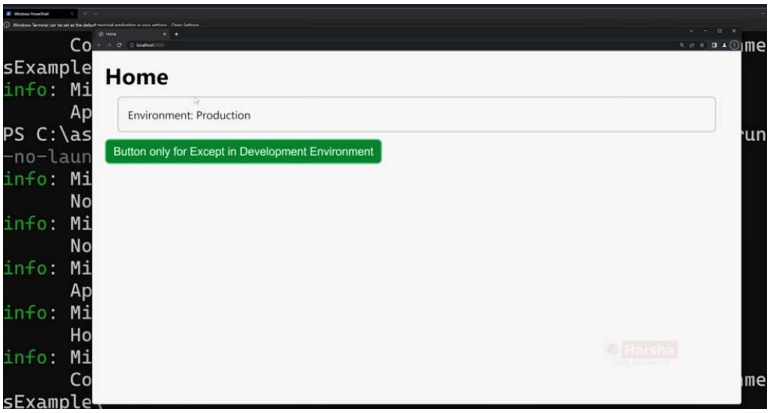


Now, return to the prompt and type:
dotnet run --no-launch-profile
This command will run the application again, but the launch profile will not be respected.

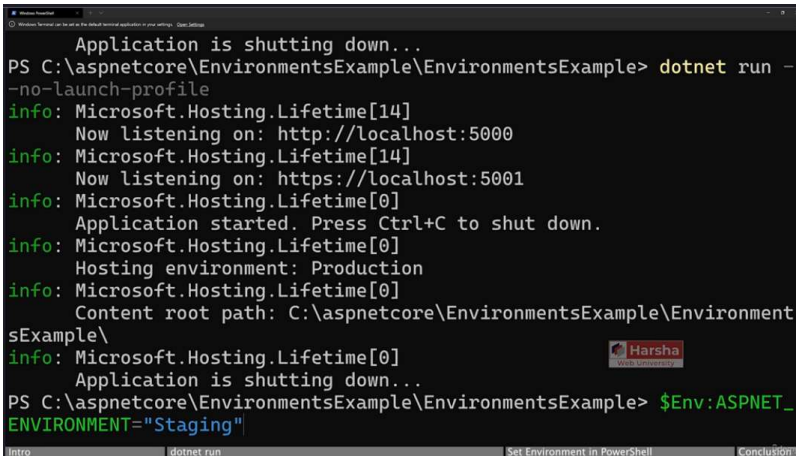
When you run the application with the --no-launch-profile option, the default port number localhost:5000 will be used for all applications.
The URLs and environment variables set in the launchSettings.json file will not be loaded because you specified no-launch-profile. This means the launchSettings.json file is effectively disabled.
You can **Ctrl + Click** on the URL to open it. The application will run with the default port number, 5000, which is the default for all ASP.NET Core applications.

Here's the revised version of your text with improved grammar and clarity:

If you disable the launch profile, ASP.NET Core will by default set the environment to **production**. That's why it is showing the environment as **production**.
However, if you want to set it to a specific environment, such as **staging** or another custom environment, you can do so.



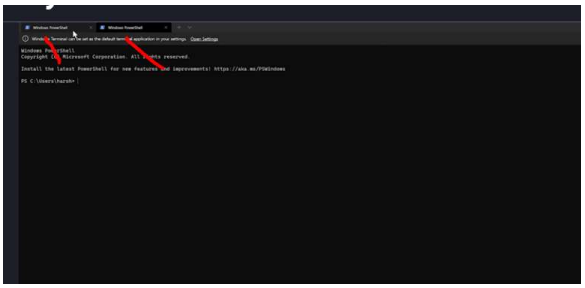
To set a specific environment, you will use a command to set environment variables from the Command Prompt. This is also known as setting **process-level environment variables**.
The environment variables set through this command prompt will only be read by this specific Command Prompt session, not by other processes.
This makes it one of the most secure ways to set environment variables.
To proceed, press **Ctrl + C** to stop the application. Then, run the following command



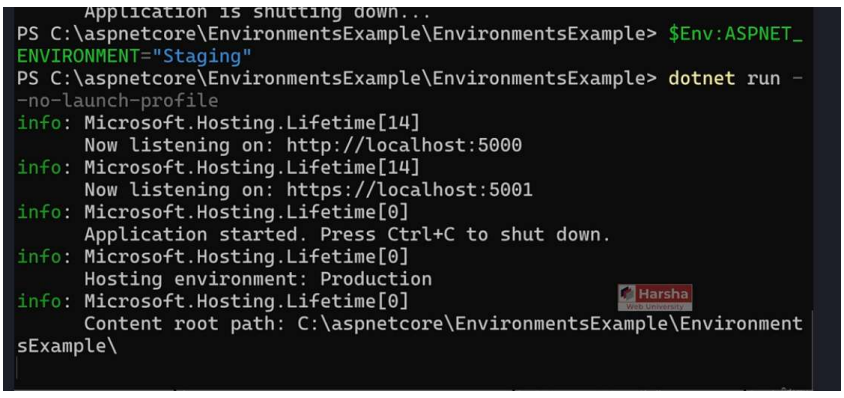
$env:ASP_NET_ENVIRONMENT="staging"

Note that this is **case-sensitive**: env should be lowercase, while ASP_NET_ENVIRONMENT should be uppercase. You can replace "staging" with any other environment value, such as production or development.

so this will be equivalent to setting up
the environment variables
but this will be saved only for the same
command prompt that means the same
powershell tab
in case if you open up a different tab
the environment variables of the other
tab cannot be read from this
so it is secured



Switch back to the previous tab. Now make sure you mentioned the command correctly: $env (where E is uppercase), followed by a colon without space, then the key equal to value, with the value in double quotes. Then press the Enter key. Okay, that environment variable has been saved.



Now, you can use the same command:

dotnet run --no-launch-profile as usual.
Since we set the environment variable within the same process through this command prompt, this environment variable will be respected instead of the default, which is **production**.
So, **Ctrl + Click** on localhost:5000 to access the application.