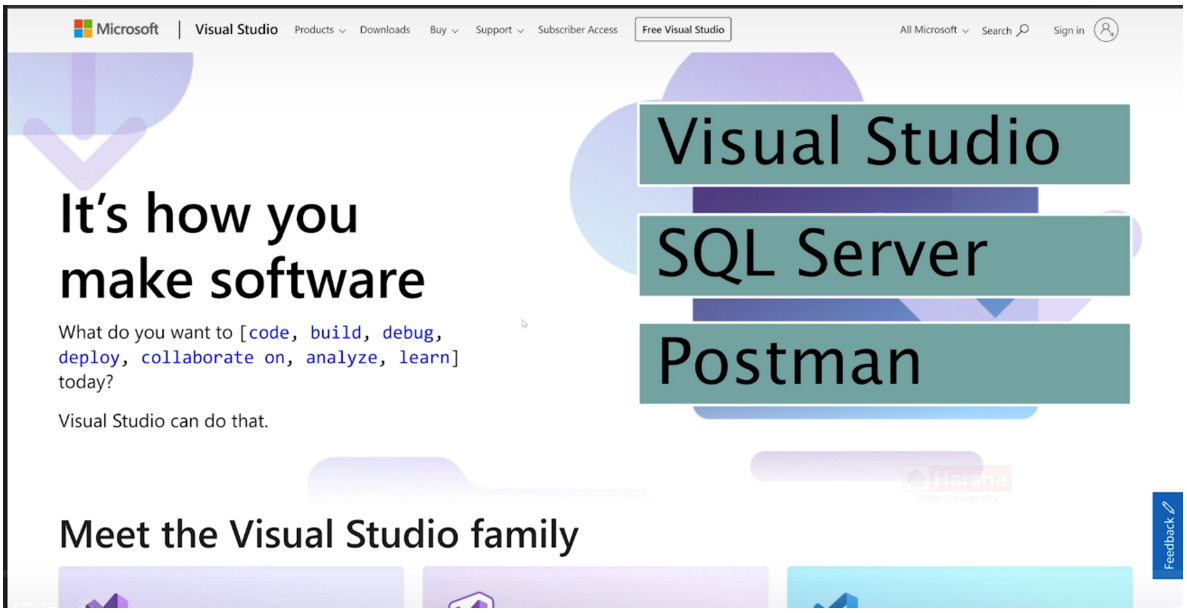
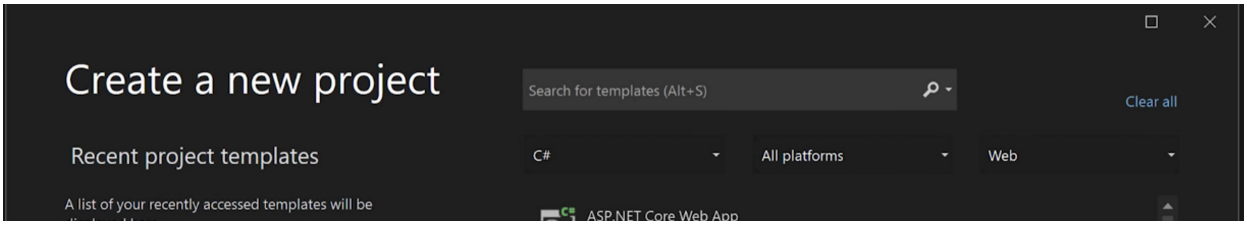
Section 2: Getting Started [MVC and Web API]

29 December 2024 12:40

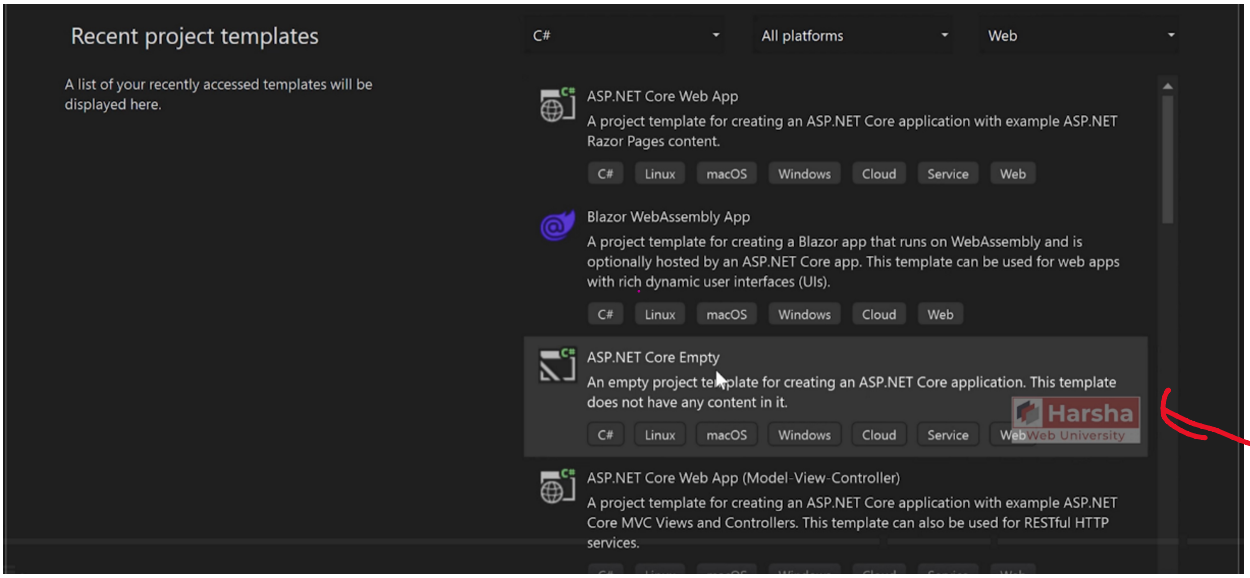
1. Setup Environment



2. Create First Asp.Net Core App



06

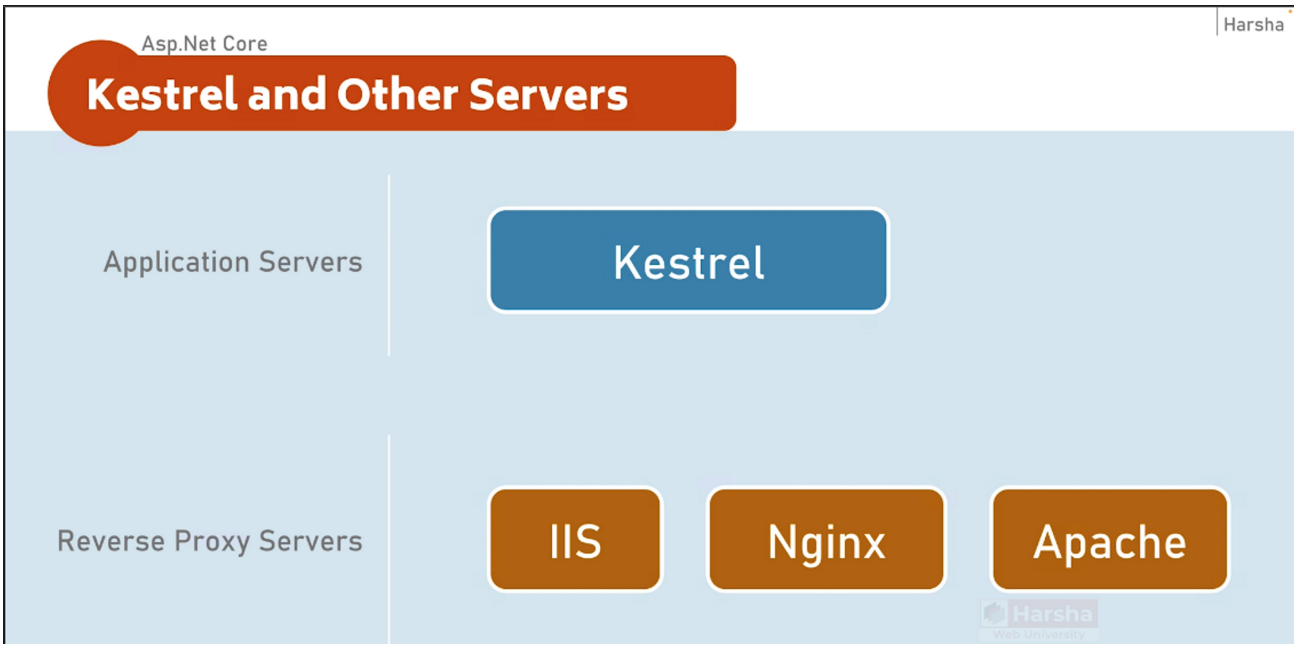


06

For, Learning + Real World Project Developement

The execution starts from 'program.cs' file.

3. Kestrel and Other Servers



ASP.NET Core Applications and Servers

1. Kestrel (Default HTTP Server)
Kestrel is the default cross-platform HTTP server for ASP.NET Core applications.
It serves as both a development server and an application server during real-world production scenarios.

2. Development Phase
During development, Kestrel is used by developers to:
Test and run the application.
Serve local HTTP requests while writing and debugging ASP.NET Core code.

3. Production Phase
In production, Kestrel acts as the application server, but it is typically combined with a reverse proxy server for additional functionality and robustness.

Reverse proxy servers like IIS, Nginx, or Apache are commonly used

in front of Kestrel to:

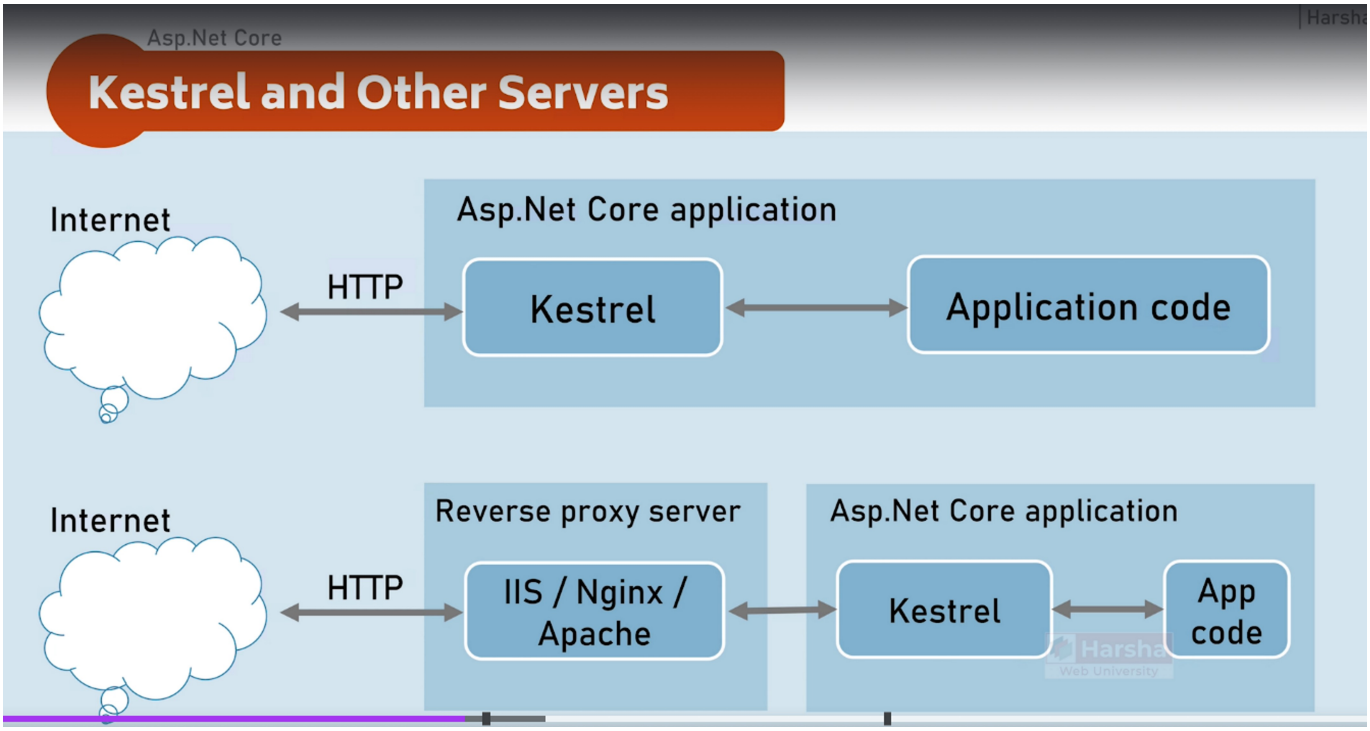
Handle requests from the internet.
Manage tasks such as load balancing, SSL termination, and request routing.

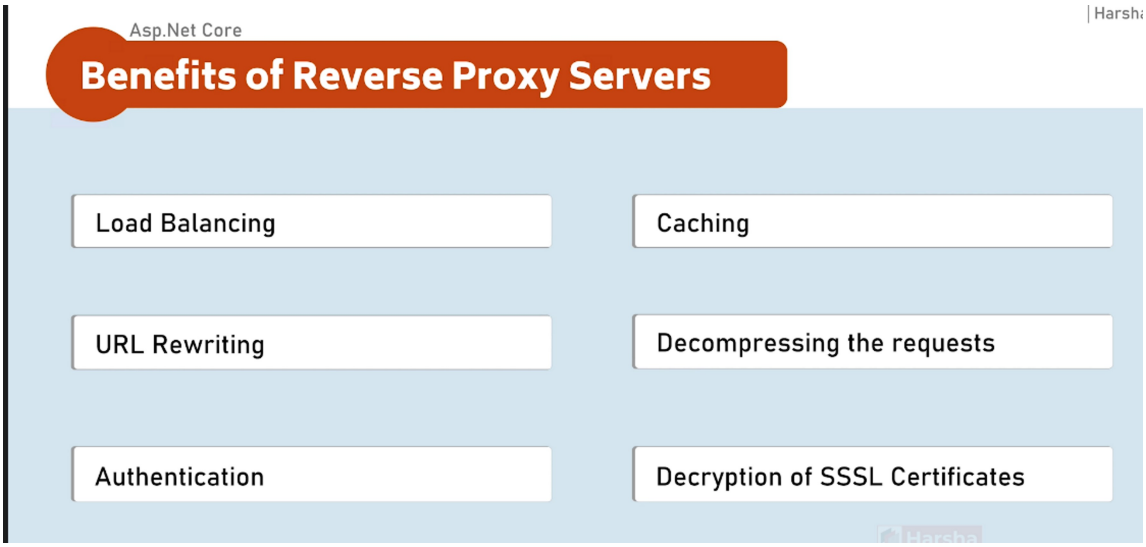
4. Typical Architecture
Development

Browser → Kestrel

Production:
Browser → Reverse Proxy (IIS/Nginx/Apache) → Kestrel → ASP.NET Core Application

Key Points
Kestrel is lightweight, efficient, and designed for high performance.
While Kestrel can serve requests directly, reverse proxy servers add important production-level capabilities such as scalability, security, and better request handling.





With only Kestrel:

The HTTP request will be received from the local network or the internet.
Kestrel will receive that request and forward it to the application code.
Kestrel will receive the request and fill the information in the form of an object called HTTP Context.
Kestrel prepares an HTTP context object containing the details of the request and sends it to the application code.

The application code receives that context, processes it, and provides the response back to Kestrel.
Kestrel sends the response back to the internet or the client.

This is the general process that happens with Kestrel.
However, Kestrel doesn't support some advanced features required on the internet today, such as:

1. Load balancing
2. URL rewriting

These features are not supported by default in Kestrel.
But don't worry—this is not a drawback or problem.
Most websites today use a reverse proxy server.

Meaning:
The actual request from the client will be received by a reverse proxy server such as IIS, Nginx, or Apache.
These reverse proxy servers offer advanced features such as:

1. Load balancing
2. URL rewriting
3. Authentication
4. Caching

These servers then transfer the same request to the actual application server, which is Kestrel.

In this way, even though Kestrel can receive requests from the internet, it typically doesn't in real-world cases.
In most production scenarios, reverse proxy servers are preferred due to their advanced features.
The reverse proxy server transfers the request to Kestrel.
Kestrel receives the request, prepares an HTTP context containing request details such as the request body, headers, session, cookies, etc.
The application code processes the request and provides the response back to Kestrel.
Kestrel sends the response back to the reverse proxy server, which sends it to the client.

In real production scenarios, this is the process that happens.

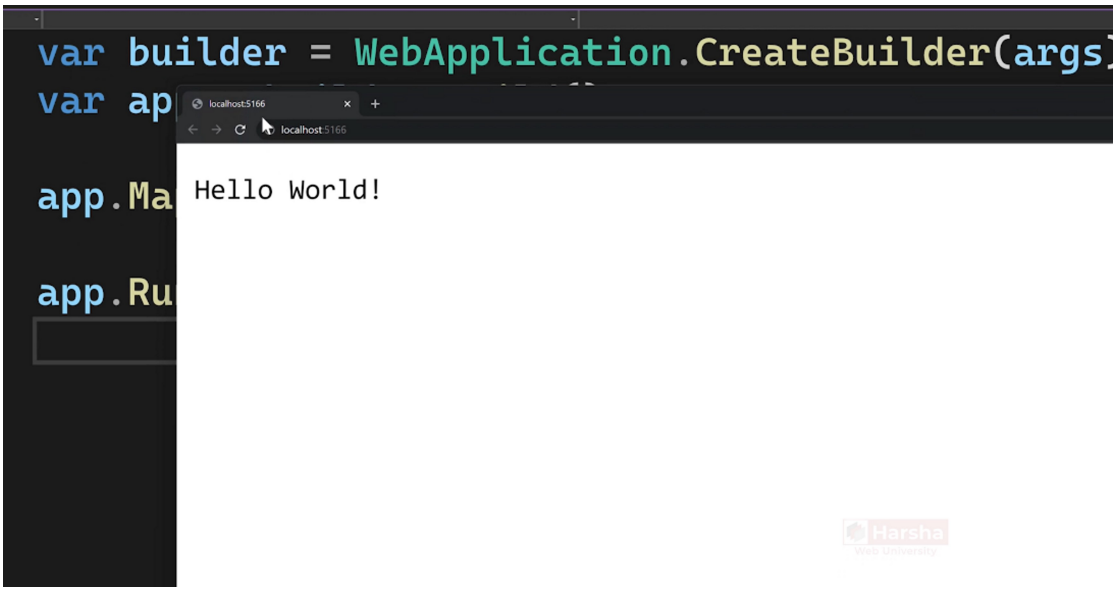
For now, you don't need to worry much about reverse proxy servers during the initial stages.
You can concentrate on Kestrel.
Kestrel is the default HTTP server for ASP.NET Core applications—this is the key point to remember.

Additionally, during development, you can simulate the features of a reverse proxy server using a dummy reverse proxy server called IIS Express.
IIS Express simulates IIS, acting as a reverse proxy server that can receive requests and forward them to Kestrel.

During development, using IIS Express is optional.
IIS Express is a lightweight version of IIS, simulating the process of real IIS.

Both IIS Express and actual IIS are supported by Windows operating systems only.
Nginx and Apache are mainly supported by Linux but are also supported by Windows.

This is an overview of the servers used in ASP.NET Core applications.

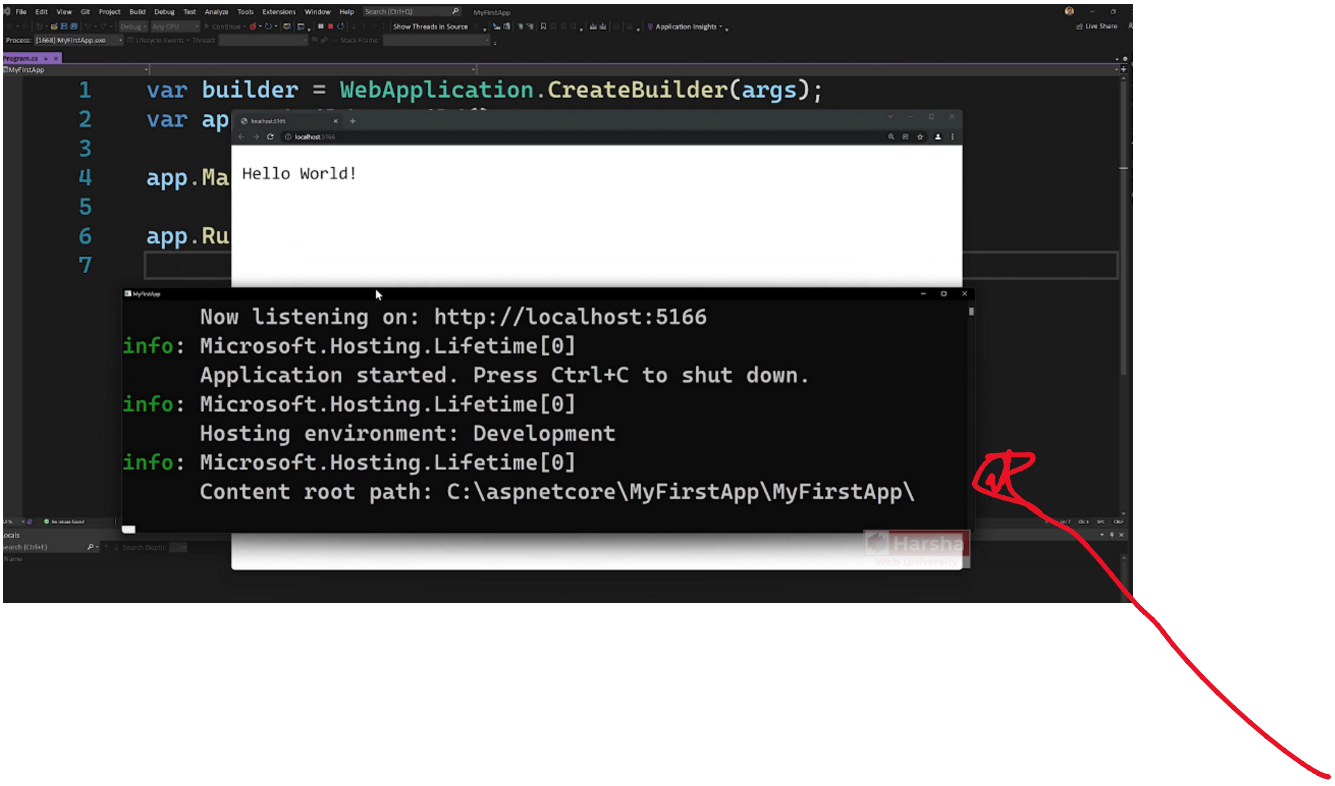


But where is Kestrel? Can we see that?

By default, when you run the application, it automatically opens your default browser (e.g., Chrome or Firefox) and sends the request.

Our application receives the request via Kestrel and provides the response, such as "Hello World." This is the common process.

but before this thing happens means before the browser can send request
internally on the terminal the kestrel server has been opened. you can see that here.

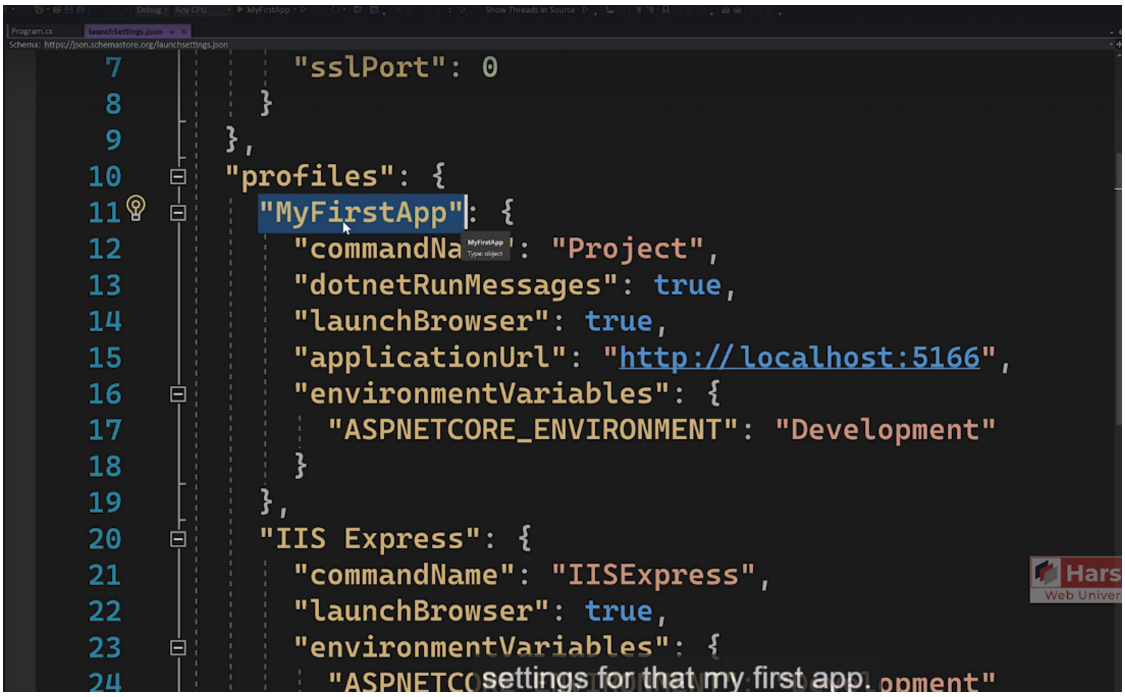


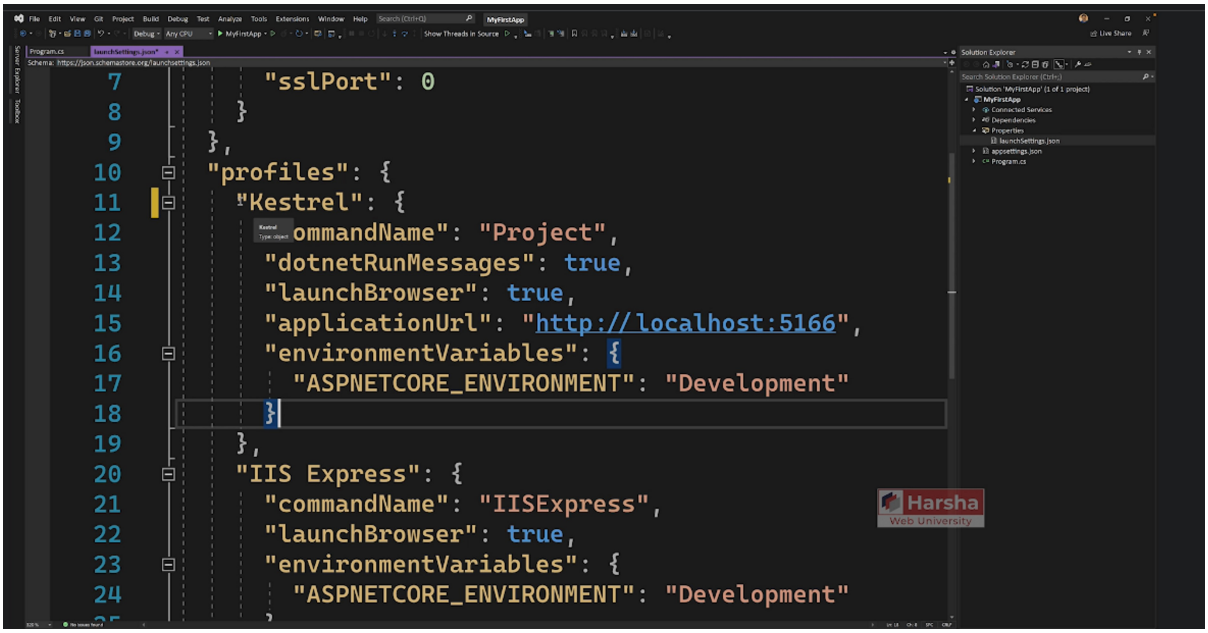
but how do you enable the reverse proxy server that is IIS express? let us try that by using launch settings in the next lecture.

Kestrel is the default server implementation for ASP.NET Core.

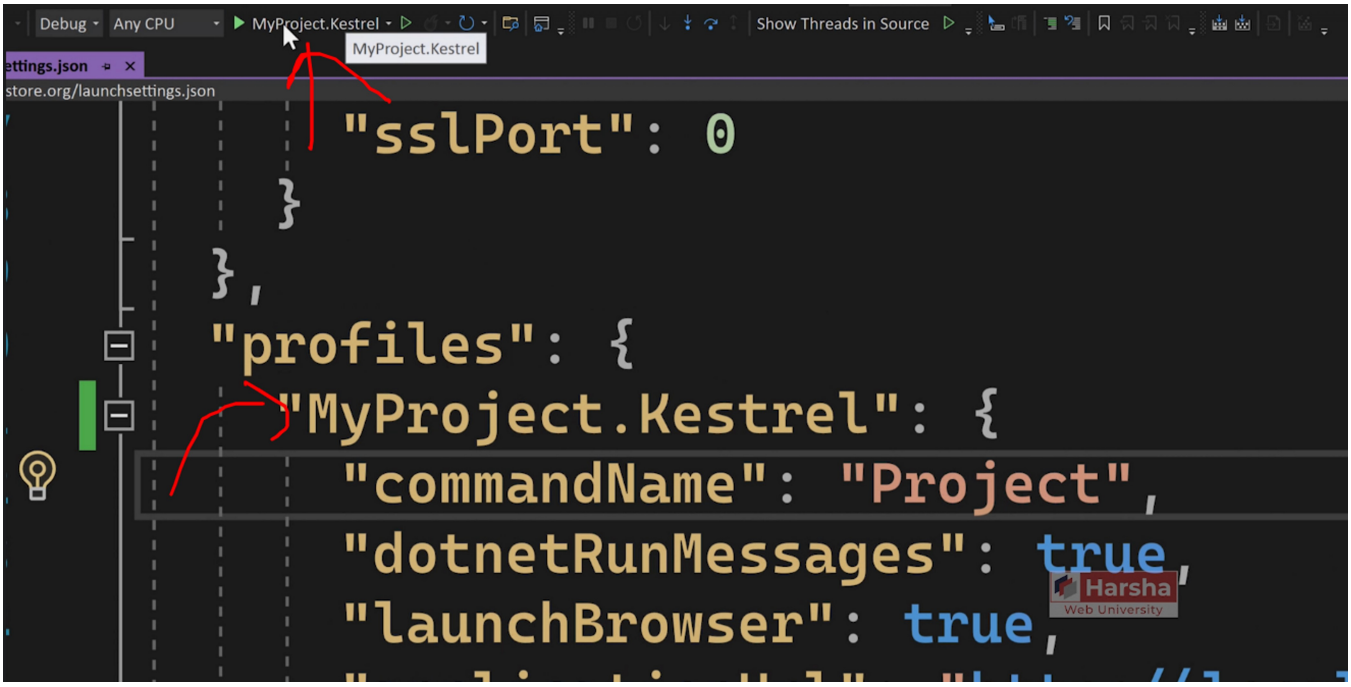
By default, when you run the application, Kestrel runs in the background and opens on the terminal window.
Additionally, you can use a reverse proxy server like IIS Express. In this case, the browser's request will be received by IIS Express and forwarded to Kestrel.

In case you want to enable it or switch between them, you require to use a file called launchSettings.json. So, in your solution explorer, expand the folder called Properties, and there you will find a file called launchSettings.json. Just double-click on that, and you will be able to see some JSON configuration. In case you're new to JSON, don't be confused about that—it's just a collection of key/value pairs. For example, iisSettings is the key, and the braces will be the value for the same. Similarly, you have something called profiles, and this is the value for the same. By default, you have two profiles configured when you create a new project. First, focus on the first profile, which is named "my first app." These are the configuration settings for "my first app."



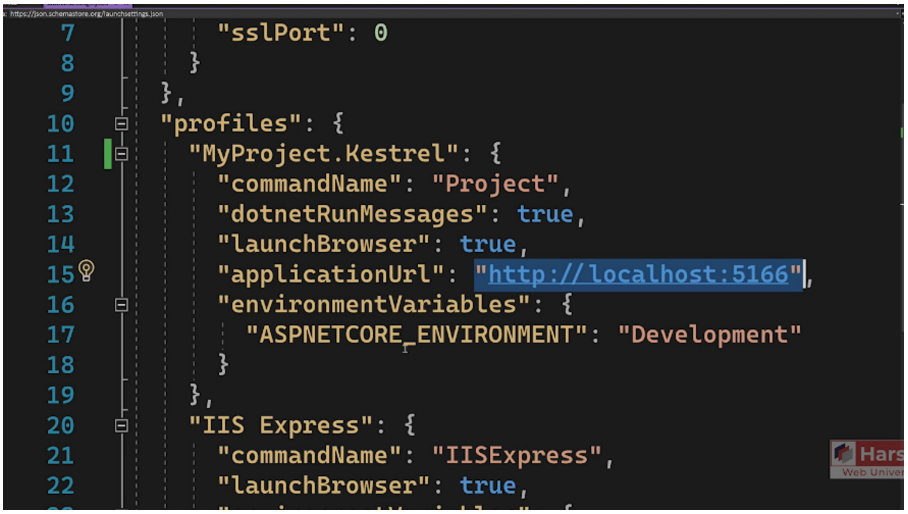


This will be the name of your profile. So, what does the profile do for us? A profile is a collection of settings that enables a particular server to run our application when you start the project. This is the name of the profile, and you can rename it. For example, I am trying to give it the name "kestrel" instead of "my first app," which is a more meaningful name. Because by default, when you use this profile to run, it uses the Kestrel server.cf



Let us explore these settings. The first setting is commandName, which has the value project. This means we are enabling the Kestrel server. The two options for commandName are project and IIS. If commandName = project, it indicates you would like to use Kestrel only.

The second option is dotnet run messages: true. This is related to the .NET CLI, which is used to run commands like creating applications, adding files, etc., through the terminal, similar to the Angular CLI in the Angular framework. If you run any .NET command and want to see relevant messages or updates in the terminal, you can enable this setting by setting it to true. Otherwise, set it to false. This will display the relevant messages in the Kestrel window if any .NET commands are used in the terminal. However, we haven't covered the .NET CLI yet, so we won't show that demo right now. When we learn about the .NET CLI in the future, we'll do that.



The third setting is easy to understand: launchBrowser: true. When set to true, the browser automatically opens when you try to use Kestrel. In my case, the default browser on my machine is Chrome, but you can change your default browser in your Windows settings. So, with launchBrowser: true, the browser launches automatically with the appropriate URL, which you can configure using the applicationUrl setting.

There is a separate section that explains environment variables. You can choose the environment to run the application in, such as development, production, or staging. You can also enable environment variables like API keys, server names, or redirection URLs, which are global and can be accessed anywhere in the code. You will learn more about environment variables later. These are the default settings available in your Kestrel server.