Introduction to ASP.NET Core

ASP.NET Core

By Jafar Muzeyin

OBJECTIVES

You are going to learn:

- 👉 Defining ASP.Net Core
- 👉 Defining .NET CORE
- 👉 .NET standard
- 👉 What can we build with .NET Core
- 👉 .NET Framework Vs .NET CORE
- 👉 .NET 5

WHAT IS ASP.NET CORE?

ASP.NET core

ASP.NET Core is a cross-platform, open source, web application framework that you can use to quickly build dynamic, server-side rendered applications.

System that can work across multiple types of platforms or operating environments

Software with source code that anyone can inspect, modify, and enhance.

A set of resources and tools for software developers to build and manage web applications, web services and websites.

WHAT IS .NET CORE?

.NET CORE

- The .NET Core is a runtime. It is a complete redesign of .NET Framework.
- The main design goal of the .NET Core is to support developing cross-platform .NET applications.
- It is supported on Windows, Mac OS & Linux.
- .NET Core is an Open Source Framework maintained by Microsoft and the .NET community on GitHub
- The .NET Core is a subset of Full .NET Framework. WebForms, Windows Forms, WPF are not part of the .NET Core
- It implements .NET Standard specification.

WHAT IS .NET STANDARD?

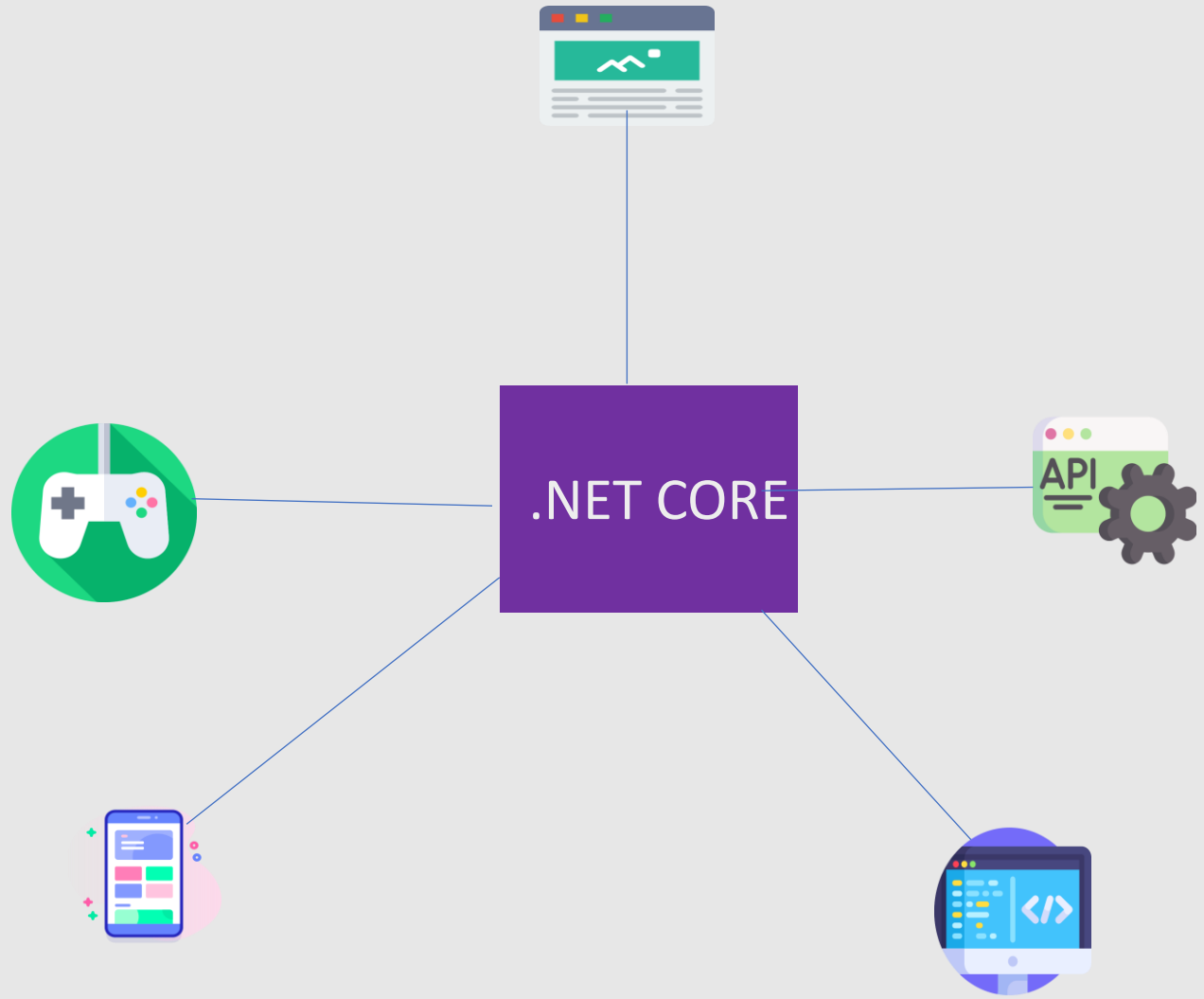
.NET STANDARD

- .NET Standard is a formal specification of .NET APIs that are intended to be available on all .NET implementations.
- It defines a uniform set of rules that need to be followed across all .NET implementations
- You can read more about **.NET Standard** from here

WHAT CAN WE BUILD .WITH NET CORE?

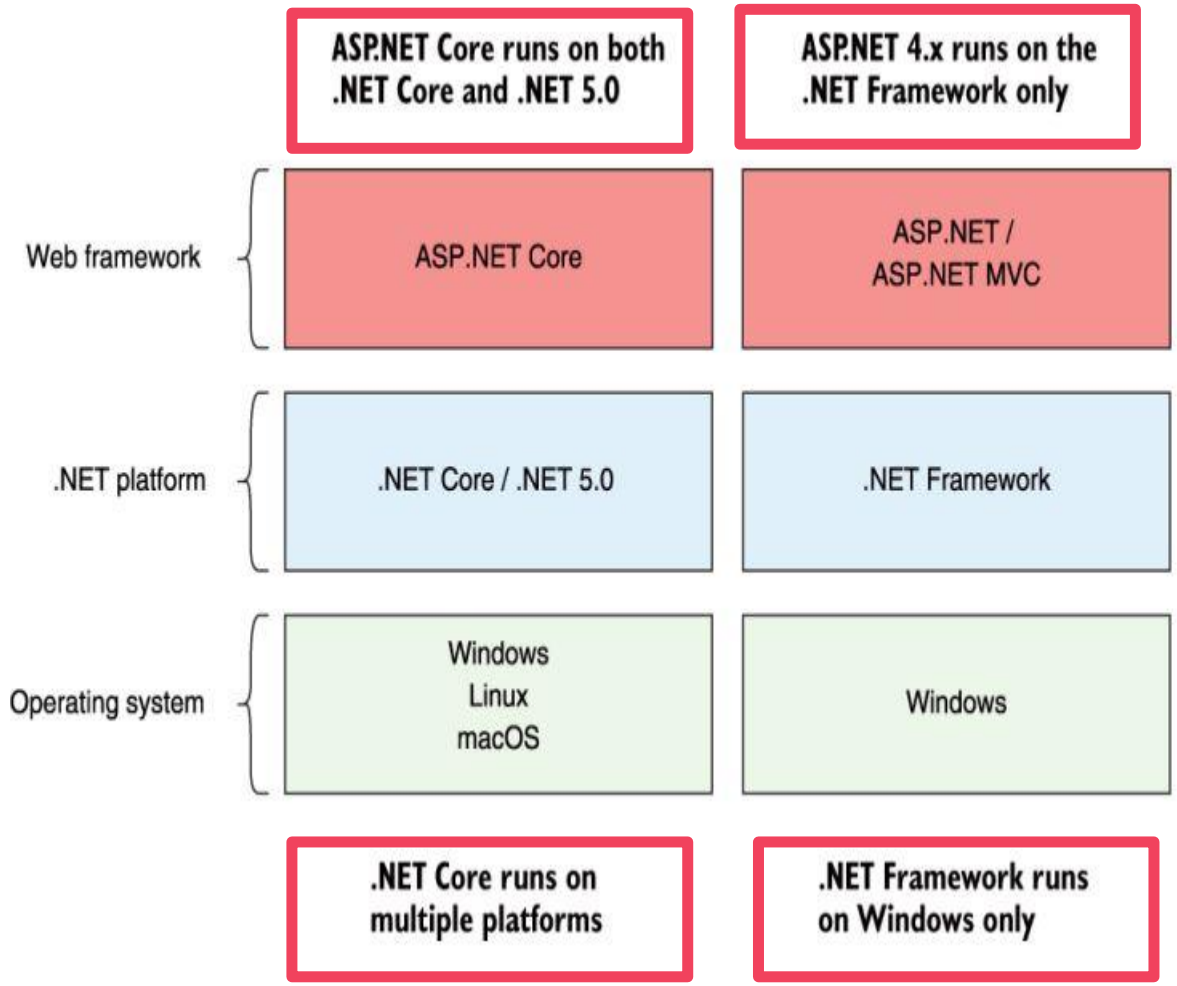
.NET CORE

- Different frameworks within .NET CORE will help to develop to build any application type.



DIFFERENCE BETWEEN .NET FRAMEWROK AND .NET CORE?

- .NET Framework and .NET Core are .NET implementations for building server-side applications.
- The .NET Core supports the subset of features supported by the .NET Framework. The features like WebForms, WindowsForms, WPF are unlikely to make into the .NET Core
- .NET Framework only runs on windows. .NET Core applications can run on any platform



IMPORTANT FEATURES OF .NET CORE?

.NET CORE FEATURES

- You can build and run cross-platform ASP.NET apps on Windows, Mac and Linux (Open source and community focused
- ASP.NET Core Unifies MVC & Web API.
- Ability to host on IIS or self-host in your own process.
- Built-in Dependency Injection.
- Easy integration with client-side frameworks like Angular, Knockout etc.
- An Environment based configuration system.
- light-weight and modular HTTP request pipeline.
- Ships entirely as NuGet packages.

WHAT IS .NET 5?

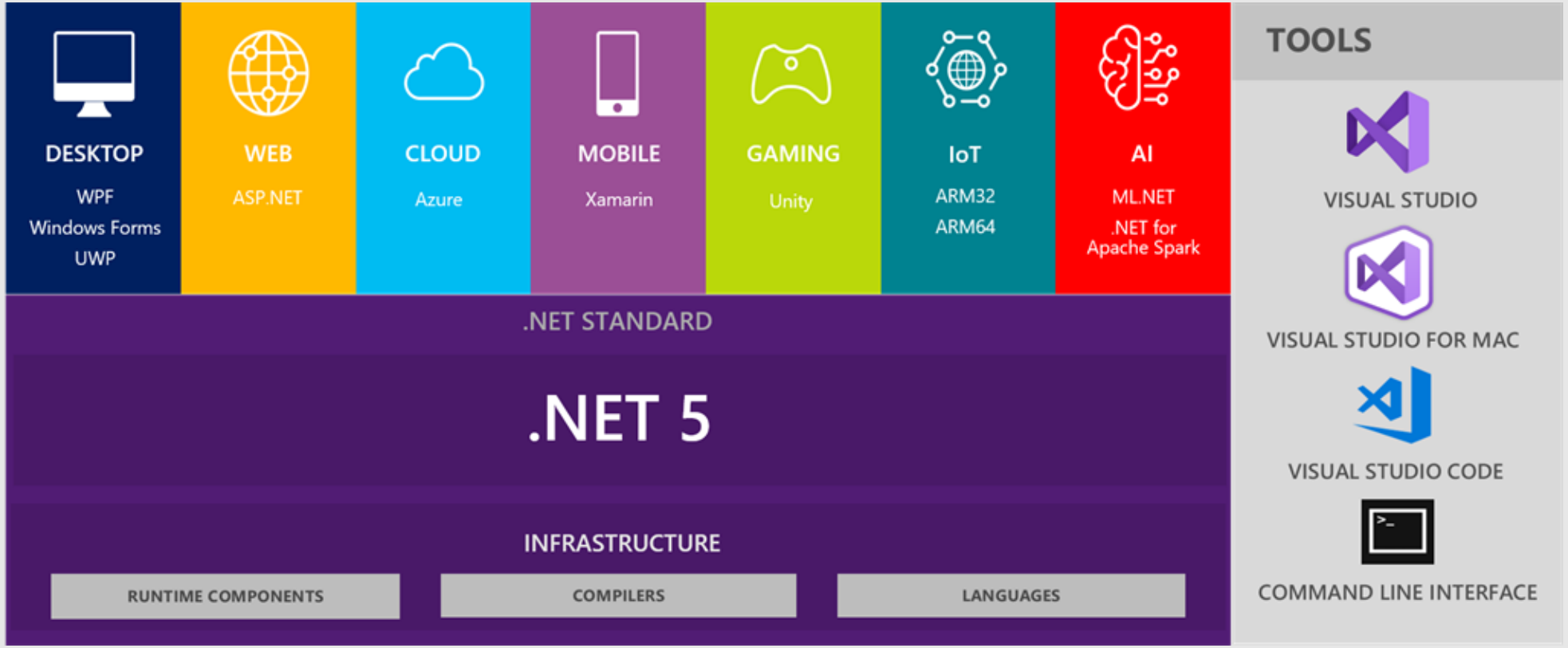
.NET 5

- 👉 The latest version of .NET Core is version 3.1, and it is on “Long Term Support” until December 3, 2022.
- 👉 The latest version of the .NET Framework is version 4.8
- 👉 Microsoft has planned to merge these two versions into a single entity, which is named .NET 5

WHAT IS .NET 5?

.NET

☞ .NET - A unified Platform



Dotnet CLI The command line tool for ASP.NET Core

.NET CLI

By Jafar Muzeyin

OBJECTIVES

You are going to learn:

- 👉 .NET CLI
- 👉 Commonly used commands
- 👉 .Using CLI to create .NET project

WHAT IS .NET CLI?

.NET CLI

- The dotnet CLI is a command-line interface (CLI) is a new tool for developing the .NET application.
- It is a cross-platform tool and can be used in Windows, MAC or Linux.

HOW TO USE .NET CLI?

USING .NET CLI

- The Dot Net CLI is installed as part of the Net Core SDK.
- The syntax of Dotnet CLI consists of three parts. The driver, the “verb”, and the “arguments”
  - dotnet [verb] [arguments]
- The Dot Net CLI is installed as part of the Net Core SDK.
- The driver is named “dotnet”
- The “verb” is the command that you want to execute. The command performs an action
- The “arguments” are passed to the commands invoked

COMMANLY USED COMMANDS

COMMAND	DESCRIPTION
new	Creates a new project, configuration file, or solution based on the specified template.
restore	Restores the dependencies and tools of a project.
build	Builds a project and all of its dependencies.
publish	Packs the application and its dependencies into a folder for deployment to a hosting system.
run	Runs source code without any explicit compile or launch commands.
test	.NET test driver used to execute unit tests.
vstest	Runs tests from the specified files.
pack	Packs the code into a NuGet package.
migrate	Migrates a Preview 2 .NET Core project to a .NET Core SDK 1.0 project.
clean	Cleans the output of a project.
sln	Modifies a .NET Core solution file.
help	Shows more detailed documentation online for the specified command.
store	Stores the specified assemblies in the runtime package store.

CREATING ASP.NET PROJECT USING DOTNET CLI

DOTNET CLI

- Open the command prompt or Windows PowerShell and create a Folder named "HILCOE"
- dotnet new command is used to create the new project. The partial syntax is as follows
- dotnet new <TEMPLATE> [--force] [-i|--install] [-lang|--language] [-n|--name] [-o|--output]

CREATING ASP.NET PROJECT USING DOTNET NEW

DOTNET NEW

- The following command creates a new dotnet project using the TEMPLATE
- dotnet **new** <TEMPLATE>

LIST OF TEMPLATES

TEMPLATE	DESCRIPTION
console	Console Application
classlib	Class library
mstest	Unit Test Project
xunit	xUnit Test Project
web	ASP.NET Core Empty
mvc	ASP.NET Core Web App (Model-View-Controller)
razor	ASP.NET Core Web App
angular	ASP.NET Core with Angular
react	ASP.NET Core with React.js
reactredux	ASP.NET Core with React.js and Redux
webapi	ASP.NET Core Web API

RESTORING DEPENDENCIES

DOTNET RESTORE

- Once we created the new project, we have to download the dependencies. This is done using the restore command
- Dotnet restore

RUNNING THE APPLICATION

DOTNET RUN

- Use dotnet run to start the application

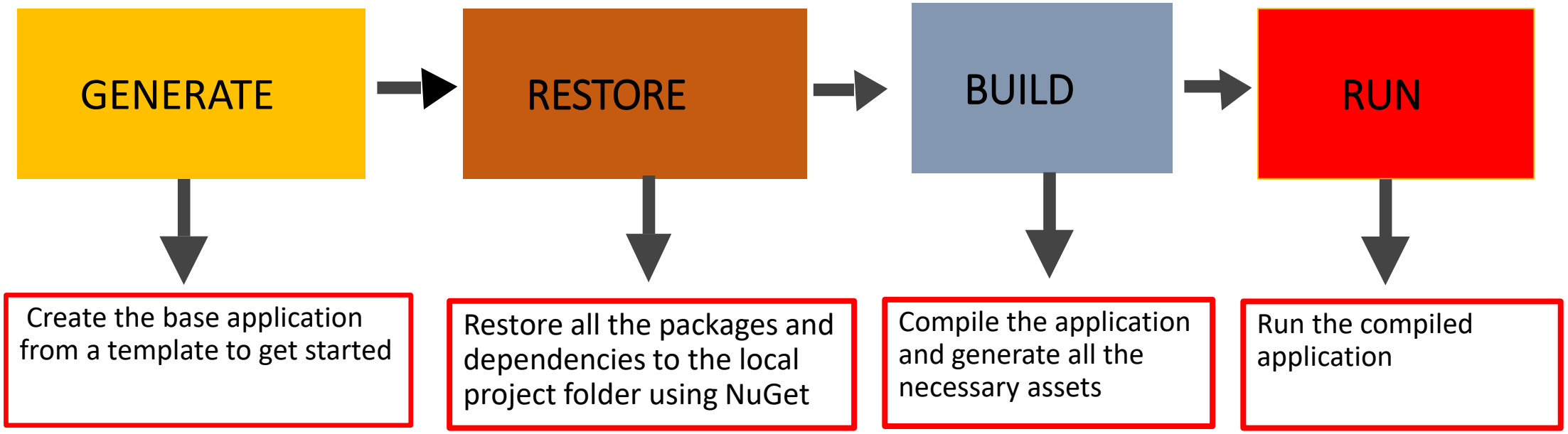
Creating Asp.Net Application

OBJECTIVES

You are going to learn:

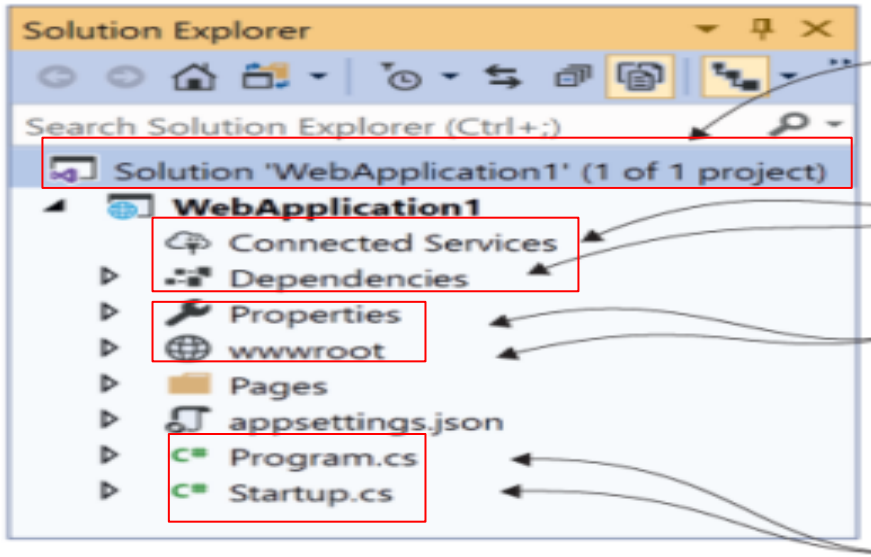
- 👉 Project Layout
- 👉 Kestrel Web server
- 👉 Kestrel Vs IIS

CREATING ASP.NET APPLICATIONS



PROJECT LAYOUT

Solution explorer from Visual Studio



The root project folder is nested in a top-level solution directory.

The Connected Services and Dependencies nodes do not exist on disk.

The wwwroot and Properties folders are shown as special nodes in Visual Studio, but they do exist on disk.

Program.cs and Startup.cs control the startup and configuration of your application at runtime.

The .csproj file contains all the details required to build your project, including the NuGet packages used by your project.

RUNNING THE .CS PROJECT FILE

.CSPROJECT

- 👉 Defines the type of project being built (web app, console app, or library)
- 👉 Platform the project targets (.NET Core 3.1, .NET 5.0, and so on),
- 👉 NuGet packages the project depends on.

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
</Project>
```

→ The SDK attribute specifies the type of project you're building

→ The Target Framework is the framework you'll run on, in this case, .NET 5.0

ROOT AND PROPERTIES FOLDER

💥 www root folder

- 👍 The only folder in the application that browsers are allowed to directly access when browsing the web app
- 👍 It stores CSS, JavaScript, images, or static HTML files and browsers will be able to access them
- 👍 Browser won't be able to access any file that lives outside of wwwroot

💥 Properties folder

- 👍 Contains all the information required to lunch the application
- 👍 Browser Configuration details about what action to perform when the application is executed and contains details like IIS settings, application URLs, authentication, SSL port details, etc.

DEPENDENCIES AND APPSETTINGS

💥 **Dependencies & Connected services**

👍 Dependency is contain collection of all the dependencies, such as NuGet packages

👍 Connected services contain remote services that the project relies on.

💥 **appsettings.json and appsettings.Development.json**

👍 Provide configuration settings that are used at runtime to control the behavior the app.

👍 configuration details like logging details, database connection details.

OVERVIEW

- The program class creates a web server in its Main method, while the starup class configure services and the application's request pipeline
    - Program.cs
    - Starup.cs

PROGRAM CLASS

PROGRAM.CS

- 👉 All .Net Core Applications are console applications. The other type of applications like MVC, SPA etc. are built on console application.
- 👉 The console application starts with Program.cs which must contain static void main method
- 👉 Main method called whenever the application starts
- 👉 It is the entry point of the application. This main method will create a host, build and run it. This host is a web server that will listen for HTTP Requests.

PROGRAM CLASS

PROGRAM.CS

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args)
            .Build()
            .Run();
    }
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
    }
}
```

a static method that configures, builds, and returns a Host object.

1. Create an IHostBuilder using the CreateHostBuilder method
2. Build and return an instance of IHost from the IHostBuilder
3. Run the IHost and start listening for requests and generating responses.
4. Create an IHostBuilder using the default configuration.
5. Configure the application to use Kestrel and listen to HTTP requests.
6. The Startup class defines most of your application’s configuration.

PROGRAM CLASS

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });
```

- 👍 The above method uses the WebHost class
- 👍 **CreateDefaultBuilder** method of the **WebHost** class is responsible for initializing the **IHostBuilder instance** with the required configurations.

💥 The jobs performe:
by **CreateDefaultBuilder** are:

- 👍 Configure Kestrel as a web server
- 👍 Set application root directory using **Directory.GetCurrentDirectory()**
- 👍 Load configuration
- 👍 Enable logging
- 👍 **Kestrel** integration running with IIS

THE STARTUP CLASS

.CSPROJECT

- Startup class is a simple class that does not inherit or implement any class or interface.
- It has two main functions:
  - Service registration—Any classes that your application depends on for providing functionality—both those used by the framework and those specific to your application—must be registered so that they can be correctly instantiated at runtime.
  - Middleware and endpoints—How your application handles and responds to requests.

THE STARTUP CLASS

CONFIGURE SERVICE

- The ConfigureServices method allows us to add or register services to the application.
- Other parts of the application may need these services for dependency injection.
- The ConfigureServices method needs instances of the services.
- Instances of the services will be injected into the ConfigureServices method through Dependency Injection.

```
public void ConfigureServices(IServiceCollection services) {

}
```

→ Configure services by registering them with the IServiceCollection

THE STARTUP CLASS

CONFIGURE

- 👉 Configure method allows you to configure the HTTP Request Pipeline
- 👉 The HTTP Request Pipeline shows how the application should respond to HTTP Requests.
- 👉 The components that make up the request pipeline are called middleware.
- 👉 The configure method needs instances of IapplicationBuilder and HostingEnvironment. These two instances will be injected into Configure via Dependency Injector.
- 👉 We will add middleware to the IapplicationBuilder instance.

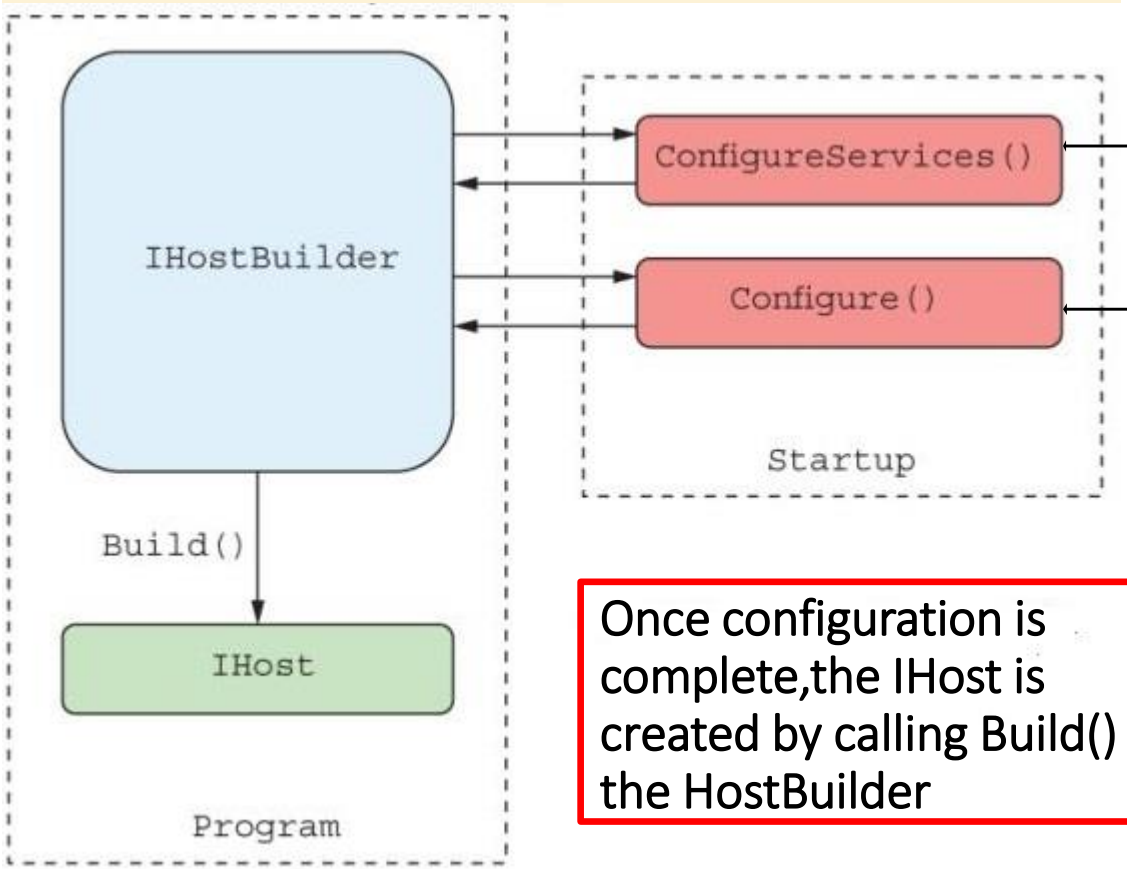
```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```

→ Configure the middleware pipeline for handling HTTP requests.

THE STARTUP CLASS

The HostBuilder calls out to Startup to configure your application

The IHost is created in Program using the builder pattern, and the CreateDefaultBuilder and CreateWebDefualts helper methods



To correctly create classes at runtime, dependencies are registerd with a container in the ConfigureService method.

The middleware pipeline is defined in the Configure method. It controls how the application responds to request.

Once configuration is complete,the IHost is created by calling Build() on the HostBuilder

kestrel Web Server for ASP.NET Core

KESTERAL

KESTERAL

- 👉 Kestrel is an open source, cross-platform, event-driven, and asynchronous I/O HTTP web server.
- 👉 It was developed to run ASP.NET Core applications on any platform.
- 👉 It is added by default in ASP.NET Core applications.

WHY WE USE KESTERAL?

IIS

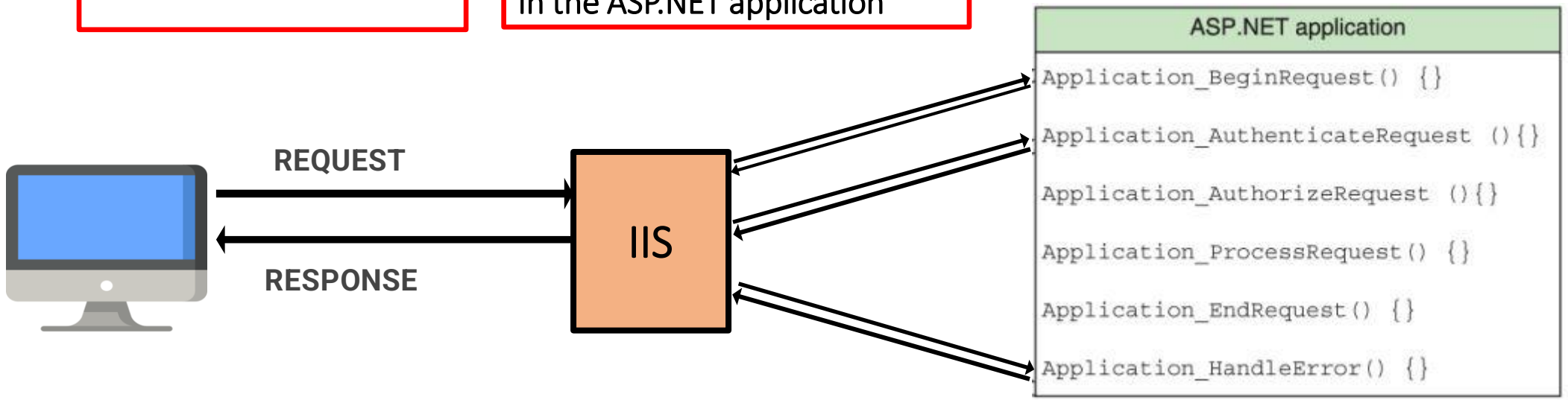
- 👉 Old ASP.NET applications are often tightly tied to **IIS** (Internet Information Service).
- 👉 IIS is a web server with all the features you need.
- 👉 It's been in development for quite some time and is very mature, but it's bulky and heavy
- 👉 It's become one of the best Web servers at the moment, but it's also one of the slowest.
- 👉 The new design of the ASP.NET Core application is now completely decoupled from IIS. This allows ASP.NET Core to run on any platform. But it can still listen for HTTP Requests and send the response back to the client. That's Kestrel.

HOW ASP.NET WORKS?

👉 **legacy ASP.NET app relly on IIS to invoke methods directly in your app**

IIS receives a request

IIS calls into specific methods in the ASP.NET application



Controllers transfers back and forth between IIS and ASP.NET as events raised

WHY WE USE KESTERAL?

KESTERAL

- Kestrel runs in-process in an ASP.NET Core application. So it runs independent of the environment.
- The Kestrel web server resides in the Microsoft.AspNetCore.Server.Kestrel library.
- **The Main** method calls **CreateDefaultBuilder** , which is responsible for creating a host for the application. (Host is the place where the application to run).
- **CreateDefaultBuilder** registers **Kestrel** as the server to use in the application.

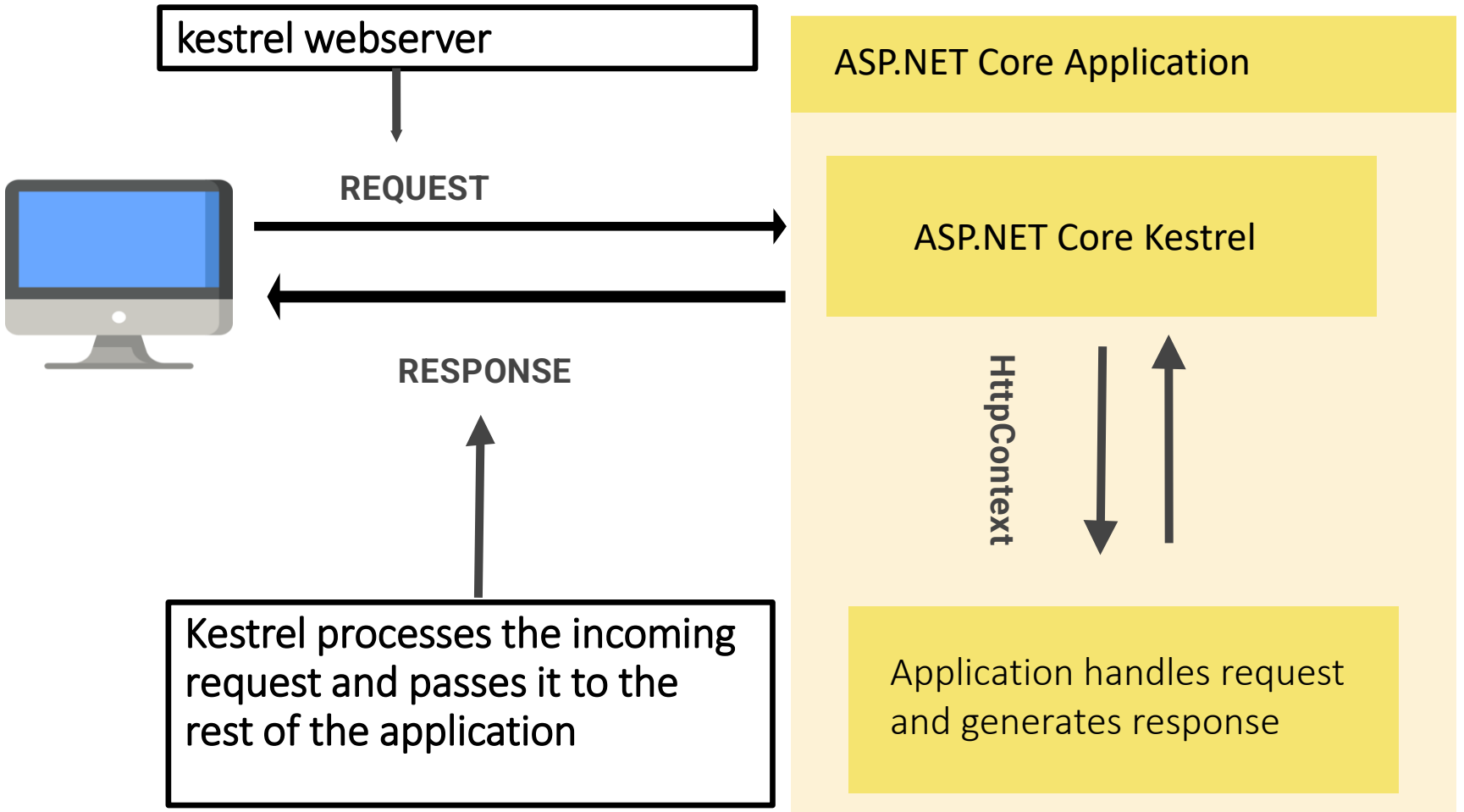
```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args)                                          1
            .Build()                                                     2
            .Run();                                                      3
    }
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)                                  4
            .ConfigureWebHostDefaults(webBuilder =>                      5
            {
                webBuilder.UseStartup<Startup>();                        6
            });
    }
}
```

WHY WE USE KESTERAL?

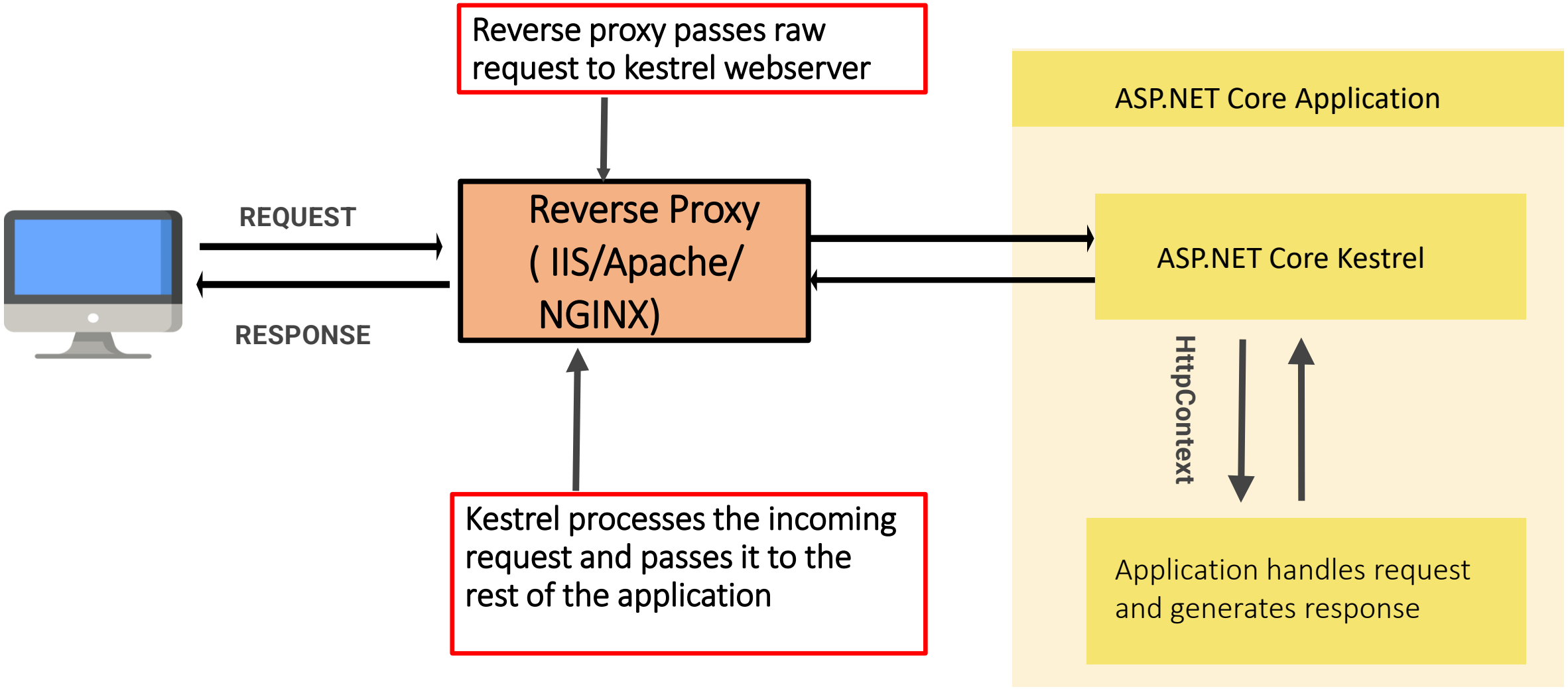
The **HttpContext** constructed by the ASP.NET Core web server is used by the application as a sort of storage box for a single request

Anything that's specific to this particular request and the subsequent

The web server fills the initial HttpContext with details of the original HTTP request and other configuration details and passes it on to the rest of the application.



REVERSE PROXY



HOW ASP.NET CORE WORKS?