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?

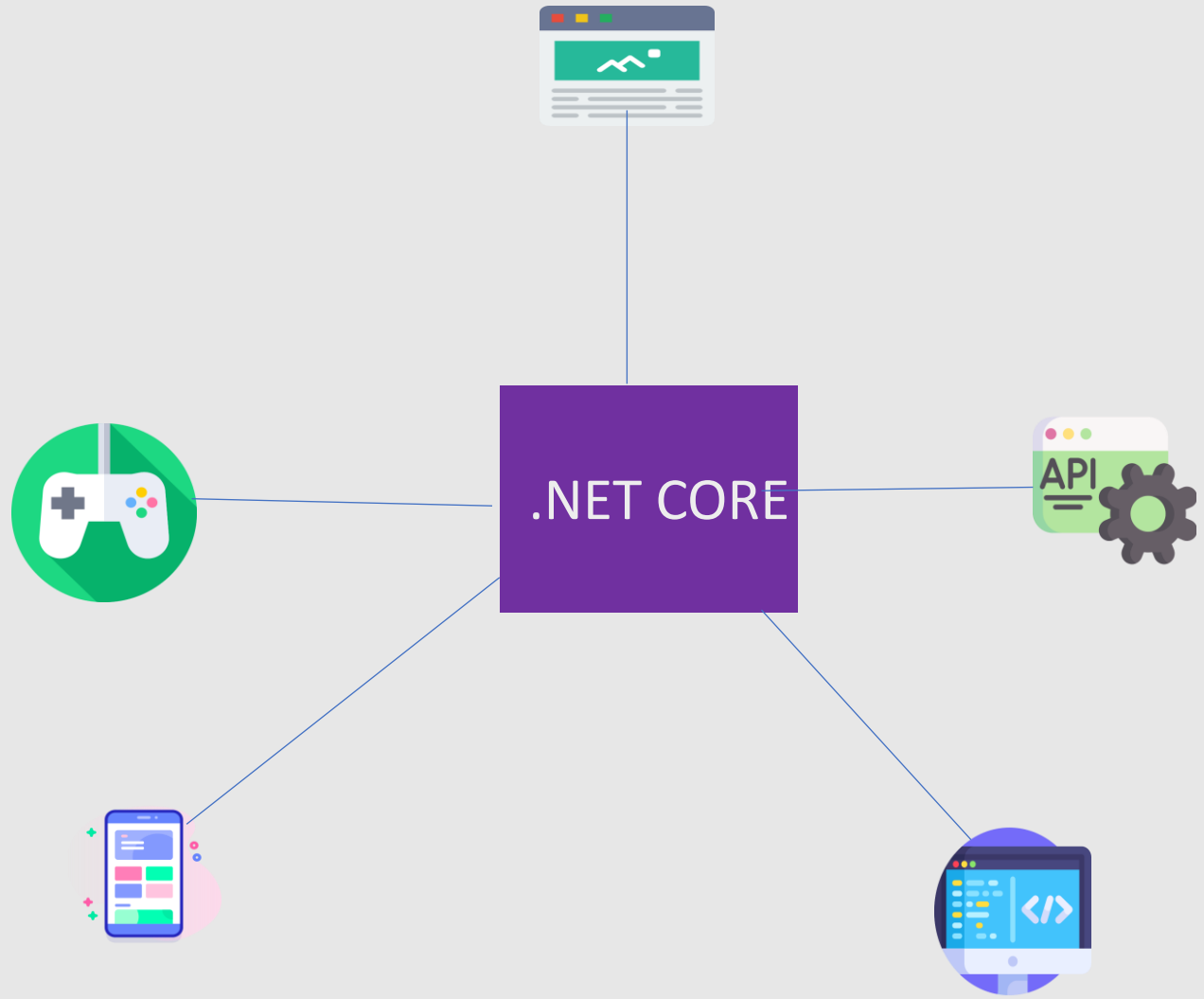
.NET STANDARD

- .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?

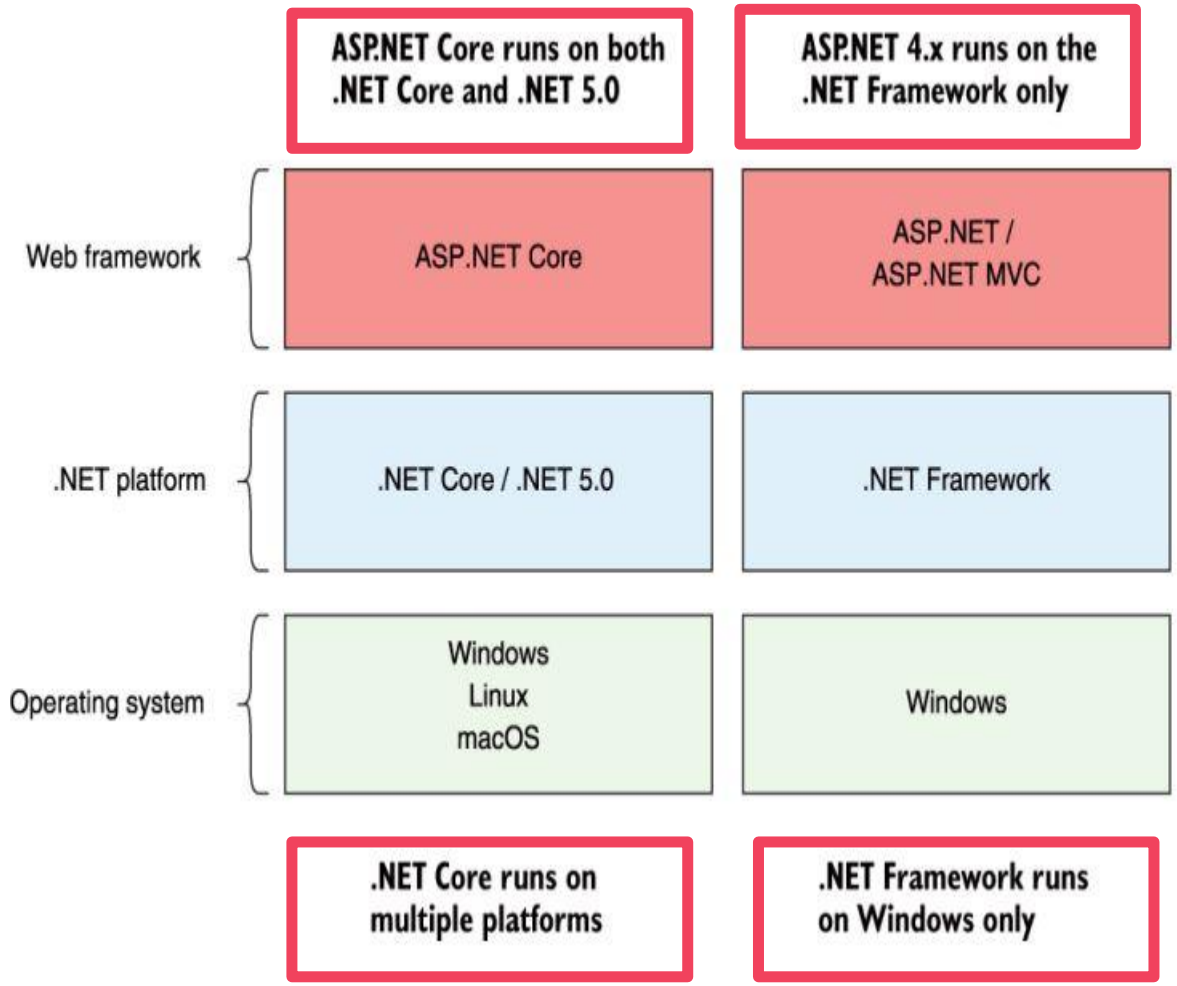
.NET CORE

- 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?

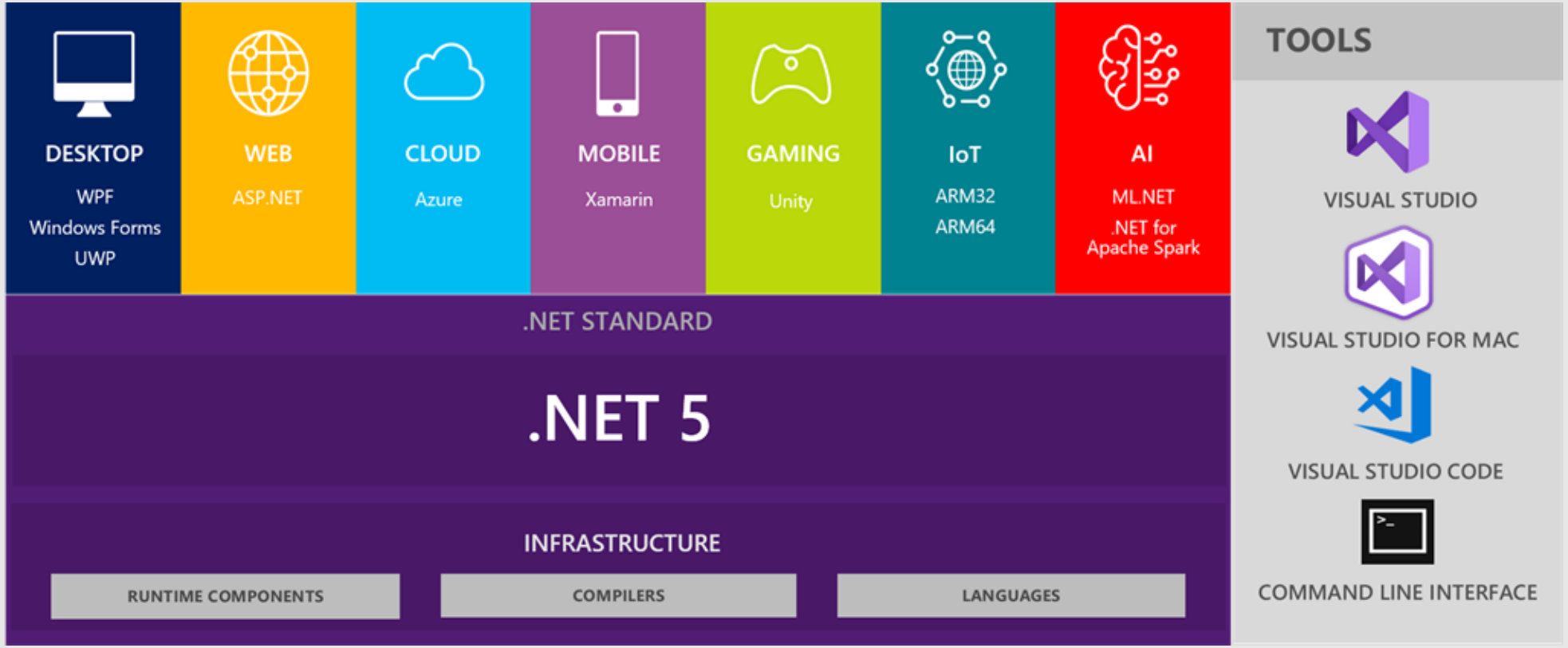
.NET 5

- 👉 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

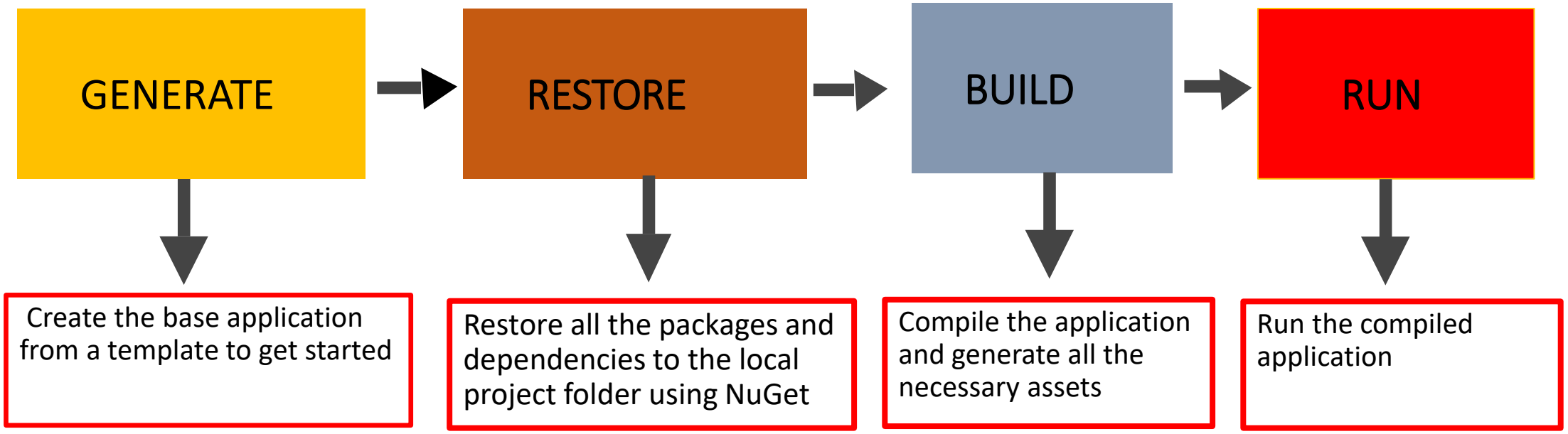
Creating Asp.Net Application

OBJECTIVES

You are going to learn:

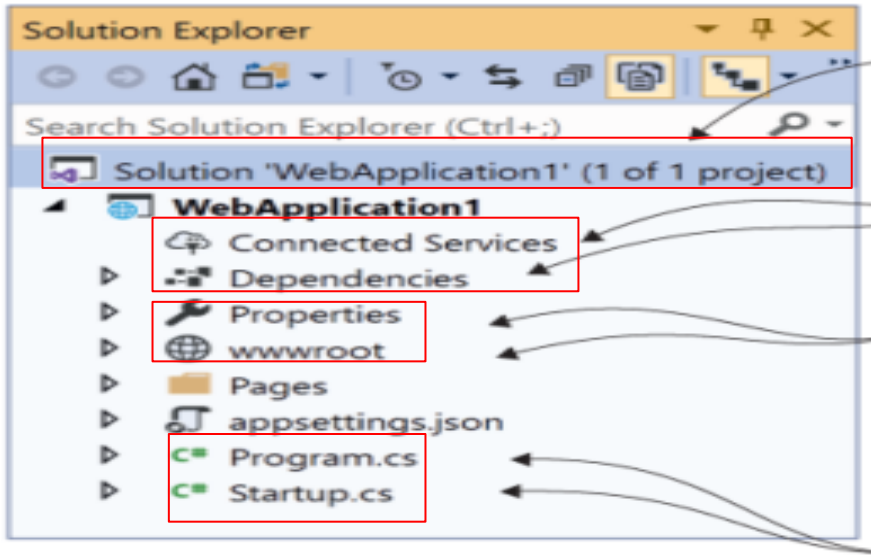
- 👉 Project Layout
- 👉 Kestrel Web server
- 👉 Kestrel Vs IIS

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

```
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 **WebHostBuilder 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

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.

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.

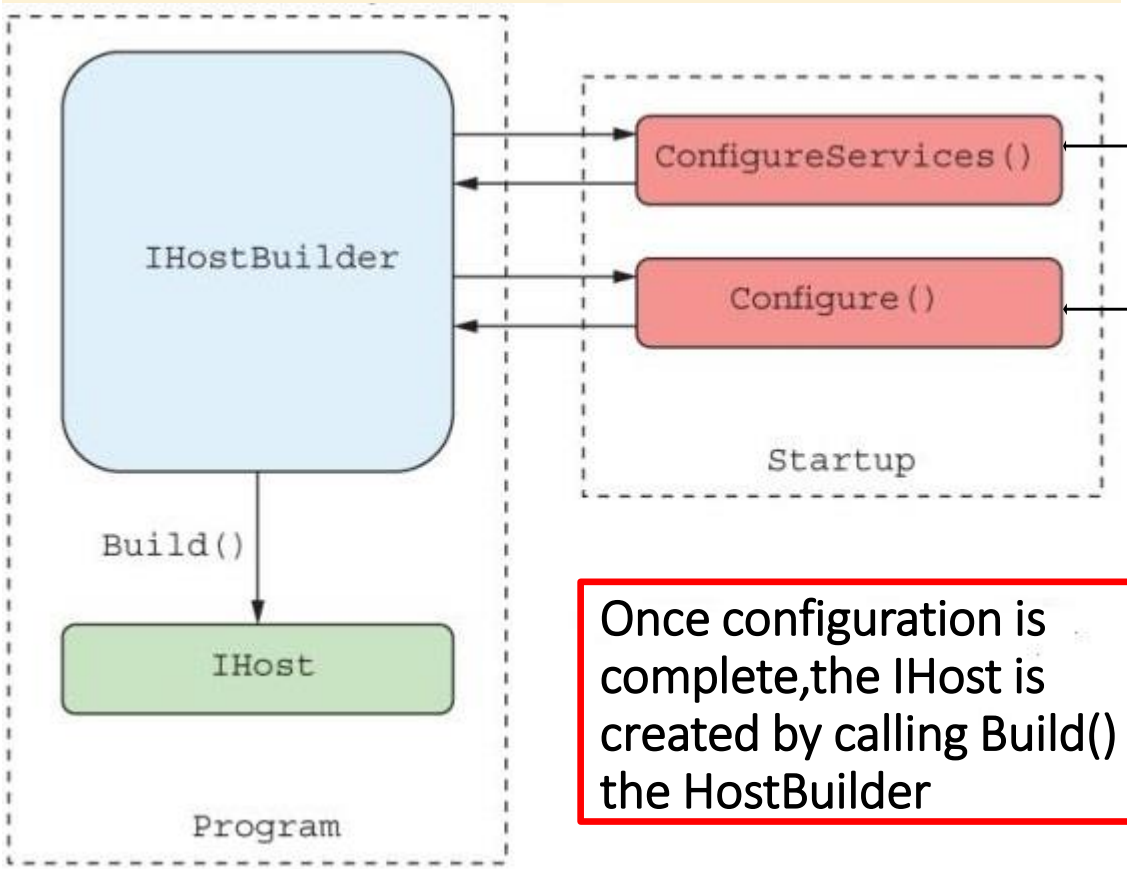
```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    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

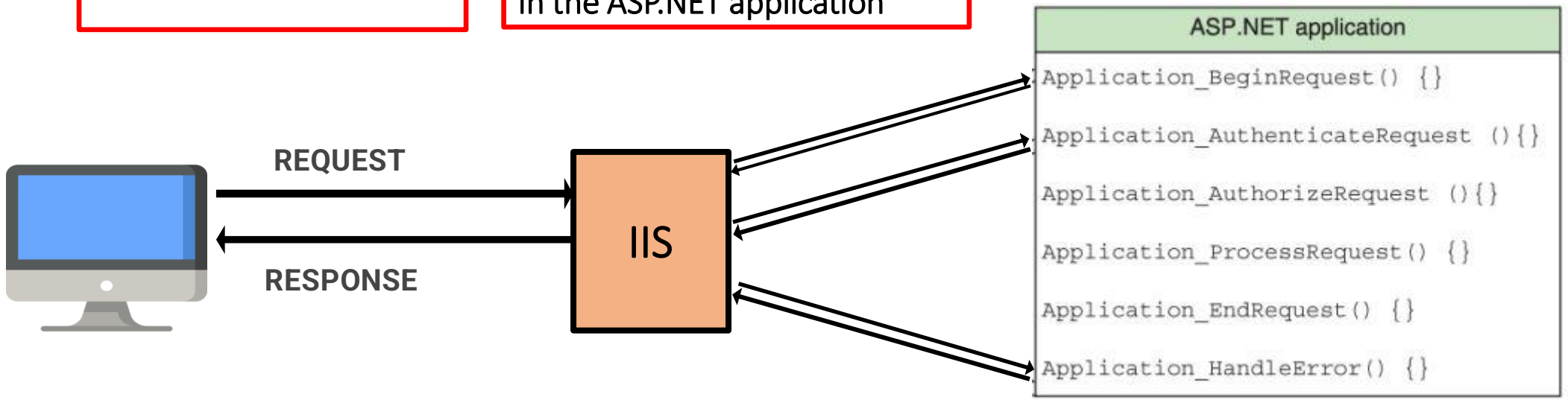
- 👉 Old ASP.NET applications are often tightly tied to **IIS** (Internet Information Service).
- 👉 IIS is a web server with all the features you need.
- 👉 It's been in development for quite some time and is very mature, but it's bulky and heavy
- 👉 It's become one of the best Web servers at the moment, 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.

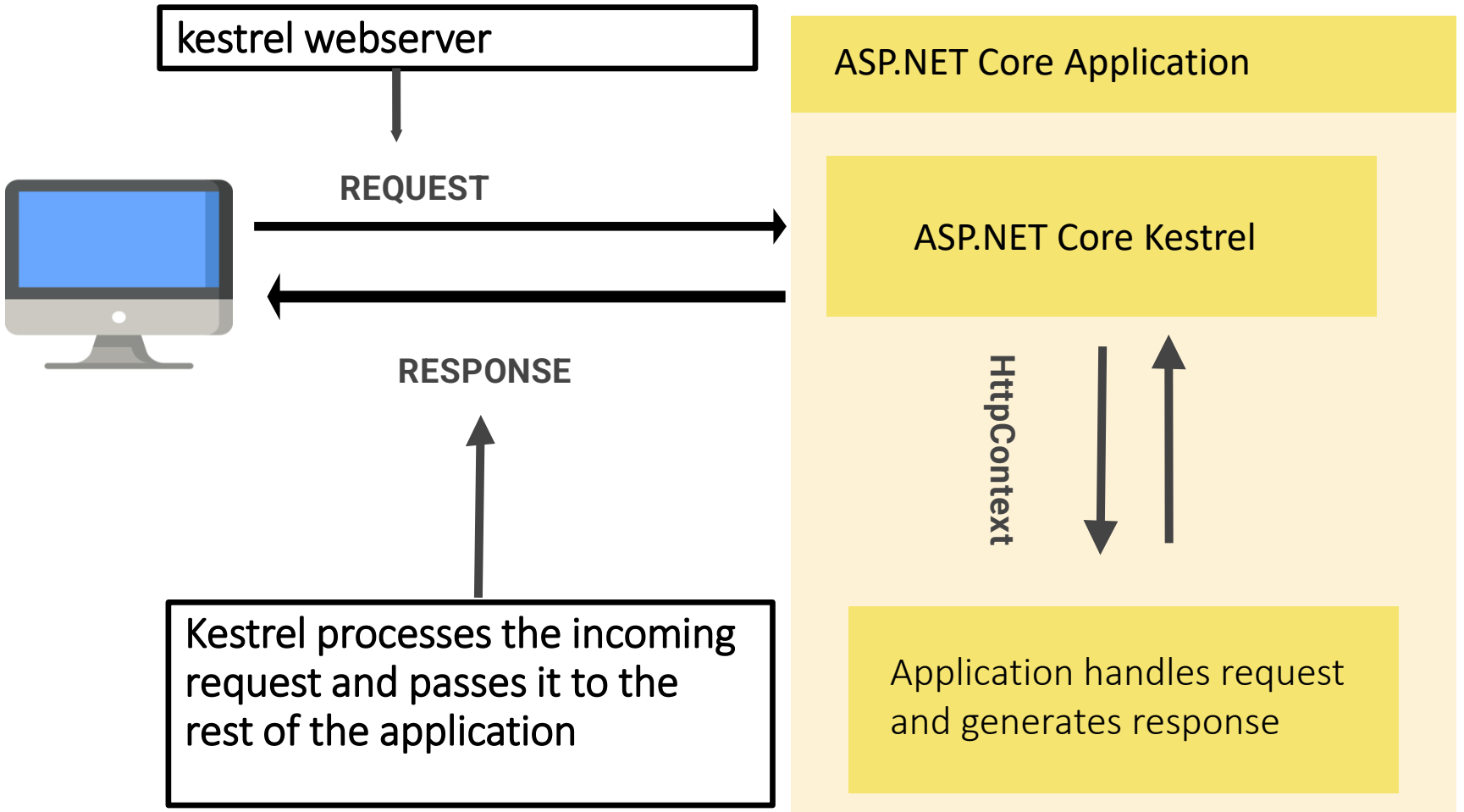
```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args)                                          1
            .Build()                                                     2
            .Run();                                                      3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

WHY WE USE KESTERAL?

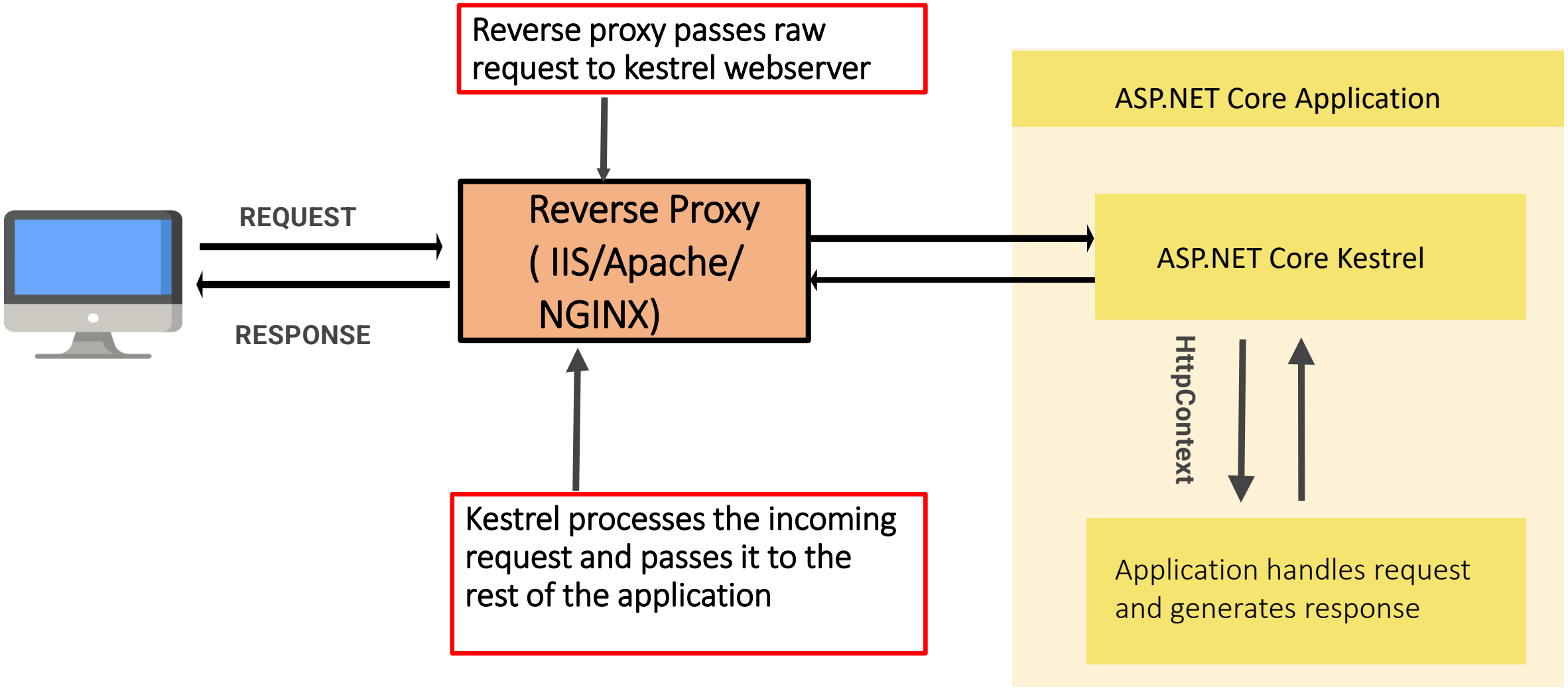
The **HttpContext** constructed by the ASP.NET Core web server is used by the application as a sort of storage box for a single request

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.



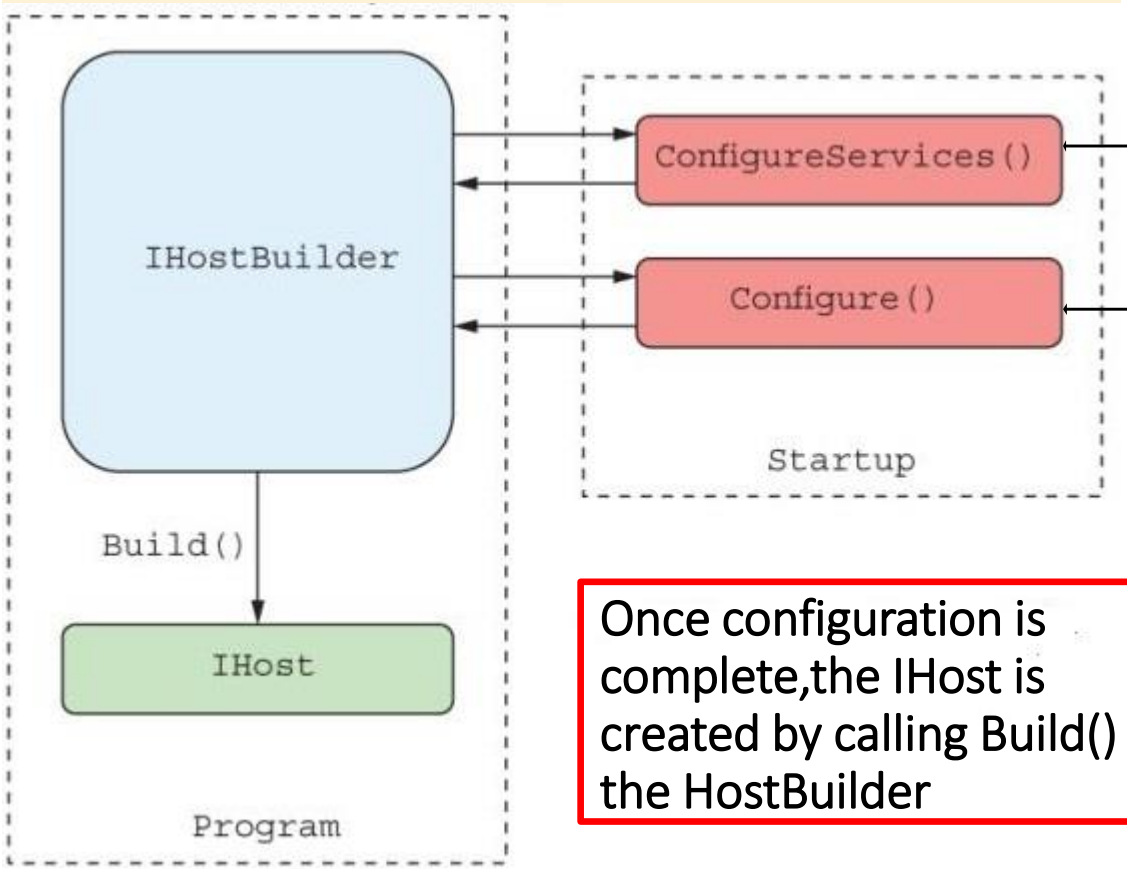
REVERSE PROXY



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

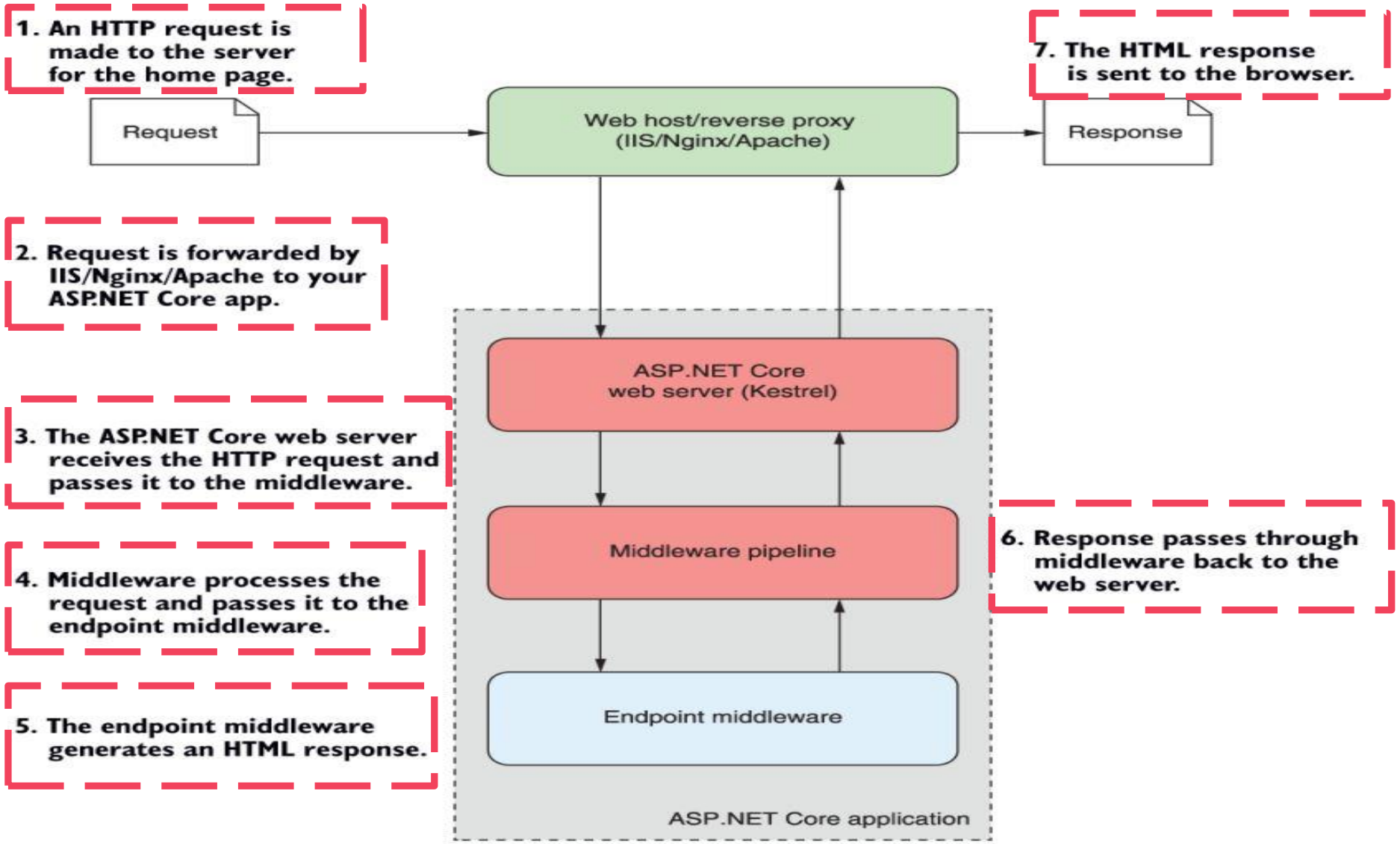


To correctly create classes at runtime, dependencies are registerd with a container in the ConfigureService method.

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

HOW ASP.NET CORE WORKS?



Dependency Injection

WHAT IS DEPENDENCY INJECTION

Dependency Injection

Without DI

Email Sender

Dependency Graph

Dependency Graph

IoC Container

Coupling

Coupling

loosely Coupled

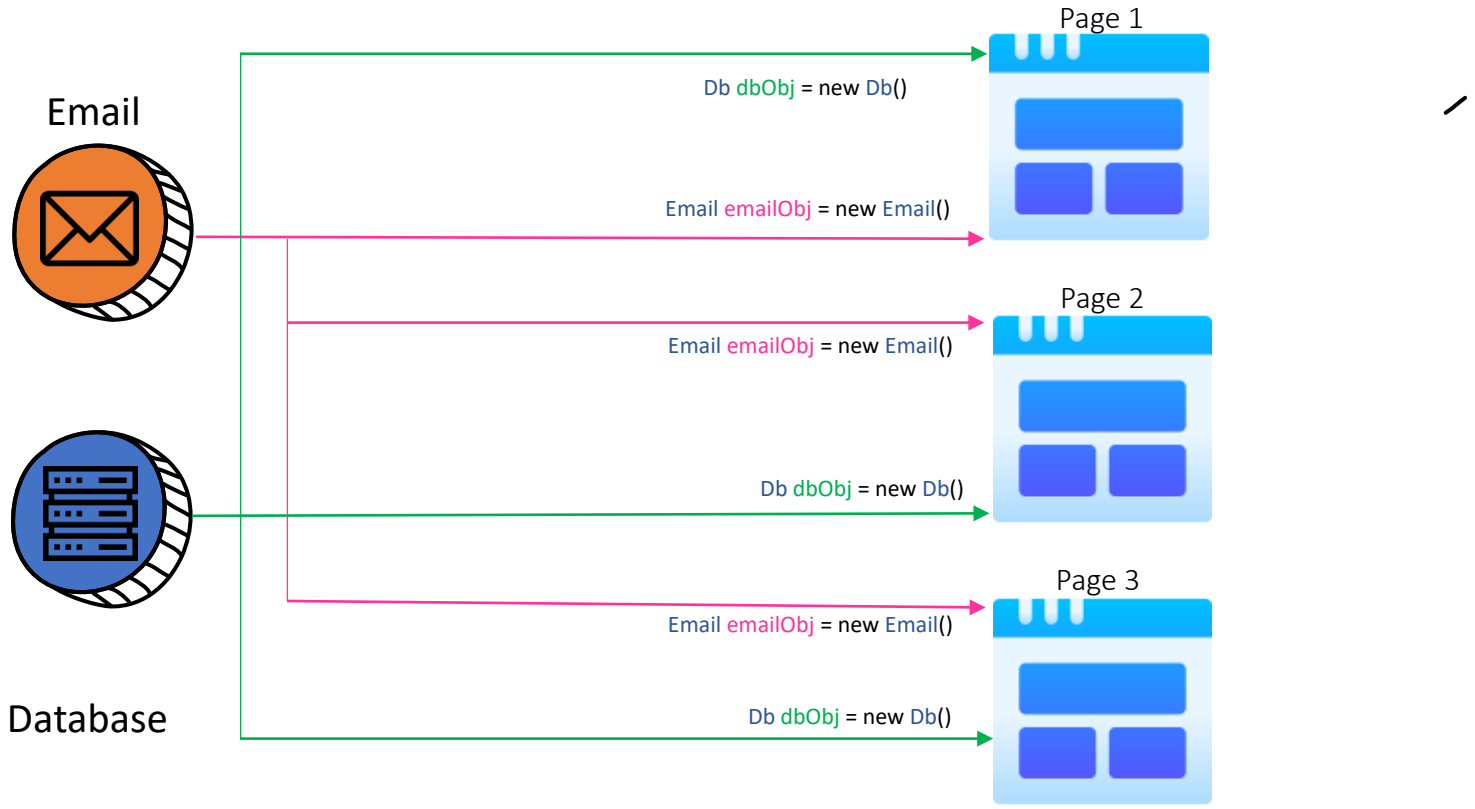
WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled

- DI is a design pattern that helps you develop loosely coupled code.
- Dependency injection allows for dependent objects of a class to be created in another class
- Dependency Injection is the fifth principle of S.O.L.I.D
  - Single Responsibility Principle
  - Open closed Principle
  - Liskov substitution Principle
  - Interface Segregation Principle
  - Dependency Inversion Principle

WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled



WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled

```
public class UserController : ControllerBase
{
    [HttpPost("register")]
    public IActionResult RegisterUser(string username)
    {
    var emailSender = new EmailSender();
    emailSender.SendEmail(username);
    return Ok();
    }

}
```

```
public class EmailSender
{
    public EmailSender() {
        Console.WriteLine($"Email sent to {username}!");
    }
}
```

WHAT IS DEPENDENCY INJECTION

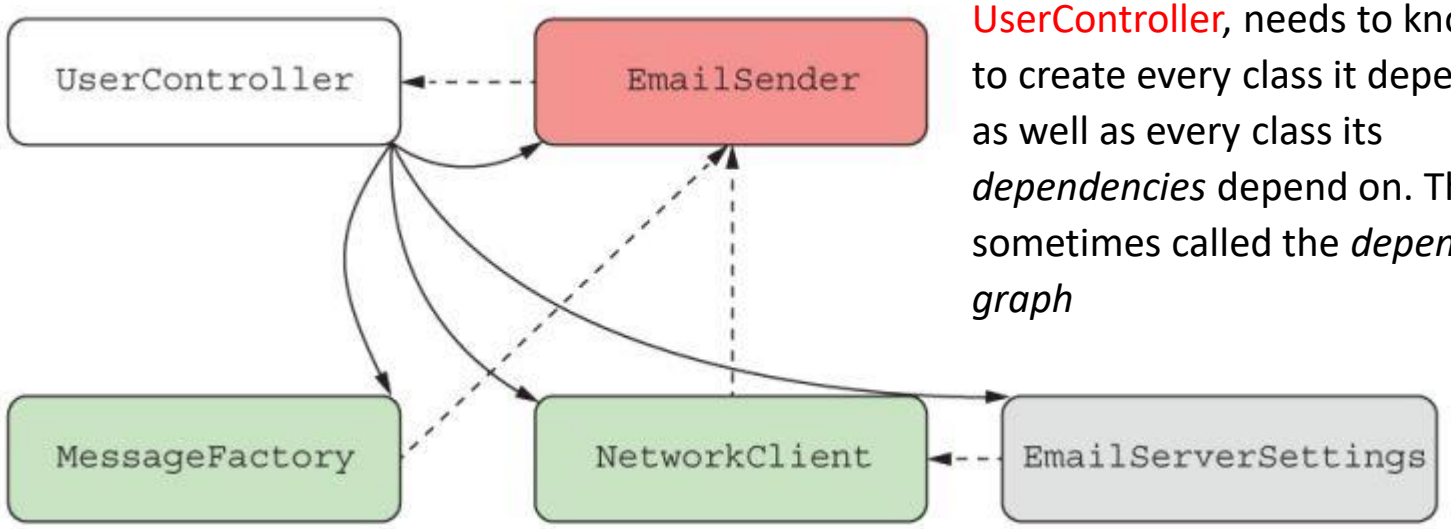
- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled

👉 In practice, EmailSender would need to do many things to send an email. It would need to

1. Create an email message
2. Configure the settings of the email server
3. Send the email to the email server

To use EmailSender, the user control must create all of the dependencies

The user control depends on the email sender



UserController, needs to know how to create every class it depends on, as well as every class its *dependencies* depend on. This is sometimes called the *dependency graph*

The EmailSender on MessageFactory

The NetworkClient on the EmailServerSettings

WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- Coupling
- Single-threaded
- loosely Coupled

UserController has an *implicit* dependency on the EmailSender class, as it manually creates the object itself as part of the RegisterUser method

```
public class UserController : ControllerBase
{
    [HttpPost("register")]
    public IActionResult RegisterUser(string username)
    {
        var emailSender = new EmailSender(
        new MessageFactory(),

        new NetworkClient(
        new EmailServerSettings

        (
                host: "smtp.server.com",

                port: 25
        ))
    );
        emailSender.SendEmail(username);
        return Ok();

    }

}
```

WHAT IS DEPENDENCY INJECTION

Dependency Injection

Without DI

Email Sender

Dependency Graph

Dependency Graph

IoC Container

IoC Container

Coupling

loosely Coupled

👉 Dependency injection aims to solve the problem of building a dependency graph by inverting the chain of dependencies.

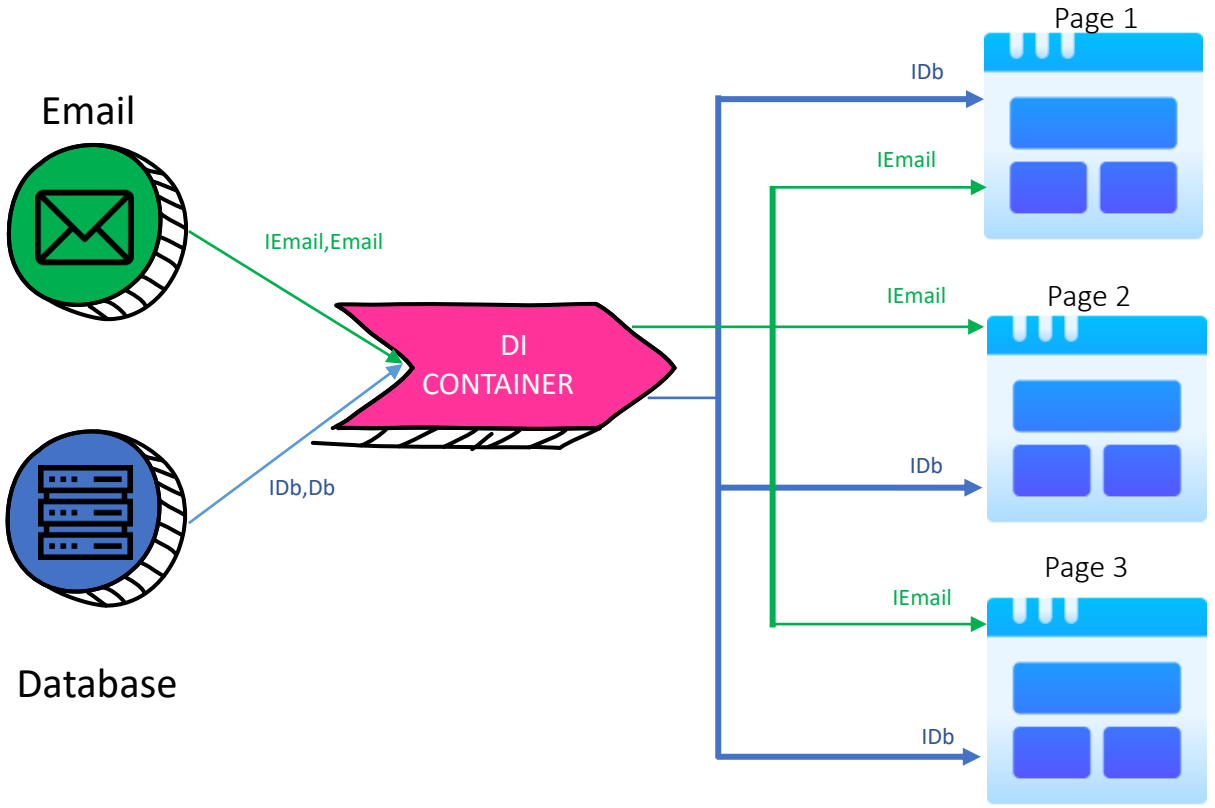
👉 The service responsible for creating an object is called a *DI container* or an *IoC container*

👉 The *DI container* or *IoC container* is responsible for creating instances of services

👉 It knows how to construct an instance of a service by creating all its dependencies and passing to the constructor.

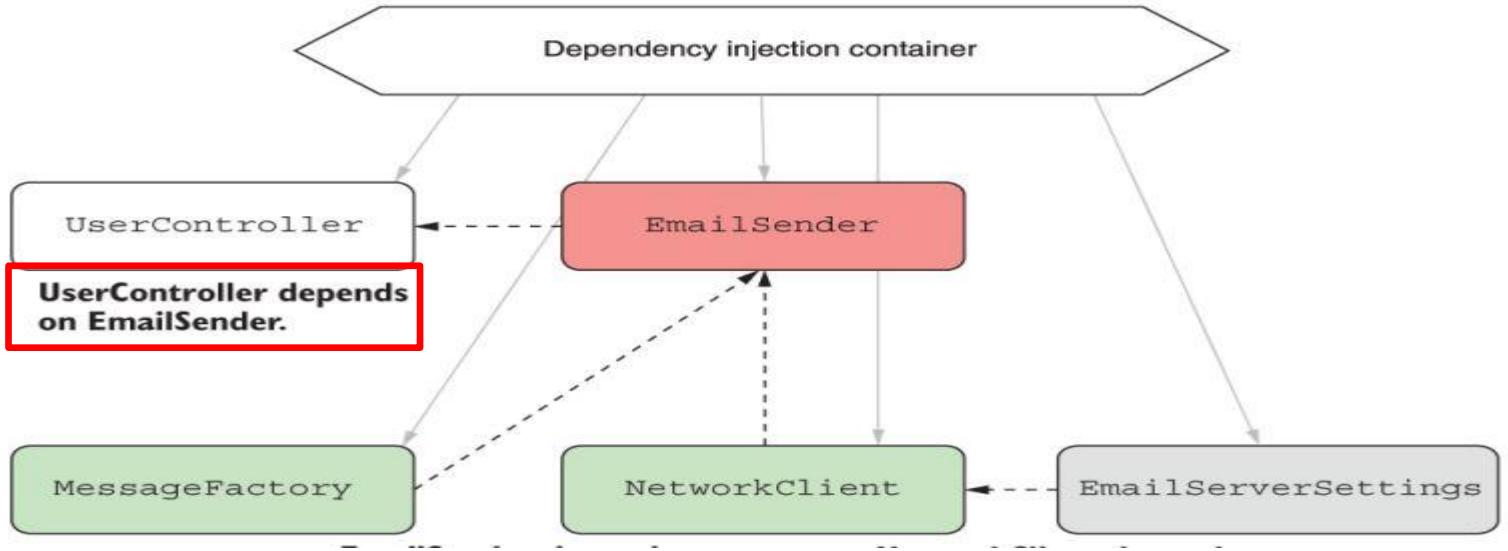
WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled



WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled

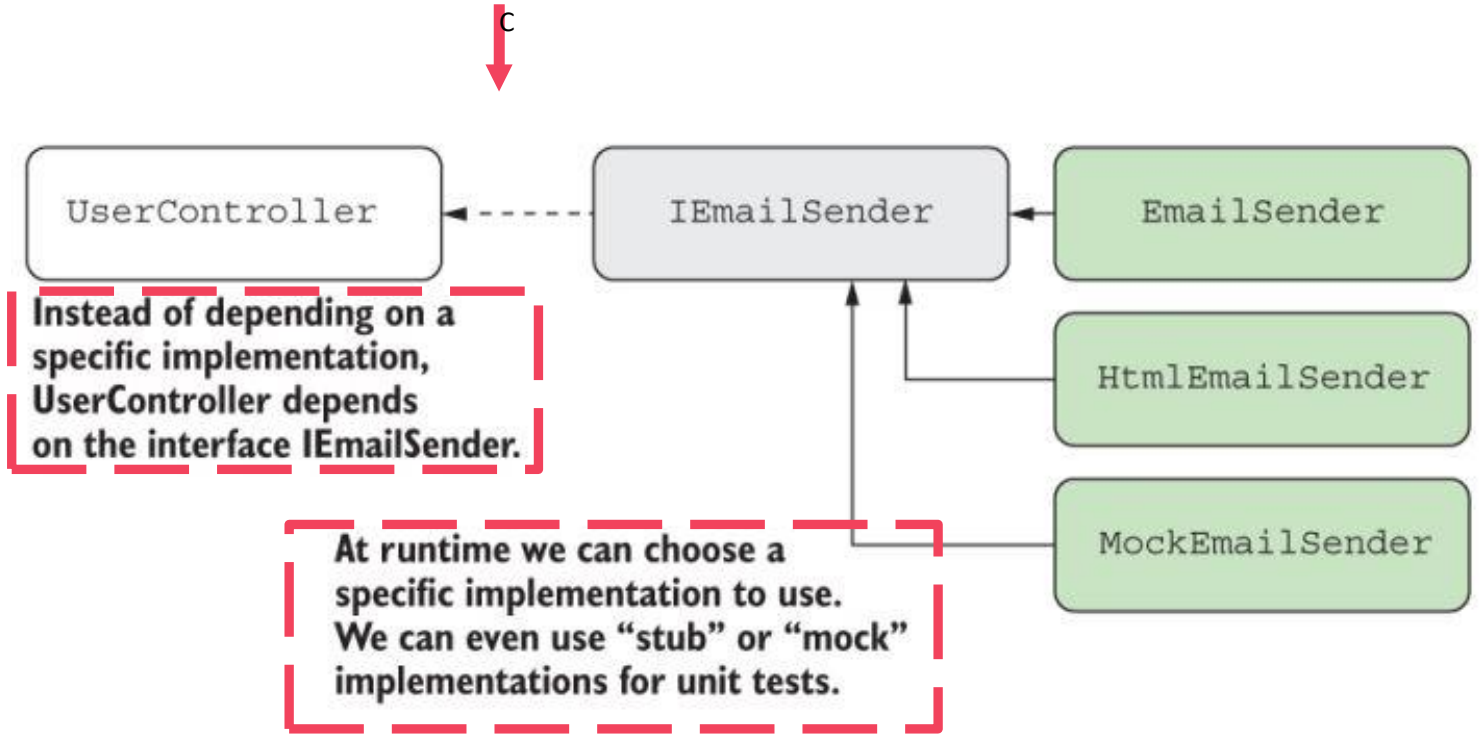



```
public class UserController : ControllerBase
{
private readonly EmailSender  emailSender;
public UserController(EmailSender emailSender)
{
    _emailSender = emailSender;
}
[HttpPost("register")]
public IActionResult RegisterUser(string username)
{
    _emailSender.SendEmail(username);
    return Ok();
}
```

WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled

- **Coupling**: refers to how a given class depends on other classes to perform its function
- Coding to interfaces is a common design pattern that helps further reduce the coupling of a system, as you're not tied to a single implementation.



WHAT IS DEPENDENCY INJECTION

- Dependency Injection
- Without DI
- Email Sender
- Dependency Graph
- Dependency Graph
- IoC Container
- IoC Container
- Coupling
- loosely Coupled

```
public class UserController : ControllerBase
{
private readonly EmailSender _emailSender;
public UserController(IEmailSender emailSender)
{
    _emailSender = emailSender;
}
[HttpPost("register")]
public IActionResult RegisterUser(string username)
{
    _emailSender.SendEmail(username);
    return Ok();
}
}
```

- How does the application know to use EmailSender in production instead of DummyEmailSender?

- The process of telling your DI container “when you need IEmailSender, use EmailSender” is called *registration*.

Dependency Injection .NET Core

WHAT DEPENDENCY INJECTION IN ASP.NET CORE

- Dependency Injection
- Framework Service
- Own Services
- Own Services
- Service Lifetime
- AddSingleton
- AddScoped
- Single-threaded

WHAT IS DEPENDENCY INJECTION IN ASP.NET CORE

- Dependency Injection
- Framework Service
- Own Services
- Own Services
- Service Lifetime
- AddSingleton
- AddScoped
- Single-threaded

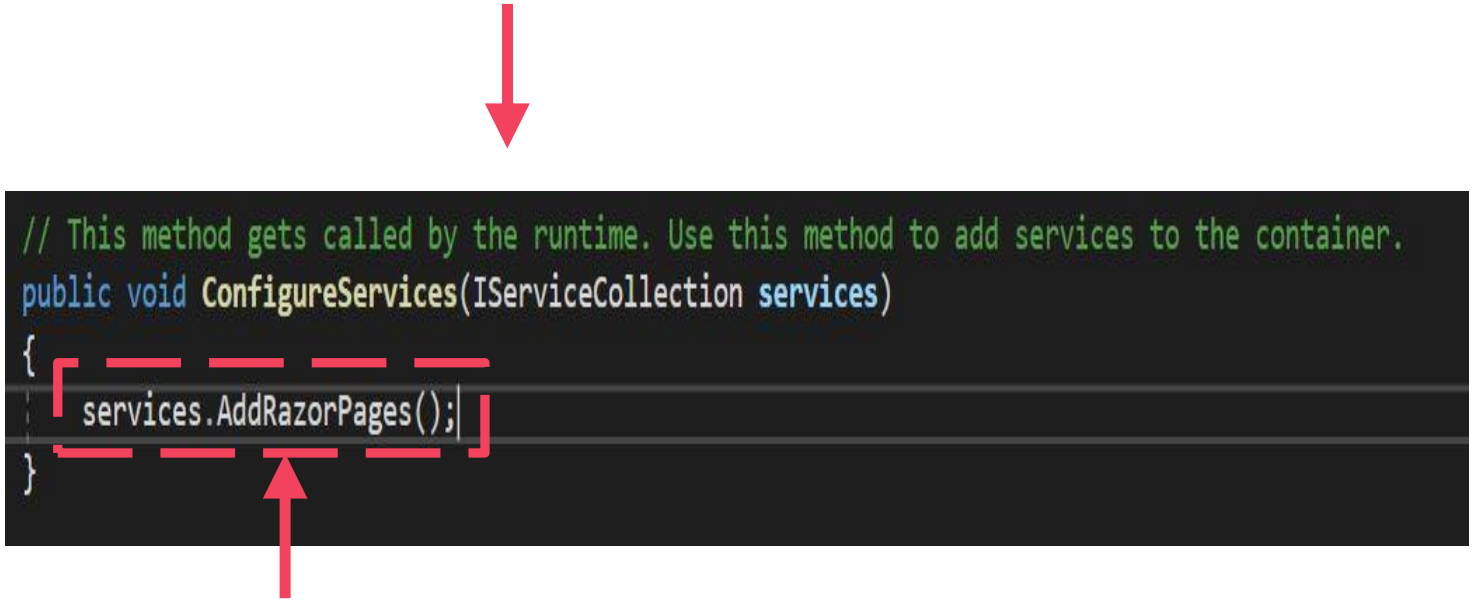
- DI is an integral part of ASP.NET Core (.NET 5)
- ASP.NET Core includes a simple DI container that all the framework libraries use to register themselves and their dependencies.
- The built-in container is represented by IServiceProvider
- Types of service in ASP.NET Core (.NET 5)
  - Framework Services
  - Application Services

HOW TO ADD ASP.NET CORE FRAMEWORK SERVICE TO THE CONTAINER?

- Dependency Injection
- Framework Service
- Own Services
- Own Services
- Service Lifetime
- AddSingleton
- AddScoped
- Single-threaded

👉 The dependency injection container is set up in the ConfigureServices method of your Startup class in Startup.cs.

👉 Registering the MVC services with the DI container

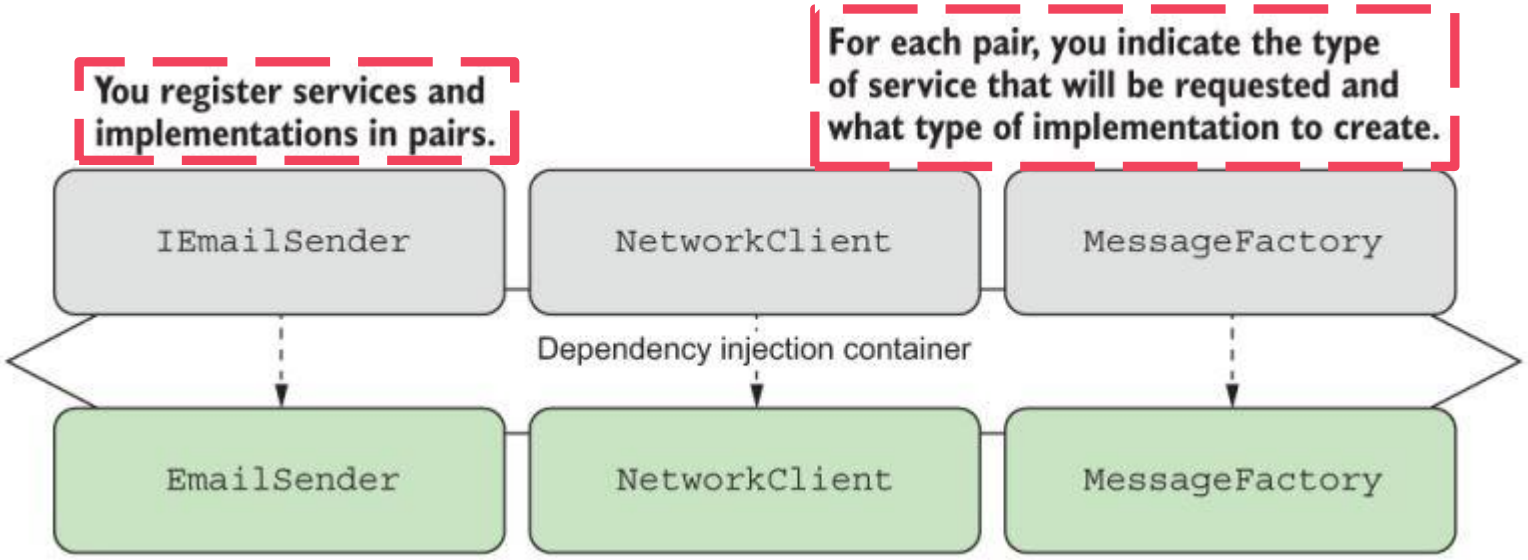


👉 The Razor Pages framework exposes the AddRazorPages() extension method. Invoke the extension method in ConfigureServices of Startup.

HOW TO REGISTOR OWN SERVICES WITH THE CONTAINER?

- Dependency Injection
- Framework Service
- Own Services
- Own Services
- Service Lifetime
- AddSingleton
- AddScoped
- Single-threaded

👉 In order to completely configure the application, you need to register EmailSender and all of its dependencies with the DI container



👉 When a service requires IEmailSender, use an instance of EmailSender.

👉 When a service requires NetworkClient, use an instance of NetworkClient.

👉 When a service requires MessageFactory, use an instance of MessageFactory.

HOW TO REGISTOR OWN SERVICES WITH THE CONTAINER?

- Dependency Injection
- Framework Service
- Own Services
- Own Services
- Service Lifetime
- AddSingleton
- AddScoped
- Single-threaded

👉 In practice, EmailSender would need to do many things to send an email. It would need to

1. You're using API controllers, so you must call AddControllers
2. Whenever you require an IEmailSender, use EmailSender.
3. Whenever you require a NetworkClient, use NetworkClient.
4. Whenever you require a MessageFactory, use MessageFactory.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();
    services.AddScoped<IEmailSender, EmailSender>();
    services.AddScoped<NetworkClient>();
    services.AddSingleton<MessageFactory>();
```

SERVICE LIFETIME

Dependency Injection

Framework Service

Own Services

Own Services

Service Lifetime

AddSingleton

AddScoped

Single-threaded

Singleton

Same instance for the life of application (unless restarted).

Scoped

Same Instance for one scope (one request in most cases).

Transient

Different instance every time the service is injected.

SERVICE LIFETIME

Dependency Injection

Framework Service

Own Services

Own Services

Service Lifetime

AddSingleton

AddScoped

AddTransient

👉 Syntax to Register : services.AddSingleton<>

👉 Singleton service sends same instance for the life of the application.

👉 E.g. If you click on all view or link on a website, whenever an instance is requested it will send same object. It will change only when application restarts.

SERVICE LIFETIME

Dependency Injection

Framework Service

Own Services

Own Services

Service Lifetime

AddSingleton

AddScoped

AddTransient

- Syntax to Register : services.AddScoped<>
- Scoped service sends a new instance for each request.
- E.g. If you click on a view or link for that page load if instance is requested 10 times it will send same object!

SERVICE LIFETIME

- Dependency Injection
- Framework Service
- Own Services
- Own Services
- Service Lifetime
- AddSingleton
- AddScoped
- AddTransient

👉 Syntax to Register : services.AddTransient<>

👉 Always try to register a service as transient if unsure.

👉 Transient service sends a new instance every time it is requested

👉 E.g. If you click on a view or link for that page load if instance is requested 10 times it will send 10 different objects!

.NET CORE PIPELINE

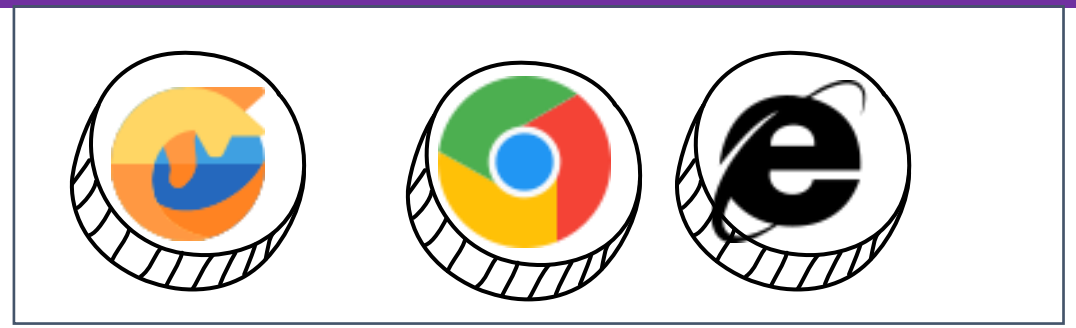
WHAT IS MIDDLEWARE ?

MIDDLEWARE

- Middleware are C# classes or functions that handle an HTTP request or response
- Middleware can
  - Handle an incoming HTTP *request* by generating an HTTP *response*
  - Process an incoming HTTP *request*, modify it, and pass it on to another piece of middleware
  - Process an outgoing HTTP *response*, modify it, and pass it on to either another piece of middleware or the ASP.NET Core web server
- They are chained together, with the output of one middleware acting as the input to the next middleware, to form a pipeline.

WHAT IS MIDDLEWARE ?

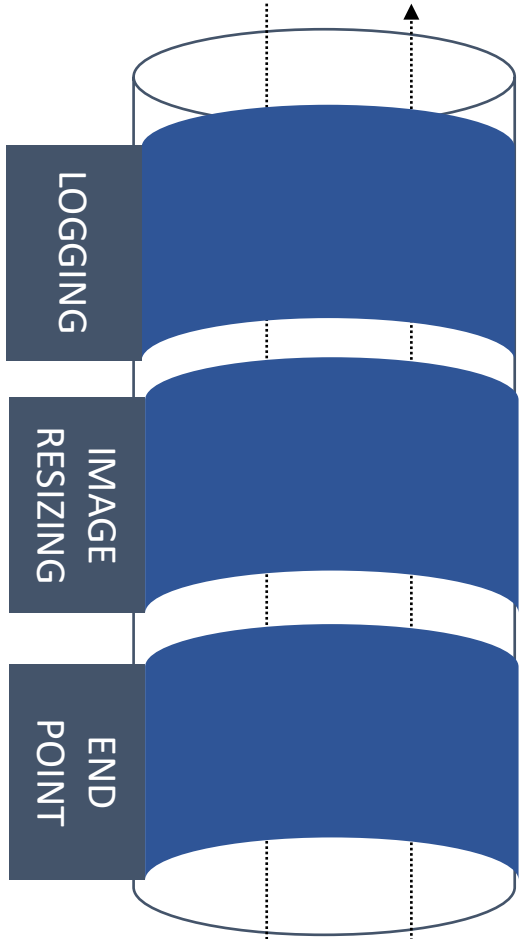
This arrangement, where a piece of middleware can call another piece of middleware, which in turn can call another, and so on, is referred to as a *pipeline*.



ASP.NET Core web server passes the request to the middleware pipeline.

The logging middleware notes down the time the request arrived and passes the request on to the next middleware.

If the request is for an image of a specific size, the image resize middleware will handle it. If not, the request is passed on to the next middleware.



The response is returned to ASP.NET Core web server.

The response passes through each middleware that ran previously in the pipeline.

If the request makes it through the pipeline to the endpoint middleware, it will handle the request and generate a response.

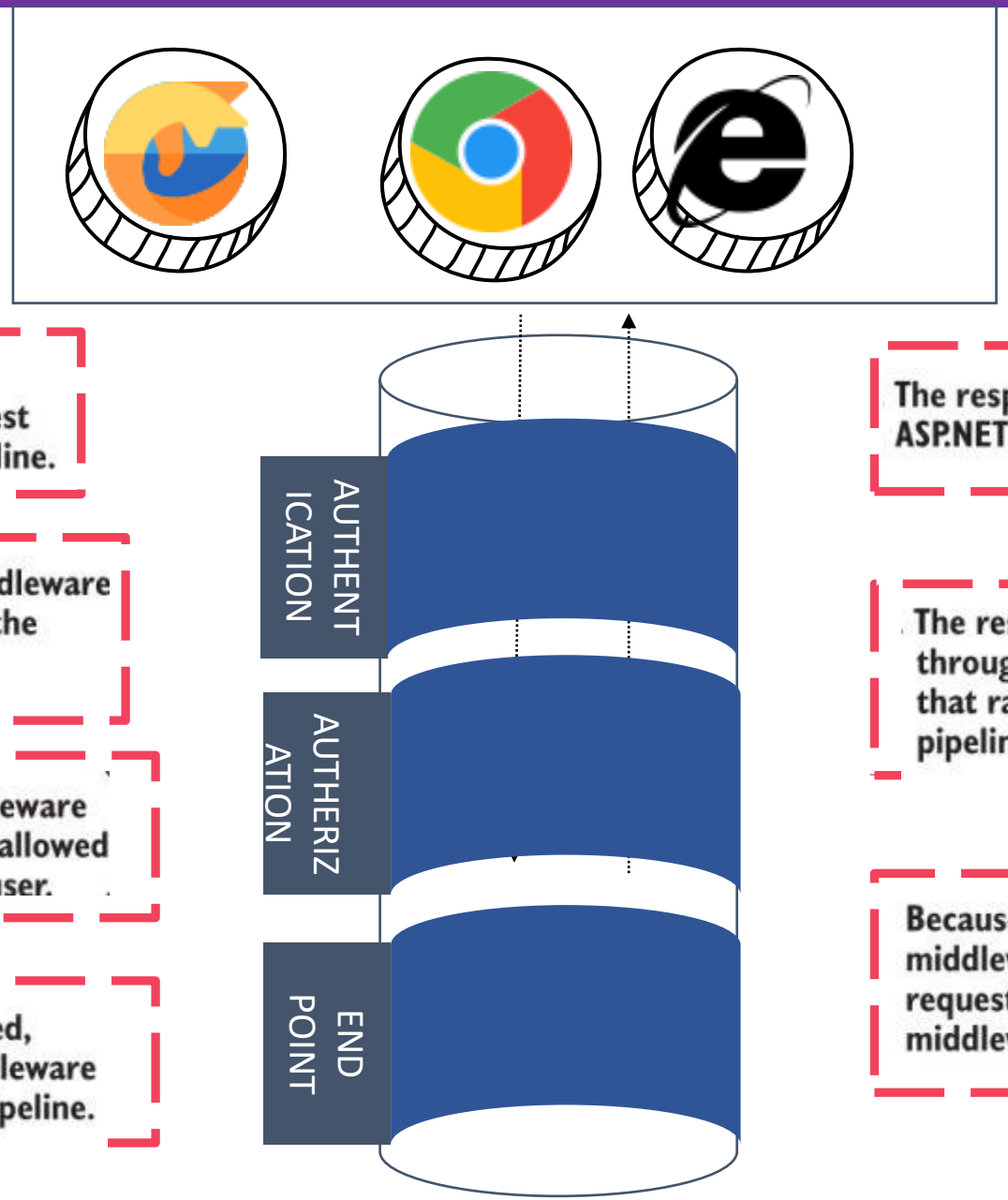
WHAT IS MIDDLEWARE ?

MIDDLEWARE

- In the examples, the middleware would receive a request, modify it, and then pass the request on to the next piece of middleware in the pipeline.

- Subsequent middleware could use the details added by the earlier middleware to handle the request in some way

WHAT IS MIDDLEWARE ?



CREATING A WEBSITE WITH ASP.NET MVC CORE

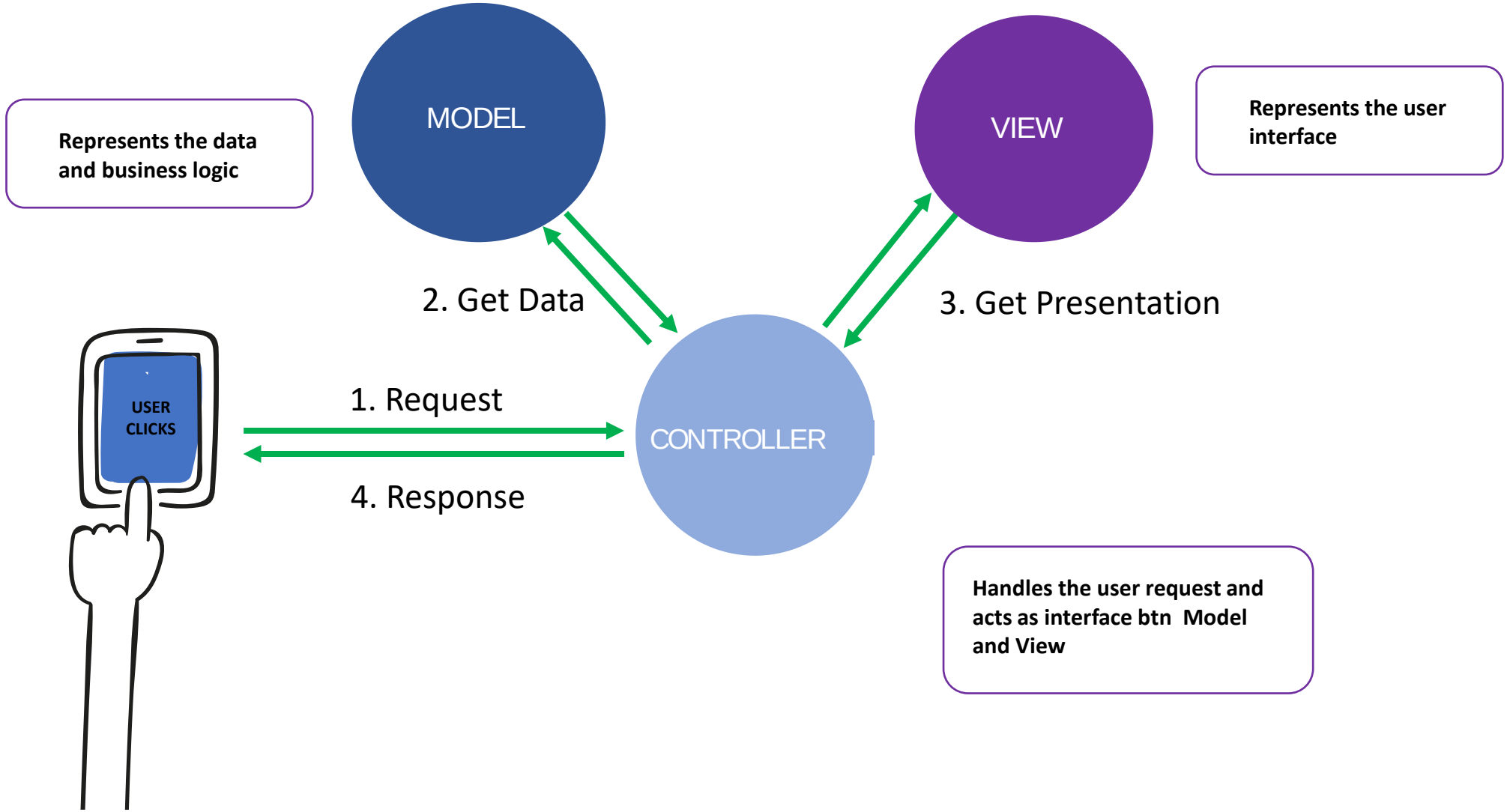
WHAT IS ASP.NET CORE MVC?

ASP.NET CORE MVC

ASP.NET CORE MVC : IS A WEB DEVELOPMENT FRAMEWORK COMBINES THE FEATURES OF MVC (MODEL-VIEW-CONTROLLER) ARCHITECTURE 🤔

.

WHAT IS MVC?


Represents the data and business logic
MODEL
VIEW
Represents the user interface
2. Get Data
3. Get Presentation
USER CLICKS
1. Request
CONTROLLER
4. Response
Handles the user request and acts as interface btn Model and View

HOW TO SETUP MVC IN ASP.NTE CORE

1. Add the MVC services to the dependency Injection

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
}
```

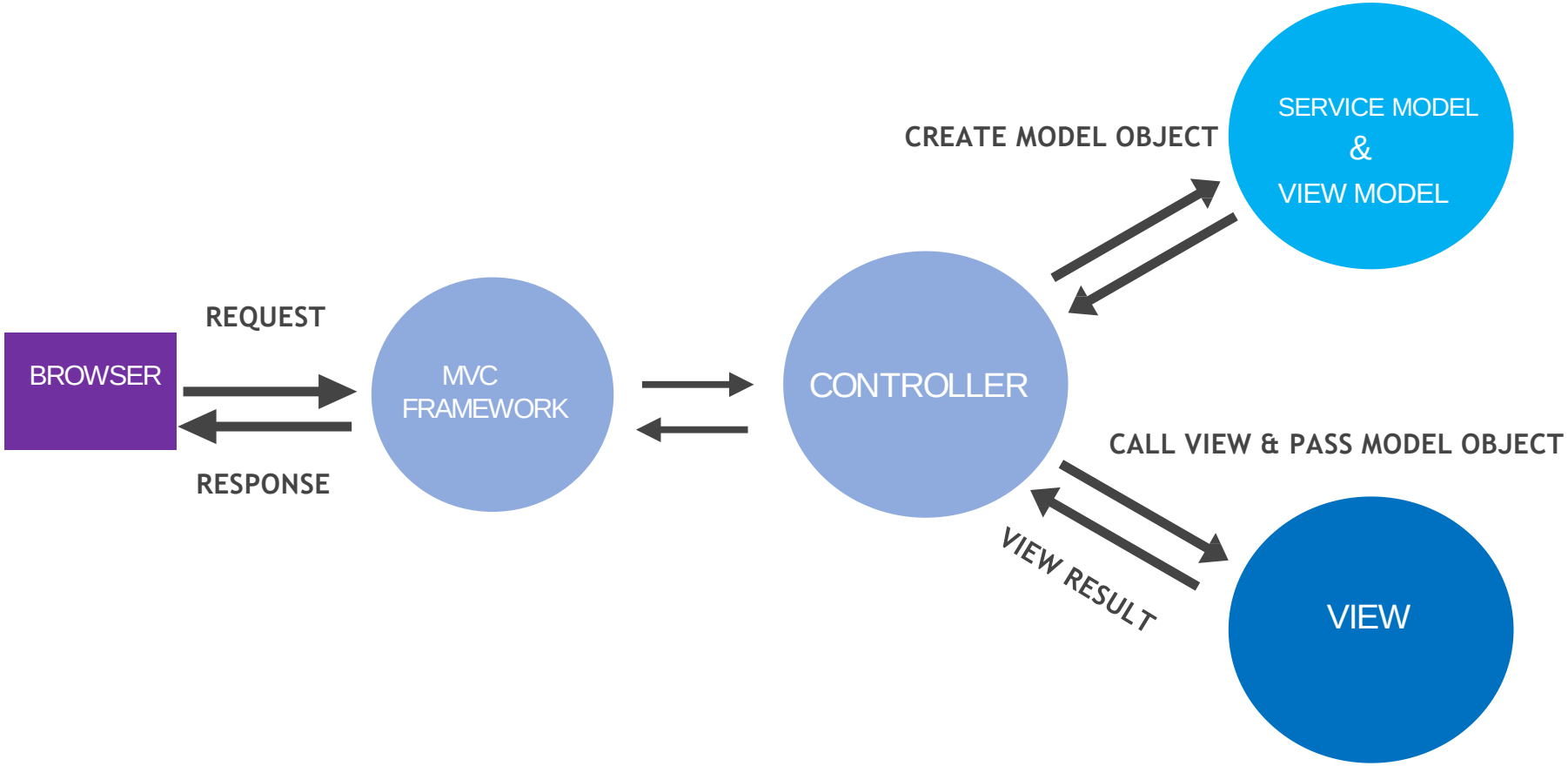
2. Add endpoint middleware to the request pipeline

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Controller and Action Methods

CONTROLLERS

👉 Controller is a class , which receives HTTP request from the browser and sends HTTP response to the browser



CONTROLLERS

- The controller is a class.
- Optionally a public class
- Controller should be inherited from “Microsoft.AspNetCore.Mvc”.
- Controller’s name should have suffix “Controller”. EX. HomeController

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

CONTROLLER ACTIONS

- Controller actions are the methods used to handle the HTTP request and act accordingly to return a view or other result
- Any public method on a controller type is actions
- Action Method cannot be overloaded in the controller
- Action Method cannot be static

ACTION RESULT

- 👉 ActionResult is a result of action methods/pages or return types of action methods/page handlers
- 👉 Action result is a parent class for many of the derived classes that have associated helpers.
- 👉 The IActionResult is an interface and ActionResult is abstract class from which different action results inherit.

ACTION RESULT

Action Result	Helper Method	Description
ViewResult	View	Renders a view as a Web page.
PartialViewResult	PartialView	Renders a partial view, which defines a section of a view that can be rendered inside another view.
RedirectResult	Redirect	Redirects to another action method by using its URL.
RedirectToRouteResult	RedirectToAction RedirectToRoute	Redirects to another action method.
ContentResult	Content	Returns a user-defined content type.
JsonResult	Json	Returns a serialized JSON object.
JavaScriptResult	JavaScript	Returns a script that can be executed on the client.
FileResult	File	Returns binary output to write to the response.
EmptyResult	(None)	Represents a return value that is used if the action method must return a **null** result (void).

Razor View Engine

RAZOR VIEW ENGINE

RAZOR View Engine

```
@{
    C#.net code
}
```

- A Razor view contains both C# code and HTML. That's why its file extension is .cshtml.

- In ASP.NET Core MVC, the Razor view engine uses server-side code to embed C# code within HTML elements.

- To execute one or more C# statements, you can declare a Razor code block by coding the @ sign followed by a pair of curly braces ({ }).

Razor View Engine

Razor Expression

Razor View Engine

Razor Code Blocks

Razor View Engine

Razor If

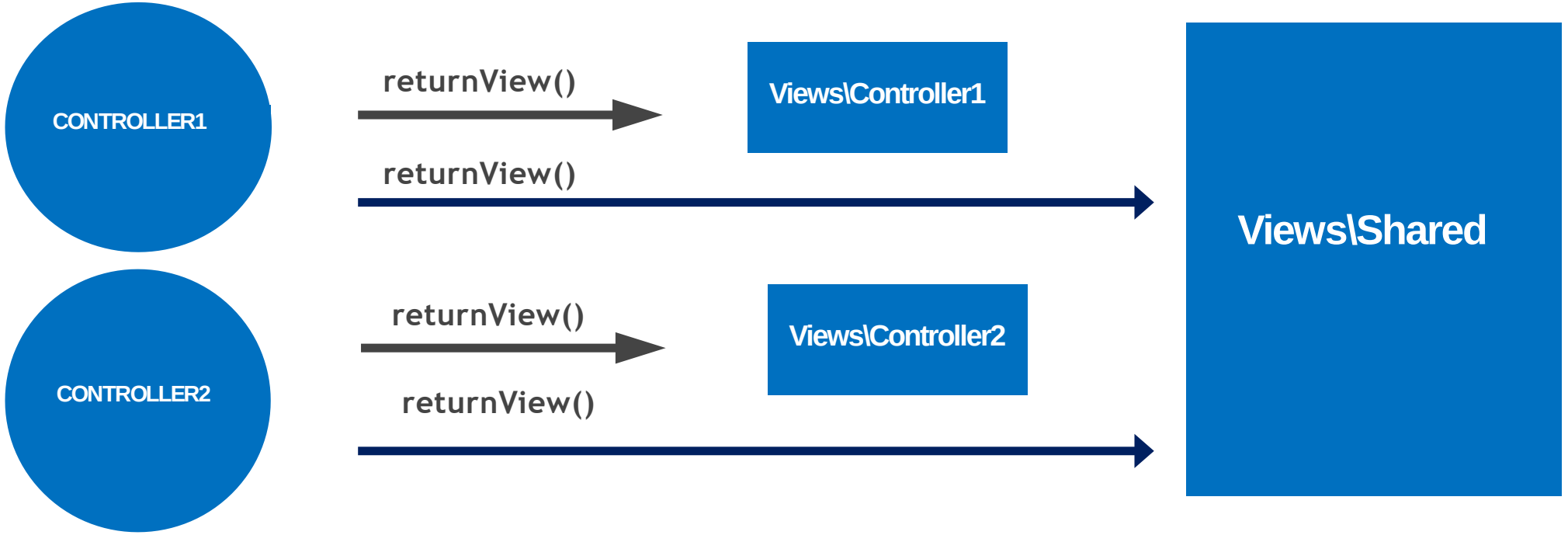
Razor View Engine

Razor For and Foreach

SHARED VIEWS

WHAT IS SHARED VIEWS ?

- Shared views are present in the “Views\Shared” folder.
- Shared views are the views that can be called from any controller of the entire project
- Views that belongs to all controllers created as shared views
- When we call a view , it checks the folder for the view in the “views\controller name “ folder first , if it is not found, it will search in the “Views\Shared” folder



Shared Views

Passing Data to shared Views

LAYOUT VIEWS
PARTIAL VIEWS

LAYOUT VIEWS

Layout views

- Layout views contain “Page template”, which contains common parts of the UI, such as logo, header, menubar, side bar etc..
- @RenderBody() method represents the reserved area for the actual content of view
- Execution Flow

Controller → View → Layout View → Generate View Result → Controller → Browser

Layout Views

Sharing Data From View to Layout Views

SECTIONS IN LAYOUT VIEWS

- Sections are used to display view-specific content in the layout view.
- Sections are defined in the view and rendered in the layout views

View

```
@section sectionname
{
   content
}
```

Layout View

```
@RenderSection("Section name")
```

_ViewStart.cshtml

- It defines the default layout view of all the views of folder .
- It can be present either in “Views” folder or in “Views\controllername” folder
- If it is present in “Views” folder, it specifies the path of the default layout view of all the views in the entire project.
- If it is present in “Views\controllername” folder, it specifies the path of layout view of all the views in the same folder only.

_ViewStart.cshtml

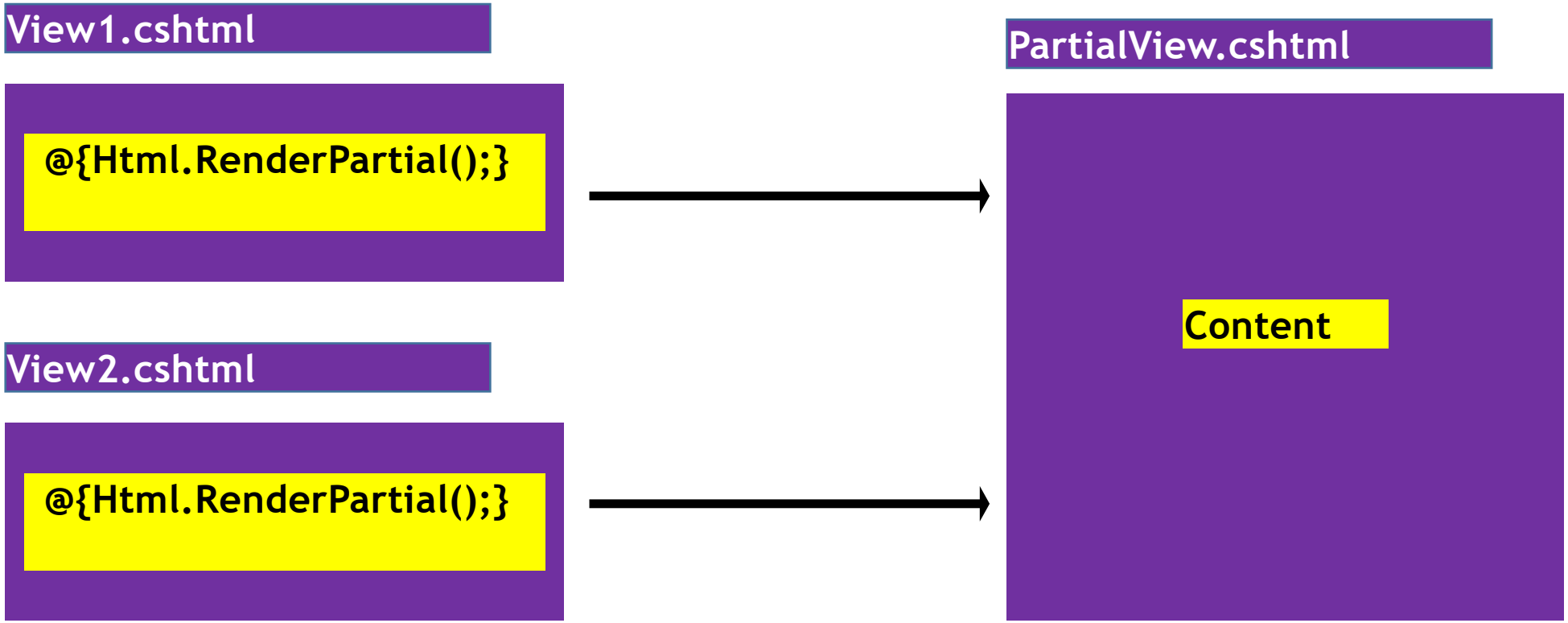
```
@
{
  Layout=“Path of Layout View”;
}
```

Layout Views

Creating Multiple Layout Views

PARTIALVIEWS

- Sections Partial View is a small view that contains content that can be shared among multiple views
- Can be presented in “Views\Controllername” folder or in “Views\Shared” folder.



URL Routing

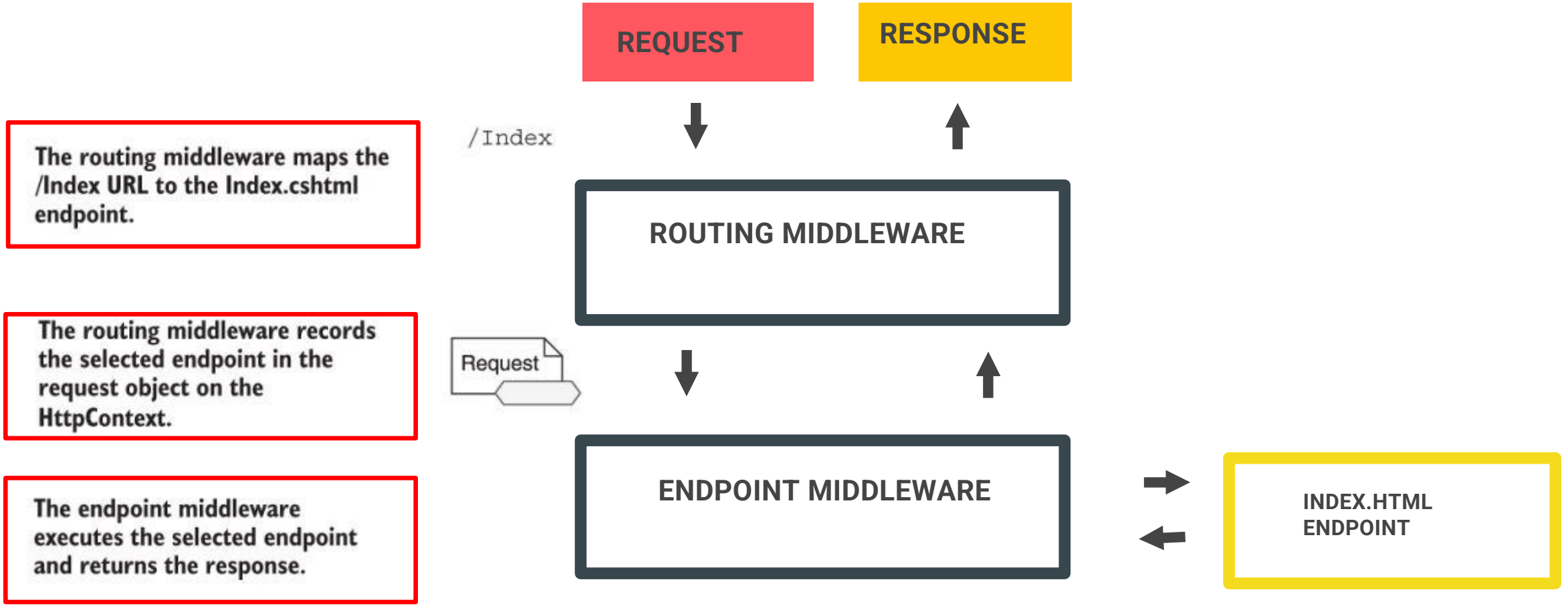
OBJECTIVES

You are going to learn:

- Introduction to Routing
- How Routing Works in ASP. Net Core
- Conventional Routing
- Attribute Routing

WHAT IS ROUTING?

- 👉 Routing is the process of mapping an incoming request to a method that will handle it.
- 👉 The process of mapping a request to a handler is called *routing*



ENDPOINT ROUTING IN ASP.NET CORE

- Endpoint routing is fundamental to all ASP.NET Core apps.
- It's implemented using two pieces of middleware:
  - **EndpointMiddleware**: Used to *register* the endpoints in routing the system when the application start. The middleware *executes* one of the endpoints at runtime.
  - **`EndpointRoutingMiddleware`**: This middleware chooses *which* of the endpoints registered by the **`EndpointMiddleware`** should execute for a given request at runtime.

USING ENDPOINT ROUTING IN ASP.NET CORE

- An *endpoint* in ASP.NET Core is some handler that returns a response.
- Each endpoint is associated with a URL pattern
- MVC controller action methods typically make up the bulk of the endpoints in an application

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

REGISTORING ENDPOINT IN ASP.NET CORE APPLICATION

- To register endpoints in your application, call UseEndpoints in the Configure method of Startup.cs.
- This method takes a configuration lambda action that defines the endpoints in the application
- Each endpoint is associated with a *route template* that defines which URLs the endpoint should match
- A *route template* is a URL pattern that is used to match against request URLs.

```
app.UseRouting();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

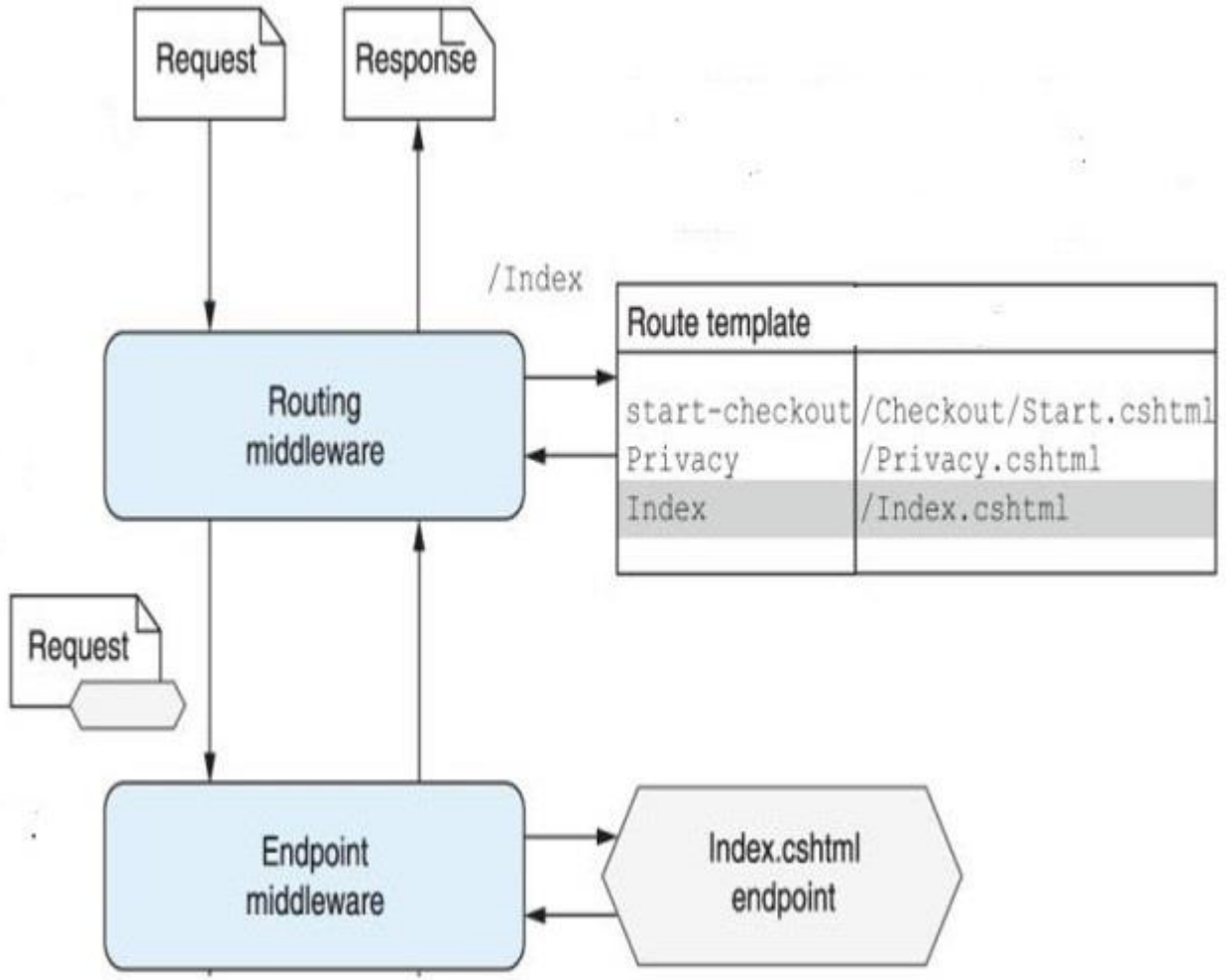
Add the EndpointRoutingMiddleware to the middleware pipeline.

Add the EndpointMiddleware to the pipeline and provide a configuration lambda.

Register an endpoint inline that returns page

HOW ENDPOINT ROUTING WORKS IN ASP.NET CORE?

- The EndpointMiddleware stores the registered routes and endpoints in a dictionary, which it shares with the RoutingMiddleware
- At runtime the RoutingMiddleware compares an incoming request to the routes registered in the dictionary
- If the RoutingMiddleware finds a matching endpoint, it makes a note of which endpoint was selected and attaches that to the request's HttpContext object.
- It then calls the next middleware in the pipeline. When the request reaches the EndpointMiddleware, the middleware checks to see which endpoint was selected and executes it,



USING ENDPOINT ROUTING IN ASP.NET CORE

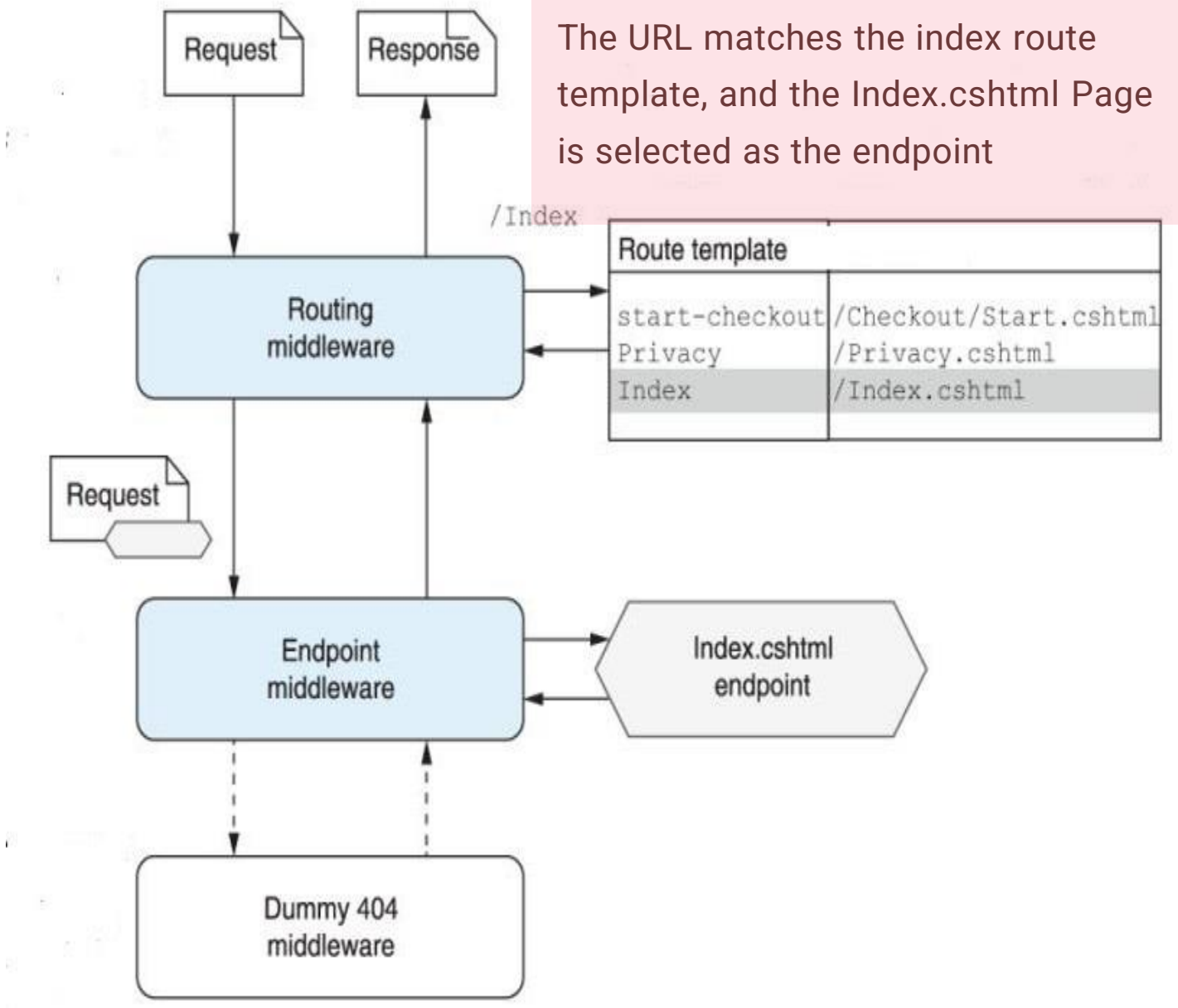
The routing middleware checks the incoming request a against the list of route template

The routing middleware records the selected endpoint in the request object. All the subsequent middleware can view which endpoint was selected.

The endpoint middleware executes the selected endpoint and returns the response

.

If no route matches the request URL, no endpoint is selected, the endpoint middleware does not run, and the dummy middleware returns a 404.



The URL matches the index route template, and the Index.cshtml Page is selected as the endpoint

HOW THE DEFUALT ROUTE WORKS

- The pattern for the default route
  {controller=Home}/{action=Index}/{id?}

- The first segment specifies the controller. Since the pattern sets the Home controller as the default controller, this segment is optional

- The second segment specifies the action method within the controller. Since the pattern sets the Index() method as the default action, this segment is optional.

- The third segment specifies an argument for the id parameter of the action method. The pattern uses a question mark (?) to specify that this segment is optional.

Request URL	Controller	Action	Id
http://localhost	Home	Index	null
http://localhost/Home	Home	Index	null
http://localhost/Home/Index	Home	Index	null
http://localhost/Home/About	Home	About	null
http://localhost/Product	Product	Index	null
http://localhost/Product/List	Product	List	null
http://localhost/Product/List/Guitars	Product	List	Guitars
http://localhost/Product/Detail	Product	Detail	0
http://localhost/Product/Detail/3	Product	Detail	3

Route

Creating Custom Route

Custom Route

```
endpoints.MapControllerRoute(
name: "StudentByAdmisonDate",
pattern: "{controller=Home}/{action=StudenByAdmisionDate}/{month}/{year}");
```

Route

Attribute Routing

ATTRIBUTE ROUTING

- The attribute routing uses the attributes defined directly on the controller action to define the routes.
- Attribute routing gives you more control over the URLs in your web application.
- To make use of the attribute routing use the MapControllers method.
    - `endpoints.MapControllers();`

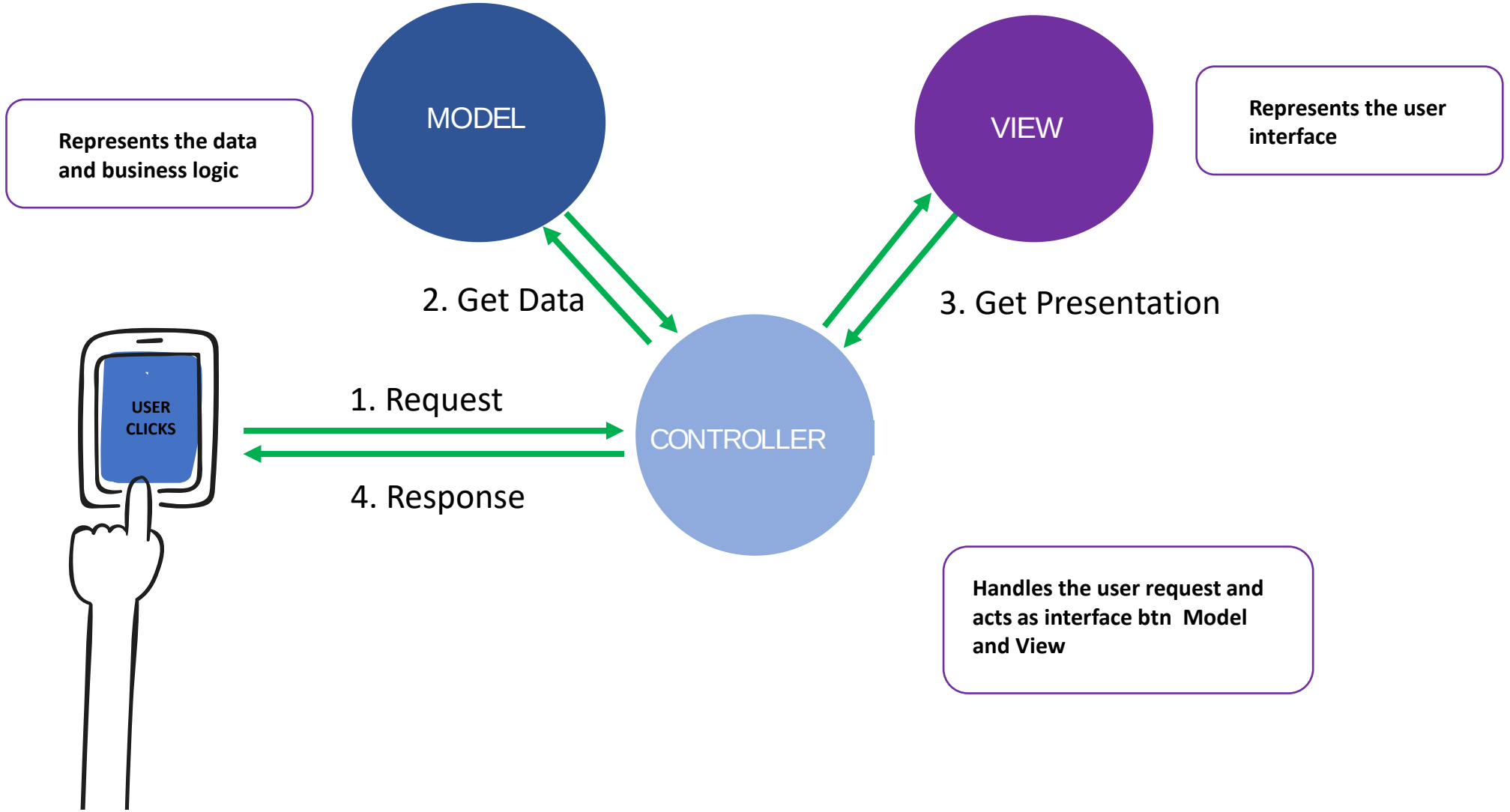
Models & Strongly Typed Views

OBJECTIVES

You are going to learn:

- Models
- Types of Models
- Model Binding
- Bind Attribute

WHAT IS MVC?


Represents the data and business logic
MODEL
VIEW
Represents the user interface
2. Get Data
3. Get Presentation
USER CLICKS
1. Request
CONTROLLER
4. Response
Handles the user request and acts as interface btn Model and View

Models

- Model is a class that defines the structure of the data that you want to store /display
- Also contains business logic
- Model will be called by controller

View Model

```
Public class ViewModel
{
    public datatype propertName{get;set}

}
```

Represents the structure of data that you want to display to user

Domain Model

```
Public class DomainModel
{
    public datatype propertName{get;set}

}
```

Represents the structure of data that you want to store in the database table

Service Model

```
Public class ServiceModel
{
    public datatype propertName{get;set}

}
```

Represents business logic (code that needs to be executed before inserting /updating etc).

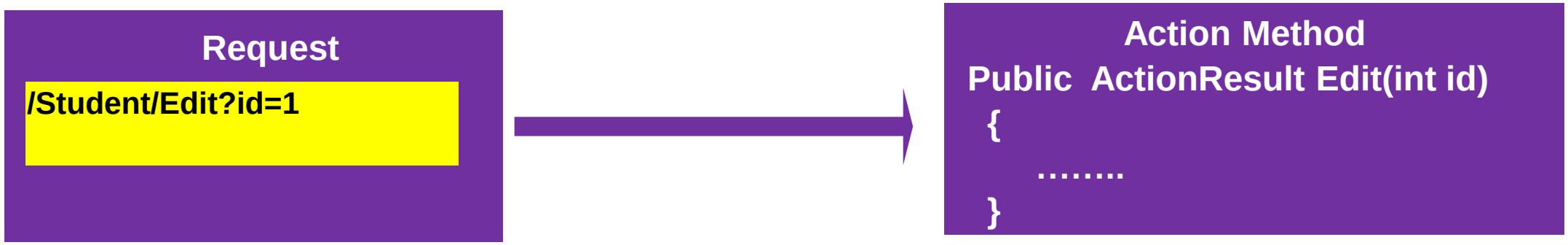
Strongly typed View

- A view associated to a specific model class is called as “Strongly Typed View”.
- Strongly Typed View have to specify the model class name with @model directive at the top of view
- Strongly Typed View can receive model objects from controller

Model Binding

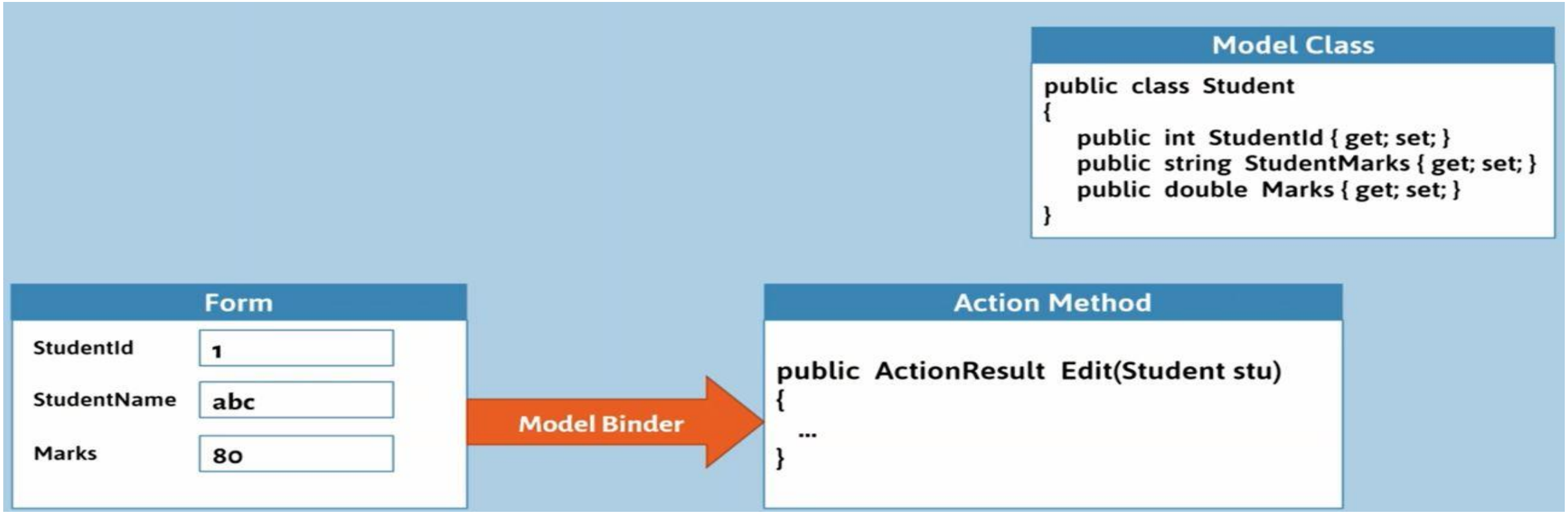
Model Binding

- A Process of receiving values from different sources of the request and passing them as arguments to action method
- Assigns values to different parameters of the action method automatically



Model Binding

- A Model Binding can work with complex types
- Assigns Model binding can automatically can covert form field data or query string values to the properties of complex type parameter of an action method.
- Default values are null or Zero (0)

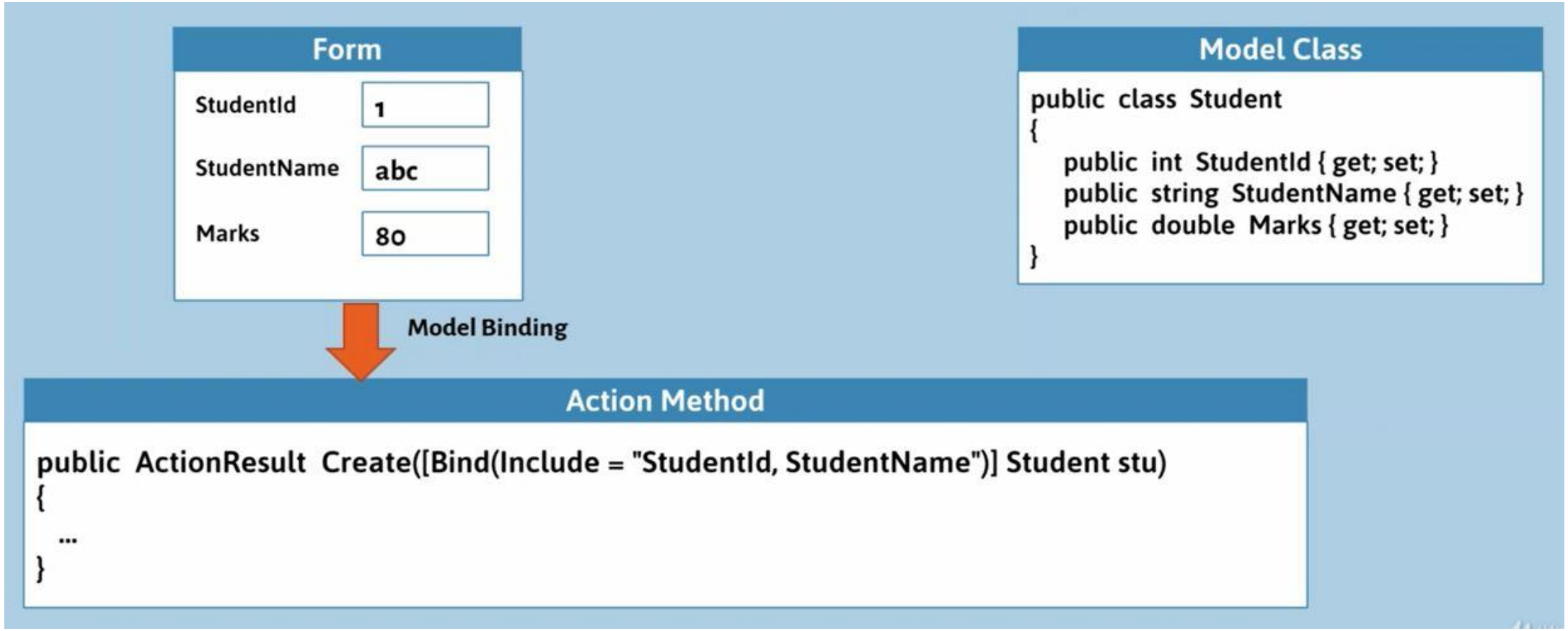


SOURCES OF MODEL BINDING

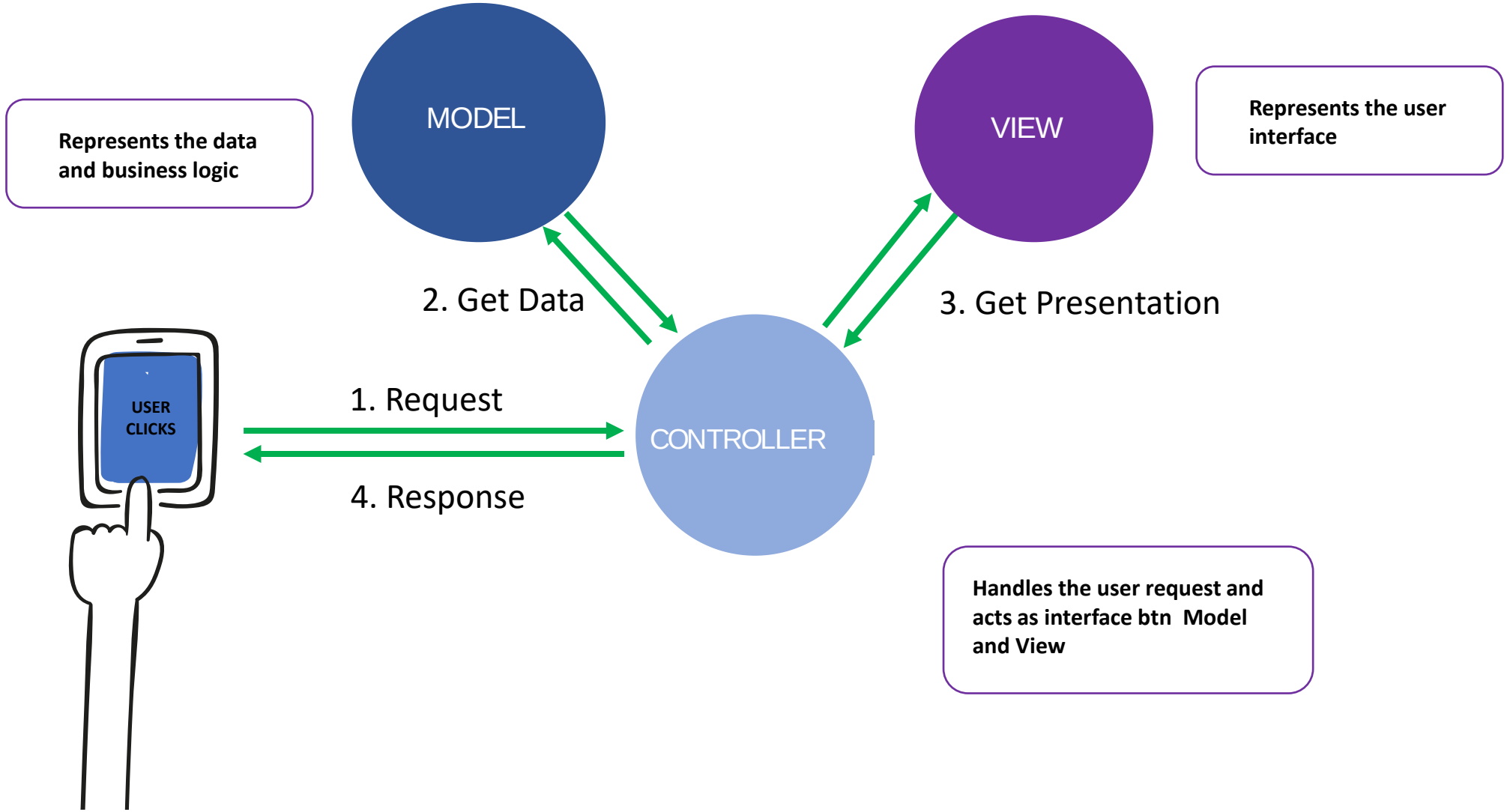
- Query String.
- From Data Ex: <input type="text" name="stdName" >
- Route Data
- JSON Request body (in case of AJAX)

BIND ATTRIBUTE

- The bind attribute allows you to specify the list of properties that you want to bind into the model object, that received using “Model Binding”



WHAT IS MVC?


Represents the data and business logic
MODEL
VIEW
Represents the user interface
2. Get Data
3. Get Presentation
USER CLICKS
1. Request
CONTROLLER
4. Response
Handles the user request and acts as interface btn Model and View

Entity Framework Core

OBJECTIVES

You are going to learn:

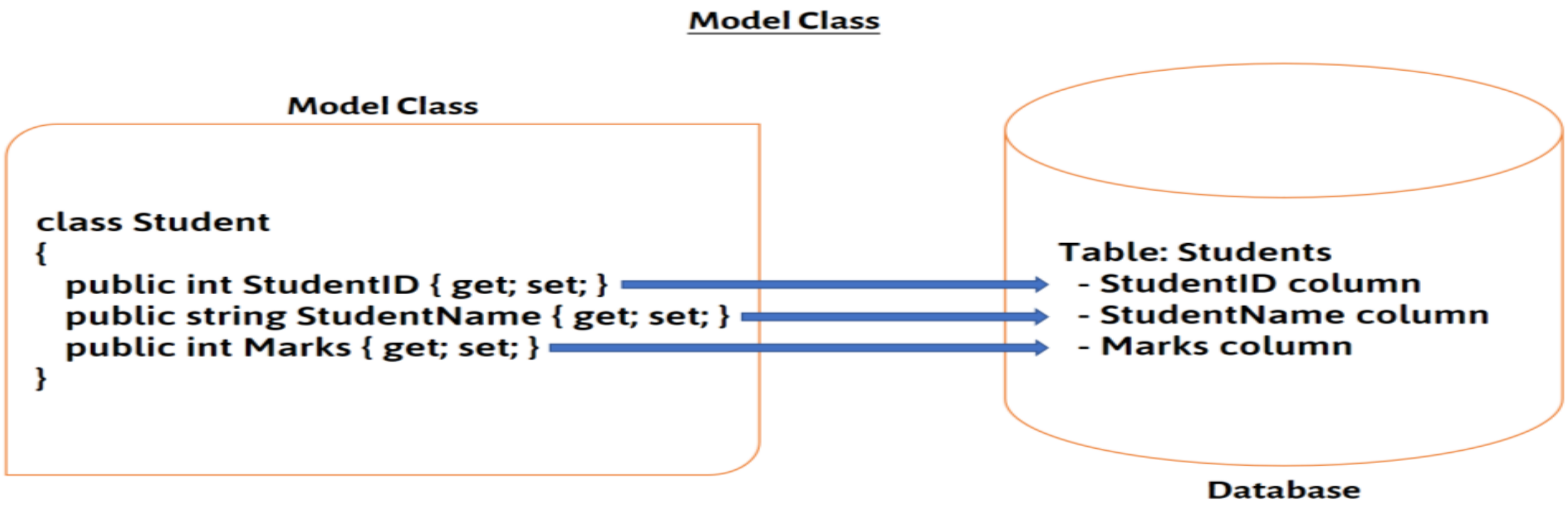
- Introduction Entity Framework Core
- Query Operations
- Database First Approach
- Code First Approach

WHAT IS ENTITY FRAMEWORK?

- 👉 EF Core is a database technology that provides an object-oriented way to access databases.
- 👉 EF acts as an *object-relational mapper* (ORM), communicating with the database for you and mapping database responses to .NET classes and objects,
- 👉 EF Core focuses on the communication between an application and a database.
- 👉 An *entity* is a .NET class that's mapped by EF Core to the database. These are classes you define, typically as POCO classes, that can be saved and loaded by mapping to database tables using EF Core.

OBJET RELATIONAL MAPPING ORM

- Model class reflects structure of the table
- Table is treated as a class.
- Column is treated as a property.
- Then, EF Core to generate a database from those classes.
- This approach is known as EF Code First.



DbContext and DbSet

- **DbContext:** The primary class for communicating with a database
- **DbContext:** represents a collection of DbSet's.
- **DbSet:** represents a collection of model classes, also known as entity classes, or domain model classes, that map to a database table
- **DbSet:** provides methods to retrieve, insert, update and delete data.
- **DbContextOptions:** provides configuration information to the DbContext class.

```
Using ModelClassName;
class className: DbContext

{
    public DbSet<ModelClass> tableName {get;set}
    public DbSet<ModelClass> tableName {get;set}

}
```

Features of Entity Framework Core

- **Modeling:** EF Model classes represent structure of tables, so the developers can easily understand the fields and manipulate them in code.

- **Query / Non-Query:** EF supports both querying (selecting data from database) and non-querying (inserting, updating, deleting) also.

Database First Approach

DATABASE FIRST APPROACH

Database First

- We design Our tables
- EF generates domain classes

DATABASE FIRST APPROACH

1

Packages needs to be installed

1

PACKAGES

- EF Core is inherently modular, so you'll need to install multiple packages
- ***Microsoft.EntityFrameworkCore.SqlServer***—This is the main database provider package for using EF Core at runtime. It also contains a reference to the main EF Core NuGet package.
- ***Microsoft.EntityFrameworkCore.Design***—This contains shared design-time components for EF Core.
- ***Microsoft.EntityFrameworkCore.Tools***—This contains shared design-time components for EF Core.

DATABASE FIRST APPROACH

1. **Packages needs to be installed**

2. **Connection Strings:** Break Connecting to existing database

2 CONNECTING TO DATABASE

💬 **Reverse engineering** is the process of scaffolding entity type classes and a DbContext class based on a database schema.

💬 It can be performed using the **Scaffold-DbContext** command of the EF Core Package Manager Console (PMC) tools or the **dotnet ef dbcontext scaffold** command of the .NET Command-line Interface (CLI) tools.

👉 ***Scaffold-DbContext "Data Source=PC1\JAF;Initial Catalog=Picky;Integrated Security=True;Pooling=False" Microsoft.EntityFrameworkCore.SqlServer -outputDir Models***

👉 ***dotnet ef DbContext "Data Source=PC1\JAF;Initial Catalog=Picky;Integrated Security=True;Pooling=False" Microsoft.EntityFrameworkCore.SqlServer -outputDir Models***

DATABASE FIRST APPROACH

1. **Packages needs to be installed**

2. **Connection Strings:** Break
   Connecting to existing database

3. **Adding Controllers**

3 CONTROLLERS

💬 **Adding controllers to manipulate the**

Query Operations in Entity Framework

HOW TO RETRIEV ALL ROWS FROM DATABASE?

- **ToList()** method is used to select all columns and all rows of the table.
- It executes the query immediately and returns corresponding data as a "collection of objects of model class".

Syntax:

```
DbContextclass referencevariable = new DbContextclass();
referencevariable.Dbsetname.ToList();
```

HOW TO RETRIEVE MULTIPLE ROWS CONDITIONALLY

- Where(lambda) Filters the entities according to the logic of the lambda expression. It retrieves the rows that are matching with specified condition.
- The Where() method receives a lambda expression as argument, which contains an argument (temp) that represents a row of the table, checks the condition and returns true / false.
- If true is returned, the row will be taken into the result; otherwise the row will not be taken into the result.

Syntax:

```
DbContextclass referencevariable = new DbContextclass();
referencevariable.Dbsetname. Where(temp => condition).ToList();
```

HOW TO RETRIEVE SINGLE ROW CONDITIONALLY?

- FirstOrDefualt() the first instance of the entity identified by the lambda expression parameter, or null if nothing is found.
- It will return NULL if there is no record

Syntax:

```
DbContextclass referencevariable = new DbContextclass();
referencevariable.Dbsetname. Where(temp => condition).FirstOrDefualt();
```

Inserting, updating and Delating Data

HOW TO INSERT, UPDATE AND DELATE DATA?

- 👉 Add(entity) → Adds an entity to the DbSet collection and marks it as Added
- 👉 Update(entity) → Marks the entity as Modified
- 👉 Remove(entity) → Marks the entity as Deleted.
- 👉 SaveChanges() → Saves changes to the database

e

Inserting record to the Database

e

Updating record in the Database

e

Deleting record from the Database