MVC

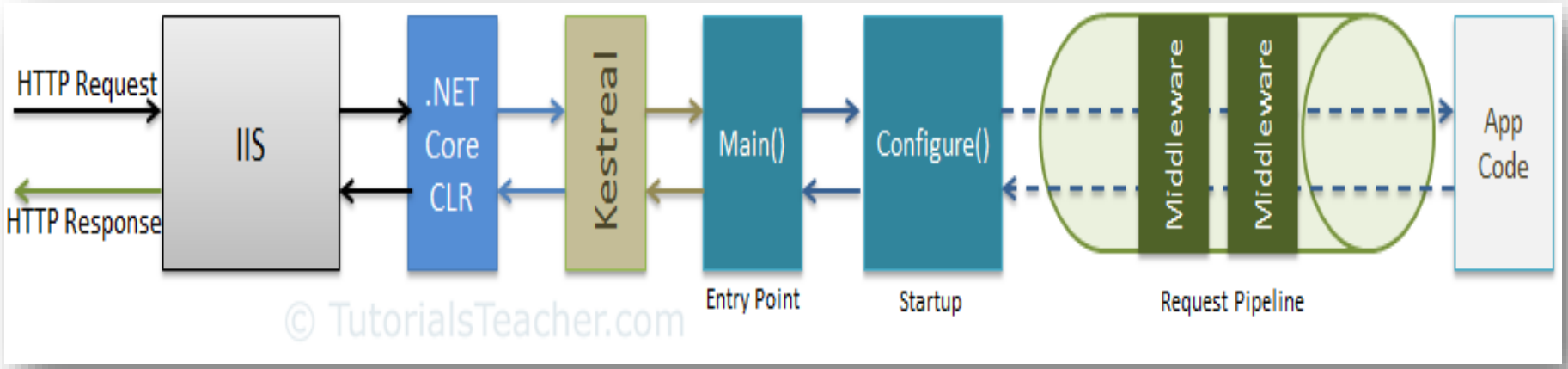
Christen Zarif

outline

- Routing

ASP.NET Core Request Processin

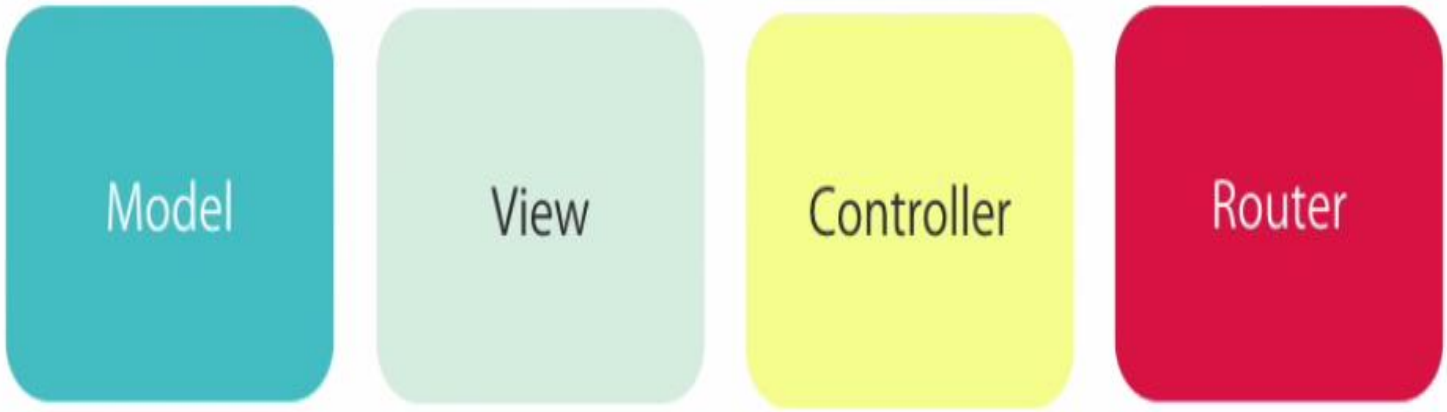
- Middlewares build the request pipeline



Routing

- ASP.NET Core MVC is the .NET Core counterpart of the ASP.NET MVC framework.
- You can take advantage of ASP.NET Core MVC
  - to build cross-platform, scalable, high-performance web applications and APIs using the Model-View-Controller design pattern.
- ASP.NET Core takes advantage of routing to map incoming requests to respective controller actions.

MVC Architectural Pattern



Router

Selects the right controller to handle a request.

Routing OutLine

- Understanding the routing mechanism
- Adding custom entries to a route table
- Defining defaults, parameters, and validation
- Custom route constraints
- Attribute routing

ASP.NET Core URL Routing

- ASP.NET introduced Routing to ***eliminate*** needs of mapping each URL with a physical file.
- Routing enable us to define *URL pattern* that ***maps*** to the *request handler*.

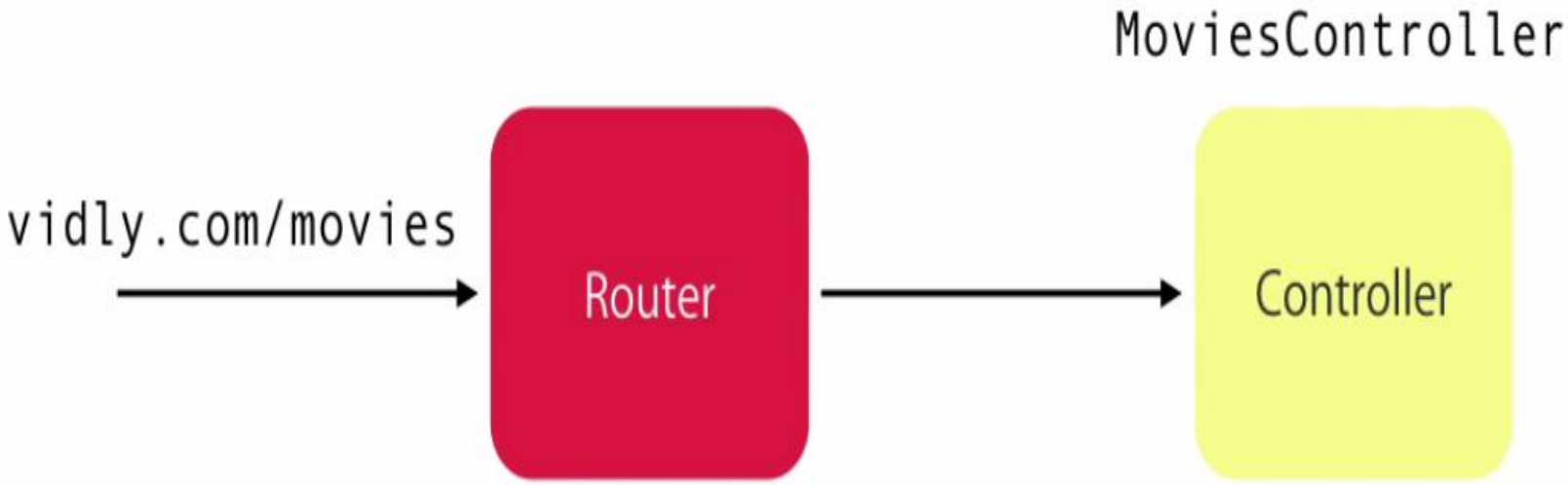
- It is used to create **URL patterns** for your application. It matches incoming HTTP requests and dispatching those requests to the application's **endpoints**.

- Endpoints are those units that are responsible for request-handling. These are defined in the Configure() method of Startup.cs class.

Routing Technoques

- MVC & Web API require us to setup Endpoint to each controller action methods. There are two different ways by which we can set up routes
  - Conventional Routing
  - Attribute Routing

Router

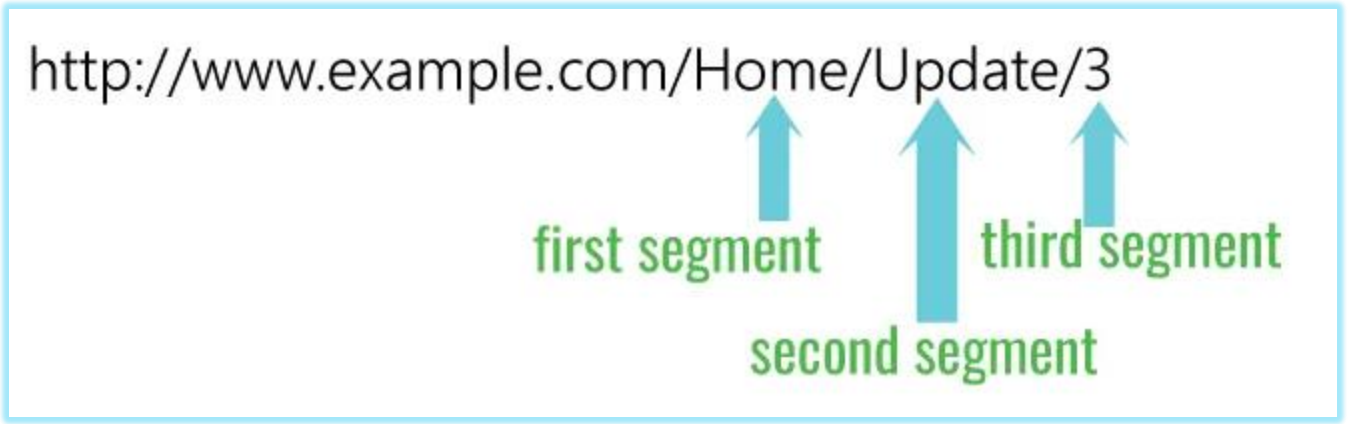


Structure of URL Pattern

- {controller} and {action} are two reserved placeholder in MVC

Segment in URL

- The parts of the URL, excluding the hostname and query string, and are separated by the '/' character.



Default Rout

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

- The default value for segments is optional.
- It only tells the routing system to take a given default value for the corresponding segment if the segment is missing from the URL.

Convention-Based Routing

- Where the routes are applied inside the Configure() method of the Startup class.
- This is done by using app.UseEndpoints() method

Static Text

```
"Student/{action=Index}/{id?}"
```

- This route contain 3 segment
- The first segment should be Student
- We can merge between static and placeholder

```
"News{controller}/{action}"
```

Custom Segment Variables

```
"{controller=Home}/{action=Index}/{id}"
```

- In this route I have added a Custom Segment variables by the name **id**
- Any action can obtains the value of the custom segment variable called id in the route URL pattern using the RouteData.Values property

```
RouteData.Values["id"];
```

Custom Segment Variables(con.)

```
"{controller=Home}/{action=Index}/{id}"
```

- Access URL Parameter from Action Parameter
  - If you define Parameters in Action Methods in such a way that the names of the parameters are same like the URL parameter variables names, then the ASP.NET Core MVC framework will automatically pass the values obtained from these URL parameter variables to the parameters of the action method. "**Model Binding**."

```
public IActionResult Check(int id)
{
```

Custom Segment Variables(con.)

```
"{controller=Home}/{action=Index}/{id?}")
```

- Optional URL Segment
  - The **Segment which you don't need to specify for the route is called as optional URL Segment**.
  - You denote it by placing a question mark (?) after it.

Route Constraint

- The **constraints** are provided after the segment name with a color (:) character.

```
{controller=Home}/{action=Index}/{id:int}
```

- I have applied the int constraint on the route.
- This tells to match the URL only when the *id segment* in the URL is of type int.

- http://localhost:58470/Home/Check/Hello. → Status Code: 404; Not Found i

Constraint Supported

Constraint	Description	Example
alpha	Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)	{x:alpha}
bool	Matches a Boolean value.	{x:bool}
datetime	Matches a **DateTime** value.	{x:datetime}
decimal	Matches a decimal value.	{x:decimal}
double	Matches a 64-bit floating-point value.	{x:double}
float	Matches a 32-bit floating-point value.	{x:float}
guid	Matches a GUID value.	{x:guid}
int	Matches a 32-bit integer value.	{x:int}
length	Matches a string with the specified length or within a specified range of lengths.	{x:length(6)} {x:length(1,20)}
long	Matches a 64-bit integer value.	{x:long}
max	Matches an integer with a maximum value.	{x:max(10)}
maxlength	Matches a string with a maximum length.	{x:maxlength(10)}
min	Matches an integer with a minimum value.	{x:min(10)}
minlength	Matches a string with a minimum length.	{x:minlength(10)}
range	Matches an integer within a range of values.	{x:range(10,50)}
regex	Matches a regular expression.	{x:regex(^\d{3}-\d{3}-\d{4}$)}

Combining Constraints

- You can also **combine constraints** by character colon (:)

```
"{controller=Home}/{action=Index}/{id:alpha:regex(^H.*)?}");
```

Attribute Routing

- In Attribute Routing you apply route as C# attributes directly to the controller and actions.

```
[Route("CallRoute")]
public IActionResult Index()
{
    return View();
}
```

- If there is an **Attribute Routing** applied to a Controller or an Action then the Convention Routes for the Controller or Action will not work

Atribute Routing

```
[Route("[controller]/CallMe/[action]")]
public string Show()
{
    return "'Admin' Controller, 'Show' View";
}
```

- This route has the term called **controller** which states that the URL should have the Controller segment in it.
- And action the same should specify

Attribute Routing (Con.)

```
public class BooksController : Controller
{
    // eg: /books
    // eg: /books/1430210079
    [Route("books/{isbn?}")]
    public ActionResult View(string isbn)
    {
        if (!String.IsNullOrEmpty(isbn))
        {
            return View("OneBook", GetBook(isbn));
        }
```

```
    // eg: /books/lang
    // eg: /books/lang/en
    // eg: /books/lang/he
    [Route("books/lang/{lang=en}")]
    public ActionResult ViewByLanguage(string lang)
    {
        return View("OneBook", GetBooksByLanguage(lang));
    }
```

ASP.NET Core Endpoint Routing

- EndPoint Routing is the new way to implement the Routing in ASP.NET Core.
- It splits into two separate middleware’s
- UseRouting & UseEndpoints methods in the Configure method of the startup class.

Endpoint Routing

- **Endpoint Routing** which provides routing information to middlewares in the Startup.cs class.
- We **define** the Endpoint during the application startup.
- The **Routing Module** then **matches** the incoming URL to an Endpoint and **dispatches** the request to it.

How Endpoint Routing Works

- The Endpoint Routing has three components
  - Defining the Endpoints.
  - Route matching & Constructing an Endpoint (UseRouting).
  - Endpoint Execution (UseEndpoints).

```
app.UseRouting();

app.UseAuthorization();

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

UseRouting

- It resolves the incoming HTTP requests and constructs an Endpoint.

- We register it very early in the middleware pipeline using the UseRouting method.

EndPoint Routing

- Endpoint routing's two extension methods are:
  - UseRouting: It matches request to an endpoint.
  - UseEndpoints: It execute the matched endpoint.

UseEndpoints

- The EndpointMiddleware is responsible for execution the Endpoint.
- We register it using the UseEndpoints method.
- It reads the Endpoint, which was selected by the Route Matching middleware and runs the delegate associated with it.
  - The EndpointMiddleware middleware is terminal Middleware when a match is found
  - The middleware after UseEndpoints execute only when no match is found

Improve your Information

- How to convert HTTP to HTTPS
- Asp.net.coreEndpointRouting

☺ Thank You ☺