

ASP.NET Core - MVC

Jente Vets

AXXES.

Jente Vets

.NET Developer & Team Captain .NET
Backend

@ Axxes since 08/2015

Herenthout/Viersel – De Kempen

KPD Services / Emakina / NP-Bridging /
Katoen Natie / Reynaers Aluminium

AXXES_



And you are?



Brief history of (ASP).NET

- 2002: ASP.NET 1.0 as part of .NET Framework 1.0 with Web Forms
- 2003: 1.1 with performance & developer tools ++
- 2005: 2.0 with Master Pages, Roles & Membership, Provider model for DB access
- **2008: 3.5 with AJAX support and ASP.NET MVC Framework**
- 2010: 4.0 with improvements in routing, dynamic data and Web Forms
- 2012: 4.5 with async/await
- **2016: ASP.NET Core 1.0, complete rewrite of ASP.NET**
- 2017: ASP.NET Core 2.0 with new features and .NET Standard compatibility
- 2018: ASP.NET Core 2.1 LTS with SignalR support, RazorPages++ and MVC++
- 2019 ASP.NET Core 3.0 with Blazor, enhanced JSON API's
- 2019 ASP.NET Core 3.1 as LTS version, emphasizing stability



Brief history of ASP.NET

- 2020: ~~ASP.NET Core 4.0~~ → ASP.NET 5.0
- 2021: ASP.NET 6.0 LTS with .NET MAUI, Blazor Hybrid apps, hot reloading and Minimal API's
- 2022: ASP.NET 7.0 with nullable models in MVC views, Minimal API improvements
- 2023: ASP.NET 8.0 (Current) with changes to performance and Blazor alongside the updates regarding the AOT, Identity, SignalR and Metrics

Extra:

- ASP stands for Active Server Pages
- Skipped V4 to avoid confusion with .NET Framework 4.x
- Dropped “Core” from the name to emphasize this is the way forward

ASP.NET

Windows

Performance+

IIS hosting

Third Party IOC containers

...Core

Multi platform (Win, Mac, Linux, ...)

Performance ++

IIS, Kestrel, HTTP.sys

Built-in Dependency Injection

What is MVC?

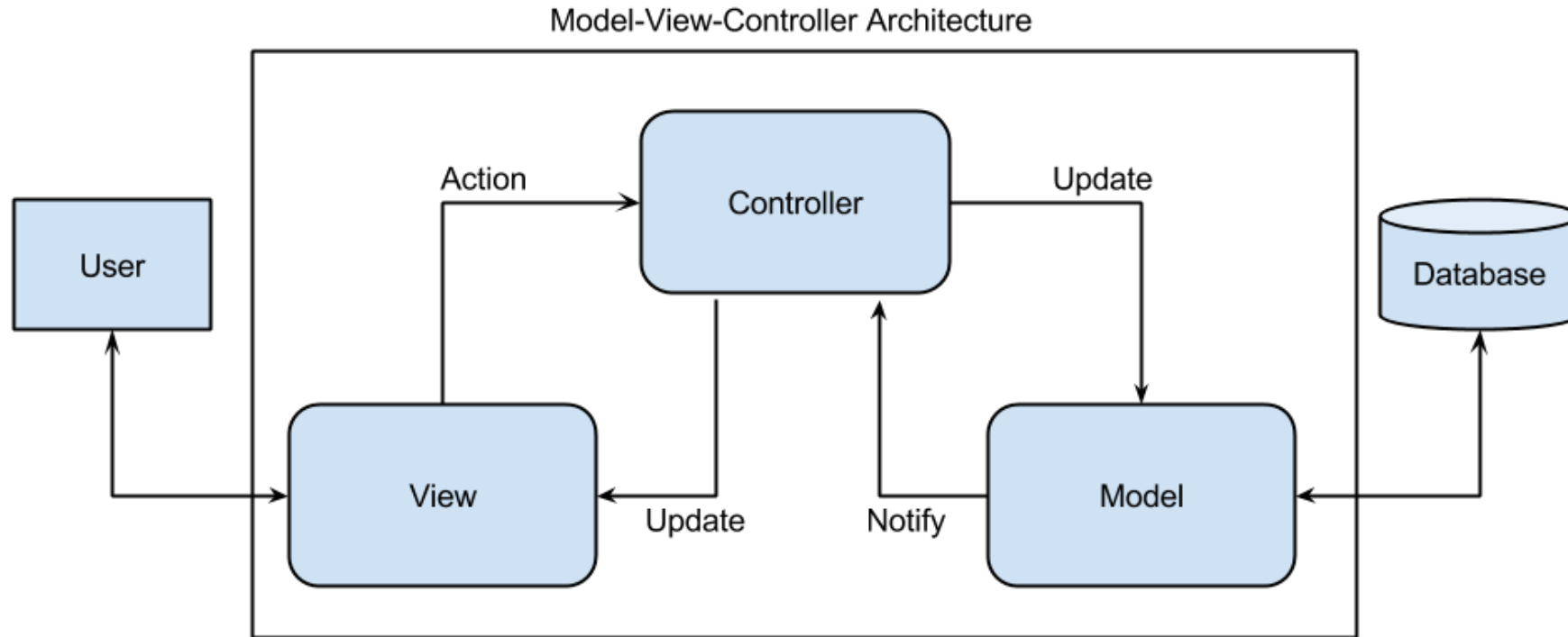
Model – View – Controller

- Design pattern to write better/structured code which is easy to maintain.
- Separation of concerns
- Modular
- Easy to learn
- Testability
- Not ASP.NET specific (Spring MVC, Ruby on Rails, Django, ...)
- Server side technology

ASP.NET Core implementation of MVC

- Design pattern aims to split the application in 3 main parts (Model-View-Controller)
 - Efficiency, maintainability, structured
- ASP.NET MVC focuses this philosophy specifically on Web Apps
 - C#, Controllers, Routing, Razor Views, ...
- **Model**: this contains the data and logic of the application, such as databases and calculations
- **View**: shows the model data coming from the Controller to the user
- **Controller**: receives and processes the user's commands, communicates with the Model to retrieve or edit data, and selects the appropriate View to display the results to the user.

Basic principles of MVC



What MVC is not...

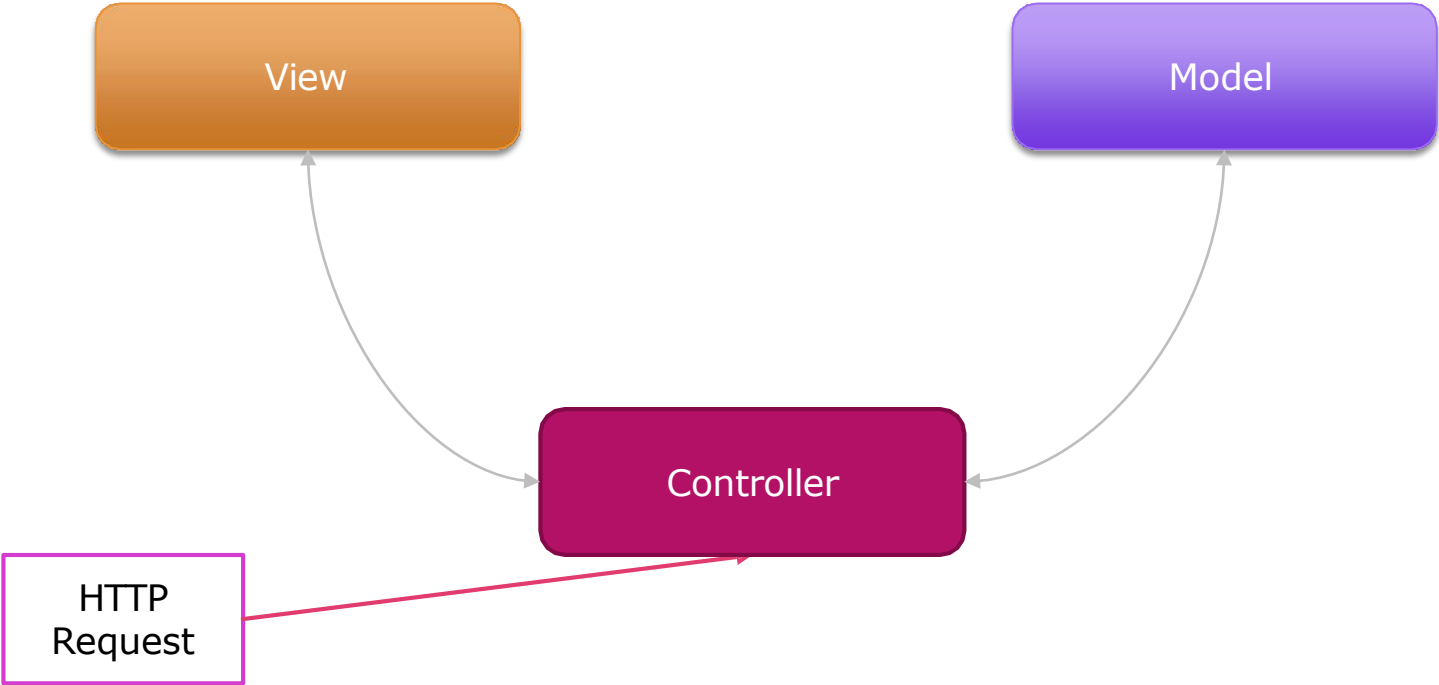
- It's not a complete design pattern.
 - It doesn't define how to implement business logic, that is up to you.
- It's not a full frontend framework, so it's not suitable for SPA's.
- It's not super complex.



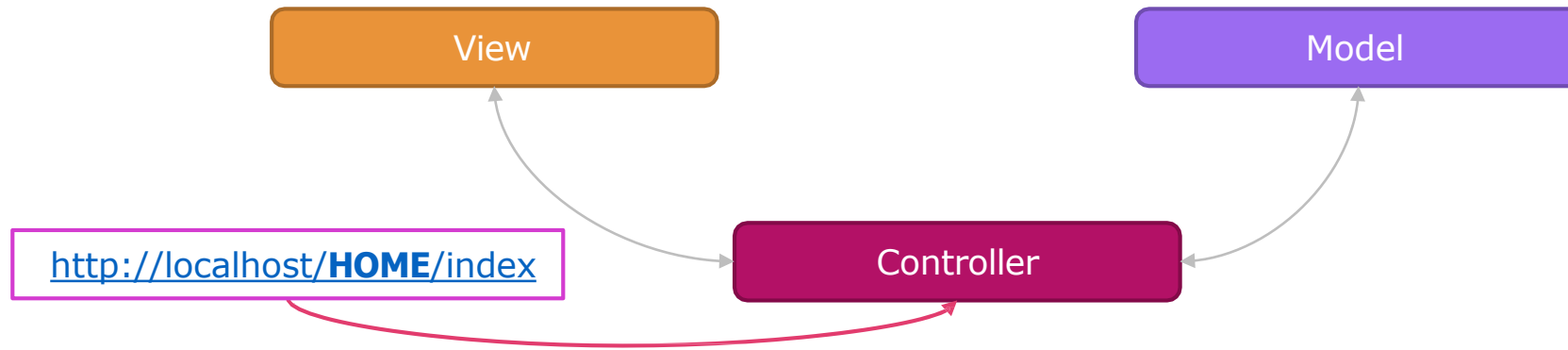
Controller

- Interprets Actions of a user to produce results
- Acts as a bridge between user input and application behavior.
- Determines which actions to take based on user input, but does not contain business logic
- Prepares data for presentation in the View layer.
- Inherit from **Controller** for View Controllers and from **ControllerBase** for all other controllers
 - Controller extends ControllerBase with ViewBag, ViewData, ViewResult, ...
- Return type IActionResult so we can return different types depending on the logic

Controllers



Controllers

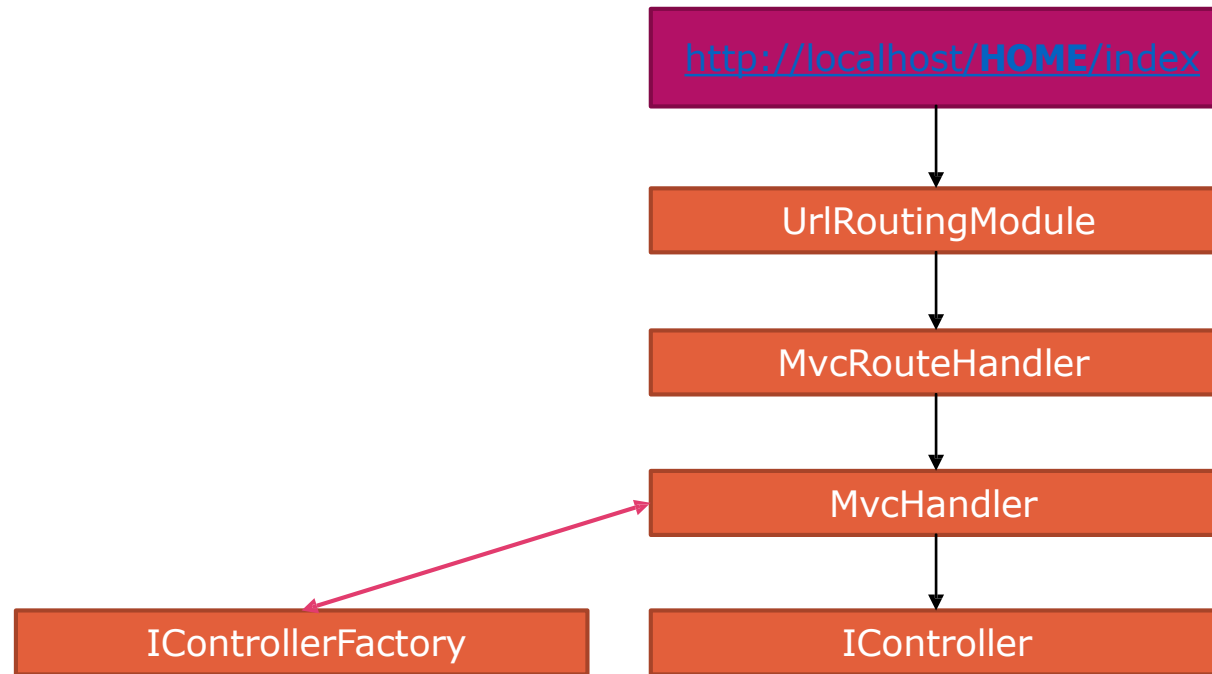


```
ASP.NET Core  
app.MapDefaultControllerRoute();
```

```
ASP.NET MVC  
app.MapControllerRoute(  
    name:"default",  
    pattern:"{controller=Home}/{action=Index}/{id?}");
```

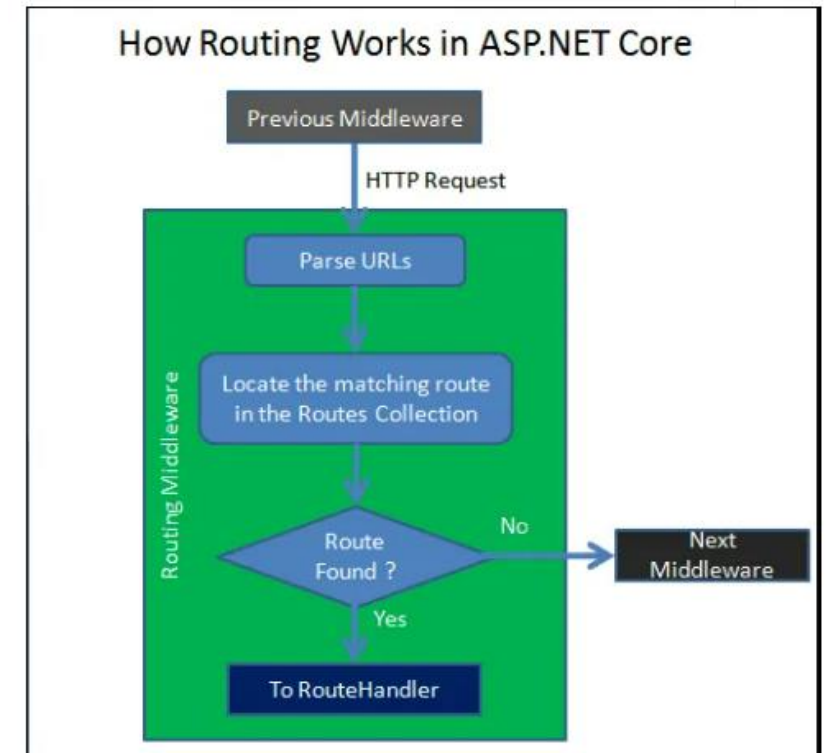
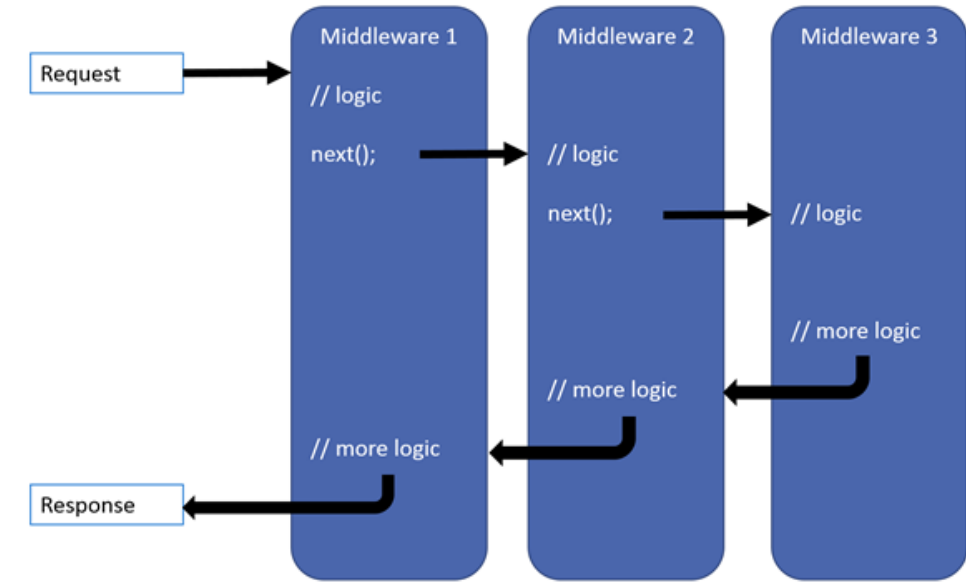
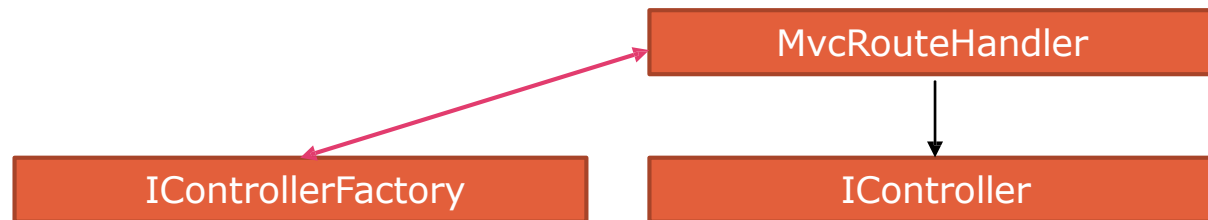
Controllers ASP.NET MVC

- MvcRouteHandler
- MvcHandler
 - Create
 - Execute
 - Dispose
- IController
 - Lowest level abstractie



Controllers ASP.NET core

- MvcRouteHandler
 - Controller Factory
- Icontroller
 - Lowest level abstractie

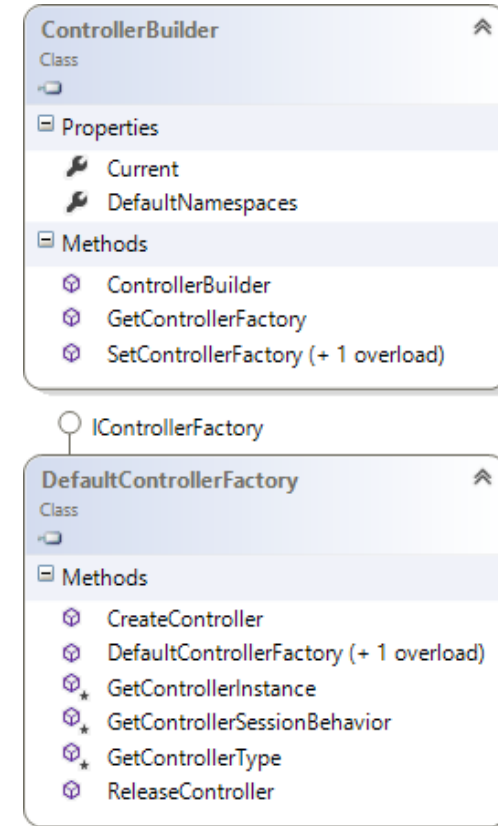


Controllers

- DefaultControllerFactory
 - Searches for the controllers
 - Checks referenced assemblies & namespaces
 - Type name ends with "Controller" & implements IController
- Does need a default constructor.

```
public class ConferenceController : Controller
{
    private readonly ILogger _logger;

    public ConferenceController(ILogger logger)
    {
        _logger = logger;
    }
}
```



HTTP Verbs

GET	To retrieve the information from the server. Parameters will be appended in the query string.
POST	To create a new resource.
PUT	To update an existing resource.
HEAD	Identical to GET except that server do not return the message body.
OPTIONS	It represents a request for information about the communication options supported by the web server.
DELETE	To delete an existing resource.
PATCH	To full or partial update the resource.

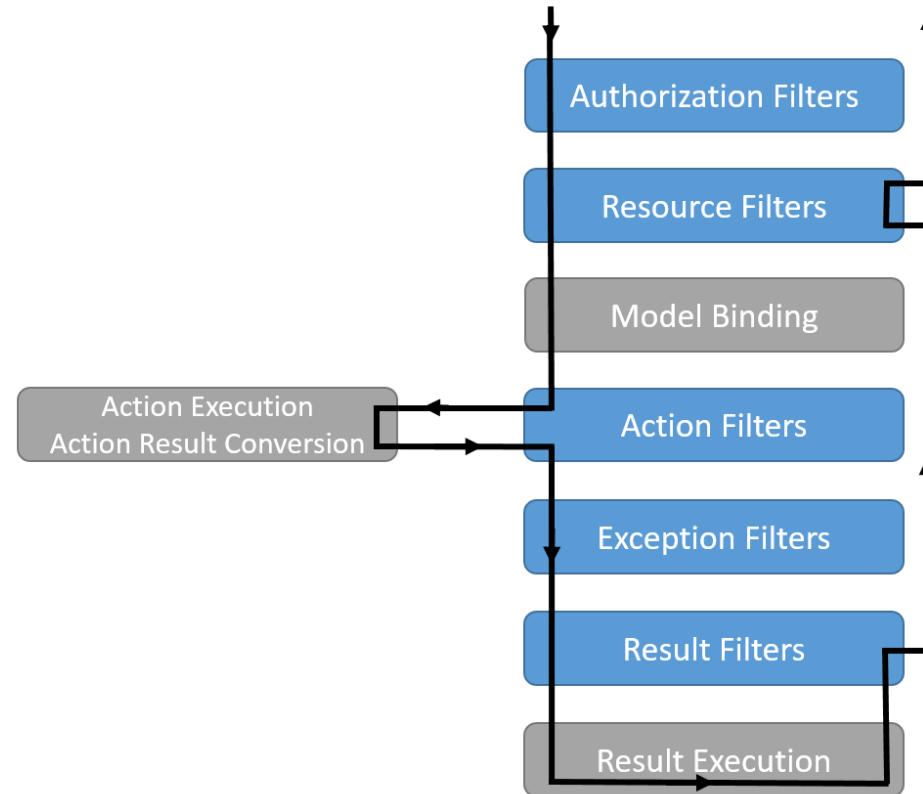
Routing to controller actions

```
app.MapDefaultControllerRoute(); //same as below
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

- MapControllerRoute and MapAreaRoute
 - Assign an order value based on order they are registered
 - Usually these 2 suffice for most apps
 - Good starting point
- Area? Organize related functionality into a group
 - namespace for routing
 - folder structure for views / ctrl with attr [Area("Admin")]
- Attribute Routing also available
 - [HttpGet("api/products/{id}")]

Action Result & Selectors

- Attributes
 - HttpGet
 - HttpPost
 - Route
- Filters
 - Authorization filter
 - Resource filters
 - Action filter
 - Exception filter
 - Result filter



Action Results

Action Result	Helper Method	Description
ViewResult	View	Renders a view as web page
PartialViewResult	PartialView	Renders a partial view, a section of a view that can be rendered inside another view.
RedirectResult	Redirect	Redirects to another action method by using URL
RedirectToRouteResult	RedirectToAction or RedirectToRoute	Redirects to another action method
ContentResult	Content	Returns a user-defined content type
JsonResult	Json	Returns serialized Json object
StatusCodeResult	StatusCode(int statusCode)	Returns a specific HTTP response code and description.
OkResult	Ok	Returns an empty object with 200 OK status code
UnauthorizedResult	Unauthorized	Returns the result of an unauthorized HTTP request. → 401
ForbiddenResult	Forbidden	Returns forbidden → 403
NotFoundResult	NotFound	Indicates the requested resource was not found. → 404
FileResult	File	Returns binary output to write to the response.
FileContentResult	Controller.File(Byte[], String (,String))	Sends the file content as an in-memory byte array
PhysicalFileResult	PhysicalFile	Sends an on-disk file identified by a physical path
VirtualFileResult	VirtualFile	Sends a file identified by a virtual path
FileStreamResult	Controller.File(Stream, String (,String))	Sends binary content to the response through a stream.
EmptyResult	(None)	Return value that is used if the action method must return a null result (void).

ActionResults

- No longer used in ASP.NET Core / replaced by others:

Action Result	Helper Method	Description
JavaScriptResult	JavaScript	Returns a script that can be executed on the client. → ContentResult
HttpStatusCodeResult	(None)	Returns a specific HTTP response code and description. → StatusCodeResult
HttpUnauthorizedResult	(None)	Returns the result of an unauthorized HTTP request. → UnauthorizedResult
HttpNotFoundResult	HttpNotFound	Indicates the requested resource was not found. → NotFoundResult
FilePathResult	Controller.File(String, String (,String))	Sends the contents of a file to the response. → VirtualFileResult or PhysicalFileResult

Controller - Helper Method

```
public IActionResult GetPerson()
{
    var person = new
    {
        FirstName = "John",
        LastName = "Doe",
        Age = 30,
        Email = "john.doe@example.com"
    };
    return Json(person);
}
```

Custom Action Filter

Base `ActionFilterAttribute` class has the following methods that you can override:

- `OnActionExecuting` – called **before** a controller **action** is executed.
- `OnActionExecuted` – called **after** a controller **action** is executed.
- `OnResultExecuting` – called **before** a controller action **result** is executed.
- `OnResultExecuted` – called **after** a controller action **result** is executed.

```
public class LogActionFilter : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        Log("OnActionExecuting", filterContext.RouteData);
    }
    ...
}
```

Views

- Default name of the action
 - Can be overridden
- .cshtml → Razor pages
- Fetched via HttpGet, parameters via query string. No state modification.
- Send back data via HttpPost, data goes via payload, state can be adjusted.
- Shared folder containsshared items :-)
- Model binding
- Only UI logic in views, no actual logic!

```
<form name="login" method="post" action="/login/">  
    <input type="text" name="username" required />  
    <input type="password" name="password" required />  
    <input type="submit" />  
</form>
```


Razor

- Markup syntax
- General purpose templating engine
→ no direct connection with ASP.NET MVC
- Mix of HTML and C# code
- Successor of .aspx files
- @ for code
- Html / Tag Helpers
- New: @await in view
- No generics (<int> carets interpreted as html)

AXXES_

```
@using System.Linq
@model List<PersonViewModel>

@if (Model.Any())
{
    <ul>
        @foreach (var person in Model)
        {
            <li>@person.Name - Age: @person.Age</li>
        }
    </ul>
}
else
{
    <p>No people to display.</p>
}

<form method="post">
    @Html.Label("name", "Name:")
    @Html.TextBox("name")
    <br />
    @Html.Label("age", "Age:")
    @Html.TextBox("age")
    <br />
    <button type="submit">Add Person</button>
</form>
```

Code in Razor

- `@{ //c# here }` → executes code but doesn't output anything
- `@DateTime.Now` → implicit razor expression without block, outputs date
- `@(...) //explicit expression` → outputs code
- `<p>Last week: @(DateTime.Now - TimeSpan.FromDays(7))</p>` → subtracted
`<p>Last week: @DateTime.Now - TimeSpan.FromDays(7)</p>` → rendered with –
 - So no space, except between parentheses, braces or after `@await`
- Strings are html encoded! Except `HtmlHelper.Raw` and `IHtmlContent`
- Local functions in code block possible for templating

Razor

Razor keywords

- model
- section
- functions
- helper (Not currently supported by ASP.NET Core)
- inherits

C# Razor keywords

- case
- do
- default
- for
- foreach
- if
- else
- lock
- switch
- try
- catch
- finally
- using
- while

Razor syntax, not Razor Pages

- `@page` directive on first line of file
- The main difference with MVC: the model and controller code is included within the page itself. This eliminates the need to add code separately.
- `.cshtml` and `.cshtml.cs` code-behind file, but no controllers
- Simpler and easier for basic pages
- Only 1 root level folder: "Pages" as opposed to 3 with MVC
- Less scalable than MVC
- You CAN use Razor Pages in MVC, simply add a "Pages" folder and add Pages to it
- While it uses a lot of the MVC features, it follows more of a Model-View-ViewModel (MVVM) pattern

AXXES_



wwwroot

- Web root folder
- Serve static files
 - Css
 - Html (like error pages etc)
 - Js
 - Fonts
 - Icons
 - Images
- Also lib static files like jquery, bootstrap, ...

Layout

- `_Layout.cshtml` by convention
- Assign Layout in `_ViewStart` or in View itself
 - ```
@{
 Layout = "_Layout";
}
```
- `@RenderBody()` → renders the contents of the child view
- Can be nested, set layout to parent layout in child layout:
- DRY → Don't Repeat Yourself and put things in Layout file
- Typically Header, footer, sidebar, search, ...
- Can also use Partial views in Layout

```
@{
 Layout = "../Shared/_Layout.cshtml";
}
<div style="border: 1px solid red">
 @RenderBody()
</div>
```

## \_ViewImports.cshtml

- Centralise directives that apply Views to avoid having to add them everywhere
- **@namespace** directive is used to specify the namespace of the pages affected by the ViewImports file. Only 1 per file

@using MyApplication

@namespace MyApplication.Pages

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@tagHelperPrefix th:

- @addTagHelper
- @inherits
- @namespace
- @inject
- @model
- @removeTagHelper
- @tagHelperPrefix
- @using

## `_ViewStart.cshtml`

- Code that needs to be executed before each page
  - Setting Default Layout
  - Common ViewBag data
- Should never have html tags inside it
- MVC will look for a ViewStart file in the same folder as the requested View first and then move up the folder hierarchy



## Render Section

- Render section in Layout file
- @section scripts in views
- Gets rendered in layout block
- Useful for Scripts and Styles
- Also perhaps Meta tags or SEO, or for widget or sidebar content ...
- Inject C# values in your JavaScript or CSS!

```
_Layout.cshtml:
 @RenderSection("scripts", required: false)
```

```
View.cshtml
 @section scripts { <script>...</script> }
```

```
<script>
 var jsValue = '@Model.CSharpValue';
</script>
<style>
 .my-element {
 background-color: @Model.CssColor;
 }
</style>
```

## Partial views

- Reusable
- Break up large markup files into smaller components.
- Reduce the duplication of common markup content across markup files.
- Rendered directly in a view (or passed via controller as a PartialViewResult)
  - Tag helper:  
`<partial name="_PartialName" />`
  - Async html helper:  
`@await Html.PartialAsync("_PartialName")`
  - @{  
    `await Html.RenderPartialAsync("_AuthorPartial");`  
} → doesn't return IHtmlContent but streams directly to the response
- Receives a copy of parent's ViewData dictionary on instantiation, changes not persisted back
- NO Layout file for partial views

## RenderPartial vs RenderAction

@Html.RenderPartial("\_LoginForm")

- Name of the partial view and returns an MvcHtmlString that represents the rendered partial view
- static view without executing any logic

@{

    Html.RenderAction("RecentNews", "Home");

}

- Name of action method and optionally controller name and route values
- Action result, which can contain dynamic content and execute logic

## Razor Pipeline

- Evaluates **\_ViewStart.cshtml**
  - Looks up the tree until it finds one, if any
- Then it parses and evaluates **View.cshtml** itself
- Then it parses and evaluates (if present) **Layout.cshtml**
  - Fills @RenderBody with result of View.cshtml

## Tag Helpers & Html Helpers

- Both are server-side code to help creating & rendering HTML
- Html Helpers were first, they are C# methods for generating HTML elements and attributes.
- Html Helpers allow for inline C# logic.
- Tag Helpers were created to provide more HTML-like syntax to enhance readability and maintainability.
- Tag helpers can modify the element it's attached to and within it
- Tag Helpers encourage SOC by minimizing C# code in views
- You can use both together.
- You can make custom helpers of both.
- There's no direct mapping between the two and tag helpers do not replace html helpers

Tag: `<label asp-for="FirstName" class="caption"></label>`

Html: `@Html.Label("FirstName", "First Name:", new {@class="caption"})`

## Custom Html Helper

```
public static IHtmlContent RelevantXkcd(this IHtmlHelper _, string id, string
description = "")
{
 TagBuilder tb = new TagBuilder("img");
 var xkcdImageUrl = $"https://imgs.xkcd.com/comics/{id}.png";
 tb.Attributes.Add("src", xkcdImageUrl);
 if (!string.IsNullOrEmpty(description))
 {
 tb.Attributes.Add("alt", description);
 }
 return tb;
}
```

- Html.ActionLink()
- Html.BeginForm()
- Html.CheckBox()
- Html.DropDownList()
- Html.EndForm()
- Html.Hidden()
- Html.ListBox()
- Html.Password()
- Html.RadioButton()
- Html.TextArea()
- Html.TextBox()
- ...

@Html.RelevantXkcd("woodpecker", "If you don't have an extension  
cord I can get that too. Because we're friends! Right?")



## Custom Html Helper - BAD

```
public static async Task<IHtmlContent> RelevantXkcd(this IHtmlHelper _, int id)
{
 TagBuilder tb = new TagBuilder("img");
 TagBuilder tba = new TagBuilder("a");

 var xkcdUrl = $"https://xkcd.com/{id}";
 var xkcdJsonUrl = $"{xkcdUrl}/info.0.json";

 tba.Attributes.Add("href", xkcdUrl);
 using HttpClient client = new HttpClient();
 try
 {
 var xkcdInfoJson = await client.GetStringAsync(xkcdJsonUrl);
 dynamic myObject = JsonConvert.DeserializeObject(xkcdInfoJson);
 tb.Attributes.Add("src", (string)myObject.img);
 tb.Attributes.Add("alt", (string)myObject.alt);
 }
 catch (System.Exception)
 {
 tb.Attributes.Add("src", "www.google.com/images/errors/robot.png");
 }
 tba.InnerHtml.AppendHtml(tb);
 return tba;
}
```

@await Html.RelevantXkcd(1)

```



```



## Tag Helpers

- Can combine multiple tag helpers on single element
  - Better Intellisense support than Html Helpers
  - You can disable a Tag Helper at the element level with the Tag Helper opt-out character ("!") `<!form ...`
  - `ITagHelperInitializer<TTagHelper>` can be used to configure all tag helper instances of a specific kind
    - `builder.Services.AddSingleton<ITagHelperInitializer<ScriptTagHelper>, AppendVersionTagHelperInitializer>();`
- Anchor Tag
  - Cache Tag
  - Component Tag
  - Distributed Cache tag
  - Environment Tag
  - Form Tag
  - Form Action Tag
  - Image Tag Helper
  - Input Tag Helper
  - Label Tag Helper
  - Link Tag
  - Partial Tag
  - Script Tag
  - Select Tag
  - Textarea Tag
  - Validation Message Tag
  - Validation Summary Tag



## Custom Tag Helper

```
public class BoldParagraphTagHelper : TagHelper
{
 public override void Process(TagHelperContext context, TagHelperOutput output)
 {
 if (context.TagName == "p")
 {
 output.Attributes.SetAttribute("style", "font-weight: bold;");
 }
 }
}
```

`<p bold>Text with bold font</p>`

## Passing data

- **Model**
- ViewBag
- ViewData
- TempData
- Session
- Cookies
- Query strings
- Hidden fields (form)
- HttpContext.Items
- Cache

# Model

- Application Data
  - Business logic
  - Storing & Retrieving data
- ASP.NET MVC does not specify how exactly you take care of this
- Validation
- Business objects, data entities, ViewModels, DTO's, ...
  - Depending on app needs: mix and match!
  - Consistency!
- Often it is opted to use view-specific ViewModels to present in the Views and map them where needed
  - Promotes decoupling & SOC
  - Use Service/Business layer to do the work
  - Helps keep views simple

## Model binding

[FromQuery] - Gets values from the query string.

[FromRoute] - Gets values from route data.

[FromForm] - Gets values from posted form fields.

[FromBody] - Gets values from the request body.

[FromHeader] - Gets values from HTTP headers.

## ViewBag

- Dynamic type property of Controller class
- Pass data from controller to view (NOT back)
- Is wrapper around ViewData
- Skips compile-time checking so can throw run-time exceptions
  - If wrong method is called on wrong type
  - If names in controller and view don't match when writing manually
- Try not to use ViewBag because you're too lazy to add to the model

```
@{
 ViewBag.Title = "Privacy Policy";
}
<h1>@ViewBag.Title</h1>
```

## ViewData

- Object dictionary
- Used as underlying dictionary for the ViewBag
- Can also be assigned to with the [ViewData] attribute in the controller
- Can use ViewData and ViewBag at the same time, but remember they refer to the same values
- ViewData is faster than ViewBag (although negligible probably)
- Returns Objects so need to cast

```
@{
 ViewData["Title"] = "Privacy Policy";
}
<h1>@ViewData["Title"]</h1>
```

```
[ViewData]
public string Title { get; set; }
```

## TempData

- Stores the data temporarily and **automatically removes it after retrieving a value.**
  - `TempData.Keep("name");` // Marks the specified key in the TempData for retention.
  - `TempData.ContainsKey("name")`
  - `@TempData.Peek("Message")`
- Controller to view
- View to controller
- Between action methods
- Between controllers
- Try to use only for redirects
- [TempData] attribute available
- `.AddSessionStateTempDataProvider();`

## Session

- Store maintained by the app to persist data across requests from a client
- Session state by providing a cookie to the client that contains a session ID
- `builder.Services.AddDistributedMemoryCache();`
- `builder.Services.AddSession(options =>`
- `app.UseSession());`



## HttpContext.Items

- Store data while processing a single request
- Discarded after request is processed
- Often used for middleware to communicate between different points in the pipeline
- Other code can also access it

# Forms

## Weakly typed:

```
[HttpPost] public IActionResult AddUser(string name,
string email)
```

```
<form action="AddUser" method="post">
 <table>
 <tr>
 <td>Enter Name: </td>
 <td><input type="text" name="name" /></td>
 </tr>
 <tr>
 <td>Enter Email: </td>
 <td><input type="text" name="email" /></td>
 </tr>
 <tr>
 <td colspan="2">
 <input type="submit" value="Submit Form" />
 </td>
 </tr>
 </table>
</form>
```

## Strongly typed:

```
[HttpPost] public IActionResult AddUser(UserModel user)
```

```
<form asp-action="AddUser" asp-controller="Home" method="post">
 <div asp-validation-summary="ModelOnly" class="text-danger"></div>
 <div class="form-group">
 <label asp-for="Name"></label>
 <input asp-for="Name" class="form-control" />

 </div>
 <div class="form-group">
 <label asp-for="Email"></label>
 <input asp-for="Email" class="form-control" />

 </div>
 <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

# Form Validation

## Data Annotations at Model / Fluent Validation

```
[Required]
[MaxLength(20)]
public string? Name { get; set; }

[Required]
[EmailAddress(ErrorMessage = "Invalid Email Address")]
public string Email { get; set; }
```

## Validation messages in View:

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>

```


```
@section Scripts {
 @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

AXXES\_

UserModel

Name

Email

 Please enter a part following '@'. 'test@' is incomplete.

[Back to List](#)

## Security

- Authentication and Authorization in ASP.NET Core.
- Configuring authentication schemes (e.g., cookies, JWT).
- Role-based and policy-based authorization.
- Cross site request forgeries

## Security

- XSS
- CSRF



## Cross Site Request Forgeries

- Hidden input in <form>
- Cryptographic strong random number
- Client has to support cookies
- Controller action must contain ValidateAntiForgeryToken
- @Html.AntiForgeryToken() helper method in form

```
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public ActionResult Register(RegisterViewModel model){
 ...
}
```

# Authorization

- [Authorize] attribute
- Roles and Policies
  - [Authorize(Roles = "Admin")]
  - [Authorize(Policy = "MinimumAgePolicy")]

```
builder.Services.AddAuthorization(options =>
{
 options.AddPolicy("MinimumAgePolicy", policy =>
 {
 policy.RequireAuthenticatedUser();
 policy.RequireClaim("Age", "18");
 });
});
```

## Dependency injection

ASP.NET Core MVC provides three lifetime options for registered services:

- **Transient:** A new instance is created every time it's requested.
- **Scoped:** A single instance is created for each HTTP request.
- **Singleton:** A single instance is shared across the entire application's lifetime.

Controllers themselves are also managed by the DI container

Constructor injection: (injecting a logger here)

```
public class HomeController : Controller
{
 private readonly ILogger<HomeController> _logger;

 public HomeController(ILogger<HomeController> logger)
 {
 _logger = logger;
 }
}
```



## Testing

Through SOC, Unit testing for:

- Controllers to validate Action results and response generation
- Model Testing to verify behavior of model classes
- Additional unit testing for business layer if it exists
- Integration testing between Controllers and Views

## Bundling & Minification

- Before: BundleConfig.cs
- Then: BundleConfig.json
- Now: WebOptimizer / WebPack / Gulp → Choose your own
  - Configure in Program.cs
- Environment-based → only deploy bundled&minified files in production

Bundling combines multiple files into a single file. → less files / server requests

Minification removes unnecessary characters from code without altering functionality. → significant size reduction

## Bundling & Minification

- `dotnet add package LigerShark.WebOptimizer.Core`
- `app.UseWebOptimizer();`
  - (before `app.UseStaticFiles()`)
- `builder.Services.AddWebOptimizer(pipeline =>`  
    `{`  
        `pipeline.AddJavaScriptBundle("/js/custombundle.js", "/js/*.js");`  
    `});`

## Ajax calls

(in the past via API Controllers) AJAX Calls to controller

```
<script type="text/javascript">
function addUser(e) {
 e.preventDefault(); //to prevent submit of form and round trip
 var data = $("#AddUserForm").serialize();
 $.ajax({
 type: 'POST',
 url: '@Url.Action("AddUser","Home")',
 contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
 data: data,
 success: function (result) {
 alert('Successfully received Data ');
 },
 error: function () {
 alert('Failed to receive the Data');
 }
 })
}
</script>
```

onclick="addUser(event)" on button

## Alternative Design Patterns

- MVVM: Model – View – ViewModel
  - Similar to MVC, but the ViewModel aims to take away any UI logic so the view can concentrate on just rendering
- MVP: Model – View – Presenter
  - Evolved from MVC. Controller → Presenter but with a 1-1 relationship with the View, and an abstraction between the two
- MVI: Model – View – Intent
- VIPER: View – Interactor – Presenter – Entity – Router
- MVW: Model – View – Whatever

Swagger

<https://www.c-sharpcorner.com/article/swagger-for-net-core-mvc-web-api/>

AXXES\_

## Umbraco CMS

- Content Management System
- Define document types & data types
- Create templates → Razor & MVC!
- Partial views
- Content structured in a tree view, easy to create and structure
- Backoffice customization
- Packages and extensions available
- Large community, good documentation
- FREE



DIY



## New Project

```
> dotnet new sln --name MvcExample
> dotnet new mvc --name MvcExample.Web
> dotnet sln add MvcExample.Web
> (dotnet new gitignore)
```

(or the user-friendly way →)

### Configure your new project

ASP.NET Core Web App (Model-View-Controller)

C#

Linux

macOS

Windows

Project name

TestProject

Location

C:\git\Traineeship\MVC\

Solution

Create new solution

Solution name ⓘ

TestSolution

☐

Place solution and project in the same directory

Project will be created in "C:\git\Traineeship\MVC\TestSolution\TestProject\"

## Exercises

- Create Solution/Project
- Create model with a list of items, use a partial view in @foreach to render
- Execute javascript function on button click, pass message from C# code.
- Add dark theme!
  - Submit button
  - Class on body
  - Update class by ViewBag