howest
hogeschool

.NET core - MVC

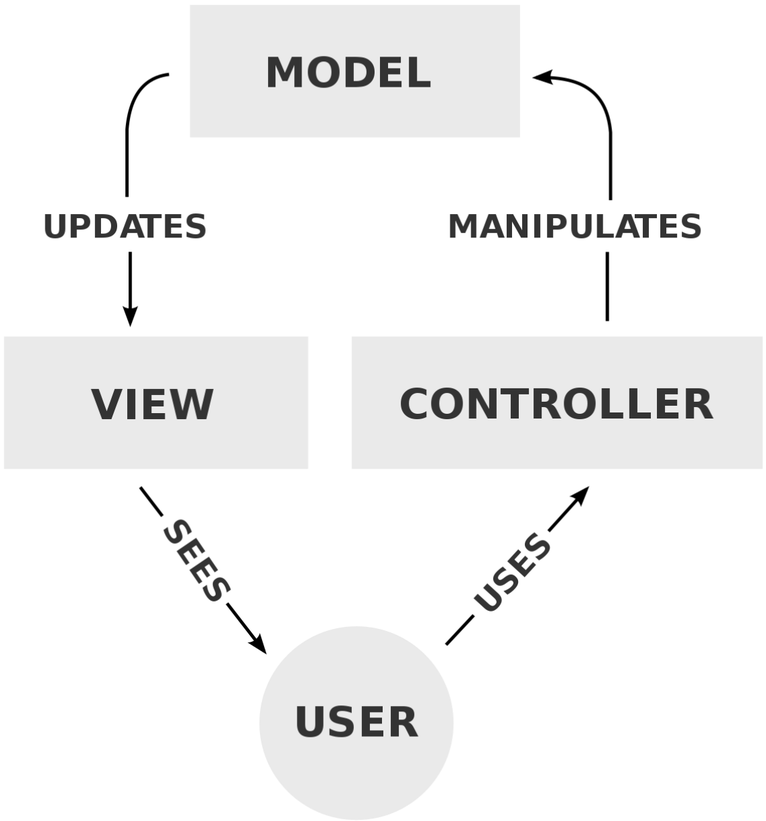
Model View Controller

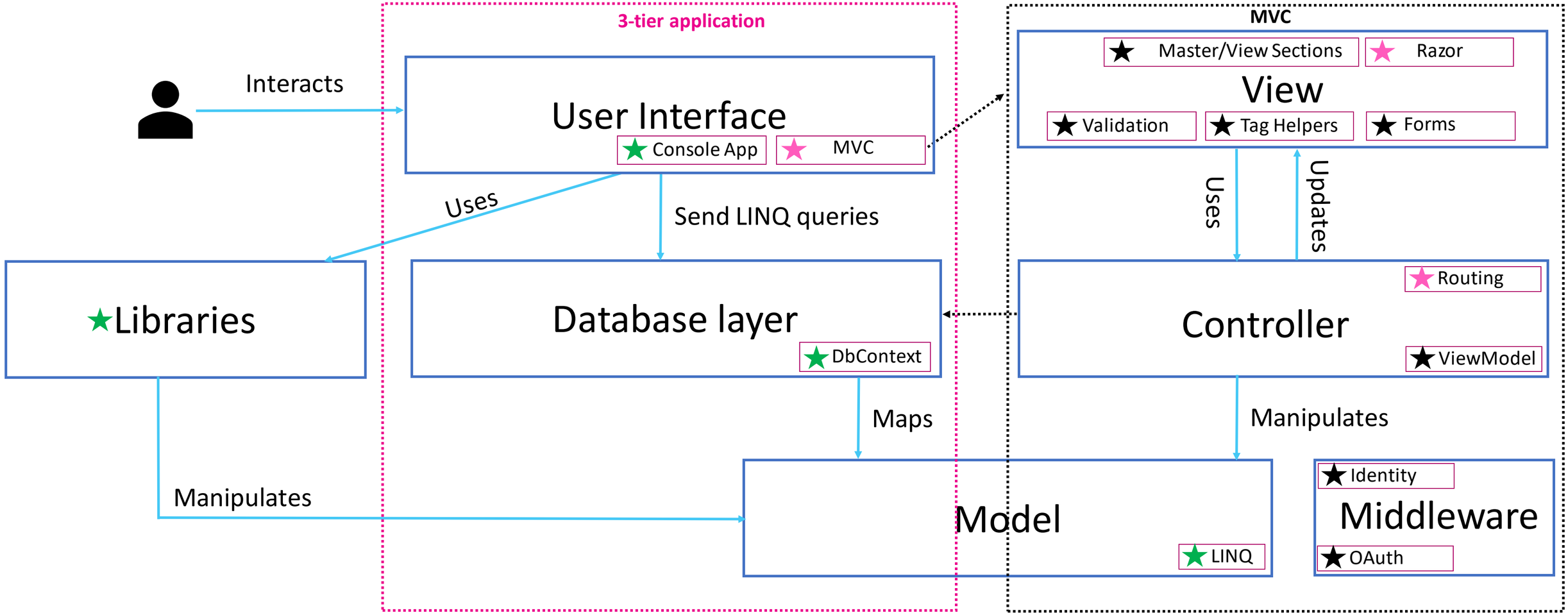
howest
hogeschool

Agenda

1. MVC: recap
2. Demo: controllers
3. MVC: controller
4. Demo: routing
5. MVC: routing
6. Demo: views
7. MVC: views

MVC: Recap





Roadmap .NET CCCP

Current topic(s)

Covered topic

Futured topic

howest hogeschool

howest
hogeschool

A very simple MVC application

DEMO Controllers

Creating & Running

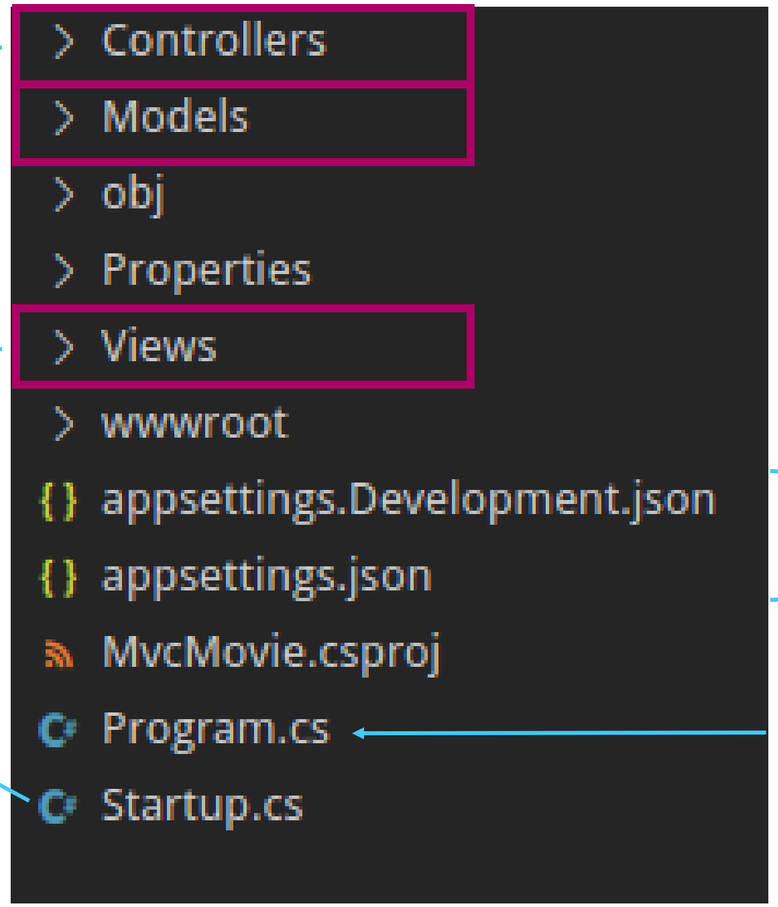
- Creating an MVC application
  - **dotnet new mvc –out *projectName***
  - *A* dummy mvc application is automatically created.

- Add the code generating package
  - This allow the developer easily create MVC objects.
  - **dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design**

- Running
  - Execute **dotnet run** from the project root folder.
  - By default: https://localhost:5001

Folder structure

MVC



Application configuration: server locations; ports, connection strings, custom configuration; ….

The Startup class is where **services** required by the app are **configured** and the r**equest handling pipeline** is defined.

- Services are components that are used by the app.
- The request handling pipeline is composed as a series of middleware components. For example, a middleware might handle requests for static files or redirect HTTP requests to HTTPS.

The Program class contains the entry method **Main**.

The Main method i**s like the Main method of a console Applications**. That is because all the .NET Core applications basically are console applications. We build other types of applications like MVC Web Application or Razor page application over the console app.

Controllers

- Controllers are components that are available for end users or other applications.
  - View, api's, ...
- They only contain orchestration logic between the model and the view. Keep them small!
- Controllers should be created in de **controller** folder by executing:
  - **dotnet aspnet-codegenerator controller –name *NameController* -outDir Controllers**
  - Navigate to the controller by surfing to https://localhost:5001/**HelloWorld**

The controller name ends with **Controller** but is called **without** it.

In controllers, public methods are called **actions.**

```
0 references
public class HelloWorldController : Controller
{
    0 references
    public string index()
    {
        return "This is my default action";
    }

    0 references
    public string Hello(string name, int numTimes = 1)
    {
        return HtmlEncoder.Default.Encode($"Hello {name}, you called {numTimes} times");
    }
}
```

Controllers: actions

- Actions are the **entry points** available to the end user or other applications by using the HTTP protocol. (eg. Api's).
- Actions are called by executing a **HTTP** command **GET /Controller/Action** (e.g. https://localhost:5001/HelloWorld/Hello)
- Actions are by default consumed by **HTTP GET**
- Every controller has one **default** action: **index**
  - Not necessary to explicitly mention index (e.g. https://localhost:5001/HelloWorld)

Actions are nothing more than plain old **public** methods.

```
0 references
public class HelloWorldController : Controller
{
    0 references
    public string index()
    {
        return "This is my default action";
    }

    0 references
    public string Hello(string name, int numTimes = 1)
    {
        return HtmlEncoder.Default.Encode($"Hello {name}, you called {numTimes} times");
    }
}
```

Controllers: actions: query parameters

- Query parameters can be provided to the actions of the controller.
- How?
  - The query parameters must match the name of the action parameters (case insensitive).
  - e.g. https://localhost:5001/HelloWorld/Hello?name=Matthias&numTimes=10
  - Actions are callable from any HTTP client (e.g. Postman)

Name and numTimes must be available in the query parameters.

```
0 references
public class HelloWorldController : Controller
{
    0 references
    public string index()
    {
        return "This is my default action";
    }

    0 references
    public string Hello(string name, int numTimes = 1)
    {
        return HtmlEncoder.Default.Encode($"Hello {name}, you called {numTimes} times");
    }
}
```

Controllers: why everything works automatically?

- Like many frameworks .NET core uses a lot of conventions.
- You like it or you don't.
- By naming the controllers correctly and placing them in the appropriate folder they are automatically detected.
- How?
  - Remember the definition of the **Startup.cs** file?

howest
hogeschool

A very simple MVC application

DEMO Routing

Routing

- Routing in the context of web applications is the **mapping** of HTTP requests to **certain controller actions.**

- A lot of default routing is already available in a .NET core MVC application (conventions).
  - The next slides explain how to do this manually.
  - Why? Some scenarios must be manually implemented and therefore you need to know the basics of routing.
    - Which ones? E.g. when the action name is not the desired name for the end users. (See slide custom routes.).
    - Or if you want to define a default controller.

- Routing can both be done in:
    - the **Startup.cs** file with **route definitions**
    - the controller with **route attributes** .
  - In this module **only the attribute routing** is covered!

Routing: attributes

- In .NET attributes are single lines of code above the start of a method/class that provides extra functionality.
- Define your route(s) as an attribute to **each** Action Method.
- Syntax:

Route attribute

```
[Route("..")]
```

Route attribute **value**

howest hogeschool

Routing: attributes

- The order in which the routes are interpreted is **not important**

/home/Details = route to the Details action
/home = route to the Index action (default action)
/home/Index = route to the index action

[controller] = use the name of this controller as part of the route.

[action] = use the name of this action as part of the route.

"" = use the name of this controller/action as the default controller/method. *(Warning only **one** controller and **one** action can use the **"" route attribute.)** See next slide.*

```
[Route("[controller]")]
public class HomeController : Controller
{
    [Route("")]
    [Route("[action]")]
    public string Index()
    {
        return "Index() Action Method of HomeController";
    }

    [Route("[action]")]
    public string Details()...
}
```

Routing: attributes

- **In this module we are always going to add routing attribute to all controllers and actions!**

```
[Route("[controller]")]
public class HomeController : Controller
{
    [Route("")]
    [Route("[action]")]
    public string Index()
    {
        return "Index() Action Method of HomeController";
    }

    [Route("[action]")]
    public string Details()...
}
```

Routing: attributes: custom routes

Sometimes the public available names of controllers and actions must be different.

Choose a name of choice!

```
[Route("my-app")]
0 references
public class MyUglyNamedController : Controller
{

    [Route("check")]
    0 references
    public string QuickDirtyMethod(string name, int numTimes = 1)
    {
        return $"Hello {name}, you called {numTimes} times";
    }
}
```

Syntax warning!

[Route(["actions"])] vs [Route("my-app")]

howest hogeschool

Routing: attributes: custom routes

Long paths are possible!

```
[Route("my-app")]
0 references
public class MyUglyNamedController : Controller
{

    [Route("check")]
    0 references
    public string QuickDirtyMethod(string name, int numTimes = 1)
    {
        return $"Hello {name}, you called {numTimes} times";
    }

    [Route("/my-app/my/very/long/name/check")]
    0 references
    public string QuickDirtyMethod2(string name, int numTimes = 1)
    {
        return $"Hello {name}, you called {numTimes} times";
    }
}
```

Routing: attributes: create default controller/action

Always create a default controller and action!

```
[Route("")]
[Route("[controller]")]
0 references | 0 changes | 0 authors, 0 changes
public class StudentController : Controller
{
    [Route("")]
    [Route("[action]")]
    0 references | 0 changes | 0 authors, 0 changes
    public IActionResult Index()
    {
        return View();
    }
}
```

Routing: attributes: create default controller/action

Caution: following code does not work!
The following code has **TWO** default actions.
As previously mentioned, only **ONE** default controller/action.
An action **without** attributes = **default action!**

Solution: add [Route("[action]"] to the Hello action.

```
[Route("")]
0 references
public class HelloWorldController : Controller
{
    [Route("")]
    0 references
    public string index()
    {
        return "This is my default action";
    }

    0 references
    public string Hello(string name, int numTimes = 1)
    {
        return $"Hello {name}, you called {numTimes} times";
    }
}
```

Routing: attributes: path parameters

Path parameters are variable parts of a URL **path**.
They are typically used to point to a specific resource within a collection, such as a user identified by ID.
A URL can have several **path parameters**,
each denoted with curly braces { } .

Path parameters are mapped onto the action parameters.

By default path parameters are mandatory!
Not providing them = page not found.

Just like query parameters:
Names must match!

```
[Route("")]
[Route("Index")]
[Route("Index/{naam}")]
0 references | 0 changes | 0 authors, 0 changes
public IActionResult Index(string naam)
{
    return Content(naam);
}
```

e.g. https://../Index/matthias
"matthias" is captured into the variable naam.

Routing: attributes: optional path parameters

It's is possible to use not mandatory path parameters.
Add a **?** to the end of the path parameter name.

Warning like in many programming languages: optional parameters must be **AFTER** the mandatory parameters and the **ORDER** matters.

Works:
https://../Student/Index/matthias
https://../Student/Index/matthias/33
https://../Student
https://../Student/Index

Doesn't work:
https://../Student/Index/33
https://../Student/matthias/33

```
public class StudentController : Controller
{
    [Route("")]
    [Route("[controller]/[action]/{naam?}/{leeftijd?}")]
    0 references | 0 changes | 0 authors, 0 changes
    public IActionResult Index(string naam, int leeftijd=0)
    {
        if (!string.IsNullOrEmpty(naam))
        {
            return Content($"{naam} is {leeftijd} jaar oud");
        }
        return View();
    }
```

howest
hogeschool

A very simple MVC application

DEMO Views

Views

- In the Model-View-Controller (MVC) pattern, the **view** handles the app's data presentation and user interaction.
- A view is an HTML template with embedded **Razor markup**.
- **Razor markup** is **C# code** that interacts with HTML markup to produce a webpage that's sent to the client.

Razor code = **C# code** (in HTML)

Code always starts with @

Uses

@{ … }

@(…)

@:

@* … * @

@if, @for, @foreach, …

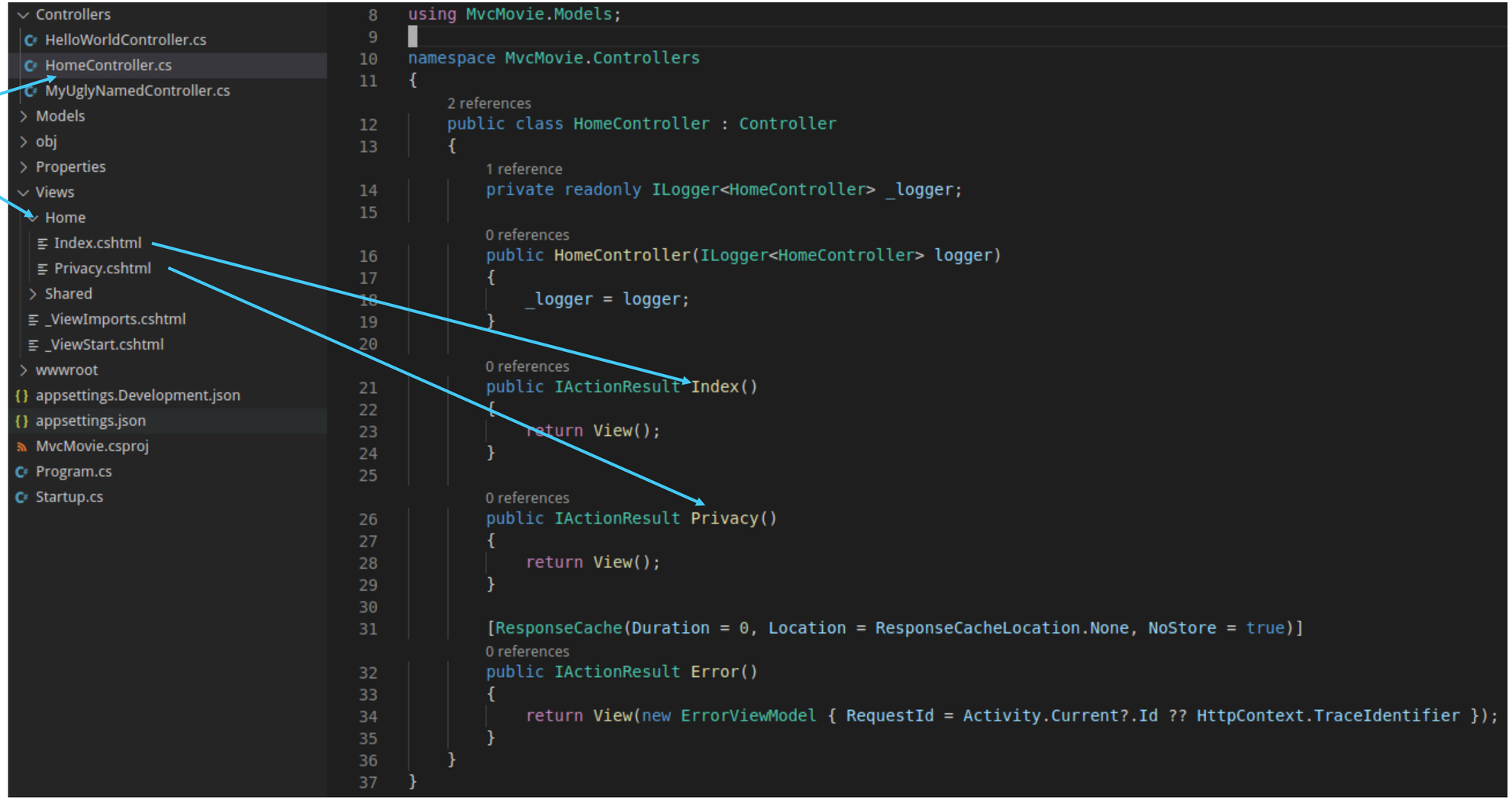
```
<ul>
  @foreach (var p in products) {
     <li>
         @p.ProductName

         @if (p.UnitsInStock == 0) {
            @: (Out of stock!)
         }
         else if (p.UnitsInStock < 4) {
            @: (Only @p.UnitsInStock left!)
         }
     </li>
  }
</ul>
```

Views: conventions

Controller name =
View folder name

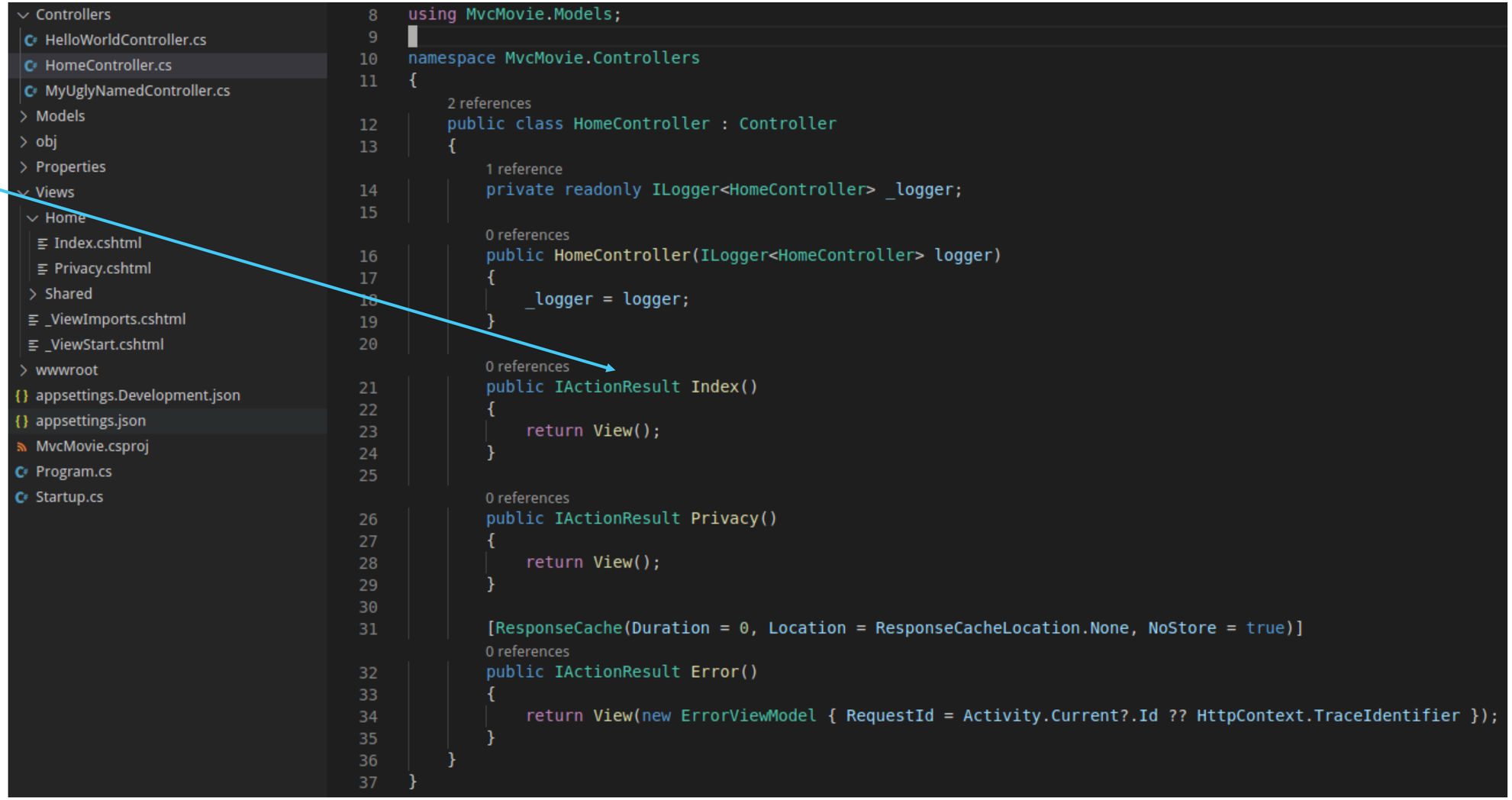
Action name =
View name
One file per action!



Views: IActionResult

IActionResult represents various HTTP status codes.

Return View(); results in an IActionResult of type OK.



Passing models to views.

Pass the model to the view functions.

```
[Route("[action]")]
0 references
public IActionResult Index()
{
    var categories = categoryService.AllCategories();
    return View(categories);
}

[Route("Categories/{categoryId}/Products")]
0 references
public IActionResult GetProductsByCategoryId(int categoryId) {
    var products = productService.GetProductsByCategoryId(categoryId);
    return View(products);
}
```


howest hogeschool

Passing models to views.

IMPORT the model with
@model (mind the lowercase m)

USE the model with
@Model.PropertyName (mind the uppercase M)
All the properties of the passed object are available.

```
@model IEnumerable<northwind_app.Library.Models.Categories>
<h1>Overview</h1>
<table class="table">
    <thead>
        <tr>
            <th>@Html.DisplayNameFor(Model => Model.CategoryName)</th>
            <th>@Html.DisplayNameFor(Model => Model.Description)</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var category in Model) {
        <tr>
            <td>@category.CategoryName</td>
            <td>@category.Description</td>
            <td>
                <a asp-controller="Overview"
                   asp-action="GetProductsByCategoryId"
                   asp-route-categoryId="@category.CategoryId">
                   Products
                </a>
            </td>
        </tr>
}
    </tbody>
</table>
```

Views: razor syntax: mix models, logic and html.

- **Again: Razor markup** is **C# code** that interacts with HTML markup to produce a webpage that's sent to the client.
- Don't google endlessly but use the following reference:
    - https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-3.1

MVC: reference

MVC tutorial (skip database part):

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-3.1

Views (skip tag helpers):

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-3.1

Attribute Routing:

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-3.1#attribute-routing