howest
hogeschool

.NET core - MVC - II

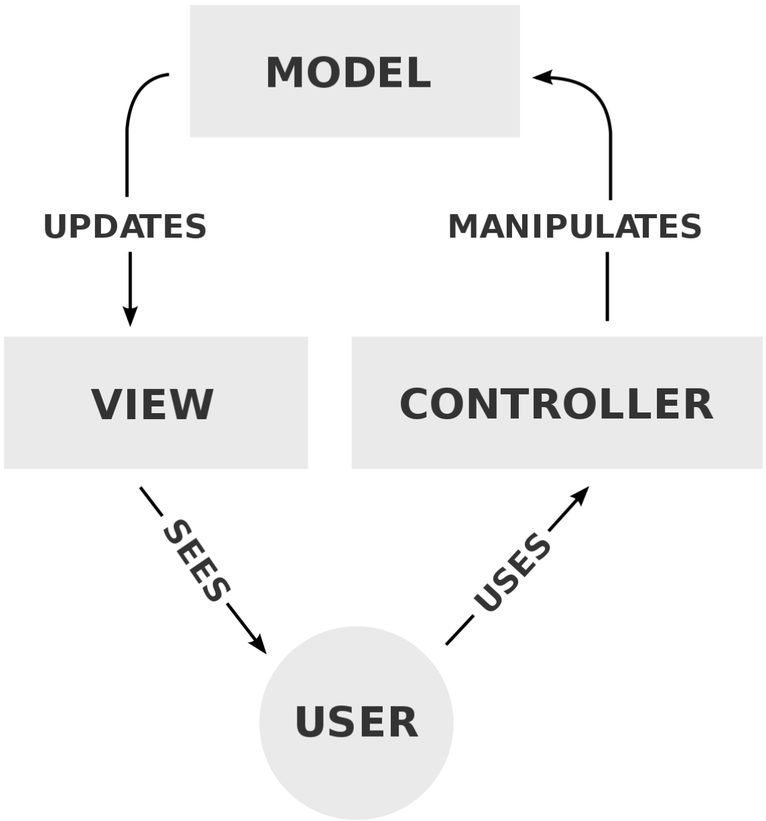
ViewModels, Forms, TagHelpers

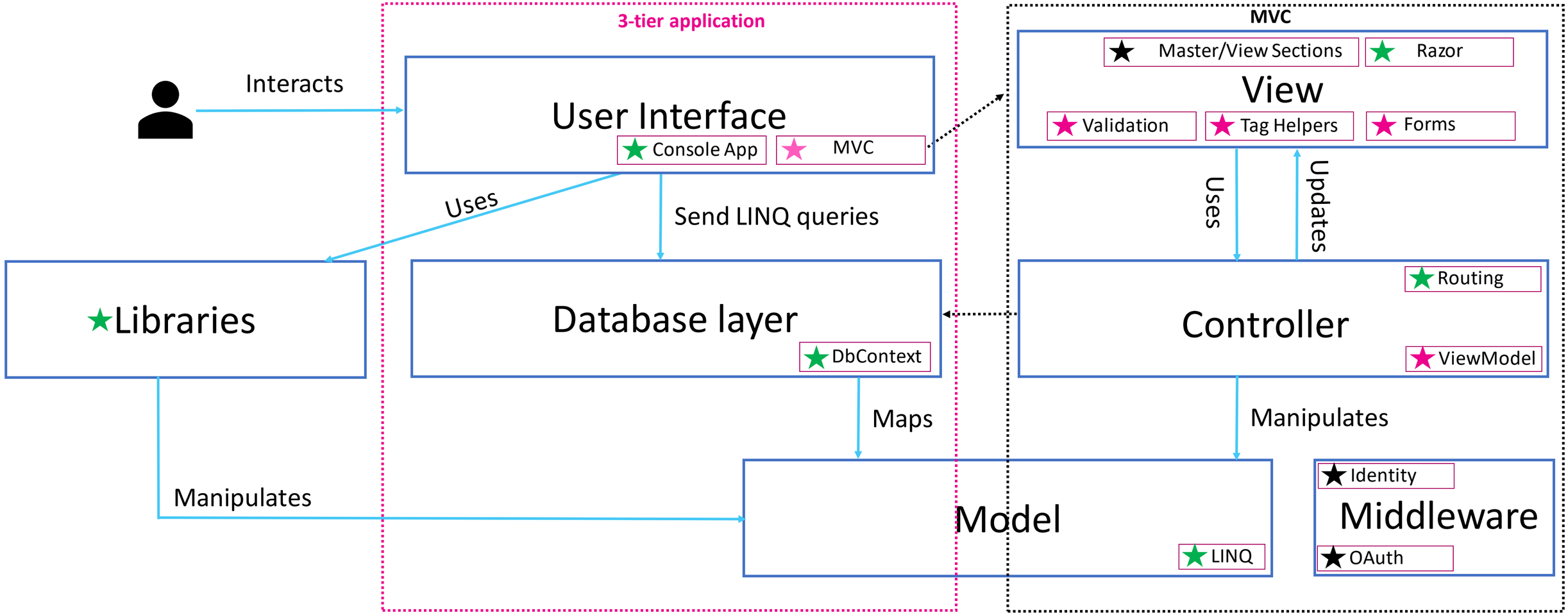
howest
hogeschool

Agenda

1. ViewModels
2. Demo: ViewModels
3. Forms & TagHelpers
4. Demo: Forms & TagHelpers
5. Form validation
6. Demo: Form validation
7. Sessions
8. Demo: Sessions

MVC: Recap





Roadmap .NET CCCP

Current topic(s)

Covered topic

Futured topic

howest hogeschool

howest
hogeschool

MVC – Part II

DEMO ViewModels

What is a ViewModel

- **Model**
  - Logical entities of an application.
  - Can be seen as a business class.
  - Uses OO-concepts as composition, inheritance, …
- **ViewModel**
  - **Class** tweaked to a **View** of the application.
  - Only contains the needed properties.
  - Gathers information out of different model classes (mostly).
  - Can contain extra information: totals, averages, …

Why a ViewModel

- Adapt SQL-request
  - Load a Customer from the Customers context could give us:

Alfreds Futterkiste

- Sales Representative: Maria Anders
- Berlin, Germany

- But **all data** is loaded, even if you only need the name.
  - Be careful what you wish for!


```
SELECT TOP(1) [c].[CustomerID], [c].[Address], [c].[City], [c].[CompanyName],
[c].[ContactName],    [c].[ContactTitle],[c].[Country], [c].[Fax], [c].[Phone],
[c].[PostalCode], [c].[Region]
FROM [Customers] AS [c]
```

Why a ViewModel

- Avoid **over-posting / mass assignment** vulnerability

```
public class User
{
    public string FirstName { get; set; }
    public bool IsAdmin { get; set; }
}
```

```
public class UserInputViewModel
{
    public string FirstName { get; set; }
}
```

If a controller accepts this **User** in an http-post scenario, the request object could be adapted (Fiddler, Postman, …)

ViewModels: how

- Create folder **ViewModels** at the root of your project.
- Create a subfolder with the name of the controller.
- Create ViewModel Class
- Adjust controller method
- Adjust View

It's just a class.

Nesting is possible.

```
namespace northwind_app.ViewModels.Overview
{
    4 references
    public class CategoriesViewModel {
        2 references
        public string TitlePage {get; set;}
        2 references
        public string TitleCategoryName {get; set;}
        2 references
        public string TitleCategoryDescription {get; set;}
        2 references
        public string TitleLink {get; set;}
        2 references
        public List<CategoryViewModel> Categories {get; set;}
    }
}
```

```
@model northwind_app.ViewModels.Overview.CategoriesViewModel;
```

```
public IActionResult Index()
{
    return View(new CategoriesViewModel
    {
        Categories = categoryService
            .AllCategories()
            .Select(c => new CategoryViewModel{
                Description = c.Description,
                Name = c.CategoryName,
                Id = c.CategoryId
            })
            .ToList(),
        TitlePage = "Overview",
        TitleCategoryName = "Category name",
        TitleCategoryDescription = "Category description",
        TitleLink = "Products"
    });
}
```


howest hogeschool

howest
hogeschool

MVC – Part II

Tag Helpers & Forms

What are tag helpers.

- Back to normal HTML-tags, but with special .NET core attributes.
- Extremely suitable for front-end developers without any knowledge of C#!

```
<a   asp-controller="Overview"
     asp-action="GetProductsByCategoryId"
     asp-route-categoryId="@item.Id">
     @Model.TitleLink
</a>
```

Last part = name of **path parameter** or **query parameter.**

Tag Helpers examples:

Defines where the post request should arrive.

Enumeration of validation errors.

Asp-for = the html for attribute.

If the input field **Name** gives any validation errors, display them in this span attribute.

```
<form
    asp-controller="Overview"
    asp-action="SaveProduct"
    method="Post">

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

    <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>

    <div class="form-group">
        <input type="submit" value="Save" class="btn btn-primary" />
    </div>
</form>
```

Some HTML elements and there supporting tag helpers.

asp-controller	<form>, <a>
asp-action	
asp-for	<input>, <textarea>, <select>
asp-validation-for	
asp-validation-summary	
asp-items	<select>, <option>
asp-route	
asp-route-key	<form>

key = custom route data

howest hogeschool

Forms: how: views

```
@model northwind_app.ViewModels.Overview.ProductViewModel
```

1) Define where the post body is going.

```
<form
    asp-controller="Overview"
    asp-action="SaveProduct"
    method="Post">

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

    <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>

    <div class="form-group">
        <input type="submit" value="Save" class="btn btn-primary" />
    </div>
</form>
```

2) **asp-for** must equal a property name of the **@Model object**!

Forms: how: controllers

Create the proper actions in the controller that was marked as destination in the view.

This **action** is needed to show the CreateProduct **view**.

Already prefill the form with some data (categoryId).
[HttpGet] is not needed (default).

This prefilling is off course not mandatory.

```
[Route("[action]/{categoryId}")]
0 references
public IActionResult CreateProduct(short categoryId) {
    return View(new ProductViewModel {
        CategoryId = categoryId
    });
}
```

This **action** receives the post body.
The body values automatically match the model properties.
(If equally named)
The **action** must be annotated with **[HttpPost]**

```
[HttpPost]
[ValidateAntiForgeryToken]
[Route("[action]")]
0 references
public IActionResult SaveProduct(ProductViewModel product) {
    productService.Add(product.Name, product.Price, product.CategoryId);

    return RedirectToAction( "GetProductsByCategoryId",
        "Overview",
        new {CategoryId = product.CategoryId});
}
```

?

Forms: how: controllers

- Allows redirection to **another action** as a result.
- The result type of **RedirectToAction == View().**
- This redirect directs the user back to the action **GetProductsByCategoryId** from the controller **Overview**.
- It is also possible to choose a view of choice by using **return View("ViewName").**

```
[Route("[action]/{categoryId}")]
0 references
public IActionResult CreateProduct(short categoryId) {
    return View(new ProductViewModel {
        CategoryId = categoryId
    });
}

[HttpPost]
[ValidateAntiForgeryToken]
[Route("[action]")]
0 references
public IActionResult SaveProduct(ProductViewModel product) {
    productService.Add(product.Name, product.Price, product.CategoryId);

    return RedirectToAction( "GetProductsByCategoryId",
        "Overview",
        new {CategoryId = product.CategoryId});
}
```

howest
hogeschool

MVC – Part II

Form Validation

Form validation: view

Enumeration of ModelState errors.

If the input field **Name** gives any validation errors, display them in this span attribute.

But what is invalid input?

```
<form
    asp-controller="Overview"
    asp-action="SaveProduct"
    method="Post">

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

    <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>

    <div class="form-group">
        <input type="submit" value="Save" class="btn btn-primary" />
    </div>
</form>
```

Form validation: model

- Any type of model can be annotated with **validation *attributes*.**
  - **Using System.ComponentModel.DataAnnotations.**
  - Add validation attributes on top of the model property.

- Examples
  - Required
  - Range
  - Compare
  - RegularExpression
  - MinLength, MaxLength, StringLength
  - Url, Phone, EmailAddress
  - Display
  - DataType

```
using System.ComponentModel.DataAnnotations;
using northwind_app.Library.Models;

namespace northwind_app.ViewModels.Overview
{
    7 references
    public class ProductViewModel {

        [Required]
        [Range(typeof(float), "1", "1000")]
        6 references
        public float? Price {get; set;}

        [Required]
        [MinLength(3), MaxLength(50)]
        6 references
        public string Name {get; set;}

        [Required]
        7 references
        public short CategoryId {get; set;}
    }
}
```

Form validation: controller

- Use the build-in property **ModelState.IsValid**
- On validation errors, this property will be false.
- Redirect back to the original view (CreateProduct).
- Pass this view the incorrect model.
- Add a c**ustom error to the Model state** if needed.

```
[HttpPost]
[ValidateAntiForgeryToken]
[Route("[action]")]
0 references
public IActionResult SaveProduct(ProductViewModel product) {
    if (ModelState.IsValid) {
        productService.Add(product.Name, product.Price, product.CategoryId);

        return RedirectToAction( "GetProductsByCategoryId",
            "Overview",
            new {CategoryId = product.CategoryId});
    }

    ModelState.AddModelError("", "Please fill in all required fields.");
    return View("CreateProduct", product);
}
```

Form validation: result

Add

Product

- Please fill in all required fields.

Name

The Name field is required.

Price

1234

The field Price must be between 1 and 1000.

CategoryId

2

Save

Back

howest hogeschool

howest
hogeschool

MVC – Part II

Sessions

Keeping state: overview

View state

Hidden fields

Cookies

Query strings

Application state

Session state

Profile Properties

howest hogeschool

Keeping state: what

- Session state is an ASP.NET Core scenario for storage of user data.
- Session state uses a store maintained by the app to persist data across requests from a client.
- The session data is backed by a cache and considered ephemeral (short lived) data.

Keeping state: how

- The Microsoft.AspNetCore.Session package is **included** implicitly by the framework and provides middleware for managing session state

- In **Startup**.cs, method ***ConfigureServices***(…):

```
services.AddDistributedMemoryCache();
services.AddSession(options =>
{
    options.Cookie.Name = ".Meals.Session";
    options.Cookie.IsEssential = true;
});
```

- In Startup.cs, method ***Configure***(…):

```
app.UseSession();
```

Keeping state: how

```
// Needed to use the session store.
using Microsoft.AspNetCore.Http;
// Needed to use Json.
using System.Text.Json;
```

```
1 reference
private List<string> StoreSearchQuery(string query) {
    // Add the search string to the session state to remember it.
    List<string> previousQueries = GetPreviousQueries();

    if (!previousQueries.Contains(query)) {
        previousQueries.Add(query);
    }

    // Overwrite the current searchQueries with the new results.
    HttpContext.Session.SetString("searchQueries", JsonSerializer.Serialize(previousQueries));

    return previousQueries;
}
```

Overwrite the previous data with the new data.

```
2 references
private List<String> GetPreviousQueries(){
    // Get the raw json from the session store.
    string previousQueriesJson = HttpContext.Session.GetString("searchQueries");
    List<string> previousQueries = new List<string>();

    if (!string.IsNullOrEmpty(previousQueriesJson)) {
        // When the searchQueries key is available deserialize into a list of strings.
        previousQueries = JsonSerializer.Deserialize<List<string>>(previousQueriesJson);
    }

    return previousQueries;
}
```

Get the session data from the HttpContext by key.

Deserialize into the **desired** type.

howest hogeschool