HoGent

BEDRIJF
EN
ORGANISATIE

Hoofdstuk 9: Display/Edit annotaties, Validatie en Authenticatie/Authorisatie

https://github.com/WebIII/09thBeerhallVal.git
https://github.com/WebIII/09thBeerhallAuth.git

Validatie en Authenticatie/authorisatie

1. Inleiding
2. Display/Edit annotaties
3. Validatie
4. Authenticatie
5. Authorisatie
6. async en await
7. OAuth en OpenId
8. CSRF: Cross-Site Request Forgery
9. Appendix: Hashing
10. Referenties

1. Inleiding

- We breiden de Beerhall applicatie verder uit
  - **Display/Edit annotaties**
  - **Validatie**
  - **Authenticatie en authorisatie**
    - **Authenticatie**: Gebruiker moet aanmelden en wordt al dan niet geauthenticeerd
    - **Authorisatie**: éénmaal de gebruiker geauthenticeerd is, krijgt de gebruiker bepaalde rechten in de applicatie afhankelijk van zijn rol/claims.
  - Opmerking: er werden een aantal extra properties aan BrewerEditViewModel toegevoegd…

Validatie

THE
BEERHALL

Validatie

HoGent

2. Validatie

- Validatie in MVC kan gebeuren aan de hand van annotaties. De annotaties brengen we **1 keer** aan in ons model (of viewmodel) en worden dan gebruikt voor **én client side validatie én server side validatie**…
  - de annotaties zijn voorgedefinieerde attributen
  - **we zullen eerst enkele annotaties bespreken die niet gebruikt worden voor validatie**

Overzicht [bewerken]

Aspectoriëntatie is bedoeld als antwoord op een probleem waar alle "klassieke" paradigma's mee kampen, namelijk de vraag hoe om te gaan met de zogeheten *crosscutting concerns*: handelingen die door het hele programma heen uitgevoerd moeten worden. Typische voorbeelden hiervan zijn logging en beveiliging: op vrijwel ieder punt in een gemiddeld programma bestaat de behoefte aan de mogelijkheid om zaken naar een log weg te schrijven om fouten te traceren en zeer veel programma's moeten op verschillende punten controleren of de gebruiker van het programma wel gerechtigd is om de opgevraagde handeling uit te voeren. De "klassieke" paradigma's hebben hiervoor geen makkelijke oplossing en vallen daarom terug op het meerdere malen opnemen van precies dezelfde code op iedere plaats waar dezelfde handeling uitgevoerd wordt. Met als resultaat dat een verandering aan de uitvoering van een dergelijk crosscutting concern betekent dat door het hele programma heen code moet worden aangepast.

Als oplossing hiervoor biedt AOD (Aspect Oriented *Development*) de mogelijkheid een stukje code te schrijven dat op een groot aantal, door de programmeur te definiëren punten, ingevoegd wordt. Dit invoegen gebeurt zonder dat de geschreven code **waarin** ingevoegd wordt, aangepast wordt op de invoeging (dat er op een gegeven plaats iets ingevoegd wordt, kan men dus niet zien aan de programmacode). Om dit voor elkaar te krijgen, geeft de programmeur extern aan de programmacode een aantal punten op waarin bepaalde code ingevoegd dient te worden. Een apart programma neemt de code van de programmeur en zijn aanwijzingen over invoegen en stelt daarmee een nieuw programma samen waarin de juiste code op de juiste plaats ingevoegd is.

Er zijn in principe twee tijdstippen waarop het *weven* (het samenvoegen van code) kan plaatsvinden: tijdens de compilatie van code naar programma (dit wordt *statisch weven* genoemd) en tijdens het uitvoeren van het gecompileerde programma (*dynamisch weven*). In het eerste geval leeft de AOD-uitbreiding op de bestaande techniek in de compiler, in het tweede geval wordt de uitbreiding extern aangebracht via een preprocessor.

2. Display/Edit Annotaties

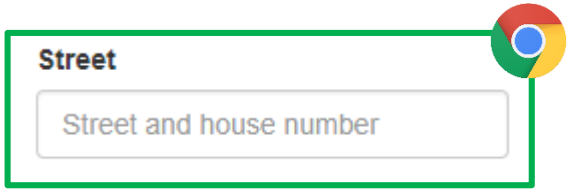
- **System.ComponentModel.DataAnnotations**
  - voorziet in attribuut klassen die toelaten meta-data te definiëren voor ASP.NET MVC en ASP.NET MVC data controls
  - voorbeeld: **DisplayAttribute** class

gebruik van het attribuut: de naam van de klasse zonder "Attribute"

instellen van properties van de klasse adhv benoemde parameters

```
[Display(Name = "Street", Prompt = "Street and house number")]
public string Street {
    get; set;
}
```

gebruik van het Display attribuut in de klasse BrewerEditViewModel




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

Edit.cshtml

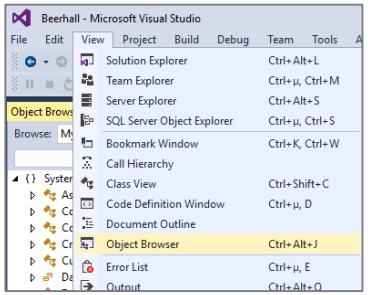
de tag helpers gebruiken de meta-data om de volgende HTML te genereren

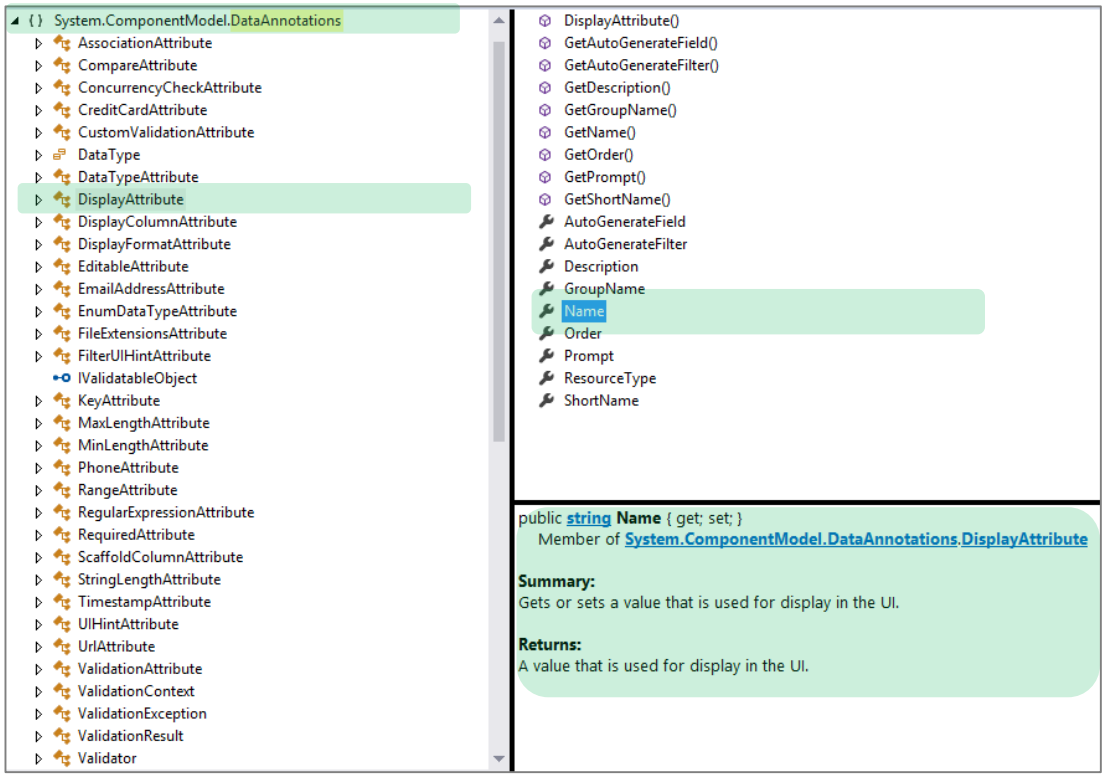
```
<div class="form-group">
    <label for="Street">Street</label>
    <input class="form-control" type="text" id="Street" name="Street" placeholder= "Street and house number" value=""
/>
</div>
```

broncode Brewer/Create

2. Display/Edit Annotaties

- **System.ComponentModel.DataAnnotations**
  - maak gebruik van de Object Browser om in detail te zien wat er voorzien is in klassen…



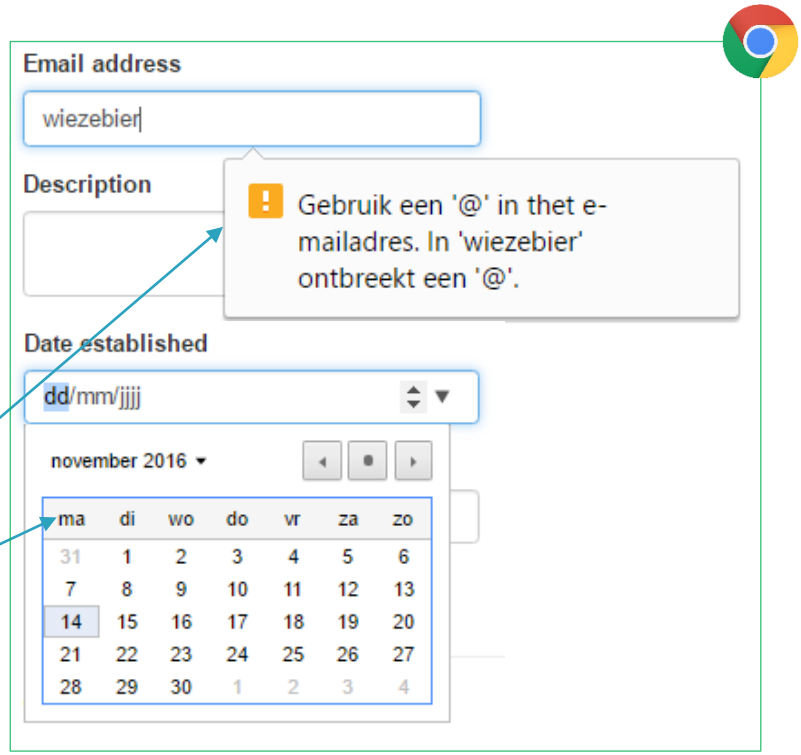


2. Display/Edit Annotaties

- **System.ComponentModel.DataAnnotations** vervolg
  - voorbeeld 2: **DataTypeAttribute**
    - laat toe een meer specifiek type te selecteren; vertelt iets over de semantiek van de property, dit wordt door browser gebruikt

```
[DataType(DataType.Currency)]
public int? Turnover {
            get; set;
}
[Display(Name = "Email address")]
[DataType(DataType.EmailAddress)]
public string ContactEmail {
            get; set;
}
[Display(Name = "Date established")]
[DataType(DataType.Date)]
public DateTime? DateEstablished {
            get; set;
}
```

gebruik van het DataType attribuut in de klasse BrewerEditViewModel



*browser maakt gebruik van de opgegeven types (bekijk het **type attribuut van de input elementen** in de HTML broncode…)*

2. Display/Edit Annotaties

- **System.ComponentModel.DataAnnotations** vervolg
  - voorbeeld 2: **DataTypeAttribute, de enum DataType**

Member name	Description
CreditCard	Represents a credit card number.
Currency	Represents a currency value.
Custom	Represents a custom data type.
Date	Represents a date value.
DateTime	Represents an instant in time, expressed as a date and time of day.
Duration	Represents a continuous time during which an object exists.
EmailAddress	Represents an e-mail address.
Html	Represents an HTML file.
ImageUrl	Represents a URL to an image.
MultilineText	Represents multi-line text.
Password	Represent a password value.
PhoneNumber	Represents a phone number value.
PostalCode	Represents a postal code.
Text	Represents text that is displayed.
Time	Represents a time value.
Upload	Represents file upload data type.
Url	Represents a URL value.

2. Display/Edit Annotaties

- Extra
  - enkele annotations vinden we in andere namespaces
    - voorbeeld: **HiddenInputAttribute**

```
public class BrewerEditViewModel {
        [HiddenInput]
        public int BrewerId {
            get; set;
        }
…
```

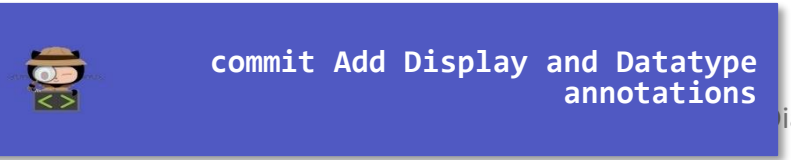
public sealed class **HiddenInputAttribute** : System.Attribute
Member of Microsoft.AspNetCore.Mvc

Summary:
Indicates associated property or all properties of associated type should be edited using an <input> element of type "hidden".

  - je kan meertaligheid in je applicatie inbouwen
    - zie https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization

2. Display/Edit Annotaties

- **Merk op: DataType** zorgt **niet voor client- of server side validatie,** maar
  - zorgt dat de browser HTML5 features kan gebruiken
    - bv. calendar control, currency symbol
  - zorgt dat de browser per default de data zal renderen in een correct formaat gebaseerd op je locale
  - zorgt dat MVC het juiste field template kan kiezen
- Indien we **robuuste validatie willen inbouwen in onze applicatie** zijn er ook annotaties die we hiervoor kunnen gebruiken…
  - MVC genereert dan de nodige **JavaScript** code voor de controle aan de **client side**,
  - bovendien gebeurt de controle ook door de **Model Binders** aan de **server side**. Dit heeft als voordeel dat de validatie ook werkt als de gebruiker javascript disabled in de browser



3. Validatie

- We hebben reeds data validatie in onze applicatie
  - Domein (via code)
  - Database (null, not null, constraints)
- Deze validatie wordt pas aan de server side uitgevoerd…
- In web applicaties is het aangewezen de validatie **ook op de client** te doen, zodat een **round-trip naar de server** niet nodig is

3. Validatie

- System.ComponentModel.DataAnnotations
  - enkele handige validatie attributen

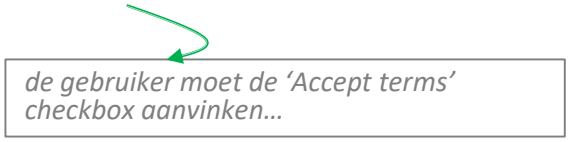
Attribute	Example	Description
`Compare`	`[Compare ("OtherProperty")]`	This attribute ensures that properties must have the same value, which is useful when you ask the user to provide the same information twice, such as an e-mail address or a password.
`Range`	`[Range(10, 20)]`	This attribute ensures that a numeric value (or any property type that implements `IComparable`) does not lie beyond the specified minimum and maximum values. To specify a boundary on only one side, use a `MinValue` or `MaxValue` constant—for example, `[Range(int. MinValue, 50)]`.
`RegularExpression`	`[RegularExpression ("pattern")]`	This attribute ensures that a string value matches the specified regular expression pattern. Note that the pattern has to match the *entire* user-supplied value, not just a substring within it. By default, it matches case sensitively, but you can make it case insensitive by applying the `(?i)` modifier—that is, `[RegularExpression("(?i)mypattern")]`.
`Required`	`[Required]`	This attribute ensures that the value is not empty or a string consisting only of spaces. If you want to treat whitespace as valid, use `[Required(AllowEmptyStrings = true)]`.
`StringLength`	`[StringLength(10)]`	This attribute ensures that a string value is not longer than a specified maximum length. You can also specify a minimum length: `[StringLength(10, MinimumLength=2)]`.

3. Validatie

- **[Required]**
  - geeft aan dat een veld verplicht in te vullen is
  - er wordt een foutmelding getoond wanneer het veld niet is ingevuld, we kunnen zelf bepalen welke foutmelding getoond wordt…
  - `[Required]`
    - default foutmelding: “The <prop-name> field is required”
  - `[Required(ErrorMessage=“Dit veld is verplicht”)]`
    - foutmelding zoals in de string
  - `[Required(ErrorMessage=“{0} is verplicht”)] :`
    - foutmelding zoals in de string met {0} vervangen door de DisplayName van de property.
  - `[Required(AllowEmptyStrings = false)]`
    - al dan niet toelaten van lege strings

3. Validatie

- **[Range]**
  - geeft aan dat een property een waarde moet aannemen die in een specifiek interval ligt
  - het bereik is inclusief de opgegeven grenzen
  - zonder type specificatie werkt Range op int en op double
    - `[Range(0, 20)]`
    - `[Range(0.00, 49.99)]`
  - voor andere types moet je expliciet het type opgeven, de grenzen geef je dan als strings mee
    - `[Range(typeof(decimal), "0.00", "49.99"]`
    - `[Range(typeof(bool), "true", "true", ErrorMessage = "You must accept the terms")]`



3. Validatie

- **[StringLength]**
  - bij deze annotatie kan je ook een minimum lengte vermelden
    - [StringLength(160, MinimumLength=10)]
- **[Compare]**
  - 2 properties die dezelfde waarde moeten hebben.
  - voorbeeld: property ConfirmationPassword moet dezelfde waarde bevatten als de property Password
    - [Compare(“Password”, ErrorMessage=“Password and confirmation password must match”)]
- In Microsoft.AspNetCore.Mvc namespace: **[Remote]**
  - uitvoeren van client side validatie met een server callback
  - voorbeeld: controleren of een opgegeven e-mail uniek is via een action method IsUniqueEmail in AccountController

```
[Remote(“IsUniqueEmail", "Account")]
public string Email { get; set; }
```

```
public async Task<JsonResult> IsUniqueEmail(string email) {
    var result = await _userManager.FindByEmailAsync(email);
    return Json(result != null);
}
```

3. Validatie - ViewModel

- Annotaties voor validatie in **BrewerEditViewModel**

```
public class BrewerEditViewModel {
        …
        [Required]
        [StringLength(50, ErrorMessage = "{0} may not contain more than 50 characters")]
        public string Name {
            get; set;
        }
        public string Street {
            get; set;
        }
        …
        [DataType(DataType.Currency)]
        [Range(0, int.MaxValue, ErrorMessage = "{0} may not be a negative value.")]
        public int? Turnover {
            get; set;
        }
        …
        [Display(Name = "Email address")]
        [DataType(DataType.EmailAddress)]
        [RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}", ErrorMessage = "Email address is not valid")]
        public string ContactEmail {
            get; set;
        }
```

3. Validatie – Client side

- Om de validatie aan de **client side** te enablen moeten we de gepaste jQuery libraries toevoegen aan de view
  - **jquery.js**
    - wordt reeds toegevoegd via _Layout.cshtml
  - **jquery.validate.js**
    - de jQuery Validation library
    - zie http://docs.jquery.com/Plugins/Validation
  - **jquery.validate.unobstrusive.js**
    - adapter library voor omzetten van MVC meta data naar jquery validate

```
@section scripts {
    <script asp-src-include="lib/jquery-validation/dist/jquery.validate.js"></script>
    <script asp-src-include="lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>
}
```

toegevoegd in Edit.cshtml

deze static javascript content werd bij de creatie van de applicatie automatisch toegevoegd in de wwwroot-folder

3. Validatie – Client side

- Aanpassen van **de view**
  - gebruik **validation tag helpers** om aan te geven waar foutmeldingen getoond moeten worden
    - **validation message** tag helper: asp-validation-for
      - toont foutmelding voor 1 bepaalde property

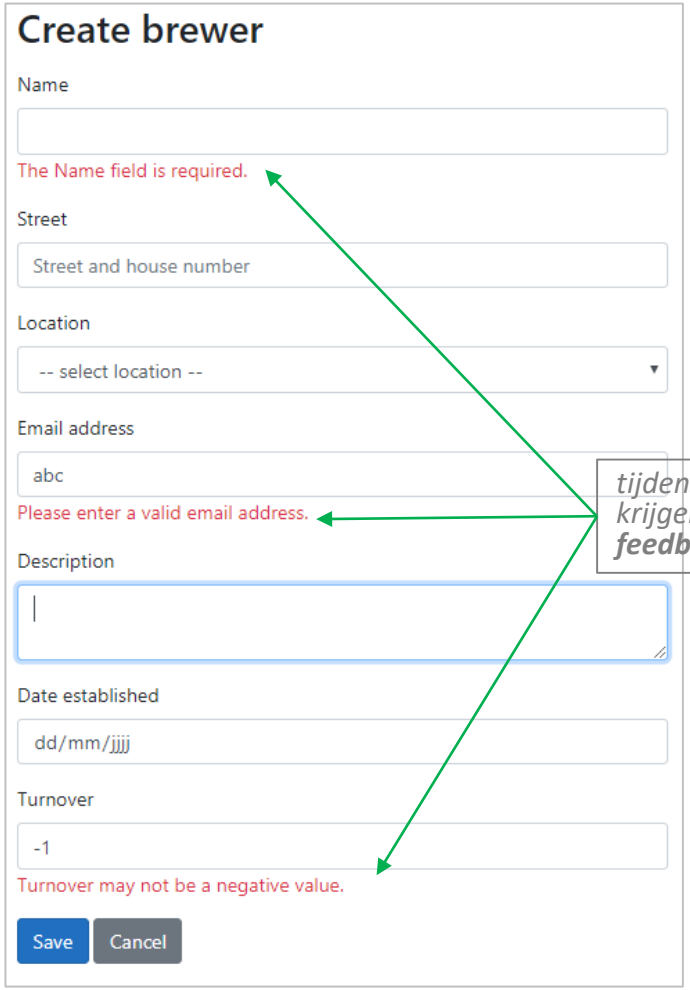
```
<div class="form-group">
    <label asp-for="ContactEmail"></label>
    <input asp-for="ContactEmail" class="form-control" />
    <span asp-validation-for="ContactEmail" class="text-danger"></span>
</div>
```

    - **validation summary** tag helper: asp-validation-summary
      - toont alle foutmeldingen

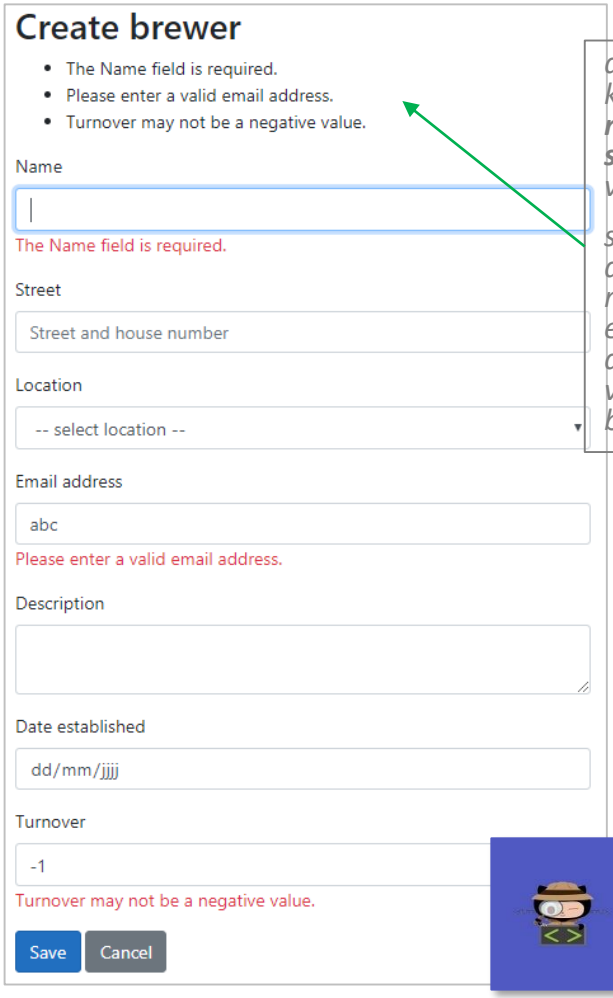
```
<form asp-action="@action" method="post">
    <div asp-validation-summary="All"></div>
    <input type="hidden" asp-for="BrewerId" />
    ...
```

3. Validatie – Client side

- Resultaat van de client side validatie…



*tijdens het editeren krijgen we **directe feedback…***



*als we op de Save knop klikken wordt **er niet gepost naar de server**; de summary verschijnt;*

straks zullen we zien dat deze summary meer dan alleen maar een opsomming van de 'property validation errors' kan bevatten

Focus on Client Side validation

- Bekijk de gegenereerde HTML code.
  - paginabron weergeven of via developer tools
- Elke input tag bevat attributen startend met **data-**. Deze attributen zijn een feature in HTML5, maar volledig backward compatible met alle moderne browsers (including IE6).

```
    <div class="form-group">
        <label for="Name">Name</label>
        <input class="form-control" type="text" data-val="true" data-val-length="Name may not contain more than 50 characters" data-val-length-max="50"
data-val-required="The Name field is required." id="Name" name="Name" value="" />
        <span class="text-danger field-validation-valid" data-valmsg-for="Name" data-valmsg-replace="true"></span>
    </div>
```

  - data-val = true: geeft aan dat deze input tag validatie nodig heeft, en deze wordt dan uitgevoerd door jquery validate

Focus on Client Side validation

- De attributen

`data-val`	indicates that the field contains validation data and should be processed by the unobtrusive adapters script.
`data-val-{validator name}`	e.g. data-val-length - contains the error message for the validator.
`data-val-{validator name}-{argument name}`	e.g. data-val-length-min - zero or more arguments necessary for performing validation.

The data- prefix is defined by the HTML5 standard. The specification states 'Custom data attributes are intended to store custom data private to the page or application, for which there are no more appropriate attributes or elements'. In the past, people tended to use hacks such as embedding data in css classes to add such data, but thankfully this is no longer necessary.

  - deze attributen worden door jquery.validate.unobstrusive vertaald naar jquery validate.

Focus on Client Side validation

- De attributen

```
[Range(0, int.MaxValue, ErrorMessage = "{0} may not be a negative value")]
    public int? Turnover {
        get; set;
}
```

De annotaties worden vertaald naar data-val-xxx attributen

```
<div class="form-group">
    <label for="Turnover">Turnover</label>
    <input class="form-control" type="number" data-val="true" data-val-range="Turnover may not be a negative value." data-val-range-max="2147483647"
data-val-range-min="0" id="Turnover" name="Turnover" value="" />
    <span class="text-danger field-validation-valid" data-valmsg-for="Turnover" data-valmsg-replace="true"></span>
</div>
```

Focus on Client Side validation

- ValidationSummary
  - creëert placeholder voor renderen van alle validatiefouten. Rendert alle fouten in een lijst <ul>

```
<div asp-validation-summary="All"></div>
```

```
<div class="validation-summary-valid" data-valmsg-summary="true">
  <ul>
    <li style="display:none"></li>
  </ul>
</div>
```

Na klik op de knop Save worden de fouten getoond…

```
<div class="validation-summary-errors" data-valmsg-summary="true">
  <ul>
    <li>The Name field is required.</li>
    <li>Email address is not valid</li>
    <li>Turnover may not be a negative value.</li>
  </ul>
</div>
```

Focus on Client Side validation

- Beheer van de client side libraries gebeurt via de **library manager**
  - zie https://docs.microsoft.com/en-us/aspnet/core/client-side/libman/libman-vs?view=aspnetcore-3.0

3. Validatie – Server side

- De annotaties aangebracht in het ViewModel zullen we ook gebruiken om **server-side validation** te doen
  - tijdens **model binding** gebeurt er validatie adhv de annotaties
  - in de **ModelState property** van de Controller klasse wordt informatie over de binding bijgehouden
    - deze property is van het type ModelStateDictionary

```
public class ModelStateDictionary
    Member of Microsoft.AspNetCore.Mvc.ModelBinding

Summary:
Represents the state of an attempt to bind values from an HTTP Request to an action method, which includes validation information.
```

    - via de property **IsValid** kunnen we te weten komen of er validatie fouten zijn

```
public bool IsValid { get; }
    Member of Microsoft.AspNetCore.Mvc.ModelBinding.ModelStateDictionary

Summary:
Gets a value that indicates whether any model state values in this model state dictionary is invalid or not validated.
```

3. Validatie – Server side

- De Controller
  - indien bij de HttpPost ModelState errors zijn gaan we het formulier opnieuw presenteren zodat de gebruiker fouten kan verbeteren

```
[HttpPost]
public IActionResult Create(EditViewModel brewerEditViewModel) {
    if (ModelState.IsValid) {
        try {
            Brewer brewer = new Brewer();
            MapBrewerEditViewModelToBrewer(brewerEditViewModel, brewer);
            _brewerRepository.Add(brewer);
            _brewerRepository.SaveChanges();
            TempData["message"] = $"You successfully added brewer {brewer.Name}.";
        }
        catch (Exception e) {
             TempData["error"] = $"Sorry, er liep iets fout, brouwer {brewer?.Name} kon niet worden gewijzigd";
        }
       return RedirectToAction(nameof(Index));
    }
     ViewData["IsEdit"] = false;
     ViewData["Locations"] = GetLocationsAsSelectList();
    return View(nameof(Edit), brewerEditViewModel);
}
```

Als je simpelweg javascript inhoud uitschakelt in de browser zie je hoe belangrijk deze validatie is… Maak gebruik van een breakpoint en bekijk de ModelState…

Als de validatie niet lukt toon dan de Create view opnieuw. De SelectList moet ook opnieuw worden doorgegeven.

De data die reeds werd ingevuld in het formulier zal opnieuw getoond worden

3. Validatie – Server side

- De Modelstate kan ons nog meer interessante informatie aanreiken
  - voor elke property die de model binder probeert te binden vind je in de dictionary een entry
    - key = naam property, value informatie over de binding.

```
[Required]
[StringLength(50, ErrorMessage = "{0} may not contain more than 50 characters")]
public string Name {
    get; set;
}
```

Geen client side validatie, javascript disabled...

Name is required...

Results View	Expanding the Results View will enumerate the IEnumerable
[0]	{[Name, Microsoft.AspNetCore.Mvc.ModelBinding.ModelStateDictionary+ModelStateNode]}
Key	"Name"
Value	SubKey={Name}, Key="Name", ValidationState=Invalid
AttemptedValue	""
ChildNodes	null
Children	null
Errors	Count = 1
[0]	{Microsoft.AspNetCore.Mvc.ModelBinding.ModelError}
ErrorMessage	"The Name field is required."
Exception	null
Raw View	
IsContainerNode	false
Key	"Name"
RawValue	""
SubKey	{Name}
ValidationState	Invalid
Non-Public members	
Non-Public members	

HoGent

3. Validatie – Server side

- Model binding en de ModelState
  - tijdens de model binding worden **data annotations** gecontroleerd en wanneer een waarde niet voldoet wordt dit geregistreerd in de ModelState
  - tijdens model binding kunnen zich nog **andere problemen** voordoen
    - bv. de string “appel” wordt doorgegeven voor een DateTime property
    - de model binder gaat geen exceptions werpen maar dit registreren in de ModelState
      - zoals je kon zien op vorige slide wordt de “attempted value” steeds bijgehouden

3. Validatie – Server side

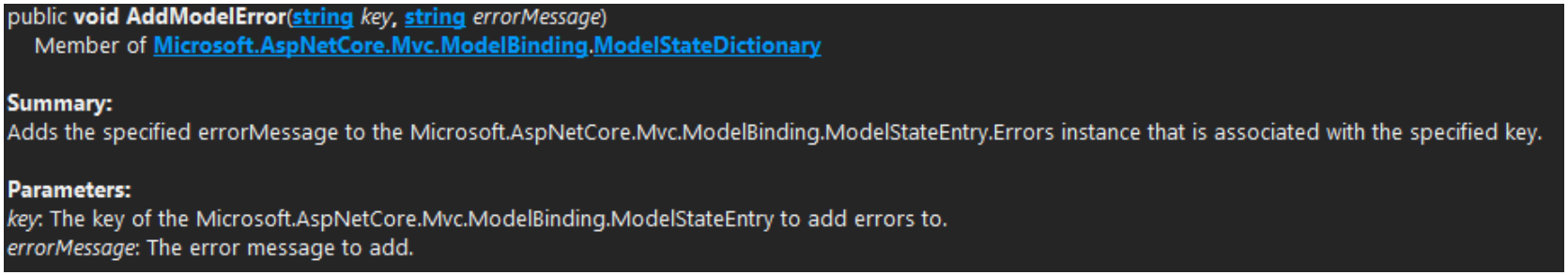
- De View en de ModelState
  - de tag helpers ValidationMessageFor en ValidationSummary geven de fouten uit de ModelState weer.
    - het resultaat is identiek aan de client side validatie

```
▼<div class="validation-summary-errors" data-valmsg-summary="true">
  ▼<ul>
      <li>The Name field is required.</li>
    </ul>
  </div>
```

```
<span class="text-danger field-validation-error" data-valmsg-for="Name" data-valmsg-replace="true">The
Name field is required.</span>
```

3. Validatie – Server side

- De ModelState
  - we kunnen ook zelf errors toevoegen aan de ModelState



  - de key kan de naam van een property bevatten
    - dit is een error op **property niveau**, en zal ook zo opgepikt worden door de taghelper validation-message-for
  - wanneer we de key leeg laten dan voegen we een error toe op **model niveau**
    - dit is een error die niet gelinkt is aan een specifieke property
    - deze error wordt enkel getoond in de validation summary

3. Validatie – Server side

- In de view kunnen we aangeven welke errors in de validation summary moeten worden getoond

```
<div asp-validation-summary="All"></div>
```

én property errors én model errors

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

enkel model errors

```
<div asp-validation-summary="None"></div>
```

geen errors

3. Validatie – Server side

- voorbeeld: ipv TempData te gebruiken wanneer er exceptions in het domein geworpen worden kunnen we een model error toevoegen aan de modelstate en het formulier opnieuw aanbieden

```
[HttpPost]
public IActionResult Create(BrewerEditViewModel brewerEditViewModel) {
    if (ModelState.IsValid) {
        try {
            Brewer brewer = new Brewer();
            MapBrewerEditViewModelToBrewer(brewerEditViewModel, brewer);
            _brewerRepository.Add(brewer);
            _brewerRepository.SaveChanges();
            TempData["message"] = $"You successfully added brewer {brewer.Name}.";
            return RedirectToAction(nameof(Index));
        }
        catch (Exception e) {
            ModelState.AddModelError("", e.Message);
        }
    }
     ViewData["IsEdit"] = false;
     ViewData["Locations"] = GetLocationsAsSelectList();
    return View(nameof(Edit), brewerEditViewModel);
}
```

in combinatie met een ModelOnly validation summary zullen we nu geen herhaling krijgen van property level errors in de summary

3. Validatie – Unit testen

- We kunnen dit ook gebruiken voor onze **unit testen**…
  - bij unit testen gebeurt er geen model binding
  - door zelf errors aan de ModelState toe te voegen kunnen we wel testen of de server side validatie correct verloopt
  - **voorbeeld**: simulatie van ModelState errors

```
[Fact]
public void Create_ModelStateErrors_DoesNotCreateNorPersistsBrewerAndPassesViewModelAndViewDataToEditView()
{
            _locationRepository.Setup(m => m.GetAll()).Returns(_dummyContext.Locations);
            _brewerRepository.Setup(m => m.GetBy(1)).Returns(_dummyContext.Bavik);
            BrewerEditViewModel brewerEvm = new BrewerEditViewModel(_dummyContext.Bavik);
            _controller.ModelState.AddModelError("", "Error message");
            var result = Assert.IsType<ViewResult>(_controller.Create(brewerEvm));
            Assert.Equal("Edit", result.ViewName);
            Assert.Equal(brewerEvm, result.Model);
            var locations = Assert.IsType<SelectList>(result.ViewData["Locations"]);
            Assert.Equal(3, locations.Count());
            var isEdit = Assert.IsType<bool>(result.ViewData["IsEdit"]);
            Assert.False(isEdit);
            _brewerRepository.Verify(m => m.Add(It.IsAny<Brewer>()), Times.Never());
            _brewerRepository.Verify(m => m.SaveChanges(), Times.Never());
        }
```

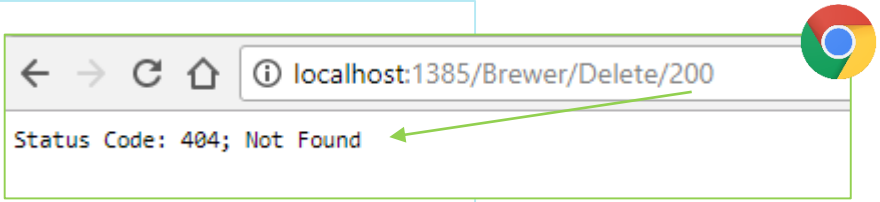
3. Validatie – HttpGet methodes

- Ook de HttpGet methodes voor Edit en Delete kunnen we robuuster maken en gepast reageren als een brewer niet gevonden wordt…
  - gebruik maken van Tempdata[“ErrorMessage”] en redirecten naar de Index, of
  - een NotFound() retourneren

```
public IActionResult Edit(int id) {
            Brewer brewer = _brewerRepository.GetBy(id);
            if (brewer == null)
                return NotFound();
            ViewData["Locations"] = GetLocationsAsSelectList(brewer.Location?.PostalCode);
            return View(new BrewerEditViewModel(brewer));
        }
```

  - dit retourneert een NotFoundResult
  - je kan de middleware configureren om gepast op de NotFound te reageren
    - simpelste vorm: UseStatusCodePages()

```
public void Configure(…) {
    …
    app.UseStaticFiles();
    app.UseStatusCodePages();
    app.UseSession();
    …
}
```



    - zie ook https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling#configuring-status-code-pages

3. Validatie – HttpGet methodes

- We kunnen ook unit testen indien de controller bij HttpGet een NotFoundResult retourneert indien een onbestaande brouwer wordt opgevraagd

```
[Fact]
        public void Delete_UnknownBrewer_ReturnsNotFound()
        {
            _brewerRepository.Setup(m => m.GetBy(1)).Returns((Brewer)null);
            IActionResult action = _controller.Delete(1);
            Assert.IsType<NotFoundResult>(action);
        }
```

Add server side validation and extend unit tests

3. Validatie - Oefening

- Oefening:
  - Server side validatie voor Edit
  - Extra unit testen voor Edit

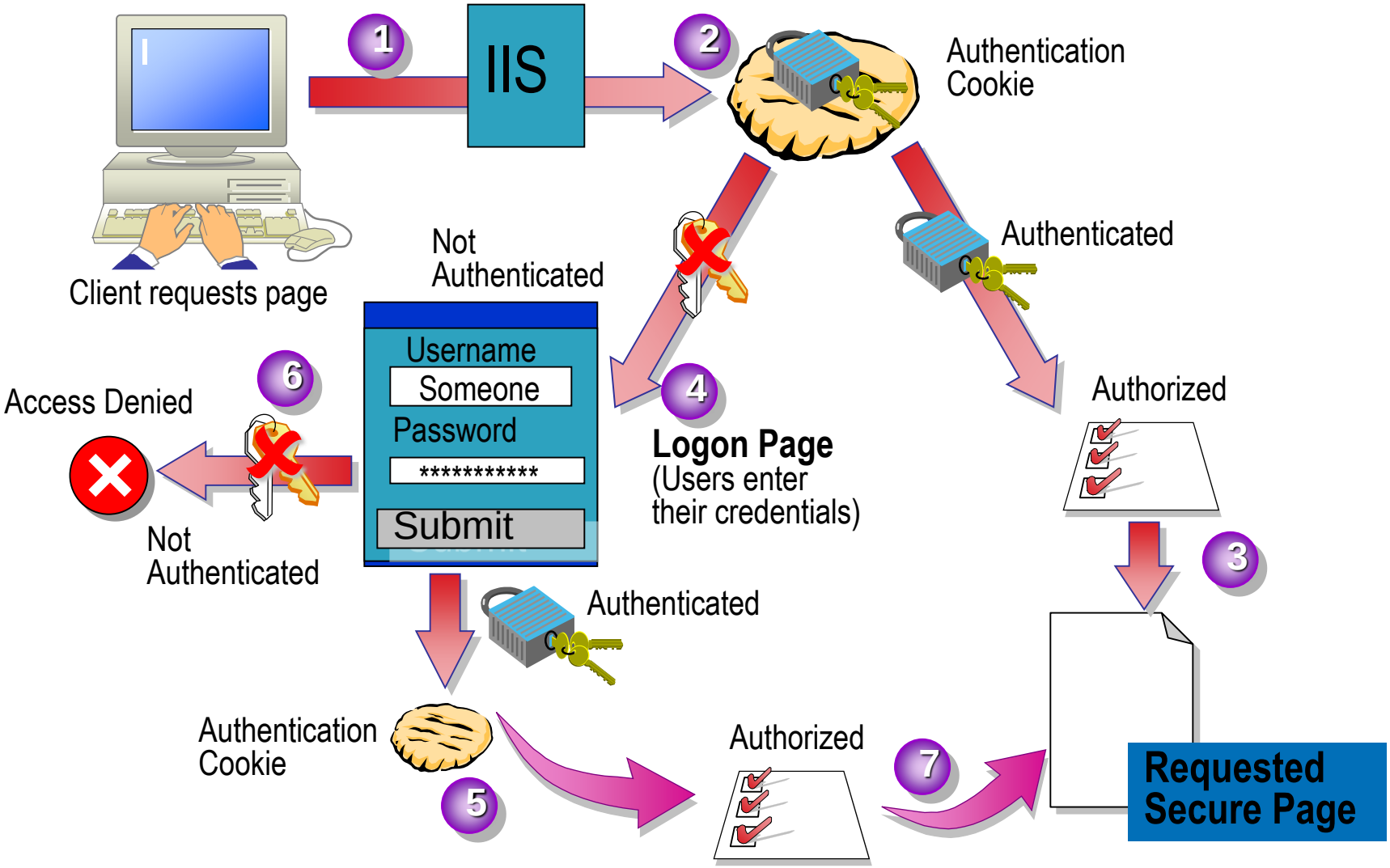
Authenticatie

ASP.NET CORE
Individual User Accounts

4. Authenticatie en Authorisatie

- Beveiliging?
  - **Authenticatie** (verificatie): wie is de gebruiker en heeft hij/zij toegang tot de applicatie? Gebruiker geeft identificatie in en deze wordt gevalideerd. Mogelijkheden binnen ASP.NET:
    - **Individual User Accounts:** op formulieren gebaseerde verificatie met cookieondersteuning. Gebruiker heeft identiteit in, in aanlog formulier, en na verificatie wordt een geëncrypteerd authenticatie cookie gegenereerd. Niet geverifieerde verzoeken worden automatisch omgeleid naar een aanlog formulier.
    - **OAuth en OpenID:** externe login via 3th party (Facebook, …)
    - **Organizational accounts:** single-sign-on voor bedienden en business partners via Active Directory, Azure Active Directory of Office 365
    - **Windows authenticatie:** voor Intranet. Single Sign on via Active Directory
  - **Authorisatie**: o.b.v. de identificatiegegevens van gebruiker wordt nagegaan wat een gebruiker mag doen binnen de applicatie. Kan worden toegekend per gebruiker of per rol.

4. Authenticatie: Individual User Accounts



4. Authenticatie: Individual User Accounts

- Welcome to **ASP.NET Core Identity**

ASP.NET Core Identity is a membership system which allows you to add login functionality to your application. Users can create an account and login with a user name and password or they can use an external login providers such as Facebook, Google, Microsoft Account, Twitter and more.

You can configure ASP.NET Core Identity to use a SQL Server database to store user names, passwords, and profile data. Alternatively, you can use your own persistent store to store data in another persistent storage, such as Azure Table Storage.

4. Authenticatie : Identity Framework

- Identity Framework
  - API met klassen en interfaces voor het beheren van gebruikers, rollen en claims voor een ASP.NET web applicatie en authenticeren van gebruikers.
    - Administratief beheer van User Accounts, Rollen, Claims
    - Support voor cookie based authenticatie, 2-Factor Authenticatie via email of SMS messaging, claim based authenticatie, role based authorisatie
    - Support voor Social log-ins
    - …
  - Namespace Microsoft.AspNetCore.Identity

  - We bekijken de klassen die een project met “Individual User Account authentication” bevat: de Models, Views, Controllers, en andere componenten nodig voor de basis authenticatie/authorisatie
  - https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?tabs=visual-studio%2Caspnetcore2x

4. Authenticatie: Identity Framework

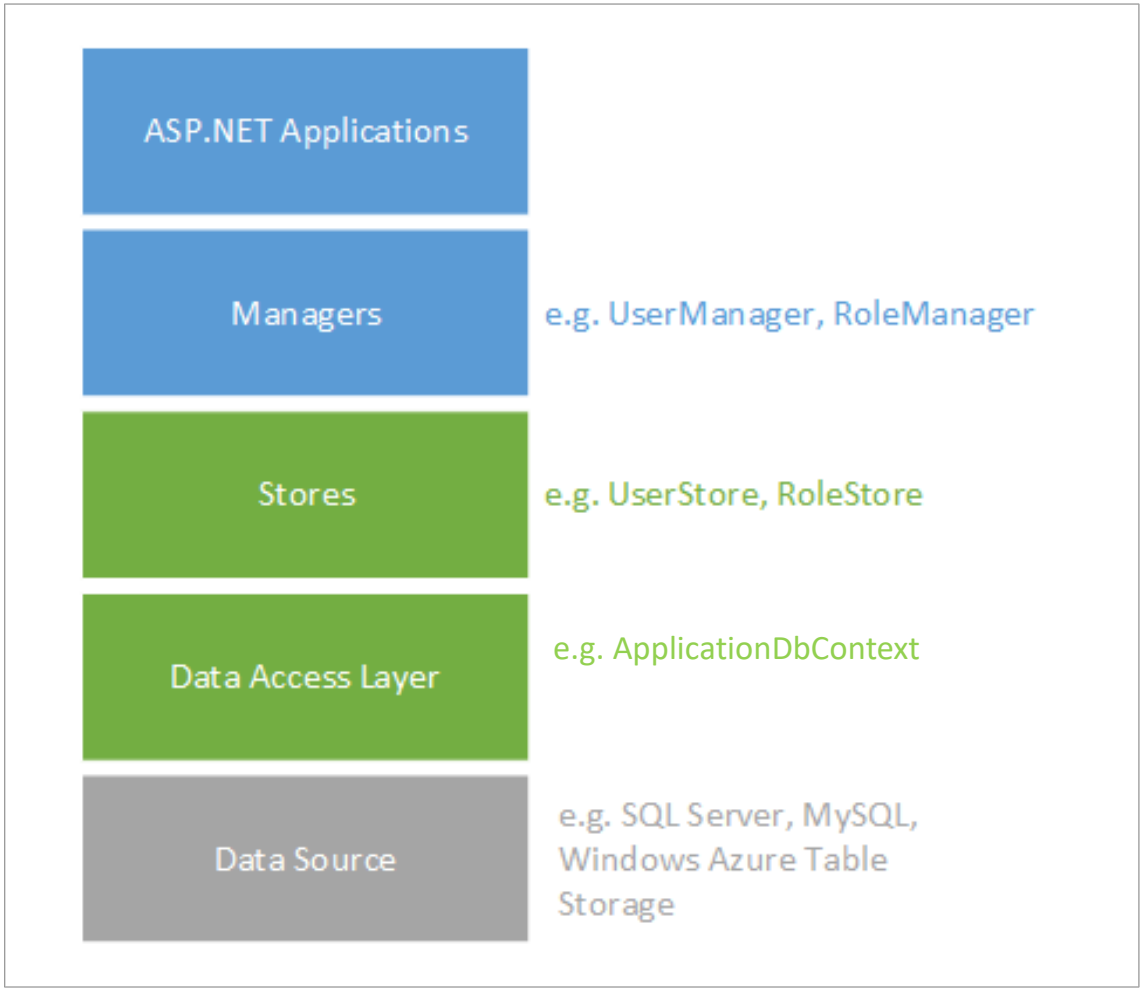
- Twee belangrijke abstracties worden gebruikt in Identity

Managers are high-level classes which an application developer uses to perform operations, such as creating a user, in the ASP.NET Identity system.

Stores are lower-level classes that specify how entities, such as users and roles, are persisted. Stores are closely coupled with the persistence mechanism, but managers are decoupled from stores which means you can replace the persistence mechanism without disrupting the entire application.

4. Authenticatie: Identity Framework

- De architectuur



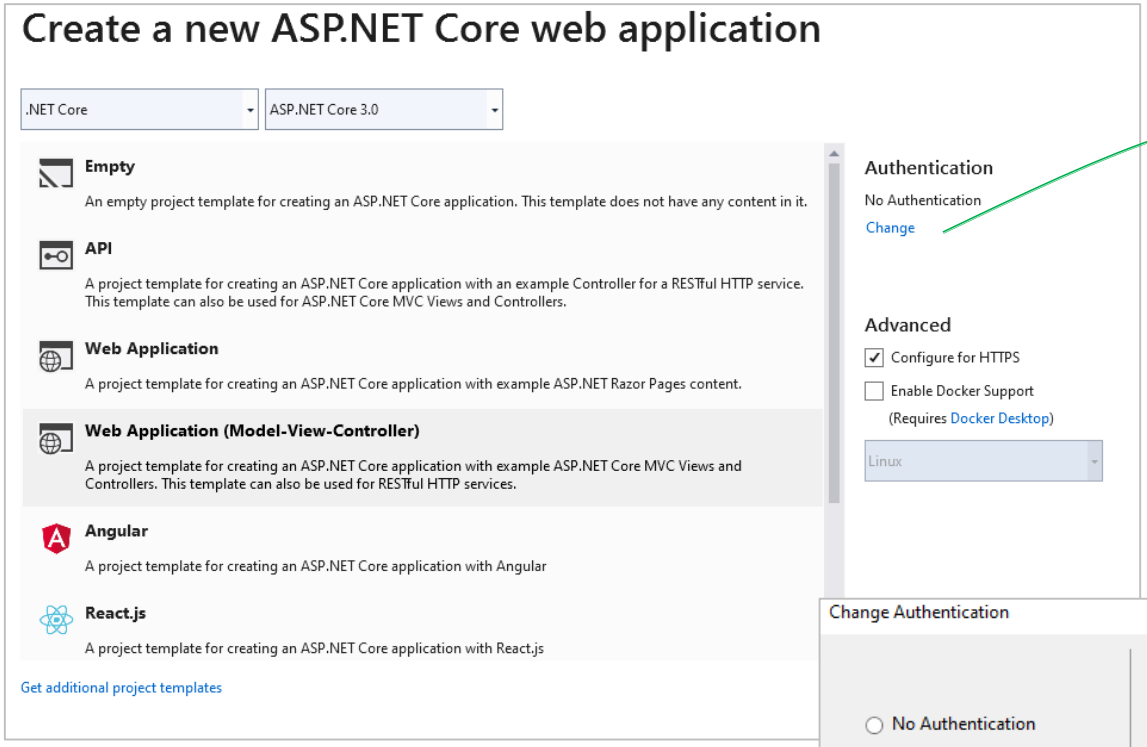
4. Authenticatie: Identity Framework

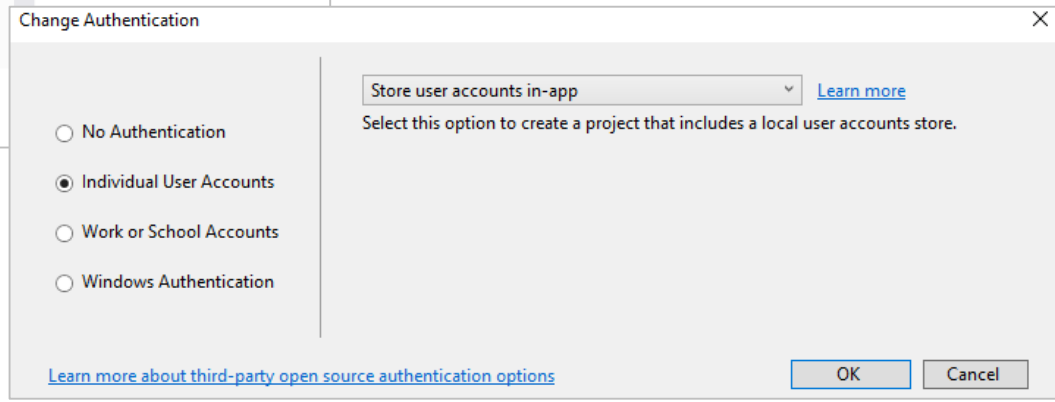
- De data die Identity Framework gebruikt bevat o.a.

Data	Description
Users	Registered users of your web site. Includes the user Id and user name. Might include a hashed password if users log in with credentials that are specific to your site (rather than using credentials from an external site like Facebook), and security stamp to indicate whether anything has changed in the user credentials. Might also include email address, phone number, whether two factor authentication is enabled, the current number of failed logins, and whether an account has been locked.
User Claims	A set of statements (or claims) about the user that represent the user's identity. Can enable greater expression of the user's identity than can be achieved through roles.
User Logins	Information about the external authentication provider (like Facebook) to use when logging in a user.
Roles	Authorization groups for your site. Includes the role Id and role name (like "Admin" or "Employee").

4. Authenticatie: IUA template

- Project met authenticatie – **Individual User Accounts**
  - instelling bij selectie template tijdens aanmaken van de applicatie



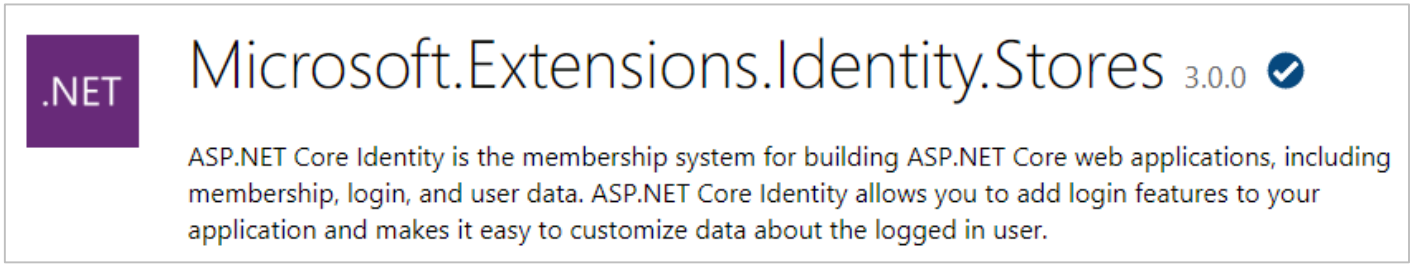


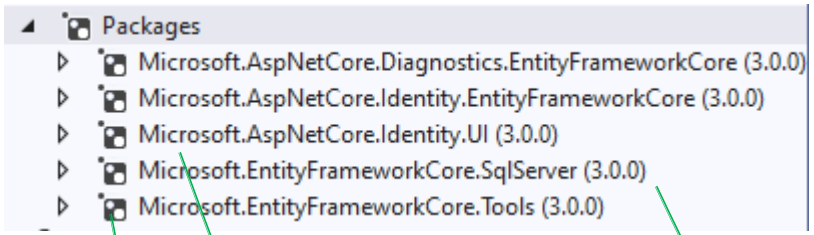
4. Authenticatie: IUA template

- Een blik op een verse MVC Web applicatie met Individual User Accounts authenticatie
  - de nuget package Microsoft.EntityFrameworkCore.Tools
    - laat toe om met migrations te werken

4. Authenticatie: IUA template

- Een blik op een verse Web applicatie met Individual User Accounts authenticatie
  - de nuget packages





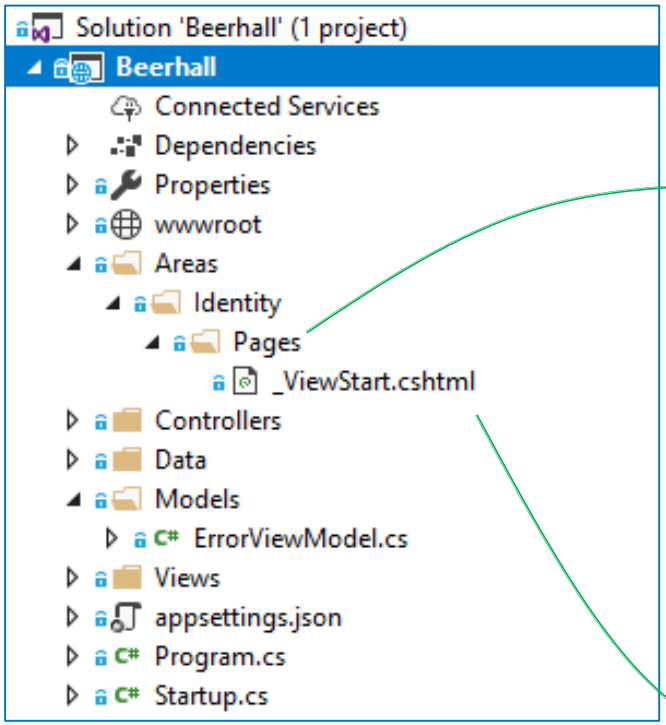
deze package zorgt ervoor dat we kunnen werken met Entity Framework zodat we gebruik zullen kunnen maken van een SQL Server DB om gegevens van gebruikers, rollen, … op te slaan.

dit is de defaut razor pages buiklt-in UI voor Identity

laat toe om met DB migrations te werken

4. Authenticatie: IUA template

- Een blik op een verse Web applicatie met IUA authenticatie



Identity maakt gebruik van ***razor pages***

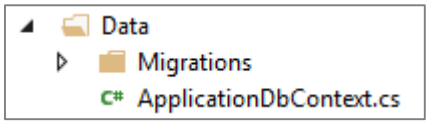
slechts 1 bestand!?

hier wordt aangegeven welk Layout page gebruikt wordt voor Identity,

al de rest zit in de nuget package ***Identity.UI***

4. Authenticatie: Identity

- Een blik op een MVC Web applicatie met Individual User Accounts authenticatie
  - enkele gewijzigde klassen



Er is een ApplicationDbContext aanwezig en een eerste migratie die het aanmaken van de DB tabellen voor Identity bevat…

4. Authenticatie: configuratie Identity

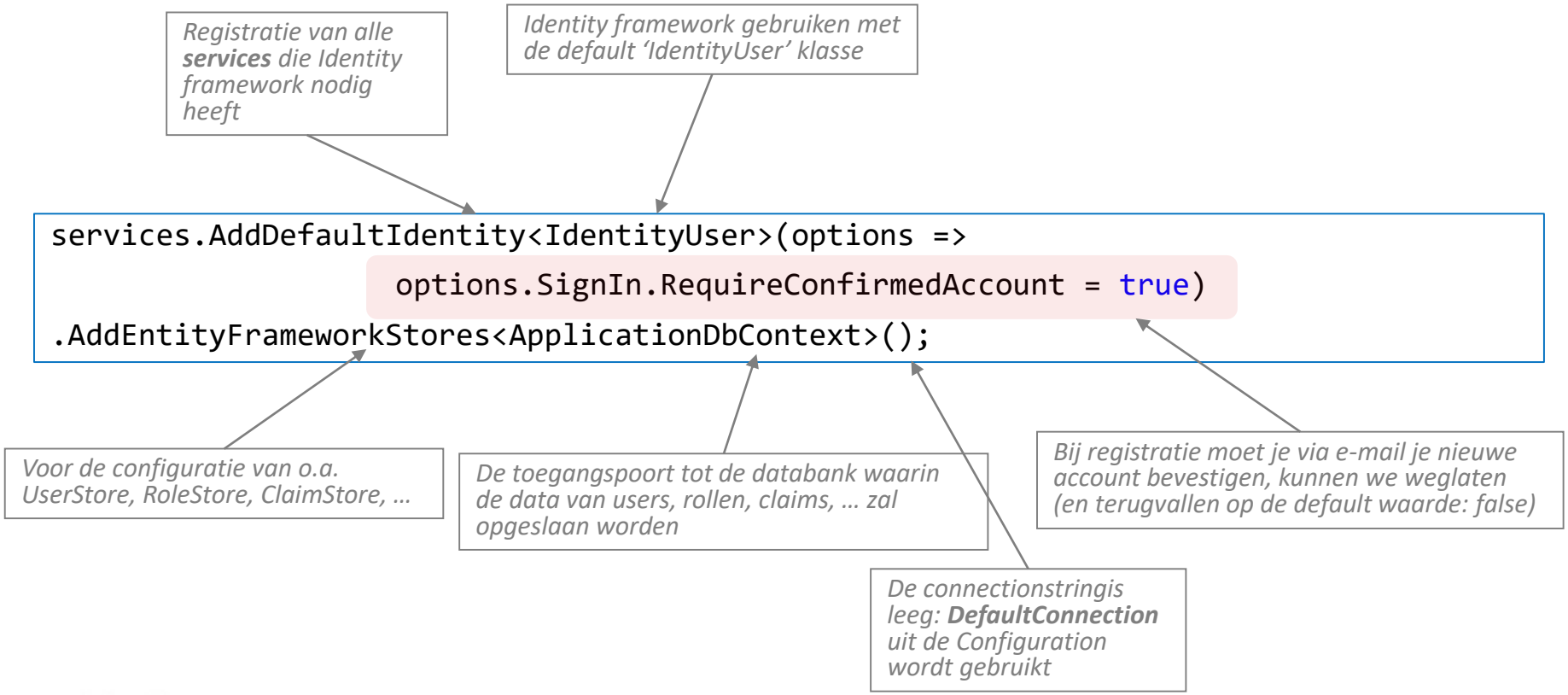
- Een blik op een MVC Web applicatie met Individual User Accounts authenticatie
  - de **ApplicationDbContext** erft van **IdentityDbContext**
  - het type user voor Identity is impliciet **IdentityUser**

```
namespace Beerhall.Data
{
    public class ApplicationDbContext : IdentityDbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
    }
}
```

  - straks zullen we onze mapping aan deze klasse toevoegen…

4. Authenticatie: configuratie Identity

- Een blik op een MVC Web applicatie met IUA authenticatie – StartUp.cs
  - Configuratie van IdentityFramework in **ConfigureServices(…)**



4. Authenticatie: configuratie Identity

- Een blik op een MVC Web applicatie met IUA authenticatie – StartUp.cs
  - Configuratie van IdentityFramework in **Configure(…)**

```
app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

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

Authenticatie & Authorization checks gebeuren in de MVC controllers, het verwerken van cookies en detectie van 401 errors moeten gebeuren vooraleer de request in het MVC framework zitten

***Merk op**: het is belangerijk dat UseRouting, UseAuthentication, UseAuthorization en UseEndpoints in deze volgore in de pipeline voorkomen!*

4. Authenticatie: Identity

- Een blik op een MVC Web applicatie met Individual User Accounts authenticatie
  - de klasse **IdentityUser**
    - de default implementatie van **IdentityUser<TKey>** die een string gebruikt als primaire sleutel

public class **IdentityUser** : Microsoft.AspNetCore.Identity.IdentityUser<string>
Member of Microsoft.AspNetCore.Identity

Summary:
The default implementation of Microsoft.AspNetCore.Identity.IdentityUser`1 which uses a string as a primary key.

per default is de primary key voor een user van het type string, desgewenst kan je dit type veranderen in een type die je zelf kiest…

  - IdentityUser : IdentityUser<String>

IdentityUser erft volgende properties:

*Per default zal het **email** adres van een gebruiker ook als UserName gebruikt worden*

*De primary key: **Id***

*Essentiele properties: een unieke **UserName** en een hashed version van het **paswoord***

- AccessFailedCount
- Claims
- ConcurrencyStamp
- Email
- EmailConfirmed
- Id
- LockoutEnabled
- LockoutEnd
- Logins
- NormalizedEmail
- NormalizedUserName
- PasswordHash
- PhoneNumber
- PhoneNumberConfirmed
- Roles
- SecurityStamp
- TwoFactorEnabled
- UserName

Identity Framework kan bijvoorbeeld bijhouden hoeveel keer een user een niet geslaagde inlogpoging doet, en de account desgwenst een tijd afsluiten

4. Authenticatie: configuratie Identity

- Merk op: je kan desgewenst gebruik maken van een eigen gedefinieerde user-klasse. Je laat deze erven van IdenityUser en kan deze dan uitbreiden met extra properties.
  - voorbeeld

```
public class ApplicationUser : IdentityUser {
        public string Name { get; set; }
        public string FirstName { get; set; }
        public string Street { get; set; }
        public Location Location { get; set; }
    }
```

  - je moet nu in de configuratie wel aangeven dat je met jouw type user wil werken en eveneens je ApplicationDbContext laten erven van IdentityDbContext<ApplicationUser>

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
```

```
services.AddDefaultIdentity<ApplicationUser>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

4. Authenticatie: configuratie Identity

- Een blik op een MVC Web applicatie met Individual User Accounts authenticatie
  - De connectionstring DefaultConnection in **appsettings.json**

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-Beerhall-03BE1F6C-C59E-4046-AA32-69C14CCCF6CA;Trusted_Connection=True;MultipleActiveResultSets=true"
},
```

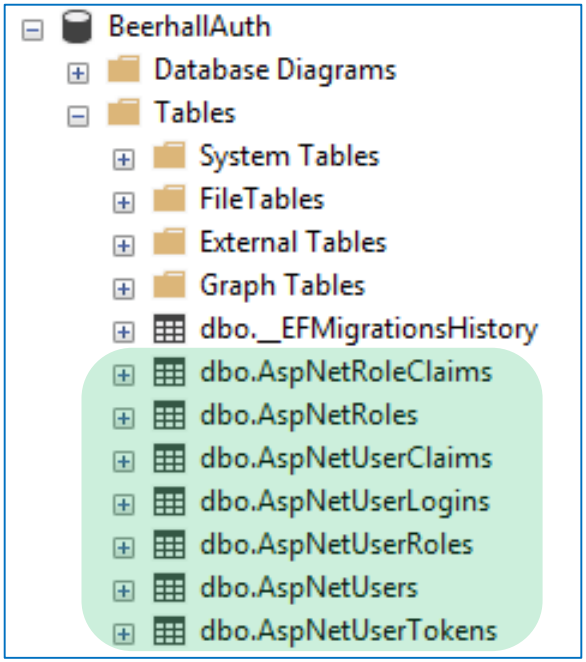
Per default wordt localdbserver gebruikt, we kunnen dit nu reeds aanpassen, ook de naam van de DB passen we aan…

```
"ConnectionStrings": {
  "DefaultConnection": "Server=.;Database=Beerhall;Trusted_Connection=True;MultipleActiveResultSets=true"
},
```

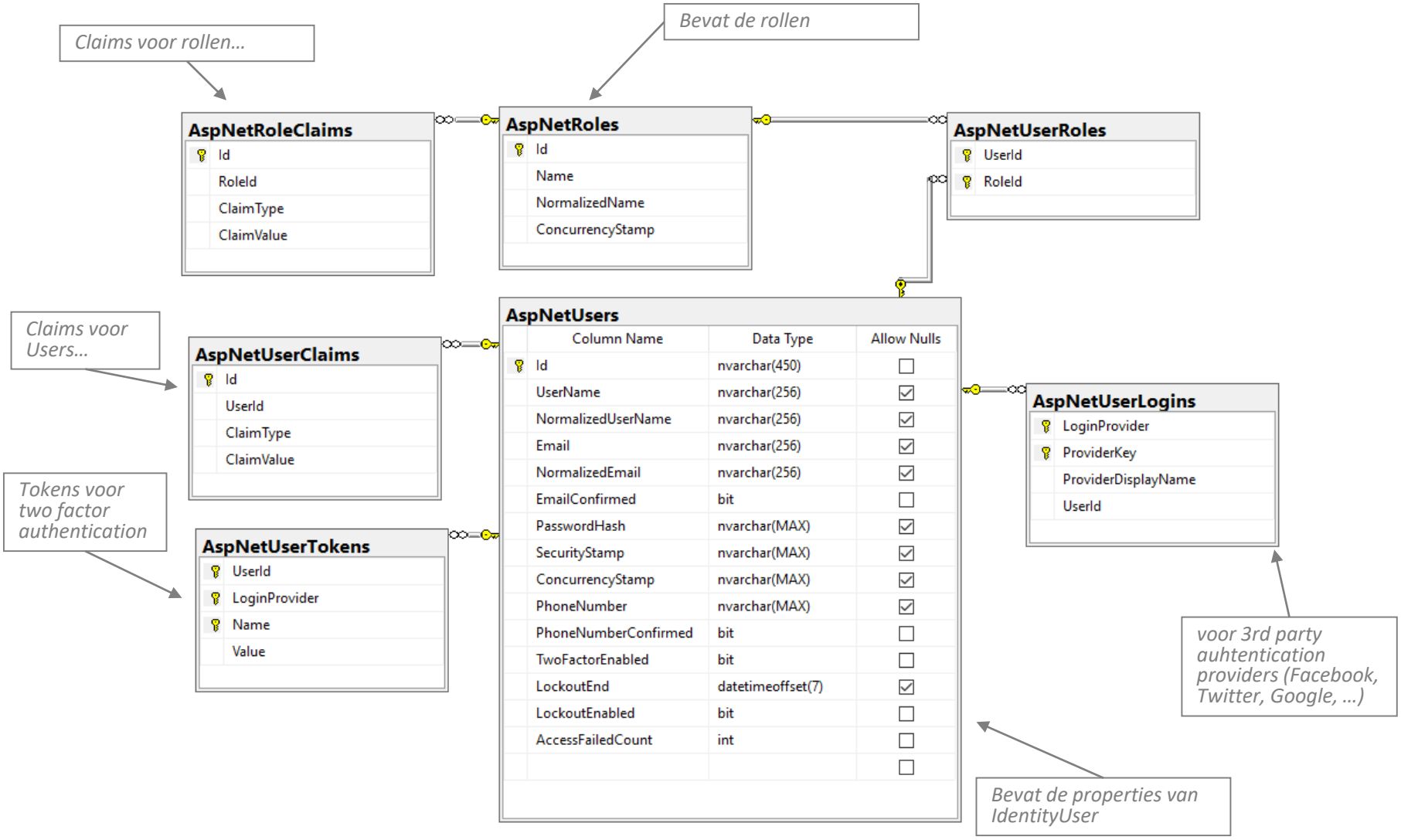
LocalDB is created specifically for developers. It is very easy to install and requires no management, yet it offers the same T-SQL language, programming surface and client-side providers as the regular SQL Server Express. In effect the developers that target SQL Server no longer have to install and manage a full instance of SQL Server Express on their laptops and other development machines. Moreover, if the simplicity (and limitations) of LocalDB fit the needs of the target application environment, developers can continue using it in production, as LocalDB makes a pretty good embedded database too.

4. Authenticatie: de databank

- Als we het **update-database** commando geven kunnen we de Identity tabellen terugvinden in de databank
  - merk op: daar wij gebruik maken van drop-create strategie gaan we verder in deze cursus geen gebruik maken van migrations

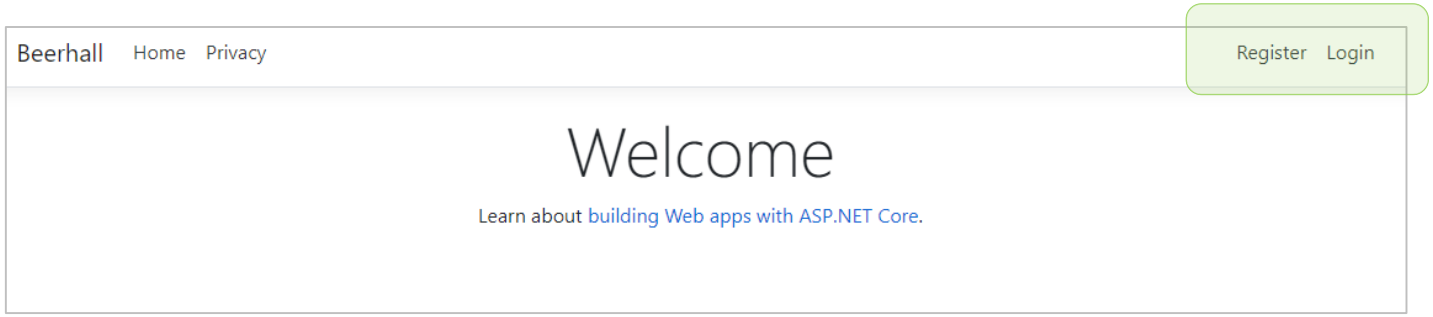


4. Authenticatie: de databank



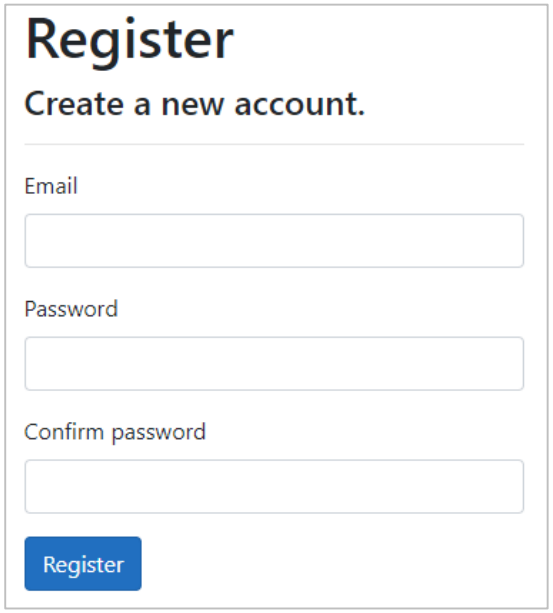
4. Authenticatie: Register

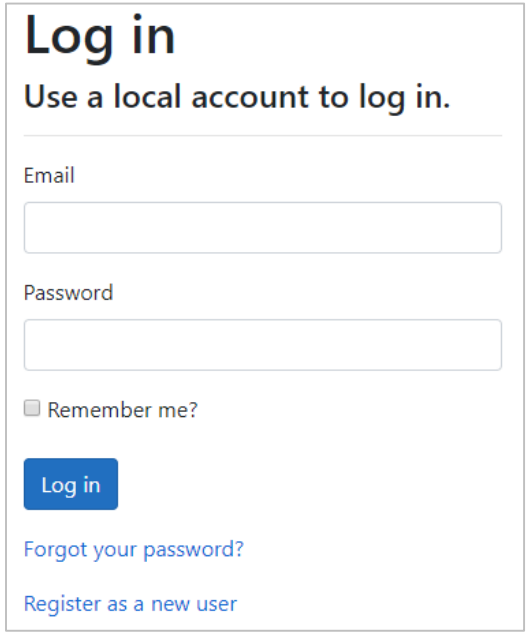
- Als je de applicatie runt kan je kennismaken met de authenticatie…



4. Authenticatie: Register

- Als je de applicatie runt kan je kennismaken met de authenticatie en het default gedrag van Identity…



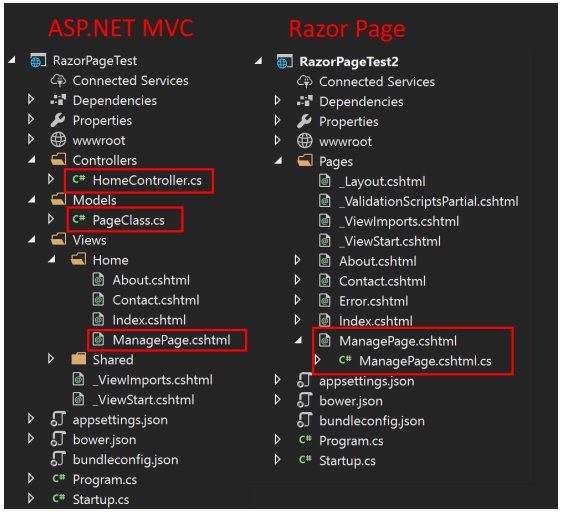


4. Authenticatie: Register

- Indien we de default implementatie willen **bekijken en/of wijzigen** zullen we moeten gebruik maken van **scaffolding** om aan de onderliggende code te kunnen
- Identity framework is gebaseerd op **Razor pages**, dit is een nieuw onderdeel van ASP.NET Core, geïntroduceerd in ASP.NET Core 2.0

Focus on Razor Pages

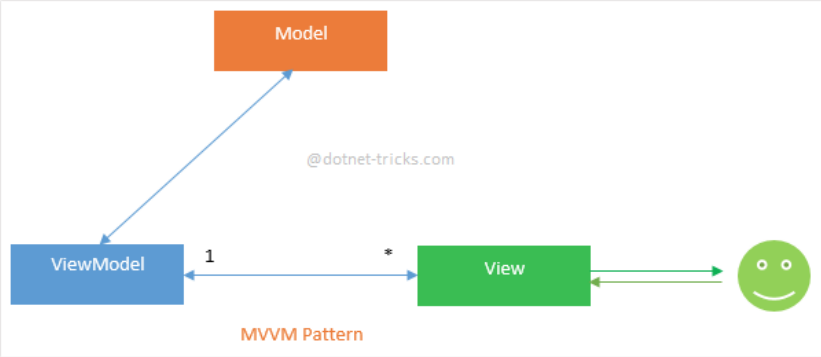
- **Razor Pages** is a new feature of ASP.NET Core that makes coding page-focused scenarios easier and more productive.
  - **MVVM**-framework Model – View – ViewModel
    - bij MVC zitten het Model en de Controller actions niet bij de View zelf, de code zit verspreid
    - een razor page bevat bij de View ook het Model en de Controller actions, alle verantwoordelijkheden van de pagina zitten samen



Focus on Razor Pages

MVVM pattern

MVVM stands for Model-View-View Model. This pattern supports two-way data binding between view and View model. This enables automatic propagation of changes, within the state of view model to the View. Typically, the view model uses the observer pattern to notify changes in the view model to model.



01. Model

The Model represents a set of classes that describes the business logic and data. It also defines business rules for data means how the data can be changed and manipulated.

02. View

The View represents the UI components like CSS, jQuery, html etc. It is only responsible for displaying the data that is received from the controller as the result. This also transforms the model(s) into UI..

03. View Model

The View Model is responsible for exposing methods, commands, and other properties that helps to maintain the state of the view, manipulate the model as the result of actions on the view, and trigger events in the view itself.

Focus on Razor Pages

- In een notendop:
  - alle razor pages worden in een folder **Pages** geplaatst
    - de plaats van de razor page in de folder Pages of een subfolder hiervan bepaalt de overeenkomstige URL die leidt naar deze pagina
      - per default zoekt de runtime naar razor pages in deze map
      - Index is de default pagina wanneer de url geen pagina bevat
    - voorbeeld

File name and path	matching URL
/Pages/Index.cshtml	`/` or `/Index`
/Pages/Contact.cshtml	`/Contact`
/Pages/Store/Contact.cshtml	`/Store/Contact`
/Pages/Store/Index.cshtml	`/Store` or `/Store/Index`

Focus on Razor Pages

- In een notendop:
  - de View
    - .cshtml bestand die op de eerste lijn de directive **@page** bevat
      - via de @page geef je aan dat dit een razor-page is
      - een razor page handelt requests direct af, de requests passeren dus geen controller
    - gebruik **@model** om een model te specifiëren die je kan gebruiken in de razor page
      - het model wordt geïmplementeerd in een .cshtml.cs bestand

Focus on Razor Pages

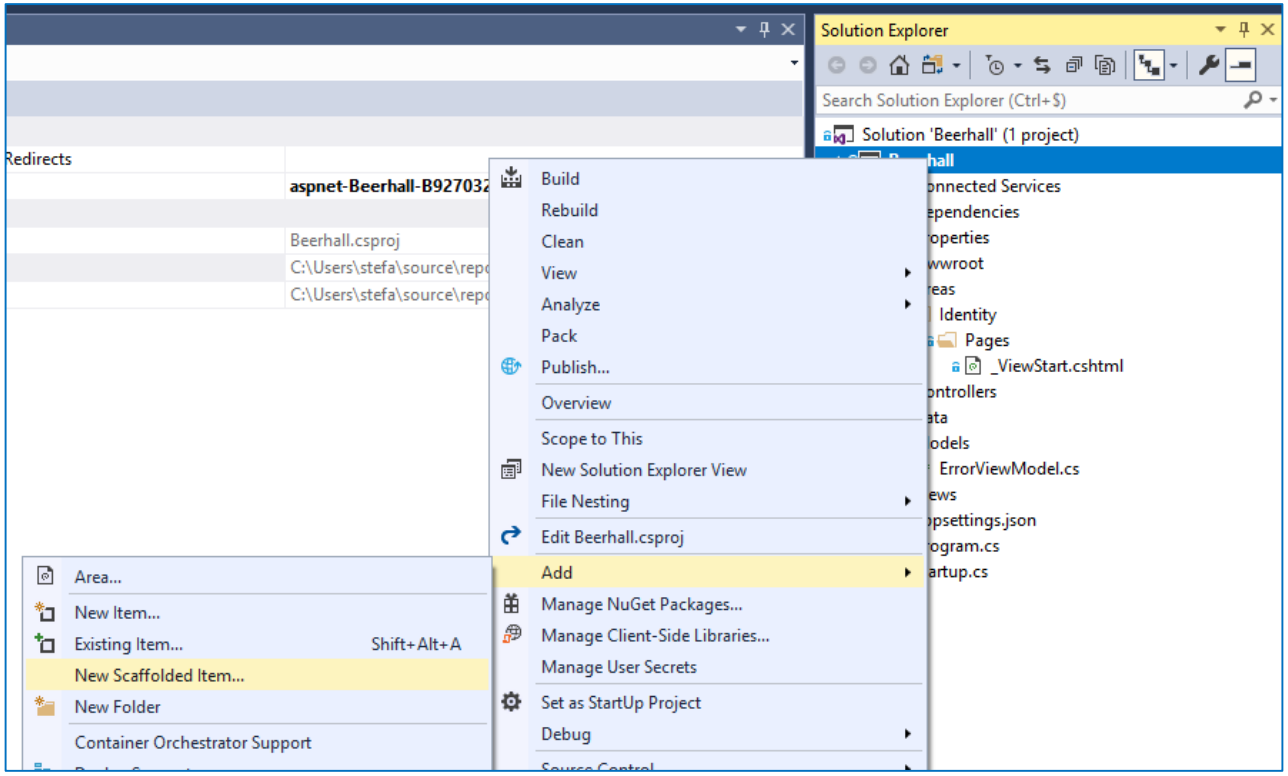
- In een notendop:
  - het Model
    - .cshtml.cs bestand
    - het bestand noemt **<PageName>Model** en leeft in dezelfde namespace als de View
    - via deze klasse scheidt je de logica van de presentatie
    - bevat
      - page handlers voor requests die naar de pagina worden gestuurd
      - data nodig om de pagina te renderen
    - gebruik DI om dependencies te injecteren en deze klasse unit testbaar te maken

Focus on Razor Pages

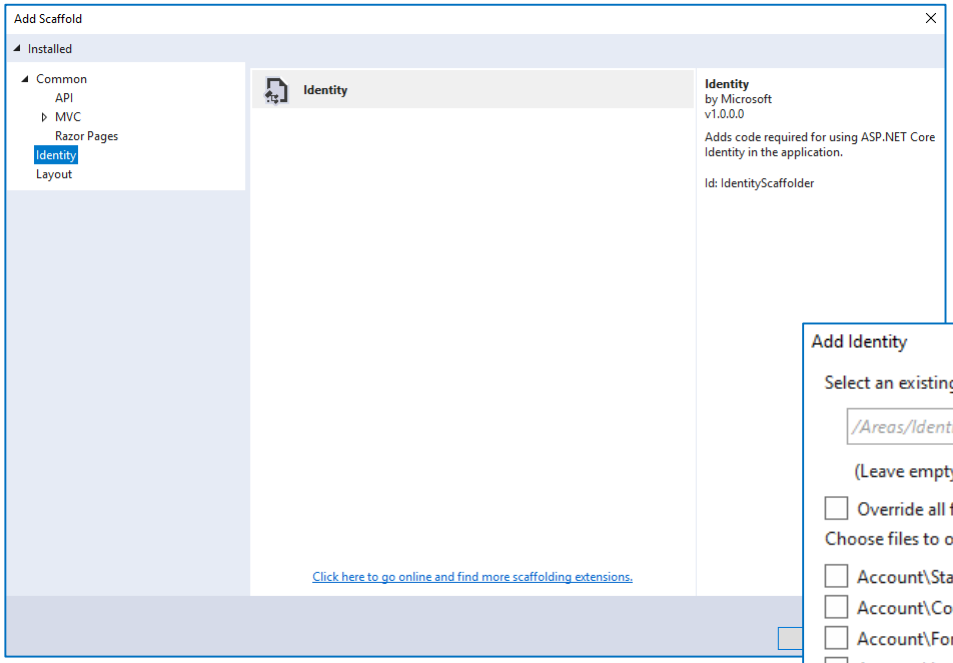
- In een notendop:
  - het Model (vervolg)
    - je definieert in deze klasse de handler methods
    - typische handlers:
      - **OnGet** – initializeer de toestand om de pagina te kunnen presenteren
      - **OnPost** – afhandeling van form submits
    - maak gebruik van de optionele suffix Async voor assynchrone functies
    - maak gebruik van client/server side validatie attributen in deze klasse

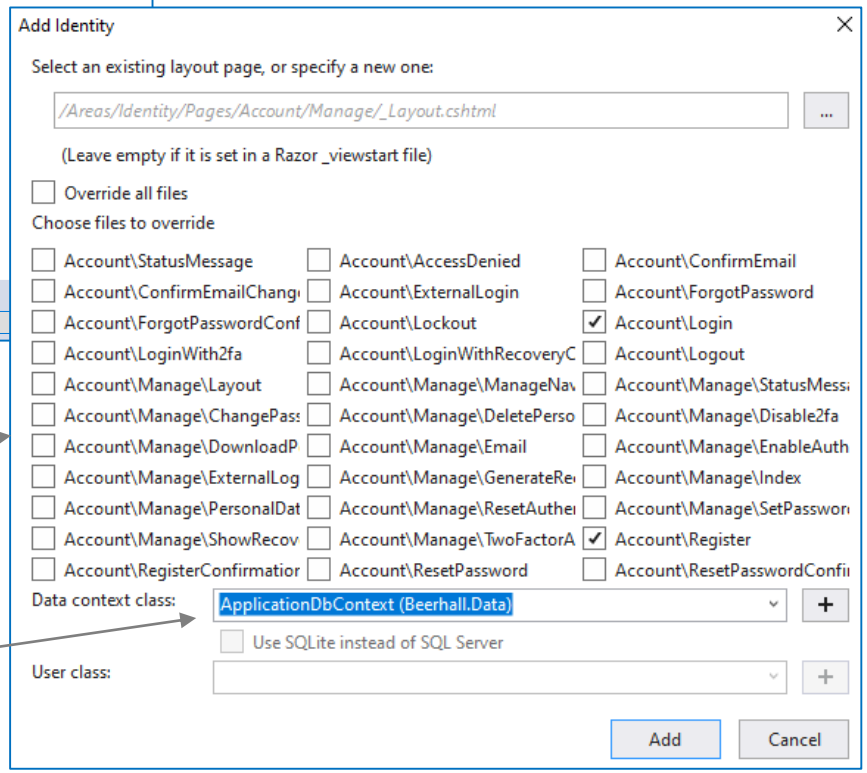
4. Authenticatie: Register

- voorbeeld: implementatie van **Register**
  - laat toe dat een nieuwe gebruiker zich registreert
  - indien we de code voor register willen bekijken/aanpassen moeten we gebruik maken van **scaffolding**



4. Authenticatie: Register

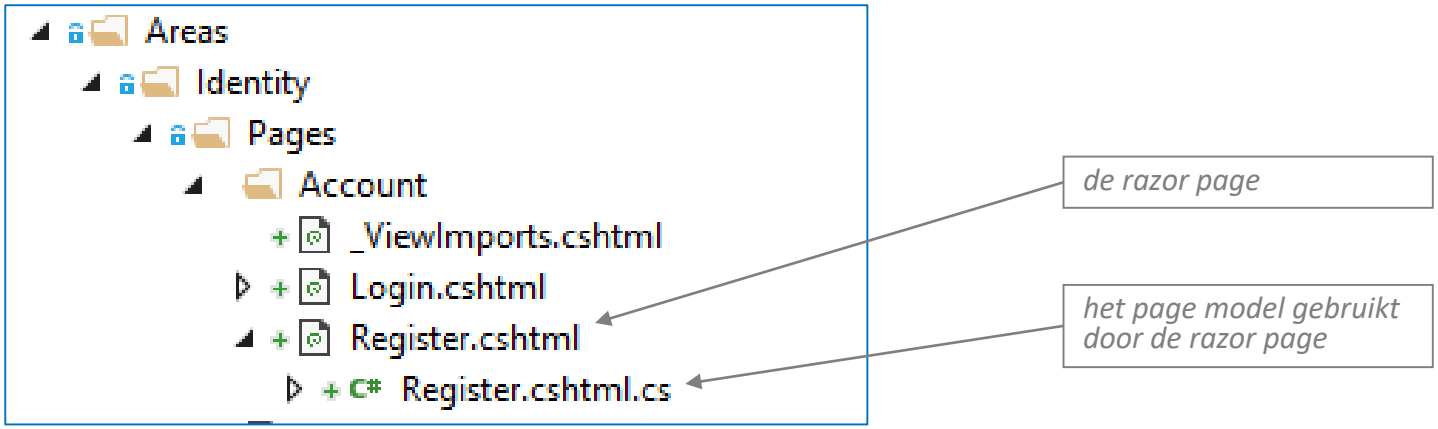




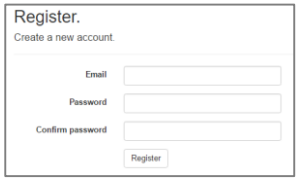
Selecteer hier de onderdelen die je wenst te wijzigen

Hier geef je aan welke de DbContext klasse is die gebruikt wordt

4. Authenticatie: Register



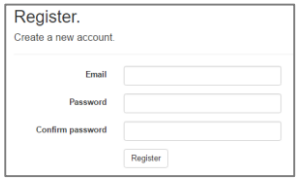
4. Authenticatie: Register



- Registratie van een nieuwe user: **Register**
  - Via constructor injectie worden een aantal essentiële services, Managers, van Identity Framework beschikbaar in het page model **Register.cshtml.cs**

```
public class RegisterModel : PageModel
    {
        private readonly SignInManager<IdentityUser> _signInManager;
        private readonly UserManager<IdentityUser> _userManager;
        private readonly ILogger<RegisterModel> _logger;
        private readonly IEmailSender _emailSender;

        public RegisterModel(
            UserManager<IdentityUser> userManager,
            SignInManager<IdentityUser> signInManager,
            ILogger<RegisterModel> logger,
            IEmailSender emailSender)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _logger = logger;
            _emailSender = emailSender;
        }
```

4. Authenticatie: Register



- Registratie van een nieuwe user: **Register HttpGet**
  - Register.cshtml.cs - OnGet

in MVC komt dit overeen met een HttpGet action method Register in een controller genaamd Account

```
public async Task OnGetAsync(string returnUrl = null)
        {
            ReturnUrl = returnUrl;
            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
        }
```

de returnUrl wordt aangeleverd via model binding; na een succesvolle registratie zal je terug gestuurd worden naar de pagina vanwaar de registratie werd aangeroepen

  - Register.cshtml.cs - InputModel

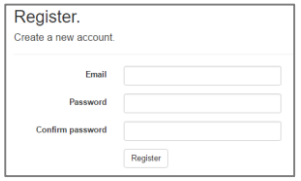
deze inner klasse bevat het viewmodel met de validatie attributen

```
public class InputModel
        {
            [Required]
            [EmailAddress]
            [Display(Name = "Email")]
            public string Email { get; set; }

            [Required]
            [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters
long.", MinimumLength = 6)]
            [DataType(DataType.Password)]
            [Display(Name = "Password")]
            public string Password { get; set; }

            [DataType(DataType.Password)]
            [Display(Name = "Confirm password")]
            [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
            public string ConfirmPassword { get; set; }
        }
```

via het [BindProperty] attribuut wordt aangegeven dat de publieke property Input moet worden gebruikt tijdens model binding

```
[BindProperty]
public InputModel Input { get; set; }
```

4. Authenticatie: Register



- Registratie van een nieuwe user: **Register HttpGet**
  - Register.cshtml

```
@page
@model RegisterModel
@{
    ViewData["Title"] = "Register";
}
<h2>@ViewData["Title"]</h2>
<div class="row">
    <div class="col-md-4">
        <form asp-route-returnUrl="@Model.ReturnUrl" method="post">
            <h4>Create a new account.</h4>
            <hr />
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Input.Email"></label>
                <input asp-for="Input.Email" class="form-control" />
                <span asp-validation-for="Input.Email" class="text-danger"></span>
            </div>
            <div class="form-group">
            … some code ommitted here …
            <button type="submit" class="btn btn-default">Register</button>
        </form>
    </div>
</div>
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

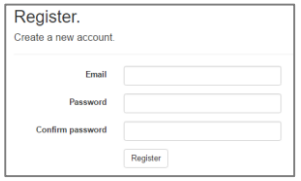
dit is een razor page

de page maakt gebruik van RegisterModel

via model binding zal de ingevoerde waarde gekoppeld worden aan Input.Email die deel uitmaakt van RegisterModel

*invoegen van **JQuery validation** via een partial view op een async manier (uitleg later)*

4. Authenticatie: Register



- Registratie van een nieuwe user: **Register - HttpPost**
  - Register.cshtml.cs - OnPostAsync

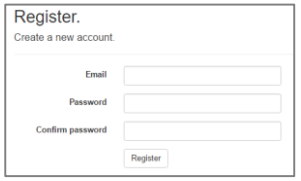
```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl = returnUrl ?? Url.Content("~/");
            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid)
            {
                var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
                var result = await _userManager.CreateAsync(user, Input.Password);
                if (result.Succeeded)
                {
                    _logger.LogInformation("User created a new account with password.");
                   // some code omitted here
                    {
                        await _signInManager.SignInAsync(user, isPersistent: false);
                        return LocalRedirect(returnUrl);
                    }
                }
                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
            }

            // If we got this far,
            return Page();
        }
```

Bij een geldige ModelState wordt eerst een IdentityUser object aangemaakt, de nodige properties krijgen hun waarde via de Input-property die via model binding een invulling kreeg

Merk op:

- *de property UserName krijgt de waarde van e-mail*
- *het paswoord van een identityUser kunnen we niet instellen, het wordt enkel in gehashte vorm bijgehouden*

4. Authenticatie: Register



- Registratie van een nieuwe user: **Register - HttpPost**
  - Register.cshtml.cs - OnPostAsync

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl = returnUrl ?? Url.Content("~/");
            if (ModelState.IsValid) {
                var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
                var result = await _userManager.CreateAsync(user, Input.Password);
                //
                ret
            }
```

*De **UserManager** wordt gebruikt om een volwaardige Idenity user te creëren. De user heeft gegarandeerd een unieke UserName (~e-mail adres) en is met het gehashte paswoord opgeslagen in de DB in de tabel AspNetUsers. Via de returnwaarde result kunnen we achterhalen of deze operatie al dan niet succesvol was.*

OnPostAsync is een asynchrone methode! Asynchrone methodes worden verderop toegelicht.

Belangrijk:

- *maak steeds gebruik van **await** bij een aanroep naar een asynchrone methode*
- *wanneer je await gebruikt in een methode dan moet je de methode **async** declareren en moet die methode een **Task<ReturnType>** retourneren (de runtime weet hoe het hiermee verder kan werken…)*

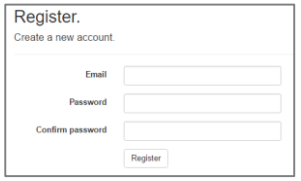
public virtual System.Threading.Tasks.Task<Microsoft.AspNetCore.Identity.IdentityResult>
CreateAsync(**TUser** *user*, string *password*)
Member of Microsoft.AspNetCore.Identity.UserManager<TUser>

Summary:
Creates the specified user in the backing store with given password, as an asynchronous operation.

Parameters:
user: The user to create.
password: The password for the user to hash and store.

Returns:
The System.Threading.Tasks.Task that represents the asynchronous operation, containing the Microsoft.AspNetCore.Identity.IdentityResult of the operation.

4. Authenticatie: Register



- Registratie van een nieuwe user: **Register - HttpPost**
  - Register.cshtml.cs - OnPostAsync

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl = returnUrl ?? Url.Content("~/");
            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid)
            {
                var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
                var result = await _userManager.CreateAsync(user, Input.Password);
                if (result.Succeeded)
                {
                    _logger.LogInformation("User created a new account with password.");
                    // some code omitted here
                    {
                        await _signInManager.SignInAsync(user, isPersistent: false);
                        return LocalRedirect(returnUrl);
                    foreach (var error in result.Errors)
                    {
                                                                    error.Description);
```

*Bij een succesvolle CreateAsync operatie wordt de **SignInManager** gebruikt om de gecreëerde gebruiker in te loggen en wordt er geredirect naar de returnUrl. Dit is een **LocalRedirect** die de applicatie beschermt voor Open Redirect Attacks*

public virtual System.Threading.Tasks.Task **SignInAsync**(**TUser** *user*, bool *isPersistent*, [string *authenticationMethod* = null])
Member of Microsoft.AspNetCore.Identity.SignInManager<TUser>

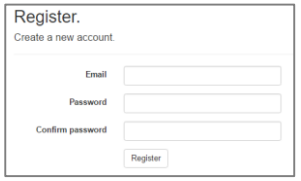
Summary:
Signs in the specified user.

Parameters:
user: The user to sign-in.
isPersistent: Flag indicating whether the sign-in cookie should persist after the browser is closed.
authenticationMethod: Name of the method used to authenticate the user.

Returns:
The task object representing the asynchronous operation.

De cookie wordt niet gepersisteerd, en dus verdwijnt wanneer we de browser sluiten

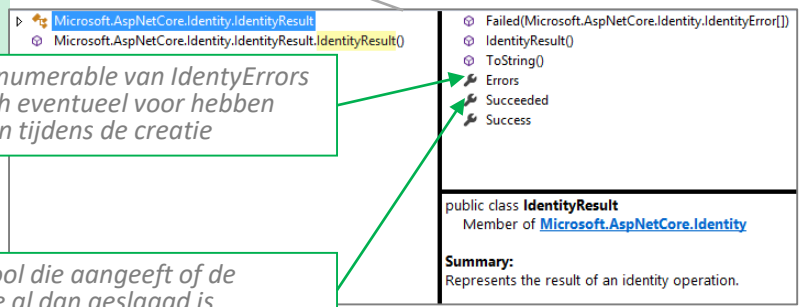
4. Authenticatie: Register



- Registratie van een nieuwe user: **Register - HttpPost**
  - Register.cshtml.cs - OnPostAsync

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl = returnUrl ?? Url.Content("~/");
            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid)
            {
                var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
                var result = await _userManager.CreateAsync(user, Input.Password);
                if (result.Succeeded)
                {
                    _logger.LogInformation("User created a new account with password.");
                    // some code omitted here
                    {
                        await _signInManager.SignInAsync(user, isPersistent: false);
                        return LocalRedirect(returnUrl);
                    }
                }
                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError(string.Empty, error.Descri
                }
            }

            // If we got this far, something failed, redisplay form
            return Page();
        }
```

deze errors zullen getoond worden in de Validation summary

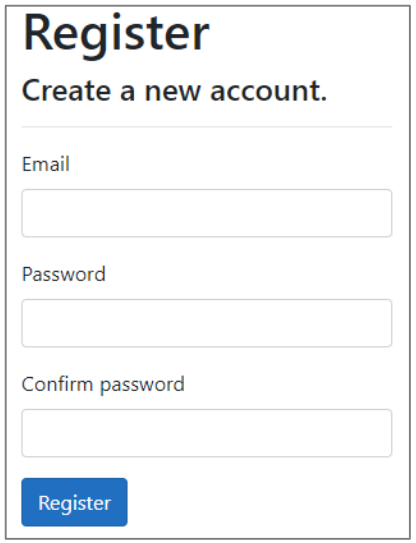


een IEnumerable van IdentyErrors die zich eventueel voor hebben gedaan tijdens de creatie

een bool die aangeeft of de creatie al dan geslaagd is

4. Authenticatie: Register

- Register – **het resultaat**
  - bij een succesvolle register vinden we de nieuwe user in de DB, en werd een cookie aangemaakt



Name	Value	Domain	Pa...	Expires / Max-...	Size	Http...	Secure	Sam...
.AspNetCore.Antiforgery.-evGAmvD1Ik	CfDJ8I2NN...	localhost	/	Session	190	✓		Strict
.AspNetCore.Antiforgery.k2IoCaeisPY	CfDJ8I2NN...	localhost	/	Session	190	✓		Strict
.AspNetCore.Identity.Application	CfDJ8I2NN...	localhost	/	Session	656	✓	✓	Lax
.AspNetCore.Mvc.CookieTempDataProvider	CfDJ8I2NN...	localhost	/	Session	236	✓		Lax

Deze cookie wordt nu met elke request meegestuurd en geanalyseerd in de middleware…

```
SELECT TOP 1000 [Id]
      ,[Email]
      ,[NormalizedUserName]
      ,[PasswordHash]
      ,[UserName]
  FROM [aspnet-WebAppIdentityTemplate-d05d73cb-dbd0-419c-90bc-0d865904695c].[dbo].[AspNetUsers]
```

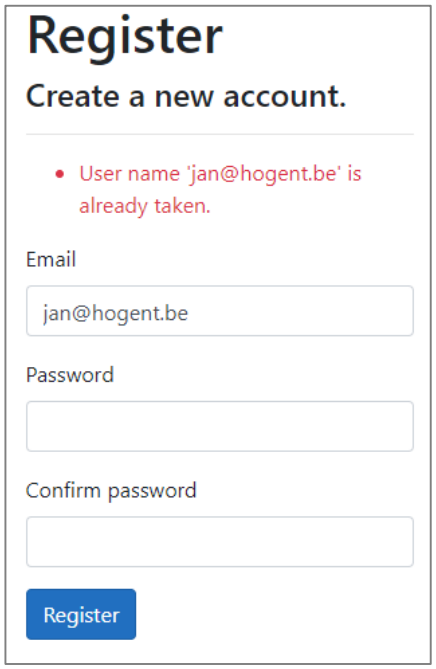
100 %

Results | Messages

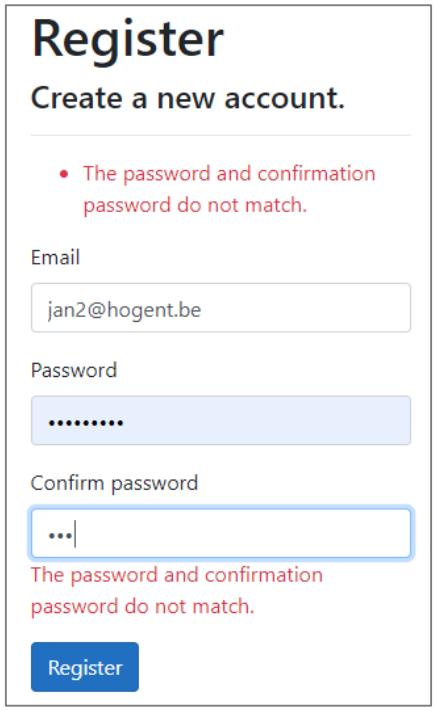
	Id	Email	NormalizedUserName	PasswordHash	UserName
1	2a9b20ee-5aeb-4665-ac3b-e36d3ff186fa	Jupke@hogent.be	JUPKE@HOGENT.BE	AQAAAAEAACcQAAAAEHjRLDklGE8wlk/GOApaCPzPfdIDd8DE...	Jupke@hogent.be

4. Authenticatie: Register

- Register [HttpPost] – **het resultaat**
  - bij een onsuccesvolle register krijgen we een overzicht van de errors

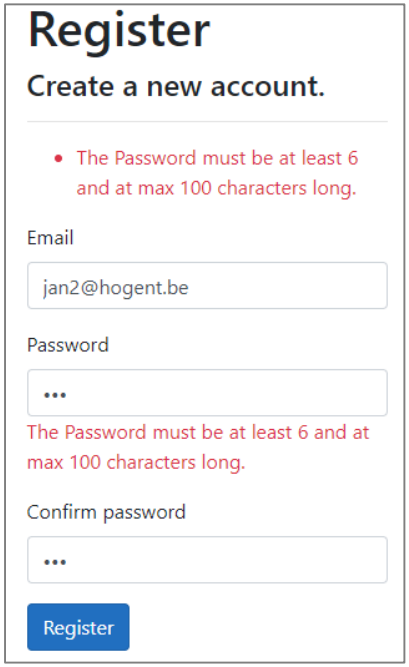


UserName (~e-mail) moet uniek zijn!



Deze error wordt herhaald. In de validation summary kiezen we in plaats van All beter voor ModelOnly! We kunnen dit aanpassen in Register.cshtml:

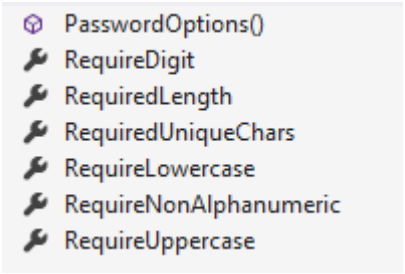
```
<div asp-validation-summary="ModelOnly"
class="text-danger"></div>
```



Blijkbaar legt Identity by default heel wat restricties op wachtwoorden…

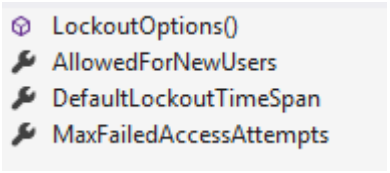
4. Authenticatie: Configuratie

- Identity werkt met default gedrag die je kan overschrijven in de StartUp klasse. Enkele voorbeelden:
  - Password policy



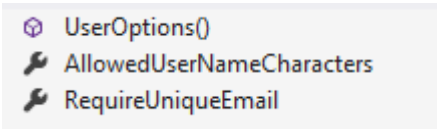
By default is de RequiredLength = 6, staan de boolse properties op true en is RequiredUniqueChars = 1.

  - User's lockout



By default wordt een user na 5 verkeerde pogingen locked out voor 5 minuten; ook nieuwe gebruikers kunnen locked out worden.

  - User validation



https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity-configuration?tabs=aspnetcore2x

4. Authenticatie: Configuratie

- Voorbeeld configuratie in de methode ConfigureServices van StartUp.cs

```
services.Configure<IdentityOptions>(options =>
            {
                // Password settings
                options.Password.RequiredLength = 8;
                options.Password.RequireNonAlphanumeric = false;

                // Lockout settings
                options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);

                // User settings
                options.User.RequireUniqueEmail = true;
            });
```

4. Authenticatie: Controller/Views

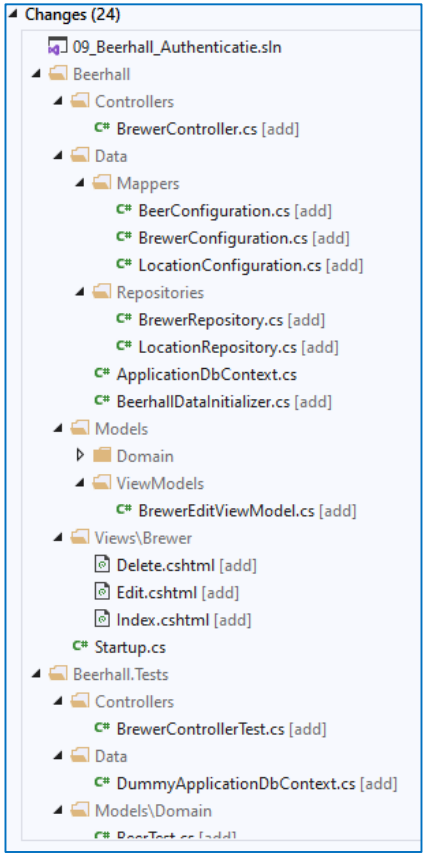
- Logout
  - AccountController > LogOff methode
    - Logt de gebruiker uit
    - Redirect naar Home
- Log in
  - AccountController > LogIn methode
    - Logt de gebruiker in via de SignInManager
    - Redirect naar de Url in de Request string
- AccountController bevat ook
    - ForgotPassword/ResetPassword
    - VerifyCode: 2 factor authenticatie
    - Externe login (via social providers)

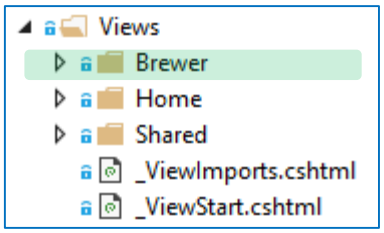
Authenticatie

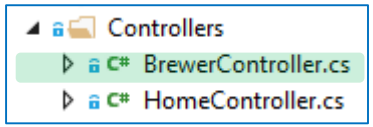
HoGent

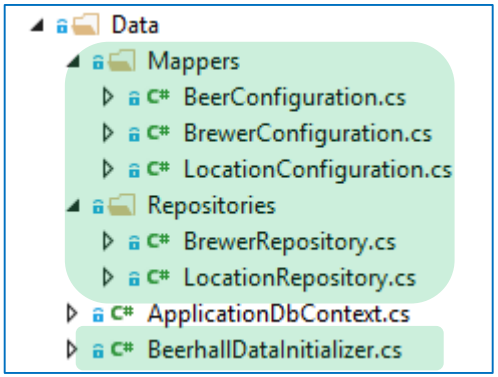
4. Authenticatie: Integratie Beerhall

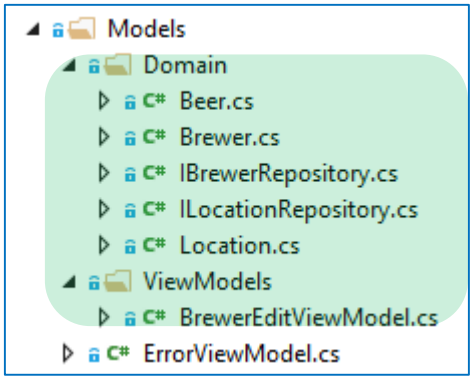
- We gaan onze Beerhall integreren in de application template gebaseerd op Individual User Accounts

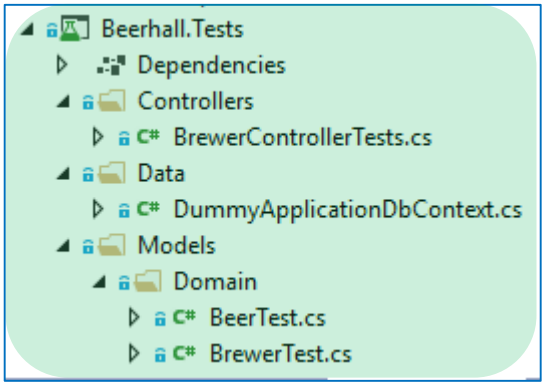












4. Authenticatie: Integratie Beerhall

- We gaan onze Beerhall integreren in de application gebaseerd op Individual User Accounts (vervolg)

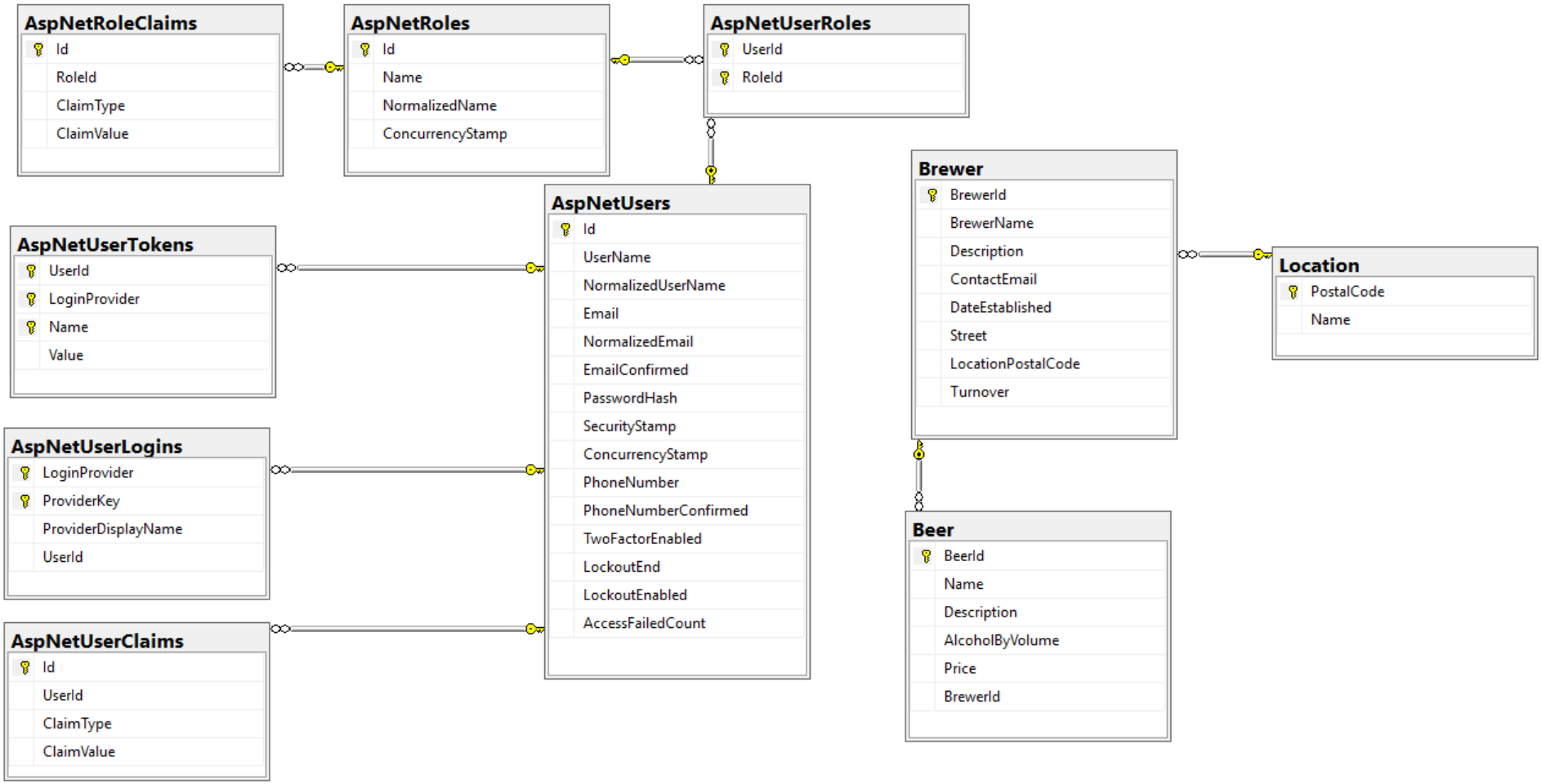
```
public class ApplicationDbContext : IdentityDbContext
  {

      public DbSet<Brewer> Brewers { get; set; }
      public DbSet<Location> Locations { get; set; }

      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
          : base(options)
      {
      }

      protected override void OnModelCreating(ModelBuilder builder)
      {
          base.OnModelCreating(builder);
          builder.ApplyConfiguration(new BrewerConfiguration());
          builder.ApplyConfiguration(new LocationConfiguration());
          builder.ApplyConfiguration(new BeerConfiguration());
...
      }
  }
```

Onze DbSets en de Mappers werden toegevoegd aan de ApplicationDbContext

```
"ConnectionStrings": {
    "DefaultConnection":
"Server=.;Database=BeerhallAuth;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
```

De databank zal BeerhallAuth noemen

4. Authenticatie: Integratie Beerhall



`commit Integrate Beerhall with the IUA template`

4. Authenticatie: Integratie Beerhall

- We kunnen nu onze DataInitializer methode bijwerken en enkele IdentityUsers toevoegen tijdens de seeding
  - in onze DataInitializer hebben we hiervoor naast de ApplicationDbContext ook nood aan de UserManager
  - we kunnen dit via een DI chain bewerkstelligen
    - vergelijk dit met de DI chain voor de repositories
      - in de BrewerController injecteren we een BrewerRepository
      - in de BrewerRepository injecteren we de ApplicationDbContext
  - we gaan de applicatie refactoren zodat dit mogelijk wordt…

4. Authenticatie: Integratie Beerhall

- Stap 1
  - in de klasse BeerhallDataInitializer injecteren we de UserManager

```
public class BeerhallDataInitializer {
        private readonly ApplicationDbContext _dbContext;
        private readonly UserManager<IdentityUser> _userManager;

        public BeerhallDataInitializer(ApplicationDbContext dbContext,
                                       UserManager<IdentityUser> userManager) {
            _dbContext = dbContext;
            _userManager = userManager;
}
        …
```

4. Authenticatie: Integratie Beerhall

- Stap 2
  - in een private async method maken we de users aan
    - de methodes van Identity zijn async, de methode waarin we ze gebruiken wordt dus eveneens async
    - een async void methode retourneert een Task

```
private async Task InitializeUsers() {
          string eMailAddress = "beermaster@hogent.be";
          IdentityUser user = new IdentityUser { UserName = eMailAddress, Email = eMailAddress };
          await _userManager.CreateAsync(user, "P@ssword1");

          eMailAddress = "jan@hogent.be";
          user = new IdentityUser { UserName = eMailAddress, Email = eMailAddress };
          await _userManager.CreateAsync(user, "P@ssword1");
      }
```

4. Authenticatie: Integratie Beerhall

- Stap 2 - vervolg
  - in de DataInitializer roepen we de methode InitializeUsers aan
    - de DataInitializer wordt dus ook een async methode
    - de aanroep wordt voorafgegaan door await

```
public async Task InitializeData() {
            _dbContext.Database.EnsureDeleted();
            if (_dbContext.Database.EnsureCreated()) {
                await InitializeUsers();
                Location bavikhove = new Location { Name = "Bavikhove", PostalCode = "8531" };
                …
```

4. Authenticatie: Integratie Beerhall

- Stap 3
  - In de StartUp klasse moeten we nu de aanroep naar InitializeData aanpassen want dit is nu een async methode
    - merk op: de Configure methode zelf kunnen we niet async maken, dit is een methode van het framework
    - via Wait() kunnen we wel aangeven dat we hier zullen wachten tot de geretourneerde Task beëindigd is

```
 public void Configure(IApplicationBuilder app, IHostingEnvironment env,
BeerhallDataInitializer beerhallDataInitializer) {
            …
            beerhallDataInitializer.InitializeData().Wait();
        }
```

4. Authenticatie: Integratie Beerhall

- Je kan de applicatie runnen en de resulterende DB bekijken
- Je kan in/uitloggen als een van de users uit de BeerhallDataInitializer
  - beermaster@hogent.be of jan@hogent.be

```
SELECT *
FROM [BeerhallAuth].[dbo].[AspNetUsers]
```

100 %

Results | Messages

	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash
1	0ade4356-74dd-43a6-a487-c4d8a8a84a39	beermaster@hogent.be	BEERMASTER@HOGENT.BE	beermaster@hogent.be	BEERMASTER@HOGENT.BE	0	AQAAAAEAACcQAAAAE
2	ea11855e-cd80-4b97-8c9d-8fe33875f032	jan@hogent.be	JAN@HOGENT.BE	jan@hogent.be	JAN@HOGENT.BE	0	AQAAAAEAACcQAAAAE

- Je kan je als nieuwe user registreren

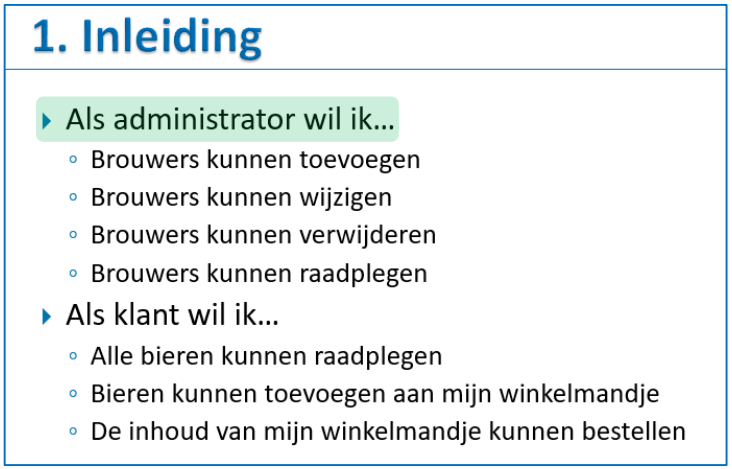
```
commit Seed IdentityUsers
```

Authorisatie

HoGent

4. Authorisatie

- Momenteel kan iedereen op onze site Brewers beheren. We gaan nu zorgen dat enkel gebruikers die geauthenticeerd zijn én administrator zijn dit kunnen…



uit Hfdstk 8…

4. Authorisatie

- Via het **AuthorizeAttribute** kunnen we actions in een controller afschermen
  - een request zal de action(s) voorzien van dit attribuut niet uitvoeren vooraleer een check is gebeurd
    - default check: een user is ingelogd
    - je kan dit uitbreiden met parameters en/of je eigen authorisatie check bouwen
- Via het **AllowAnonymousAttribute** kunnen we aangeven dat actions kunnen uitgevoerd worden zonder authorization check

- Deze attributen zijn gedefinieerd in de namespace Microsoft.AspNetCore.Authorization

4. Authorisatie

```
namespace BeerhallMVC.Controllers {
    [Authorize]
    public class BrewerController : Controller {
        private readonly IBrewerRepository _brewerRepository;
        private readonly ILocationRepository _locationRepository;

        public BrewerController(IBrewerRepository brewerRepository, ILocationRepository locationRepository) {
            _brewerRepository = brewerRepository;
            _locationRepository = locationRepository;
        }

        [AllowAnonymous]
        public IActionResult Index() {
            IEnumerable<Brewer> brewers = _brewerRepository.FindAll().OrderBy(b => b.Name).ToList();
            ViewData["TotalTurnover"] = brewers.Sum(b => b.Turnover);
            return View(brewers);
        }

        public IActionResult Edit(int id) {
            Brewer brewer = _brewerRepository.FindBy(id);
            ViewData["Locations"] = GetLocationsAsSelectList(brewer?.Location?.PostalCode);
            return View(new BrewerEditViewModel(brewer));
        }

        [HttpPost]
        public IActionResult Edit(BrewerEditViewModel brewerEditViewModel) {
            if (ModelState.IsValid)
              …
```

*De toegang tot de action methods in deze controller is nu beperkt tot **ge-authenticeerde** users.*

*Als je dit attribuut boven de controller klasse plaatst scherm je automatisch **elke** action in de controller af. Je kan het attribuut ook specifiek boven acties die je wil afschermen plaatsen…*

Dit attribuut overschrijft het attribuut dat op de controller werd geplaatst. Op deze manier is de Index toch voor iedereen bereikbaar…

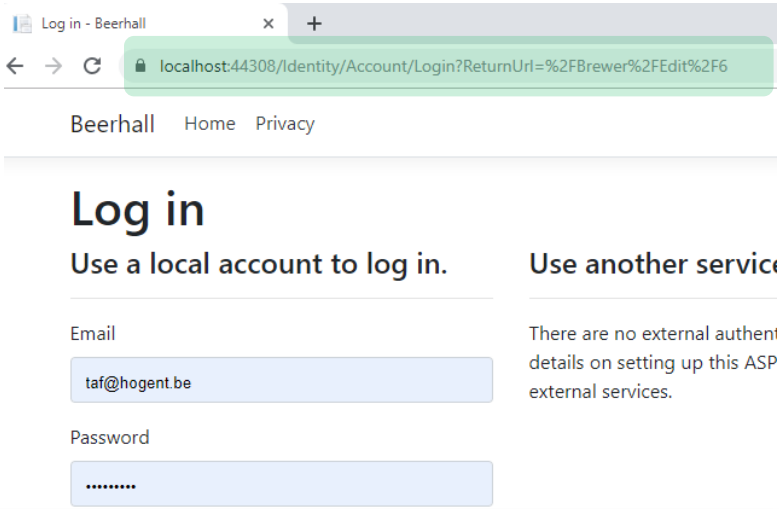
Warning

`[AllowAnonymous]` bypasses all authorization statements. If you apply combine `[AllowAnonymous]` and any `[Authorize]` attribute then the Authorize attributes will always be ignored. For example if you apply `[AllowAnonymous]` at the controller level any `[Authorize]` attributes on the same controller, or on any action within it will be ignored.

4. Authorisatie

- Wanneer we nu zonder inloggen een brewer proberen te editeren worden we omgeleid naar de login pagina. De return-url wordt als request parameter doorgegeven, eens we zijn ingelogd komen we op de Edit pagina
  - merk op dat de index nog steeds bereikbaar is voor iedereen…

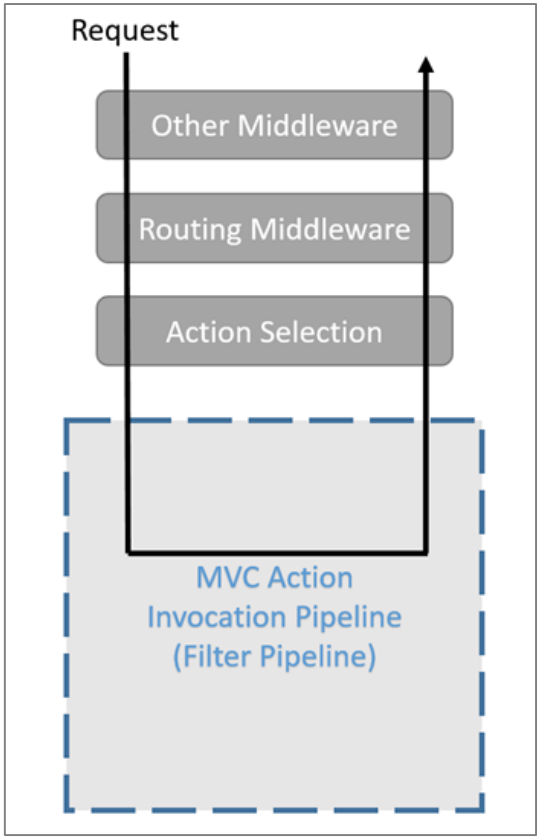
HOW?



`commit Add Authorize and AllowAnonymous attributes`

Focus on Identity middleware en filters

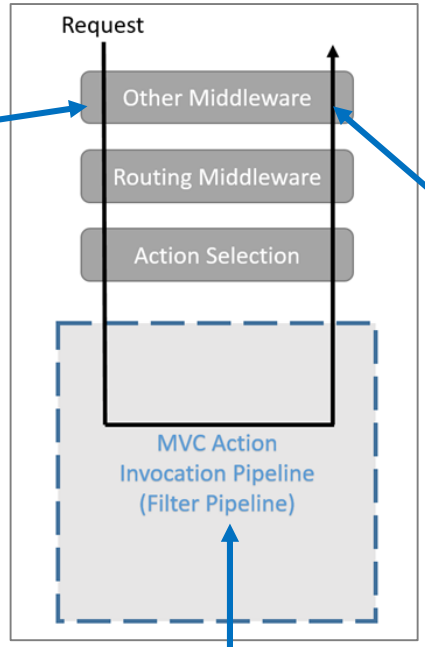
- Middleware en filters…
  - een request passeert door de middleware pipeline (zie hoofdstuk 6)
  - eens de request in het MVC framework komt worden enkele filters uitgevoerd voor/tijdens/na de uitvoering van de action method (dit noemen we de filter pipeline)
  - de response passeert op de weg terug weer door de middleware pipeline



Focus on Identity middleware en filters

- Wat gebeurt precies wanneer een gebruiker niet is ingelogd en klikt op Brewer/Edit?

de request passeert inkomend de Identity middleware
- er wordt geen authenticatie cookie gevonden en dus is er geen 'Signed In User'

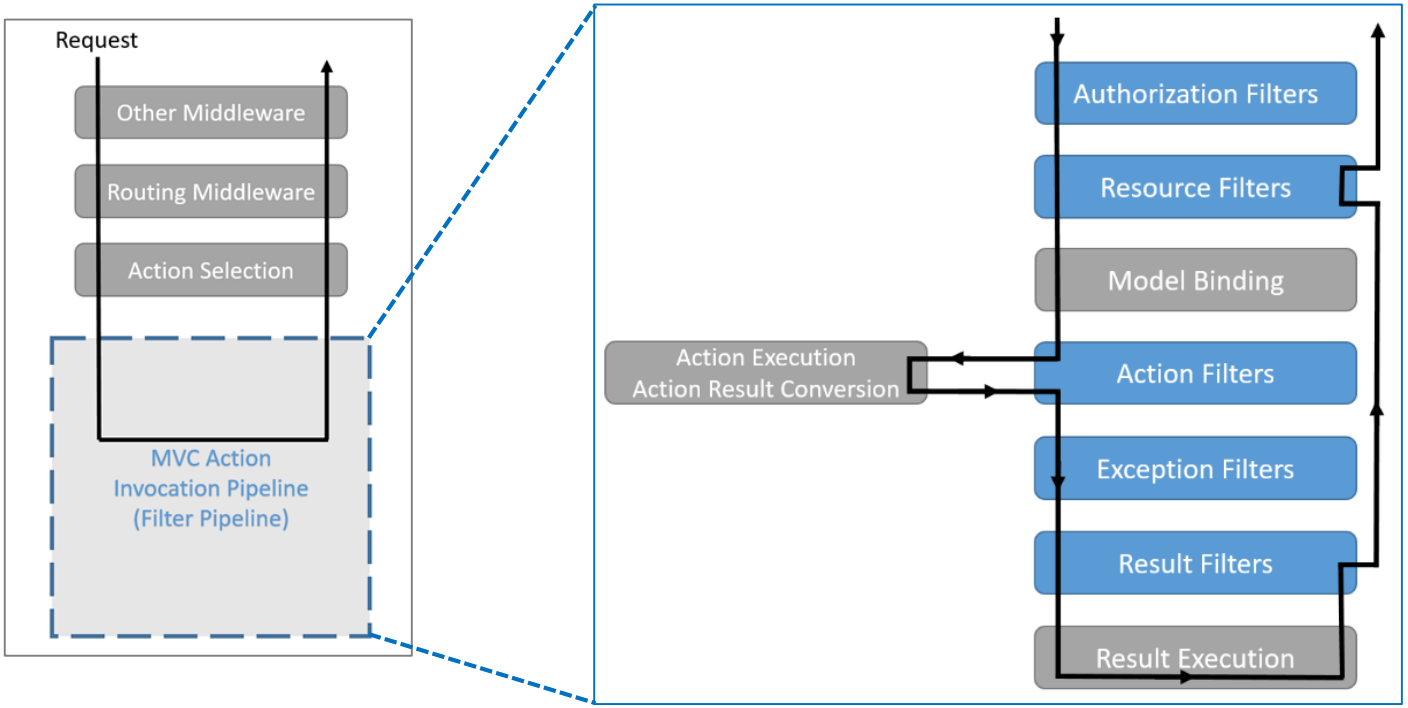


de response passeert uitgaand de Identity middleware, deze is default geconfigureerd
- om 401 responses om te zetten naar 302: found response
- de redirect URL is Account/Login
- de oorspronkelijke URL wordt in de vorm van request parameters toegevoegd aan de redirect URL

de Authorize filter wordt uitgevoerd en hier wordt vastgesteld er geen SignedInUser is, en dat er wel een [Authorize] attribuut is…
- het resultaat is een 401 response: Unauthorized

Focus on Identity middleware en filters

- De filter pipeline

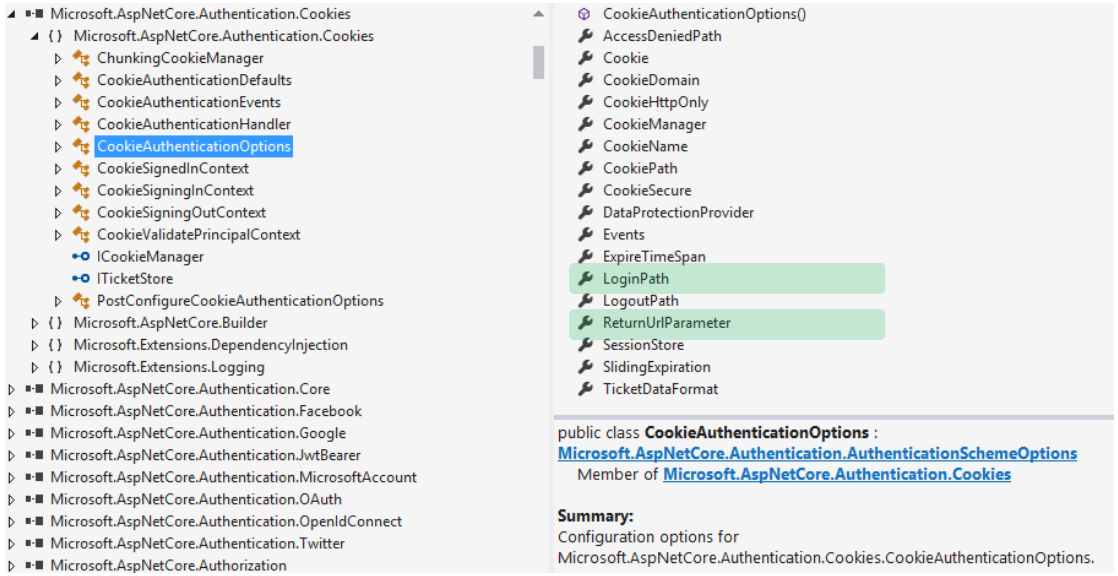


Authorization filters are used to determine whether the current user is authorized for the request being made

Different filter types run at different points within the pipeline. Some filters, like authorization filters, only run before the next stage in the pipeline, and take no action afterward. Other filters, like action filters, can execute both before and after other parts of the pipeline execute, as shown below.

Focus on Identity middleware en filters

- De configuratie van de application's cookie gebeurt in de ConfigureServices methode van de StartUp klasse



Summary:
The ReturnUrlParameter determines the name of the query string parameter which is appended by the middleware when a 401 Unauthorized status code is changed to a 302 redirect onto the login path. This is also the query string parameter looked for when a request arrives on the login path or logout path, in order to return to the original url after the action is performed.

Summary:
The LoginPath property informs the middleware that it should change an outgoing 401 Unauthorized status code into a 302 redirection onto the given login path. The current url which generated the 401 is added to the LoginPath as a query string parameter named by the ReturnUrlParameter. Once a request to the LoginPath grants a new SignIn identity, the ReturnUrlParameter value is used to redirect the browser back to the url which caused the original unauthorized status code.

Focus on Identity middleware en filters

- Configuratie Identity middleware
  - voorbeeld

```
services.ConfigureApplicationCookie(options =>
            {
                options.Cookie.Name = "YourAppCookieName";
                options.Cookie.HttpOnly = true;
                options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
                options.LoginPath = "/Account/MyLogin";
                options.LogoutPath = "/Account/Logout";
                options.AccessDeniedPath = "/Account/AccessDenied";
                // Requires `using Microsoft.AspNetCore.Authentication.Cookies;`
                options.ReturnUrlParameter =
CookieAuthenticationDefaults.ReturnUrlParameter;
            });
```

Bij een 401 wordt nu geredirect naar de action method MyLogin in de AccountController…

4. Authorisatie

- De Login – HttpPost (Login.cshtml.cs)

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
            returnUrl = returnUrl ?? Url.Content("~/");
            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid)
            {
                var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
                var result = await _userManager.CreateAsync(user, Input.Password);
                if (result.Succeeded)
                    result = await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Role, "customer"));
                if (result.Succeeded)
                {
                    _logger.LogInformation("User created a new account with password.");
// further code omitted
```

4. Authorisatie

- We kunnen de authorisatie verfijnen door gebruik te maken van rollen en/of claims.
  - Role based authorization
    - rollen laten toe dat we groepen van gebruikers als eenzelfde gaan beschouwen voor authorisatie
  - Claims based authorization
    - maakt gebruik van key/value pairs die iets vertellen over een gebruiker (geboortedatum, ssn, gender, sirname, …)
    - is flexibeler en krachtiger dan role based authorization
  - We gaan claims based authorization gebruiken om het onderscheid te maken tussen administrators en klanten

When an identity is created it may be assigned one or more claims issued by a trusted party. A claim is name value pair that represents what the subject is, not what the subject can do. For example you may have a Drivers License, issued by a local driving license authority. Your driver's license has your date of birth on it. In this case the claim name would be `DateOfBirth`, the claim value would be your date of birth, for example `8th June 1970` and the issuer would be the driving license authority. Claims based authorization, at its simplest, checks the value of a claim and allows access to a resource based upon that value. For example if you want access to a night club the authorization process might be:
The door security officer would evaluate the value of your date of birth claim and whether they trust the issuer (the driving license authority) before granting you access.

4. Authorisatie: Claims

- Claims based authorisation
  - Stap 1: een authorisatie policy definiëren
    - een policy is een verzameling van condities waaraan een user moet voldoen om toegelaten te worden tot een bepaalde resource.
    - je definieert een policy door de Authorization service te registreren en via de options de policy te beschrijven

```
services.AddAuthorization(options => {
                options.AddPolicy("AdminOnly", policy => policy.RequireClaim(ClaimTypes.Role, "admin"));
                options.AddPolicy("Customer", policy => policy.RequireClaim(ClaimTypes.Role, "customer"));
            });
```

in de methode ConfigureServices(…) van StartUp.cs

De Customer policy beschrijft dat de user een claim van het type 'Role' moet hebben en dat deze bovendien de waarde "customer" moet hebben.

4. Authorisatie: Claims

- Claims based authorisation
  - de policy die we definieerden verwacht dat een bepaalde claim aanwezig is én dat die een bepaalde waarde heeft
  - een simpele policy verwacht gewoon de aanwezigheid van een claim, voorbeeld

```
services.AddAuthorization(options => {
       options.AddPolicy("HasMobile", policy => policy.RequireClaim(ClaimTypes.MobilePhone));
});
```

4. Authorisatie: Claims

- Claims based authorisation
  - de klasse ClaimTypes

public static class **ClaimTypes**
Member of **System.Security.Claims**

Summary:
Defines constants for the well-known claim types that can be assigned to a subject. This class cannot be inherited.

- Email
- Expiration
- Expired
- Gender
- GivenName
- GroupSid
- Hash
- HomePhone
- IsPersistent
- Locality
- MobilePhone
- Name
- NameIdentifier
- OtherPhone
- PostalCode
- PrimaryGroupSid
- PrimarySid
- Role
- Rsa
- SerialNumber

Enkele voorbeelden van ClaimTypes

4. Authorisatie: Claims

- Claims based authorisation
  - Stap 2: zorgen dat de users de gepaste claims krijgen
    - BeerhallDataInitializer voor users die ge-seed worden

```
private async Task InitializeUsers() {
            string eMailAddress = "beermaster@hogent.be";
            ApplicationUser user = new ApplicationUser { UserName = eMailAddress, Email = eMailAddress };
            await _userManager.CreateAsync(user, "P@ssword1");
            await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Role, "admin"));
            eMailAddress = "jan@hogent.be";
            user = new ApplicationUser { UserName = eMailAddress, Email = eMailAddress };
            await _userManager.CreateAsync(user, "P@ssword1");
            await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Role, "customer"));        }
```

    - Register voor nieuwe users (dit worden automatisch ‘customers’)

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null) {
            returnUrl = returnUrl ?? Url.Content("~/");
            if (ModelState.IsValid) {
                var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
                var result = await _userManager.CreateAsync(user, Input.Password);
                if (result.Succeeded)
                    result = await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Role, "customer"));
                if (result.Succeeded)
                {
```

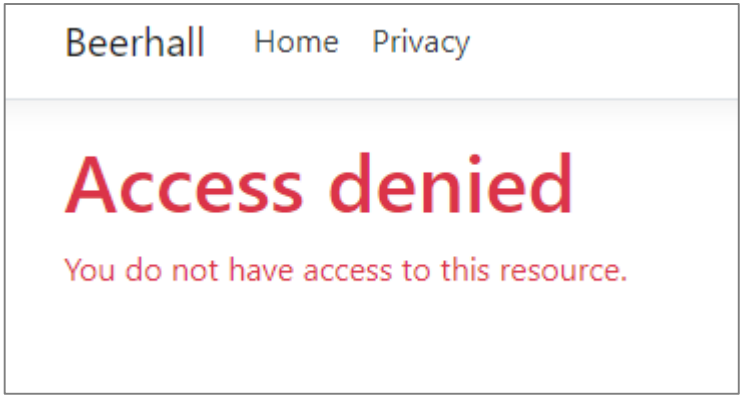
4. Authorisatie: Claims

- Claims based authorisation
  - Stap 3: gebruik de policy samen met het Authorize attribuut op controllers en/of action methods

```
[Authorize(Policy = "AdminOnly")]
   public class BrewerController : Controller {
       private readonly IBrewerRepository _brewerRepository;
       private readonly ILocationRepository _locationRepository;

       public BrewerController(IBrewerRepository brewerRepository, ILocationRepository locationRepository) {
           _brewerRepository = brewerRepository;
           _locationRepository = locationRepository;
       }
       [AllowAnonymous]
       public IActionResult Index() {
           IEnumerable<Brewer> brewers = _brewerRepository.GetAll().OrderBy(b => b.Name).ToList();
           ViewData["TotalTurnover"] = brewers.Sum(b => b.Turnover);
           return View(brewers);
       }
```

4. Authorisatie: Claims

- Indien een gebruiker is ingelogd maar geen claims heeft die aan de policy voldoen
  - krijg je van MVC een 403 response: Forbidden
    - per default wordt er geredirect naar Account/AccesDenied
    - gebruik scaffolding van Identity om deze pagina desgewenst aan te passen



4. Authorisatie: Claims

- Claims based authorisation
  - Merk op dat indien er meerdere policies op een controller/action gebruikt worden ze allemaal moeten slagen om toegang te verkrijgen

```
[Authorize(Policy = "AdminOnly")]
   public class BrewerController : Controller {
       private readonly IBrewerRepository _brewerRepository;
       private readonly ILocationRepository _locationRepository;

       public BrewerController(IBrewerRepository brewerRepository, ILocationRepository locationRepository) {
           _brewerRepository = brewerRepository;
           _locationRepository = locationRepository;
       }
       [Authorize(Policy = “Customer")]
       public IActionResult Index() {
           IEnumerable<Brewer> brewers = _brewerRepository.GetAll().OrderBy(b => b.Name).ToList();
           ViewData["TotalTurnover"] = brewers.Sum(b => b.Turnover);
           return View(brewers);
       }
```

Om nu aan de Index methode te kunnen moet je én aan de policy AdminOnly én aan de policy Customer voldoen…

4. Authorisatie: Claims

- Claims based authorisation
  - Je kan meer complexe policies bouwen door zelf een custom policy handler te implementeren
    - voorbeeld: op basis van een DateOfBirth claim toegang verlenen aan users die ouder zijn dan 21 jaar
    - zie https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies#security-authorization-policies-based

4. Authorisatie: Claims

- resultaat in DB na de seeding

```
SELECT TOP 1000 [Id]
      ,[ClaimType]
      ,[ClaimValue]
      ,[UserId]
  FROM [BeerhallAuth].[dbo].[AspNetUserClaims]
```

100 %

Results | Messages

	Id	ClaimType	ClaimValue	UserId
1	1	http://schemas.microsoft.com/ws/2008/06/identity/claims/role	admin	8085bda6-e272-4d8b-9bc7-c66e11a7bb60
2	2	http://schemas.microsoft.com/ws/2008/06/identity/claims/role	customer	f3be4b65-2872-49fe-9a0e-e4c3ba7d1286

```
SELECT TOP 1000 [Id]
      ,[Email]
      ,[UserName]
  FROM [BeerhallAuth].[dbo].[AspNetUsers]
```

%

Results | Messages

Id	Email	UserName
8085bda6-e272-4d8b-9bc7-c66e11a7bb60	beermaster@hogent.be	beermaster@hogent.be
f3be4b65-2872-49fe-9a0e-e4c3ba7d1286	jan@hogent.be	jan@hogent.be

```
commit Add claims-based authorization
```

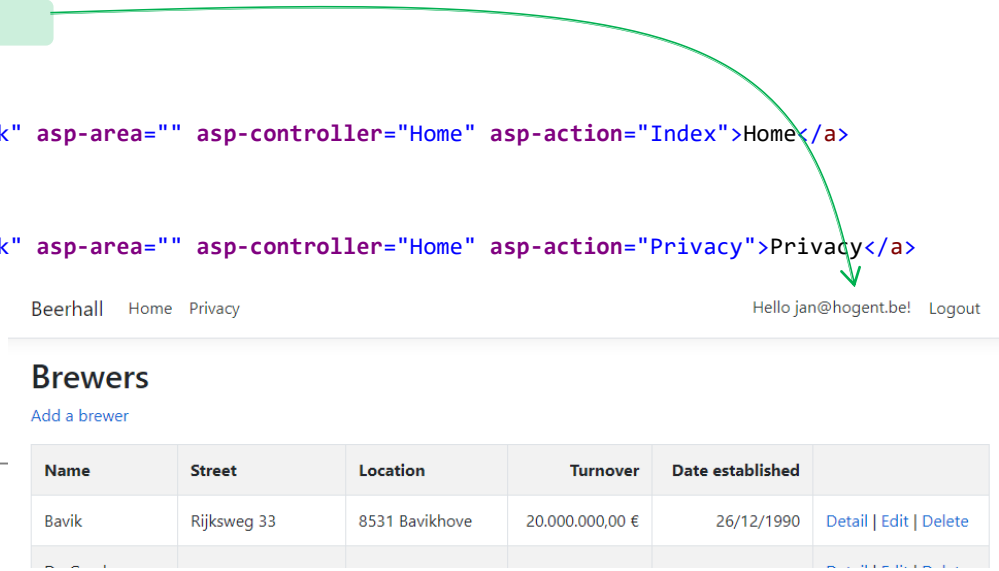
Identity in Views

HoGent

5. Identity in de Views

- De _Layout view maakt gebruik van een partial view _LoginPartial
  - Partial views worden in detail in volgend hoofdstuk behandeld

```
<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Beerhall</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                    aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                <partial name="_LoginPartial" />
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
```



5. Identity in de Views

- _LoginPartial

```
@using Microsoft.AspNetCore.Identity

@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager

@if (SignInManager.IsSignedIn(User))
{
    <form asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Action("Index", "Home", new { area = "" })"
method="post" id="logoutForm" class="navbar-right">
        <ul class="nav navbar-nav navbar-right">
            <li>
                <a asp-area="Identity" asp-page="/Account/Manage/Index" title="Manage">Hello @UserManager.GetUserName(User)!</a>
            </li>
            <li>
                <button type="submit" class="btn btn-link navbar-btn navbar-link">Logout</button>
            </li>
        </ul>
    </form>
}
else
{
    <ul class="nav navbar-nav navbar-right">
        <li><a asp-area="Identity" asp-page="/Account/Register">Register</a></li>
        <li><a asp-area="Identity" asp-page="/Account/Login">Login</a></li>
    </ul>
}
```

De managers worden geinjecteerd in de cshtml pagina

User is de gebruiker die momenteel is ingelogd

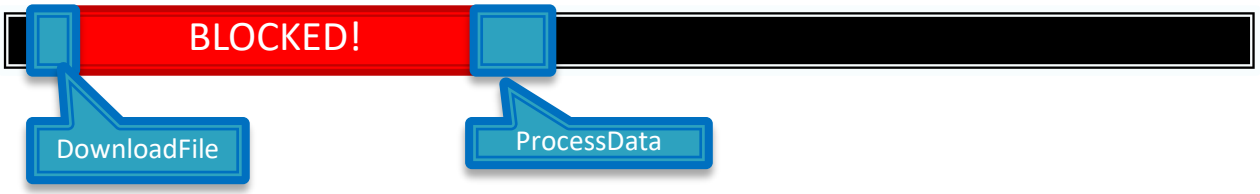
De Name claim van de ingelogde gebruiker wordt opgehaald

Asynchronous programming

HoGent

6. Synchronous vs Asynchronous

- Synchronous – Wacht op het resultaat alvorens te returnen
  - IData data = DownloadFile(…);
    ProcessData(data);



- Asynchronous – Returnt onmiddellijk, zal call back uitvoeren als resultaat ontvangen
  - Task<IData> future = DownloadFileAsync(…);
    future.ContinueWith(data => ProcessData(data));

Task representeert een asynchrone operatie die in dit voorbeeld een IData object zal returnen

6. Asynchronous

- Asynchroon programmeren – eenvoudig.
  - “async” and “await” keywords
    - Geen callbacks meer nodig
    - Zorgt voor compiler magic
    - Voorbeeld: registreren van een gebruiker

```
public async Task<IActionResult> Register(RegisterViewModel model, string returnUrl = null)
        {
            ViewData["ReturnUrl"] = returnUrl;
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
                var result = await _userManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    // For more information on how to enable account confirmation and password reset please …
                    await _signInManager.SignInAsync(user, isPersistent: false);
                    _logger.LogInformation(3, "User created a new account with password.");
                    return RedirectToLocal(returnUrl);
                }
                AddErrors(result);
            }
```

6. Asynchronous

- Compiler magic
  - **async** keyword informeert de compiler dat het gaat om een asynchrone methode
    - Een asynchrone methode
      - Conventie: Async suffix voor de methode naam (behalve in de Controller)
      - Returnt steeds een Task (voor void) of Task<T> voor returntype T
      - Bevat minstens 1 await
  - **await** markeert een punt in de code waar met de uitvoering van de rest van die code wordt gewacht totdat de uitvoering van de asynchrone methode is beëindigd. Wel wordt de controle geretourneerd naar de “caller” methode waar wordt verdergegaan met de uitvoering van de code. De code na de await wordt gecompileerd als een callback van de asynchrone Task(thread). De bijhorende context (variabelen,….) wordt bijgehouden. Als de taak beëindigd is wordt verdergegaan met de uitvoering van de rest van de methode. Hierbij wordt gebruik gemaakt van de synchronisation context.

6. Asynchronous

- Open het project MultiThreading.sln
- Run de applicatie

```
Continues executing....
The main thread continues executing .....
Byebye
back again
```

```
static void Main(string[] args)
{
   DoLongTask();
   Console.WriteLine("The main thread continues executing .....");
   Console.ReadLine();
}

public static void DoLongTask()
{
  Task.Run(new Action(LongTask)); //Start een Task(thread)
   Console.WriteLine("Continues executing....");
}
public static void LongTask()
{
   Console.WriteLine("Byebye");
   Thread.Sleep(5000);  //5 seconden
   Console.WriteLine("back again");
}
```

Stap 1

Stap 2

Stap 3

Stap 2 gaat verder in aparte thread

6. Asynchronous

- Vervang DoLongTaskMethod door DoLongTaskMethodAsync

```
The main thread continues executing .....
Byebye
back again
Continues executing
```

```
static void  Main(string[] args)
{
   DoLongTaskAsync();
   Console.WriteLine("The main thread continues executing .....");
   Console.ReadLine();
}

 public static async Task DoLongTaskAsync()
{
   await Task.Run(new Action(LongTask));
   Console.WriteLine("Continues executing");
}

public static void LongTask()
{
   Console.WriteLine("Byebye");
   Thread.Sleep(5000);  //5 seconden
   Console.WriteLine("back again");
}
```

Stap 1

Stap 2

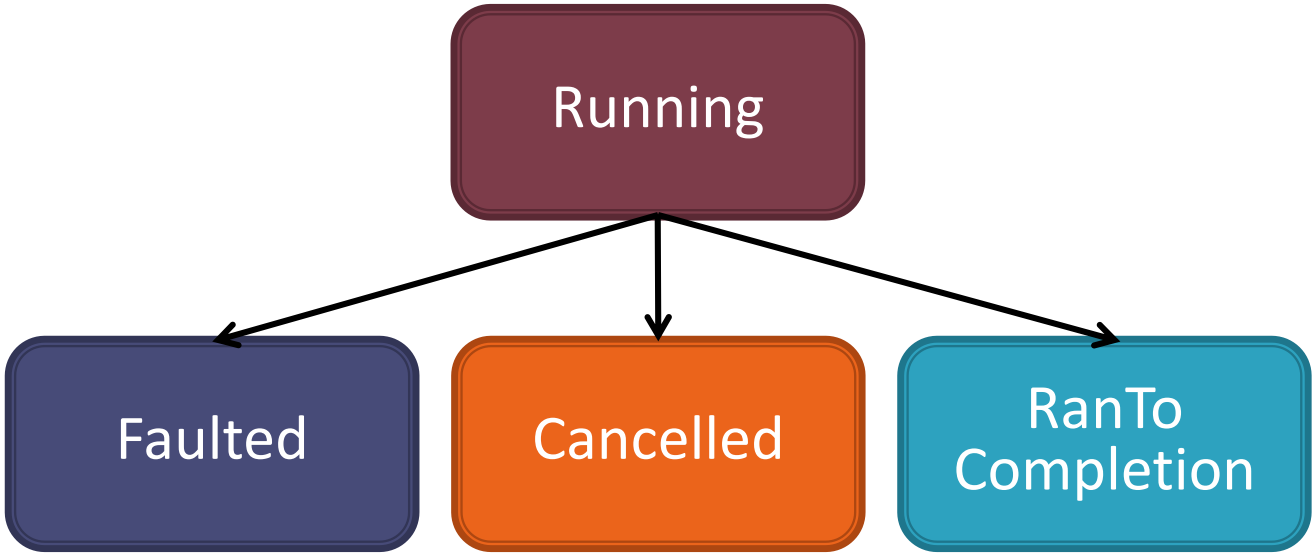
stap3

Dit wordt gecompileerd als callback methode van de asynchrone Task en wordt pas uitgevoerd als de taak beëindigd is.

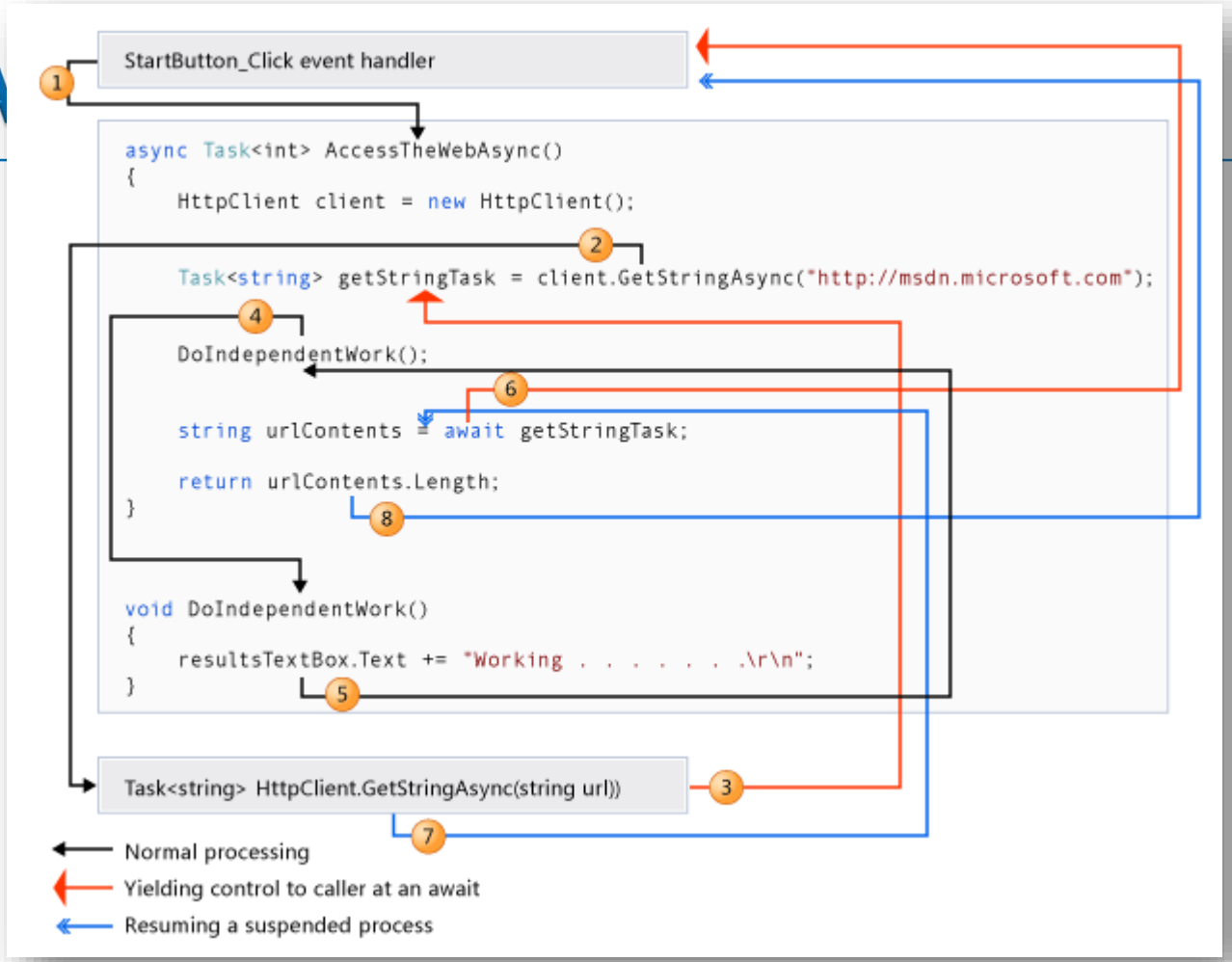
Ho

6. Asynchronous

- Een taak (class Task) representeert een asynchrone operatie die een resultaat kan teruggeven. (Task retourneert void, Task<T> retourneert T). De status van een taak kan worden opgevraagd door de calling code. Meer over de status op http://msdn.microsoft.com/en-us/library/system.threading.tasks.taskstatus(v=vs.110).aspx



6. A



- http://msdn.microsoft.com/en-us/library/vstudio/hh191443.aspx#BKMK_WhatHappensUnderstandingan AsyncMethod

6. Asynchronous

- Synchroon

```
public ActionResult Index()
{
        IList<Brewer> brewers = brewerRepository.GetAll();
        IList<BrewerViewModel> vm = ….
        return View(vm);
}
```

- Asynchroon
  - Duurt even lang
  - Maar de server kan intussen andere requests bedienen

Meer op http://msdn.microsoft.com/en-us/library/ee728598(VS.100).aspx en http://www.asp.net/mvc/tutorials/mvc-4/using-asynchronous-methods-in-aspnet-mvc-4

```
public async Task<ActionResult> Index()
{
        IList<Brewer> brewers= await
        brewerRepository.FindAllAsync();
        IList<BrewerViewModel> vm = ….
        return View(vm);
}
```

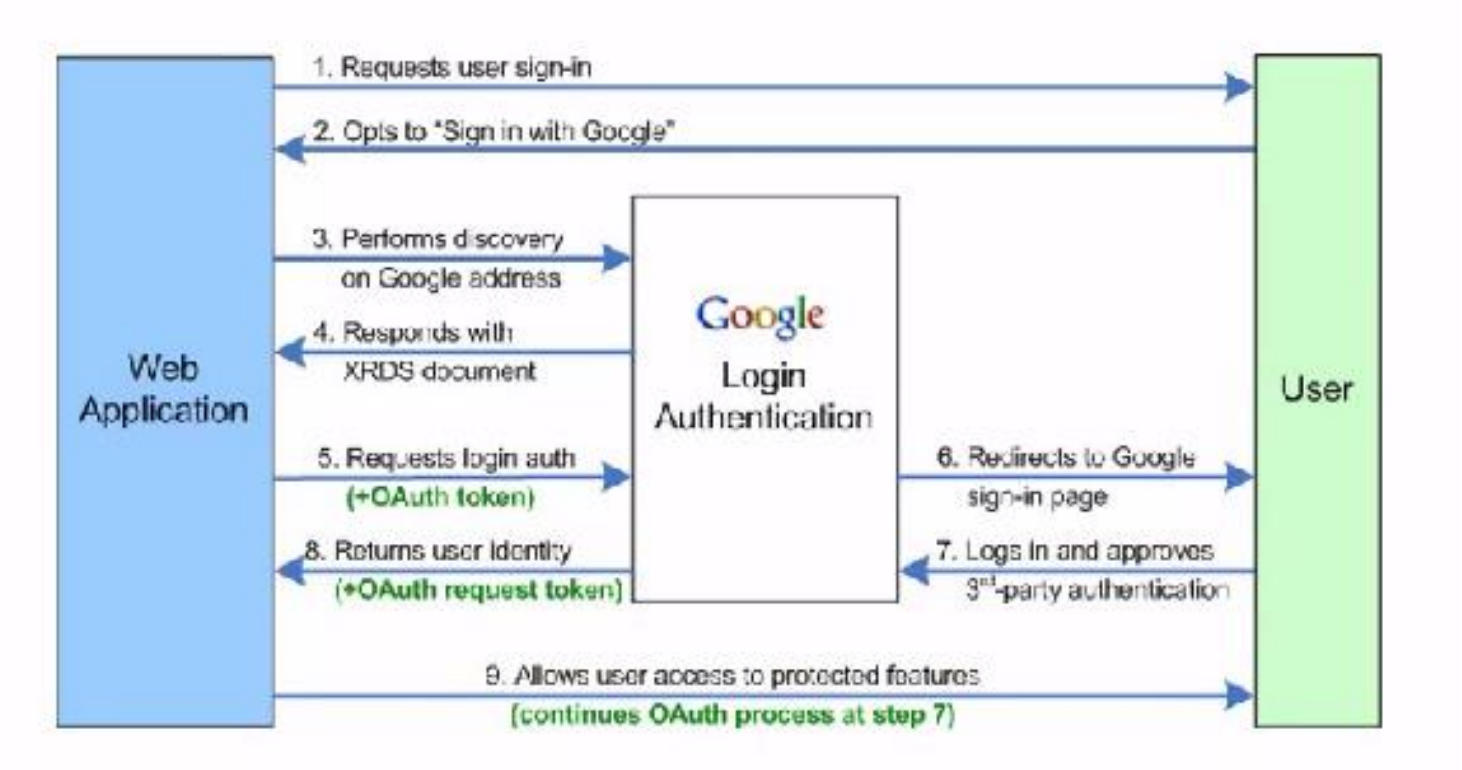
Authenticatie

7. Authenticatie: OpenId en OAuth

- Authenticatie via externe sites (Facebook, Twitter, …)
- Meer op
  - OpenId: http://openid.net/
  - OAuth: http://oauth.net/
  - DotNetOpenAuth: http://www.dotnetopenauth.net/
- Configuratie
  - Registreer eerst je site op Facebook, Google…. => je krijgt een key en een secret. Meer op https://github.com/aspnet-contrib/AspNet.Security.OAuth.Providers

7. Authenticatie: OpenId en OAuth

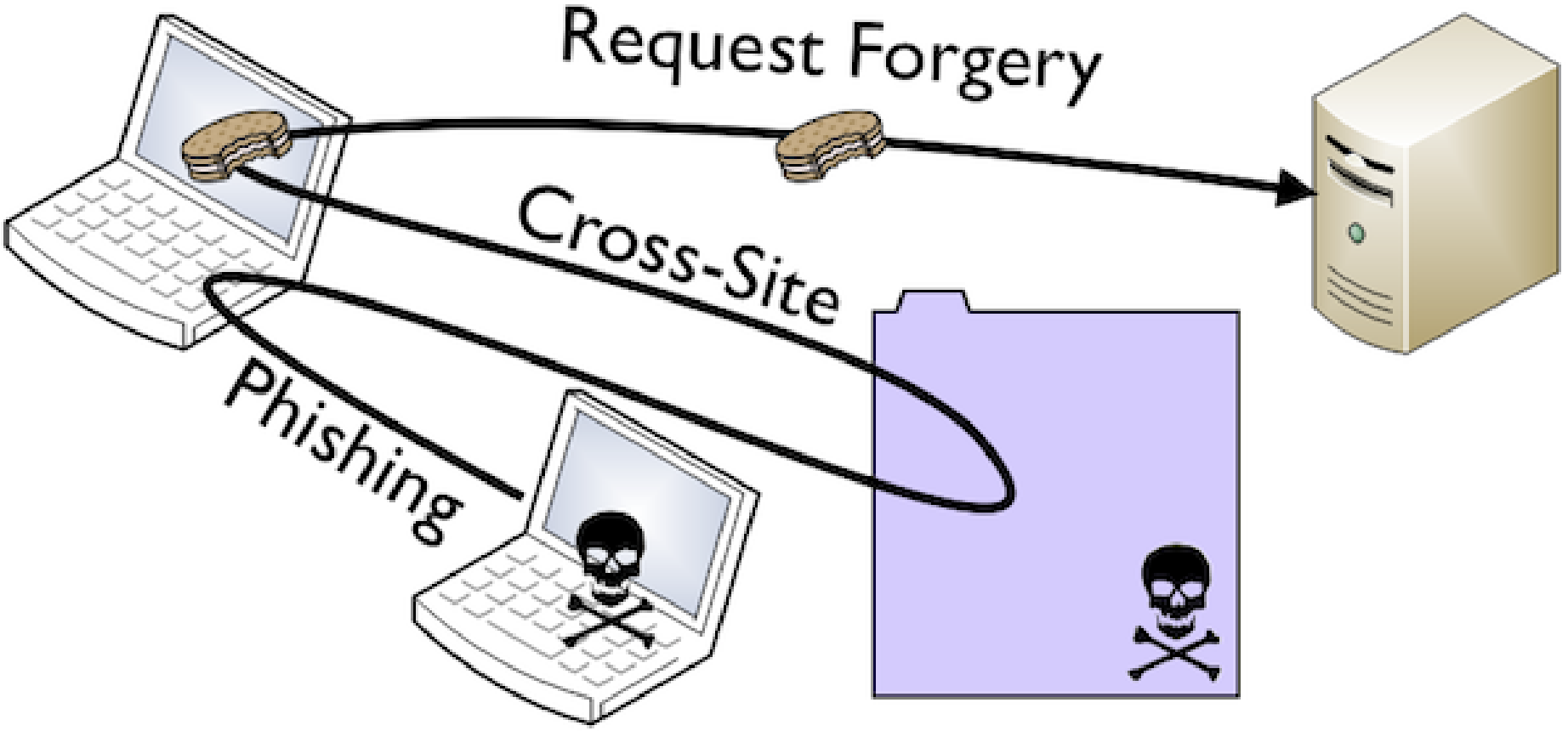
- Authenticatie via externe sites (Facebook, Twitter,…)



Security

HoGent

8. CSRF



8. CSRF

- CSRF : Cross-Site Request Forgery

```
public class UserProfileController : Controller
{
    public ViewResult Edit() { return View(); }

    public ViewResult SubmitUpdate()
    {
        // Get the user's existing profile data (implementation omitted)
        ProfileData profile = GetLoggedInUserProfile();

        // Update the user object
        profile.EmailAddress = Request.Form["email"];
        profile.FavoriteHobby = Request.Form["hobby"];
        SaveUserProfile(profile);

        ViewData["message"] = "Your profile was updated.";
        return View();
    }
}
```

  - Op andere site

```
<body onload="document.getElementById('fm1').submit()">
    <form id="fm1" action="http://yoursite/UserProfile/SubmitUpdate" method="post">
        <input name="email" value="hacker@somewhere.evil" />
        <input name="hobby" value="Defacing websites" />
    </form>
</body>
```

8. CSRF

- 2 manieren om te stoppen
  - Referer header moet naar jouw domein verwijzen
  - Plaats user-specific token in een hidden field in een formulier

- Wanneer je gebruik maakt van de form tag helper wordt automatisch een hidden '__RequestVerificationToken' geplaatst in de form

```
▼<form method="post" action="/Brewer/Edit/1" novalidate="novalidate">
  ▶<div class="validation-summary-valid" data-valmsg-summary="true">…</div>
   <input type="hidden" data-val="true" data-val-required="The BrewerId field is required."
   id="BrewerId" name="BrewerId" value="1">
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div>…</div>
   <input name="__RequestVerificationToken" type="hidden" value=
   "CfDJ8OIHYFYge0dPrEOE92DFCD9iV3HI3Q7VyvYsXH339DJ5AWAZYfUug6AO-
   NYXCETaYn2AZh1N1uFIQv5E0yn4FcCa1JQaNSnOpY2tG0EY_St05rLjFBvUFRMpRqyUM5KEIzkpPR6wXkwgsX_zKj
   Bx4L68pbqtpCfNXZZc46opuKYrw0ZBV3kkIZbxAeZzrJ8dhA">
</form>
```

8. CSRF

- In Controller: HttpPost
  - plaats filter [ValidateAntiForgeryToken] boven HttpPost methode
    - incoming request bevat een cookie **__RequestVerificationToken**
    - incoming request heeft een Request.Form entry **__RequestVerificationToken**
    - beide moeten matchen
    - indien deze niet overeenkomen krijg je een authorization failure: “A required anti-forgery token was not supplied or was invalid”.

```
[ValidateAntiForgeryToken]
[HttpPost]
public IActionResult Edit(EditViewModel brewerEditViewModel) {
…
}
```

    - Je kan dit attribuut ook boven de controller plaatsen maar dan werk je te restrictief op HttpGet methodes

8. CSRF

- In Controller: HttpPost
  - filter [AutoValidateAntiforgeryToken]
    - HttpPost acties zijn automatisch beveiligd
    - Er zijn geen tokens nodig voor de HttpGet requests

Appendix: Security

HoGent

Appendix : Hashing

Appendix Hashing

- Identity Framework bevat klasse PasswordHasher (gebruikt PBKDF2 with HMAC-SHA1, 128-bit salt, 256-bit subkey, 1000 iterations)

```
PasswordHasher hasher = new PasswordHasher();
string pwd = hasher.HashPassword("password");
```

- voor meer info zie https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing

- Meer info over cryprographic hashing : https://crackstation.net/hashing-security.htm

Referenties

HoGent

Referenties

- Pro ASP.NET Core MVC: Sixth edition by Adam Freeman - Apress - ISBN-13 (pbk): 978-1-4842-0398-9 ISBN-13 (electronic): 978-1-4842-0397-2
- Microsoft docs https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity
- PluralSight – ASP.NET Core Fundamentals – Scott Allen – see Chapter ASP.NET Identity https://app.pluralsight.com/library/courses/aspdotnet-core-fundamentals/table-of-contents
- PluralSight – ASP.NET Core Fundamentals – Scott Allen – see Chapter ASP.NET Identity https://app.pluralsight.com/library/courses/aspdotnet-core-fundamentals/table-of-contents
- .NET Web Development and Tools – Get Started with ASP.NET Core Authorization – Part 1 of 2 https://blogs.msdn.microsoft.com/webdev/2016/03/15/get-started-with-asp-net-core-authorization-part-1-of-2/
- .NET Web Development and Tools – Get Started with ASP.NET Core Authorization – Part 2 of 2 https://blogs.msdn.microsoft.com/webdev/2016/03/23/get-started-with-asp-net-core-authorization-part-2-of-2/
- Binary Intellect site – Implement Security Using ASP.NET Core Identity in 10 easy steps http://www.binaryintellect.net/articles/b957238b-e2dd-4401-bfd7-f0b8d984786d.aspx