

# HoGent

BEDRIJF  
EN  
ORGANISATIE

## Hoofdstuk 10: MVC Advanced

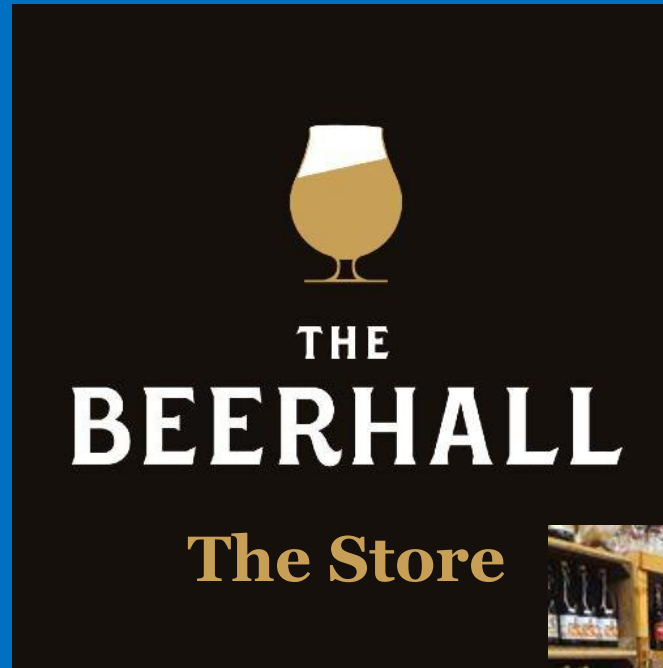
<https://github.com/WebIII/10thBeerhallMvcAdv.git>

# Hoofdstuk 10: MVC Advanced

---

- ▶ Store and Cart
- ▶ De use case Checkout
- ▶ TDD Cart – Checkout GET
  - Model binding
  - Viewmodels
- ▶ TDD Cart - Checkout POST
- ▶ TDD Register

# De Beerhall applicatie uitbreiden









HoGent

# De store

## ► Index

[Beerhall](#) [Brewer](#) [Store](#) [Cart](#) [Contact](#) [About us](#) [Register](#) [Log in](#)

### The Beer Store


	Bavik Pils	0,80 €	<input type="text" value="1"/>	<a href="#">Add to cart</a>		Belle-Vue	1,25 €	<input type="text" value="1"/>	<a href="#">Add to cart</a>
	Black Hole	1,68 €	<input type="text" value="1"/>	<a href="#">Add to cart</a>		De Koninck	0,79 €	<input type="text" value="1"/>	<a href="#">Add to cart</a>
	Dobbel Palm	1,15 €	<input type="text" value="1"/>	<a href="#">Add to cart</a>		Duvel	1,78 €	<input type="text" value="1"/>	<a href="#">Add to cart</a>

# De store

## ► StoreController - Index

```
public class StoreController : Controller {  
    private readonly IBeerRepository _beerRepository;  
  
    public StoreController(IBeerRepository beerRepository) {  
        _beerRepository = beerRepository;  
    }  
    public ActionResult Index() {  
        return View(_beerRepository.GetAll().OrderBy(b => b.Name).ToList());  
    }  
}
```

```
public interface IBeerRepository {  
    IEnumerable<Beer> GetAll();  
    Beer GetBy(int beerId);  
}
```



```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser> {  
    ...  
    public DbSet<Brewer> Brewers { get; set; }  
    public DbSet<Beer> Beers { get; set; }  
    ...  
}
```

*we willen nu rechtstreeks met Beers werken, niet via Brewers*

# De store

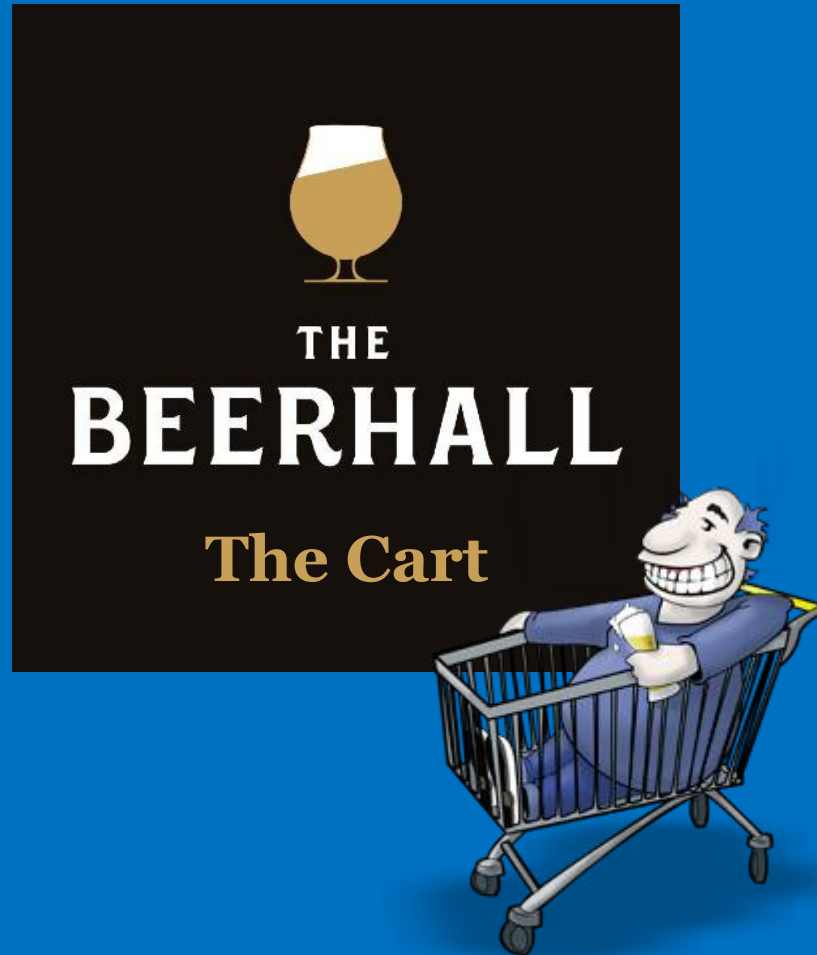
## ► Store – View - Index

```
@foreach (var beer in Model) {  
    <div class="col-md-6 col-xs-12">  
          
        <h3>  
            @beer.Name  
            <span class="pull-right">@($"{beer.Price:N} €")</span>  
        </h3>  
        <form asp-controller="Cart" asp-action="Add" asp-route-id="@beer.BeerId">  
            <div class="form-group pull-right">  
                <label class="sr-only" for="quantity">Quantity</label>  
                <input type="number" name="quantity" value="1" min="1" style="width:5em" />  
                <button type="submit" class="btn btn-default">  
                    <span class="glyphicon glyphicon-shopping-cart"></span> Add to cart  
                </button>  
            </div>  
        </form>  
    </div>  
}
```

quantity is onderdeel van de form data

wanneer we een beer met id 3 toevoegen aan de cart wordt de URL: Cart/Add/3

# MVC Advanced Action filters



HoGent

## ► Cart - Index

Beerhall

Brewer

Store

Cart

Contact

About us

Register

Log in

Your shopping cart

Product	Unit price	Subtotal	
5 x Ename	2,19 €	10,95 €	<a href="#">✕ Remove</a>
24 x Jupiler	1,19 €	28,56 €	<a href="#">✕ Remove</a>
		Total: 39,51 €	

Continue shopping

Check out



# De store

## ► CartController - Index

```
public IActionResult Index() {  
    Cart cart = ReadCartFromSession();  
    ViewData["Total"] = cart.TotalValue;  
    return View(cart.CartLines.Select(c => new IndexViewModel(c)).ToList());  
}
```

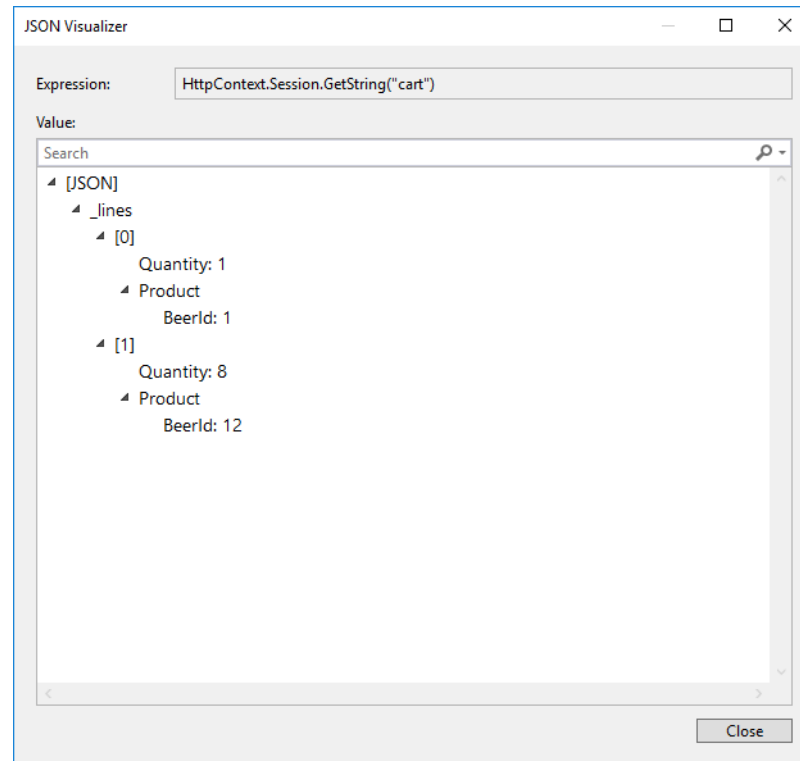
we zullen de cart niet opslaan in de databank maar gebruik maken van een session...

via een Select vormen we de IEnumerable<CartLine> om tot een IEnumerable<CartLineViewModel>

```
public class IndexViewModel {  
    [HiddenInput]  
    public int BeerId { get; }  
    public int Quantity { get; }  
    public string Beer { get; }  
    public decimal Price { get; }  
    public decimal SubTotal { get; }
```

# De store

- ▶ CartController – de cart session
  - we gaan de session zo klein mogelijk houden via gepaste annotaties in het domein
    - zie Cart, CartLine, Beer



# De store

- ▶ CartController – de cart session
  - telkens we de session lezen, halen we de producten (beers) op via de repository

```
private Cart ReadCartFromSession() {  
    Cart cart = HttpContext.Session.GetString("cart") == null  
        ? new Cart()  
        : JsonConvert.DeserializeObject<Cart>(HttpContext.Session.GetString("cart"));  
    foreach (var l in cart.CartLines)  
        l.Product = _beerRepository.GetBy(l.Product.BeerId);  
    return cart;  
}  
  
private void WriteCartToSession(Cart cart) {  
    HttpContext.Session.SetString("cart", JsonConvert.SerializeObject(cart));  
}
```

# De store

## ► Cart – View - Index

```
<tbody>
    @foreach (var line in Model) {
        <tr>
            <td class="text-right">@($"{line.Quantity} x")</td>
            <td>@line.Beer</td>
            <td class="text-right">@($"{line.Price:N2} €")</td>
            <td class="text-right">@($"{line.SubTotal:N2} €")</td>
            <td>
                <form method="post" asp-action="Remove" asp-route-id="@line.BeerId">
                    <button type="submit" class="btn btn-xs"><span class="glyphicon-remove
glyphicon"></span> Remove</button>
                </form>
            </td>
        </tr>
    }
</tbody>
```

# De store

## ► CartController Add/Remove

```
[HttpPost]
public ActionResult Remove(int id) {
    try {
        Cart cart = ReadCartFromSession();
        Beer product = _beerRepository.GetBy(id);
        cart.RemoveLine(product);
        TempData["message"] = $"{product.Name} was removed from your cart";
        WriteCartToSession(cart);
    }
    catch {
        TempData["error"] = "Sorry, something went wrong, the product was not removed from your cart...";
    }
    return RedirectToAction("Index");
}
```

```
[HttpPost]
public IActionResult Add(int id, int quantity = 1) {
    try {
        Cart cart = ReadCartFromSession();
        Beer product = _beerRepository.GetBy(id);
        if (product != null) {
            cart.AddLine(product, quantity);
            TempData["message"] = $"{quantity} x {product.Name} was added to your cart";
            WriteCartToSession(cart);
        }
    }
    catch {
        TempData["error"] = "Sorry, something went wrong, the product could not be added to your cart...";
    }
    return RedirectToAction("Index", "Store");
}
```



commit Add Cart - Index/Add/Remove

# Focus on action filters

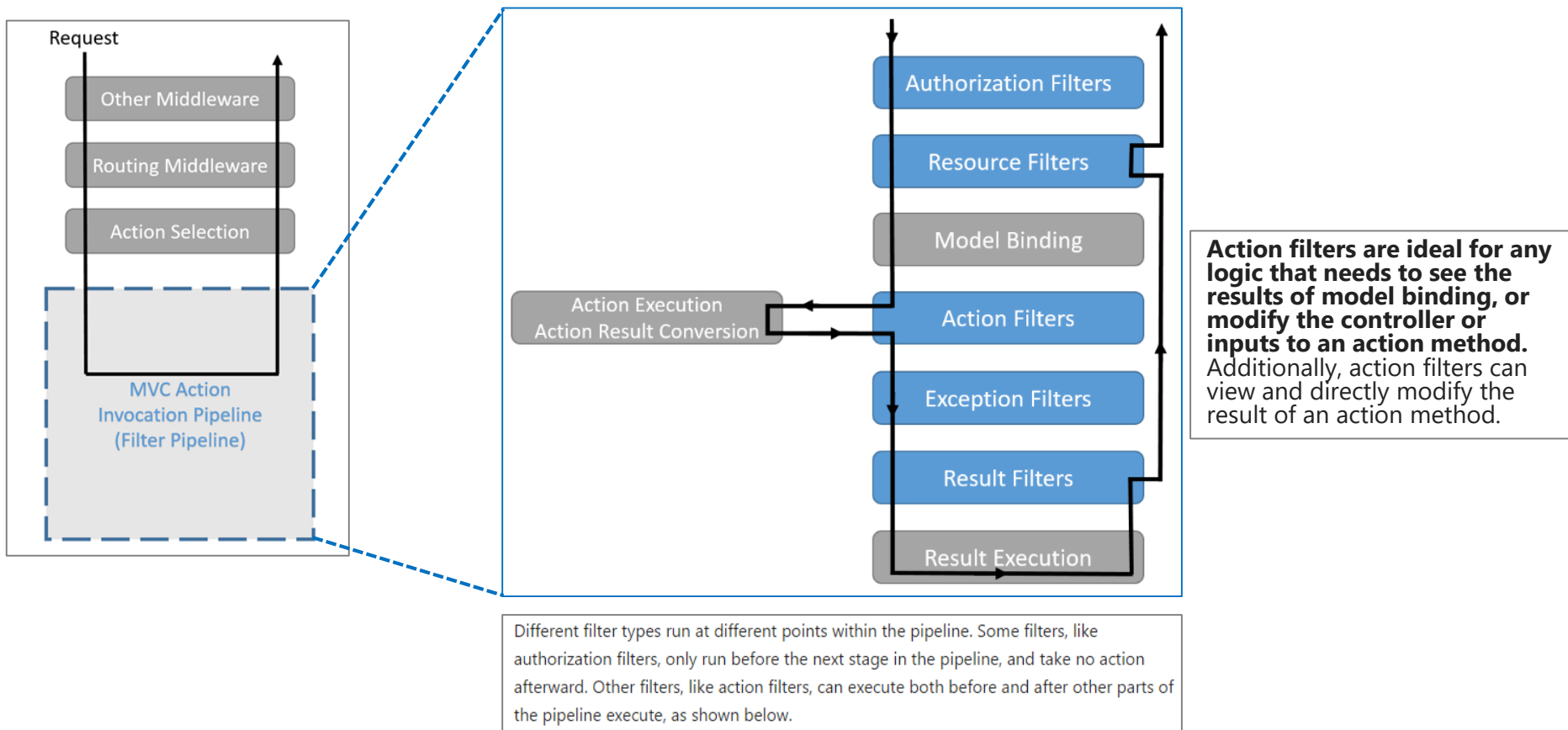
- ▶ De cart wordt bijgehouden in een Session.
  - Dit vormt een probleem voor het unit testen
  - Oplossing?
    - we kunnen gebruik maken van een mocking framework om de session te mocken, of
    - we kunnen gebruik maken van de MVC pipeline om de **cart als argument** aan de action method door te geven
      - de action methodes die de cart nodig hebben, krijgen die nu aangeleverd via een Cart parameter
      - dit zal leiden tot duidelijke en testable code in de controller:



```
public IActionResult Index(Cart cart) {  
    ViewData["Total"] = cart.TotalValue;  
    return View(cart.CartLines.Select(c => new IndexViewModel(c)).ToList());  
}
```

# Focus on action filters

## ► Hoe?



# Focus on action filters

## ► Action filters

- implementeren `IActionFilter` of `IAsyncActionFilter`
- twee belangrijke methodes

- `OnActionExecuting`



The `OnActionExecuting` method runs before the action method, so it can manipulate the inputs to the action by changing `ActionExecutingContext.ActionArguments` or manipulate the controller through `ActionExecutingContext.Controller`. An `OnActionExecuting` method can short-circuit execution of the action method and subsequent action filters by setting `ActionExecutingContext.Result`. Throwing an exception in an `OnActionExecuting` method will also prevent execution of the action method and subsequent filters, but will be treated as a failure instead of successful result.

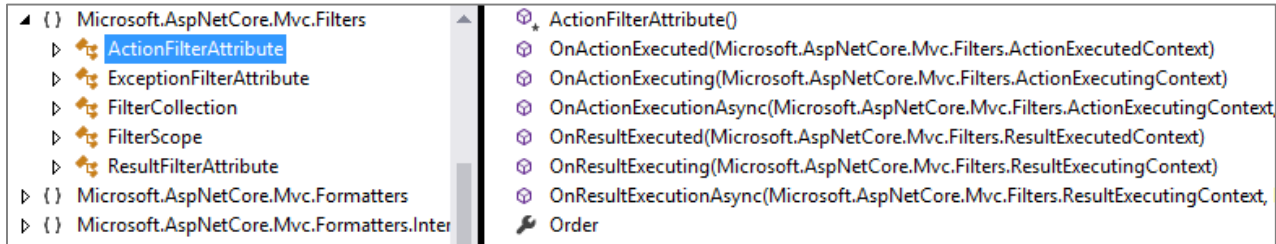
- `OnActionExecuted`

The `OnActionExecuted` method runs after the action method and can see and manipulate the results of the action through the `ActionExecutedContext.Result` property. `ActionExecutedContext.Canceled` will be set to true if the action execution was short-circuited by another filter. `ActionExecutedContext.Exception` will be set to a non-null value if the action or a subsequent action filter threw an exception. Setting `ActionExecutedContext.Exception` to null effectively 'handles' an exception, and `ActionExecutedContext.Result` will then be executed as if it were returned from the action method normally.



# Focus on action filters

## ► Action filters



```
public class CartSessionFilter : ActionFilterAttribute {  
    private Cart _cart;
```

```
    public override void OnActionExecuting(ActionExecutingContext context) {  
        _cart = ReadCartFromSession(context.HttpContext);  
        context.ActionArguments["cart"] = _cart;  
        base.OnActionExecuting(context);  
    }
```

De HttpContext is onderdeel van de ActionExecutingContext

De Cart parameter van de action method krijgt dit argument aangereikt!

```
    public override void OnActionExecuted(ActionExecutedContext context) {  
        WriteCartToSession(_cart, context.HttpContext);  
        base.OnActionExecuted(context);  
    }
```

# Focus on action filters

## ► Action filters

- we moeten de filter registreren als een service in de RegisterServices methode van Startup.cs...

```
services.AddScoped<CartSessionFilter>();
```

- merk op dat je in de constructor van CartSessionFilter gebruik kunt maken van DI
  - we maken hier gebruik van om de BeerRepository te injecteren...

```
public class CartSessionFilter : ActionFilterAttribute {  
    private readonly IBeerRepository _beerRepository;  
    private Cart _cart;
```

```
    public CartSessionFilter(IBEerRepository beerRepository) {  
        _beerRepository = beerRepository;  
    }
```

```
    private Cart ReadCartFromSession(HttpContext context) {  
        Cart cart = context.Session.GetString("cart") == null ?  
            new Cart() : JsonConvert.DeserializeObject<Cart>(context.Session.GetString("cart"));  
        foreach (var l in cart.CartLines)  
            l.Product = _beerRepository.GetBy(l.Product.BeerId);  
        return cart;  
    }
```

# Focus on action filters

- ▶ we kunnen nu de nodige action methods decoreren met het `ServiceFilter` attribuut
  - ipv alle action methods te decoreren kunnen we het attribuut boven de klasse plaatsen...

```
[ServiceFilter(typeof(CartSessionFilter))]  
public class CartController : Controller {  
    ...  
}
```

- het resultaat zijn unit testable action methods, voorbeeld Add

```
[HttpPost]  
public IActionResult Add(Cart cart, int id, int quantity = 1) {  
    try {  
        Beer product = _beerRepository.GetBy(id);  
        if (product != null) {  
            cart.AddLine(product, quantity);  
            TempData["message"] = $"{quantity} x {product.Name} was added to your cart";  
        }  
    }  
    catch {  
        TempData["error"] = "Sorry, something went wrong, the product could not be added to your cart...";  
    }  
    return RedirectToAction("Index", "Store");  
}
```

# De store

- ▶ CartController – de unit testen
  - voorbeeld

```
[Fact]
public void Add_RedirectsToActionIndexInStore() {
    var actionResult = _controller.Add(_cart, 1) as RedirectToActionResult;
    Assert.Equal("Index", actionResult?.ActionName);
    Assert.Equal("Store", actionResult?.ControllerName);
}

[Fact]
public void Add_AddsProductToCart() {
    _beerRepository.Setup(b => b.GetBy(1)).Returns(_context.BavikPils);
    _controller.Add(_cart, 1, 4);
    Assert.Equal(2, _cart.NumberOfItems);
}
```

# MVC advanced



# MVC in depth

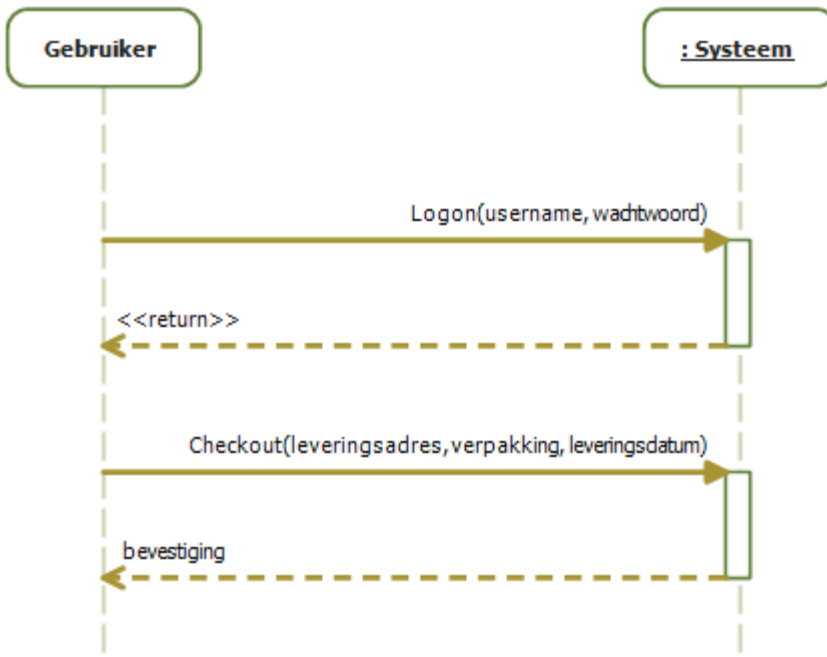
- ▶ Check out Use case – Normaal verloop
  1. Actor kiest om naar de kassa te gaan
  2. Het systeem valideert
  3. Systeem vraagt actor om in te loggen
  4. Actor heeft username en wachtwoord in
  5. Systeem valideert
  6. Systeem vraagt klant de leveringsdetails in te geven (leveringsadres, leveringsdatum (indien gewenst, minstens 3 dagen en niet op zondag), al dan niet kadoverpakking)
  7. Klant vult gegevens in
  8. Systeem valideert
  9. Systeem registreert winkelmandje als order in de database
  10. Systeem ledigt winkelmandje
  11. Systeem bevestigt order

Alternatief :

Actor moet zich eerst registreren

# MVC in depth

## ▶ Controllers (routing)



Worden telkens 2 methodes in de Controller :

- GET (tonen van formulier)
- POST (Posten van formulier data)

# MVC in depth

---

- ▶ Stappenplan uitwerken van de UC
  1. Ontwerp van de UI
  2. Ontwerp van domein en controllers
  3. Domein aanpassen waar nodig – TDD [commit]
  4. DbSet aggregate root/mapping/migratie/initializer [commit]
  5. Extra repositories, helpers, ... [commit]
  6. Controller – TDD & View [commit]



# UC Checkout

## Stappenplan uitwerken van de UC

1. Ontwerp van de UI
2. Ontwerp van domein en controllers
3. Domein aanpassen waar nodig – TDD [commit]
4. DbSet aggregate root/mapping/migratie/initializer [commit]
5. Extra repositories, helpers, ... [commit]
6. Controller - TDD & View [commit]

# UC Checkout - Ontwerp UI

Beerhall

...../Cart/Checkout

the navbar


## Checkout

### Delivery address

Street

Location

### Options

Delivery date  

Gift wrapping ☒

footer

Beerhall

...../Login

the navbar

## Login

Email

Password

☒ Remember me

[Register as new user?](#)

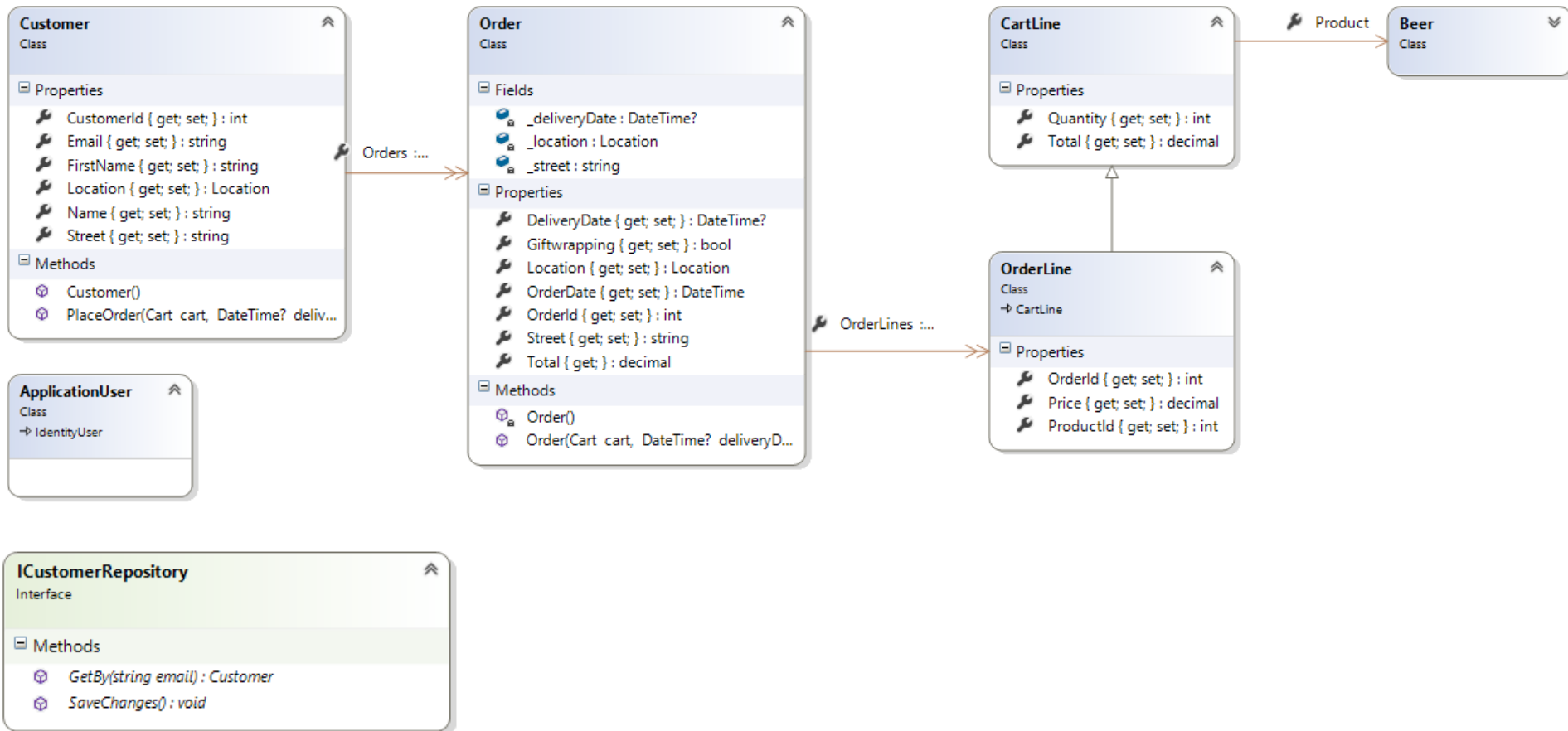
footer

# UC Checkout

## Stappenplan uitwerken van de UC

1. Ontwerp van de UI
2. Ontwerp van domein en controllers
3. Domein aanpassen waar nodig – TDD [commit]
4. DbSet aggregate root/mapping/migratie/initializer [commit]
5. Extra repositories, helpers, ... [commit]
6. Controller - TDD & View [commit]

# UC Checkout - Ontwerp domain



# UC Checkout - Ontwerp Controllers

## CartController

Class

→ Controller

### Methods

- ⊗ Add(Cart cart, int id, [int quantity = 1]) : IActionResult
- ⊗ Checkout(Cart cart) : IActionResult
- ⊗ Checkout(Customer customer, Cart cart, ShippingViewModel shippingVm) : IActionResult
- ⊗ Index(Cart cart) : IActionResult
- ⊗ Remove(Cart cart, int id) : IActionResult

## AccountController

Class

→ Controller

### Methods

- ⊗ Login([string returnUrl = null]) : IActionResult
- ⊗ Login(LoginViewModel model, [string returnUrl = null]) : Task<IActionResult>
- ⊗ LogOff() : Task<IActionResult>
- ⊗ Register([string returnUrl = null]) : IActionResult
- ⊗ Register(RegisterViewModel model, [string returnUrl = null]) : Task<IActionResult>

# UC Checkout

## Stappenplan uitwerken van de UC

1. Ontwerp van de UI
2. Ontwerp van domein en controllers
3. Domein aanpassen waar nodig – TDD [commit]
4. DbSet aggregate root/mapping/migratie/initilializer [commit]
5. Extra repositories, helpers, ... [commit]
6. Controller - TDD & View [commit]

# UC Checkout - Domain

## ▶ TDD van Checkout - **Domain**

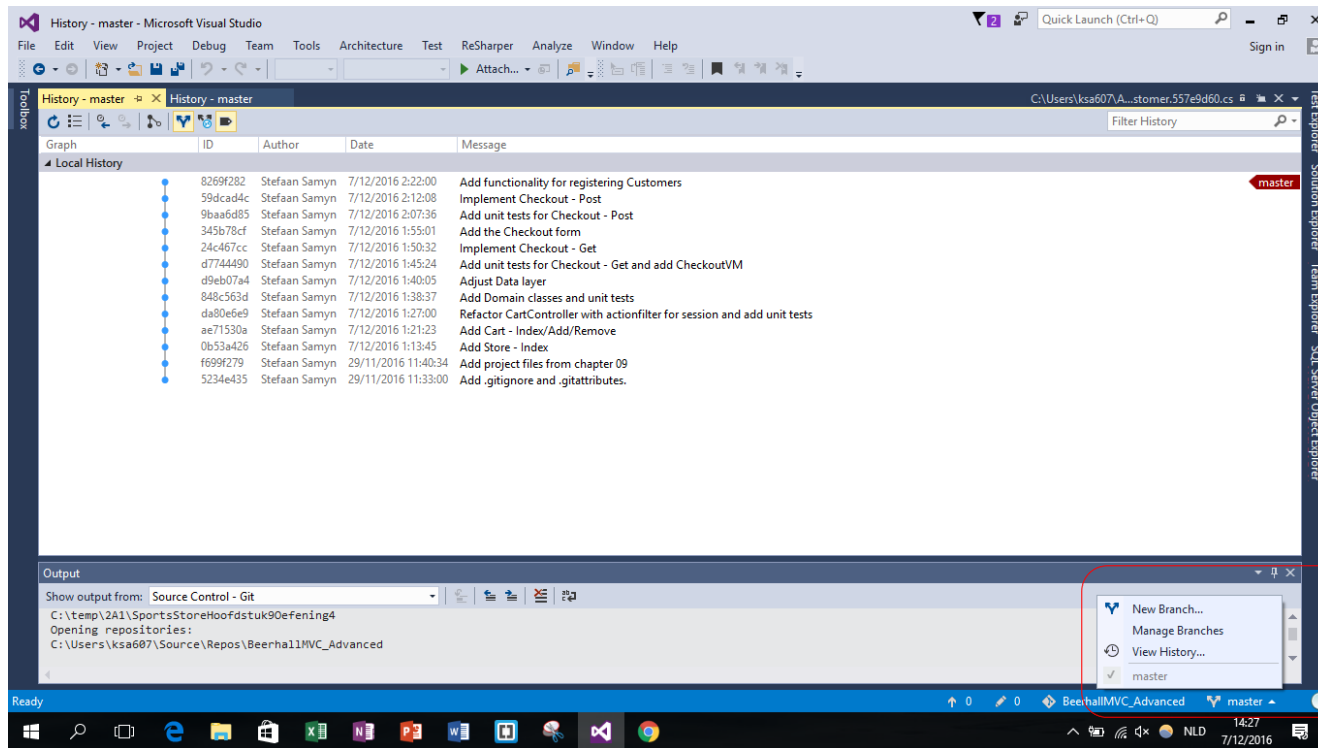
- Nieuwe klassen Customer, ICustomerRepository, Order en Orderline
- Bekijk de unit testen
- Oefening:
  - schrijf **unit testen** voor Customer



# UC Checkout - Domain

## ► Tip

- Om te zien wat er in deze commit is aangepast/toegevoegd :
  - Klik onderaan footer bar op master > View History

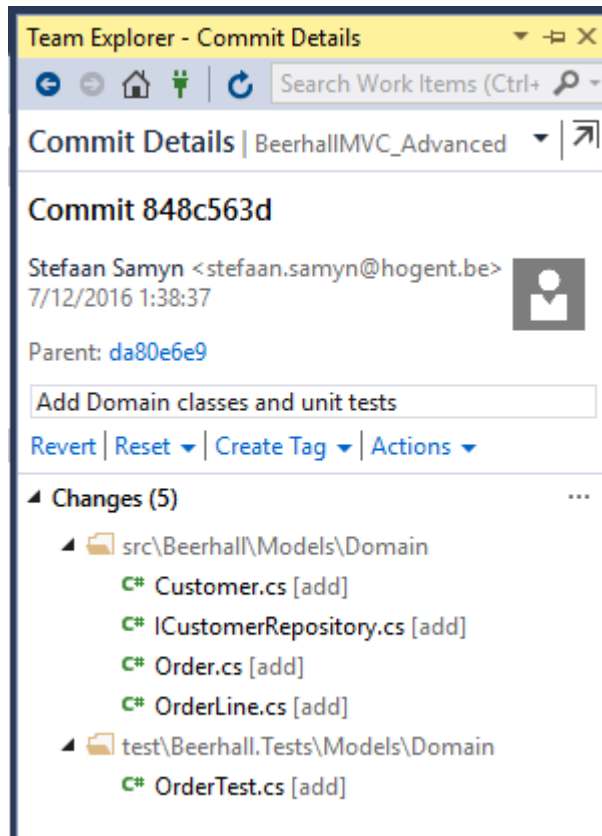




# UC Checkout - Domain

## ► Tip

- Rechtsklik de commit “Add domein classes and unit tests” > View commit details. Dit toont alle gewijzigde, toegevoegde bestanden.



Klikken op een file toont de code igv [add]. Igv [update] kan je ook de wijzigingen zien tov de vorige commits.

# UC Checkout

## Stappenplan uitwerken van de UC

1. Ontwerp van de UI
2. Ontwerp van domein en controllers
3. Domein aanpassen waar nodig – TDD [commit]
4. DbSet aggregate root/mapping/migratie/initializer [commit]
5. Extra repositories, helpers, ... [commit]
6. Controller - TDD & View [commit]

# UC Checkout - Data

## ► TDD van Checkout - Data

- OrderConfiguration, OrderLineConfiguration en CustomerConfiguration werden toegevoegd
- In ApplicationDbContext wordt gezorgd dat deze configurations toegepast worden.
- *herhaal, bekijk en begrijp voor bv. CustomerConfiguration*
  - *Name, Firstname, Email zijn verplicht en maximaal 100 karakters*
  - *Customer - Location is optioneel, wanneer een location wordt verwijderd, wordt de location voor de desbetreffende customers null*
  - *Customer – Order, een order moet verplicht tot een customer behoren, wanneer een customer wordt verwijderd, worden automatisch al zijn orders verwijderd*

# UC Checkout - Data

---

- ▶ TDD van Checkout - **Data**
  - Er werd een DbSet voor onze aggregate root Customer toegevoegd aan de Context
  - De CustomerRepository : ICustomerRepository werd geïmplementeerd

# MVC in depth – Auth

- ▶ TDD van Checkout - **Data**
  - Initializer: toevoegen van een Customer

```
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"));  
  
var customer = new Customer {  
    Email = eMailAddress,  
    FirstName = "Jan",  
    Name = "De man",  
    Location = _dbContext.Locations.SingleOrDefault(l => l.PostalCode == "9700"),  
    Street = "Nederstraat 5"  
};  
  
_dbContext.Customers.Add(customer);  
_dbContext.SaveChanges();
```

# UC Checkout

## Stappenplan uitwerken van de UC

1. Ontwerp van de UI
2. Ontwerp van domein en controllers
3. Domein aanpassen waar nodig – TDD [commit]
4. DbSet aggregate root/mapping/migratie/initializer [commit]
5. Extra repositories, helpers, ... [commit]
6. Controller – TDD & View [commit]

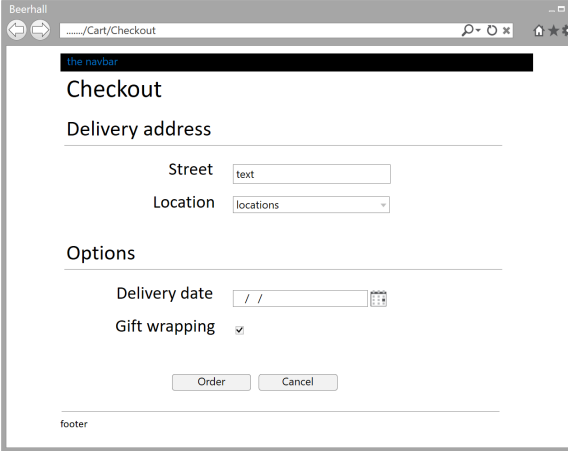
# Checkout - GET



# MVC in depth – ViewModel

- ▶ TDD van Checkout HttpGet - **ViewModels**
  - Er moet een formulier gepresenteerd worden waarop de gebruiker de checkout details kan ingeven

- street
- location (dropdown list)
- gift wrapping
- date of delivery



The screenshot shows a web browser window with the title 'Beerhall' and the address bar displaying '...../Cart/Checkout'. The page has a dark navigation bar at the top. Below the navigation bar, the heading 'Checkout' is displayed. Underneath, there is a section titled 'Delivery address' which contains two input fields: 'Street' (a text box with the placeholder 'text') and 'Location' (a dropdown menu with the placeholder 'locations'). Below this is a section titled 'Options' which contains two items: 'Delivery date' (a text box with two slashes '/' and a calendar icon) and 'Gift wrapping' (a checkbox that is checked). At the bottom of the form, there are two buttons: 'Order' and 'Cancel'. A footer is visible at the very bottom of the page.

- Checkout is enkel toegankelijk voor geauthenticeerde gebruikers die tot de rol customer behoren



# MVC in depth – ViewModels

- ▶ TDD van Checkout HttpGet - **ViewModels**
  - We maken gebruik van een CheckoutViewModel
  - Het viewmodel bevat de properties voor **street, postal code (~location), gift wrapping en date of delivery** én deze keer geven we ook de **SelectList met locations** door via dit viewmodel.

# MVC in depth – ViewModels

- ▶ TDD van Checkout HttpGet - ViewModels
  - We maken gebruik van een CheckoutViewModel

```
namespace Beerhall.Models.ViewModels.CartViewModels {  
    public class CheckOutViewModel {  
        public DateTime? DeliveryDate { get; set; }  
        public string ShippingStreet { get; set; }  
        public string ShippingPostalCode { get; set; }  
        public bool Giftwrapping { get; set; }  
        public SelectList Locations { get; }  
  
        public CheckOutViewModel(IEnumerable<Location> locations, DateTime? deliveryDate = null,  
            bool giftWrapping = false, string shippingStreet=null, string shippingPostalCode = null) {  
            Locations = new SelectList(locations,  
                nameof(Location.PostalCode),  
                nameof(Location.Name),  
                shippingPostalCode);  
            DeliveryDate = deliveryDate;  
            Giftwrapping = giftWrapping;  
            ShippingStreet = shippingStreet;  
            ShippingPostalCode = shippingPostalCode;  
        }  
    }  
}
```

De SelectList als onderdeel van het ViewModel

Optionele parameters: krijgen de opgegeven waarde indien er voor de parameter geen argument wordt voorzien

# MVC in depth – ViewModels

- ▶ TDD van Checkout HttpGet - **ViewModels**
  - We maken gebruik van een CheckoutViewModel
    - HttpGet 😊
      - dit VM bevat alles dat moet aangereikt worden aan de view
      - we hoeven geen extra data via de ViewData door te geven
    - HttpPost 😞
      - dit VM bevat te veel
        - de selectlist met Locations heeft read-only purpose
      - we willen er voor zorgen dat de MVC model binder enkel de inputs van het formulier bindt

# MVC in depth – ViewModels

- ▶ TDD van Checkout HttpGet - **ViewModels**
  - We maken gebruik van een CheckoutViewModel

```
namespace Beerhall.Models.ViewModels.CartViewModels {  
    public class CheckOutViewModel {  
        public SelectList Locations { get; }  
        public ShippingViewModel ShippingViewModel { get; set; }  
        public CheckOutViewModel(IEnumerable<Location> locations, ShippingViewModel shippingViewModel) {  
            Locations = new SelectList(locations,  
                nameof(Location.PostalCode),  
                nameof(Location.Name),  
                shippingViewModel?.PostalCode);  
            ShippingViewModel = shippingViewModel;  
        }  
    }  
  
    public class ShippingViewModel {  
        public DateTime? DeliveryDate { get; set; }  
        public bool Giftwrapping { get; set; }  
        public string Street { get; set; }  
        public string PostalCode { get; set; }  
    }  
}
```

We kunnen de onderdelen die we via de HttpPost willen ontvangen in een apart ViewModel opnemen

Dit willen we via de HttpPost form data binnenkrijgen

# MVC in depth – ViewModels

- ▶ TDD van Checkout HttpGet – **Unit testen method Checkout**
  - De testen

[Fact]

```
public void Checkout_EmptyCart_RedirectsToIndexOfStore()
{
    var actionResult = _controller.Checkout(new Cart()) as RedirectToActionResult;
    Assert.Equal("Index", actionResult?.ActionName);
    Assert.Equal("Store", actionResult?.ControllerName);
}
```

[Fact]

```
public void Checkout_NonEmptyCart_PassesACheckOutViewModelInViewResultModel()
{
    var actionResult = _controller.Checkout(_cart) as ViewResult;
    var model = actionResult?.Model as CheckOutViewModel;
    Assert.Null(model.ShippingViewModel.DeliveryDate);
    Assert.Null(model.ShippingViewModel.PostalCode);
    Assert.Null(model.ShippingViewModel.Street);
    Assert.False(model.ShippingViewModel.Giftwrapping);
    Assert.Equal(3, model.Locations.Count());
}
```

## ▲ CartControllerTest (9)

- ✗ Beerhall.Tests.Controllers.CartControllerTest.Checkout\_EmptyCart\_RedirectsToIndexOfStore
- ✗ Beerhall.Tests.Controllers.CartControllerTest.Checkout\_NonEmptyCart\_PassesACheckOutViewModell..
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Add\_AddsProductToCart
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Add\_RedirectsToActionIndexInStore
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Index\_EmptyCart\_PassesCartToDefaultView
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Index\_NonEmptyCart\_PassesCartToDefaultView
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Index\_NonEmptyCart\_StoresTotalInViewData
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Remove\_RedirectsToActionIndexInDefaultController
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Remove\_RemovesProductFromCart



commit Add unit tests for  
Cart/Checkout - HttpGet

# MVC in depth – ViewModels

## ► TDD van Checkout HttpGet – Implementatie Checkout

```
public IActionResult Checkout(Cart cart) {  
    if (cart.NumberOfItems == 0)  
        return RedirectToAction("Index", "Store");  
    IEnumerable<Location> locations = _locationRepository.GetAll().OrderBy(l => l.Name).ToList();  
    return View(new CheckoutViewModel(locations, new ShippingViewModel()));  
}
```

▲ CartControllerTest (9)

- ✓ Beerhall.Tests.Controllers.CartControllerTest.Add\_AddsProductToCart
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Add\_RedirectsToActionIndexInStore
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Checkout\_EmptyCart\_RedirectsToIndexOfStore
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Checkout\_NonEmptyCart\_PassesACheckoutViewModel
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Index\_EmptyCart\_PassesCartToDefaultView
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Index\_NonEmptyCart\_PassesCartToDefaultView
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Index\_NonEmptyCart\_StoresTotalInViewData
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Remove\_RedirectsToActionIndexInDefaultController
- ✓ Beerhall.Tests.Controllers.CartControllerTest.Remove\_RemovesProductFromCart

# MVC in depth – View

- ▶ TDD van Checkout HttpGet – **Display & Validatie**
  - Voeg display en **validatie annotaties** toe aan het ShippingViewModel



## Checkout

Delivery address

---

Street

Location

Options

---

Delivery date

Gift wrapping ☐

# MVC in depth – View

- ▶ TDD van Checkout HttpGet – **View**
  - Maak gebruik van **scaffolding** voor de Checkout view aan te maken bij het toevoegen van de view

Add View

View name:

Template:

Model class:

Data context class:

Options:

☐ Create as a partial view

☒ Reference script libraries

☒ Use a layout page:

...

(Leave empty if it is set in a Razor \_viewstart file)

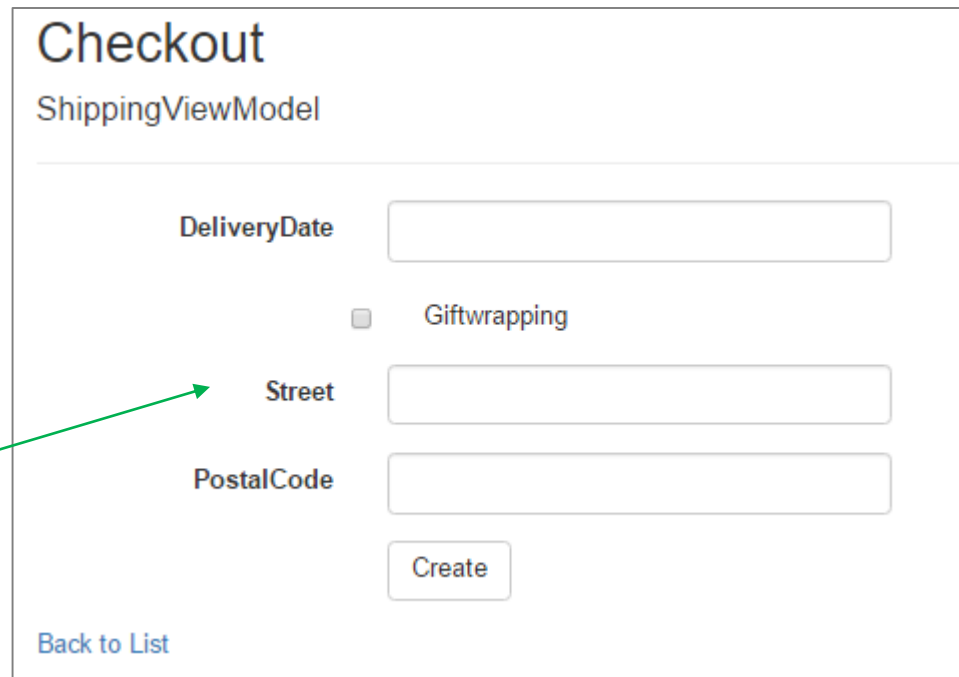
Add Cancel

Kies ShippingViewModel, indien je CheckoutViewModel kiest heb je weinig aan de scaffolding...



# MVC in depth – View

- ▶ TDD van Checkout HttpGet – **View**
  - Maak gebruik van **scaffolding** voor de Checkout view aan te maken (right-mouse-click op folder Cart, kies MVC View)



The screenshot shows a web form titled "Checkout" for the "ShippingViewModel". It contains the following elements:

- DeliveryDate**: A text input field.
- Giftwrapping**: A checkbox.
- Street**: A text input field.
- PostalCode**: A text input field.
- Create**: A button to submit the form.
- Back to List**: A link at the bottom left.

A green arrow points from a text box on the left to the "Street" input field.

*Dit resultaat van de scaffolding kunnen we nu naar onze wensen aanpassen...*

# MVC in depth – View

- ▶ TDD van Checkout HttpGet – **View**
  - Aanpassen van de scaffolded view

```
@model Beerhall.Models.CartViewModels.ShippingViewModel
```

*Dit moeten we wijzigen in CheckoutViewModel*

```
@{  
    ViewData["Title"] = "Checkout";  
}
```

```
<h2>Checkout</h2>
```

*De default method is HttpPost ☺*

```
<h4>ShippingViewModel</h4>
```

```
<hr />
```

```
<div class="row">
```

```
    <div class="col-md-4">
```

```
        <form asp-action="ViewMe">
```

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

```
            <div class="form-group">
```

```
                <label asp-for="DeliveryDate" class="control-label"></label>
```

```
                <input asp-for="DeliveryDate" class="form-control" />
```

```
                <span asp-validation-for="DeliveryDate" class="text-danger"></span>
```

```
            </div>
```

```
            <div class="form-group">
```

```
                <div class="checkbox">
```

```
                    <label>
```

```
                        <input asp-for="Giftwrapping" /> @Html.DisplayNameFor(model => model.Giftwrapping)
```

```
                    </label>
```

```
                </div>
```

```
            </div>
```

*Alle deze properties moeten we nu laten voorafgaan door **@Model.ShippingViewModel**, een slimme find/replace doet dit in 1 keer voor ons...*

# MVC in depth – View

- ▶ TDD van Checkout HttpGet – **View**
  - Aanpassen van de scaffolded view

```
<div class="form-group">
    <label asp-for="PostalCode" class="control-label"></label>
    <input asp-for="PostalCode" class="form-control" />
    <span asp-validation-for="PostalCode" class="text-danger"></span>
</div>
<div class="form-group">
    <input type="submit" value="Create" class="btn btn-default" />
</div>
</form>
</div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

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

Hier voegen we onze dropdownlist in

op een asynchrone manier worden de jQuery validation libraries scripts aangeleverd voor `_Layout`.

`_ValidationScriptsPartial` is een partial view, die je in de folder `Views/Shared` vindt. We behandelen partial views nog verderop...

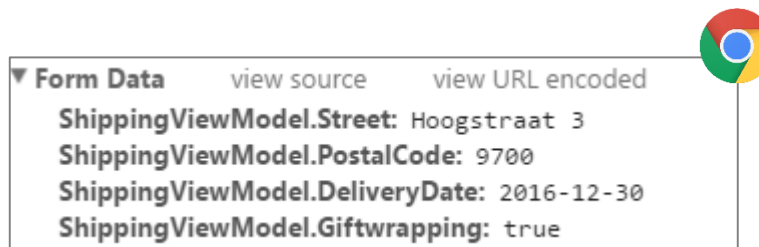
# MVC in depth – View

## ► TDD van Checkout HttpGet – View

- Run de applicatie
  - ga in de Browser naar “bron weergeven”
    - name attributen beginnen nu allemaal met ShippingViewModel. Bvb ShippingViewModel.Street; De Id's met ShippingViewModel\_

```
<input class="form-control" type="text" id="ShippingViewModel_Street" name="ShippingViewModel.Street" value="" />
```

- vul het formulier in, ga naar developer tools > Netwerk, klik op submit



# MVC in depth – Auth

## ► TDD van Checkout HttpGet – Auth

- enkel de ingelogde users die customer zijn kunnen van de checkout gebruik maken
  - we kunnen gebruik maken van de policy die we in **hoofdstuk 9** hebben gemaakt

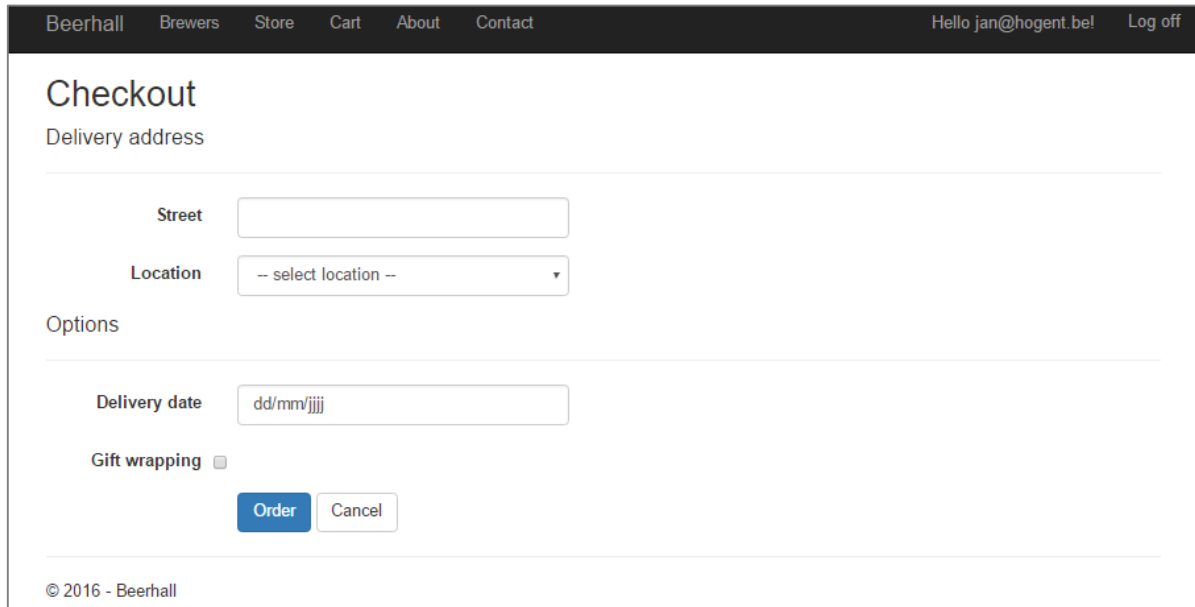
```
[Authorize(Policy = "Customer")]  
public IActionResult Checkout(Cart cart) { ...
```

- indien er geen user is ingelogd gaat de Identity middleware ervoor zorgen dat we automatisch omgeleid worden naar de inlog pagina
- in de initializer hebben we reeds een IdentityUser aangemaakt die customer is (via claim), als deze inlogt kom je terug bij de checkout...

```
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"));
```

# MVC in depth – Auth

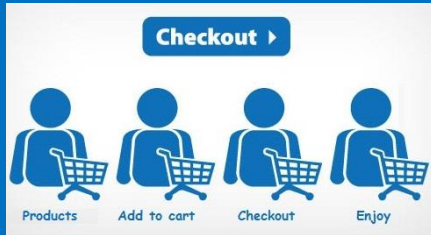
## ► TDD van Checkout HttpGet – Auth



The screenshot shows the Beerhall Checkout page. At the top, there is a navigation bar with links: Beerhall, Brewers, Store, Cart, About, and Contact. On the right side of the navigation bar, it says "Hello jan@hogent.be!" and "Log off". The main heading is "Checkout". Below it, there is a section for "Delivery address" with a "Street" text input field and a "Location" dropdown menu showing "-- select location --". Below this is a section for "Options" with a "Delivery date" text input field showing "dd/mm/yyyy" and a "Gift wrapping" checkbox. At the bottom of the form are two buttons: "Order" (in blue) and "Cancel" (in white). The footer of the page says "© 2016 - Beerhall".

- wanneer het order wordt geplaatst zal onze Customer zijn verantwoordelijkheid nemen en een order creëren en deze toevoegen aan zijn lijst van orders...

# Checkout - POST



# TDD CheckOut - Post

## ► TDD van Checkout HttpPost

- In de Checkout Post zal de Customer moeten beschikbaar zijn
  - verantwoordelijk voor het aanmaken van het nieuwe Order
  - persisteren van zijn Order
  - via de HttpContext kunnen we aan de ingelogde ApplicationUser, en via het email adres kunnen we de Customer ophalen uit de repository

```
HttpContext.User.Identity.Name
```

- ook nu willen we dit niet in de CartController doen want we willen deze unit testbaar houden
- we kunnen dezelfde techniek gebruiken als daarnet voor de Cart
  - een **actionfilter** kan de Customer als argument aanleveren aan de Checkout Post action method...



# TDD CheckOut - Post

- ▶ TDD van Checkout HttpPost
  - de action filter die de Customer aanlevert...

```
namespace Beerhall.Filters {  
    public class CustomerFilter : ActionFilterAttribute {  
        private readonly ICustomerRepository _customerRepository;  
  
        public CustomerFilter(ICustomerRepository customerRepository) {  
            _customerRepository = customerRepository;  
        }  
  
        public override void OnActionExecuting(ActionExecutingContext context) {  
            context.ActionArguments["customer"] = context.HttpContext.User.Identity.IsAuthenticated ?  
            _customerRepository.GetBy(context.HttpContext.User.Identity.Name) : null;  
            base.OnActionExecuting(context);  
        }  
    }  
}
```


*we hoeven enkel  
OnActionExecuting te  
overschrijven, bij het  
beëindigen van de  
action method  
hoeven we met de  
customer niets te  
doen*

```
public void ConfigureServices(IServiceCollection services) {  
    ...  
    services.AddScoped<ICustomerRepository, CustomerRepository>();  
  
    services.AddScoped<CustomerFilter>();  
}
```

*niet vergeten!*

# TDD CheckOut - Post

- ▶ TDD van Checkout HttpPost
  - de signatuur van de Checkout HttpPost action method - 1

```
[HttpPost, Authorize(Policy = "Customer")]  
[ServiceFilter(typeof(CustomerFilter))]   
public IActionResult Checkout(Customer customer,  
                             Cart cart,  
                             [Bind(Prefix = "ShippingViewModel")]ShippingViewModel shippingVm) {
```

# TDD CheckOut - Post

## ► TDD van Checkout HttpPost

- de signatuur van de Checkout HttpPost action method - 2
  - in de Checkout view hebben we gebruik gemaakt van een model van het type CheckoutViewModel
    - in de POST gaan we enkel de properties van ShippingViewModel binden!
  - alle form data die we willen binden heeft een key die begint met ShippingViewModel
  - we kunnen aangeven dat de MVC Model Binder de form data met prefix ShippingViewModel moet binden aan de properties van ShippingViewModel

```
[HttpPost, Authorize(Policy = "Customer")]  
[ServiceFilter(typeof(CustomerFilter))]  
public IActionResult Checkout(Customer customer,  
                             Cart cart,  
                             [Bind(Prefix = "ShippingViewModel")]ShippingViewModel shippingVm) {
```

▼ Form Data    view source    view URL encoded

ShippingViewModel.Street:	Hoogstraat 3
ShippingViewModel.PostalCode:	9700
ShippingViewModel.DeliveryDate:	2016-12-30
ShippingViewModel.Giftwrapping:	true



# TDD CheckOut - Post

- ▶ TDD van Checkout HttpPost – Unit testen
  - zie CartControllerTest
  - voorbeeld

```
[Fact]
public void CheckOut_ModelErrors_PassesCheckOutViewModelInViewResultModel()
{
    _controller.ModelState.AddModelError("any key", "any error");
    var actionResult = _controller.Checkout(_customerJan, _cart, _shippingVm) as ViewResult;
    var model = actionResult.Model as CheckOutViewModel;
    Assert.Equal(_shippingVm, model.ShippingViewModel);
    Assert.Equal(3, model.Locations.Count());
}
```

# TDD CheckOut - Post

- ▶ TDD van Checkout HttpPost – Checkout!
  - de implementatie

```
[HttpPost, Authorize(Policy = "Customer")]
[ServiceFilter(typeof(CustomerFilter))]
public IActionResult Checkout(Customer customer, Cart cart, [Bind(Prefix = "ShippingViewModel")]ShippingViewModel shippingVm) {
    if (ModelState.IsValid) {
        try {
            if (cart.NumberOfItems == 0)
                return RedirectToAction("Index");
            Location location = _locationRepository.GetBy(shippingVm.PostalCode);
            customer.PlaceOrder(cart, shippingVm.DeliveryDate, shippingVm.Giftwrapping, shippingVm.Street,
location);
            _customerRepository.SaveChanges();
            cart.Clear();
            TempData["message"] = "Thank you for your order!";
            return RedirectToAction("Index", "Store");
        }
        catch (Exception ex) {
            ModelState.AddModelError("", ex.Message);
        }
    }
    IEnumerable<Location> locations = _locationRepository.GetAll().OrderBy(l => l.Name);
    return View(new CheckoutViewModel(locations, shippingVm));
}
```



# MVC advanced



# MVC advanced – Auth

## ► Register

- we hebben reeds een Customer in de Initializer aangemaakt
- maar hoe kunnen nieuwe Customers zich **registreren**?
  - we zullen het registreren van onze **Customer** integreren met de registratie van de **IdentityUser**
  - het e-mail adres van beide laten we samenvallen
  - de **Register** functionaliteit van de **Identity – Account** gaan we uitbreiden zodat
    - bij registratie naast de identity gegevens ook de gegevens van de Customer (name, firstname, address, ...) worden gevraagd
      - InputModel (Register.cshtml.cs) en View (Register.cshtml) aanpassen
    - bij creatie van een IdentityUser ook een Customer wordt aangemaakt en gepersisteerd via de CustomerRepository
      - OnPostAsync method (Register.cshtml.cs) aanpassen

# MVC advanced – Auth

- ▶ Register
  - Account - Register - InputModel

```
public class InputModel {  
    [Required]  
    [EmailAddress]  
    [Display(Name = "Email")]  
    public string Email { get; set; }  
  
    ...  
  
    [Required]  
    [StringLength(100)]  
    public string Name { get; set; }  
    [Required]  
    [Display(Name = "First name")]  
    [StringLength(100)]  
    public string FirstName { get; set; }  
    [StringLength(100)]  
    public string Street { get; set; }  
    [Display(Name = "Location")]  
    public string PostalCode { get; set; }  
}
```

## Register

Create a new account.

Email

Name

First name

Street

Location

Password

Confirm password

Register

© 2018 - Beerhall



# MVC advanced – Auth

## ► Register: de view

```
<form asp-route-returnUrl="@ViewData["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="Email"></label>
    <input asp-for="Email" class="form-control" />
    <span asp-validation-for="Email" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Name"></label>
    <input asp-for="Name" class="form-control" />
    <span asp-validation-for="Name" class="text-danger"></span>
  </div>
```

*extra inputs voor Customer*

# MVC advanced – Auth

## ► Register

- Om de vers geregistreerde Customer te persisteren hebben we nood aan een Add methode in ICustomerRepository

```
namespace Beerhall.Models.Domain {  
    public interface ICustomerRepository {  
        Customer GetBy(string email);  
        void Add(Customer customer);  
        void SaveChanges();  
    }  
}
```

# MVC advanced – Auth

- ▶ Register
  - Aanpassingen in Register.cshtml.cs
    - DI van de repositories

```
public class RegisterModel : PageModel {  
    private readonly SignInManager<IdentityUser> _signInManager;  
    private readonly UserManager<IdentityUser> _userManager;  
    private readonly ILogger<RegisterModel> _logger;  
    private readonly IEmailSender _emailSender;  
    private readonly ICustomerRepository _customerRepository;  
    private readonly ILocationRepository _locationRepository;  
  
    public RegisterModel(  
        UserManager<IdentityUser> userManager,  
        SignInManager<IdentityUser> signInManager,  
        ILogger<RegisterModel> logger,  
        IEmailSender emailSender,  
        ICustomerRepository customerRepository,  
        ILocationRepository locationRepository) {  
        _userManager = userManager;  
        _signInManager = signInManager;  
        _logger = logger;  
        _emailSender = emailSender;  
        _customerRepository = customerRepository;  
        _locationRepository = locationRepository;  
    }  
}
```

# MVC advanced – Auth

## ► Register

- Aanpassingen in de Register.cshtml.cs
  - **OnPostAsync** creeërt een customer en persisteert ze via de CustomerRepository

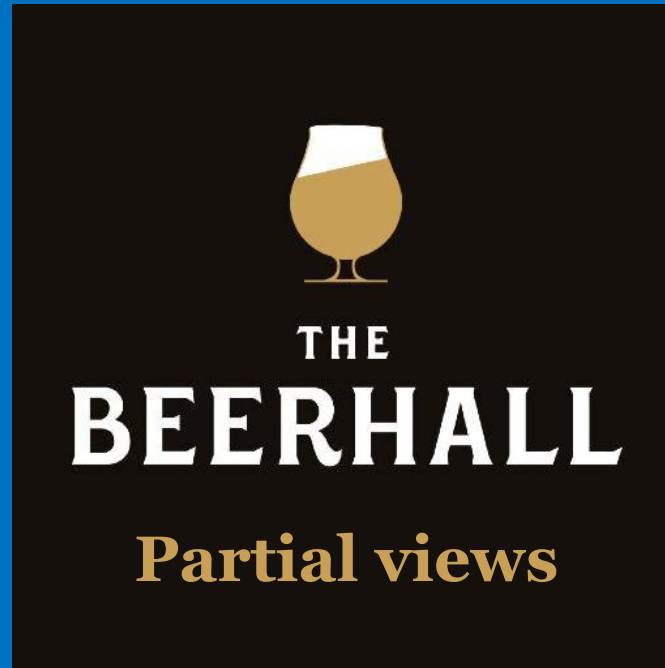
```
public async Task<IActionResult> OnPostAsync(string returnUrl = null) {  
    returnUrl = returnUrl ?? Url.Content("~/");  
    if (ModelState.IsValid) {  
        var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };  
        var result = await _userManager.CreateAsync(user,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

```
        var customer = new Customer {  
            Email = Input.Email,  
            Name = Input.Name,  
            FirstName = Input.FirstName,  
            Street = Input.Street,  
            Location = _locationRepository.GetBy(Input.PostalCode)  
        };  
        _customerRepository.Add(customer);  
        _customerRepository.SaveChanges();  
    }  
}
```

...



# MVC advanced



# MVC advanced – Partial Views

- ▶ Partial views zijn handig wanneer je een onderdeel van een view wil gebruiken in verschillende views
  - ze worden gerendered in een andere view, de parent view
  - ze laten toe grote views op te splitsen in kleinere onderdelen die herbruikbaar zijn
  - ze laten toe om grote, complexe views, gestructureerd uit te werken
    - de verschillende kleinere stukken kunnen apart ontwikkeld worden
      - onafhankelijk van de parent view
    - de parent view wordt overzichtelijker
      - ze bevat de globale structuur met aanroepen om de partial view(s) te renderen

# MVC advanced – Partial Views

- ▶ Een partial view is eveneens een .cshtml bestand
  - in feite is er geen verschil tussen een partial view en een view, beide kunnen als ViewResult van een controller action method geretourneerd worden
    - aan een partial view kan je dus ook een model doorgeven
    - een partial view kan de ViewData van de parent view gebruiken
    - in combinatie met Javascript kunnen controllers als antwoord op Ajax requests partial views retourneren die via javascript op de juiste plaats gerendered worden. Zo kunnen we specifieke onderdelen van een pagina verversen zonder de volledige pagina opnieuw te moeten laden.
  - wanneer een view als een partial view gerendered wordt, wordt de \_ViewStart.cshtml niet uitgevoerd

# MVC advanced – Partial Views

- ▶ Aanroepen van een partial view uit een parent view gebeurt via de tag helper `<partial name = “...”/>`
  - het name-attribuut is verplicht en heeft aan waar de partial view zich bevindt

zie <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/partial-tag-helper?view=aspnetcore-2.1> voor meer info



# MVC advanced – Partial Views

- ▶ Voorbeeld: we willen dat de klant straks ook de inhoud van zijn winkelkarretje op andere plaatsen op onze site kan bekijken. Het tonen van de CartLines van de cart splitsen we af in een partial view `_CartLines`

```
@model IEnumerable<Beerhall.Models.CartViewModels.IndexViewModel>
```

```
@{
```

```
    ViewData["Title"] = "Cart";
```

```
}
```

```
<h2>Your shopping cart</h2>
```

```
@if (Model.Count() != 0)
```

```
{
```

```
    <partial name="_CartLines" />
```

```
    <div align="center" class="actionButtons">
```

```
        <a asp-action="Index" asp-controller="Store" class="btn btn-default">Continue shopping</a>
```

```
        <a asp-action="CheckOut" asp-controller="Cart" class="btn btn-default">Check out</a>
```

```
    </div>
```

```
}
```

```
else
```

```
{
```

```
    <h4>
```

```
        You don't have any products in your shopping cart,
```

```
        <a asp-controller="Store" asp-action="Index">start shopping here...</a>
```

```
    </h4>
```

```
}
```

*de view `Index.cshtml` maakt nu gebruik van de partial view `_CartLines.cshtml`, het model wordt ook doorgegeven aan de partial view*

# MVC advanced – Partial Views

## ► Voorbeeld vervolg: de partial view \_CartLines.cshtml

```
@model IEnumerable<Beerhall.Models.CartViewModels.IndexViewModel>
<table class="table">
  <thead>
    <tr>
      ...
    </tr>
  </thead>
  <tbody>
    @foreach (var line in Model)
    {
      <tr>
        ...
      </tr>
    }
  </tbody>
  <tfoot>
    <tr>
      <td colspan="4" class="text-right">@($"Total: {ViewData["Total"]:N2} €")</td>
      <td></td>
    </tr>
  </tfoot>
</table>
```

*de partial view kan gebruik maken van de  
ViewData van de parent*

# Appendix



Caching

# Appendix - Caching

**Response caching** adds cache-related headers to responses. These headers specify how you want **client, proxy and middleware** to cache responses.

Response caching can reduce the number of requests a client or proxy makes to the web server. Response caching can also reduce the amount of work the web server performs to generate the response.

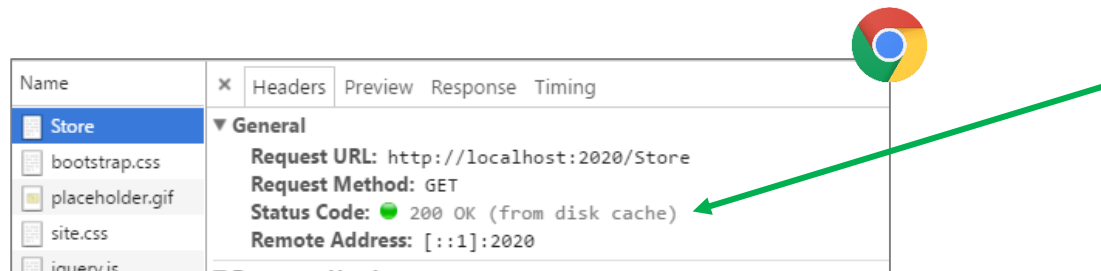
- ▶ Is interessant om op pagina's te plaatsen die steeds naar de database gaan en toch meestal dezelfde data retourneren
- ▶ voorbeeld: StoreController – Index
  - wanneer binnen de minuut na een request voor Index weer een request voor Index komt zal er niet naar de database gegaan worden, maar wordt de gecachte versie van de pagina aangeleverd

```
[ResponseCache(Duration = 60)]  
public ActionResult Index() {  
    return View(_beerRepository.GetAll().OrderBy(b => b.Name).ToList());  
}
```

*de duration is uitgedrukt in seconden*

# Appendix - Caching

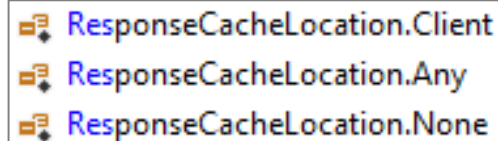
- ▶ Test it yourself...
  - Plaats een breakpoint bij de eerste lijn code in de Index methode
  - Run. De code uitvoering stopt bij dit breakpoint. Run verder.
  - De store wordt getoond in de browser. Druk binnen de minuut weer op F5. De code in de Index wordt niet meer uitgevoerd. De gecachte pagina wordt weergegeven



# Appendix - Caching

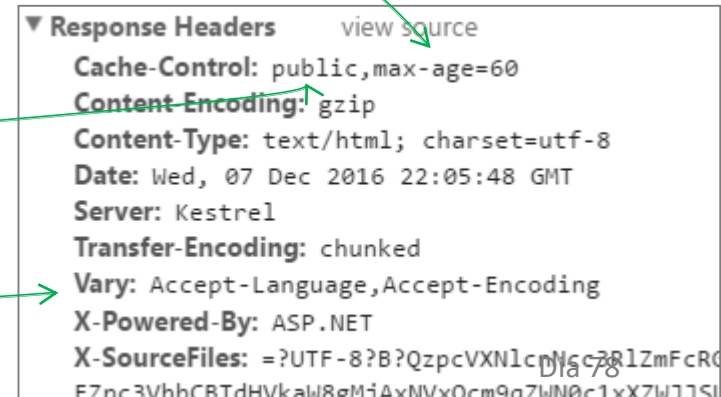
## ► Enkele interessante parameters

- Duration: duur caching in seconden
- Location: bepaalt waar de gecachte gegevens mogen bewaard worden
  - enkel op de client
  - op client en proxy servers
  - niet
- VaryByHeader: als een gespecificeerd onderdeel van de header verandert zullen de gecachte gegevens ongeldig zijn
- voorbeeld



```
ResponseCacheLocation.Client  
ResponseCacheLocation.Any  
ResponseCacheLocation.None
```

```
[ResponseCache(Duration = 60,  
Location = ResponseCacheLocation.Any,  
VaryByHeader = "Accept-Language")]
```



```
▼ Response Headers view source  
Cache-Control: public,max-age=60  
Content-Encoding: gzip  
Content-Type: text/html; charset=utf-8  
Date: Wed, 07 Dec 2016 22:05:48 GMT  
Server: Kestrel  
Transfer-Encoding: chunked  
Vary: Accept-Language,Accept-Encoding  
X-Powered-By: ASP.NET  
X-SourceFiles: =?UTF-8?B?QzpcVXNlc281ZmFRC0  
F7nc3VhhCBTdHVkaW8eMjAxNVx0cm9n7WN0c1xX7w115l
```

voor meer info over caching zie ook <https://docs.microsoft.com/en-us/aspnet/core/performance/caching/response>