

WEB III OEFENINGEN - HOOFDSTUK 8 MVC DEEL 2

Clone https://github.com/WebIII/08exSportsStore_Part2.git

1. Exception handling en TempData in ProductController – Edit HttpPost

In deze starter werd in SportsStore.Tests een **DummyApplicationContext** toegevoegd. Er is eveneens een klasse **ProductControllerTest** (bij aanvang kleuren nog 2 testen geel en 1 test rood). Het **Moq** framework is reeds toegevoegd.

Maak de HttpPost method voor Edit robuuster door gebruik te maken van try..catch blokken

1. Voeg in _Layout code toe om **TempData["message"]** en **TempData["Error"]** te tonen net boven de @renderbody().
2. Een test methode voor Edit-HttpPost kleurt momenteel nog rood. Pas de **HttpPost Edit** methode aan
 - a. Vang exceptions uit domein en data laag op via try..catch
Tip: selecteer een blok code en maak gebruik van VS refactorings om het blok code te wrappen in een try.. catch
 - b. Zorg voor een gepaste TempData["message"] melding wanneer alles succesvol verloopt en voor een gepaste TempData["error"] melding wanneer exceptions geworpen worden
 - c. De test moet nu groen kleuren

Tip: tijdens het runnen kan je bij een edit een overtreding maken tegen de domeinregels om het effect te zien:

- Name is verplicht, minstens 5 karakters lang en maximum 100
- Price is gelegen tussen 1 en 3000.

2. Unit testen & exception handling voor ProductController – Create/Delete HttpPost

1. Pas de Create en Delete op analoge manier aan.
2. Twee unit testen kleuren nog geel. Haal in het Fact-attriboot boven deze methodes de parameter "Skip = ..." weg. Werk de test methodes uit en zorg dat de testen groen kleuren.


3. De Store

We beginnen een nieuw deel en bouwen de winkel. Surfen naar /Store geeft een overzicht van alle producten die online beschikbaar zijn in alfabetische volgorde. Merk op: product LifeJacket wordt niet getoond want het is niet online beschikbaar (~Availability). De afbeelding is placeholder.gif die in de wwwroot/images map werd geplaatst).

/Store/Index


SportsStore Home Products Store Privacy


The Store



Bling-bling King
Gold plated, diamond-studded king


1200,00 €


 Add to cart



Corner flags
Give your playing field that professional touch


34,00 €


 Add to cart



Football
WK colors


25,00 €


 Add to cart



Human chess board
A fun game for the whole extended family!


75,00 €


 Add to cart



Kayak
High quality


170,00 €

 Add to cart



Running shoes
Protective and fashionable

95,00 €

 Add to cart

2

“Add to cart” voegt het product en opgegeven aantal toe aan de Cart. De gebruiker blijft op de Store pagina en krijgt bovenaan een melding te zien.

[/Store/Index](#)

SportsStore Home Products Store Your cart Privacy

Product Corner flags has been added to the cart

The Store



Bling-bling King

Gold plated, diamond-studded king

1200,00 €

Add to cart



Corner flags

Give your playing field that professional touch

34,00 €

Add to cart



Football

WK colors

25,00 €

Add to cart

Als de gebruiker surft naar zijn/haar winkelmandje:

[/Cart/Index](#)

SportsStore Home Products Store Your cart Privacy

Your cart

Quantity	Item	Price	SubTotal	
1	Corner flags	34,00 €	34,00 €	
2	Bling-bling King	1200,00 €	2400,00 €	
Total :			2434,00 €	

Continue shopping

Check out

De gebruiker kan het aantal producten verhogen, verminderen of kan een product verwijderen. De gebruiker kan terug naar de store. Checkout hoeft je nog niet te implementeren.

3

4. TDD van de Index action method in StoreController

1. Maak in de Controllers folder een lege controller genaamd **StoreController** aan.
2. Pas de routing aan zodat de startpagina van de site /Store/Index wordt.
3. Voeg in _Layout een link naar /Store/Index toe in de navbar.
4. Zorg voor constructor injectie van IProductRepository in StoreController.
5. Voeg een methode GetByAvailability toe aan IProductRepository en zorg voor een concrete implementatie in ProductRepository. Gebruik volgende signatuur:
`public IEnumerable<Product> GetByAvailability(IEnumerable<Availability> availabilities)`
6. Ontwikkel een unit test voor de Index action method
 - Maak een unit test klasse **StoreControllerTest** aan in de folder Controllers
 - Implementeer volgende test: Index moet een lijst van Producten die online kunnen aangekocht worden (gebaseerd op de Products die in de repository zitten) doorgeven aan de view via het model.
 - Maak in de constructor een mockProductRepository aan en instantieer de StoreController
 - Train de mock voor de repository methode GetByAvailability. Maak hiervoor gebruik van de property ProductsOnline in de DummyApplicationDbContext. Merk op: het product LifeJacket is enkel in de shop beschikbaar en niet online, het aantal online available producten is dus 10.
 - Schrijf de gepaste unit test voor de Index methode.
 - Run de test. Deze moet falen.
7. Implementeer nu de Index action method zodat de test slaagt.

5. De Store/Index View

1. Voeg de View Index toe.
2. Pas de view aan zodat ze overeenkomt met het voorbeeld. Maak gebruik van responsive web design. Op "small devices" worden de producten onder mekaar geplaatst, op "medium" worden er 2 en "large devices" worden er 3 producten naast mekaar geplaatst.

Voor het toevoegen van een item aan de cart gebruik je onderstaande html.

```
<form asp-controller="Cart" asp-action="Add" asp-route-  
id="@product.ProductId">  
    <div class="form-group">  
        <label class="sr-only" for="quantity">Quantity</label>  
        <input type="number" name="quantity" id="quantity" value="1"  
class="input-medium" />  
    </div>  
    <button type="submit" class="btn btn-outline-primary">  
        <span class="fa fa-shopping-cart pr-1"></span>Add to cart  
    </button>  
</form>
```

Merk op dat we gebruik maken van de font-awesome library om het winkelkarretje op de button te plaatsen (.fa, .fa-shopping-cart)

3. Bekijk het resultaat.

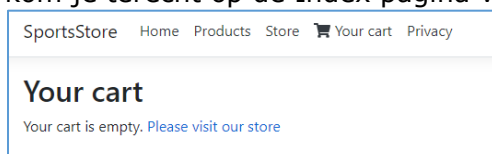
6. De Index methode in de CartController

1. Maak een **CartController** aan en zorg voor constructor injectie van ProductRepository.
2. Maak een **action method Index** aan. De Index methode moet de lijst van CartLines uit de Cart doorgeven aan de View via het model, het totaal van alle cartlines wordt doorgegeven via de ViewData. De Cart wordt niet bijgehouden in de databank en we zullen gebruik maken van een **session** variabele om de Cart op te slaan. Volgende stappen moeten hiervoor ondernomen worden:
 - a. **Configureer** de applicatie om sessions te gebruiken.
 - In Startup – ConfigureServices voeg je de nodige service toe: AddSession()
 - In Startup – Configure voeg je sessions aan de pipeline toe net voor UseEndpoints: UseSession()
 - b. Implementeer een methode **private Cart ReadCartFromSession()**
 - De methode retourneert een nieuwe cart als de session nog leeg is
 - De methode deserialiseert en retourneert de cart opgeslagen in de session indien de session niet leeg is.
 - c. Implementeer een method **private void WriteCartToSession(Cart cart)**
 - Deze methode serialiseert de cart en schrijft ze weg in de session variabele
 - d. Maak gebruik van **Json attributen** in je domeinklassen om de properties en fields te bepalen die zullen worden ge(de)serialiseerd.
 - Decoreer de klassen Cart, CartLine en Product met [JsonObject(MemberSerialization.OptIn)]
 - Cart: zorg dat de _cartlines geserialiseerd worden
 - CartLine: zorg dat Product en Quantity properties geserialiseerd worden
 - Product: zorg dat de Id property geserialiseerd wordt en voorzie de volgende constructor

```
[JsonConstructor]
private Product(int productId) {
    ProductId = productId;
}
```
 - e. Je merkt dat we niet het volledige product serializeren, denk eraan in de methode ReadCartFromSession alle producten uit de repository te halen op basis van het ProductId
 - f. Vervolledig de Index action method.
 - Geef bij een niet lege cart de lijst van CartLines via het model door aan de Index view. Analooog geef je het totaal van de cartlines door via de ViewData.
 - Bij een lege cart zorg je dat de view genaamd EmptyCart gerenderd wordt

7. De Cart views

1. Maak een eenvoudige view aan voor **EmptyCart** (via de link 'Please visit our store' kom je terecht op de Index pagina van de Store):



2. Maak de View **Index** aan volgens het voorbeeld bovenaan dit document.
- Per CartLine maak je een formulier aan. Deze bevat 3 knoppen + en – en x. Bij de <button> tags kan je gebruik maken van het "**formaction**" attribuut om de juiste action method aan te roepen. Bvb voor de + :

```
<button type="submit" formaction="/Cart/Plus/id" class="btn btn-xs" />
```

Id moet je vervangen door het id van het item op de CartLine. Maak gebruik van een glyphicon uit bootstrap voor de afbeelding op de knop.

- De Checkout-knop moet je niet verder implementeren, dit is voor later.

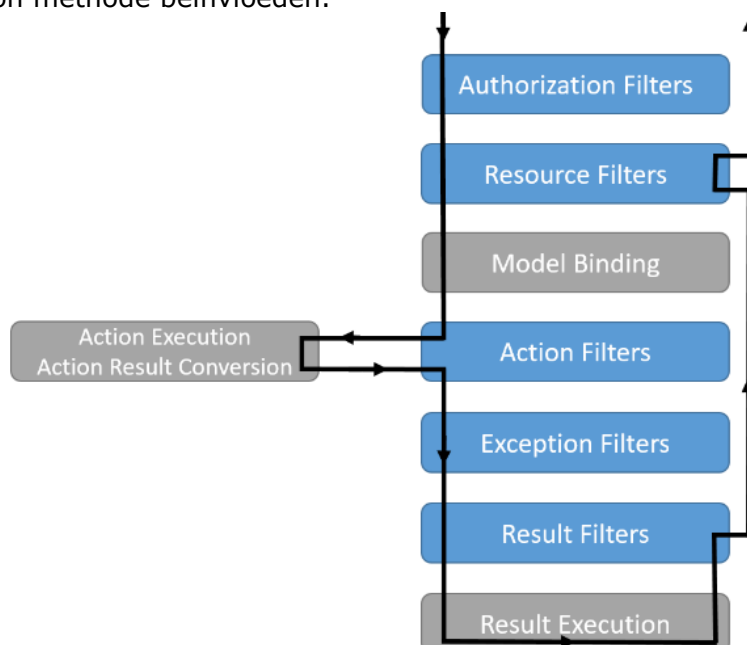
8. Aanmaken van een Filter

De index methode kan je niet unit testen doordat je met een Session werkt. Een Session is gekoppeld aan een Http Request. De Controller biedt toegang tot de Http Request informatie via de property HttpContext. Een Unit test project heeft echter geen toegang tot de Http Request informatie, want daar roep je enkel een actie methode aan en wordt dit niet verstuurd vanuit een browser.

Het probleem zou je kunnen oplossen door een mock aan te maken van `ISession`, zoals in onderstaand voorbeeld

```
var sessionMock = new Mock<ISession>();
sessionMock.Setup(s => s.Get("MyVar")).Returns("Hi");
```

Bovendien dienen we de code om de Cart op te halen en op te slaan in een Session object in elke action methode in deze controller te herhalen. **Filters** laten toe om code die geïmplementeerd wordt in 1 of meerdere action methodes uit de controller te verwijderen en te plaatsen in een "reusable" class. Er bestaan verschillende soorten filters (zie later), maar Action Filters zijn stukjes code die uitgevoerd worden juist voor en/of na de uitvoering van een actie methode en deze kunnen de parameters die doorgegeven worden aan een action methode of het resultaat na uitvoering van een action methode beïnvloeden.













Een Action Filter erft van **ActionFilterAttribute** en kan 2 methodes implementeren:

- a. **OnActionExecuting**: runt juist voor de uitvoering van een action methode. Hier kunnen we het Session object ophalen en de parameter van het type Cart aanmaken.

- b. **OnActionExecuted**: runt na de uitvoering van een action method. Zo kunnen we de Cart terug in het Session object opslaan.

Een Action Filter heeft toegang tot de context via de ActionExecutingContext en ActionExecutedContext. Beiden erven van de ControllerContext en bevatten naast de HttpContext nog enkele andere properties:

	Name	Description
	ActionDescriptor	Gets or sets the action descriptor.
	ActionParameters	Gets or sets the action-method parameters.
	Controller	Gets or sets the controller.(Inherited from ControllerContext.)
	DisplayMode	Gets the display mode.(Inherited from ControllerContext.)
	HttpContext	Gets or sets the HTTP context.(Inherited from ControllerContext.)
	IsChildAction	Gets a value that indicates whether the associated action method is a child action.(Inherited from ControllerContext.)
	ParentActionViewContext	Gets an object that contains the view context information for the parent action method.(Inherited from ControllerContext.)
	RequestContext	Gets or sets the request context.(Inherited from ControllerContext.)
	Result	Gets or sets the result that is returned by the action method.
	RouteData	Gets or sets the URL route data.(Inherited from ControllerContext.)

1. De filter klasse is reeds aangemaakt. Bekijk de code in de klasse **CartSessionFilter** in de Filters folder. Merk op hoe de cart nu als een argument zal kunnen aangeleverd worden aan de action method via een parameter genaamd 'cart':
context.ActionArguments["cart"] = _cart;
2. Een filter dien je te registreren als service in Startup.cs
services.AddScoped<CartSessionFilter>();
3. Filters pas je toe door gebruik te maken van een annotatie boven een action methode of controller (indien van toepassing op alle action methodes in de controller).
[ServiceFilter(typeof(CartSessionFilter))]
public class CartController : Controller
4. Pas de methode Index aan. De signatuur van de Index method wordt nu:
public ActionResult Index(Cart cart){}
5. Verwijder de methodes ReadCartFromSession en WriteCartToSession en alle verwijzingen ernaar uit de Index method.
6. Run je programma en surf eens naar Cart/Index, je krijgt de EmptyCart te zien.

9. Aanmaken unit testen Index methode

1. Maak een unit test klasse **CartControllerTest** aan.
2. Definieer een private field _cart van het type Cart, een private field _controller van het type CartController.
3. De constructor

- a. Instantieer een `Mock<IProductRepository>` en gebruik deze om de `CartController` te instantiëren.
 - b. Instantieer een `DummyApplicationDbContext`. Instantieer een `Cart` en voeg er een `Football` uit de `DummyApplicationDbContext` aan toe.
`_cart = new Cart(); _cart.AddLine(context.Football, 2);`
4. Maak volgende unit testen die zouden moeten slagen...
- a. `public void Index_EmptyCart_ShowsEmptyCartView()`
 - b. `public void Index_NonEmptyCart_PassesCartLinesToViewViaModelAndStoresTotalInViewData ()`

10. TDD van de Add methode

1. Voeg de `HttpPost` action method **Add** toe. De action method heeft 3 parameters: `id` (het id van het product die aan de cart zal worden toegevoegd), `quantity` (het aantal stuks) en de `cart` (wordt via de `Filter` aangereikt).
 Throw in eerste instantie een `NotImplementedException`
2. Schrijf volgende unit test. Je zal de `Mock` nu moeten trainen voor het ophalen van een product op basis van een `Id`.
`public void Add_Successful_RedirectsToActionIndexOfStoreAndAddsProductToCart()`
 Run de testen, ze zullen falen.
3. Implementeer de method `Add`
4. Run de testen. Ze slagen.
5. Voer de web applicatie uit en voeg een product aan de cart toe. Surf naar `Cart/Index` om de `Cart` te bekijken. Je kan in de navbar in `_Layout` een link naar de `Cart` voorzien.

11. TDD van CartController - Remove

1. Analooq aan `Add`
 - a. Declareer de action method en werp `NotImplementedException`
`[HttpPost]`
`public IActionResult Remove(int id, Cart cart)`
 - b. Schrijf de unit testen. Denk eraan: het product dat je wenst te verwijderen moet je uit de repository ophalen: train de mock!
`public void Remove_Successful_RedirectsToIndexAndRemovesProductFromCart()`
 - c. Run de testen. Ze falen.
 - d. Implementeer de `Remove` action method

12. TDD van de Plus methode

1. Analooq: de **action method Plus**
 Extra: Voeg in `Cart` zelf een methode `IncreaseQuantity` toe (met unit testen) en maak gebruik van deze methode in de action method.

13. TDD van de Min methode

1. Analoog: de **action method Min**

Extra: Voeg in Cart zelf een methode DecreaseQuantity toe (met unit testen) en maak gebruik van deze methode in de action method.

14. Gebruik TempData

1. Maak gebruik van TempData om een message aan de gebruiker te tonen na het toevoegen van een item aan de Cart.

Let op: In de CartControllerTest moet je nu de property TempData instantiëren bij de instantiatie van de controller.