Distributed Applications WS2025/26 (Prof. Dr. Cathrin Möller)

Exercise 7 - 04./08.12.2025

Do not hesitate to ask questions or if you want feedback for your coding style.

Please commit and push to GitLab regularly and ensure @*cathrin.moeller* and my two tutors @pinky-jitendra.tinani and @rahul.patil are invited as a reporter.

Learning goals for today

Today we want to **practice creating reusable (and thus configurable) code** using application.properties.

As a good practice, even if a configurable value is not extracted yet, a to be configured property is saved using an internal state attribute in a service. This way we can provide it from the backend (service) and copy it to a model displayed in the frontend (view template). So we avoid diverging duplicate config / hardcoding.

Practical Exercises

1) We want to create **components which are reusable** if they are intended to be used more than once.

Imagine a Webshop needs to deal with prices a lot. E.g. we want to apply discounts on products or carts or calculating a total price for a shopping cart. We need to ensure prices are rounded to cents / 2 digits after the comma and consider currency conversion - e.g. from Euro to Dollar.

In exercise class 6 you should have implemented a @Service **PriceCalculationService** that offers a method to round prices.

We want to make it more reusable.

So we should implement potentially needed functionality:

- **convert currencies** (You can hardcode the conversion rate for now. Create an Enum „Currency“ to offer allowed currencies - implement Euro and Dollar. Round after conversion.) => Create a generic method with 3 input parameters: BigDecimal amount, Currency fromCurrency, Currency toCurrency

- **apply percentage vouchers** (Ensure you are rounding at the end again. Hardcode the available voucher percentage (10) in an attribute of the service. Create a generic method with an input value for the price (BigDecimal) and a parameter for the percentage voucher value.)

Enhance the PriceCalculationService with these 2 methods.

2) Add **buttons to the Shopping Cart screen to apply / remove the 10% voucher**.

Ensure you persist the information on the shoppingCart whether a voucher has been used or not (as a boolean).

Assume only a single voucher usage is allowed at the same time.

Reflect the voucher usage and price savings in the view (to be displayed in HTML).

Only show one button conditionally (either the apply or remove button) using th:if and th:unless.

Use Thymeleaf logic and a pre-populated model that allows you to minimize logic in the template.

Ensure that the model on the shopping cart view is showing only the voucher value that was configured in the PriceCalculationService! Avoid duplicate hardcoding. This ensures, if we change the configuration on the backend side, we reflect it directly in the frontend view - without needing to touch multiple pieces of code - and potentially forgetting one place which would be error prone.

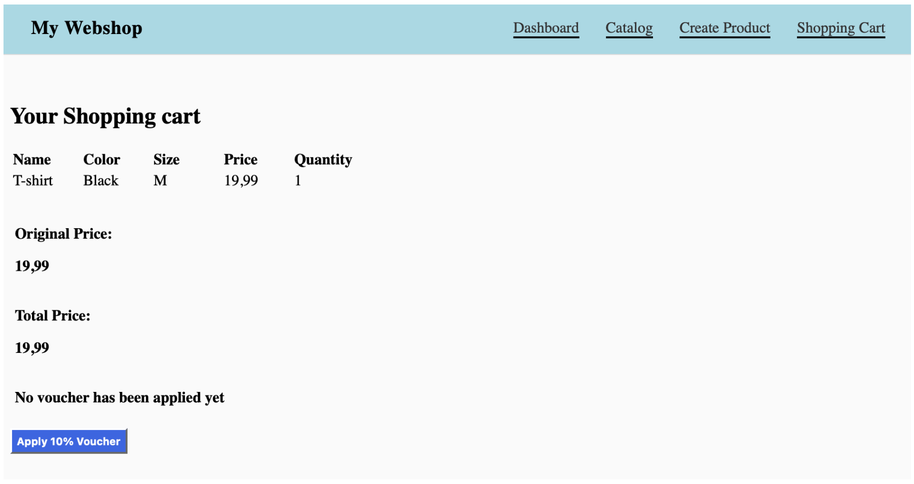
The updated total price and the updated product prices shall be displayed on the cart screen. Also a hint if the voucher has been used.

Hint: For **simplicity** you can use 3 controller endpoints, e.g. /cart (default, no voucher used), ***/cart/voucher*** to apply the voucher and ***/cart/remove-voucher*** to remove vouchers. Or you decide to use request params like /cart?voucher=true or /cart?voucher=false to show the cart with our without voucher application.

The ShoppingCartController might call the AddToCartFacade which calls the ShoppingCartService and the PriceCalculationService.

It might also make sense to add another attribute to your ShoppingCart which stores the original total price for the cart as a copy together with the effective total price. This way your PriceCalculationService does not need a remove voucher method - only your ShoppingCartService.

The result might look similar to this:



Your Shopping cart

Name	Color	Size	Price	Quantity
T-shirt	Black	M	19,99	1

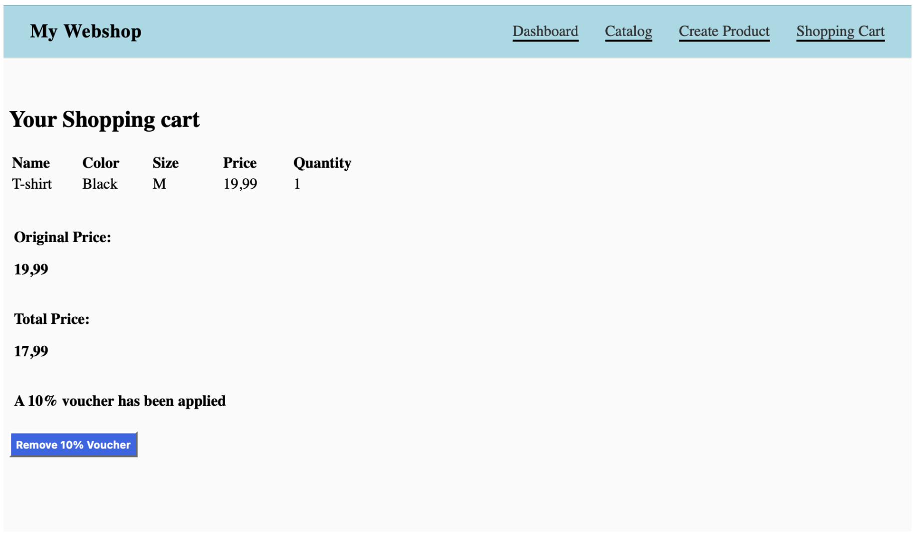
Original Price:

19,99

Total Price:

19,99

No voucher has been applied yet



Your Shopping cart

Name	Color	Size	Price	Quantity
T-shirt	Black	M	19,99	1

Original Price:

19,99

Total Price:

17,99

A 10% voucher has been applied

3) Now let's **configure** our application. In the folder `/resources` there should be a file named ***application.properties*** or ***application.yaml***. Here you can enter key value pairs.

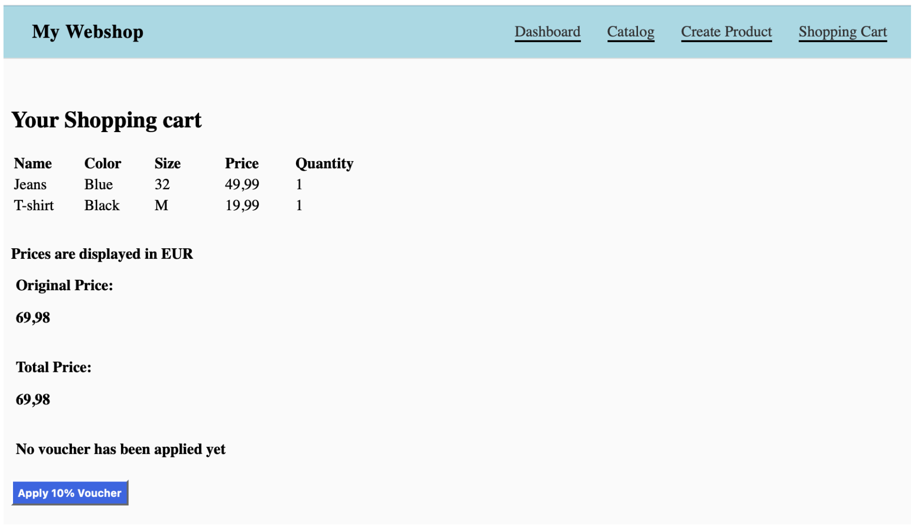
Add an entry like with a key that reflects its intention and set it to the value of your default currency which matches a possible (String) value in your enum => `app.currency.default=„EUR“`

You can now read that configured value in your service using **@Value**. If you read it at constructor „time“ you avoid side-effects from dependency construction (e.g it could happen otherwise that the config is not initialized yet).

```
import org.springframework.beans.factory.annotation.Value;

private String defaultCurrency;

public PriceCalculationService(@Value("${app.currency.default}") String
defaultCurrency) {

    this.defaultCurrency = defaultCurrency;
```

Read the default currency from the PriceCalculationService when creating your ShoppingCart and save the currency of the cart as a new attribute. Display on the shopping cart screen in which currency the prices are displayed.

Just showing the currency in an informational message, it could look like this:



Your Shopping cart

Name	Color	Size	Price	Quantity
Jeans	Blue	32	49,99	1
T-shirt	Black	M	19,99	1

Prices are displayed in EUR

Original Price:

69,98

Total Price:

69,98

No voucher has been applied yet

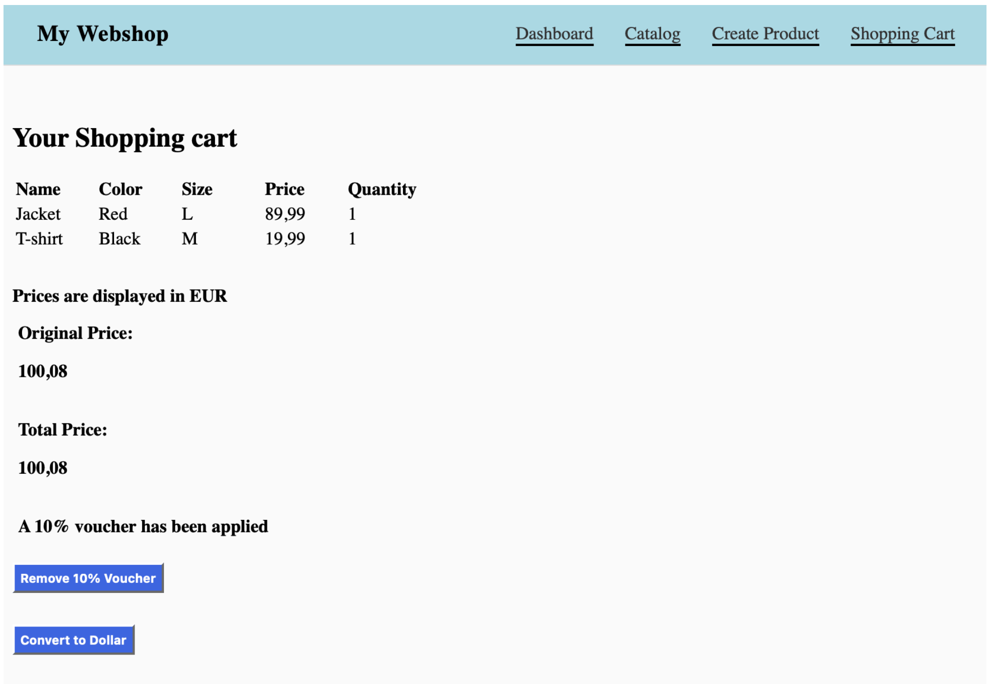
4) Add a **button** to your shopping cart screen to change the currency and **convert prices** from Euro to Dollar (or reverse). Conditionally display a button that asks for the other currency. Conversion must be done using the PriceCalculationService.

For simplicity: It is enough if you convert the total price, not per product. It is allowed to create controller endpoints like ***/cart/dollar*** and ***/cart/eur*** or working with request params **/cart?currency=eur&voucher=true**

(**/cart** will display default currency and no change in voucher application)

If you click multiple times back and forth on the conversion buttons and convert the price multiple times - what do you observe?

The result might look similar to this:



Your Shopping cart

Name	Color	Size	Price	Quantity
Jacket	Red	L	89,99	1
T-shirt	Black	M	19,99	1

Prices are displayed in EUR

Original Price:

100,08

Total Price:

100,08

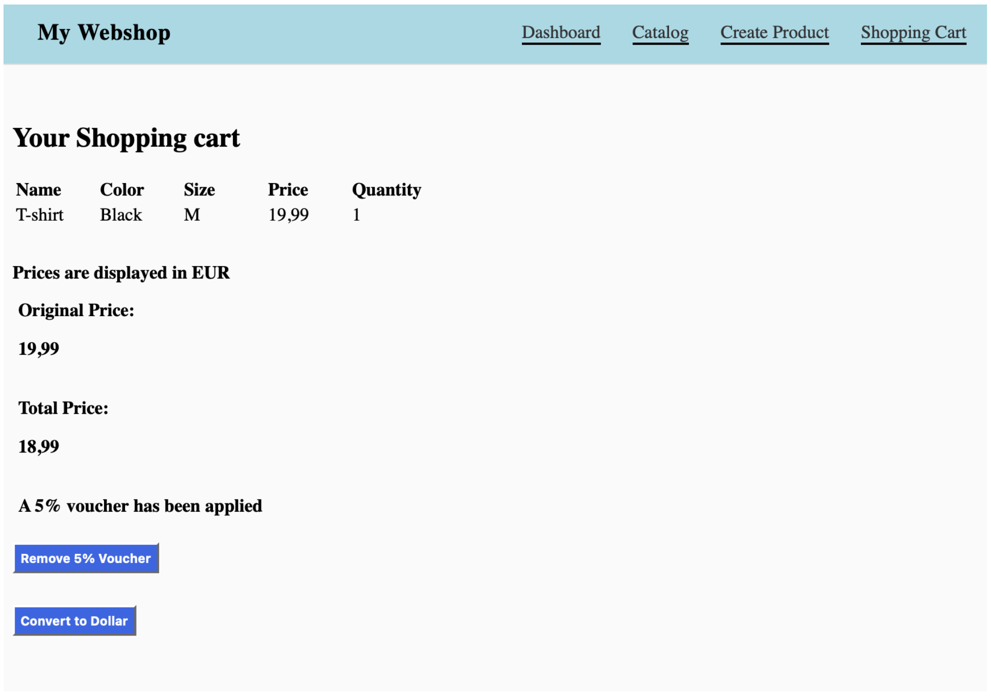
A 10% voucher has been applied

5) Now **replace the hardcoded voucher value with a configurable value** from application.properties as well. This would change the hardcoded 10% (which could stay as a default) for a configurable value

=> Assuming 5%: app.discount.percentage=5

Ensure the calculations are still working and that you do not need to touch your HTML template again to update the voucher value on the button to apply it.

Result might look like this:



My Webshop

Dashboard Catalog Create Product Shopping Cart

Your Shopping cart

Name	Color	Size	Price	Quantity
T-shirt	Black	M	19,99	1

Prices are displayed in EUR

Original Price:

19,99

Total Price:

18,99

A 5% voucher has been applied