



Façades

Exercise

SAP Customer Experience



SAP Customer Experience

THE BEST RUN



TABLE OF CONTENTS

GOAL3

INSTRUCTIONS3

Preparation3

Step 1 • Add a new StockData Property, estimatedRestockDays.....3

Step 2 • Populate the estimatedRestockDays Property3

Step 3 • Add Populator to Converter5

Verify6

Solution7

RECAP7

GOAL

On the product details page, you can already see how much stock is available for a product. If a product is out-of-stock, customers would like to see an estimate of how many days until the product is back in stock again. In this exercise, we are going to add a new attribute to a product's stock data that will allow us to display the estimated days until that product is once again in stock.

INSTRUCTIONS

Preparation

P1 Begin by running this exercise's **setup** Ant target:

- If you haven't done so in the current *Terminal* or *cmd* window, set Ant home variables by navigating to the `MYPATH/workspace/hybris/bin/platform` directory and executing:

```
./setantenv.sh
```

 (on MacOS or Linux) or `setantenv.bat` (on Windows).
- Then navigate to `MYPATH/workspace/TrainingLabTools/exercise_Facades` and execute:

```
ant -f facades_tasks.xml setup
```

The setup task creates an extension (**trainingfacades**) for you and adds it to `localextensions.xml`.

P2 Now import the **trainingfacades** extension into your IDE.

You can either use the SAP Commerce Cloud plugin (under "SAP" menu item) to automatically import the extension by clicking on "Synchronize Projects with yPlatform", or you can import the extension manually (via "Existing Projects into Workspace").

Step 1 • Add a new StockData Property, `estimatedRestockDays`

We want to enable the capability to display on the product details page the estimated days until an out-of-stock product is back in stock. Based on the accelerator's architecture, a DTO is used to transfer the data from the facade to its view. Your first task is to add the **estimatedRestockDays** property to this DTO so that it can be accessed from the product details view.

In `trainingfacades-beans.xml`, add **estimatedRestockDays** as a new property of type **Integer** to the out-of-the-box DTO, `de.hybris.platform.commercefacades.product.data.StockData`. Use your lecture slides as a reference. Run `ant all` to generate the code supporting this newly added property in the autogenerated **StockData** class before moving to the next step.

Step 2 • Populate the `estimatedRestockDays` Property

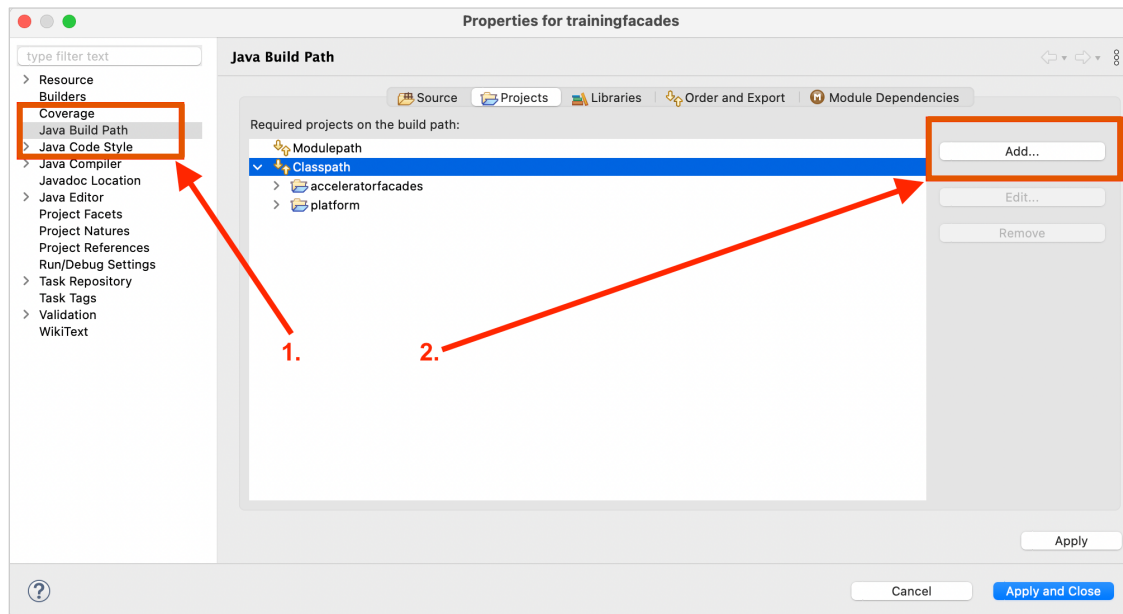
You should now make sure that the **estimatedRestockDays** property on the **StockData** DTO gets populated:

2.1 Add a new package named `my.commerce.trainingfacades.populators` and create the `my.commerce.trainingfacades.populators.DefaultProductRestockEstimatePopulator` class in your extension's `src` folder.

It should implement the `de.hybris.platform.converters.Populator` interface and provide an implementation for its `populate(...)` method. Please implement **only** the signature of the method now, but not the entire method yet, as we will finish the method till step 2.8. The source object passed to the `populate` method is a

ProductModel object (or one of its subtypes). The target object is the *StockData* object (or one of its subtypes) to which we added our **estimatedRestockDays** property in the previous step. Use your theory slides as a reference for what the populator class should look like (or you could take a peek of the solution implementation: *DefaultProductRestockEstimatePopulator.java* under *TrainingLabTools/exercise_Facades/solution*).

- 2.2 As we must reference classes defined outside our extension, our **trainingfacades** project needs to declare a dependency on another extension. Go ahead and add the **acceleratorfacades** extension as a dependency in the `extensioninfo.xml` file in the **trainingfacades** extension folder. Don't forget to update the build path of your IDE's **trainingfacades** project with the same dependency information (under **trainingfacades** | **properties** | **Java Build Path** | **Projects** | **Classpath** | **Add ...**).



- 2.3 You will later need to inject two services and a facade into your populator class:

- *de.hybris.platform.store.services.BaseStoreService*
- *de.hybris.platform.commercefacades.futurestock.FutureStockFacade*
- *de.hybris.platform.commerceservices.stock.CommerceStockService*.

To do that, declare private fields to hold references to these dependent beans and provide public setters for these member variables so that they can later be injected by the Spring framework.

- 2.4 In the populate method, we must first check whether the product argument is out-of-stock. We can do this by getting the product's **StockLevelStatus** (*de.hybris.basecommerce.enums.StockLevelStatus*) using the injected *CommerceStockService.getStockLevelStatusForProductAndBaseStore(...)* method. This method requires a *ProductModel* and **BaseStoreModel** be passed in as arguments. We already have the **ProductModel**. To get the **BaseStoreModel**, you can use the injected **BaseStoreService**'s *getCurrentBaseStore()* method.

Example:

```
commerceStockService.getStockLevelStatusForProductAndBaseStore(productModel,
baseStoreService.getCurrentBaseStore());
```

- 2.5 Once we have obtained the **stockLevelStatus** for the source Product, we can check if it is out-of-stock by comparing it to the static enum value from the **StockLevelStatus** class:

```
if(StockLevelStatus.OUTOFSTOCK.equals(stockLevelStatus))
```

- 2.6 If the product's **StockLevelStatus** is **OUTOFSTOCK**, we now need its restock date. Use the injected *FutureStockFacade.getFutureAvailability()* method, which takes in the **ProductModel**'s code as a String and returns a List of **FutureStockModel** objects. Just get the **FutureStockModel** at the 0 index of the ArrayList and call its *getDate()* method (which returns a java.util.Date object) to obtain an estimated restock date.

Example:

```
futureStockFacade.getFutureAvailability(productModel.getCode()).get(0).getDate();
```

- 2.7 Now that we have the estimated restock date, we can calculate the number of days between the current date and the restock date. You can use any approach you prefer to calculate the number of days between these two dates, as it is not important for this exercise *how* that number is calculated as long as it is the correct number. This number represents our **estimatedRestockDays**.

Example (using the *ChronoUnit.between()* method):

```
LocalDate estRestockLocalDate = LocalDate.ofInstant(  
    estRestockDate.toInstant().truncatedTo(ChronoUnit.DAYS), ZoneId.systemDefault() );  
  
LocalDate todayLocalDate = LocalDate.ofInstant(  
    Instant.now().truncatedTo(ChronoUnit.DAYS), ZoneId.systemDefault() );  
  
int estimatedRestockDays =  
    (int) ChronoUnit.DAYS.between( todayLocalDate, estRestockLocalDate );
```

- 2.8 Remember to assign the result of your computations to the **StockData** parameter object (which is, after all, the whole point of the populator).

Step 3 • Add Populator to Converter

DefaultProductRestockEstimatePopulator is now complete and ready to be used. But it won't be used unless it's associated with a converter. Like the DTO in the previous part, we want to reuse what's already there – in this case, the **stockConverter** bean. Recall how to add a populator to a pre-existing converter, usually defined in another extension, without *directly* modifying the pre-existing bean definitions. (And if you can't recall, refer to the lecture slides!)

- 3.1 Make the necessary changes in `trainingfacades-spring.xml` to create a bean for the *DefaultProductRestockEstimatePopulator* and add it to the appropriate converter. Don't forget to add your service/facade injections to your populator bean.
- 3.2 To make your changes take effect, **run 'ant all' and restart your server**.

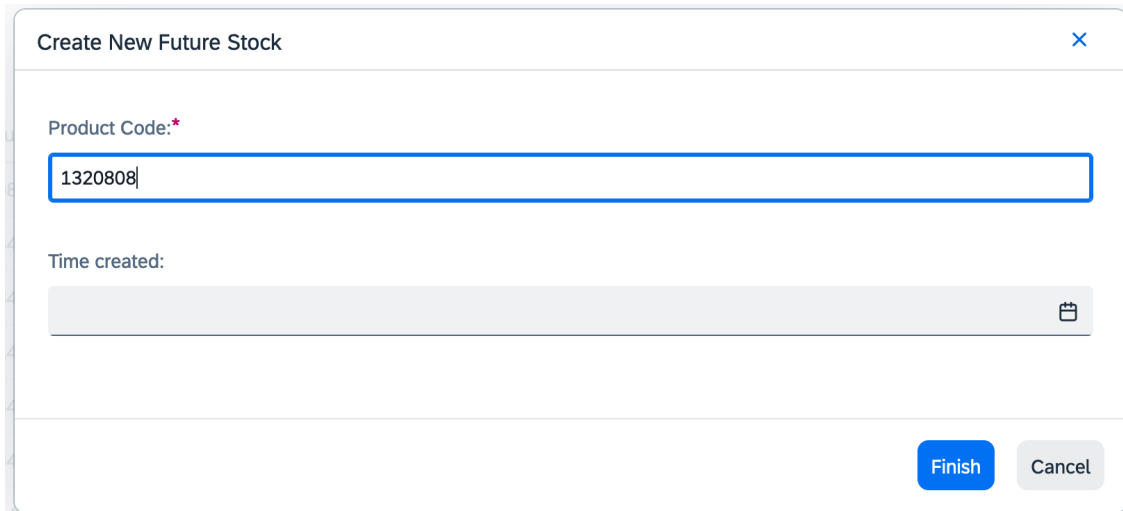
Hint: If you have trouble completing this exercise, please take a look at

[MYPATH/workspace/TrainingLabsTool/exercise_Facades/solution/trainingfacades-spring.xml](#).

Verify

To verify your solution, we need to prepare a simple future stock object for the product with the code “1320808” (we use this product in our `verifyFacadesExercise` script).

Let’s log into Backoffice, navigate to Base Commerce | Future Stock. Please click on the plus icon to create a new Future Stock with the Product Code “1320808” as below, and hit Finish.



Create New Future Stock

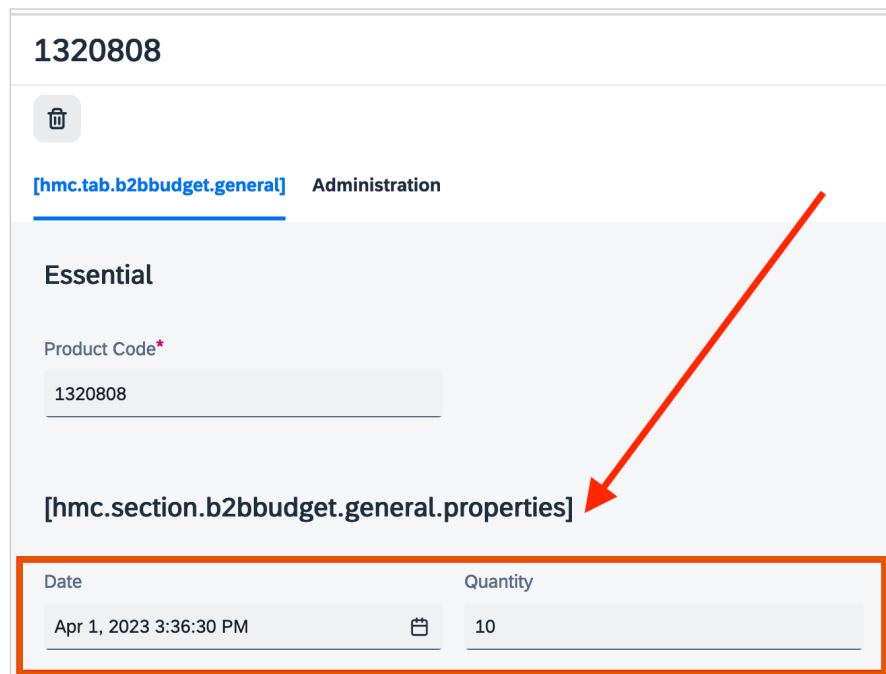
Product Code:*

1320808

Time created:

Finish Cancel

Let’s edit the created Future Stock object directly, and set the future restock date as 2 days from today on (e.g. 01. April 2023, assuming today is 30.March 2023), with an arbitrary quantity value as below. Don’t forget to save the Future Stock object.



1320808

[hmc.tab.b2bbudget.general] Administration

Essential

Product Code*

1320808

[hmc.section.b2bbudget.general.properties]

Date	Quantity
Apr 1, 2023 3:36:30 PM	10

Now you can go to **HAC** and run the script `verifyFacadesExercise`.

Note: It is not part of the exercise to actually add the display of the `estimatedRestockDays` value to the product details page. As it is, the only way to test it on the actual storefront is to include some logging in the populator

and then click on a product that you know has no stock (such as the product with code 1320808). Look for the logging output in your *Terminal* or *cmd* window. See the **solution files** (TrainingLabTools | exercise_Facades | solution | **DefaultProductRestockEstimatePopulator.java** and **project.properties**) for an example of how to use logging.

Solution

If you don't wish to complete this exercise manually, you can install the solution provided:

S1 Navigate to the *Terminal* or *cmd* window where the server is running, and if it is running, stop it by entering `CTRL-C`.

S2 Navigate to `MYPATH/workspace/TrainingLabTools/facades` and execute:

```
ant -f facades_tasks.xml solution
```

Note: if you want to verify the exercise after running the solution, please follow the verification steps above.

RECAP

In this exercise, you learned how to use *façades* and *populators* to pass data from the services layer to the frontend layer.

www.sap.com

© 2023 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See www.sap.com/copyright for additional trademark information and notices.

SAP Customer Experience

