# EGIFT STORE



|  |  |  |  |
| --- | --- | --- | --- |
| Version | Updated By | Date | Comments |
| 1.0 | Sowjanya Gattum  Parmeshwar Bhosle  Tulika Goel  Shreya Karanjekar |  |  |
|  |  |  |  |
|  |  |  |  |

Document Sign-Off

|  |  |  |
| --- | --- | --- |
| Name | Role | Signature/Date |
| Gyaneshwar | Manager |  |
|  |  |  |

**Table of Contents**

[**1.0 Installation Guide....................................................................................**](#_Toc416258348) **4**

[1.1 Purpose.............................................................................................](#_Toc416258349) 4

[1.2 Intended Audience......................................................................................](#_Toc416258350) 4

[1.3 Document Scope.......................................................................................](#_Toc416258351) 4

[1.4 Functional Rules.....................................................................................](#_Toc416258352) 4

1.5 Assumptions/Restrictions..................................................................................... 4

1.6 Introduction.............................................................................................. 6

[**2.0 Customizing eGift Store..................................................................**](#_Toc416258356) **10**

[2.1 Customizing Initialization Process................................................................ ............1](#_Toc416258358)0

[2.2 Setup for variant products:-......................................................................................](#_Toc416258358)21

[2.3 Adding promotions to our products................................................... ......................](#_Toc416258359)24

2.4 Vouchers...................................................................................................................27

[2.5 Implementation for to define new attributes in Customer Registration Page ........](#_Toc416258360)31

2.6 Implementation for Update My Account.................................................................34

2.7 Customize CSCockpit with new Customer attributes......................... .......................36

2.8 Coming Soon Products….............................................................................................39

2.9 Customize PDP to define new attributes....................................................................42

2.10 Customize Product cockpit with new PDP attributes................................................45

2.11 Implementation for Hot Folder..................................................................................47

2.12 Implementation for Pickup in store............................................................................49

2.13 WCMS Cockpit Customization.....................................................................................50

2.14.1 Defining users on HMC assign required permissions……………………………………………..52

2.14.2 Apply/Add restrictions on a component………………………………………………………………..54

2.15 Define own shipping method…………………………………………………………………………………….56

**1.0 Installation Guide**

Purpose

This document is the Technical Design Document which provides information of custom B2C store setup, Variants, promotions and Vouchers

It covers:

* High Level Design
* Application Architecture
* Installation Steps

Intended Audience

This document is aimed at Hybris developers and Technical members who want to implement the custom B2C Store in Hybris.

Document Scope

This document covers the details about the sprint 1 EgiftB2C Store.

Functional Rules

egiftB2C Store is an online store which allows to customer to place the order for their own requirements. The most important marketing message for customers is how your product or service saves their time, money, and resources. Within the hybris Commerce Accelerator, they can easily track or make changes to various aspects of their B2C orders in the self-service area. If needed, they can manage their quotes (check pending quote statuses, approve or reject quotes, and request new quotes), track their order history, and manage their order replenishment and schedules.

Assumptions/Restrictions

This module has been created for version 6.0.

The authenticated customer can visit the store to view the products. They can search the products place the order, replenishment orders. They can request quotes and manage the orders placed. The based on the cost center of the organization threshold order restrictions are applied to get approval from the authorized group or customer.

1.6 Introduction

The Hybris Commerce Suite is a very lightweight and completely Java-based application that runs on many system combinations. For a quick start, it is important to know the basic software and hardware requirements for running the Hybris Commerce Suite.

**SYSTEM REQUIRMENTS:**

**For Demonstration and Development**

These requirements apply to the use of the Hybris software itself, not to 3rd party applications or databases. These requirements assume a typical scenario based on Hybris Platform, Commerce product package, PIM product package, MySQL as database, Eclipse, and embedded Tomcat web server.

The **Minimum** scenario should be sufficient for demonstrating the software in sales and presale presentations. Our tests have shown that for smooth development you need to fulfil the system requirements as listed under **Recommended**.

| **Demonstration/Development** | **Minimum** | **Recommended** |
| --- | --- | --- |
| **CPU** | Dual Core, for example i5 | Quad Core, for example i7 |
| **RAM [2]** | 8 GB | 16 GB |
| **Hard Disk [3]** | 7200 rpm IDE | SSD [2] |
| **Hard Disk Space** | 10 GB | 20 GB |

## Database

## There is no need to install a separate database if you want to try out and demonstrate the Hybris Commerce Suite. We bundled and preconfigured the lightweight HSQLDB database, which typically is sufficient for primary tests.

When you implement a Hybris solution, we highly recommend to use a professional database system such as SAP Hana, Oracle, MySQL or Microsoft SQL Server.

THIRD PARTY SOFTWARE FOR SAP HYBRIS COMMERCE SUITE

JAVA VERSION

Oracle JRE/JDK : 8.0

SAP JVM : 8.1

APPLICATION SERVER

Apache Tomcat : 7.0

Oracle Weblogic : 12.1.3

Databases:

MySQL : 5.5,5.6

HSQLDB (Single Node): 2.2.9 (for development / evaluation only).

**Build Hybris Commerce**

After all the files have been successfully downloaded and unpacked, you must build and adapt Hybris Commerce before use.

As of Release 5.5.0, you can use the installer to build preconfigured set-ups. Please refer to the [Sample Scenarios Installation](https://wiki.hybris.com/display/release5/Sample+Scenarios+Installation) document. Also see the [Documentation Release Notes](https://wiki.hybris.com/display/release5/Documentation+Release+Notes) for version-specific information.

Why Build Hybris Commerce

The reasons why you need to build Hybris Commerce are as follows:

* Hybris Commerce is an extendable complex solution, and it has a very flexible structure. During the build process, all referenced components are integrated.
* Runtime files and configuration files are created, prepared, and validated.
* Some parts of Hybris Commerce are compiled, such as:
  + Hybris [Service Layer](https://wiki.hybris.com/display/release5/ServiceLayer).
  + Other Hybris Commerce components you have implemented.

To build Hybris Commerce, follow the steps listed in the following sections:

1. Set Up Apache Ant.
2. Build Hybris Commerce.

This document guides you through the setup process of the Hybris Commerce Accelerator on your local machine. The system you install includes sample data for the demonstration storefronts.

To facilitate your installation of SAP Hybris Commerce, Hybris introduces the Installer: an automated script that takes care of creating directories, moving files, updating configuration and properties files, and initializing the system. It allows you to easily install the Hybris Commerce flavour, that is, Hybris Commerce with the desired application, such as B2C Accelerator, Telco Accelerator, or Data hub. You install the Hybris Commerce flavour using an installer recipe; an installer recipe includes Hybris Commerce and the specific application. The installer replaces the manual procedure that you used to install and initialize previous versions of Hybris Commerce.To facilitate your installation of SAP Hybris Commerce, Hybris introduces the Installer: an automated script that takes care of creating directories, moving files, updating configuration and properties files, and initializing the system. It allows you to easily install the Hybris Commerce flavor, that is, Hybris Commerce with the desired application, such as B2C Accelerator, Telco Accelerator, or Data hub. You install the Hybris Commerce flavor using an installer recipe; an installer recipe includes Hybris Commerce and the specific application. The installer replaces the manual procedure that you used to install and initialize previous versions of Hybris Commerce.

# **Installing and Initializing SAP Hybris Commerce Using the Installer for custom store setup**

Since **Hybris Commerce Suite 6.0.**, there exist two installation .zip files:

* **Commerce-suite-x.x.x.zip**: Contains all the files to install Hybris Commerce Suite and run it out-of-the-box.
* **Commerce-suite-x.x.x-repository.zip:** This file contains the Hybris Maven dependencies. It is required to extend or generate archetypes of modules that are built on the Core+ library, such as EMS and SBG. You must download and extract the contents of the **commerce-suite-x.x.x-repository. Zip** file into the same directory to which you extract the Hybris **commerce-suite-x.x.xZip** file.  
  Go to the Hybris [Download](https://wiki.hybris.com/display/release5/Download) page to download the latest version.

To install the Commerce Suite and the desired application (flavour):

On the target machine, create the folder where you will extract the Hybris Commerce installation zip files. The directory must meet the following conditions:

* + It must be close to the system root directory (for example **C:\Hybris**. You may download the file directly to the root directory, because the ZIP files already contain directory **Hybris**). Microsoft Windows requires the directory paths to be shorter than 255 characters. Any part of a directory path that exceeds this limitation is truncated. The Hybris Commerce modules installation creates several subdirectories, therefore use short directory paths.
    1. Do not use directory paths containing spaces (such as **C:\Hybris Platform**). Building Hybris Commerce fails if the directory path contains space.
    2. The Hybris Commerce installation zip file to the newly-created installation folder, for example:**C:\Hybris**.
    3. Navigate to the *{HYBRIS\_HOME}*/installer directory

**install.bat -r b2c\_acc (OR) install.bat -r b2c\_acc\_plus**

- To generate config, data, log, roles and temp folders

4. Navigate to platform directory:

**setantenv.bat to set path to the ant tool**

**ant modulegen is the command used to create a set of extensions which are depends from the accelerator to build up our own extensions. Give your own project name, package and template as develop.**

**ant modulegen -Dinput.module=accelerator -Dinput.name=egift -Dinput.package=com.egift -Dinput.template=develop**

**5.Make entry of generated extensions in config/localextension.xml**

**6.To initialize system:**

* platform: **ant** **initialize** command

OR

* platform: if the Hybris server is up (**hybrisserver.bat**), you can select initialization in the platform tab of the Hybris Administration Console (HAC).

7. Start Hybris Server

1. platform**: hybrisserver.bat**

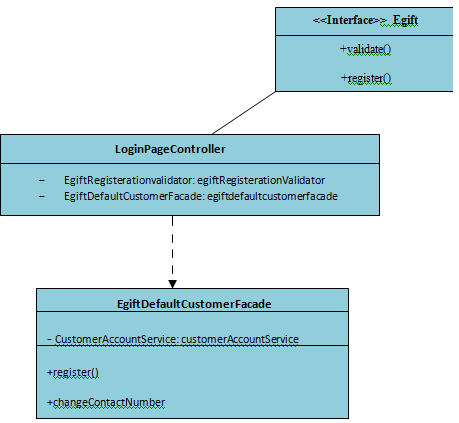
8. Check the default electronics store

https://localhost:9002/egiftstorefront/?site=electronics

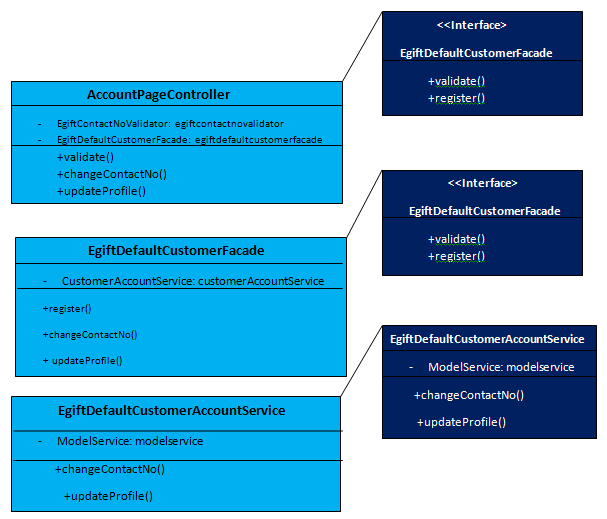
http://localhost:9001/egiftstorefront/?site=electronics

**Application Design Diagrams:-**

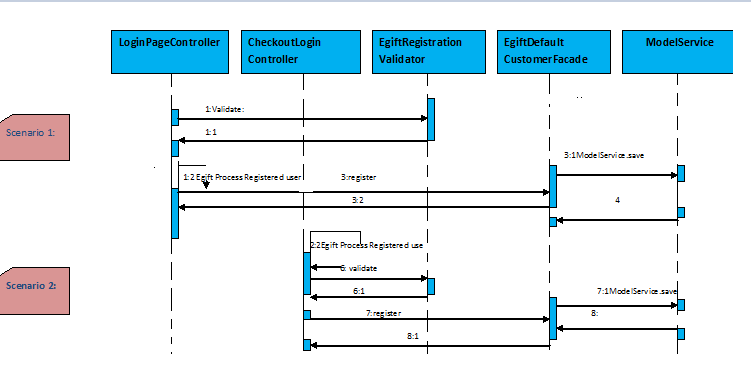
Class Diagram: Registration



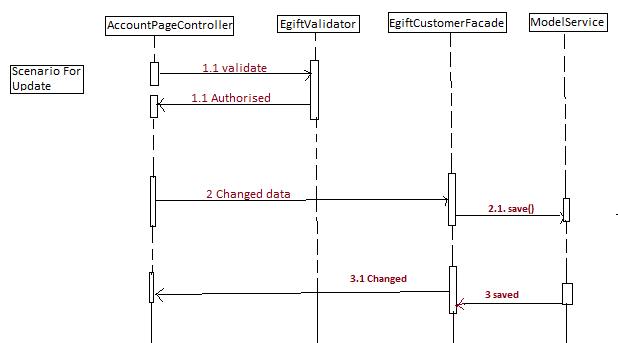
Class Diagram: My Account



**Registration Sequence Diagram:**

****

**Update data Sequence Diagram:**

****

**2.0 Customizing Egift Store :**

**2.1 Customizing initialization process :**

The main objective of these steps are to create data for our custom store. The accelerator provides two dedicated initialization hooks that allow you to import your custom data.

1. Core data hook - in our namespace that is the **CoreSystemSetup** class in egiftcore extension.
2. Sample data hook - in our namespace that is the **InitialDataSystemSetup** class in egiftinitialdata extension.

Next, we need to customize the extension's initialization hooks to setup the egift store data as well.

Both classes are annotated with @SystemSetup and defined as beans in **egiftcore-spring.xml** and **egiftinitialdata-spring.xml**

Data Import across the Accelerator Extensions

Egift Core initialization hook: By design it is meant to initialize the store's basic data: cockpit definitions, email templates, CMS layout, etc.

We have to review the com.egift.core.setup.CoreSystemSetup class try to identify the method for importing the project data. With this class we should analyze which impex’s are used to import the stores basic data and non concrete store data.



**Egift Sample Data initialization hook:**

By design this is meant to be the hook for all the product and concrete cms content initializations. All store-specific data should be defined here: catalogs, Solr configurations, etc. This hook uses import services**CoreDataImportService** and**SampleDataImportService**to import the appropriate data.

You should review the com.egift.initialdata.setup.InitialDataSystemSetup class and try to identify the method of importing project data. We will need to modify this method to import the data.

#### Egift Core Data Import Service

By design this is meant to provide the Egift Sample Data hook an easy way to import all essential core data for a new store. This service is used to import basic structure of a store (i.e.: catalog definition, email templates, site definition)

Review the de.Hybris.platform.commerceservices.dataimport.impl.CoreDataImportService class and try to identify the methods of importing data structure. Note how the store's data is imported: catalogs, solr configuration, cms content, etc. We need to do the same for our Egift shop.

#### Sample Data Import Service

By design this is meant to provide the Egift Sample Data hook an easy way to import all the store-specific data and fill the structure defined by Egift Core Data Import Service with concrete data (i.e.: product data, images)

We should review de.Hybris.platform.commerceservices.dataimport.impl.SampleDataImportService class and try to identify the methods of importing project data. Note how the store's data is imported: products, images, cms content, etc.  
We need to do the same for our Egift shop.

Modify the **InitialDataSystemSetup** to include our products, cms-content during project data initialization.



## View impex files for the Egiftcore extension

To learn more about which data you should import, you should review the createEssentialData() and createProjectData() methods of the CoreSystemSetup class. The impex files in the egiftcore extension define the essential data needed for the cms content and email content. Look through the impex’s and the methods which load them in CoreSystemSetup.

**Review impex files for the egiftinitialdata extension**

The basic Store data is loaded in the **egiftinitialdata/resources/egiftinitialdata/import/coredata** directory and loaded via the CoreDataImportService class.

To learn more about which data you should import, you should review the importAllData () method of the AbstractDataImportService as well as the overridden importCommonData (), importProductCatalog (), importContentCatalog (), importStore () and importSolrIndex () methods of the CoreDataImportService class. Recall that we set these two services to be called in the createProjectData() of InitialDataSystemSetup class in an earlier step.

For creating **product catalog** related data we need a resource structure containing following file’s :

* **egiftinitialdata/resources/egiftinitialdata/import/coredata/productCatalogs/egiftProductCatalog/catalog.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/productCatalogs/egiftProductCatalog/catalog\_en.impex**

For creation of a **content catalog** related data we need a resource structure containing following files :

* **egiftinitialdata/resources/egiftinitialdata/import/coredata/contentCatalogs/egiftContentCatalog/catalog.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/contentCatalogs/egiftContentCatalog/catalog\_en.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/contentCatalogs/egiftContentCatalog/cms-content.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/contentCatalogs/EgiftContentCatalog/cms-content\_en.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/contentCatalogs/egiftContentCatalog/email-content.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/contentCatalogs/egiftContentCatalog/email-content\_en.impex**

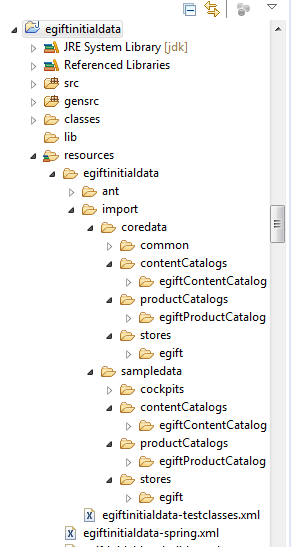
For creation of**store** related data we need a resource structure containing following files :

* **egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/store.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/store\_en.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/site.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/site\_en.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/solr.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/solr\_en.impex**
* **egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/solrtrigger.impex**

## Rename the project folders

Thankfully, starting with the 5.1 accelerator, we don't have to create the above files by hand; all we have to do is rename the catalogName*and* storeName folders in egiftinitialdata/resources/egiftinitialdata/import/coredatato the catalog and store name we want to use.

For example, Refer to the following screenshot:



Do the same for

**egiftinitialdata/resources/egiftinitialdata/import/sampledata**

**site.impex:-**

Adds basic site information to the configuration of the Egift shop - again you need to substitute several macros in the same way as in the above **store.impex** file.

**Egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/site.impex**

**Egiftinitialdata/resources/egiftinitialdata/import/coredata/stores/egift/site\_en.impex**

# Language

$lang=en

# Create CMS Site

UPDATE CMSSite;uid[unique=true];name[lang=$lang];locale[lang=$lang]

;egift;"egift Site"; en\_US

Update the basic solr configuration, including information for ranges, facets, index names, etc...

**Note:** In the version of Hybris 6. the solr server configuration should be in standalone mode.

**egiftinitialdata\resources\egiftinitialdata\import\coredata\stores\egift\solr.impex**

Add the following lines of code in **solr.impex** file.

# Create the solr server configuration

INSERT\_UPDATE SolrServerConfig;name[unique=true];mode(code);embeddedMaster

;$serverConfigName;standalone;false;

INSERT\_UPDATE SolrEndpointUrl;solrServerConfig(name)[unique=true];url[unique=true];master[unique=true,default=false]

;$serverConfigName;http://localhost:8983/solr;true

Update solr\_en, solrtrigger, site, site\_en impex’s as well.

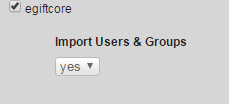
Follow the same with the content catalog data impex’s and product data impex’s in the

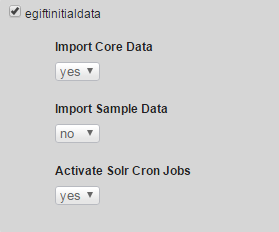
Egiftinitialdata extension.

**CatalogVersion synchronization jobs**

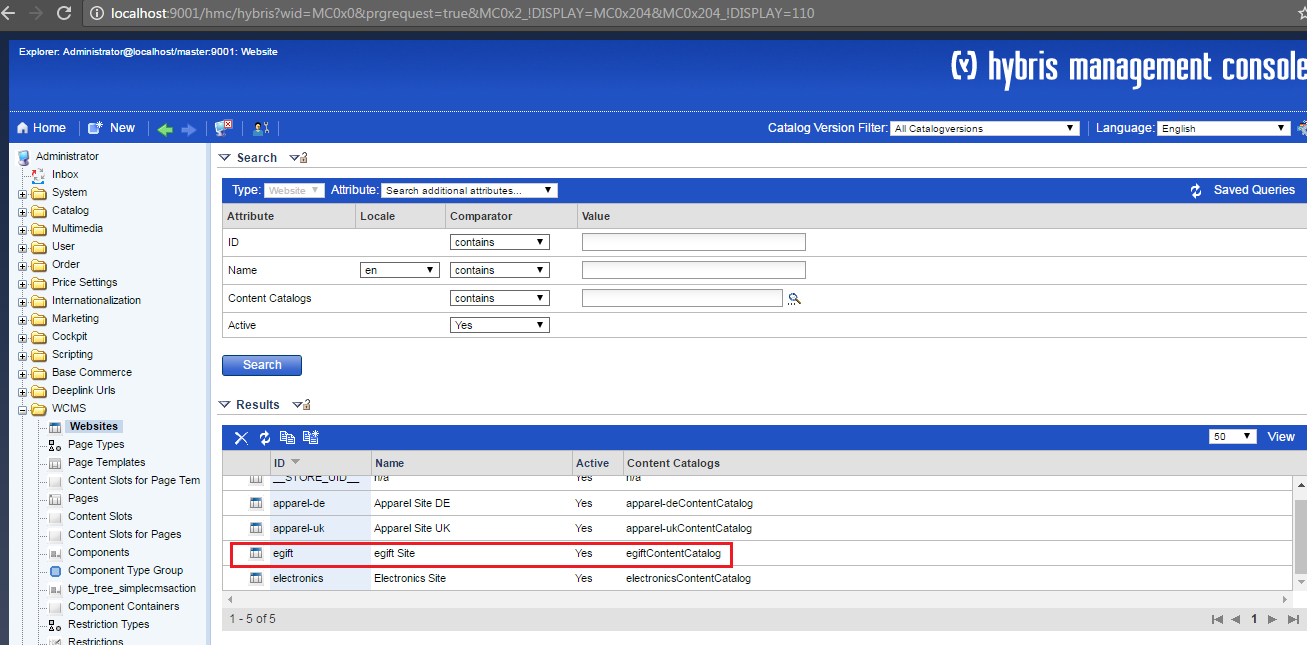
Notice that the **createProjectData ()** method includes code that creates Cronjob’s to synchronize catalog versions. These jobs target our new catalog version’s (content and product) and can be run at any time to synchronize CMS content and products from their staged to online versions.

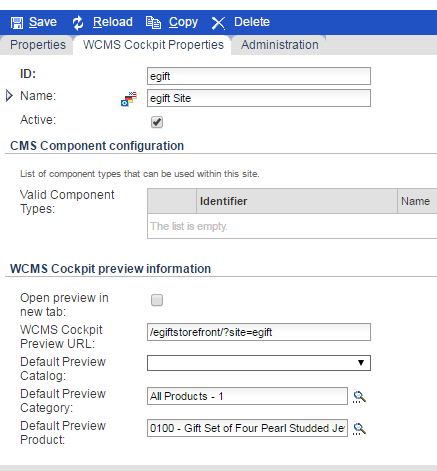
After completing the above modifications execute **ant all** command and **update the project data** with the extension **egiftcore and egiftinitialdata** but allow only **import core data and active solr** Cronjob’s active in egiftinitialdata extension as follows.





Once you have imported the data, you can go to the [hmc](http://localhost:9001/hmc/hybris)and look in the Base Commerce section that we have a new store called **'egift'** and it has a website assigned with the same name.





For Reference purpose all the impex files under core data have been given below**:**

****

**Create products in Egift store:**

Create the product categories in the **categories.impex**

The localization is done in the **categories\_en impex**

Import available products to our store with variants using the products .impex, product\_en.impex and media through product-media. Impex and stock levels in product-stocklevels.impex and prices with product-prices.impex.

**Activating Solr index for the product content.**

Accelerator indexes categories names in a way that it requires additional dedicated spring beans for resolving category names, codes per each configured store.

Add the following beans definitions in **egiftcore-spring.xml**

<bean id=*"egiftCategorySource"* parent=*"abstractCategorySource"*>

<property name=*"rootCategory"* value=*"1"*/>

</bean>

<bean id=*"egiftBrandCategorySource"* parent=*"abstractCategorySource"*>

<property name=*"rootCategory"* value=*"brands"*/>

</bean>

<bean id=*"egiftCategoryCodeValueProvider"* parent=*"abstractCategoryCodeValueProvider"*>

<property name=*"categorySource"* ref=*"egiftCategorySource"*/>

</bean>

<bean id=*"egiftBrandCategoryCodeValueProvider"* parent=*"abstractCategoryCodeValueProvider"*>

<property name=*"categorySource"* ref=*"egiftBrandCategorySource"*/>

</bean>

<bean id=*"egiftCategoryNameValueProvider"* parent=*"abstractCategoryNameValueProvider"*>

<property name=*"categorySource"* ref=*"egiftCategorySource"*/>

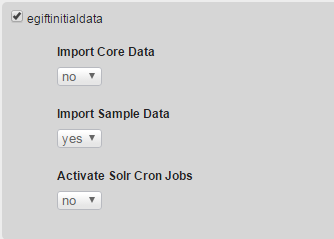
</bean>

<bean id=*"egiftBrandCategoryNameValueProvider"* parent=*"abstractCategoryNameValueProvider"*>

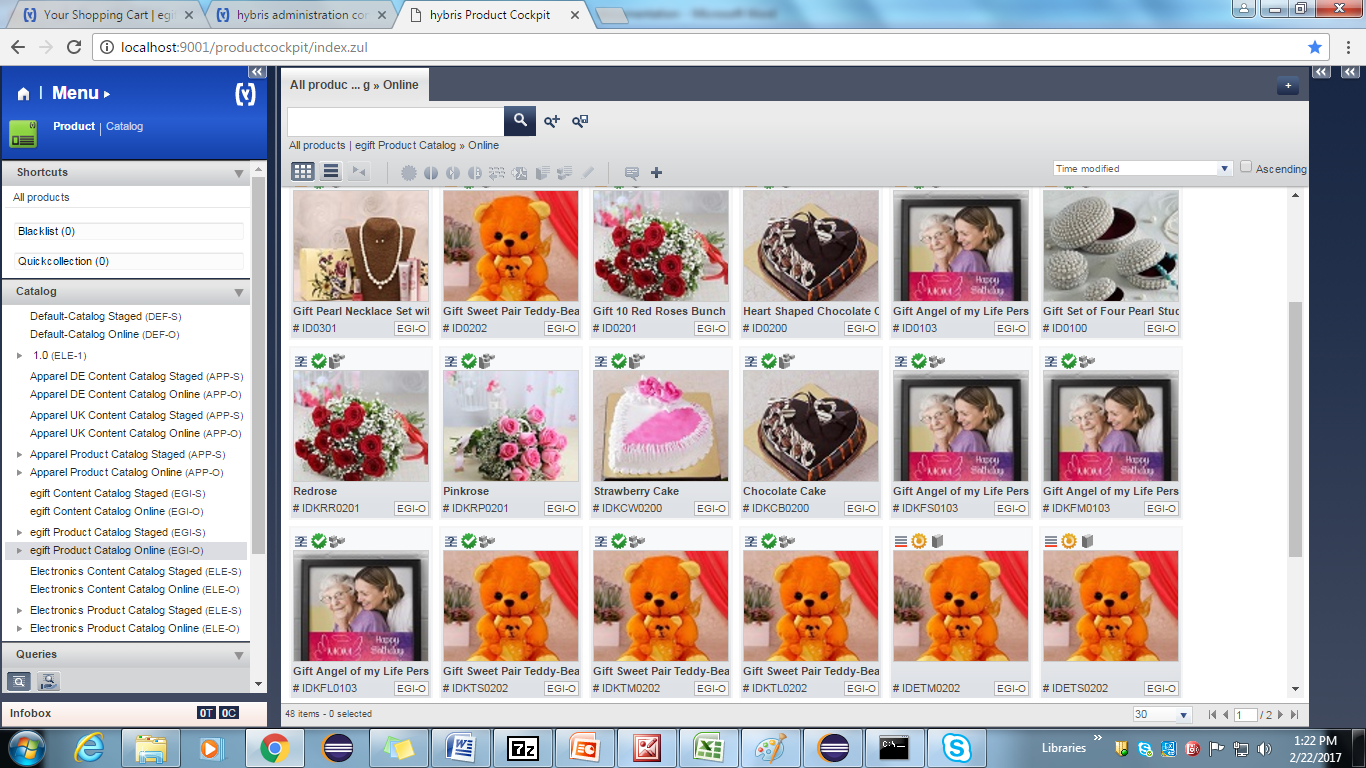
<property name=*"categorySource"* ref=*"egiftBrandCategorySource"*/>

</bean>

After this steps update project data with **egiftinitialdata**

****

After this check your product cockpit it should be updated with your products.



**Create Site CMS Content using Impex:**

Modify the cms content by changing the catalog name. Create the navigation links and navigation nodes to our store. Add the banners to the home page as follows

**Cms-content.impex**

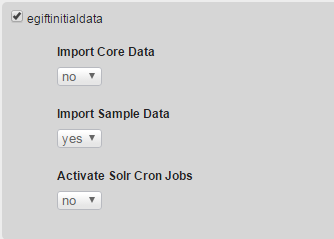
Provide the path to the site logo component to get the media from the provided images folder in the **contentCatalog/egiftContentCatalog/images/**

Add cms link components to create the navigation to products based on the categories.

Add product carousel to add best selling products into that and banner component to the any of the section.

**Cms-content\_en.impex:-**

Now update your project data with the following settings:



After successful update restart server and navigate to url home page

[https://localhost:9002/egiftstorefront/?site=egift](https://localhost:9002/burberrystorefront/?site=burberry)

http://localhost:9001/egiftstorefront/?site=egift



For Reference purpose all the impex files under sample data have been given below**:**



**2.2 Setup for variant products:-**

1. In the **egiftcore-item.xml** add egift **typegroup** and declare **EgiftSizeVariantProduct and EgiftStylerVariantProduct** type which extends **VariantProduct** with attribute style.

<itemtype code=*"EgiftSizeVariantProduct"* extends=*"VariantProduct"*

autocreate=*"true"* generate=*"true"*

jaloclass=*"com.egift.core.jalo.EgiftSizeVariantProduct"*>

<attributes>

<attribute qualifier=*"giftsize"* type=*"localized:java.lang.String"* metatype=*"VariantAttributeDescriptor"*>

<description>Size of gifts</description>

<modifiers />

<persistence type=*"property"* />

</attribute>

</attributes>

</itemtype>

<itemtype code=*"EgiftStylerVariantProduct"* extends=*"VariantProduct"*

autocreate=*"true"* generate=*"true"*

jaloclass=*"com.egift.core.jalo.EgiftStylerVariantProduct"*>

<attributes>

<attribute qualifier=*"giftcolor"* type=*"localized:java.lang.String"* metatype=*"VariantAttributeDescriptor"*>

<description>Color of gifts</description>

<modifiers />

<persistence type=*"property"* />

</attribute>

</attributes>

</itemtype>

2. Now stop the server & Run the command in Hybris platform path: **ant all**, Generates models, DAO, DTO and Jalo classes are created for above specified item types.

3. Refresh the workspace and start the server using command **hybrisserver.bat** in Hybris platform path.

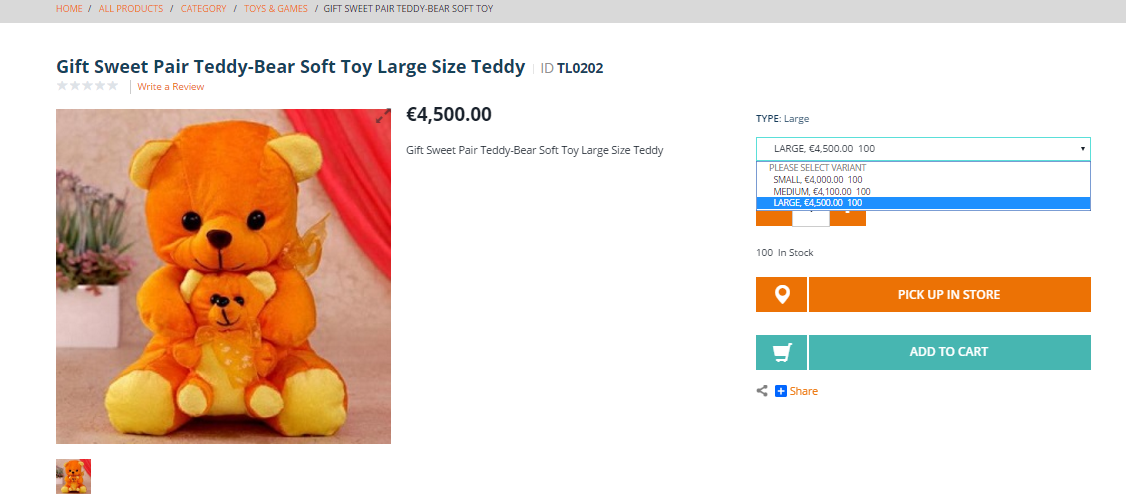
4.Update the system using HAC with the project data with egiftcore. It adds the newly created attributes to the database. After that modify the all variants related impex files:

* Products.impex
* Products-en.impex
* Products-media.impex
* Products-prices.impex
* Products-stocklevels.impex

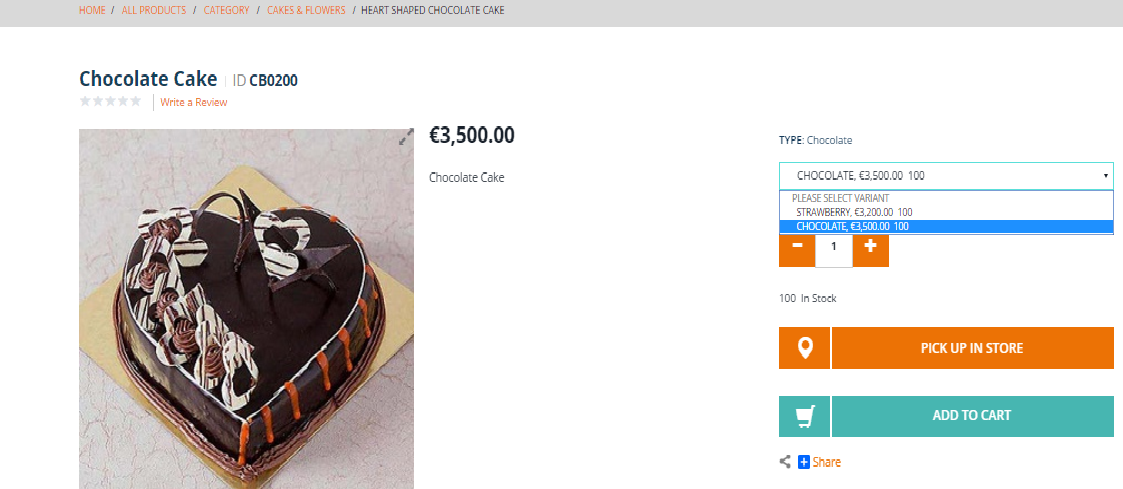
5.localize in egiftcore-locales\_en.properties.

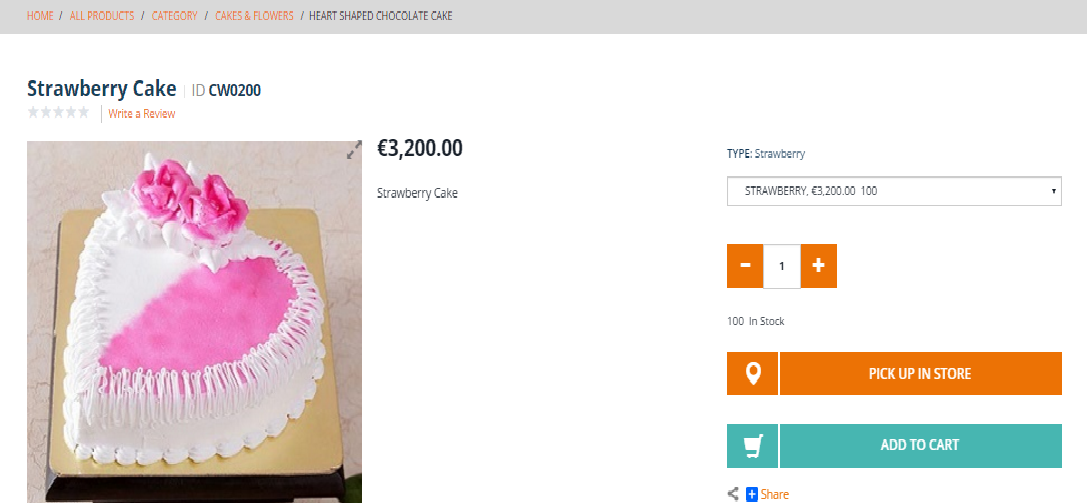
**Variants related image**

* **Size Variant:**

****

* **Style Variant:**

****



**2.3 Adding promotions to our products:-**

**Defining Customized Promotion type and enable it**

To avail offer for customers, the promotions are used. Define the new type of promotion:

**Buy X items of product A, get product B for free!**

We need to create the egiftcommerceservices extension using following steps:

1. In platform, **setantenv.bat**
2. Run the following command **ant extgen**
3. Choose the **yaddon**  template
4. Give extension as **egiftcommerceservices**
5. Package name **com.egift.egiftcommerceservices**

After extension created successfully, add following extension in your **config/localextentions.xml**

**<extension name='egiftcommerceservices' />**

And run ant **all.**

A promotion type is an entity we need to define in **egiftcommerceservices-items.xml**.  
Let's add a new promotion type that:

* extends from **ProductPromotion**
* defines **qualifyingCount** integer attribute
* defines **giftProduct** attribute of type Product
* defines the following localized String messages attributes : **messageFired** and **messageCouldHaveFired**

**egiftcommerceservices-items.xml :-**

****

Run **ant all** which generates the Jalo class **ProductQuantityThresholdFreeGiftPromotion**. Notice that abstract methods in the parent class are not generated automatically (that is, after all, the point of abstract methods: the extending class must provide an implementation for them).

Abstract methods inherited from parent classes need to be created manually. Refresh the egiftcommerceservices extension and open the generated **ProductQuantityThresholdFreeGiftPromotion** class and add.

**egiftcommerceservices/src/com/egift/jalo/ProductQuantityThresholdFreeGiftPromotion.java**



package  com.egift.jalo;

public class ProductQuantityThresholdFreeGiftPromotion extends GeneratedProductQuantityThresholdFreeGiftPromotion

{

    @Override

public List<PromotionResult> evaluate (final SessionContext ctx, final PromotionEvaluationContext promoContext)

    {

// Here need to write the Business logic and return the List of PromtionResult object

 }

}

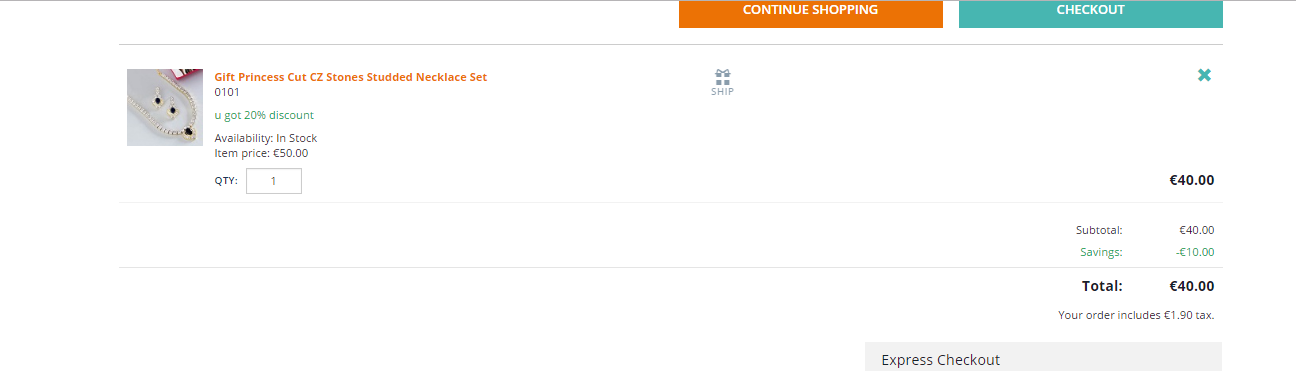
**/egiftinitialdata/resources/egiftinitialdata/import/sampledata/stores/egift/promotions.impex**

****

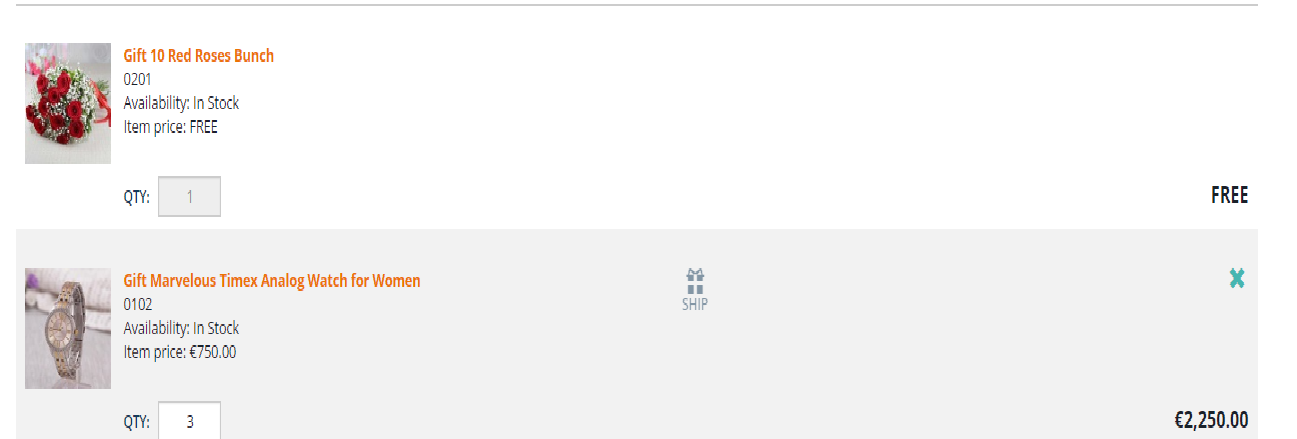
#### Rebuild and Verify

Rebuild the code base and start the server

**20% percentage discount promotion for a particular product**

****

**Buy 3 Egift Premium package will get 1 Egift standard package free:**

****

**2.4 Vouchers:-**

Voucher’s are a special form of discounts that can be applied to an order.  vouchers have to be actively redeemed by the customer.

There are 2 types of Vouchers available in HMC

1. Promotional Voucher

1. Serial Voucher

Promotional Voucher:-

With Promotional Vouchers Voucher code can be redeemed several times.

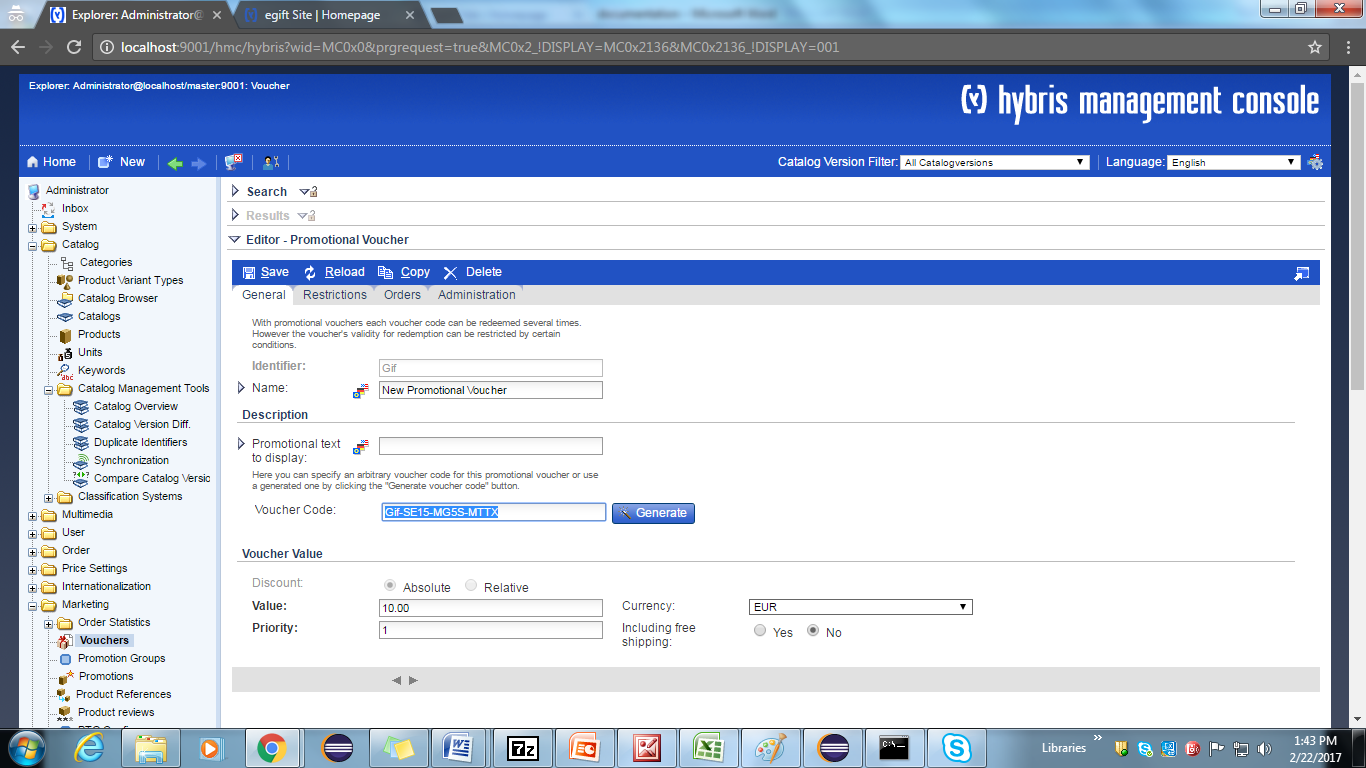
To Create the Promotional Voucher through HMC.

Go to HMC-🡪Marketing-🡪Vouchers-🡪Right-click on Vouchers-🡪click on create-🡪select Promotional Voucher

It opens new Promotional Voucher window, in that give the identifier name as like ‘ABC’ , Promotional text to display as like ‘Egift20’ , value as like ‘50’ ,Currency as like ‘EUR’ and click on Create button. After that click on Generate button it’s generate Voucher code. We have to redeem this voucher code in our Egift Site.

For this we have created one textbox & apply button in silentOrderPostPage.jsp, and create one controller like DefaultVoucherController.java for redeem this voucher code.

Creating Promotional Voucher through HMC.



After creating this New Promotional voucher & it’s code .We have to redeem this Voucher code in our Egift Site.

For this we have created one textbox & apply button in silentOrderPostPage.jsp, and create one controller like DefaultVoucherController.java for redeem this voucher code.

Add the following lines of code in the given Jsp.

Egiftstorefront/web/webroot/WEB-INF/views/desktop/pages/checkout/multi/silentorderpostPage.jsp

**silentOrderPostPage.jsp:-**

<c:url value=*"/checkout/voucher/redeem"* var=*"applyVoucherURl"*/>

<form action=*"*${applyVoucherURl}*"* method=*"GET"*

<p>Voucher Code :<input type=*"text"* class=*"text"* name=*"voucherNumber"* /> </p>

<input type=*"submit"* value=*"Apply"* />

</form>

Create the controller class like DefaultVoucherController, to add the given lines of code in your controller.

**Egiftstorefront/web/src**

**DefaultVoucherController.java**

****

package com.egift.storefront.controllers.pages;

@Controller

@Scope("tenant")

@RequestMapping(value = "/checkout/voucher")

public class DefaultVoucherController extends AbstractPageController

{

// voucher code redeem related code to check in store

@RequestMapping(value = "/redeem", method = RequestMethod.GET)

public String redeemVoucher(@RequestParam ("voucherNumber") final String voucherCode, final Model model, final HttpServletRequest request){

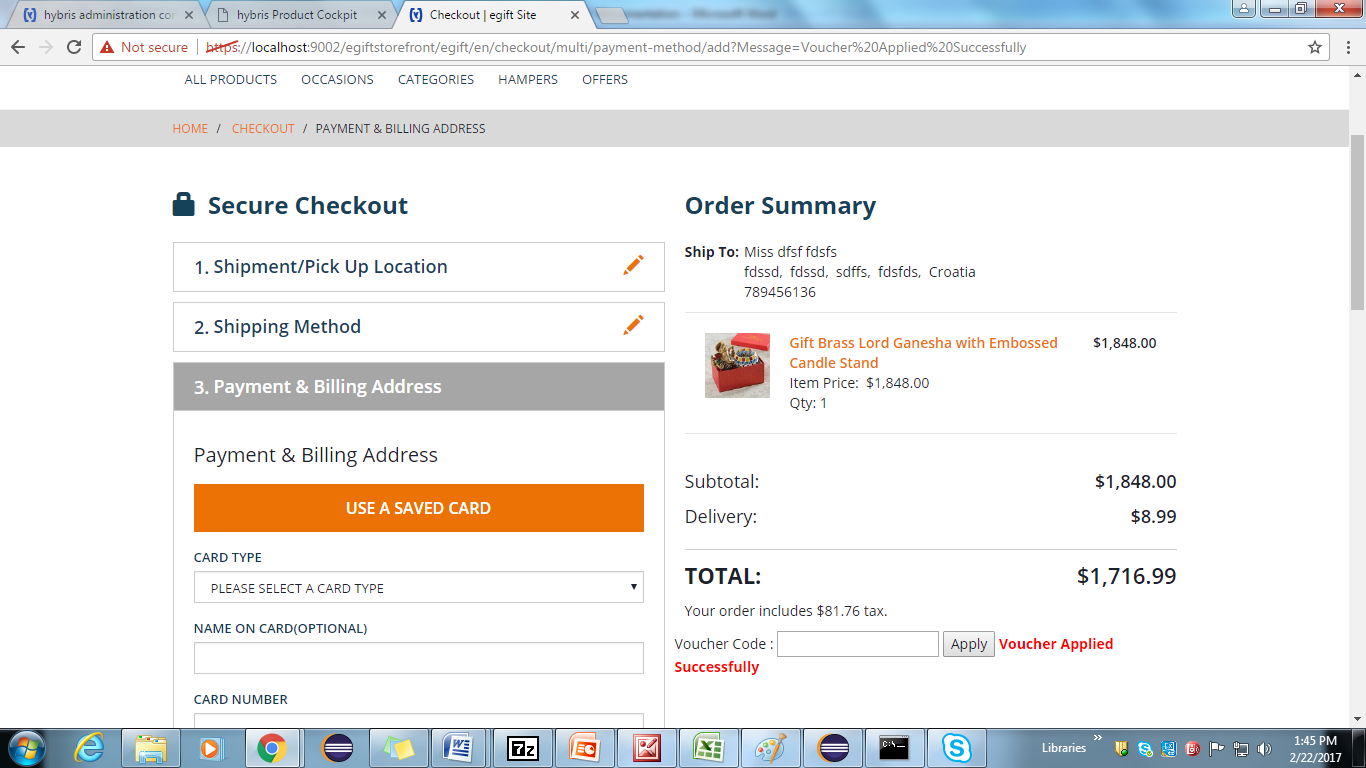
}

After that you run following command & refresh the workspace. Start the Hybris server & check the Egift Site.

<Hybris\_home>\Hybris\bin\platform>ant all

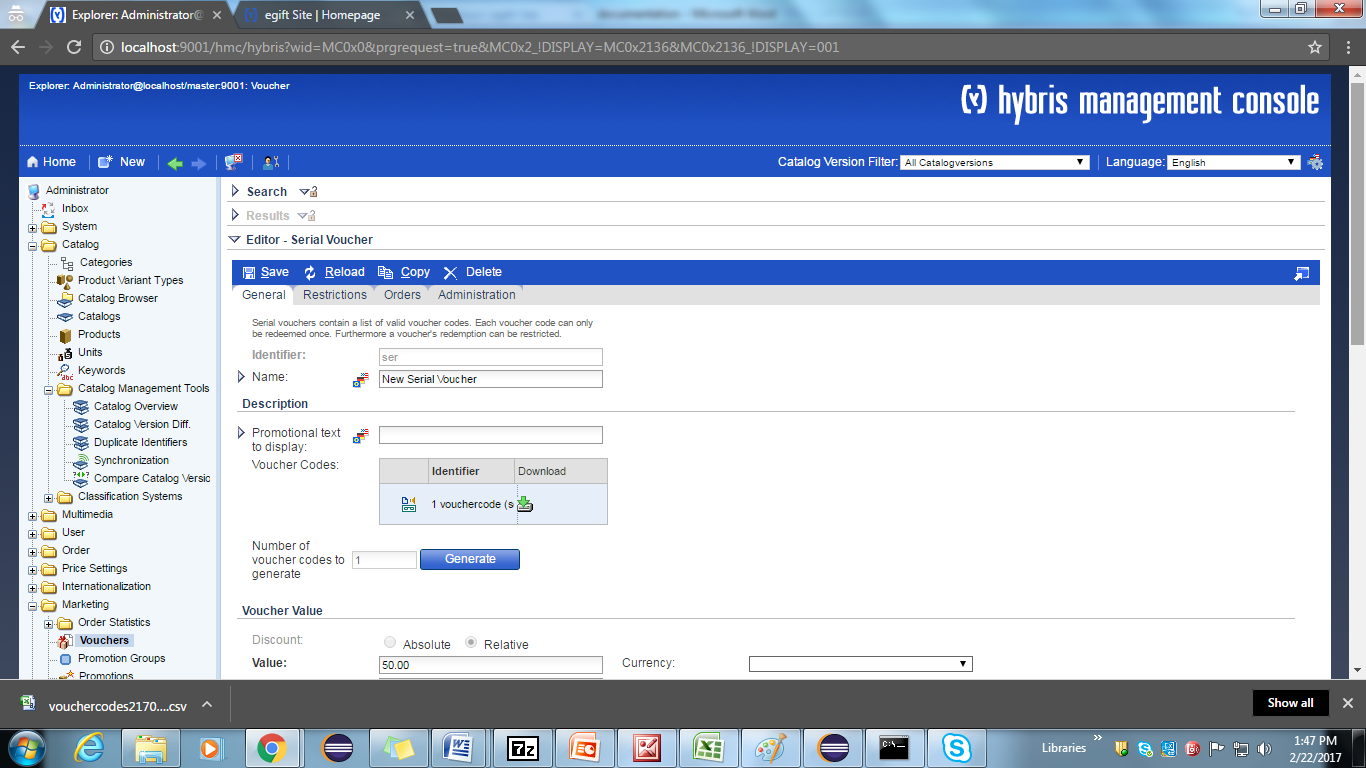
D:\HYBRISCOMM57\Hybris6.0\Hybris\bin\platform>Hybrisserver.bat

Go to Egift site check the Voucher code textbox & apply button. There you can apply your voucher code.

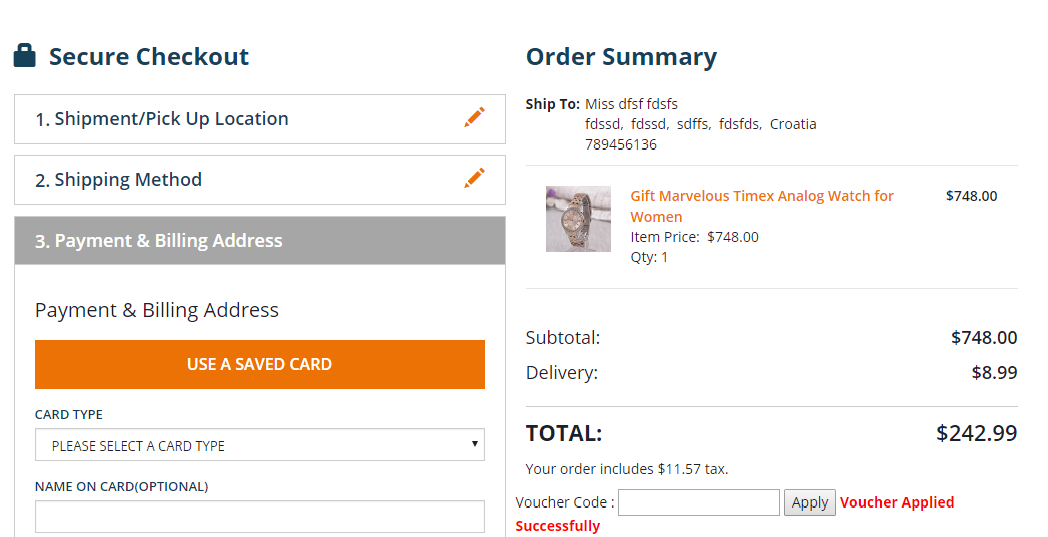


**Serial Voucher :-**

With Serial Vouchers Voucher code can be redeemed once. Serial vouchers contain a list of valid voucher codes.

Creating Serial Voucher through HMC 

Go to Egift site check the Voucher code textbox & apply button. There you can apply your voucher code



**2.5 Implementation for to define new attributes in Customer Registration Page**

****

1. Add an item type in your **egiftcore-items.xml** as follows and build your system and update.

|  |
| --- |
| <itemtype code=*"Customer"* extends=*"User"* autocreate=*"false"*generate=*"false"*> |
| <attributes> |
| <!-- auto ID which is generated by NumberSeries --> |
| <attribute autocreate=*"true"* qualifier=*" currentLocation"* type=*"java.lang.String"*> |
| <modifiers read=*"true"* write=*"true"* search=*"true"* optional=*"true"* /> |
| <persistence type=*"property"* /> |
| </attribute> |
| <attribute autocreate=*"true"* qualifier=*"hobby"* type=*"java.lang.string"*> |
| <modifiers read=*"true"* write=*"true"* search=*"true"* optional=*"true"* /> |
| <persistence type=*"property"* /> |
| </attribute> |
| </attributes> |
| </itemtype> |

1. Find the jsp which is responsible for sign in link. Using inspect element.
2. Create a **EgiftRegistrationForm.java** in our storefront with the custom and existing attributes.
3. Link this form in **register.tag** file with the commandName attribute.
4. Add new attributes in **register.tag** file
5. Add following lines to **base\_en.properties** file (custom\egift\egiftstorefront\web\webroot\WEB-INF\messages\\_en.properties)

*register.currentLocation= CurrentLocation*

*register.hobby= Hobby*

1. Now find the responsible controller by inspecting the register button from front end.
2. Go to **LoginPageController.java** file and find the method with /register.

*@RequestMapping(value = "/register", method = RequestMethod.****POST****)*

1. Create own validation class in our store front

**EgiftRegistrationValidator.java**

1. Use this class in **LoginPageController.java** for front end validation.
2. Create **AbstractEgiftRegisterPageController.java** which extends **AbstractPageController** and copy code of **AbstractRegisterPageController.java**
3. Create **AbstractEgiftLoginpageController.java** which extends **AbstractEgiftRegisterpageController** and copy the code of **AbstractLoginpageController**.**java**
4. Create **EgiftDefaultCustomerFacade.java** in egiftfacades extension and extend with **DefaultCustomerFacade.java** and create a bean in **egiftfacades-spring.xml**
5. Now update **LoginpageController.java** with extends **AbstractEgiftLoginpageController.java**
6. Add the following lines in **LoginpageController.java**

@Resource (name = "egiftRegistrationValidator")

**Private** EgiftRegistrationValidator egiftRegistrationValidator;

1. Add the following lines in **AbstractEgiftRegisterpageController.java**

@Resource(name = "egiftRegistrationValidator")

**private** Validator egiftRegistrationValidator;

@Resource(name = "egiftDefaultCustomerFacade")

**private** EgiftDefaultCustomerFacade egiftDefaultCustomerFacade;

1. In **AbstractEgiftRegisterpageController.java** go to **processRegisterUserRequest()** and change the form class.
2. Add new attributes to **RegisterData** bean id from **egiftfacades-beans.xml** and do ant all from cmd prompt.
3. Convert the form object to data object by setting the data from EgiftRegisterationForm.java
4. Now pass the data object to facade using the register ().

Ex: getEgiftDefaultCustomerFacade().register (data);

1. In **EgiftDefaultCustomerFacade** copy the register () method from **DefaultCustomerFacade.java** and override super class method.
2. In the register () method add the new attributes data from RegisterData object to CustomerModel.
3. This method will have model service which is used to save the CustomerModel in db
4. With this we will have all logs and spring validation steps.
5. To retrieve the data in myaccountpage we have to write a populator and add the new attributes to **CustomerData** from **egiftfacades-beans.xml** as follows. The **RegisterData** is used in conversion of form object to data object. After adding this run ant all command from cmd. Refresh your work space.

**Ex:** <bean class=*"de.hybris.platform.commercefacades.user.data.RegisterData"*>

<property name=*"currentLocation"* type=*"java.lang.String"*/>

<property name=*"hobby "* type=*"java.lang.String"*/>

</bean>

<bean class=*"de.hybris.platform.commercefacades.user.data.CustomerData"*>

<property name=*"currentLocation"* type=*"java.lang.String"*/>

<property name=*"hobby "* type=*"java.lang.String"*/>

</bean>

1. Create a **EgiftCustomerAttributesPopulator.java** in egiftfacades Extension add the following code
2. The below code will override the populate () method which is present in CustomerPopulator and we will call the same method using super method.

**public** **class** **EgiftCustomerAttributesPopulator** **extends** **CustomerPopulator**

{

@Override

**public** **void** populate(**final** CustomerModel source, **final** CustomerData target)

{

**super**.populate(source, target);

target.setCurrentLocation(source.getCurrentLocation());

target.setHobby(source.getHobby());

}

}

1. Add this populator to **egiftfacades-spring.xml** file as shown below.

<alias name=*"egiftCustomerAttributesPopulator"* alias=*"customerPopulator"* />

<bean id=*"egiftCustomerAttributesPopulator"* class=*"com.egift.facades.populators.EgiftCustomerAttributesPopulator"*

parent=*"defaultCustomerPopulator"* />

1. Now update the **accountprofilepage.jsp** as follows

<tr>

<td><spring:theme code=*"profile.currentLocation"* text=*"CurrentLocation"*/>: </td>

<td>${fn:escapeXml(customerData.currentLocation)}</td>

</tr>

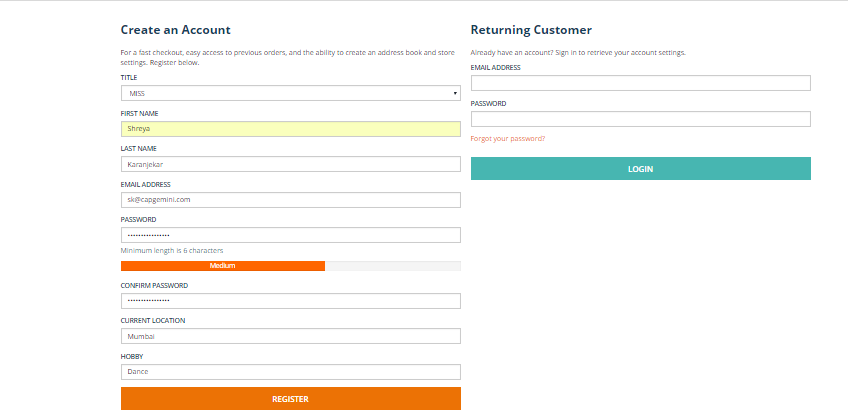
<tr>

<td><spring:theme code=*"profile.hobby"* text=*"Hobby "*/>: </td>

<td>${fn:escapeXml(customerData.hobby)}</td>

</tr>

1. Localize new attributes in **base\_en.properties** file.
2. Do ant all & start the server Update select localize types checkbox.
3. Now you will be able to see the custom data in profile page



**2.6 Implementation for Update My Account tab with Customer Register Page new attributes:-**



1. Define new customer properties in **CustomerData** bean id of **egiftfacades-beans.xml file.**

<bean class=*"de.hybris.platform.commercefacades.user.data.CustomerData"*>

<property name=*" currentLocation "* type=*"String"* />

<property name=*" hobby"* type=*"java.lang.String"*/>

</bean>

2. After that do ant all & Refresh the Workspace.

3. Define the populator in **egiftfacades-spring.xml** file

<alias name=*"egiftCustomerAttributesPopulator"* alias=*"customerPopulator"*/>

<bean id=*"egiftCustomerAttributesPopulator"* class=*"com.egift.facades.populators.EgiftCustomerAttributesPopulator"* parent=*"defaultCustomerPopulator"*/>

4. After that do ant all & update egiftfacades extension.

5. Create MyAccount CMS link component in sampledata folder of **cms-content.impex** file.

**# CMS Link Components**

INSERT\_UPDATE **CMSLinkComponent;$contentCV[unique=true];uid[unique=true];name;url;&linkRef;&componentRef;target(code)[default='sameWindow'];$category;$product;**

**;; AccountProfileLink;AccountProfileLink;/my-account/profile;AccountProfileLink;AccountProfileLink;;;;**

6. Under HeaderLink Slot data to add AccoutProfileLink. It’s look like below.

**# Content Slots**

**INSERT\_UPDATE ContentSlot;$contentCV[unique=**true**];uid[unique=true] ;cmsComponents(&componentRef)**

**;; HeaderLinksSlot; ContactInfo,AccountProfileLink**

7. Localize CMSLink component in **cms-content\_en.impex** file of Sample data.It’s look like below.

# CMS Link Components

**UPDATE CMSLinkComponent;$contentCV[unique=true];uid[unique=true];linkName[lang=$lang]**

**;; AccountProfileLink;"My Account"**

8. Add new Customer Register Page attributes in **accountProfileEditPage.jsp** file.

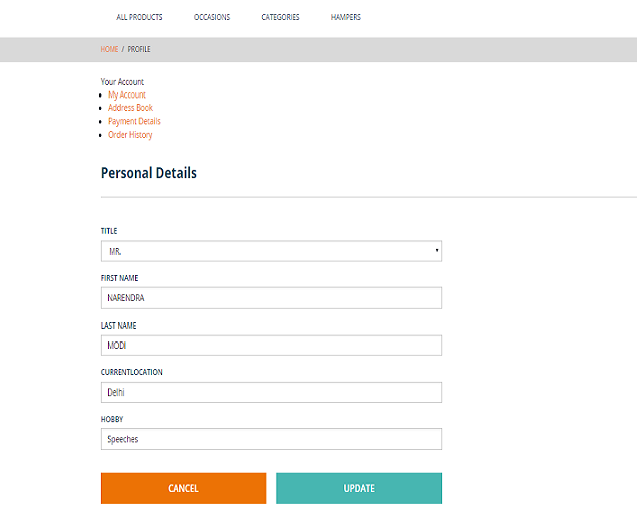
9. Create **EgiftUpdateProfileForm** which will have exiting attribute and add the new two attributes.

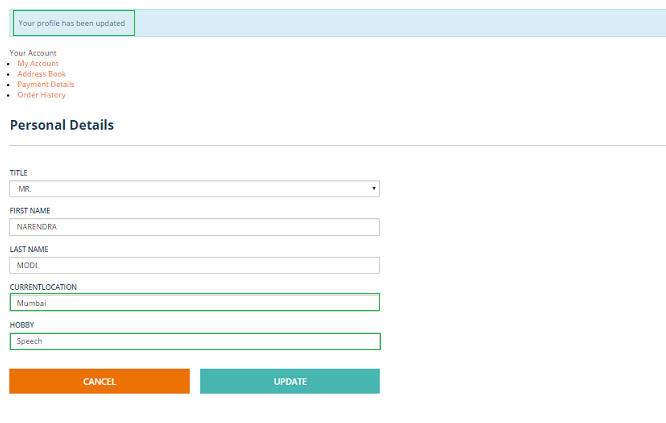
10. Create **EgiftProfileUpdateValidator** with new and existing attributes.

11. Create **EgiftDefaultCustomerFacade** class with new attributes.

12. Create **EgiftCustomerAttributesPopulator** with new attributes to populate the data.

13. Do the changes in **AccountPageController.java** file to update the new attributes in, /update-profile Requestmapping URL pattern of controller class.



****

**2.7 Implementation for Customize CSCockpit with new Customer attributes.**

1. Add the new attributes defined in Customer page, those attributes define in **Customer\_EditCustomerDetails\_CockpitGroup.xml** file of ebookcockpits extension (\hybris\bin\ext-channel\cscockpit\resources\cscockpit\import\config).

<property qualifier=*"Customer.currentLocation"* visible=*"true"*/>

<property qualifier=*"Customer.hobby"* visible=*"true"*/>

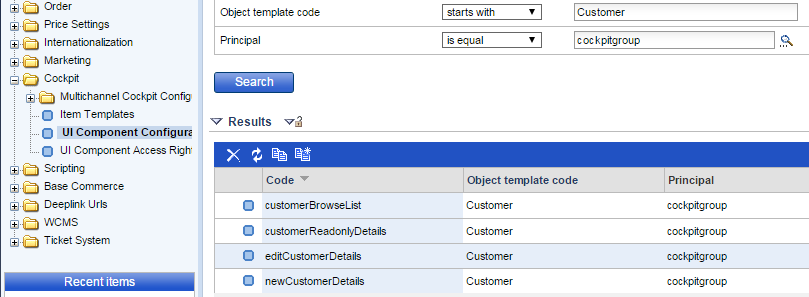
2. Localize those Customer Registration Page attributes in **commerceservices-locales\_en.properties** file.

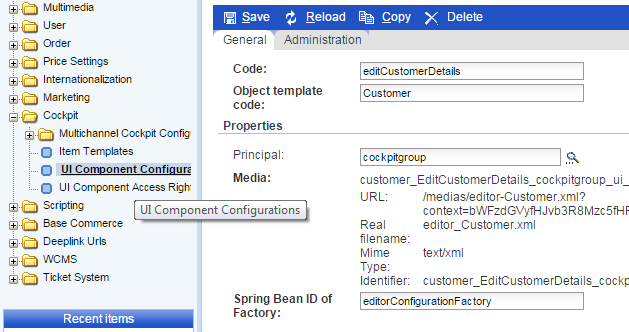
type.Customer.currentLocation.name=CurrentLocation

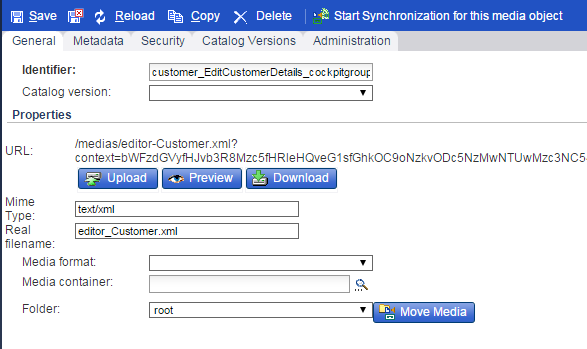
type.Customer.hobby.name=Hobby

3. Do ant all & start the server go to hac🡪platform 🡪update🡪egiftcockpits & commerce services extension.

4. Go to hmc🡪Cockpit---🡪UI Component Configuration🡪give below values(Customer, cockpitgroup) & search, u will get editorcustomerDetails, click on that link there media type click on download , add new Customer registration Page attributes in the file & upload the file , click on Save button. For that see the below screenshots.









2.8 Implementation for coming soon products:-



1. Create new attribute “**CommingSoon**” in product itemtype in **egiftcore-items.xml** file.

|  |
| --- |
| <itemtype code=*"Product"* extends=*"GenericItem"* autocreate=*"false"* generate=*"false"*>  <attributes>  <attribute qualifier=*"CommingSoon"* type=*"java.lang.Boolean"*>  <description>Coming Soon Products</description>  <modifiers />  <persistence type=*"property"* />  </attribute>  </attributes>  </itemtype> |

1. Navigate to platform directory in command prompt and run following commands.

**ant clean all**

**hybrisserver.bat**

1. Go to hybris admin console <http://localhost:9001/> and update system by selecting Update Running System.
2. Create **ComingSoon** link to display coming soon products. You can create any where you want that link.

I have created in **header.tag** file.

<div>

<ul>

<li>

<ycommerce:testId code=*"header\_ComingSoon"*>

<a href=*"*<c:url value=*"/coming-soon/products"*/>*"* target=*"\_blank"*>

<spring:theme code=*"header.link.ComingSoon"* text=*"Coming Soon"*/>

</a>

</ycommerce:testId>

</li>

</ul>

</div>

1. Create new template for coming soon product page in **coredata/cms-content.impex**
2. Create new **comingSoonLayout.jsp** for coming soon product page in **coredata/cms-content.impex**

(egiftstorefront\web\webroot\WEB-INF\views\desktop\pages\layout\comingSoonLayout.jsp)

1. Create controller **ComingSoonController.java** to redirect to that page.
2. Add **ComingSoon** attribute in **egiftfacades-beans.xml** file inside ProductData bean

<bean class=*"de.hybris.platform.commercefacades.product.data.ProductData"*>

<property name=*"genders"* type=*"java.util.List&lt;com.egift.facades.product.data.GenderData>"*/>

<property name=*"CommingSoon"* type=*"Boolean"*/>

</bean>

1. Now we are going to create **ComingSoonDao.java** to retrieve coming soon products from product table.
2. Write bean entry of **ComingSoonDao** in **egiftcore-spring.xml** file & build the system (**ant clean all**).

<bean id=*"comingSoonDao"*

class=*"com.egift.core.comingsoon.dao.ComingSoonDao"* parent=*"abstractItemDao"*>

<property name=*"flexibleSearchService"* ref=*"flexibleSearchService"*/>

</bean>

1. Create **ComingSoonService.java file**
2. Write bean entry of **ComingSoonService** in **egiftcore-spring.xml** file. Give reference of above Dao & build the system (**ant clean all**)

<bean id=*"comingSoonService"* class=*"com.egift.core.comingsoon.service.ComingSoonService"*>

</bean>

**13)** Create the **ComingSoonfacades.java** file.

**14)** Write bean entry of **ComingSoonfacades** in **egiftfacades-spring.xml** file. Give build the system (**ant clean all**)

<bean id=*"comingSoonfacades"* class=*"com.egift.core.comingsoon.facades.ComingSoonfacades"*>

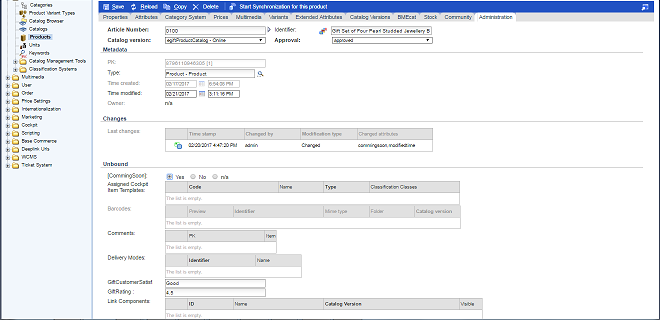
</bean>

**15)** Change the controller to get coming soon products from service & set in model.

**16)** Change **comingSoonLayout.jsp** to show your coming soon products on page.

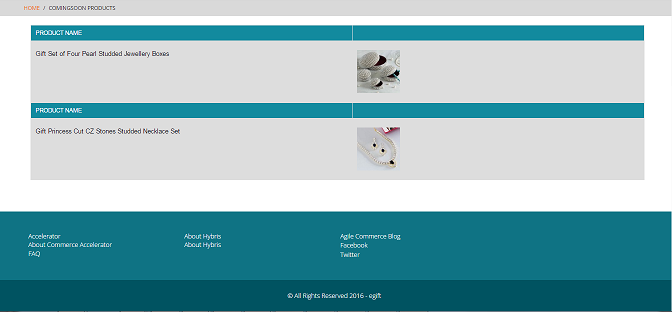
**17)** Build the System (**ant all**) and start the hybris server (**hybrisserver.bat**).

**18)** Go to Hybris Management Console & change ComingSoon status **Yes** for few products in online catalog & save.



1. Go to Your Site and click on **Coming Soon** link.

**Coming soon page**



## Implementation for customize PDP to define new attributes:-



1. Create the PDP attributes under Product type in **egiftcore-items.xml**

<itemtype code=**"Product"** autocreate=**"false"** generate=**"false"**>

<!-- PDP new 2 Attributes declaration here....start s-->

<attribute qualifier=**"giftRating"** type=**"java.lang.String"**>

<description>**Customer Rating of the gift.**</description>

<modifiers/>

<persistence type=**"property"**/>

</attribute>

<attribute qualifier=**"giftCustomerSatisfaction"** type=**"java.lang.String"**>

<description>**Customer Satisfaction of the gift.**</description>

<modifiers/>

<persistence type=**"property"**/>

</attribute>

</attributes>

</itemtype>

After that Go to Platform do ant all, Refresh the workspace.

After that Go to HMC🡪Platform🡪Update🡪egiftcore extension.

**2.** Define the 2 attributes in ProductData of **egiftfacades-beans.xml file**

<bean class=*"de.hybris.platform.commercefacades.product.data.ProductData"*>

<property name="giftRating" type="java.lang.String"/>

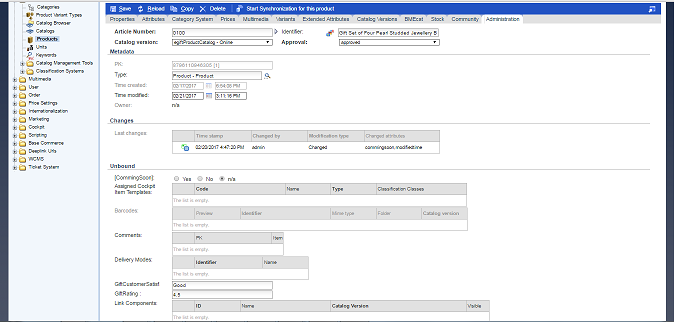
<property name="giftCustomerSatisfaction" type="java.lang.String"/>

</bean>

After that Go to Platform do ant all, Refresh the workspace.

After that Go to HAC 🡪 Platform 🡪Update 🡪 egiftfacades extension.

Open the HMC --> Catalog --> Product-->Administration tab add new attributes values

****

**3.** **ProductPageController.java** file handles request for the PDP attributes. Add all the attributes values created from HMC in the product model. Put the entire attribute’s data in List. {attributesList}.

**4.** Display the data exists in the List on PDP through **addtocartaction.jsp** file.

Add the below lines of code in jsp file.

<c:if test=*"*${attributesList **ne** null}*"*>

<div id=*"attributesId"*>

GiftRating:<c:out value=*"*${attributesList.get(0)}*"*/><br><br>

GiftCustomerSatisfaction :<c:out value=*"*${attributesList.get(1)}*"*/><br><br>

</div>

</c:if>

**5.** In **egiftfacades-spring.xml** file created the **EgiftProductAttributesPopulator** class to populate the PDP attributes data.

<alias name=*"egiftProductAttributesPopulator"* alias=*"productBasicPopulator"*/>

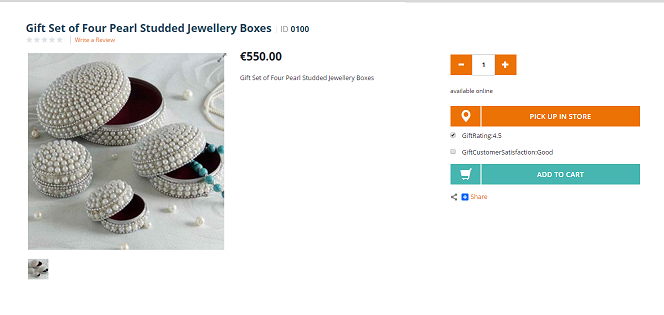
<bean id=*"egiftProductAttributesPopulator"* class=*"com.egift.facades.populators.EgiftProductAttributesPopulator"* parent=*"baseProductPopulator"*/>

Note: After that do ant all & start the server update egiftfacades extension.

**6.** After that you have created the **EgiftProductAttributesPopulator.java** file to populate the attributes data.

**7.** After that do ant all & Start the server, check the site to click any product to see new attribute’s in PDP.

See the list of attribute’s on corresponding products in product details page looks like below.



**2.10 Implementation for Customize Product cockpit with new PDP attributes:-**

****

1. Add the new attributes defined in PDP page, those attributes define in **editorArea\_Product.xml** file of **egiftcockpits** extension (\hybris\bin\custom\egift\egiftcockpits\resources\egiftcockpits-config\cockpitgroup)

<property qualifier=*"product.giftRating"* editor=*"wysiwyg"* />

<property qualifier=*"product.giftCustomerSatisfaction"* editor=*"wysiwyg"* />

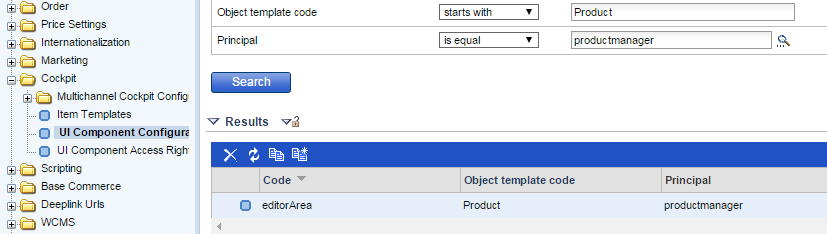
2. Localize those PDP attributes in **commerceservices-locales\_en.properties** file.

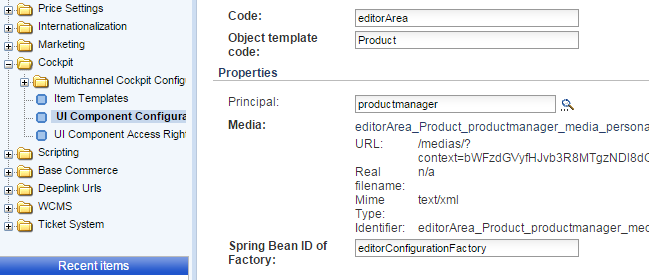
type.Product.giftRating.name=GiftRating

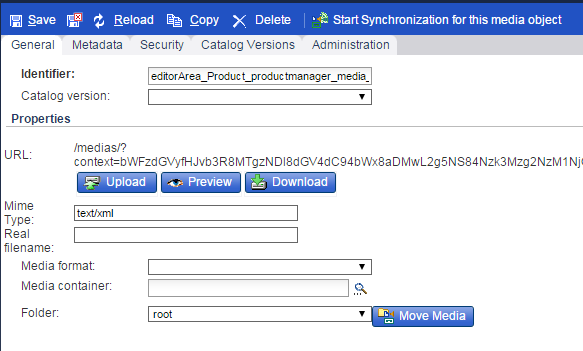
type.Product.giftCustomerSatisfaction.name=CustomerSatisfaction

3. Do ant all & start the server go to hac🡪platform -🡪update-🡪egiftcockpits & commerce services extension.

4.Go to hmc-🡪cockpit---🡪UI Component Configuration🡪give below values(Product, productmanager) & search, u will get **editorArea** , click on that link there media type click on download , add new pdp attributes in the file & upload the file , click on Save button. For that see the below screenshots.







**2.11 Implementation for Hot Folder:**



We have already seen how you can use **ImpEx** files to import data in the system. hybris supports **hot folders** which are folders from which data can be automatically imported into hybris by simply placing the data inside of the folder. The hybris **acceleratorservices extension template** comes with a batch package that enables automated importing of data from hot folders.

The Hot Folder is based on the principle of spring integration which extends the spring programming model.

**Enable the standard hot folder for Egift:**

1. Create the **egiftcore/resources/ egiftcore /integration/hot-folder-store- egiftcore -spring.xml.**

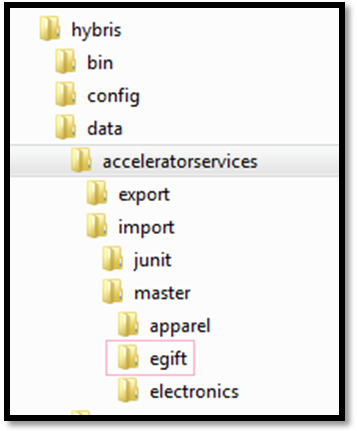
2. To enable in the spring import the above spring file in **egiftcore-spring.xml**

**<import resource="classpath:/egiftcore/integration/hot-folder-store-egift-spring.xml"/>**

3. After restarting the server it will create the egift folder as we defined the bean

**<bean id="baseDirectoryEgift" class="java.lang.String">**

**<constructor-arg value="# {baseDirectory}/${tenantId}/egift" /></bean>**



4. Restart the server to apply the changes.

5. To do the small test put price-003.csv in the egift directory if everything goes right it will go in the archive folder for the next operation. The price of the product will be in both staged and online version of the catalog. It will be reflect on only to the product detail project page not on the product list page.

**Before the importing the hot folder:**



**After the importing the Hot Folder:**

****

**2.12 Implementation for Pickup in store:**



BUY\_AND\_COLLECT enables a process in the yacceleratorstorefront extension where in the customer can choose to add items to their cart that they can pick up at one or more stores that currently have the item in stock. The user will be required to pay online before collecting the item.

1. Enabling Pickup in store on product detail page

The PickupInStoreMode mode has already been set to BUY\_AND\_COLLECT on the base store

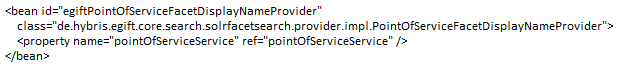
See the INSERT-UPDATE basestore in the **store.impex**.

2. Add one or more PointsOfService of type **STORE**. These PointsOfService will also appear in the Store Locator as mentioned in the **points-of-service.impex** in that we link a Warehouse to each PointsOfService that offers the Pickup From Store service.

3. For the different store the required images are stored in the **points-of-service-media.impex**

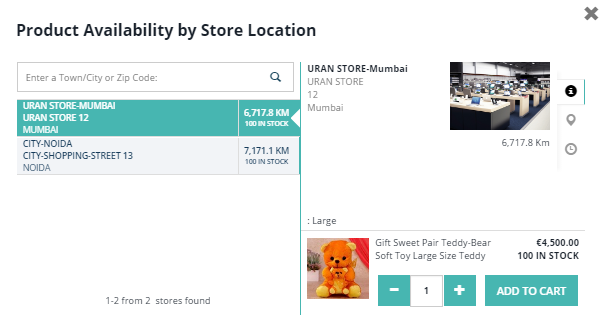
Of those reference is used in the **points-of-service.impex.**

4. Define the **egiftPointOfServiceFacetDisplayNameProvider** for the facet search



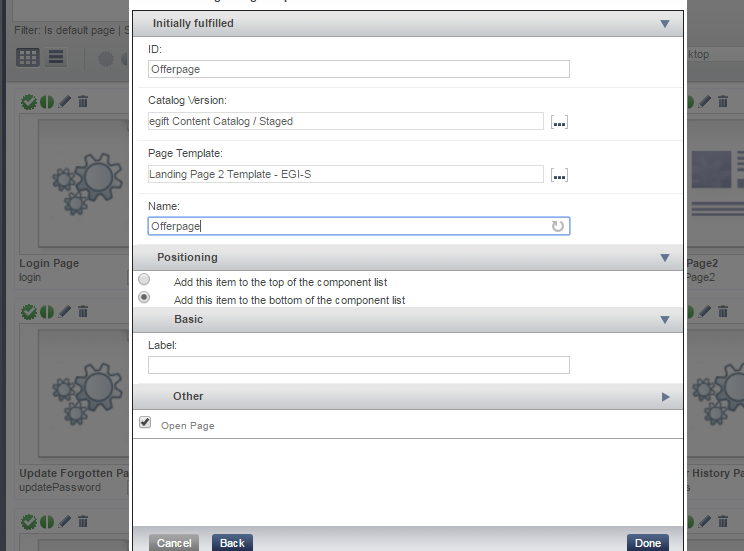
5. To bring pickup store flag on the product list page is defined in the solr.impex (SolrIndexed Property).

6. Run a system update, selecting only **Import sample data.**



**2.13 WCMS Cockpit Customization:**

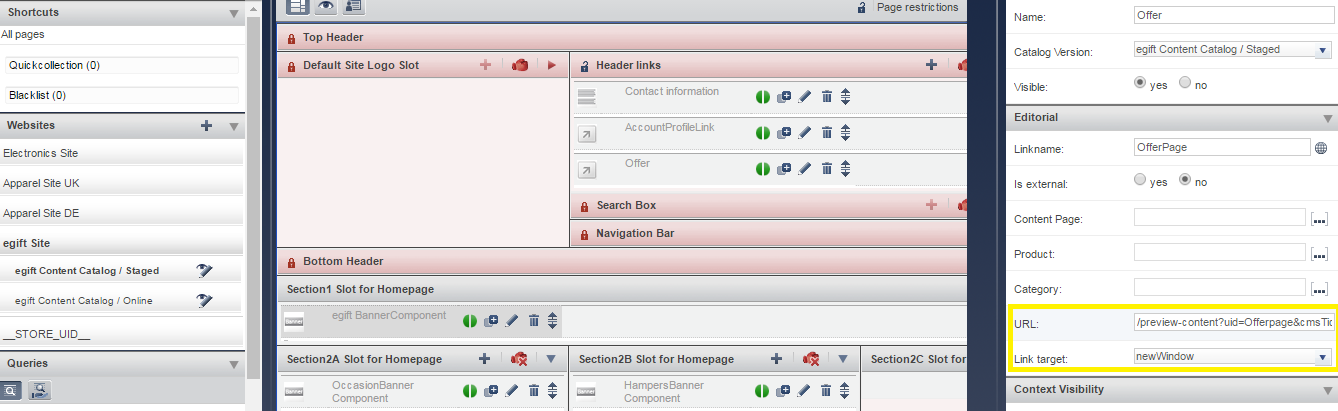
1. Login to CMScockpit : <https://localhost:9002/cmscockpit/login.zul>
2. Select your site content catalog: egift site content catalog (staged).
3. Create new OfferPage.



4. Copy the URL of OfferPage.

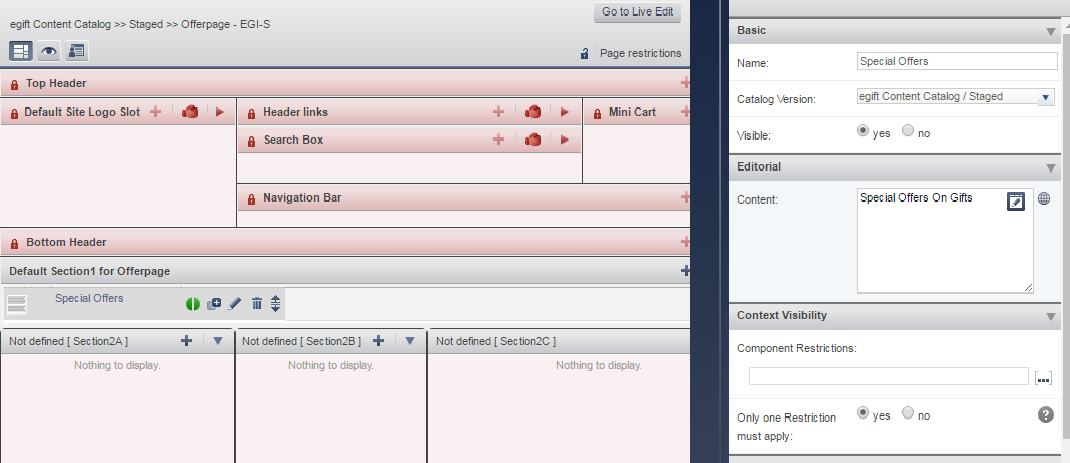
5. Search for HomePage and unlock Header link to create new link component.

6. Map the URL of Offerpage to newly created link.



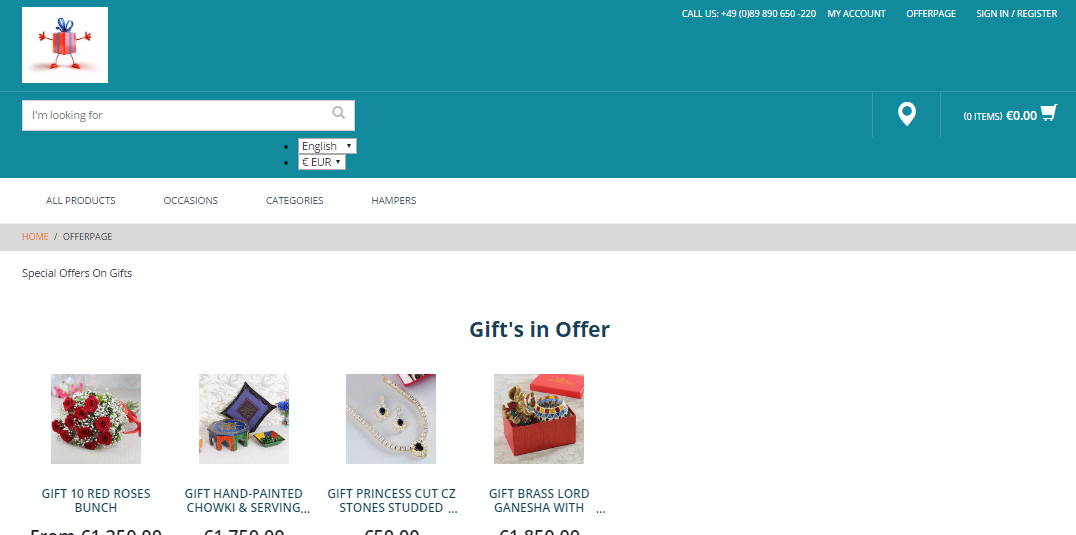
7. Synchronize OfferLink and Offer Page.

8. Create new paragraph component on Offer Page.



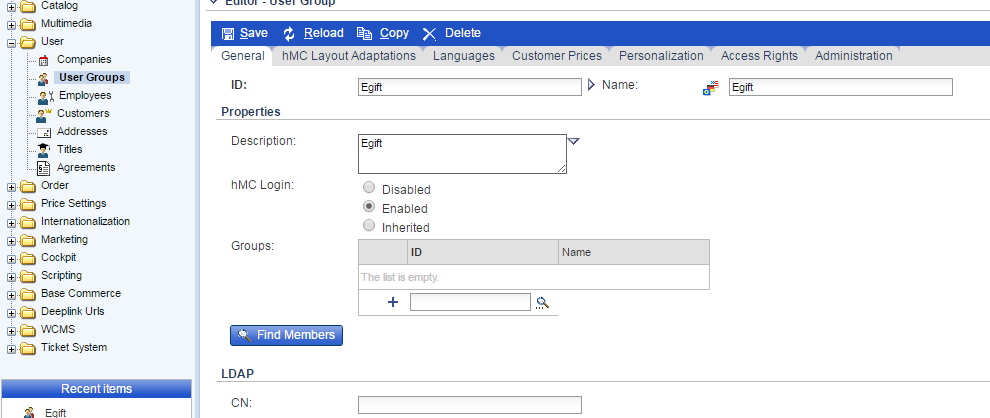
9. Create product carousel compent to add products in offers.

10. Go to store and check for offers link.



**2.14.1 Implementation for Defining users on HMC assign required permissions:**

1. Login to the hybris management console and create the new user group give the permission of the hmc login in the Genral tab.



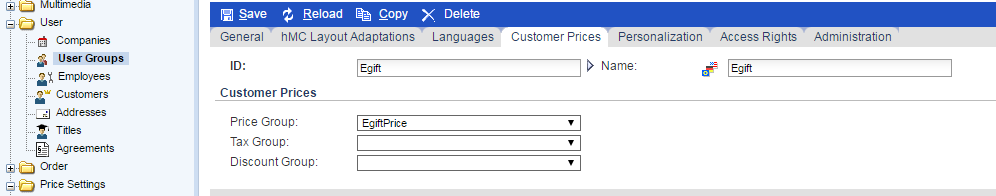
1. Create the new customer and put the new customer to the newly created group.



4. Now we are going to create the new price group which will assign to the new user group.

For that go to Price Settings/Prices/Customer Price List and create price rows for customer price list.

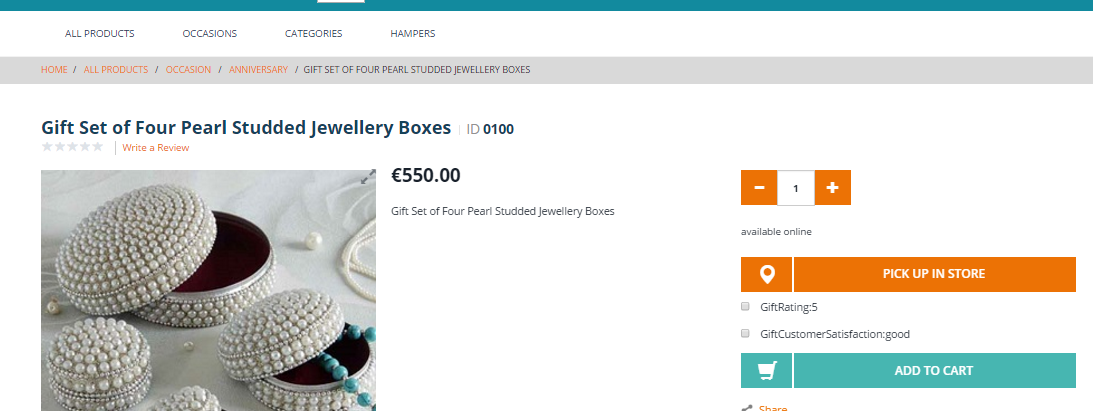
5. Assign the price group to the new user group. Hence the customer who belongs to this group will have different price.

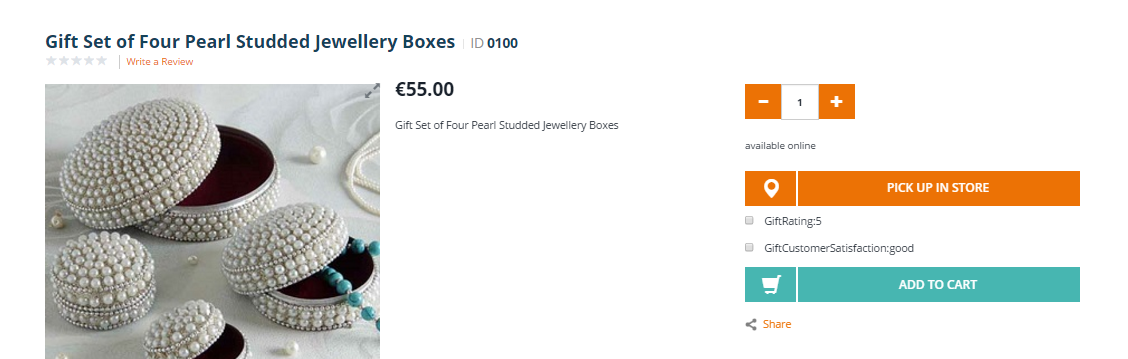


For regular person the price of product ID 0100 is 550euro.

For the new usergroup customer will have different price.

* For Normal customers:



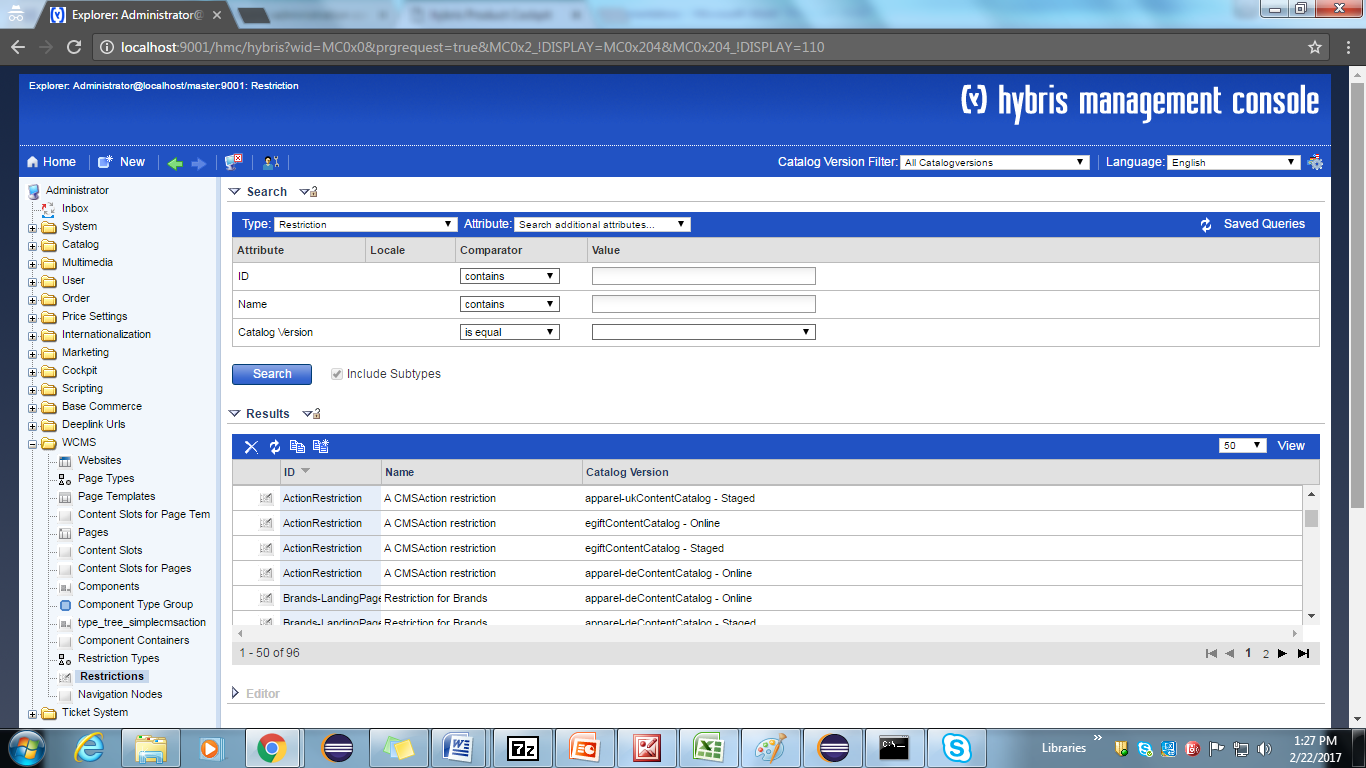
* For privileged Customers: 

**2.14.2 Apply/Add restrictions on a component:**

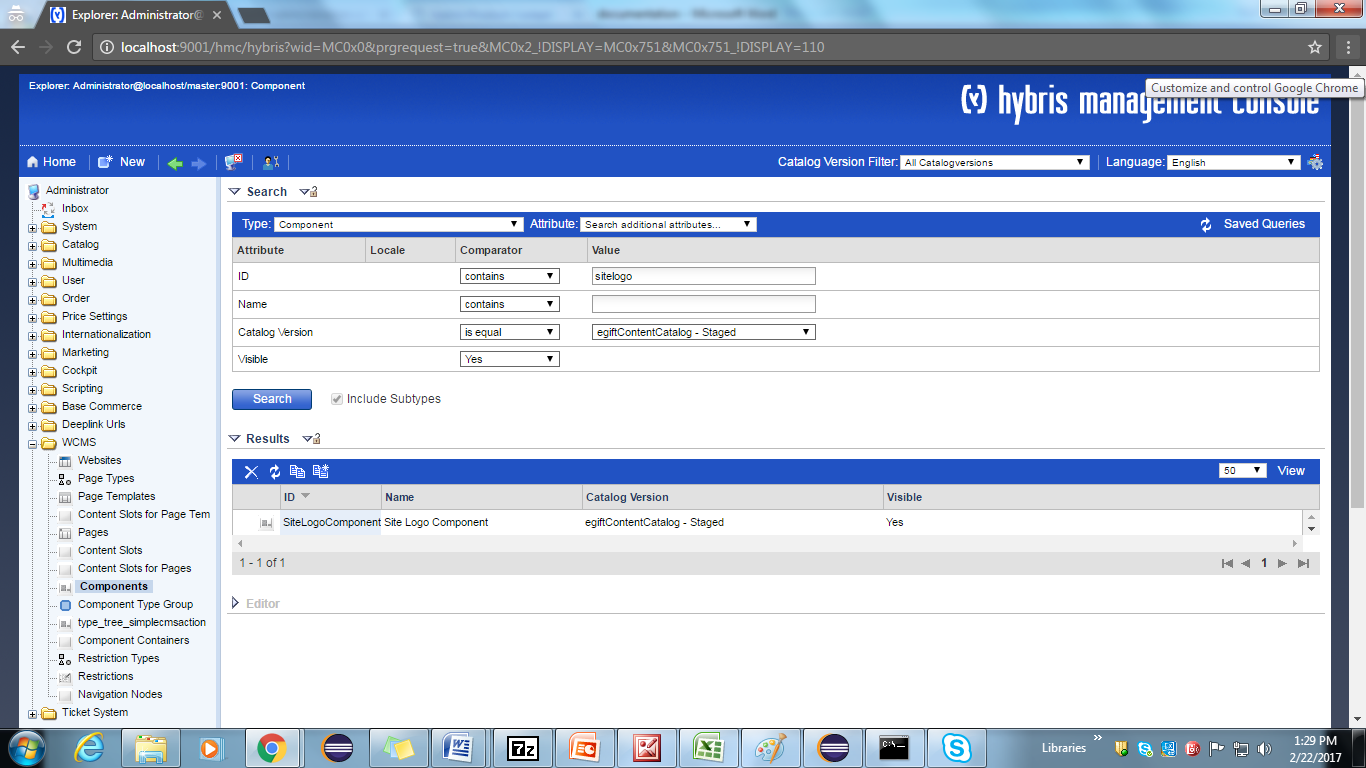
In real time scenarios sometimes we have requirements to restrict some sort of information to few customers only. So for such kind of requirements we have concept of restrictions. We have multiple type of restrictions in Hybris.

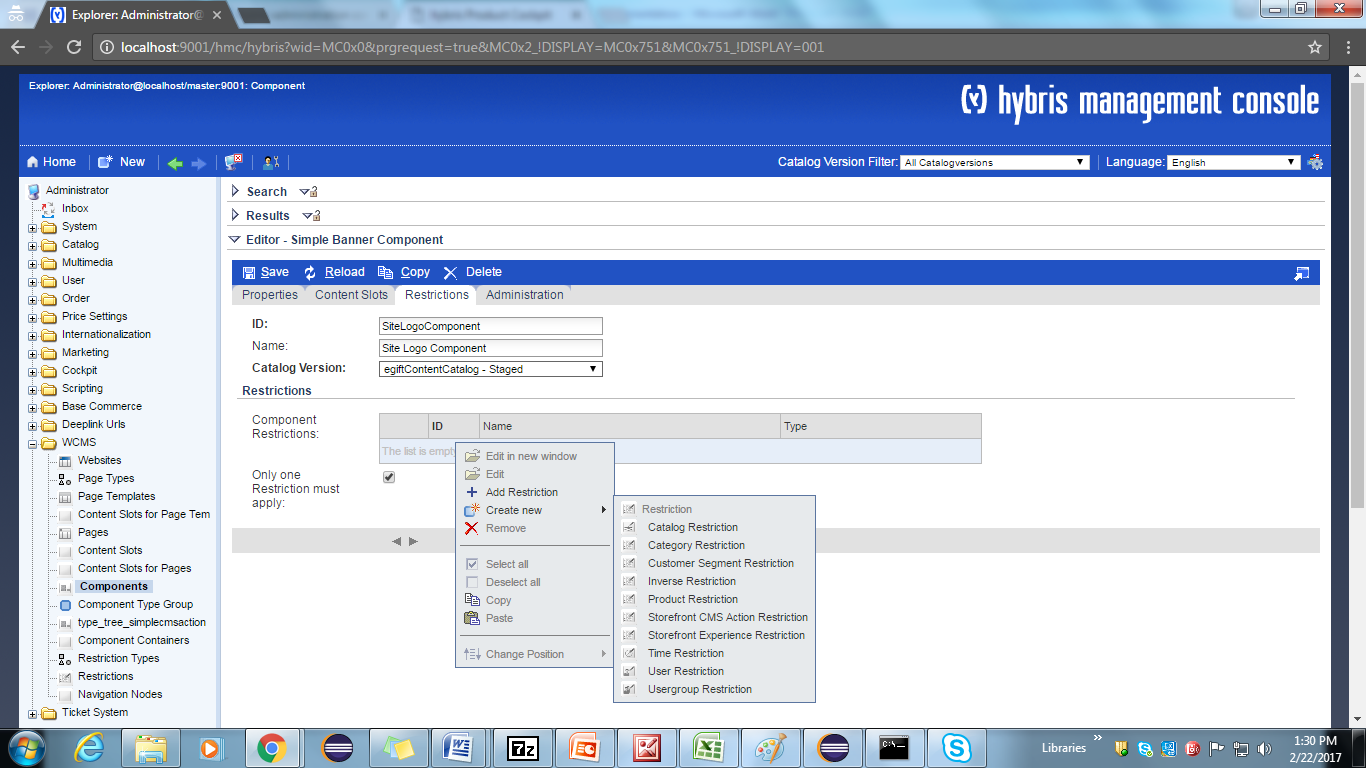
To see type of restrictions login to hmc

Tool : <http://localhost:9001/hmc/hybris>  
  
Go to WCMS -> Restrictions type. See screen shot below

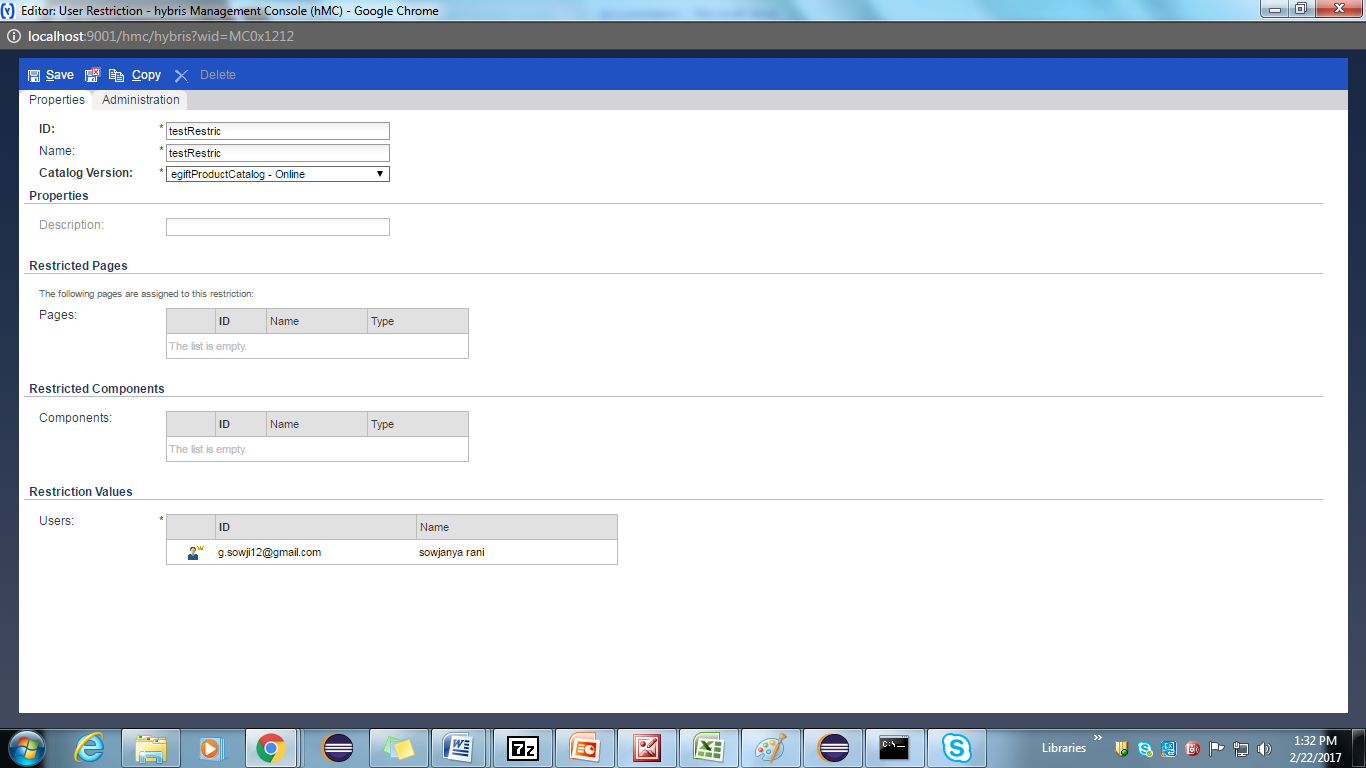


Now, how we can add restriction to a component. To explain restrictions lets take user restriction as of now. Follow below steps.  
  
1.) Go to WCMS-> Components and search for sitelogocomponent  
2.) Double click that component and go to restrictions section of that. Right click on restriction and create new restriction as shown below.





3.) Below screen will be displayed. Enter the details like name, id ,catalog version, name of the component this restriction applied to and the user id of the user as this restriction type is user restriction.   
Screen shot below for reference.



4.) Save and close  
5.) Open electronics site locally and we could see the restriction is applied on the SiteLogo component.As the site logo is restricted to the user, we can't see the site logo in general. See screen shot below.

**2.15 Define own shipping method :**

Basic store data:-

The files located in the above folders need to be modified.

In **store.impex** you must add new macro:

$storePrefix=egift

Because we are not going to use classification system we can remove the following line:

and remove that macro from BaseStore catalogs line.

We should also add some delivery (shipping) methods by adding these lines to the impex file.

For adding custom shipping method add following lines in **store.impex**

INSERT\_UPDATE BaseStore2DeliveryModeRel;source(uid)[unique=true];target(code)[unique=true]

; egift; premium-net

;egift; standard-net

;egift; premium-gross

;egift; standard-gross

;egift; pickup

;egift; free-standard-shipping

;egift; egift-shipping

Add same thing in **delivery-modes.impex** with the delivery cost and

**Delivery-modes\_en.impex** according to the zonal things.

**Egiftcore/resources/egiftcore/import/common/delivery-modes.impex**

**Egiftcore/resources/egiftcore/import/common/delivery-modes\_en.impex**

**NOTE:-**

To display the delivery mode on front-end we have to make the **net** as to be **false** to know the Jsp condition (gross=true).