

eEyewear B2B Store

Technical Design Documentation

Document History

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| version | |  | Updated By | | |  | Date | | |  | | Comments | | | |
| 1.0 | |  | Ravi Ranjan Kumar Pandey  Madhu Sudhan  Adityakumar Patel  Arfaat Afzal Kirkire  Ravleen Kaur  Shalini Gupta | | |  | 20th Feb,2017 | | |  | |  | | | |
|  | |  | | |  | | | |  | | | | |  | | |
| Document Sign-Off | |  | | |  | | | |  | | | | |  | | |
| Name |  | | | Role | | | |  | | | Signature/Date | | | |
| Gyaneshwar |  | | | Manager | | | |  | | |  | | | |
|  |  | | |  | | | |  | | |  | | | |
|  | |  | | | | | |  | | | | |  | | | | |

|  |  |  |
| --- | --- | --- |
|  | **INDEX** |  |
| **S.No** | **Content** | **PageNo** |
| 1 | Installation Guide | 4 |
| 1.1 | Purpose | 4 |
| 1.2 | Intended Audience. | 4 |
| 1.3 | Document Scope | 4 |
| 1.4 | Functional Rules | 4 |
| 1.5 | Assumptions/Restrictions | 4 |
| 1.6 | Introduction | 5 |
| 1.7 | System Requirements | 5 |
| 1.8 | Hybris Commerce B2B Accelerator | 5 |
| 1.9 | Key Features | 7 |
| 1.10 | List of specific B2B extensions and addons | 7 |
| 1.11 | Build Hybris Commerce | 9 |
| 1.12 | Application Design Diagrams | 13 |
| 2 | Customizing eEyewear B2B Store | 15 |
| 2.1 | eEyewear B2B Site Setup | 15 |
| 2.2 | Rename project folders | 19 |
| 2.3 | Creating eEyewear B2B store | 19 |
| 2.4 | Products in Productcockpit | 24 |
| 2.5 | eEyewear B2B Early Login | 25 |
| 2.6 | Installing Secureportaladdon | 26 |
| 2.7 | Variants | 27 |
| 2.8 | Customized promotions | 29 |
| 2.9 | Vouchers | 32 |
| 2.10 | Implementation for customize PDP to define new attributes | 36 |
| 2.11 | Implementation for customize productcockpit with new PDP attributes | 39 |
| 2.12 | Implementation for Customize CSCockpit with new customer attributes. | 42 |
| 2.13 | Implementation for coming soon products | 43 |
| 2.14 | Implementation for Hot Folder | 45 |
| 2.15 | Implemetation for custom Shipping Method | 46 |
| 2.16 | Implementation for Pickup in store | 47 |
| 2.17 | WCMS Cockpit Customization | 50 |
| 2.18 | define new attributes in Registration Page | 56 |

1. Introduction

Purpose

This document is the Technical Design Document which provides information of custom B2B store setup.

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 B2B Store in Hybris.

Document Scope

This document covers the details about the sprint 1 EeyewearB2B Store.

Functional Rules

Eeyewear B2B Store is an online store which allows other organisation to order products for their business 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 B2B 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.

**1.7 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 fulfill 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 |

**Software:** Java 8.0

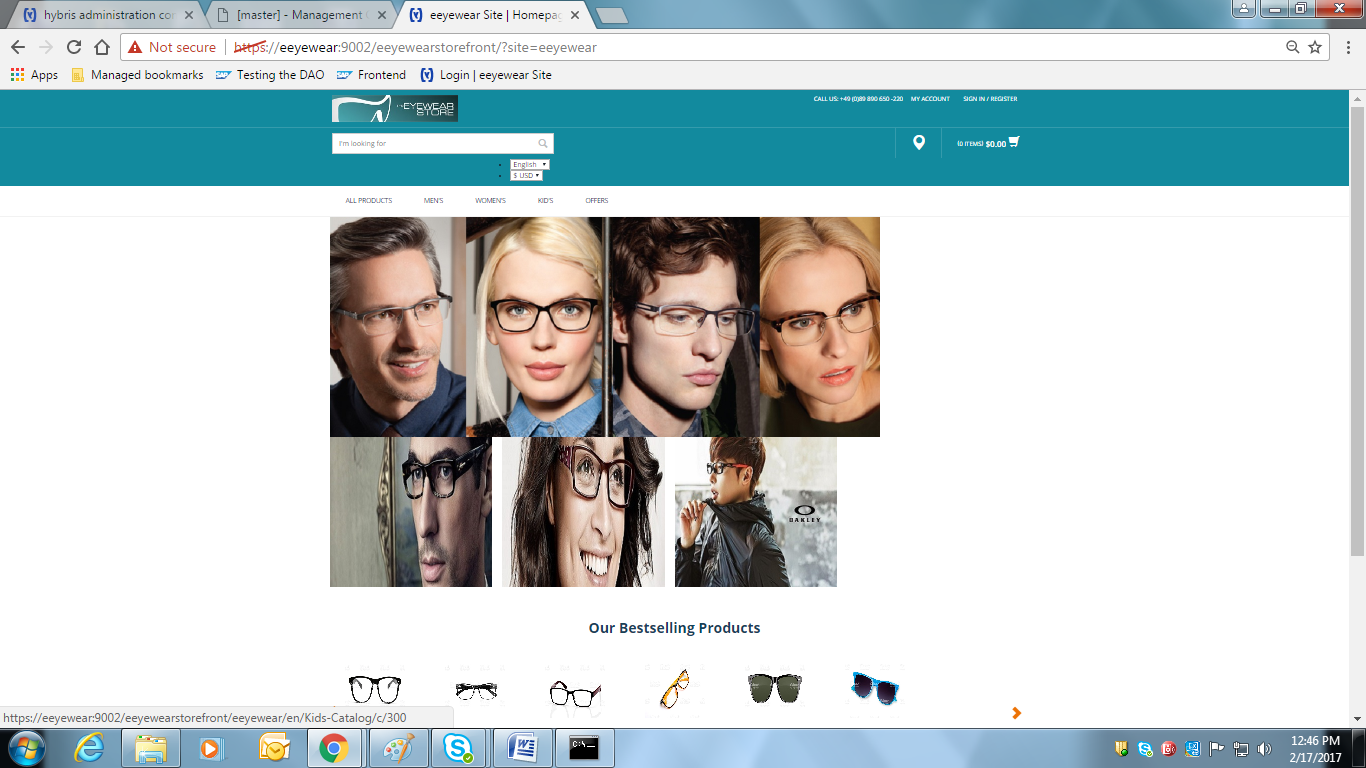
**Server:** Tomcat 7.0

**1.8 Hybris Commerce B2B Accelerator**

The hybris Commerce B2B Accelerator allows B2B organizations to rapidly deploy a best-practice B2B omni-channel commerce solution. Personalized to the unique needs of B2B organizations, the solution delivers all the capabilities you need to manage complex B2B relationships.

It automates order processing and provides B2B organizations with the ability to manage numerous B2B accounts on a single platform.  It also includes advanced self-service capabilities for the B2B customers where they can manage their users, user groups, permissions, cost centers, budgets, and approval workflows. B2B account managers also benefit from the B2B Admin Perspective in the Administration Cockpit where they can use one wizard to create an organization and another to manage the quote approval process.

The following screenshot is an example of the B2B Eeyewearb2b home page.



For B2B customers, it is also a rich customer experience. 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 B2B 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.

Customer administrators can completely control their organization based on a structure they set up that reflects the company size and complexity. Whenever B2B customers send their approval request, the company approvers are able to easily find and manage orders that need to be approved. It is also possible to select the fully integrated merchandising tools such as promotions, cross-sells, up-sells, and advanced personalization.

**1.9 Key Features**

* **Back-office integration.**
* **Self-service account management**. B2B administrators assign rights to employees, set up individual purchasing limits, configure approval rules and workflows, and manage cost centers and budgets.
* **Online ordering.** Users can place orders online and schedule orders that are automatically replenished. Orders are subject to internal and merchant order approval. Orders can be tracked online.
* **Custom catalogs**. Merchants can customize product lists and search product pages for specific organizations, departments, and customers.
* **Custom pricing.** Complex B2B pricing models are supported, which are managed either in hybris or in an ERP system.
* **Price quote requests.** Purchasers can negotiate the price of an order. Quotes requests can be restricted to a minimum order value.
* **Credit management.** Merchants can assign multiple credit limits with a variety of currency and date range options.
* **Multi-dimensional products.** Products can be defined as having multiple attributes, such as color, size and fit.
* **Advanced product search.** Searches can be performed using bar code or product code, and by text or delimited list.
* **PunchOut support.**The B2B site can be integrated with customer procurement systems. Orders are automatically translated into purchase requisitions that are saved to the customer's procurement system.
* **Early Login.** Users can be required to log in before being able to view the storefront.

**1.10 List of B2B –Specific Extensions and Add-ons**

|  |  |  |
| --- | --- | --- |
| **B2B Feature** | **Related Extensions and Add-ons** | **Add-on Definition** |
| Account Summary | [accountsummaryaddon AddOn](https://wiki.hybris.com/display/accdoc/accountsummaryaddon+AddOn) | This AddOn allows customers and merchants to view financial histories. |
| API for OCC | [b2boccaddon AddOn](https://wiki.hybris.com/display/release5/b2boccaddon+AddOn) | This AddOn is responsible for providing B2B-specific API resources. |
| B2B Accelerator AddOn | [b2bacceleratoraddon AddOn](https://wiki.hybris.com/display/accdoc/b2bacceleratoraddon+AddOn) | The **b2bacceleratoraddon** extends a target storefront with B2B-specific functionality. |
| AddOnBusiness Logic | [b2bacceleratorfacades Extension - Technical Guide](https://wiki.hybris.com/display/accdoc/b2bacceleratorfacades+Extension) [b2bacceleratorservices Extension](https://wiki.hybris.com/display/accdoc/b2bacceleratorservices+Extension) | The hybris**b2bacceleratorfacades**extension provides a number of facades that are backed by the services in the**b2bcommerce** extension.  The hybris**b2bacceleratorservices**extension organizes functionality from one or more B2B commerce services related to the company and user management, orders and carts, and workflow-related actions and events |
| Commerce Organization | [commerceorgaddon AddOn](https://wiki.hybris.com/display/accdoc/commerceorgaddon+AddOn) | The commerceorgaddon AddOn extends a target storefront with Organization Management B2B-specific functionality. |
| Early Login | [secureportaladdon AddOn](https://wiki.hybris.com/display/accdoc/secureportaladdon+AddOn) | This AddOn requires users to log in before being able to view the B2B Accelerator site content. Users can also register for a new account. |
| PunchOut | [b2bpunchout Extension](https://wiki.hybris.com/display/accdoc/b2bpunchout+Extension) [b2bpunchoutaddon AddOn](https://wiki.hybris.com/display/accdoc/b2bpunchoutaddon+AddOn) | The hybris **b2bpunchout** extension implements the cXML protocol to process cXML requests and produce response objects.  b2bpunchoutaddon is responsible for PunchOut-related HTTP requests from the procurement system as well as PunchOut-specific customizations to B2B views and functionality. |
| Sample Data | [powertoolsstore Extension](https://wiki.hybris.com/display/accdoc/powertoolsstore+Extension) | The hybris **powertoolsstore**extension adds the necessary dataset for a reference hybris B2B Commerce Accelerator site and a power tools product catalog. |
| Saved Order Forms | [savedorderforms Extension](https://wiki.hybris.com/display/accdoc/savedorderforms+Extension) [savedorderformsoccaddon AddOn](https://wiki.hybris.com/display/accdoc/savedorderformsoccaddon+AddOn) | The Saved Order Forms extension (**savedorderforms**) provides the core functionality for the Saved Order Forms feature.  saveorderformoccaddon provides the REST API for the Saved Order Forms functionality. |

**1.11 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 6.0, 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.

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 b2b\_acc (OR) install.bat -r b2b\_acc\_plus**

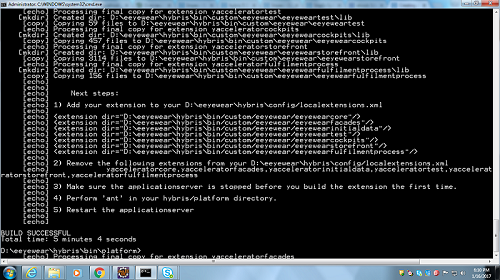
- To generate config, data, log, roles and temp folders

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=eeyewearb2b -Dinput.package=com.eeyewearb2b -Dinput.template=develop**

****

**We have to give the above mentioned entry in to localextensions.xml file.**

**For that,**

* + 1. **edit <<Hybris-commerce-suite-6.0 HOME>>\installer\recipes\b2c\_acc\build.gradle (OR) b2c\_acc\_plus\build.gradle (replace yaccelerator by eeyewear Extensions and add additional extensions to the respective gradle)**

**extensions {**

**...**

**extName 'eeyewearb2bcockpits'**

**extName 'eeyewearb2binitialdata'**

**extName 'eeyewearb2bfulfilmentprocess'**

**extName 'eeyewearb2bstorefront'**

**extName 'eeyewearb2bcore'**

**extName 'eeyewearb2bfacades'**

**extName 'eeyewearb2btest'**

**...**

**}**

**pl.project.addons {**

**...**

**template "eeyewearb2bstorefront"**

**storeFronts "eeyewearb2bstorefront"**

**...**

**}**

* + 1. Navigate to the {HYBRIS\_HOME}/installer directory

**install.bat -r b2b\_acc (OR) install.bat -r b2b\_acc\_plus**

5. platform:

**Install all the addons as per the recipe you have choosen: (Sample below)**

**---------------------------------------------------------**

**ant addoninstall -Daddonnames="captchaaddon,liveeditaddon" -DaddonStorefront.eeyewearb2bstorefront ="eeyewearb2bstorefront"**

**ant clean all**

* To set up eeyewear Store

1. To Initialize system execute the below commands
2. platform: **ant** **initialize** command

(OR)

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

(OR)

1. Navigate to the {HYBRIS\_HOME}/installer directory.

**install.bat -r b2b\_acc initialize** **(OR) install.bat -r b2b\_acc\_plus initialize**

1. Start Hybris Server
2. platform**: hybrisserver.bat**

(OR)

1. installer: **install.bat -r b2b\_acc start (OR) install.bat -r b2b\_acc\_plus start**
2. Check the default electronics store

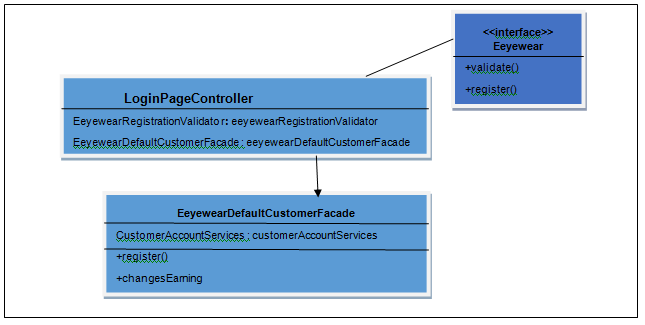
[https://localhost:9002/eeyewearb2bstorefront/?site=electronics](https://localhost:9002/epsalesb2bstorefront/?site=electronics)

http://localhost:9001/eeyewearb2bstorefront/?site=electronics

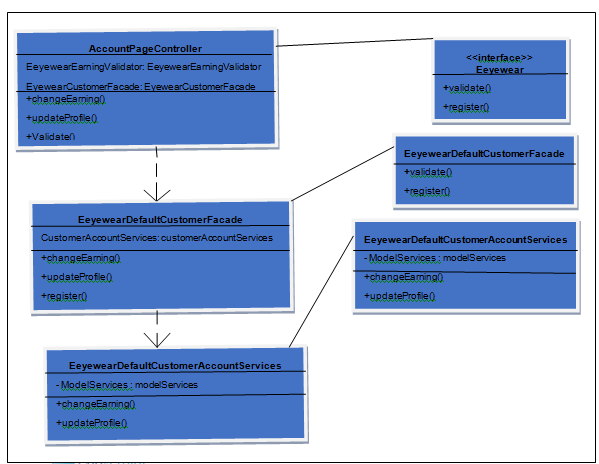
From above steps we will finish our custom store setup.

**1.12 Application Design Diagrams:-**

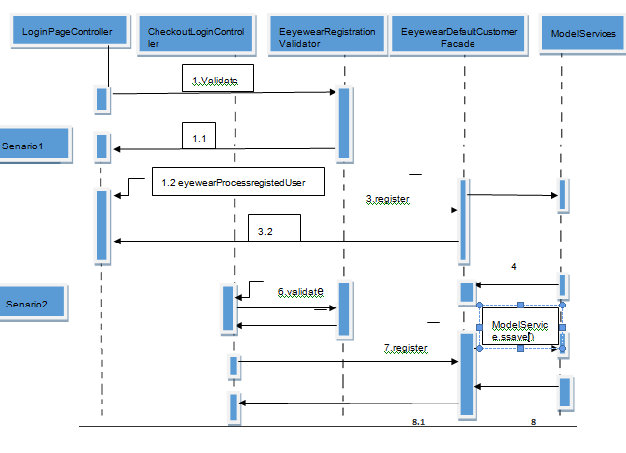
Class Diagram: Registration



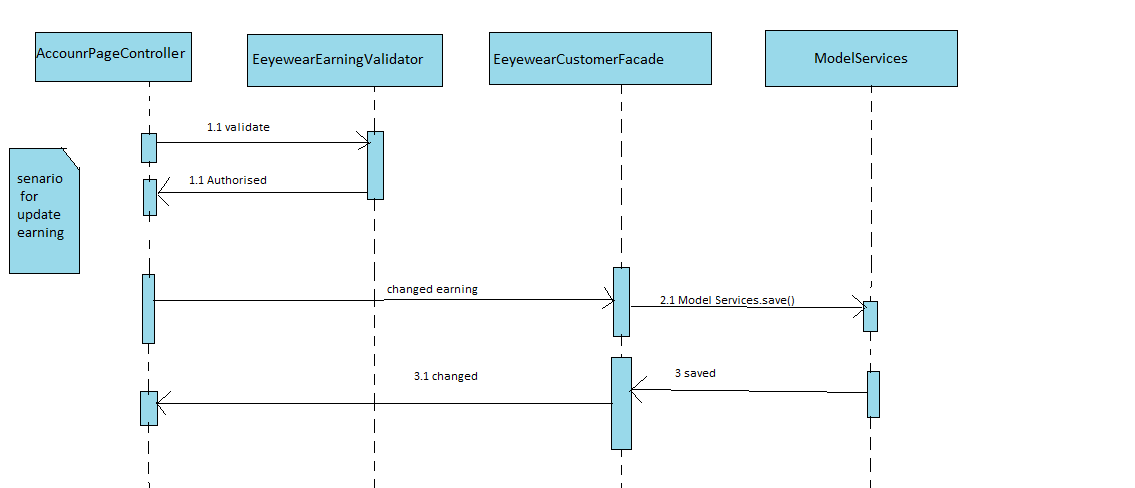
Class Diagram: My Account



**Registration Sequence Diagram:**



**Update Contact Sequence Diagram**



**2.1 Eeyewear B2B Site Setup**

**Customize Initialization Process**

One of the goals of the initialization process is to prepare the data for your store.  
The accelerator provides two dedicated initialization hooks that allow you to import your custom data.

**Core data hook** - in our namespace that is the **CoreSystemSetup** class in Eeyewearcore extension.

**Sample data hook** - in our namespace that is the **InitialDataSystemSetup** class in Eeyewearinitialdata extension.

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

#### Eeyewear Core Data Import Service

By design this is meant to provide the Eeyewear 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 (ie: catalog definition, email templates, site definition)

**Sample Data Import Service**

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

Next steps:

* Modify CoreSystemSetup to include our new shop's basic data during project data initialization.
* Modify InitialDataSystemSetup to include our products, cms-content during project data initialization.

**Review the Core System Setup**

This is where all the non store-specific data is loaded, if the extension specified in **EeyewearCoreConstants** is enabled for the current tenant.  In the event that additional core system data is required for your application, an***importImpexFile****()* call specifying the desired impex file would have to be added to the ***createProjectData****()* method.

**Modifying Eeyewear InitialDataSystemSetup**

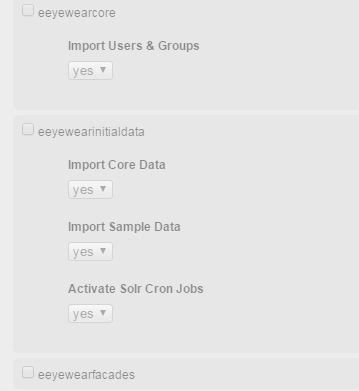
Look at the InitialDataSetup class, and modify the **createProjectData()** method to import our store data when the server is initialized or updated, and the **Eeyewearcore** project data checkbox is selected.Note that the CoreDataImportService.execute() method will use the provided catalog name to construct the names of the impex files it will try to import.



In the code above, the createProjectData() method delegates the import operation to the CoreDataImportService and SampleDataImportService classes. If we wanted to modify the files being imported, we would extend these classes.

**Calling import of Eeyewearcore** **and Eeyewearinitialdata** **data**

Build the code base and restart the hybris server. Go to the system update panel in the admin console. To import our project data, you need to tick two checkboxes in the project data section (Eeyewearcore and Eeyewearinitialdata). Please note the new initialization option for initializing our shop.

****

The screenshot shows how the update page should look like. At this point the system update would have no effect and if you try it you'll get errors since we have not provided the necessary impex files yet.

**Impex-Basic Store Data**

The basic store data is loaded in the **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata** directory and loaded via the Core Data Import Service class. We should review the import All Data method of the Abstract Data Import Service as well as the overridden import Common Data, import Product Catalog, import Content Catalog, import Store and import Solr Index methods of the Core Data Import Service class. For creating product catalog related data we need a resource structure containing following files :

* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/productCatalogs/eeyewearProductCatalog/catalog.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/productCatalogs/eeyewearProductCatalog/catalog\_en.impex**

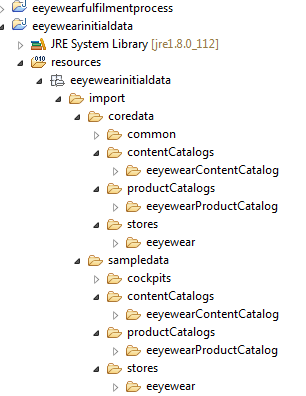
For creation of a **content catalog** related data we need a resource structure containing following files :

* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/catalog.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/catalog\_en.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/cms-content.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/cms-content\_en.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/email-content.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/email-content\_en.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/cms-mobile-content.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/contentCatalogs/eeyewearContentCatalog/cms-mobile-content\_en.impex**

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

* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/store.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/store\_en.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/site.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/site\_en.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/solr.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/solr\_en.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/solrtrigger.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/site-responsive.impex**
* **eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/eeyewear/store-responsive.impex**

**2.2 Rename the project folders**



**2.3 Creating EeyewearB2B Store**

The tables below show the impex files that has to be changed in the coredata for creating the store as well as sampledata impexes in order to populate data in our EeyewearB2B Store.

**Impex files under Coredata .**

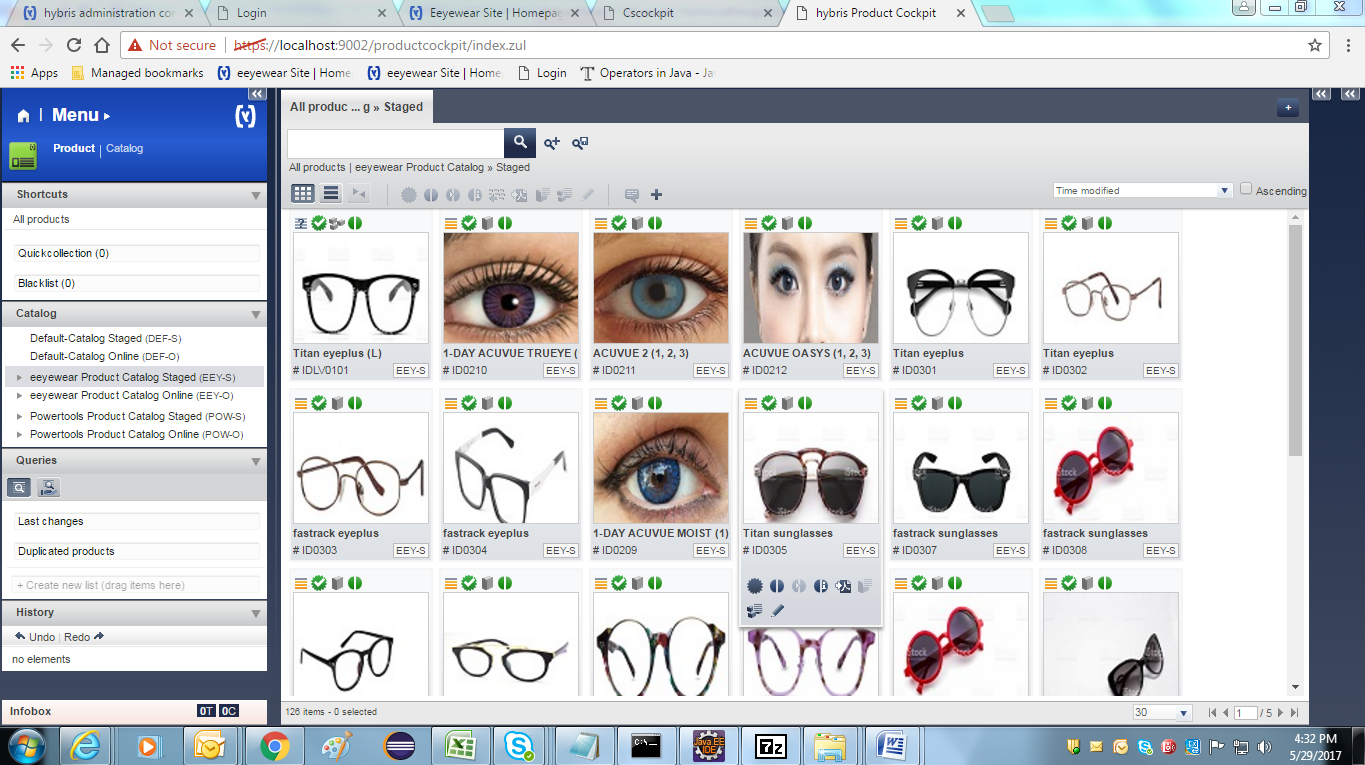
|  |  |  |
| --- | --- | --- |
| **IMPEX FILE NAME** | **IMPEX FILE** | **DESCRIPTION** |
| **store.impex** |  | This file provides shipping methods and different values for currencies. |
| **store\_en.impex** |  | This file adds English localized attribute values for the basic store items |
| **site.impex** |  | This file adds basic site information to the configuration of the Eeyewearb2b shop |
| **site\_en.impex** |  | Provide English-language localization for the Eeyewearb2b CMSSite |
| **solr.impex** |  | Updates the basic solr configuration, including information for ranges, facets, index names, etc... |
| **solr\_en.impex** |  | For the localization of the solr file. |
| **solrtrigger.impex** |  | This file is for  the Solr indexing triggers. |
| **site-responsive.impex** |  | To import the Responsive CMS Site configuration |
| **store-responsive.impex** |  | To create the base store. |
| **catalog.impex(productCatalog)** |  | To import the Product Catalog and Classification Catalog |
| **catalog\_en.impex(productCatalog)** |  | To provide the localization of catalog.impex. |
| **catalog.impex(contentCatalog)** |  | To import the content catalog. |
| **catalog\_en.impex(contentCatalog)** |  | Provides the localization of content catalog. |
| **cms-content.impex** |  | The file defines all of the content slots for the templates, naming the templates. It does not rely on any specific product/category content. |
| **cms-content\_en.impex** |  | Provides the  the related localized attributes values of cms-content.impex. |
| **email-content.impex** |  | The file updates the email notification layouts |
| **email-content\_en.impex** |  | The file populates the email localized content. |

**Impex files under Sampledata for populating data in store:**

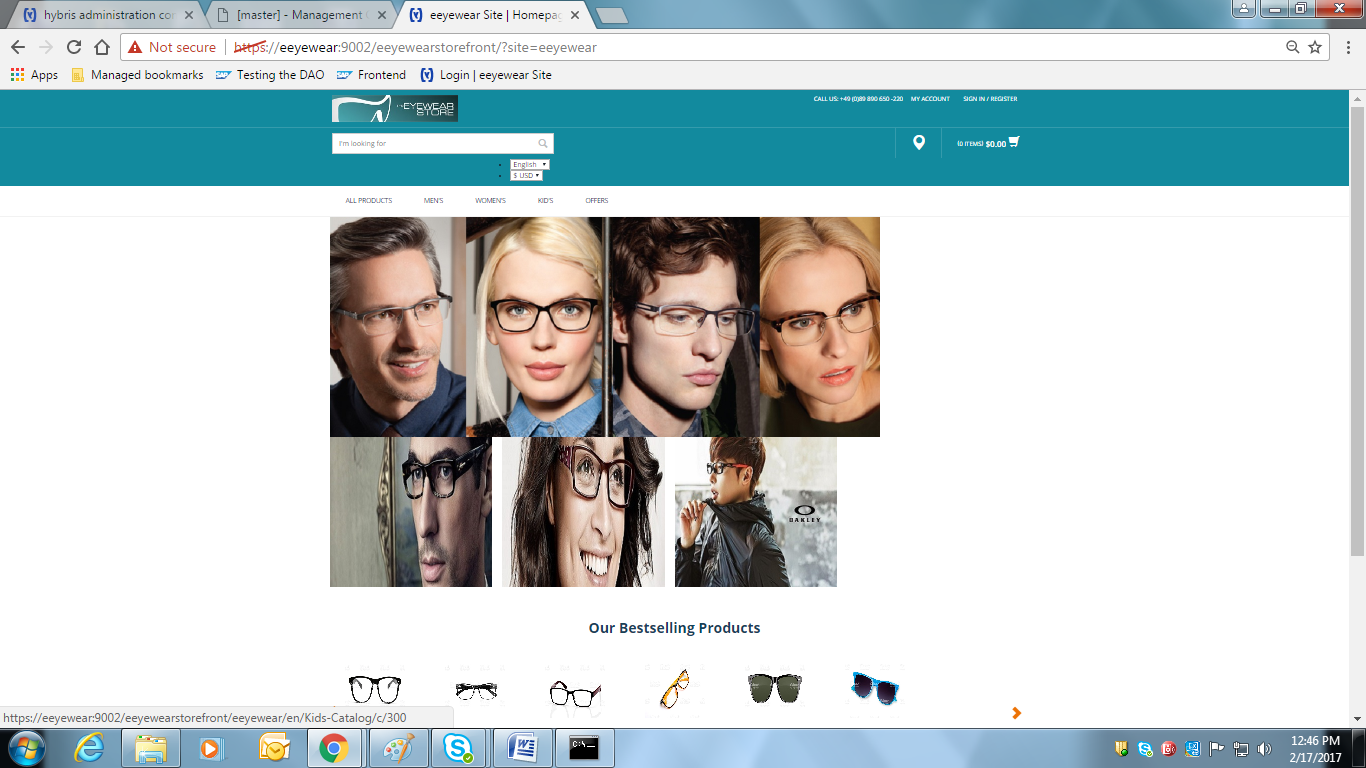
|  |  |  |
| --- | --- | --- |
| **IMPEX FILE NAME** | **IMPEX FILE** | **DESCRIPTION** |
| **categories.impex** |  | This file provides the categories of the products in our site. |
| **categories\_en.impex** |  | Provides the localization for the categories file. |
| **categories-media.impex** |  | To import the category media files. |
| **suppliers.impex** |  | Provides the suppliers for different products in our site. |
| **suppliers\_en.impex** |  | Provides the localization for the suppliers file. |
| **products.impex** |  | To update the products, product codes, manufacturer name, variant type etc in the site. |
| **products\_en.impex** |  | Provides the localization for the products file. |
| **products-media.impex** |  | To update the media, media format and the folder in which the media is present. |
| **products-prices.impex** |  | To update the product prices for the different products. |
| **products-relations.impex** |  | To update the related products using product references, catalog version etc. |
| **products-stocklevels.impex** |  | To update the stock levels of each product in the warehouse. |
| **warehouses.impex** |  | Creates the warehouses for the store. |
| **cms-content.impex** |  | To update the cms contents, site logo,cms- link components etc. |
| **cms-content\_en.impex** |  | Provides localization for the cms-content file. |

**2.4 Products in Product Cockpit:**

After inserting the products in the impex files under the sample data folder, we update the eeyewearinitialdata from the HAC. Having initialized the data, we can visit the product cockpit page to review the newly created products as shown below.



The cms-content impex data will create the header links and footer links and homepage of our store with logo of our store. The screenshot of EeyewearB2B storefront is given below.



**2.5 Eeyewear B2B Early Login**

This is one the feature of B2B store. It can be achieved using the **secureportaladdon**extension. This addon brings security to the store by allowing only authenticate customers to view the store. The addon also provides an registration link for creating new customers.

**Add On Definition:**

|  |  |
| --- | --- |
| **Name** | B2B Early Login AddOn (secureportaladdon) |
| **Description** | This AddOn requires users to log in before being able to view the B2B Accelerator site content. Users can also register for a new account. |
| **Requires** | [addonsupport](https://wiki.hybris.com/display/accdoc/addonsupport+Extension+-+Technical+Guide) [acceleratorservices](https://wiki.hybris.com/display/accdoc/acceleratorservices+Extension+-+Technical+Guide) [b2bacceleratorfacades](https://wiki.hybris.com/display/accdoc/b2bacceleratorfacades+Extension) [hmc](https://wiki.hybris.com/display/release5/hmc+Extension) [cms2](https://wiki.hybris.com/display/release5/cms2+Extension+-+Technical+Guide) [cms2lib](https://wiki.hybris.com/display/release5/cms2lib+Extension+-+Technical+Guide) [cockpit](https://wiki.hybris.com/display/release5/cockpit+Extension+-+Technical+Guide) |

**secureportaladdon AddOn**

The B2B Early Login AddOn (**secureportaladdon**) provides the following functionality:

* **Early login**: Users are required to authenticate before being able to browse the site. Password recovery is provided from the early login page that is displayed when a user tries to access the site and this feature is enabled.
* **Registration**: Users can request an account. The link to do so appears in two places: in the early login page, and in the header, if the user is not logged in.  The new account is approved by a merchant user who is part of the b2b registration approver group.

**2.6 Installing the B2B Secureportaladdon**

The B2B Early Login AddOn is installed using **ant addoninstall command**.

1. Ensure that **secureportaladdon** is listed in config/localextensions.xml



1. Run the following **ant** command from platform cmd-

ant addoninstall-Daddonnames="secureportaladdon"

-DaddonStorefront.yacceleratorstorefront="eeyewearstorefront"

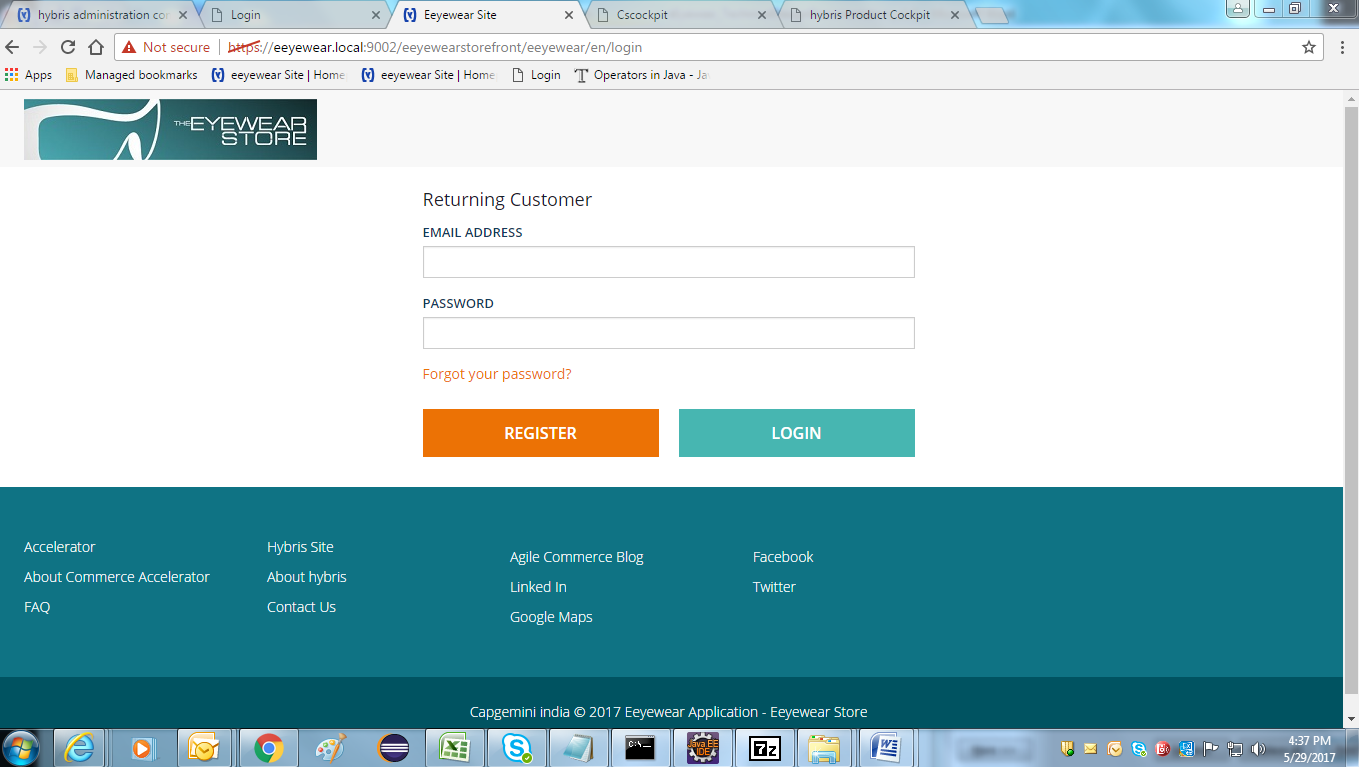
1. Run ant all
2. Refresh eclipse workspace.
3. Open project.properties of secureportal addon and check this line below

eeyewearstorefront.additionalWebSpringConfigs.secureportaladdon=classpath:/secureportaladdon/web/spring/secureportaladdon-web-spring.xml

If this line is not present add the above line into that file.

1. Rebuild the system. In the hybris Administration Console ( [localhost:9001](http://localhost:9001/) ):
2. Select the options **Update running system**, **Create essential data**, and **Localize types**.
3. In the project data settings, select the **secureportaladdon** check box, and set **Sync Content Catalogs** to **yes**.
4. Click **Update**.
5. After update do the synchronization in HMC. Open you custom website.

https://Localhost:9001/eeyewearstorefront/?site=eeyewear



**2.7 Setup for variant products:-**

In the **eeyewearcore-item.xml** add Eeyewear **typegroup** and declare **EeyewearSizeVariantProduct** type which extends **VariantProduct** with attribute size

<typegroup name="Eeyewear">

<itemtype code="EeyewearSizeVariantProduct" extends="VariantProduct"

autocreate="true" generate="true"

jaloclass="com.eeyewear.core.jalo.EeyewearSizeVariantProduct">

<attributes>

<attribute qualifier="bookSize" type="localized:java.lang.String"

metatype="VariantAttributeDescriptor">

<description>No of sizes the book.</description>

<modifiers />

<persistence type="property" />

</attribute>

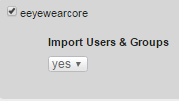
</attributes>

</itemtype>

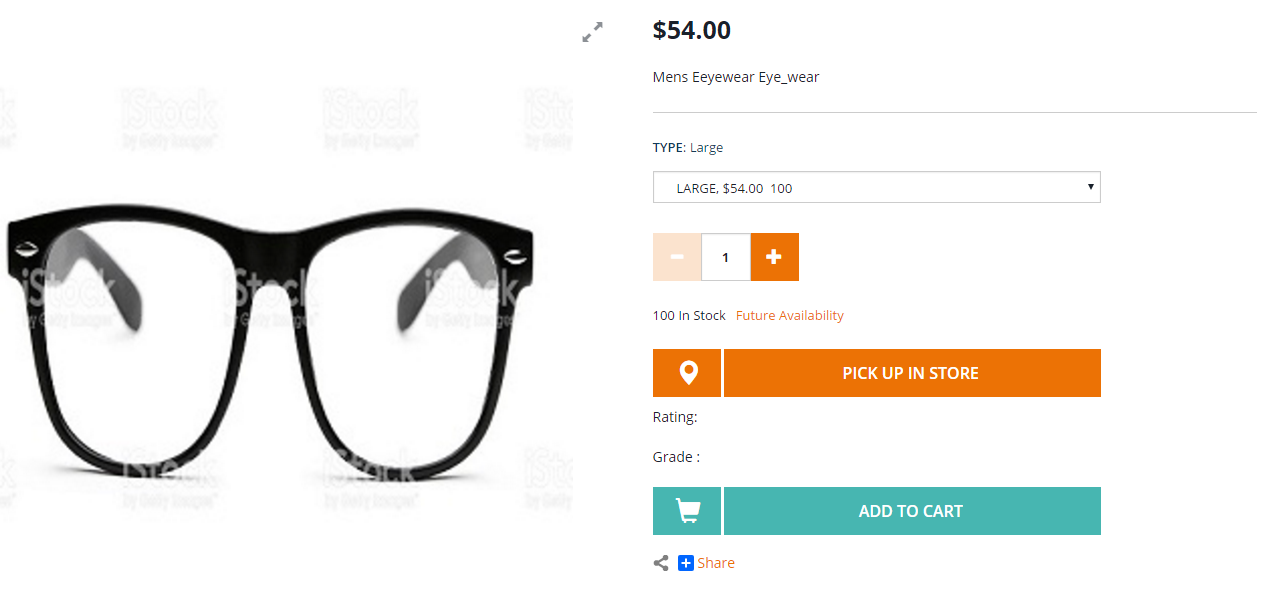
Stop the server & Run the command in Hybris platform path: **ant all**, Generates models, DAO, DTO and Jalo classes are created for above specified item types.

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

Update the system using HAC with the project data with eeyewearcore. It adds the newly created attributes to the database.



**Variants related image**

****

**2.8 Customized Promotion**

**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:

We need to create the eeyewearcommerceservices 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 **eeyewearcommerceservices**
5. Package name **com.eeyewear.eeyewearcommerceservices**

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

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

And run ant **all.**

A promotion type is an entity we need to define in **eeyewearcommerceservices-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**

**eeyewearcommerceservices-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 eeyewearcommerceservices extension and open the generated **ProductQuantityThresholdFreeGiftPromotion** class and add.

**eeyewearcommerceservices/src/de/Hybris/eeyewear/jalo/ProductQuantityThresholdFreeGiftPromotion.java**



package  com.eeyewear.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

 }

}

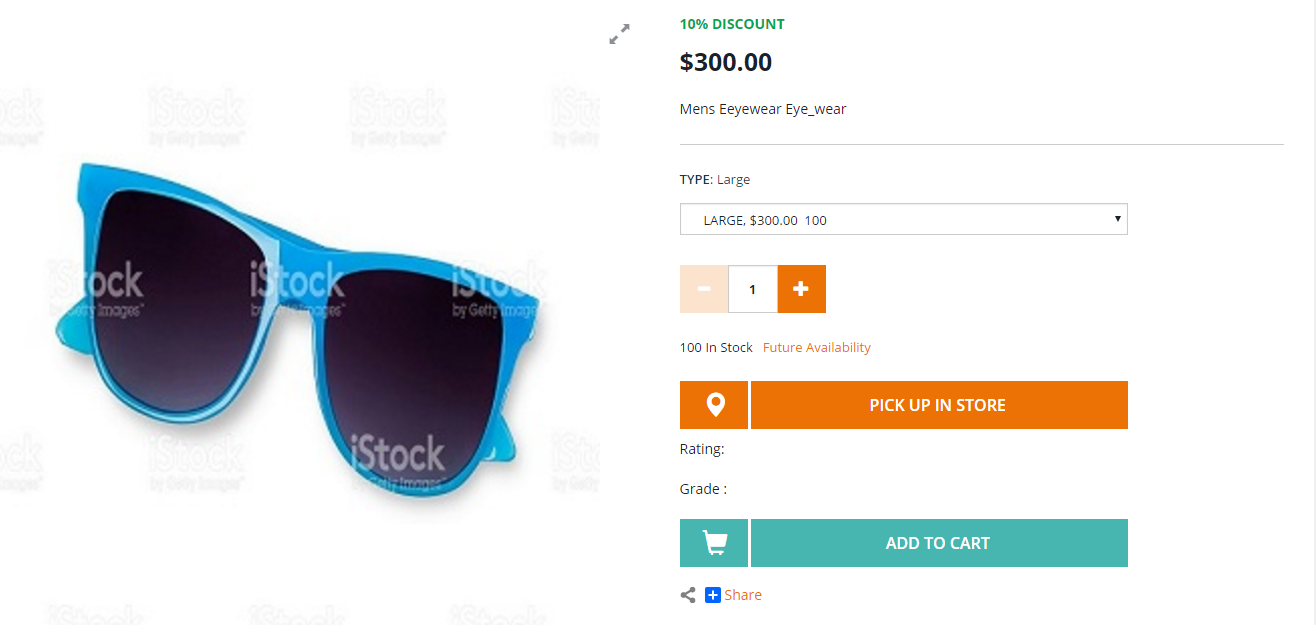
**/eeyewearinitialdata/resources/eeyewearinitialdata/import/sampledata/stores/eeyewear/promotions.impex**

****

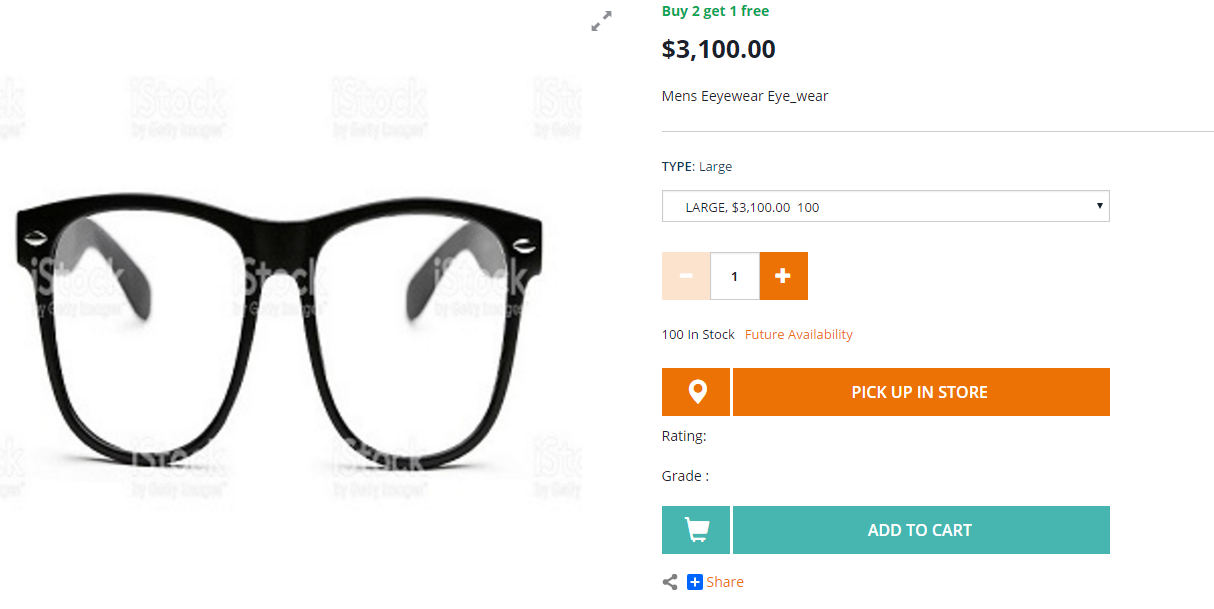
#### Rebuild and Verify

Rebuild the code base and start the server

**10% percentage discount promotion for a particular product**

****

**Buy 2 product’s will get 1 free book:**

****

**2.9 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
2. Serial Voucher

**Serial Voucher:-**

1. Log on to the Backoffice Administration Cockpit.
2. Navigate *Marketing* 🡪 *Coupon Management* .
3. Choose *Coupons* to display a list of default Coupons.
4. Choose  *+*  *Multi-Code Coupon*  to display the *New Multi-Code Coupon* window.
5. In the *Coupon Configuration* tab, enter the appropriate information in the following fields:
   * *Coupon ID*- refers to the identifier of the coupon.
   * *Name* - refers to the localized name that describes the coupon-based campaign.
   * *Number of coupon codes to generate* - refers to the number of coupon codes you want to generate to be used for promotions.
   * *Generated codes* - refers to the coupon codes that are generated in a batch; each coupon code can only be used once.
   * *Code Prefix* - refers to the prefix that is used for generated coupon codes.
   * *Code Generation Configuration* - refers to the configuration that is used along with the prefix to generate coupon codes.
   * *Start Date*- refers to the validity date and time of the coupon.
   * *End Date* - refers to the expiry date and time of the coupon.

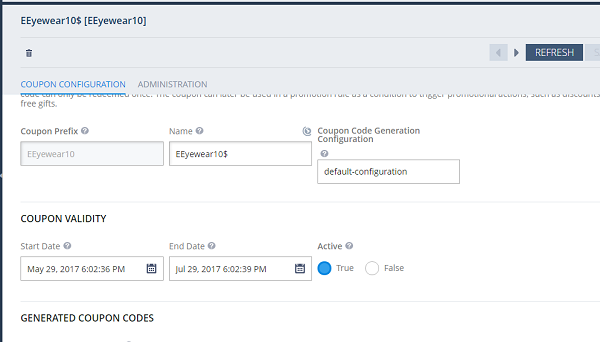
Ensure that you **have not set** the status to *True* in the *Active* field.

1. Select *Save* to save all the changes. This automatically adds the coupon to the list of available Coupons. Otherwise, select *Refresh*to undo the changes made.
2. Create a Promotion Rule and ensure that the Promotion Rule is **published**.

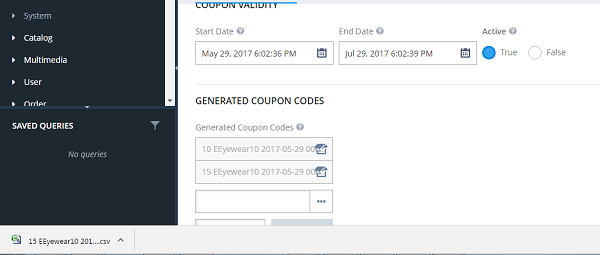
Once your Promotion Rule is created and published, perform the succeeding steps to activate your Multi-Code Coupon.

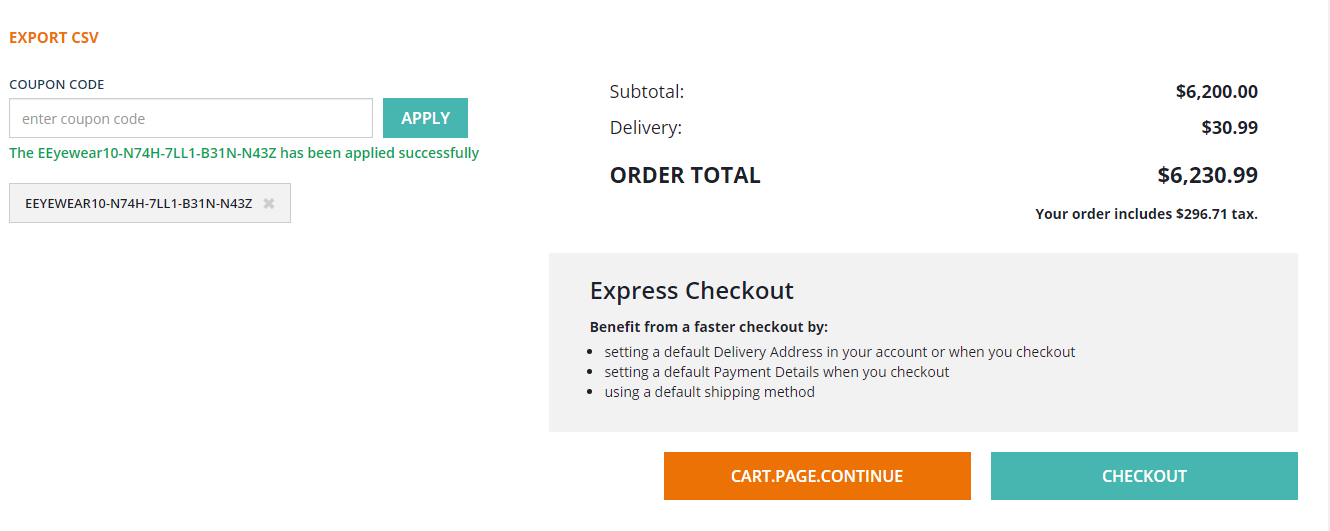
1. Specify the number of coupon codes you want to generate in the field provided. Then click the *Generate* button.
2. All generated coupon codes are then available for you to download in the *Generated codes* field. Click the *Download* button to download the coupon codes as a CSV file.
3. In the  *Coupon Configuration*  🡪 *Active*   field, set the status to *True*. Setting the status to *True* activates the coupon and makes it usable in the promotion.

**Note:** You can start distributing and redeeming the generated coupons on the storefront.



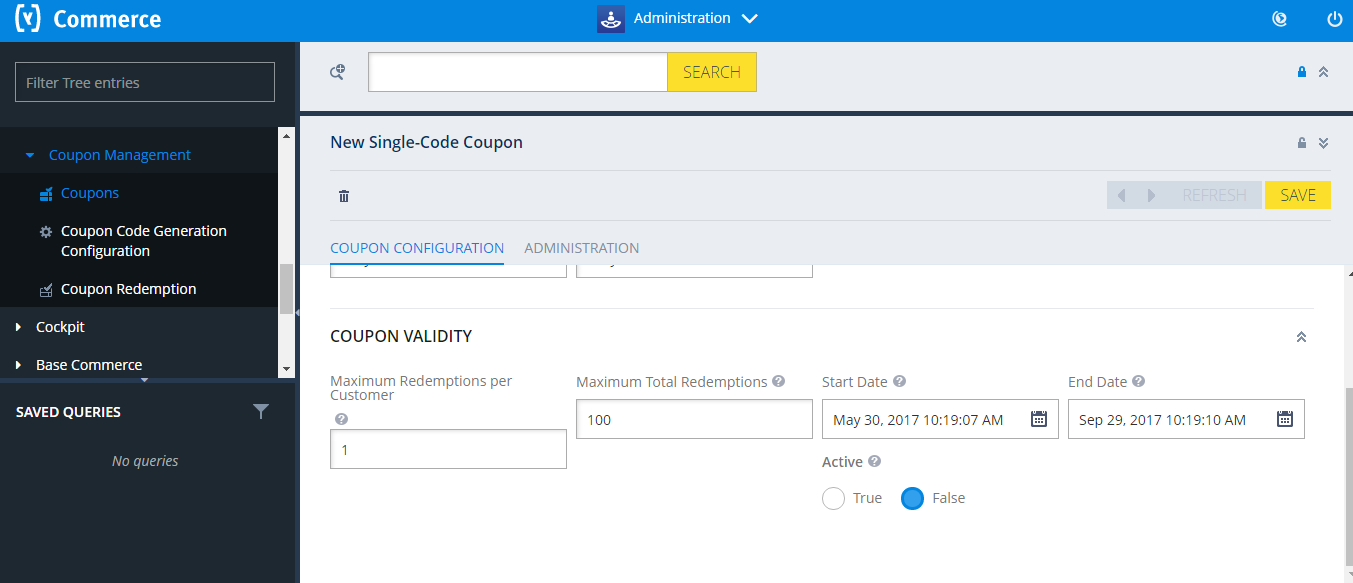
1. If you wish to generate additional coupon codes, enter the number of coupon codes you wish to generate in the *Number of coupon codes to generate* field. Then, click the *Generate* button.





**Serial Voucher :-**

1. Log on to the Backoffice Administration Cockpit.
2. Navigate to 🡪 *Marketing* 🡪 *Coupons*  to display a list of default Coupons.
3. Choose   *+*   *Single-Code Coupon*  to display the *New Single Code Coupon* window.

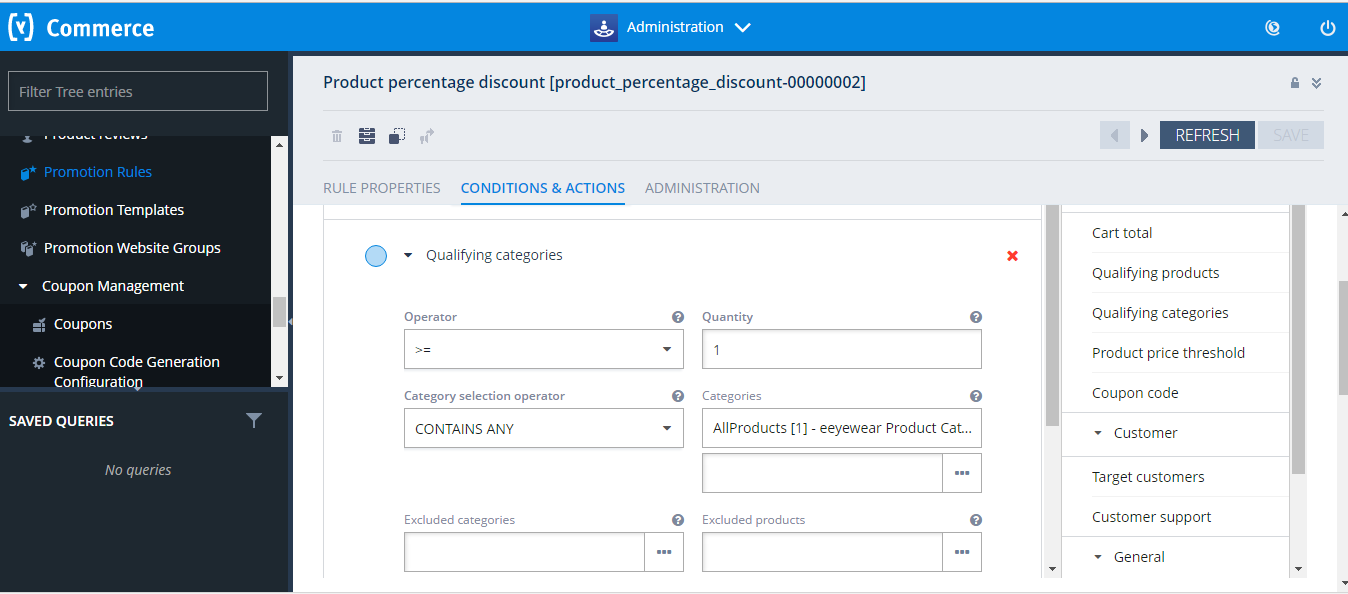


1. In the *Coupon Configuration* tab, enter the appropriate values in the fields provided:
   * *Coupon ID* - refers to the identifier of the coupon.
   * *Name* -refers to the localized name that describes the coupon-based campaign.
   * *Max Redemptions per Customer* - refers to the allowed maximum number of redemptions by one customer for this coupon code.
   * *Maximum Total Redemptions* - refers to the allowed maximum number of redemptions in total for this coupon code.
   * *Start Date* - refers to the validity date and time of the coupon.
   * *End Date* - refers to the expiry date and time of the coupon.

**Note:** Ensure that you **have not set** the status to *True* in the *Active* field.

1. Click *Save* to save the details. This automatically displays the newly created Coupon to the list of available Coupons. Otherwise, select *Refresh* to undo the changes made.
2. Create a Promotion Rule and ensure that the Promotion Rule is **published**.

Once your Promotion Rule is created and published, perform the succeeding steps to activate your Single-Code Coupon.



1. In the  *Coupon Configuration*  *Active*  field, set the status to *True*. Setting the status to *True* activates the coupon and makes it usable in the promotion.

**Note**You can start to distribute and redeem the coupon through the storefront.

## D:\Users\shalingu\Desktop\17.PNG

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



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

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

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

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

<description> **ProductRating.** </description>

<modifiers/>

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

</attribute>

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

<description> **ProductGrade.** </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 🡪 eeyewearcore extension.

1. Define the 2 attributes in ProductData of eeyewear**facades-beans.xml file**

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

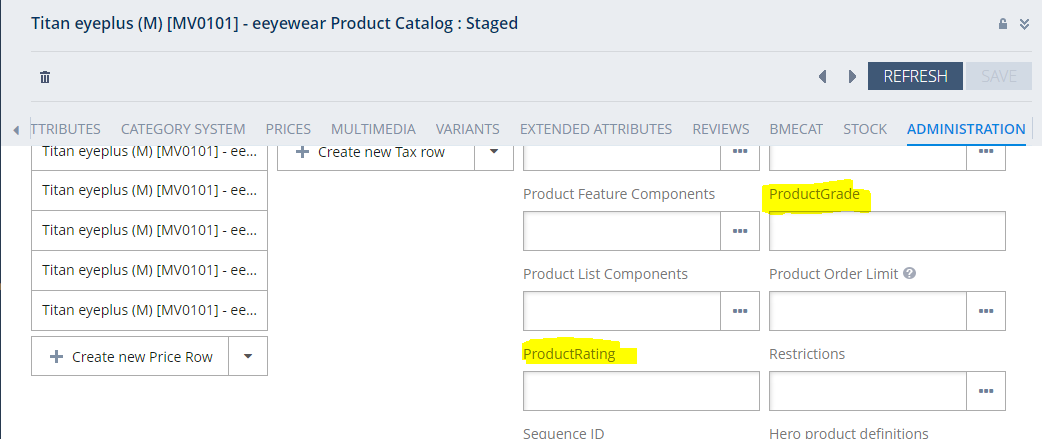
<property name=*"* *productRating "* type=*"java.lang.String"*/>

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

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

After that Go to HMC ----🡪 Platform --🡪 Update 🡪 eeyewearfacdes extension.

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



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



@RequestMapping(value =PRODUCT\_CODE\_PATH\_VARIABLE\_PATTERN, method = RequestMethod.GET)

public String productDetail(@PathVariable("productCode") final String productCode, final Model model,final HttpServletRequest request, final HttpServletResponse response) throws CMSItemNotFoundException,UnsupportedEncodingException

{

final ProductModel productModel = productService.getProductForCode(productCode);

..........

// my changes starts here

final List<String> list = new ArrayList<String>();

list.add(productModel.getProductRating());

list.add(productModel.getProductGrade()); model.addAttribute("attributesList", list);

// end here

...........

..........

}

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"*>

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

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

</div>

</c:if>

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

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

<bean id=*"eeyewearProductAttributesPopulator"* class=*"com.eeyewear.facades.populators.EeyewearProductAttributesPopulator"* parent=*"baseProductPopulator"* >

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

</bean>

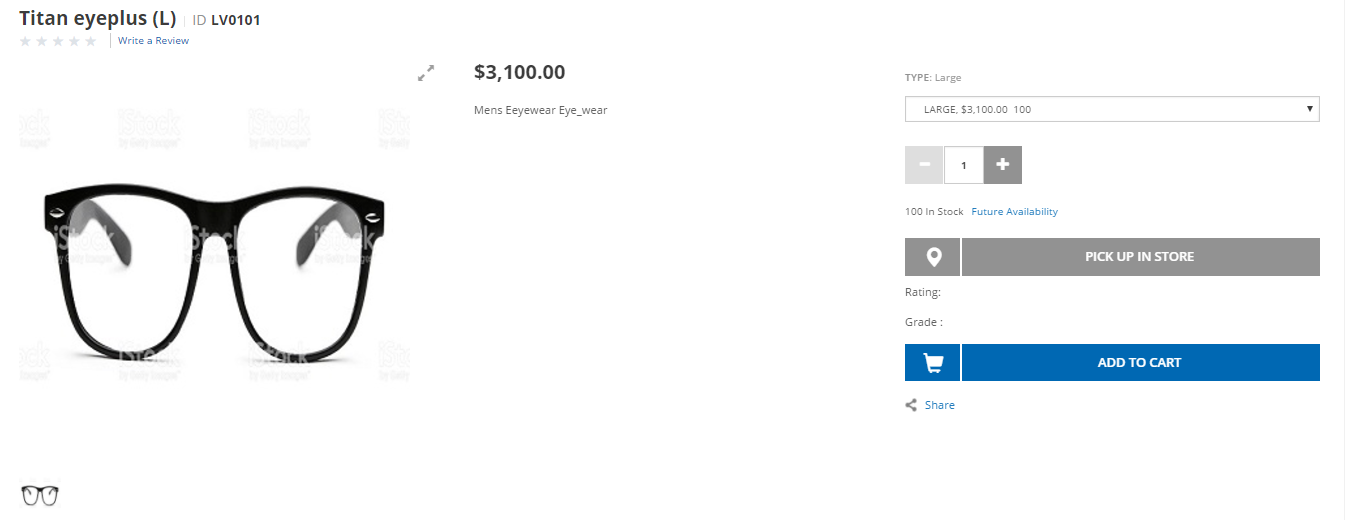
After that do ant all & start the server update eeyewearfacades extension.

6. After that I have created the **EeyewearProductAttributesPopulator.java** file to populate the attributes data.

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



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



**2.11 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 **eeyewearcockpits** extension (\hybris\bin\custom\eeyewear\eeyewearcockpits\resources\eeyewearcockpits-config\cockpitgroup)

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

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

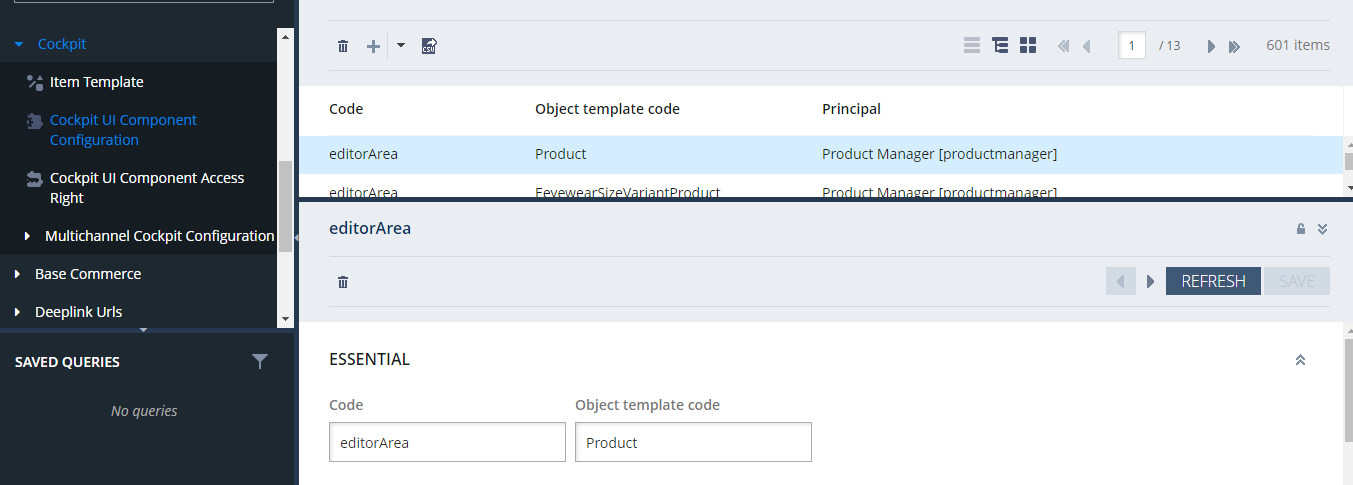
2. Localize those PDP attributes in **commerceservices-locales\_en.properties** file.

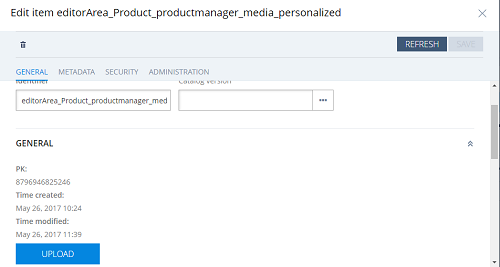
type.Product.productRating.name=Rating

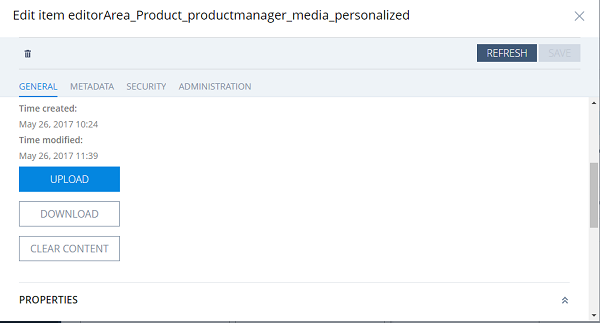
type.Product.productGrade.name=Grade

3. Do ant all & start the server go to backoffice🡪platform -🡪update-🡪eyewearcockpits & 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.







**2.12 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 eeyewearcockpits extension (\hybris\bin\ext-channel\cscockpit\resources\cscockpit\import\config).

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

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

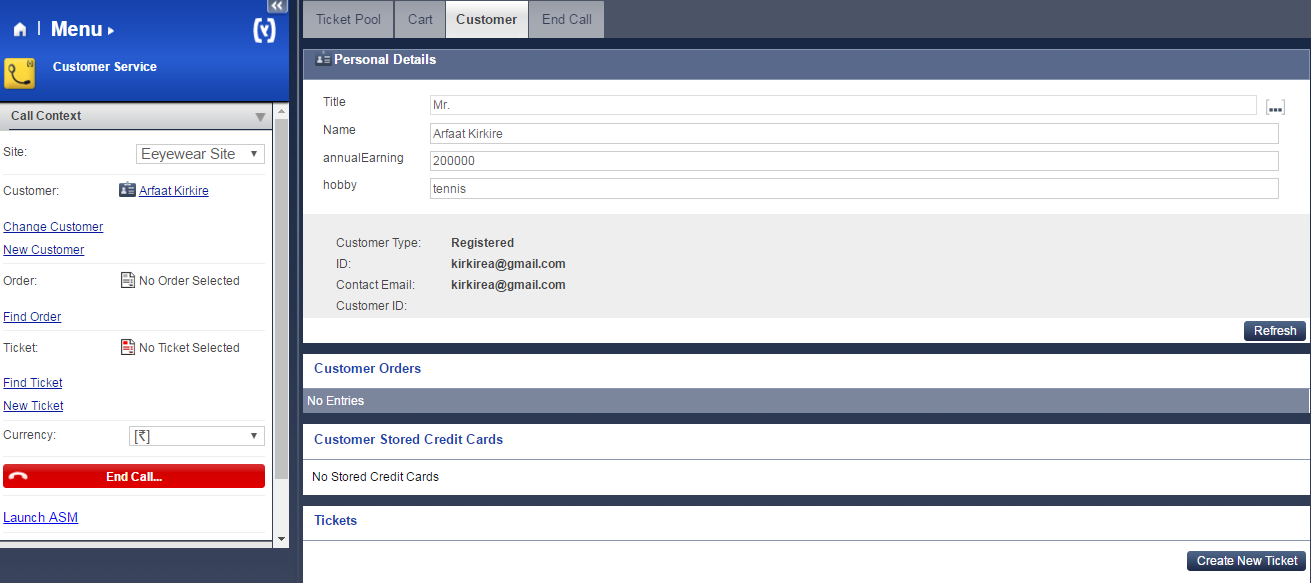
2. Localize those Customer Registration Page attributes in **commerceservices-locales\_en.properties** file.

type.Customer.currentLocation.name=annualEarning

type.Customer.contactNumber.name=hobby

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

4. Go to backoffice🡪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.

****

2.13 Implementation for coming soon products:-



1. Create new attribute “**ComingSoon**” in product itemtype in **eeyewearcore-items.xml** file.

|  |
| --- |
| <itemtype code=*"Product"* extends=*"GenericItem"* autocreate=*"false"* generate=*"false"*>  <attributes>  <attribute qualifier=*"ComingSoon"* 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**

(eeyewearstorefront\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 **eeyewearfacades-beans.xml** file inside ProductData bean

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

<property name=*"genders"* type=*"java.util.List&lt;com.eeyewear.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 **eeyewearcore-spring.xml** file & build the system (**ant clean all**).

<bean id=*"comingSoonDao"*

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

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

</bean>

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

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

</bean>

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

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

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

</bean>

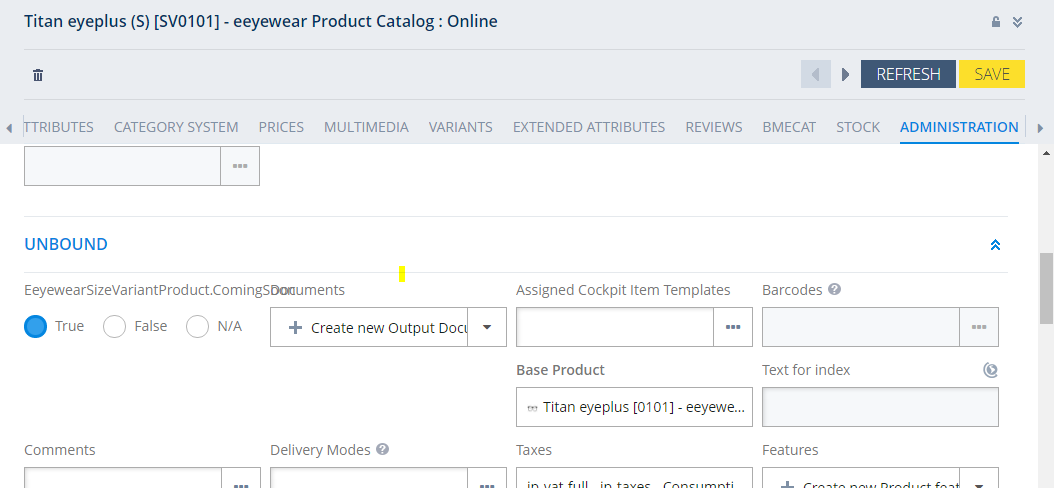
15) Change the controller to get coming soon products from service & set in model.

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**).

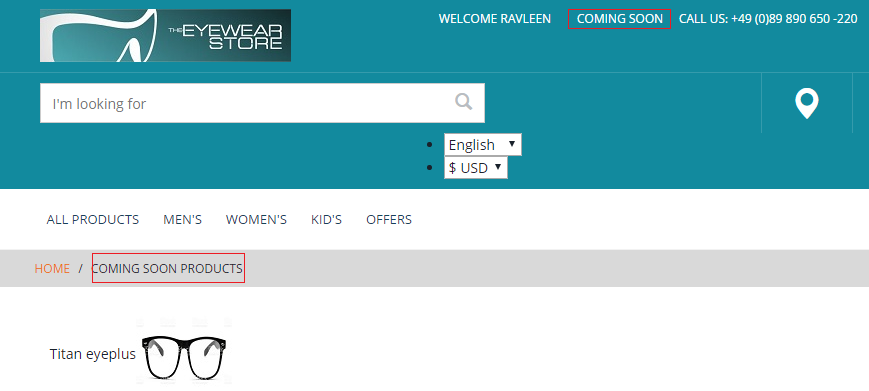
18) Now update all the corresponding extensions to which you have made changes in the above steps .

19) Go to Backoffice & change ComingSoon status **Yes** for few products in online catalog & save.



Now go to your Site and click on **Coming Soon** link.

**Coming Soon Page:**



**2.14 Implementation for Hot Folder:-**

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. It can be implemented using following steps:

1. Copy

**eeyewearcore/resources/eeyewearcore/integration/hot-folder-store-electronics- spring.xml** to

**eeyewearcore/resources/eeyewearcore/integration/hot-folder-store-eeyewear-spring.xml**

Also remember to change the occurrences of electronics to eeyewear and Electronics to Eeyewear.

****

1. Enable the same in spring.

 <!-- Spring Integration -->

<import resource="classpath:/eeyewearcore/integration/hot-folder-store-eeyewear-spring.xml"/>

1. Create the folder (optional, Hybris will automatically create it after restarting )

*Hybris/data/acceleratorservices/import/master/eeyewear*

1. Restart your server to apply the changes made.

**How to Verify the Hot Folder :**

For testing the hot folder we will change the price of any product and see whether the changes are reflected or not .

Download the given csv file and move it into the hot folder



1. If everything goes right the file is renames and moved to archive directory.
2. Now verify the price of the product(id: 0101) in your store it should be changed.

**2.15 Implemetation for custom Shipping Method:-**

Create the new Eeyewear delivery method  "Free Standard Shipping - Within 12 Hours"

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

     ;eeyewear;free-standard-shipping for source and target in store.impex

eeyewearinitialdata\resources\eeyewearinitialdata\import\coredata\stores\eeyewear

INSERT\_UPDATE BaseStore2DeliveryModeRel;source(uid)[unique=true];target(code)[unique=true]  
;eeyewear;premium-net  
;eeyewear;standard-net  
;eeyewear;premium-gross  
;eeyewear;standard-gross  
;eeyewear;pickup  
;eeyewear;free-standard-shipping

2)  free-standard-shipping delivery mode for Zone Delivery Modes, Delivery cost for every zone in delivery-modes.impex

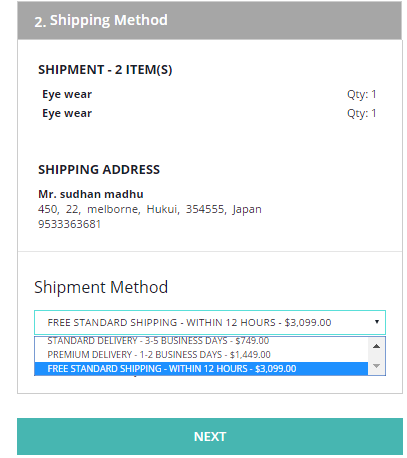
3)   We have to give localized name in delivery-modes\_en.impex

  UPDATE ZoneDeliveryMode;code[unique=true];name[lang=$lang]

;description[lang=$lang];free-standard-shipping;

"Free Standard Shipping";"Within 12 Hours"



****

**2.16 Implementation for Pickup in store:-**

BUY\_AND\_COLLECT enables a process in the yacceleratorstorefront (or our eeyewearstore) extension wherein 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. The PickupInStoreMode mode has already been set to BUY\_AND\_COLLECT on the base store. Here is what the file looks like (we don't need to modify it):

**eeyewearinitialdata/resources/eeyewearinitialdata/import/coredata/stores/hybris/store.impex**



2. Add one or more PointsOfService of type STORE. These PointOfService will also appear in the Store Locator.



Note that we link a Warehouse to each PointOfService that offers the Pickup From Store service.

3. Define the **hybrisPointOfServiceFacetDisplayNameProvider** for the facet search:

**/eeyewearcore/resources/eeyewearcore-spring.xml**

<bean id="hybrisPointOfServiceFacetDisplayNameProvider"

    class="com.eeyewear.core.search.solrfacetsearch.provider.impl.PointOfServiceFacetDisplayNameProvider">

    <property name="pointOfServiceService" ref="pointOfServiceService" />

</bean>

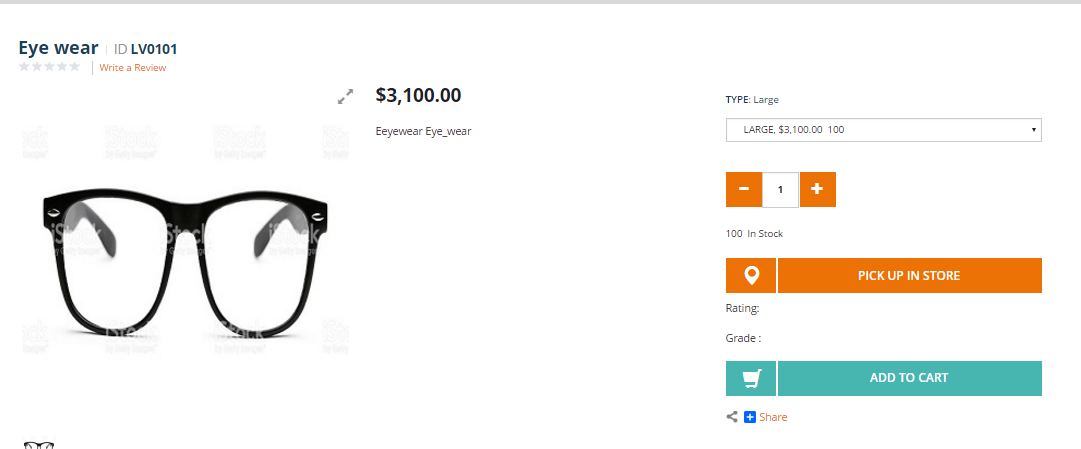
4. Run a system update, selecting only **Import sample data** under the eeyewearintitialdata extension's p

roject data.

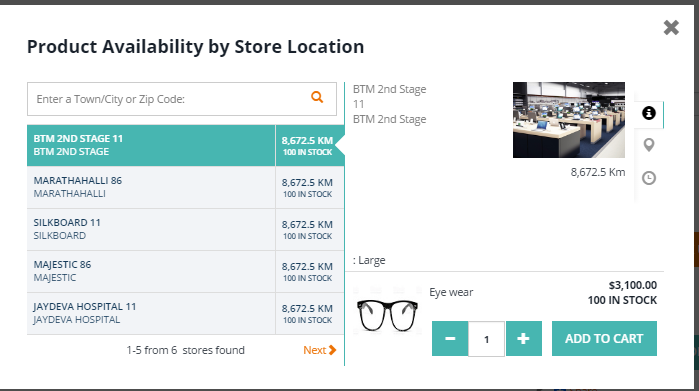
(Assigning StockLevel values to each PointOfService Warehouse was performed )

You may now see that you can find the stores when you use Store Pickup on your site:

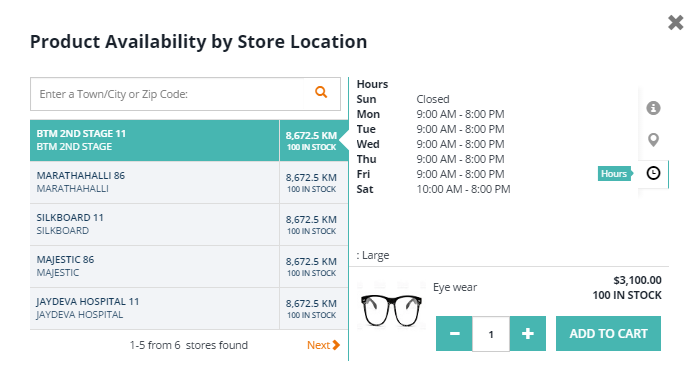
Product details page:



Store lookup and availability:



Pick Up shipping option in cart:



## Enabling Pickup in store on category page:-

1. Add the pickupAvailableFlag index property

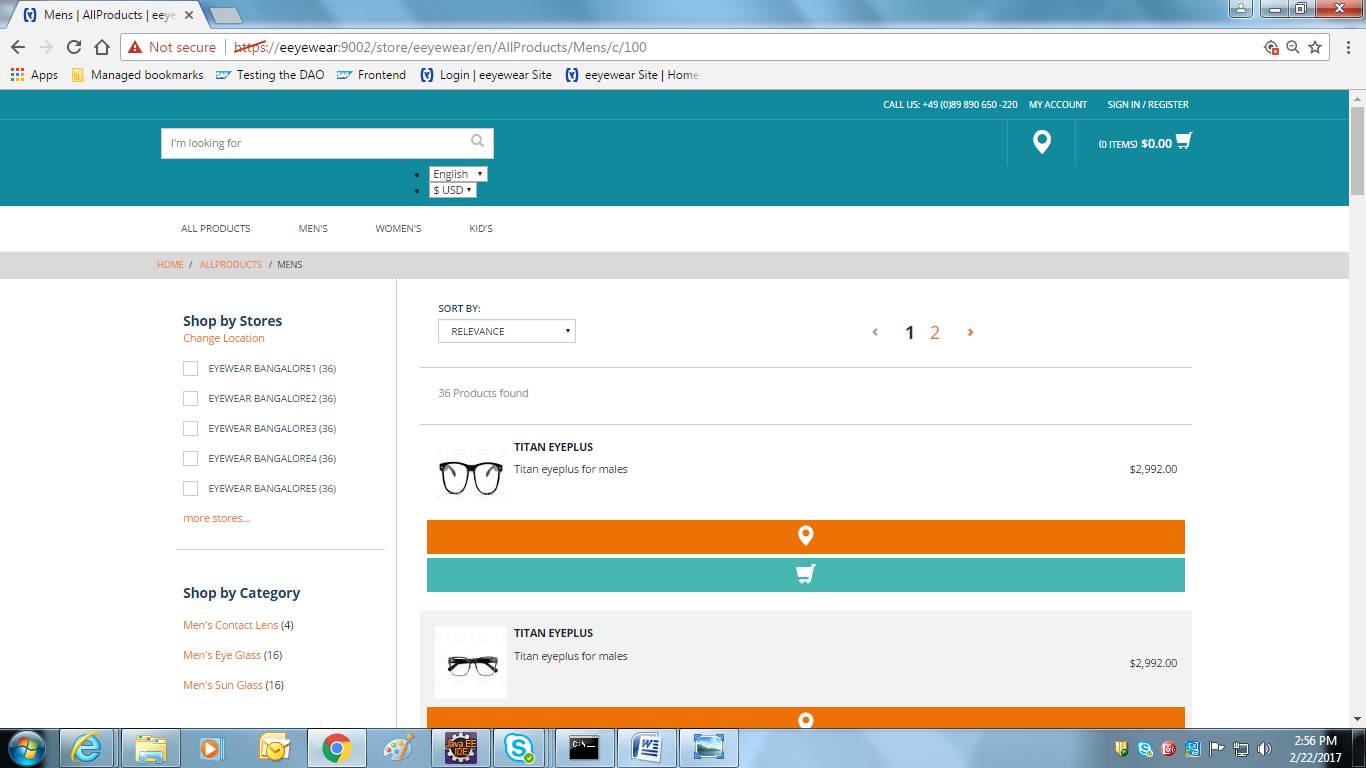
**eeyewearinitialdata/import/coredata/stores/hybris/solr.impex**

# Non-facet properties

INSERT\_UPDATE SolrIndexedProperty

;$solrIndexedType; pickupAvailableFlag    ;boolean;            ;    ;    ;    ;    ;    ;productPickupAvailabilityValueProvider;

1. In hMC, Check hybris CMSSite solr search configuration, should be "hybrisIndex".
2. [Update](http://localhost:9001/platform/update) system with eeyewearinitialdata core data and Activate Solr Cron Jobs.
3. Validate on category page



* 1. **WCMS Cockpit Customization**

****

Here we will be creating a new Page , a new Link and a new Component.

This can be done in two ways:

* Using an existing page Template
* Creating a new Template

We will be following the second approach.

STEPS **:**

1. Create a New Page Template (offersLayoutPage)

Add the following lines in cms-content.impex of eeyewearContentCatalog of coredata

INSERT\_UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];name;frontendTemplateName;restrictedPageTypes(code);active[default=true]

;;OffersPageTemplate;Offers Page Template;offers/offersLayoutPage;ContentPage

1. Create a New Content Page.

INSERT\_UPDATE ContentPage;$contentCV[unique=true];uid[unique=true];name;masterTemplate(uid,$contentCV);label;defaultPage[default='true'];approvalStatus(code)[default='approved'];homepage[default='false'];previewImage(code, $contentCV)[default='ContentPageModel\_\_function\_preview']

;;offersContentPage;Offers Content Page;OffersPageTemplate;offersCMSPage

1. Create Content Slot Names required for the Template

INSERT\_UPDATE ContentSlotName;name[unique=true];template(uid,$contentCV)[unique=true][default='OffersPageTemplate'];validComponentTypes(code);compTypeGroup(code)

;SiteLogo;;;logo

;HeaderLinks;;;headerlinks

;SearchBox;;;searchbox

;MiniCart;;;minicart

;NavigationBar;;;navigation

;Footer;;;footer

;TopHeaderSlot;;;wide

;offersParagraph;;;wide

;offersBanner;;;wide

1. Now Create the Components

Add the following lines in the cms-content.impex of eeyewearContentCatalog for sampledata

INSERT\_UPDATE CMSParagraphComponent; $contentCV[unique=true]; uid[unique=true] ; name ;& componentRef;; ;; NoSearchResultsParagraphComponent; No Search Results Paragraph Component ; NoSearchResultsParagraphComponent

;;offerHeadingComponent;Offers Heading information;offerHeadingComponent;;;;Special Offers for you;

INSERT\_UPDATE SimpleBannerComponent;$contentCV[unique=true];uid[unique=true];name;&componentRef;;;;urlLink

;;offer1BannerComponent;Offer1 Banner Component;offer1BannerComponent;;;;/Offers/BenefitOffer/p/0101

;;offer2BannerComponent;Offer2 Banner Component;offer2BannerComponent;;;;/Offers/SpecialOffer/p/0102

;;offer3BannerComponent;Offer3 Banner Component;offer3BannerComponent;;;;/Offers/ValentineOffer/p/0103

1. Create Content Slots

INSERT\_UPDATE ContentSlot;$contentCV[unique=true];uid[unique=true];name;active;cmsComponents(uid,$contentCV)

;;offersParagraphSlot;Offer Paragraph Slot;true;offerHeadingComponent

;;offersBannerSlot;Offer Banner Slot;true;offer1BannerComponent,offer2BannerComponent,offer3BannerComponent

1. Link the Content Slot for Offer Page Template

INSERT\_UPDATE ContentSlotForTemplate;$contentCV[unique=true];uid[unique=true];position[unique=true];pageTemplate(uid,$contentCV)[unique=true][default='OffersPageTemplate'];contentSlot(uid,$contentCV)[unique=true];allowOverwrite

;;SiteLogo-OffersPage;SiteLogo;;SiteLogoSlot;true

;;HomepageLink-OffersPage;HomepageNavLink;;HomepageNavLinkSlot;true

;;NavigationBar-OffersPage;NavigationBar;;NavigationBarSlot;true

;;MiniCart-OffersPage;MiniCart;;MiniCartSlot;true

;;Footer-OffersPage;Footer;;FooterSlot;true

;;HeaderLinks-OffersPage;HeaderLinks;;HeaderLinksSlot;true

;;SearchBox-OffersPage;SearchBox;;SearchBoxSlot;true

;;TopHeaderSlot-OffersPage;TopHeaderSlot;;TopHeaderSlot;true

;;BottomHeaderSlot-OffersPage;BottomHeaderSlot;;BottomHeaderSlot;true

;;offersParagraphSlot-offersPage;offersParagraph;;offersParagraphSlot;true

;;offersBannerSlot-offersPage;offersBanner;;offersBannerSlot;true

1. Add Link of Offers Page to Navigation Bar

INSERT\_UPDATE CMSNavigationNode;uid[unique=true];$contentCV[unique=true];name;parent(uid, $contentCV);links(&linkRef);&nodeRef

;OfferNavigationNode;;Offers Navigation Node;AllProductsNavNode;;;OfferNavigationNode

INSERT\_UPDATE NavigationBarComponent;$contentCV[unique=true];uid[unique=true];name;wrapAfter;link(uid, $contentCV);styleClass;navigationNode(uid,$contentCV);dropDownLayout(code)[default='AUTO']

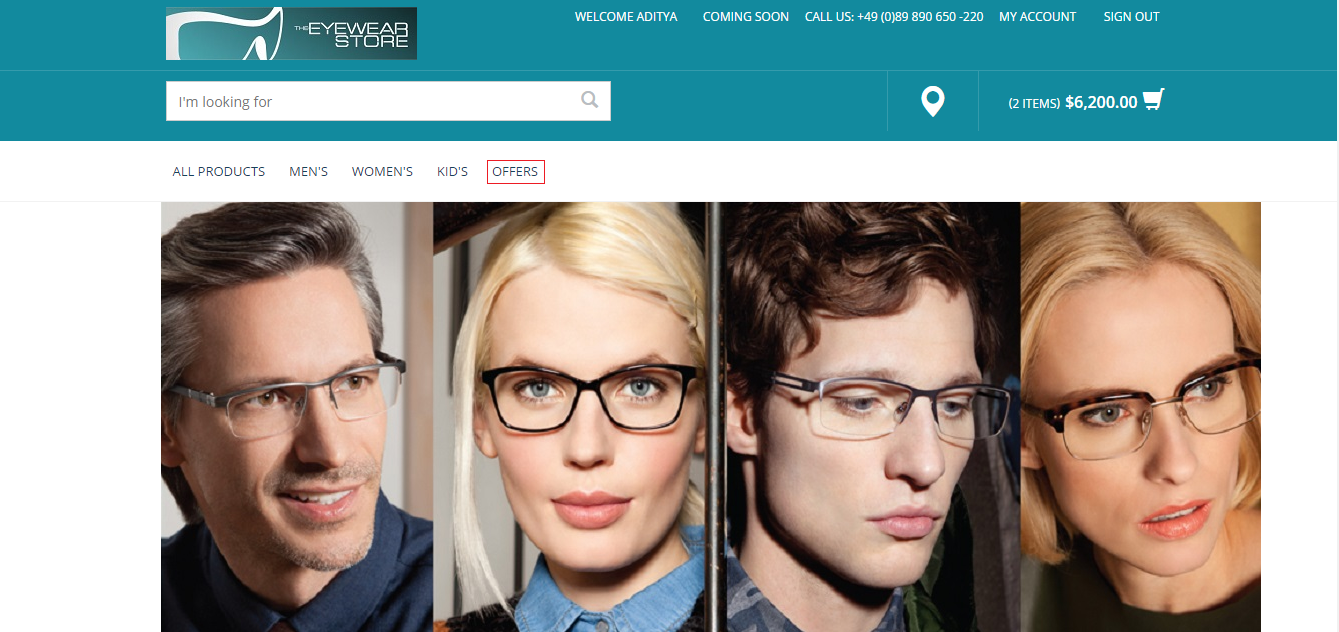
;;OffersNavigationBarComponent;Offers Navigation Bar Component;10;OffersLink;;OfferNavigationNode;;

INSERT\_UPDATE NavigationBarCollectionComponent;$contentCV[unique=true];uid[unique=true];name;components(uid, $contentCV);&componentRef

;;NavBarComponent;Navigation Bar Collection Component;AllProductsBarComponent,MensBarComponent,WomensBarComponent,KidsBarComponent,OffersNavigationBarComponent;NavBarComponent

INSERT\_UPDATE CMSLinkComponent;$contentCV[unique=true];uid[unique=true];name;url;category(code, $productCV);target(code)[default='sameWindow'];;

;;OffersLink;Offers Link;/offers;;;;



1. Now create a controller **OffersPageController** to handle the request on the click of Navigation Node.



1. Now we need to create a jsp Page on which the offer navigation node will redirect to. Thus , create a offersLayoutPage.jsp

eeyewearstorefront\web\webroot\WEB-INF\views\responsive\pages\offers\offersLayoutPage.jsp



1. Update the helper.css for the UI changes.

eeyewear\eeyewearstorefront\web\webroot\\_ui\responsive\common\css\helper.css



1. Add offer images in the below folder.

eeyewear\eeyewearinitialdata\resources\eeyewearinitialdata\import\sampledata\contentCatalogs\eeyewearContentCatalog\images\banners\offers

12 . Now Link the media to the Banner Component. Add the following lines to cms-content\_en.impex of eeyewearContentCatalog of sampledata.

INSERT\_UPDATE Media;$contentCV[unique=true];code[unique=true];@media[translator=de.hybris.platform.impex.jalo.media.MediaDataTranslator];mime[default='image/jpeg'];&imageRef;altText

;;offerBanner\_1.jpg;$siteResource/images/banners/offers/offerBanner\_1.jpg;;;"Grab the Offer soon"

;;offerBanner\_2.jpg;$siteResource/images/banners/offers/offerBanner\_2.jpg;;;"Grab the Offer soon"

;;offerBanner\_3.jpg;$siteResource/images/banners/offers/offerBanner\_3.jpg;;;"Grab the Discounted Sale soon"

UPDATE SimpleBannerComponent;$contentCV[unique=true];uid[unique=true];$picture[lang=$lang]

;;offer1BannerComponent;offerBanner\_1.jpg

;;offer2BannerComponent;offerBanner\_2.jpg

;;offer3BannerComponent;offerBanner\_3.jpg

13. Update the Components and localize them. Add the following lines to cms-content\_en.impex of eeyewearContentCatalog of sampledata.

UPDATE CMSNavigationNode;$contentCV[unique=true];uid[unique=true];title[lang=en]

;;OfferNavigationNode;"Offers"

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

;;OffersLink;"Offers"

14. To display the new template in the cmscockpit add the following vm file in

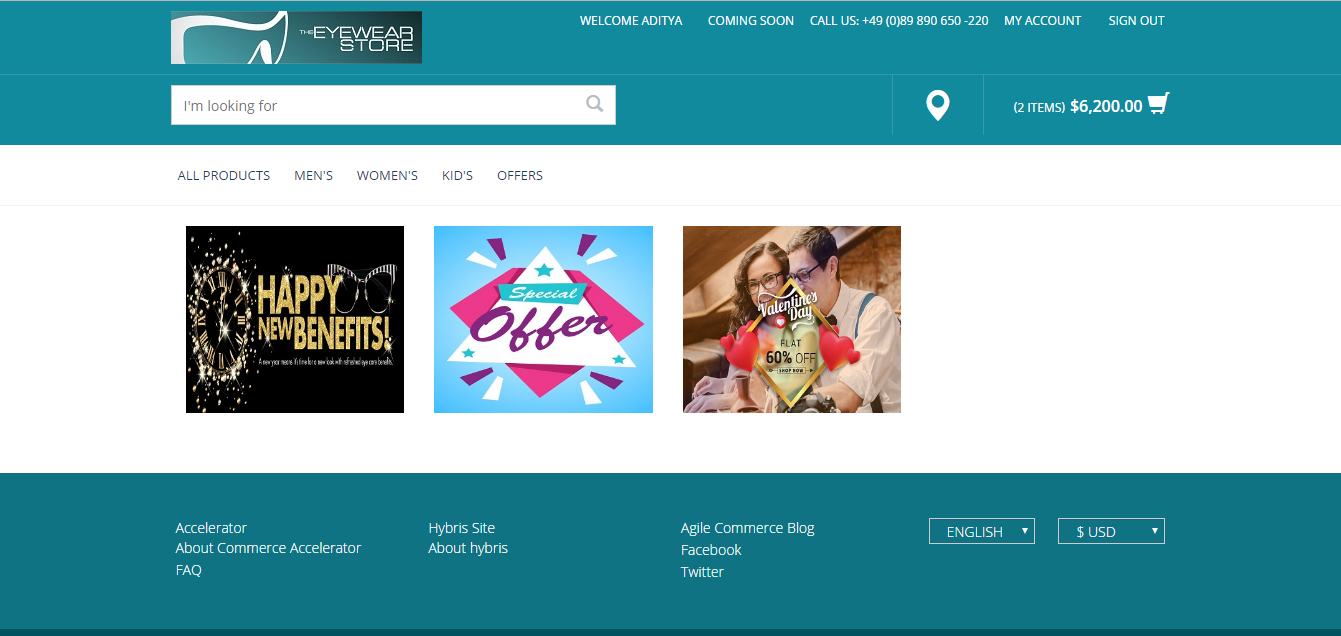
eeyewear\eeyewearcore\resources\eeyewearcore\import\cockpits\cmscockpit\structure-view



15. Give the link this vm file to the template . Add the following lines to cms-content.impex of coredata for eeyewearContentCatalog.

UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];velocityTemplate[translator=de.hybris.platform.commerceservices.impex.impl.FileLoaderValueTranslator]

;;OffersPageTemplate;$jarResourceCms/structure-view/structure\_offersPageTemplate.vm



**2.18 Implementation for to define new attributes in Customer Registration Page:**

1. Secureportal addon 🡪Create extgen.properties file like as available in yCommercewebservice extension.

YEXTNAME\_TOKEN=secureportaladdon

YMODULE\_TOKEN=customaddon

YMODULE\_PACKAGE\_ROOT=core

YMODULE\_CLASS\_PREFIX=Core

YPACKAGE\_TOKEN=com.eeyewear.customaddon

YMANAGER\_TOKEN=SecureportaladdonManager

YCLASSPREFIX\_TOKEN=customaddon

YGENERATED\_TOKEN=Generated

2. Secureportal addon 🡪in extensioninfo.xml 🡪add

<meta key="extgen-template-extension" value="true" />

3. ant all ,Restart Server & update the localizetype and secureportal addon.

4. ant extgen and choose the new template what u have created...

secureportaladdon

eeyewearregistration

com. eeyewearregistration

5. Unintsall the Secureportaladdon & comment secureportal in localextensions.xml & ant clean all and install the the newly created eeyewearregistration

addon template.

6. Now Make the changes in the eeyewearregistration

7. Add the new field in the RegistrationForm.

8. Modify the SecurePortalRegistrationValidator..

9. Go to 🡪B2BRegistrationController and check the method submitRegistration.

10. Create New attribute for Customer itemtype in eeyewearregistration core-item.xml.

<itemtype code=*"Customer"* extends=*"User"* autocreate=*"false"*

generate=*"false"*>

<attributes>

<!-- auto ID which is generated by NumberSeries -->

<attribute autocreate=*"true"* qualifier=*"annualEarning"* 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>

11. Go to eeyewearregistration-bean.xml add the new property in the exixting B2BRegistrationData

<beanclass=*"de.hybris.platform.eeyewearregistration.data.B2BRegistration Data"*>

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

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

</bean>

12. Generate Beans in eeyewearfacades-beans.xml

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

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

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

</bean>

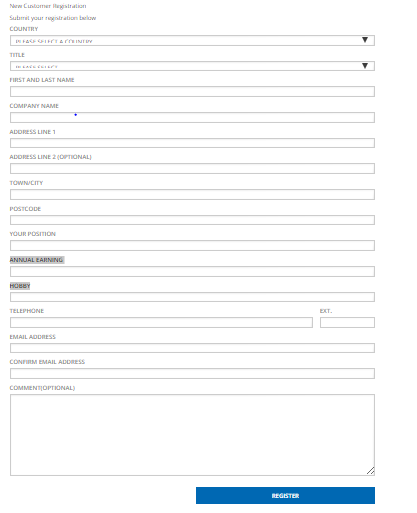
then ant all

13. Now Go to the DefaultB2BRegistrationFacade find the Method toCustomerModel & add

model.setAnnualEarning(data.getAnnualEarning());

model.setHobby(data.getHobby());

14. Edit in register.tag to display new attributes in registration form



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

Define new customer attributes in **eeyewearcore-items.xml** file

|  |
| --- |
| <itemtype code=**"Customer"** extends=**"User"** autocreate=**"false"**  generate=**"false"**>  <attributes>  <!-- auto ID which is generated by NumberSeries -->  <attribute autocreate=**"true"** qualifier=**"annualEarning"**  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> |

2. Define new customer properties in **CustomerData** bean id of **eeyewearfacades-beans.xml file.**

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

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

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

</bean>

3. After that do ant clean all & Refresh the Workspace & Start the Hybris server Update both eeyewearcore & eeyewearfacades extension.

4. Define the populator in **eeyewearfacades-spring.xml** file

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

<bean id=*"eeyewearCustomerAttributesPopulator"* class=*"com.eeyewear.facades.populators.EeyewearCustomerAttributesPopulator"* parent=*"defaultCustomerPopulator"*/>

5. After that do ant all & update eeyewearfacades extension.

6. 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;

Add this below line in that header.

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

7. 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

8.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"

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

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

11. Create **EeyewearProfileUpdateValidator** with new and existing attributes.

12. Create **EeyewearDefaultCustomerFacade** class with new attributes.

13. Create **EeyewearCustomerAttributesPopulator** with new attributes to populate the data.

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

ant all & update commerce services

