Shop Module

Provides basic e-commerce features like product and inventory management, shopping cart process.., based on data hosted outside the Magnolia JCR repository. There is no payment integrated yet.

JIRA	SHOP
Git	Git

Installation

Maven is the easiest way to install the module. Add the following dependencies to your bundle:

```
<dependency>
  <groupId>info.magnolia.shop</groupId>
  <artifactId>magnolia-shop-core</artifactId>
  <version>${version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.shop</groupId>
  <artifactId>magnolia-shop-app</artifactId>
  <version>${version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.shop</groupId>
  <artifactId>magnolia-shop-rest</artifactId>
  <version>${version}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.shop</groupId>
  <artifactId>magnolia-shop-workflow</artifactId>
  <version>${version}</version>
</dependency>
```

Versions

6.2	**Magnolia 6.2**

Configuration

The Shop Core module allows ecommerce (shop) related entities in external SQL Database.

To install this module add maven dependency to this module:

```
<dependency>
      <groupId>info.magnolia.shop</groupId>
      <artifactId>magnolia-shop-core</artifactId>
      <version>${project.version}</version>
  </dependency>
```

To function properly this module requires configurations in **module configuration** location.
Configurations can be located in JCR config node or config.yaml file.
We need to pass database connection info to the module.
This is minimum configuration that needs to be passed to the module so it can connect to a database instance:

```
datasource:
  username: [db_user]
  password: [db_assword]
  url: [db_url]
  driver: [db_driver]
  migration:
    path: shop-core/dbmigration/[db_type]
    run: [true/false]
```

example configuration for MySql database server:

```
datasource:
  username: user
  password: password
  url: jdbc:mysql://127.0.0.1:3306/shop
  driver: com.mysql.cj.jdbc.Driver
  migration:
    path: shop-core/dbmigration/mysql
    run: true
```

All properties that are prefixed with "shop." will be passed to ebean server.
This way we can easily tune ebean ORM framework.
More info on Ebean ORM is available on this ebean documentation link.

This module does not create a database.
Database MUST EXISTS and database user has to have also CREATE/DROP/ALTER RIGHTS on that database.

For database connection to work we need appropriate database driver to be available on the classpath.
For example: if we are using mysql database with tomcat server we need to place "mysql-connector-java-8.0.18.jar" in tomcat lib folder.

Shop App

The Shop App module allows browsing and managing shop related entities (catalogs, products, categories, shippings, prices, etc ...) from Magnolia's Custom Content Application.
This module is dependent on Shop Core module.

To install this module add maven dependency to this module:

```
<dependency>
      <groupId>info.magnolia.shop</groupId>
      <artifactId>magnolia-shop-app</artifactId>
      <version>${project.version}</version>
  </dependency>
```

Shop Rest

The Shop Rest module provide rest api which can be used to manage shop related entities (CRUD operations).
This module is dependent on Shop Core module.

To install this module add maven dependency to this module:

```
<dependency>
      <groupId>info.magnolia.shop</groupId>
      <artifactId>magnolia-shop-rest</artifactId>
      <version>${project.version}</version>
  </dependency>
```

Shop Workflow

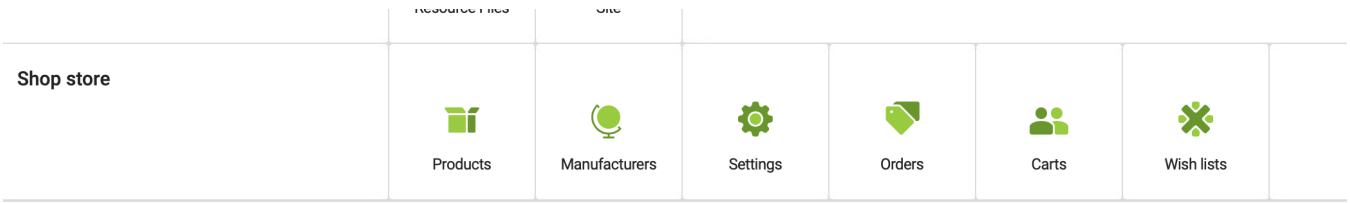
The Shop Workflow module provide functionality for managing orders and transiting order from one state to another.
For example:

- accept order
- mark order as dispatched
- etc..

To install this module add maven dependency to this module:
``` info.magnolia.shop magnolia-shop-workflow ${project.version} ```

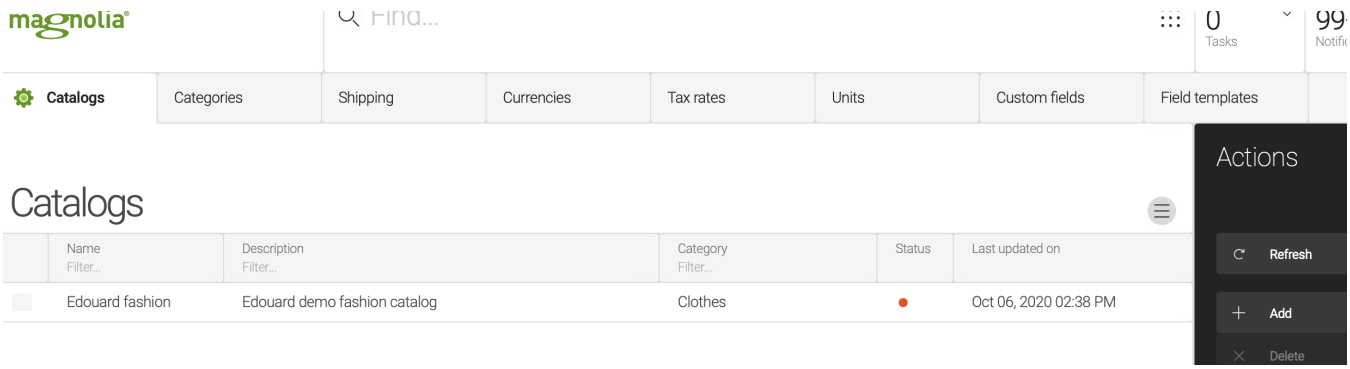
Usage

After installation there will be a new item bar in the main admin central screen as shown in below screenshot.



Settings app

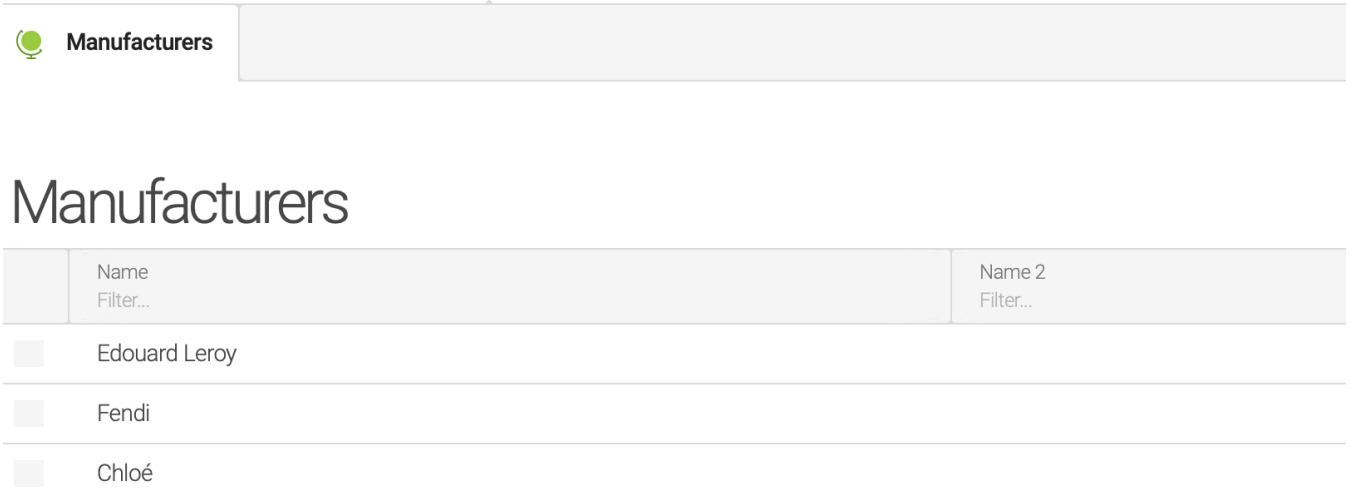
First thing to do is to open the Settings app to define all the shop basics like catalogs, currencies, taxes...
```



- Catalogs - Products will be assigned to catalogs.
- Categories - Categories of products, tree structure.
- Shipping - Shipping methods, different rates can be assigned depending on the purchase price.
- Currencies - Define currencies and exchange rates.
- Tax rates - Define different tax rates.
- Units - Define units for weight, volumen and dimension.
- Custom fields - Products fields can be extended by adding custom fields.
- Field templates - Custom fields need to be assigned to templates to be able to use them.

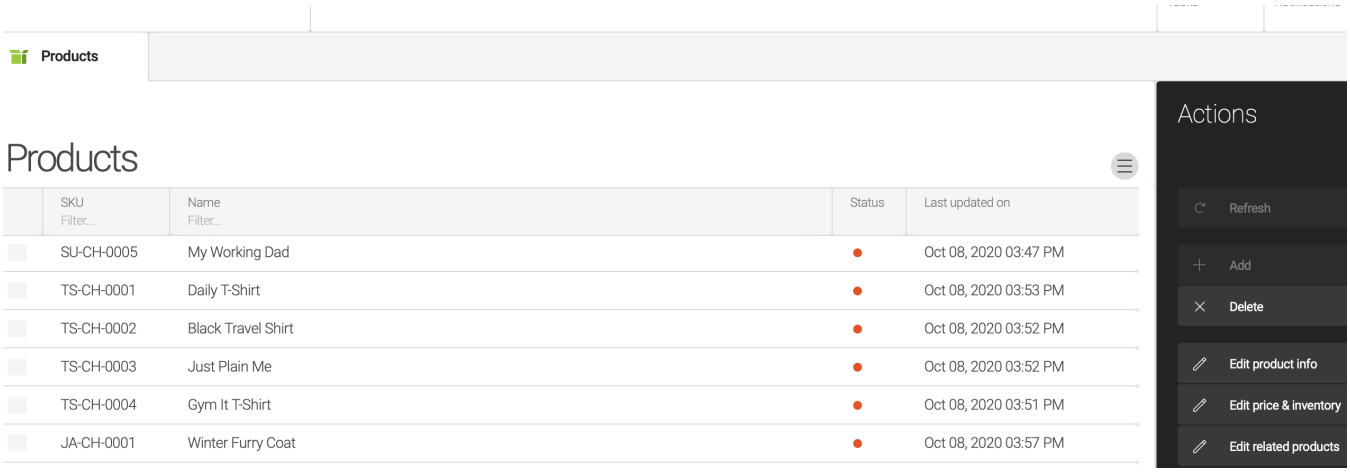
Manufacturers app

This app allows to define different manufacturers to later assign to products.



Products app

After all the above is ready products can be created, assigned to categories, with images, price, dimensions and all custom fields that the user had defined. Products can then be published and be available for the customers.



REST API

Once all of this is in place, the front end app can be created to query for products, add to shopping cart, checkout, create wish lists... by using the REST api.

API Path: /shop/v1

Method	Path	Description	Templating function
		PRODUCT	
GET	/catalogs	Return all the catalogs	True
GET	/catalog/{catalogId}	Return the requested catalog and : • The list of currencies • The category tree • The list of shipping methods	True
GET	/catalog/{catalogId} /products	Return all the catalog products Query string: - limit: Limit the number of results - offset: Index to return the results from - sort: Field to use for sorting and the order (ie: sort=date:asc, sort=date:desc) Does not return linked objects	True
GET	/category/{categoryId}	Return the requested category and its tree of categories	True
GET	/category/{categoryId} /products	Return all the category products Query string: - limit: Limit the number of results - offset: Index to return the results from - sort: Field to use for sorting and the order (ie: sort=date:asc, sort=date:desc) Does not return linked objects	True
GET	/shipping/methods/ {methodId}	Return the requested shipping method and its list of rates	True
GET	/product/{productId}	Return the requested product and • The list of prices • The list of custom fields • The list of categories • The tax rate	True
GET	/product/{productId}/price	Return the price of the product regarding the current date and the requested quantity (body of the request)	True

CART			
POST	/cart	If the user is anonymous (unable to fetch the user from the underlying http session), then it creates an anonymous cart. Otherwise, it create a cart linked to the user id. Return a cookie storing the cart id	false
GET	/cart	Return the cart The cart id is taken from the cookie	true
DELETE	/cart	Delete the cart The cart id is taken from the cookie	false
POST	/cart/item	Update the cart by adding the cart item present in the body of the request The cart id is taken from the cookie	false
PUT	/cart/item	Update the cart by updating the cart item quantity (only field which can be updated) The cart id is taken from the cookie	false
DELETE	/cart/item/{itemId}	Update the cart by deleting the given cart item The cart id is taken from the cookie	false
GET	/cart/cross-sell	Return the cart cross-sell list of products The cart id is taken from the cookie	true
WISHLIST			
POST	/wishlist	If the user is anonymous (unable to fetch the user from the underlying http session), then it creates an anonymous wish list. Otherwise, it create a wish list linked to the user id. Return a cookie storing the wish list id	false
GET	/wishlist	Return the wish list The wish list id is taken from the cookie	true
DELETE	/wishlist	Delete the wish list The wish list id is taken from the cookie	false
POST	/wishlist/item	Update the wish list by adding the wish list item present in the body of the request The wish list id is taken from the cookie	false
DELETE	/wishlist/item/{itemId}	Update the wish list by deleting the given wish list item The wish list id is taken from the cookie	false
ORDER			
POST	/order	Get the current cart, the billing/shipping addresses, the shipping methods and convert them into an order	false
GET	/orders	Get the list of orders whose the user id match with the session user id. **Does not work** for the anonymous accesses.	true
GET	/order/{orderId}	Get the requested order. Only return the order if the user id match with the session user id. **Does not work** for the anonymous accesses.	true
PUT	/order/{orderId}/cancel	Change the status of the order into CANCELLED if the user id match with the session user id. **Does not work** for the anonymous accesses	false

Warnings

- This module is at INCUBATOR level.

Changelog

- Version 0.0.1 - Initial release beta.