RainBowOF – Rainbow order fulfilment

Actrive document used for design ideas. Will be merged into a single document for documentation.

Main screen:

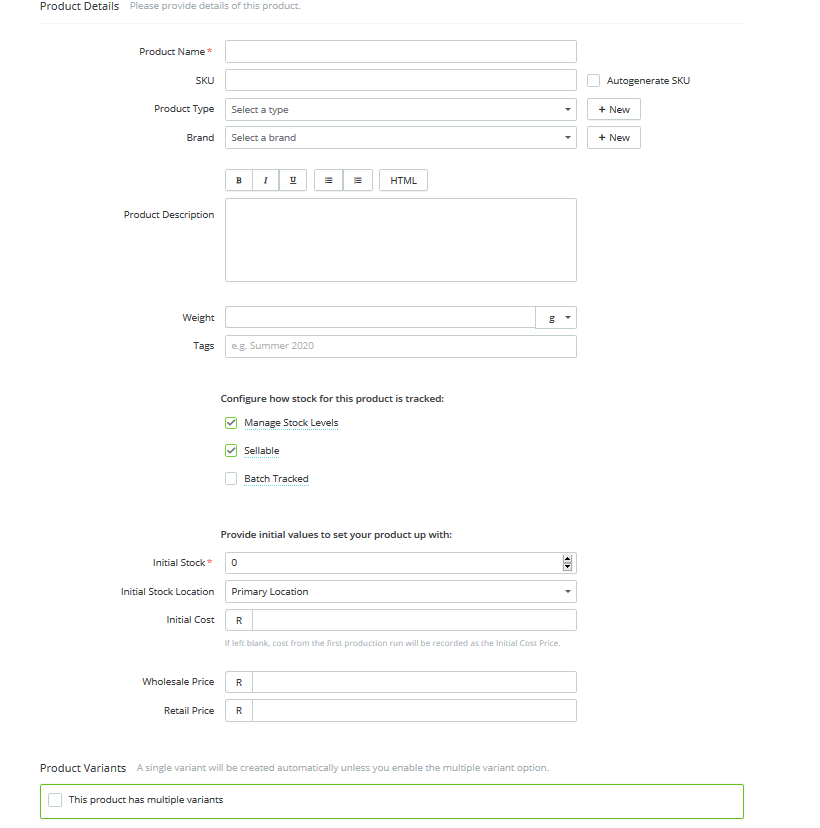
Main Menu:

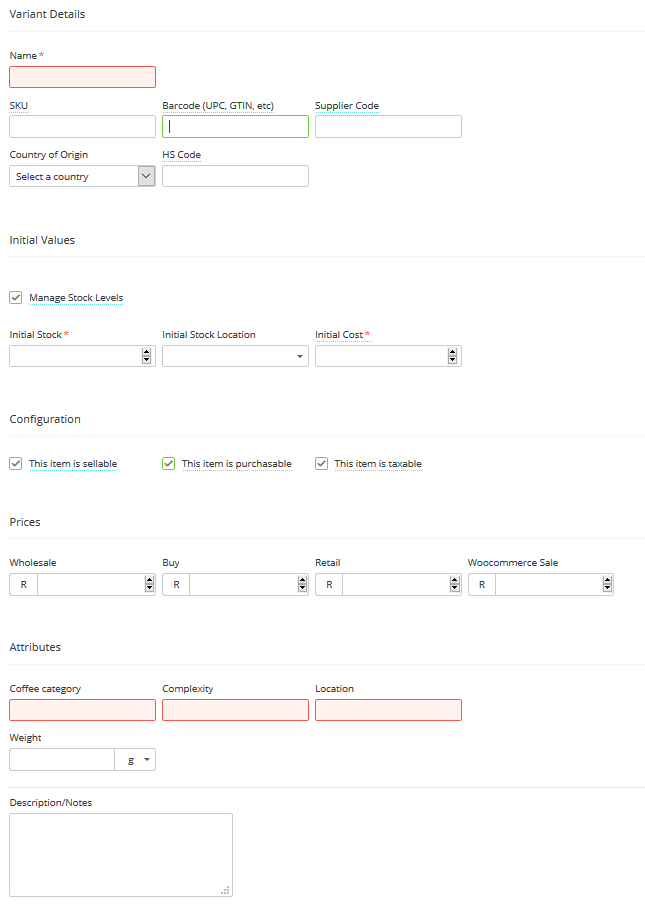
* Dashboard?
  + Order Status
  + Feed Status
  + Preparation
  + Packing
* Contacts
  + All
  + By Type
  + Preferences
    - Equipment owned?
* Inventory
  + Items
    - Active
  + Item Groups
  + Price Lists?
  + Preferences
    - Item Types
      * Should this be simple / Variety? What about equipment?
    - Packaging
    - Varieties
* Orders
  + New
  + View
    - Active
    - By Day
    - All, Today, Tomorrow, This Week, Next Week
    - By Status
      * Ready, Invoiced, Done, Pending
    - By Type?
    - Packing
  + Recurring
  + Returns
  + Preferences
* Notifications
  + Send notifications
  + Preferences
    - Email template, order, ship, done, pack, repair
* Repairs
  + Active
  + By Status
  + Preferences
    - Swop out equipment
* Integration
  + Import/Export
    - Need to add import per group
  + Live integration
    - Woo settings
  + Google form? Requests?
* Reports
  + Preparation Report
  + Packing Report
* System
  + Users/Parties
    - All
    - By type (Driver, Courier, admin, user)
  + Tools
* Settings
  + Days and dates (weekdays/closure dates)
  + Company Details
  + Preferences
    - Regional Settings
    - TAX / VAT
    - Email settings
  + Shipping Zones / Areas?
  + Accounting
    - Price levels
    - Invoice Types

Woo Items are from - <https://woocommerce.github.io/woocommerce-rest-api-docs>. We use the WooRestAPI NuGet Package - WooCommerceNET by JamesYang@NZ.

Example screens:

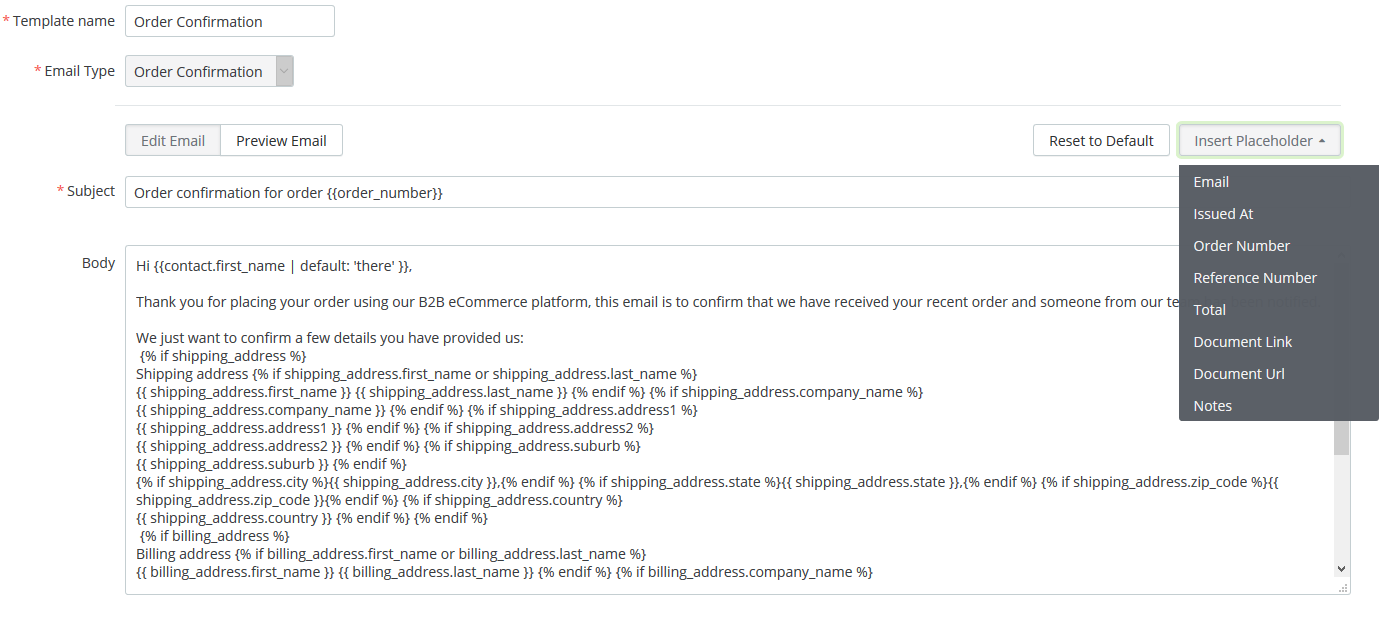
Product



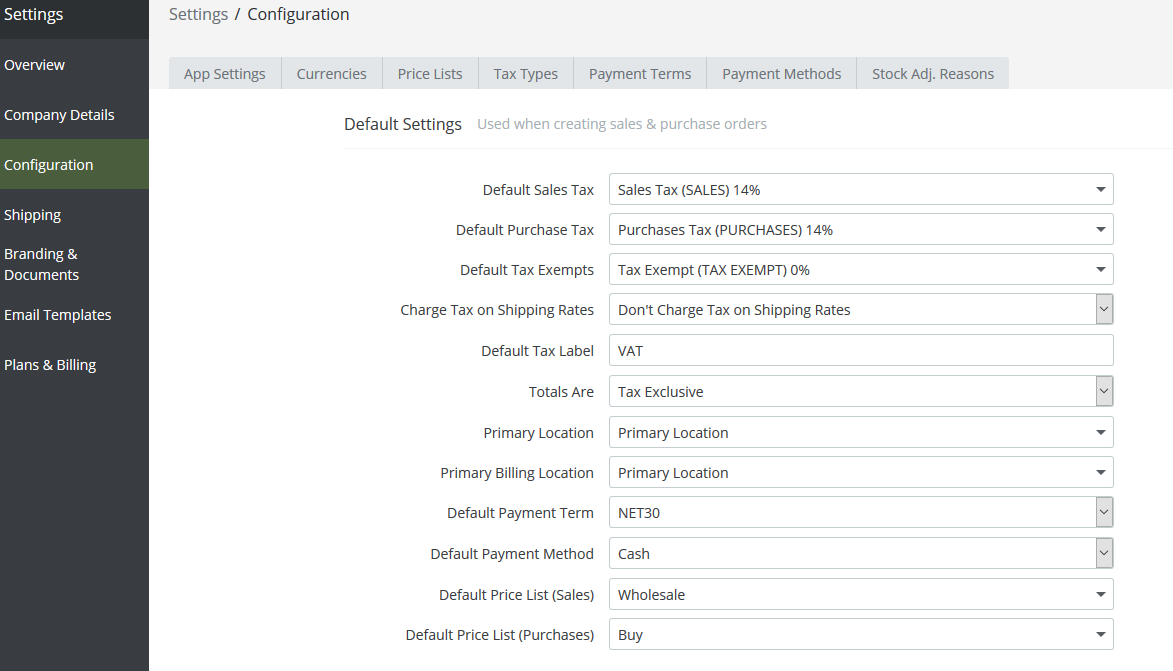
Item Variant

]

Notifications:



Settings:



Items

Will use the items from WooCommerce as a guide.

# WooItems Import

Used for importing items to the system. Using the WooSettings table.

Options:

* API settings (URL/IS HTTPS/QueryPrefixes/CustomerKey/CustomerSecret
* Import of:
  + Categories (**true**/false)
  + Attributes (**true**/false)
  + Map Varieties (**true**/false)
  + Only in stock (**true**/false)
  + Include item quantities (true/**false**)

## Categories

First need to import categories

[https://www.quaffee.com/wp-json/wc/v3/products/categories](https://www.quaffee.com/wp-json/wc/v3/products/categories?parent%3e0)

From there we can create a list of item categories. From that list we need to create a list of primary categories. These are used in the items mapping. Only those that are imported can be used

#### What Woo Gives Us:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| id | integer | Unique identifier for the resource. (read-only) |
| name | string | Category name. (mandatory) |
| slug | string | An alphanumeric identifier for the resource unique to its type. |
| parent | integer | The ID for the parent of the resource. |
| description | string | HTML description of the resource. |
| display | string | Category archive display type. Options: default, products, subcategories and both. Default is default. |
| image | object | Image data. See [Product category - Image properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-category-image-properties) |
| menu\_order | integer | Menu order, used to custom sort the resource. |
| count | integer | Number of published products for the resource.(read-only) |

#### Our Lookup class

|  |  |  |
| --- | --- | --- |
| Field name | Type | Notes |
| ItemCategoryLookupId | Guid |  |
| CategoryName | string(255) | item category |
| ParentCategoryId | Guid? | [foreignkey("parentcategoryid ")] |
| Notes | String |  |
| Rowversion | byte[] | [timestamp] |

#### Mapping:

The Table WooCategoryMaps stores the category information.

|  |  |
| --- | --- |
| **Field** | **UsedFor** |
| WooCategoryID [int] | To store the CategoryID that Woo returns |
| ~~WooCategoryName [string (size:2-255)]~~ | ~~To store Categories.name that Woo returns~~ |
| ~~WooCategorySlug [string (size:2-255)]~~ | ~~To store Categories.slug that Woo returns~~ |
| WooCategoryParentID | Can be null if not will point to a WooCategoryID |
| ItemCategoryID | This links to the Category in the system. Allows us to transfer the data. |

Item categories Should be linked to tracking so items in similar categories are tracked similarly. Not sure if we need to store all the above, can just do a link table?

Need to store the parent id, so that you can do reporting. Parentid=0 means no parent (like with woo)

Logic:

* Retrieve all the woo Categories
* if the IDs exists in the mapping update
* otherwise if the a category of the same name exists get that ID and update
  + otherwise add the category
  + if the ParentId > 0 add this category id to a list of categories to scan later to check if the parent has been imported.

Once the import is do then loop through the list of attributes that had parents and see if they exists if so set the id otherwise set parent id to Guid.empty

### Display of Categories

Categories are displayed using a DataGrid. The DataGrid will allow you to edit the category and all data related to it. Like all the other DataGrids sorting and filtering of critical fields is supported.

#### Default Sort:

Sorted by Parent Name(s)+CategoryName

#### Columns:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **Format** | **Validation** |
| Category Name | CategoryName prefixed with “—” per parent | Y | Y | Y |  |  |
| Parent Category | If there is a parent category otherwise n/a | Y | Y | Y |  |  |
| Prediction | Is this category used for prediction | Y | Y | Y | Combo |  |
| Woo Linked | Is it linked to Woo, can Woo update this? And must we update it in Woo | Y | ? | ? | Y/N |  |
| Notes | Any notes about the category | Y | Y | Y |  |  |

**Legend***: E= Editable; F= Filterable; S = Sortable*

### Editing

Editing will occur via the standard DataGrid modal popup menu. With all items besides the current available in the dropdown list linked to the current parent.

If modified, then Woo item is updated only if woo items are affected like Category Name and Parent Category.

## Attributes/ Variations

Attributes are used but Woo to create Variations of a product. A product can have normal attribute or attributes are used to create variations.

Variables are product Attributes that are used as variations in a product. If a product item is of type “variable” then it has attributes that make up the varieties. There are groups of varieties, like colour (Green, Red, White, Blue) and size (S,M,L) or packaging (box, bottle, packet bag) and preparation (beans, ground espresso, ground filter).

To retrieve attributes:

<https://www.quaffee.com/wp-json/wc/v3/products/attributes>

This returns the attribute names not the types. We still need to link the attribute to the attribute types. So this mean parent child tables

Parent Table is ItemAttributes, Child is ItemAttributeVarieties. Each Item will also need a table linking the ItemAttributes that item has to the item

For the user entering in a line item in the order / invoice, just enters in the SKU or can search on Item name (auto complete)

#### What Woo Gives Us:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| Id | integer | Unique identifier for the resource. (read-only) |
| Name | string | Attribute name. (mandatory) |
| Slug | string | An alphanumeric identifier for the resource unique to its type. |
| Type | string | Type of attribute. By default only select is supported. |
| order\_by | string | Default sort order. Options: menu\_order, name, name\_num and id. Default is menu\_order. |
| has\_archives | boolean | Enable/Disable attribute archives. Default is false. |

### Our class ItemAttributesLookup

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeLookupID | Int | Pk |
| AttributeName | String(100) | Required, indexed |
| OrderBy | Enum ? (custom, name, num and id) | Should this be an enum? |
| Notes | String |  |

#### Mapping:

The Table WooVariationMaps stores the category mapping information.

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeLookupId | GUID | PK (ItemAttributeId+WooProductAttribID) |
| WooProductAttributeId | Int |
| CanUpdate | Bool | Set true by default – false if we do not want the product to be updated |

All id fields that are to be converted to GUID – except for preferences

### Attribute Terms

Here is a list of the terms that an attribute has or the list of labels/names an attribute can have. For example. Packaging can be an attribute which has Terms: 250, 275, 500 , 750, 200 etc.

This is a list that can allocated to a product that has that attribute. A product can have any number of these terms. For our purposes we will call the variety. Since we mainly use this to determine the variety of the product selected.

These have to also be retrieved.

#### What Woo Gives Us:

Woo gives us this per attribute:

You request a list of an attributes terms using the Woo Attribute’s Id. Retrieved using

###### /wp-json/wc/v3/products/attributes/<attribute\_id>/terms

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| id | integer | Unique identifier for the resource. |
| name | string | Term name. (mandatory) |
| slug | string | An alphanumeric identifier for the resource unique to its type. |
| description | string | HTML description of the resource. |
| menu\_order | integer | Menu order, used to custom sort the resource. |
| count | integer | Number of published products for the resource. (read-only) |

##### Item Attribute Variety

We store this data is a table ItemAttributeVariety. Based on the Woo Attribute Varieties or what they call terms, but added to that the fields for displaying

|  |  |  |
| --- | --- | --- |
| Field name | Type | Comments |
| ItemAttributeVarietyID | GUID | Pk |
| ItemAttributeID | GUID | Parent or null |
| VarietyName | String(100) | Required, indexed |
| UoMID? | GUID | Optional (set by user) linked to UoM |
| SortOrder | Int | Number for order |
| Symbol | String(2) | For display |
| FGColour | String(11) | For display |
| BGColour | String(11) | For display |
| Notes | String |  |

#### WooLink

Use these table to link, and if the WooSetting is set to Import Attributes

##### WooProductAttributeMapping

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeID | Int | Pk with WooProductAttribID |
| WooProductAttributeID | Int | Links to woo attribute |
| CanUpdate | Bool | Set true by default – false if we do not want the product to be updated |

##### WooProductAttributeTermMapping

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeVarietyID | Int | Pk with WooProductAttribTermID |
| WooProductAttributeTermID | Int | Links to woo attribute |
| CanUpdate | Bool | Set true by default – false if we do not want the product to be updated |

## Items

Items are what is offered to the client, what can be ordered and delivered. The Items are imported from Woo, using the Woo Settings.

Each Woo Item has:

* Item Detail stuff (like name, SKU etc)
* Can have varieties via the attribute that are marked as variables
* Pricing details

WE allow the option to import all items or just those in stock. All items that are marked a draft are not imported. Only stock status private and publish should be included

### What Woo Gives Us:

Items are called Products in Woo. Woo gives us a lot per Item the full list can be seen here (https://woocommerce.github.io/woocommerce-rest-api-docs/#products):

###### To pull all product: /wp-json/wc/v3/products

This pulls 20 per page. What we are interested in

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| id | integer | Unique identifier for the resource. read-only |
| name | string | Product name. |
| permalink | string | Product URL. read-only |
| type | string | Product type. Options: simple, grouped, external and variable. Default is simple. |
| status | string | Product status (post status). Options: draft, pending, private and publish. Default is publish. |
| featured | boolean | Featured product. Default is false. |
| catalog\_visibility | string | Catalog visibility. Options: visible, catalog, search and hidden. Default is visible. |
| description | string | Product description. |
| short\_description | string | Product short description. |
| sku | string | Unique identifier. |
| price | string | Current product price. read-only |
| regular\_price | string | Product regular price. |
| sale\_price | string | Product sale price. |
| on\_sale | boolean | Shows if the product is on sale. read-only |
| purchasable | boolean | Shows if the product can be bought. read-only |
| virtual | boolean | If the product is virtual. Default is false. |
| downloadable | boolean | If the product is downloadable. Default is false. |
| manage\_stock | boolean | Stock management at product level. Default is false. |
| stock\_quantity | integer | Stock quantity. |
| stock\_status | string | Controls the stock status of the product. Options: instock, outofstock, onbackorder. Default is instock. |
| parent\_id | integer | Product parent ID. |
| categories | array | List of categories. See [Product - Categories properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-categories-properties) |
| tags | array | List of tags. See [Product - Tags properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-tags-properties) |
| images | array | List of images. See [Product - Images properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-images-properties) |
| attributes | array | List of attributes. See [Product - Attributes properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-attributes-properties) |
| default\_attributes | array | Defaults variation attributes. See [Product - Default attributes properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-default-attributes-properties) |
| variations | array | List of variations IDs. read-only |
| menu\_order | integer | Menu order, used to custom sort products. |

Using what we need for the system we have added items needed for the Woo Integration. Including

Using the variety, we can link to UoM a

As part of the import we select if we only import item in stock if so we only pull those. We should also consider hidden products.

Item Images

To import these or not to import that is the question. May be easier to have a URL for the image.

Item Attributes

In the Product->attributes[] array is:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| Id | integer | Attribute ID. |
| Name | string | Attribute name. |
| position | integer | Attribute position. |
| Visible | boolean | Define if the attribute is visible on the "Additional information" tab in the product's page. Default is false. |
| variation | boolean | Define if the attribute can be used as variation. Default is false. |
| options | array | List of available term names of the attribute. |

We use the variation setting to then set the Attribute Varieties for the Item, these are found in **options**, which lists the Attribute Terms (what we call varieties) we need to map the terms to the variety.

### WooProduct -> Item Table Mapping

Mapping the Products is required for continued updating. Once we know which product is what we can update it. So we need to Link the WooProduct.Id to Item.Id

##### WooProductToItemMapping

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemID | GUID | Pk with WooProductAttribID |
| WooProductID | Int | Links to woo attribute |
| CanUpdate | Bool | Set true by default – false if we do not want the product to be updated |

## Item Table

Here we look at our Item Table and see what we map.

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemId | GUID | Unique to IT not imported |
| ItemName | String(100) | Links to WooProducts.name |
| SKU | String((50) | Links to WooProduct.sku |
| IsEnabled | Bool | Links to WooProduct.stock\_status = instock |
| ItemDetail | String(255) | Links to WooProducts.short\_description |
| ItemCategoryId | int? | Links to the first id in the WooProducts.categories[] |
| ParentItemId | GUID? | Not sure how this works, but would link back to ItemId after the import if found. |
| ReplacementItemId | GUID? | leave |
| ItemAbbreviatedName | string(10) | Abbreviated name – not imported |
| SortOrder | int | linked to WooProducts. menu\_order |
| ManageStock | Bool | Do we (or Woo) manage stock |
| QtyInStock | Int | Amount in stock, if we manage stock (order must track if order came from woo or manual, if order from woo then we need not send back to woo, else we do |

There may be additional fields for Price support etc. To be done later.

#### Update?

We should consider if we want to allow updating. Things may change and we may not want items to be updated.

#### Logic

There are two options here, one is to retrieve all the products (this may take some time) and one is to get a page at a time and deal with it. To display a percentage (of how far we are) we need to retrieve all products. This may mean we end up throwing an error.

* Get All Woo Products
* Set counters
* For each product
  + If the product does not exists add and CopyWooDetails to new record increase Counter.added
  + If product does exist and CanUpdate is true then Update by copying the WooDetaits that are not keys, increase Counter.update

CopyWooDetails:

The copies the info across:

* Set ItemName, SKU, IsEnabled, ItemDetail, SourtOrder as per above mapping in table
* For ItemCategotyId – selected the first Category in the array find the CategoryId of it using the WooCategoryMapping and set. We assume the first is primary. Perhaps we should look for one those without parents? Only add if it does not exist
* If the item does not exist then also copy the attributes and for each attributes the attrivute terms, or varieties.
* For ParentItemId Add to a List ItemParents the ParentId and the ItemId

Once finished we need to map an Items parents. We do this by cycling through the list of items that have parents, then we find the item and parent’s GUID and take match them.

To view list of attributes terms

https://www.quaffee.com/wp-json/wc/v3/products/attributes/14/terms

These can map to variations.

For each product we allow people to select the “parent” and then the child, making it easier and supporting more. Then we need to do variation report linking to UOM or each.

## For UOM:

Each item can have multiple variations (or none). There are two types of variations:

* One variation per SKU per item. This means that the Item has a one-to-one mapping
* Each variation is a Unit of Measure (UoM) that is linked to a quantity or s description

So for example.

1. Coffee A, can be sold as 250g, 275g, 500g, 750g and 1kg. The variation 250g is a one to one mapping the others are mapped to quantity of .275, .5,.75, 1 kg of coffee (this uses a parent SKU that. The is determined by the variation. For 1-many mapping there needs to be a parent id and parent SKU
2. Cleaning Pills come in a box of 6 or bottle of 25. Each variation is a 1-1 mapping to the SKU. This should be the default
3. WaterFilters come in for variables white, blue, smart, smartpro. Each variation is a 1-1 mapping per SKU. Can this be change to typeof filter?

For the user entering in a line item in the order / invoice, just enters in the SKU or can search on Item name (auto complete)

The Table WooVariationMaps stores the category information.

Need to store the parent id, so that you can do reporting. Parentid=0 means no parent (like with woo)

Using -> <https://www.quaffee.com/wp-json/wc/v3/products?per_page=5&type=variable&stock_status=instock>

Scan through the attributes that have variation set to true.

From those variations look for the categories. Then offer a link between the category and the category options

Display a list of these per category, then allow the offered as options to be set.

Using the variation in each category, link the variation to the option:

* Is a separate SKU -> us parent SKU or not?
* Has UoM -> Qty per UoM

Need to set these using a default offering options

Category Options

* Packaging
* Preparation Type
* Size
* Model
* Colour
* options

Packaging

Offered as?

Packaging -> 250/275/500/750/200

Packaging - > Box of / Bottle of / each

Colour - > White / Blue / Red / Grey

Capacity ->

Size -> XS, S, M, L, XL, XXL

Model ->

We also need to add Preparation / Serving

Perhaps should be called Packed

Bottels, Packets,

To be linked to a product variation, so pull the variations and then link those preparation

# Items

WooItems->Items mapping

|  |  |
| --- | --- |
| **WooItem** | **iSeleI.Items** |
| Id | ExternalID |
| Name | ItemName |
| Permalink | ItemUrl |
| Type | IsVariable = type==variable |
| stock\_status | IsEnabled = stock\_status == “in stock” |
| short\_description | Description |
| Sku | SKU |
| Price | Price |
| Categories[] | Loop through and find a category that is marked as a item category. Use that to set default sort order |
| Images[] | Imageurl= First in the list.src |
| Attributes[] | Map to ItemAttributes |

For each variation need to query it

Not used:

Slug, date\_created,date\_created\_gmt, date\_modified, date\_modified\_gmt, status, catalog\_visibility, featured, description (?),regular\_price, "regular\_price, sale\_price, date\_on\_sale\_from, date\_on\_sale\_from\_gmt, date\_on\_sale\_to, date\_on\_sale\_to\_gmt, price\_html, on\_sale, purchasable, total\_sales, virtual, downloadable, downloads, download\_limit, download\_expiry, external\_url, button\_text, tax\_status, tax\_class, manage\_stock, stock\_quantity, backorders, backorders\_allowed, backordered, sold\_individually, weight, dimensions, shipping\_required, shipping\_taxable, shipping\_class, shipping\_class\_id, reviews\_allowed, average\_rating, rating\_count, related\_ids, upsell\_ids, cross\_sell\_ids, parent\_id, purchase\_note

ItemAttributes:

|  |  |
| --- | --- |
| **WooItem** | **iSeleI.Items** |
| Id | ExternalID |

This will mean that items will be simple

## Link of Item to Attributes

Linking of Item to Item to ItemAttributes

ActiveItemAttributes

Is used to store which attributes are active for an item

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeID | Int | Pk |
| ItemID | Int | Link to the Item that has this attribute |
| IsUsedForItemVariety | Bool | Is this used as to create item varieties |
| ItemAttributeVarietiesActiveID | Ink | Fk -> to give a list of Attributes. |

### ActiveItemAttributeVariety

Items have Attributes and Attributes have varieties or in woo language terms. So, an item has a list of attributes that has a list of attribute varieties or terms. Attributes can be used for variation, which links to the packaging / preparation through the attribute variety.

Using that ItemAttributeActive per item you can see which items are active with for this item.

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeVarietiesActiveID | Int | Pk |
| ~~ItemID~~ | ~~Int~~ | ~~Link to the Item that has this attribute (not needed as no in Attribute Table now)~~ |
| ItemAttributeID | Int | Links to parent attribute (could be excluded) |
| IsDefault | Bool | Is this the default variety (of variety type) |
| UoMID | Int? | ->ItemUoM.ID |
| QtyPerUoM | double(12,6) | The quantity this is for UoM |

This allows us to import:

1. WooAttributes and map them to ItemAttributes
2. WooAttribute->Terms and map them to ItemAttributeVarieties
3. When importing Items (later) add an Attribute per attribute in a WooProduct to Item
4. Also for each WooProduct variation we can add a AttributeVariety and then mark it as use for variable IF it is a variable product and if this attribute is marked to be used as var

## Item Variety UoM

This is used for orders and needs to be set per Item Variety. It is imported and has to be set up by editing the Item Varieties. Each Item has attributes, and each attribute can have varieties. Each variety can map to a unit of measure.

This is used to link an Item’s variety to a Unit of Measure. So each, box, bottle, bag, packet, etc.

A box of pills has a UOM of box = 6 of each

A variable of 250g has a UoM of kg and the conversion is .25\*Qty.

So we need to store UoM per variation and link that to a Qty per Base UoM

### ItemUoM Table

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemUoMID | Int | Pk |
| UoMName | String(100) | Should be smaller |
| UoMSymbol | String(10) |  |
| BaseUoMID | Int | If>0 points to the BaseUoM |
| BaseConversationFactor | Double |  |
| RoundTo | Int | Number of decimals to round to (default 4) |

These will be linked to the Items through then Items Attribute Varieties, and is optional

### Mapping of Woo Products to Items

A woo product has a parent produce and then if there are varieties then child products. In our table this is stored using the Parent ID and the Item d. So, an Item with no Parent Id is a Parent otherwise it is a variety of the parent.

So, for each variety we need to lookup that product from woo and then add that product to the item table with the correct settings. If a variety is actual a quantity, then we need to determine that from the Unit of Measure, which means we need to link Item to UoM.

So, lets say we have Product X, it is has a UoM of kg. If the varieties are then “g” or “kg” or “kg” we need to link them through UoM. So, Item X with UoM kg can be sold as different types of units but still be linked to the same item.

Items that are in the same category at any level can be tracked / Predicted. Each Category that is predictable will silo items in that category based on UoM Qty, Date supplied, and actual item.

## Import of Items with Children

For each product only the parent products are imported, not the children (in the above import). Now all items that are children need to be imported. Only the items that are not null will be specific to the child item. Otherwise, all items will be inherited.

So for each parent item who has an attribute that is used for variation (or term) we need to import those children.

### Logic

Using the ItemTbl imported, get all items that have attributes marked as Is Used For Item Variety. Retrieve that product and then import each variety as a product.

Note: the predictability has to be set manually for each Category.

## Item View and Edit

The items page will display all parent items in a DataGrid. When the page is loaded the app status is checked to see if Woo in Linked if so it enables the Woo linked fields.

### View

#### Default Sort:

Sorted by Item name

#### Columns:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Item Name | Item name | Y | Y | Y | N | String | ^\S.{2,100} |
| SKU | Item’s SKU | Y | Y | Y | N | String | ^\S.{2,100} |
| Image | First image in list’s url is displayed | N | N | N | Y | Image | n/a |
| Enabled | Is the item enabled | Y | Y | Y | N | Y/N | selection |
| Detail | The detail of the item | Y | Y | Y | Y | HTML? |  |
| Categories | Categories the Item has – primary is bold, predictive is in italics | Y | Y | ? | Y | List | selection |
| Attributes | Attributes the item has | Y | Y | ? | Y | List | selection |
| Replace By | If the item was replaced by another item | Y | Y | Y | Y | String | selection |
| Abrv | Item Abbreviation | Y | Y | Y | N | String | ^\S.{2,10} |
| Base Price | The base price of the item | Y | Y | Y | N | Decimal |  |
| ManageStock | Is the stock managed by Woo or us – needed for Woo updates | Y | Y | Y | N | Y/N | selection |
| Stock Qty | The qty of this item in stock 0 means nothing or not tracked | Y | Y | Y | N | Int | \d |
| Sort Order | Order the item is sorted | Y | Y | Y | N | Int | \[0-9]+ |
| Woo Linked | Is the item enabled | Y | Y | Y | N | Y/N | selection |

**Legend***: E= Editable; F= Filterable; S = Sortable, H = hide if small screen*

### Editing

Editing will occur via a new edit page.

Fields

If modified, then Woo item is updated only if woo items are affected like Category Name and Parent Category.

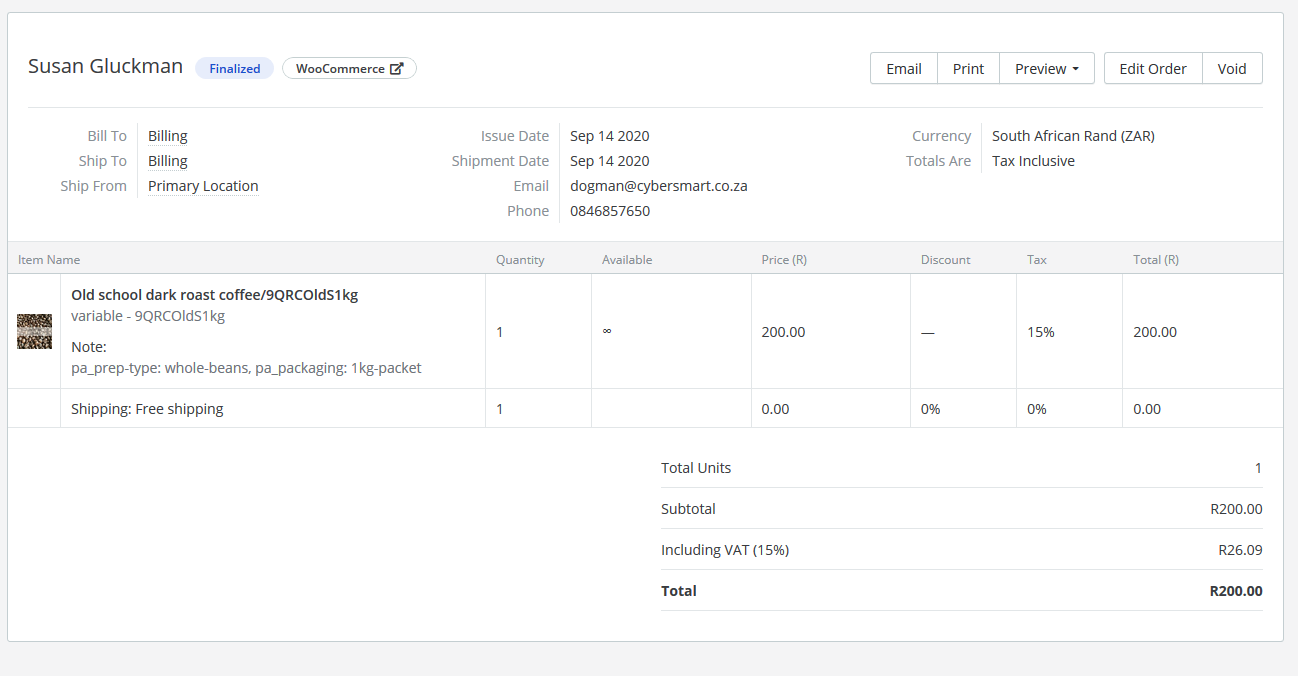
# WooSync Table

Stores the status of the synchronisation of Woo. Initially use for the import but also fired by the WooTriggers that we load. Used by the WooSyncRepository

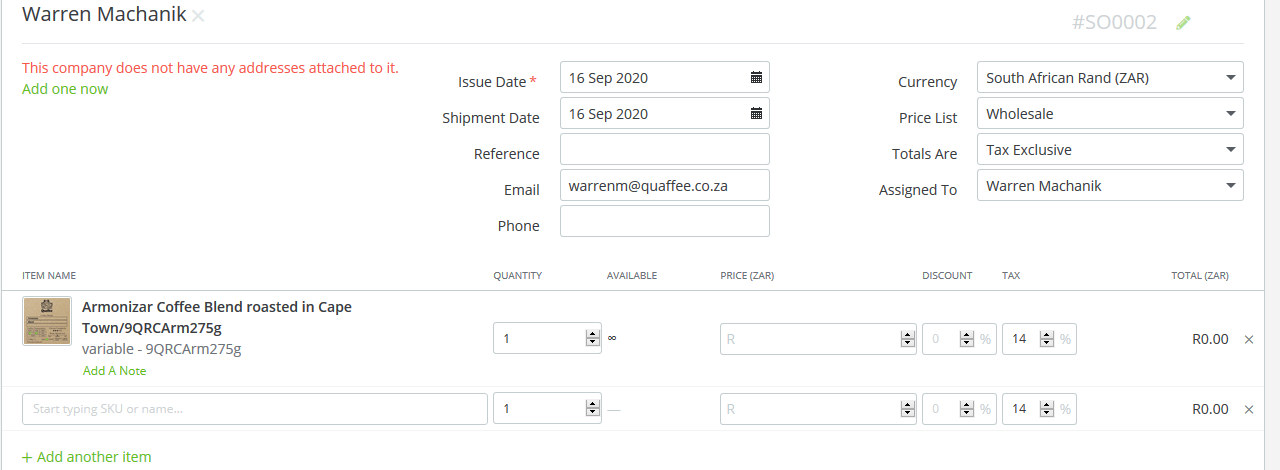
|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| WooSyncLogId | Int | PK |
| WooSyncDateTime | DateTime | The Date and time the sync was done |
| Section | Enum | Which section: Coupons, Customers, Order, OrderRefunds, Product, ProductAttributes, ProductAttributeTerms, ProductCategories, ProductTags, WebHooks, Taxes or System |
| SectionID | Int | Id if queried otherwise “0”. |
| Result | Enum | Success, Error, Timeout |
| Parameters | String | What was sent |
| Notes | String | Any notes |

# Orders

Orders have the notes in them that include the packaging and prep type variables:



You can add notes



Orders Status:

When order is closed then Rest call:

PUT - https://www.quaffee.com/wp-json/wc/v3/orders/21174?status=completed&consumer\_key=ck\_3b3fffb1d97832e4af0041104b1b839fde66dc02&consumer\_secret=cs\_06da46d3e7b3f0d62a501368afdae8f138cbb0a3

# Woo Sync Log

All interactions with Woo should be logged. They will be logged in the WooSyncLog table which will look like this.

# Woo Web Hooks

After the migration has happened, we need to ass web hooks, or modify them if they exist.

Hooks that need to be created:

* CRUD – products, and product settings like attributes and categories
* CRUD – orders and supporting tables
* CRUD – users/customers

# UI Messages

There are three type of message the system sends to the user:

1. Messages that require no user response
2. Messages that require use acknowledgment and
3. Messages that require a response

## No Response required

This type of message appears when something is done, or if we want the user to note something. For this type of message we will uses toasts. As implemented here: <https://github.com/Blazored/Toast>.

This will be the most common way of sending a message. Each time the message is sent it will also be logged. Using the ILogger service. So, this means that a Component will need to be written that combines the notifications with the logging.

The component will have a ShowMessage routine that will use an enum with the value (Info, Success, Warning, Error). Each of those enums will call the relevant log and toast message.

### Component:

Code Guide insert the component with a reference to the class, so it can be manipulated.

<PopUpAndLogNotification NotificationMessage="Message" NotificationTitle="" @ref="@PopUpRef" />

The message is what is displayed, and can be overwritten. If the title is left then it uses the message type. In the code you display the message using the ref:

PopUpRef.ShowNotification(NotificationType.*type,”message”,”title”*)

The type is as above message is what is to be displayed, title is optional.

## Messages that require acknowledgment

These have fallen outr of favour but could be used for errors. The component is added to the page and can be called with a ref and then the ref controls the call like above. It is essentially like a toast message but with an ok/cancel

## Messages that require a response

These are used when we need the user to confirm. For example a delete. The component is added to the razor file with a ref. A call via the ref can overwrite the question and acknowledgement buttons. The component merely sorts out the defaults.

### Component:

Code example:

<ConfirmModal @ref="DeleteConfirmation" ConfirmationChanged="ConfirmDelete\_Click" ConfirmationMessage="Are you sure you want to delete?") ConfirmButtonText="Delete" CancelButtonText="Cancel" />

Here the call sets the call back routine that accepts a true/false which is confirmed or not.

To call the modal you use the ref set and any variables you want to change (like the item name you want to delete):

DeleteConfirmation.ShowModal("Delete confirmation", $"Are you sure you want to delete: {*Item*}?");

If the user selects the confirmation button the ConfirmedClick routine is called with true, otherwise false.

# Attribute Lookup Page

The Attributes are displayed in a grid. If the attribute is selected the variations of the attribute are then displayed as a grid.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute Name | Varieties | S/Order | WooSync | Notes | [New] |
| Attrib1 | Var1,var2,var3 | 0 | Yes | Some notes | [Edit][Delete] |
| Attrib2 | Var4,var5 | 1 | No | Some other notes | [Edit][Delete] |

Varieties

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Variety Type | UoM | Symbol | FGColour | BGColour | S/Order | WooSync | Notes | [New] |
| Var1 | kg |  |  |  | 0 | Yes |  | [Edit][Delete] |
| Var2 | kg |  |  |  | 1 | No |  | [Edit][Delete] |
| Var3 | g |  |  |  | 2 | Yes |  | [Edit][Delete] |

As an attribute is select the varieties are displayed and can be edited? The attributes are edited in a modal since varieties can be added to them. The varieties table can only be edited in line. A new variety can be added via the modal edit of the attribute name.

# WooLinkedViews

There are a number of grids/views that will display woo linked items. This lends it self to creating classes and interfaces that do that are used to display (view) the data and perform the CRUD tasks.

## Display Classes

Included in the display call are the common elements that are use in each grid:

// Interface Stuff used for the grid -that is used by the interface

public int PageSize = 15;

public string customFilterValue { get; set; } = string.Empty;

public bool IsNarrow { get; set; }= true;

public bool IsFilterable { get; set; }= false;

protected ConfirmModal DeleteConfirmation { get; set; }

public PopUpAndLogNotification PopUpRef { get; set; }

These values are essentially a generic class used in setting up the grid and support it. For any page that uses a grid this class can be used.

## Woo Linked View

Each Grid should have a view model, that inherits the main data from the data.models (TEntity) but adds items that are needed to either include as part of the woo support or to display on the gird, like Foreground and background colours in a single column.

## CRUD

Using the Woo Linked view (TViewEntity). The crud operations for the grid that need to be

async Task<List<TEntity>> GetAllAsync(TEntity)

async Task<List<TEntityWooMap> GetWooMappedItemAsync(Guid wooTEntityID)

TViewEntity MapItemToWooItemAsync(TEntity)

async Task OnRowInsertedAsync(TVeiwEntity)

OnNewItemDefaultSetterAsync(TViewEntity)

// used for the grid new item

async Task<int> UpdateItemAsync (TViewEntity) => private?

async Task<int> UpdateWooMappingAsync(TViewEntity)

IsDuplicate(TEntity)

IsValid(TEntity)

async Task OnRowUpdatedAsync(TViewEntity)

OnRowRemovingAsync - > not sure about this

async Task ConfirmDeleteAsync(bool deleteConfirmed)

async Task<int> DoGroupActionAsync(ItemView)

Each of these will need to use the PopUpRef to log and the delete ref, so that will need to be passed in.

Woo Link?