# Product Information Management (PIM)

Table of contents

[Product Information Management (PIM) 1](#_Toc434217170)

[General processes and business rules 3](#_Toc434217171)

[Basic Product Management 5](#_Toc434217172)

[Product listing 5](#_Toc434217173)

[Product view 7](#_Toc434217174)

[Asset view / edit 9](#_Toc434217175)

[Product & properties editing 14](#_Toc434217176)

[Categories mapping 15](#_Toc434217177)

[Catalogue assignments 17](#_Toc434217178)

[Product workflow and product history 18](#_Toc434217179)

[Product Variants 22](#_Toc434217180)

[View variants 22](#_Toc434217181)

[Create a variant / clone product 23](#_Toc434217182)

[Base product handling 24](#_Toc434217183)

[Configure variants identifier 25](#_Toc434217184)

[Product Enhancements and Usability 27](#_Toc434217185)

[Related offerings 27](#_Toc434217186)

[Quick offering creation 27](#_Toc434217187)

[Quick search 27](#_Toc434217188)

[Advanced search 27](#_Toc434217189)

[SEO 27](#_Toc434217190)

[Related products 27](#_Toc434217191)

[Features 27](#_Toc434217192)

[Create/Update feature 27](#_Toc434217193)

[Assign features to products 27](#_Toc434217194)

[Product Translations 27](#_Toc434217195)

[Offering Management 27](#_Toc434217196)

[Basic Offering Management 27](#_Toc434217197)

[Offering listing 27](#_Toc434217198)

[Offering viewing 30](#_Toc434217199)

[Asset viewing / Editing 31](#_Toc434217200)

[Offering and properties editing 31](#_Toc434217201)

[Offering workflow and offering history 31](#_Toc434217202)

[Price List assignments 35](#_Toc434217203)

[Offering Variants 36](#_Toc434217204)

[View variants 36](#_Toc434217205)

[Create a variant / clone offerings 36](#_Toc434217206)

[Pricing / Pricelists 36](#_Toc434217207)

[Price lists listing 36](#_Toc434217208)

[Price list editing 36](#_Toc434217209)

[Bulk updates for price lists 36](#_Toc434217210)

[Offering Usability 36](#_Toc434217211)

[Price lists quick update 36](#_Toc434217212)

[Offering Translations 36](#_Toc434217213)

[Reference data management 36](#_Toc434217214)

[Workspace 36](#_Toc434217215)

[Caching 36](#_Toc434217216)

[General caching mechanism 36](#_Toc434217217)

[Exposing products details 36](#_Toc434217218)

[Exposing offerings (with pricelists) 36](#_Toc434217219)

[Extending exposed data 36](#_Toc434217220)

[CDN 36](#_Toc434217221)

[Offering API 36](#_Toc434217222)

[Product API 36](#_Toc434217223)

[Vouchers 36](#_Toc434217224)

## General processes and business rules

**General**

PIM stands for Product Information Management. This system is to be used to manage products, offerings, price lists and related information. The goal of the application is to be an advanced product management application that supports product variants, publishing workflows, ERP system synchronization and basic stock management. At the same time it will be merchandising tool that helps publishing offerings online.

The High Level Requirements are summarized by following table:

**Products**

Product is a definition of a saleable physical item or a service; however product can never be sold without wrapping it into an Offering.

Note: In SaaS model Products catalogues and product definitions are maintained only by a dedicated BPC team as product definitions are shared between clients and have to client agnostic.

Customization of a product per client can be than by using offerings when basic definitions and description can be overwritten.

A product has a relationship with an item in the eCommerce Core and OMS.

**Properties Templates**

Property Templates define the properties related to a specific type of an entity (product, offering etc). The template can be defined to specify properties that are required (or not), validation mechanisms and controls (future phase) to be applied e.g. a TV must have a resolution property, possible values are: 1900x1080, 1024x768 etc and the control type is a drop-down.

**Versions vs Variants**

Product versions track the history of changes to a product. A version can be either:

* Historical - a snapshot in time of a product and its related data.
* Draft – a working version of a product and its related data.
* Live – currently published version

Product variants are where similar base products can be grouped together on the basis that there are only certain attributes that distinguish it from the other variants. Examples of variants would be:

iPhone 6

* Silver 32GB
* Silver 64GB
* Space Grey 32GB
* Space Grey 64GB

T-Shirt

* White S, M, L, XL
* Black S, M, L, XL

**Media asset management**

Media asset management provides the ability to upload and manage the assignment of media assets (e.g. images, videos) to products and offerings.

Media formats, types and sizes can all be configured per site.

**Offerings**

Offerings are the way in which products and services are sold. Anything that is sold will be an offering. An offering must contain one or many products and services. An offering has at least one price and therefore belongs to at least one pricelist.

Offerings contain a primary product (i.e. the main product being sold). Offerings can use descriptive information related to the primary product (e.g. description, specification properties, images etc) or override this information.

An offering has a relationship with a package in the eCommerce Core and OMS.

**Catalogues vs Pricelists**

Catalogues are related to products. A catalogue contains the products that are available to a specific client, channel or site. The catalogues that are available to a specific user are controlled by the users permissions (users are granted access at client, channel, site levels). If a user can only see one catalogue, all catalogue related operations and views should default to that catalogue

Pricelists are related to offerings.  A pricelist is a set of offering prices that can be utilised in a number of different ways, a price list has is one of the following types:

1. STANDARD – Used for all normal pricing for a given site
2. PROMOTIONAL – used to create sets of pricing for promotional prices

These types may be extended in the future but would most likely be related to a new feature and therefore require development.

A pricelist can have a start and end date set to schedule a set of price changes.

## Basic Product Management

### Product listing

**Overview**

Product listing page should allow to display list of products filtered according to specific criteria. Listening page will work in classic way. After the user selects filters and clicks the search button a list of products that matches the filters will be displayed.

**Business Rules**

Filtering criteria:

* **Catalogue** - dropdown with hierarchy of catalogues that user has permissions to with an option to add new one + option to view all products within catalogues that user has
* **Name –** Product name – free text min 3 characters. Search product.name field
* **Manufacturer –** Multi-select searchable drop down
* **SKU –** Product code - free text – searches product.code field
* **Type –** Product type hierarchy drop down
* **Category** – Product category hierarchy drop down
* **Status** – Product status dropdown – default should be only ‘LIVE’
* **Created –** Date selector to filter product created after selected date.

Products should be displayed only if user has access to at least one catalogue that the product is in or has “product-admin” or “admin-pim” roles.

**Suggested approach**

This will be a classic page where filters where results are displayed after Search button is clicked.

Complicated part if the catalogue permission filtering: To determine if the user has access to a catalogue we need to use ApplicationSecurityService to determine allowed clients / channels / sites and list catalogues based on that. This method has to be used for both – displaying the catalogue hierarchy and filtering the visible products.

**Database operations**

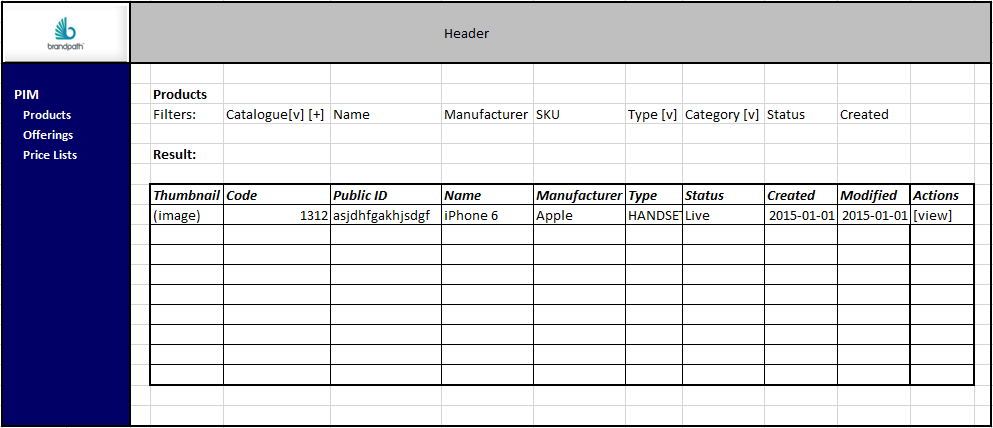
Tables that will be used:

For catalogue operations: catalogue.catalogue, catalogue.product

For filters: product.category,product.type,product.status,product.manufacturer

For results: product.product,media.asset,media.assetSetProduct,media.assetSet (+ all above)

**Mockups**



### Product view

**Overview**

Product page shows the detail of one particular product selected from product listing. Data about product is grouped in the tabs. Screen should be designed with a view that data on this view will be editable. The product view page is driven by product version data, the page should be capable of displaying any version in any state. The actions available on the page should be dependent on the current status and permission level of the current user.

**Business Rules**

List of tabs:

* General - Display basic product information (types, categories, thumbnail) and description
* Details – Detailed properties describing product
* Assets – Media asset management
* Variants & Related products – Tab used to manage related products
* Translations – Tab to manage translations of the product to other languages
* SEO – Meta tags and related
* Merchandising – list of offerings, up-sells, cross-sells
* History – history of product changes

**Suggested approach**

Product page should be a normal page (not a modal window) and should display the data grouped by tabs. PF components should be used for tabs (as per OMS Order Details screen). Before the page is viewed there should be a check if a person has access to this product (see product listing logic). While building it the fact that this screen will be editable should be taken into consideration (see Product Edit section)

General tab:

Data loaded from version tables in the database. Status change button(s) should be displayed as per the workflows described in “Product workflow and history”. Catalogue and Category Mappings will be described later in the document. Description should be shown but edit box should be greyed out.

Details:

Details tab should display properties of the product in two columns grouped by Property Type as per mock-up. It should handle situations that some properties can have multiple values.

All others tabs will be described in subsequent sections of the documents.

**Database operations**

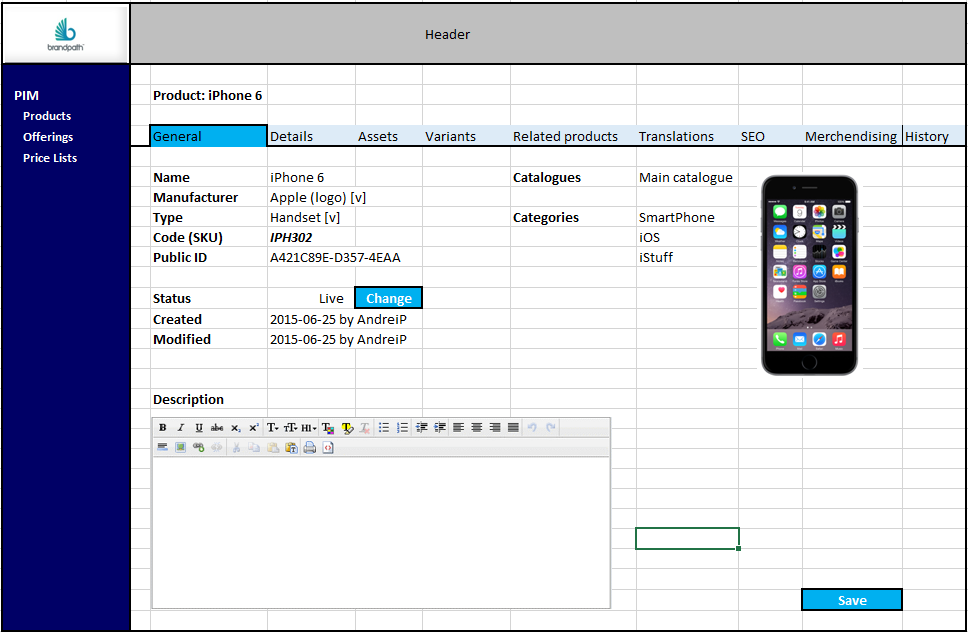
For general tab: catalogue.catalogue, catalogue.product, product.product, product.version,product.category,product.type,product.status,product.manufacturer

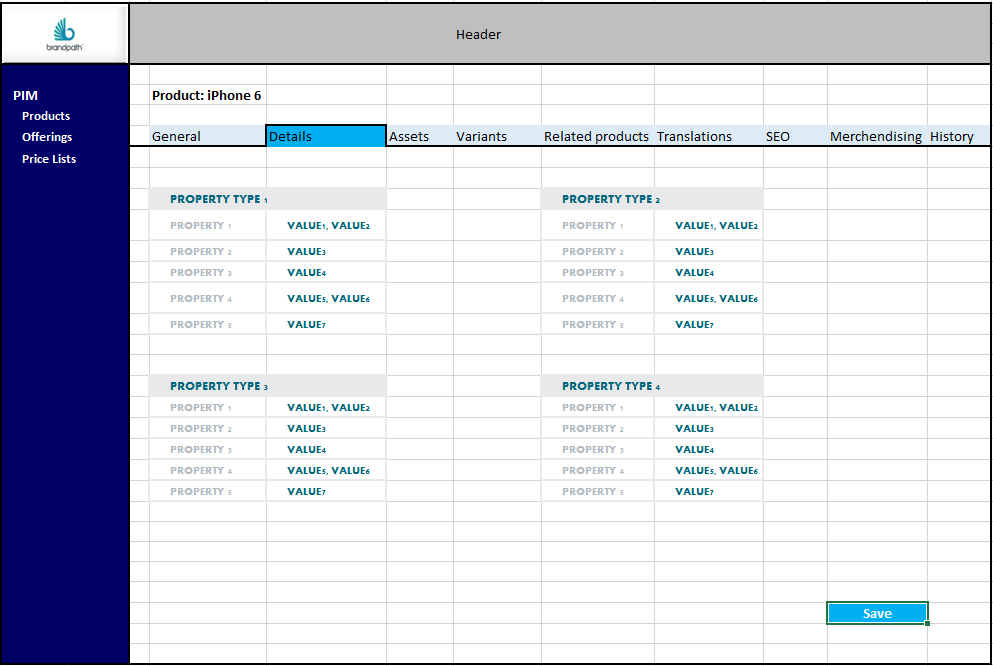
For image : media.asset,media.versionSetProduct

For details tab: product.property, product.propertyType, product.propertyValue, template.productProperty

**Mockups**

Save button should only be visible if the product is not in “LIVE” status.





### Asset view / edit

**Overview**

Media assets are grouped into sets, for example a set may contain images of different sizes representing the same view of a product e.g. Front-view of a mobile device in Thumbnail, Standard and Large images

**Business Rules**

Versioning / History

* All assets/images should be editable when there is a product.version with ‘DRAFT’ status.
* When a DRAFT is created what is in fact created is a new ‘version’ with DRAFT status
* The main set, asset and mapping tables contain only the ‘PUBLISHED’ or ‘RETIRED’ assets/sets

Uploading Assets

* The file attributes must be validated when uploaded to the temporary directory:
  + Dimensions – should be greater than or equal to image dimensions configured and the same ratio
  + Mime Type – must be valid for the current product type

**Future Proofing**

Version 1 will not include translations for media/assets but they will use the generic complex translations mechanism as per all other complex objects

**Suggested approach**

* The view / edit assets page can be the same actual page with the current user’s permissions determining the actions they can take.
* The function to add a new asset should use a popup/modal
* For version1 the media set status should be kept sync with the product status.

The MediaManager should contain methods to manage each state change:

* create() – The following operations should be performed:
  + creates a version (index = 0) of media.set in a DRAFT mode, mandatory parameters are:
    - media type (from media.type)
    - originalFileName – name of the original file uploaded
    - name
  + Auto-generated fields are:
    - Created
    - Modified
    - Id
    - sourceFileName – generated name of the original file now saved to source file location
  + Generate individual assets based on [media].productTypeSize configuration; generate files in PIM media storage location
  + For each individual asset create a record in media.asset and associate via media.versionAsset mapping table to the this version
* publish() - This should perform following operations:
  + validate the DRAFT version
  + overwrite or create the PUBLISHED (media.set & media.setProduct) data with the draft data
  + update media.asset.setId to reference the media.set record
  + update the version status to be ‘PUBLISHED’
  + update the cache (to be discussed later)
  + Copy assets to CDN/NFS
* enableEdit()
  + this should create a new DRAFT version (index+1) and copy all the data from PUBLISHED [media].[set ]
  + update media.set.*draftVersionId* column to point to new version
  + associate the media.asset records with the new version (media.versionAsset)
* abortEdit()
  + change the DRAFT status of version to ABORTED
  + update media.set.*draftVersionId* column to *NULL*
* retire()
  + change the PUBLISHED status to RETIRE
  + update the cache (to be discussed later)
  + Remove assets from CDN/NFS
  + Remove reference to media.set from media.asset (nullify)

**Database operations**

Version data needs to be saved to: media.versionSet, media,versionAsset, media.versionSetProduct

Data needs to be saved to: media.set, media.asset, media.setProduct

Data used to validate and drive the page:

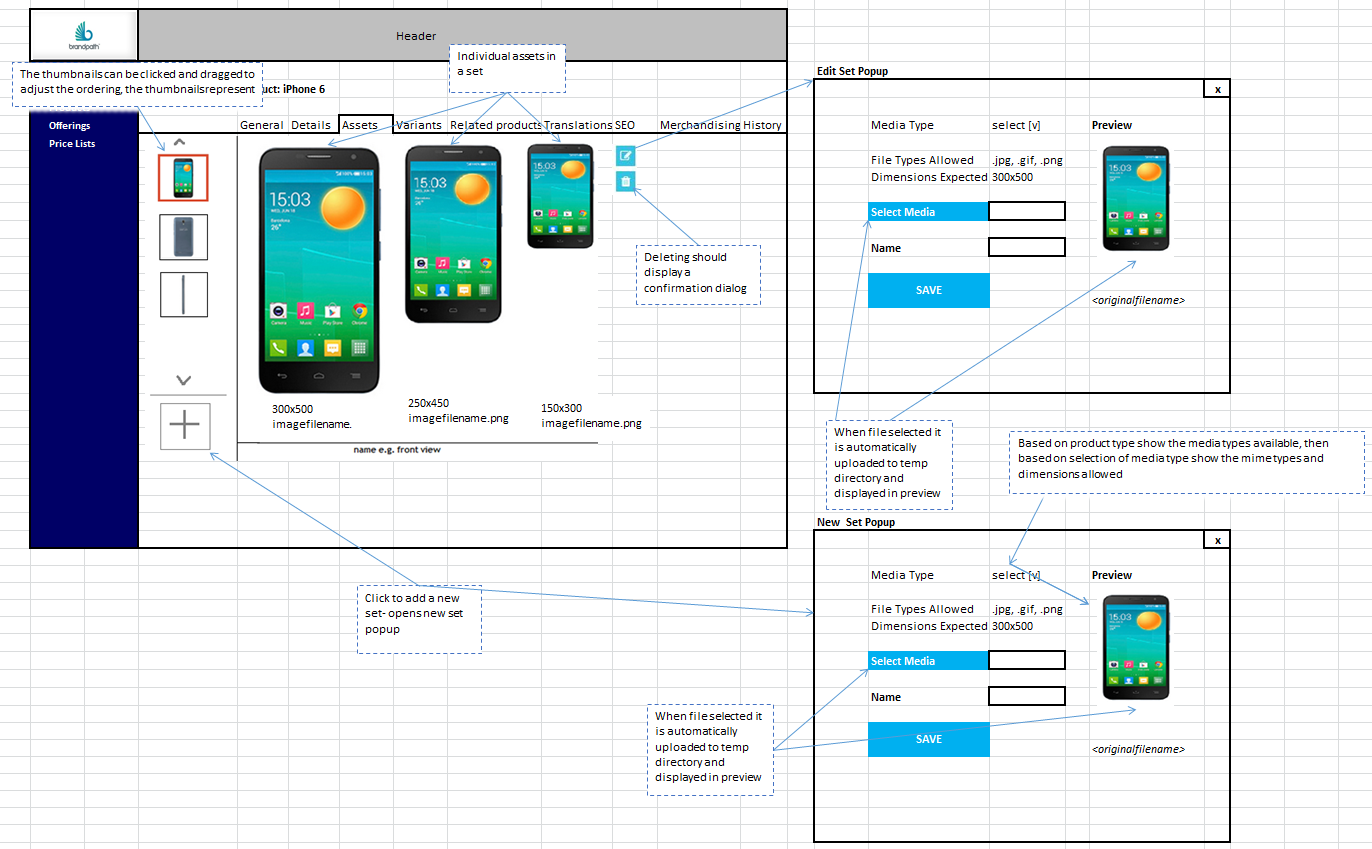
media.productTypeSize – defines the width, height, sizeType (e.g. Thumb, Standard etc), media type (e.g. Product Image)

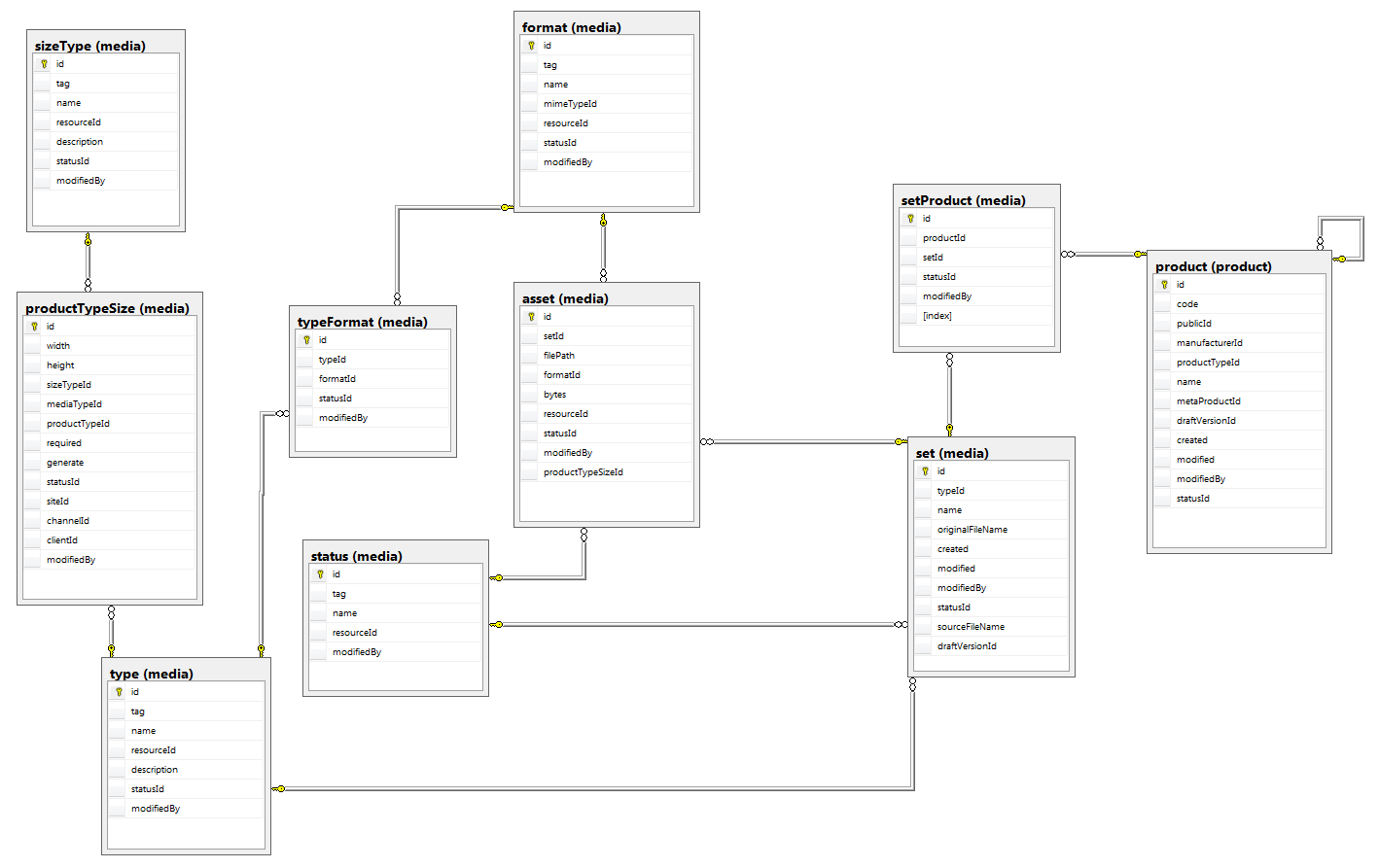
media.type,

COMMONS.reference.mimeType

**Mockups**

[see next page]



**Data Model**

### Product & properties editing

**Overview**

Product edit pages should allow edition of all of the data on the relevant tab if the product is in “DRAFT” mode. All the controls should change to be editable (edit boxes, drop downs). To approve the changes user should click the Save button to confirm the changes.

**Business Rules**

If a product was PUBLISHED already there will be set of rules that restricts on what we can do with a product i.e. you can’t change the SKU code or public ID, more rules might follow up.

**Suggested approach**

When the product is In DRAFT more all relevant controls should become editable and “SAVE” button should be visible.

There should be a Product Manager EJB service that will be responsible for save operation.

When product is In DRAFT mode properties in Details tab needs to show relevant editable controls (i.e. Drop downs, Calendar type, masked inputs for Regex validations – taking into account template.productProperty table). Data should be validated before saving.

The lifecycle of property value data is defined in ‘Product workflow and product history’

**Database operations**

Data needs to be saved to: product.product, product.propertyValue.

Data for versions to be saved to: product.version, product.versionPropertyValue

**Mockups**

As per Product View

### Categories mapping

**Overview**

Products can be mapped to categories in two ways:

1. Default categories – based on industry standard category naming and assignment
2. Catalogue specific categories – where a client wishes to use their own category naming and assignment either based on their business or a channel within their business – NOT IN SCOPE FOR PHASE1

Categories are hierarchical and have a type.

**Business Rules**

Category mappings can only be edited if the product is in ‘DRAFT’ mode.

When editing category mappings the mappings are stored in product.versionCategoryMapping and associated with the current ‘DRAFT’ product version.

When the ‘DRAFT’ product version is PUBLISHED the category mappings associated with the version will be copied to product.categoryMapping – the new mappings must completely replace any existing mappings but the logic should be optimised so any mappings that are consistent between versions are maintained and therefore there will be no unnecessary overheads.

**Suggested approach**

If there is only one catalogue in use the category assignment can be done directly in the product page. If there are more catalogues in use then the use of a popup/lightbox may be needed to provide space for the controls and also to ensure the main product page does not become cluttered.

**Database operations**

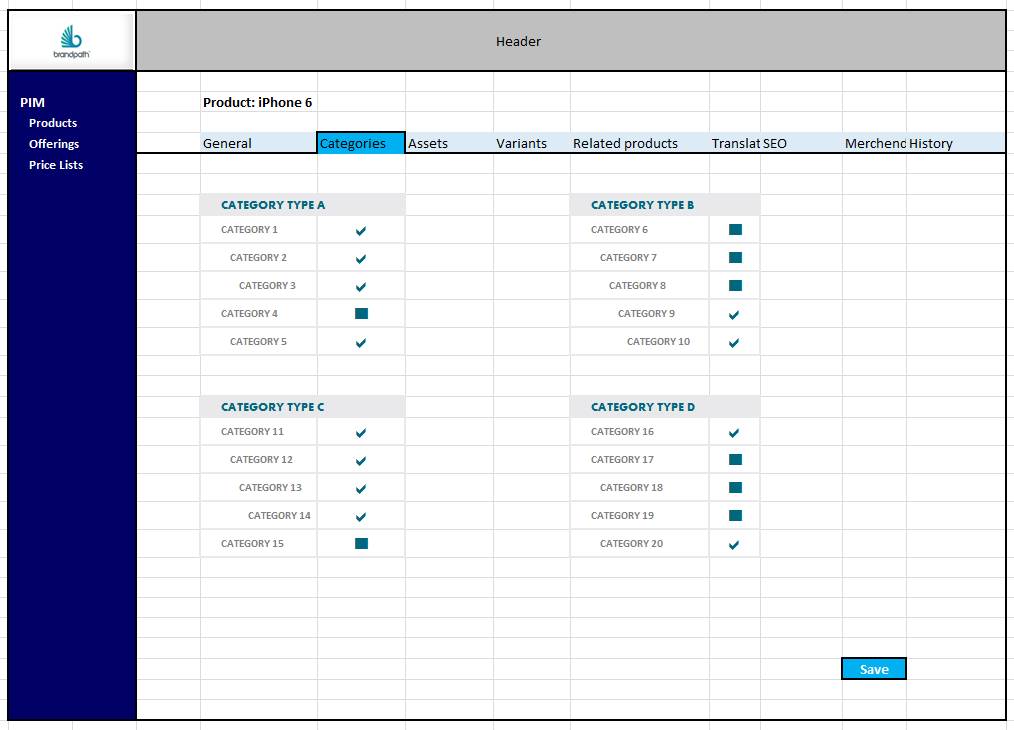
Default category mappings are stored in**:** product.categoryMapping

Version data is stored in product.versionCategoryMapping

Catalogue specific category mappings are stored in: catalogue.productCategoryMapping – NOT IN SCOPE FOR PHASE 1

Category details (name, tag etc.) are stored in: product.category, product.categoryType

**Mockups**



### Catalogue assignments

**Overview**

Catalogue assignment will be available based on the current user’s permissions at client / channel / site level.

**Business Rules**

Products can be assigned to multiple catalogues

Catalogue assignment will be used to determine which products a user can view and use, for example assigning to offerings.

**Suggested approach**

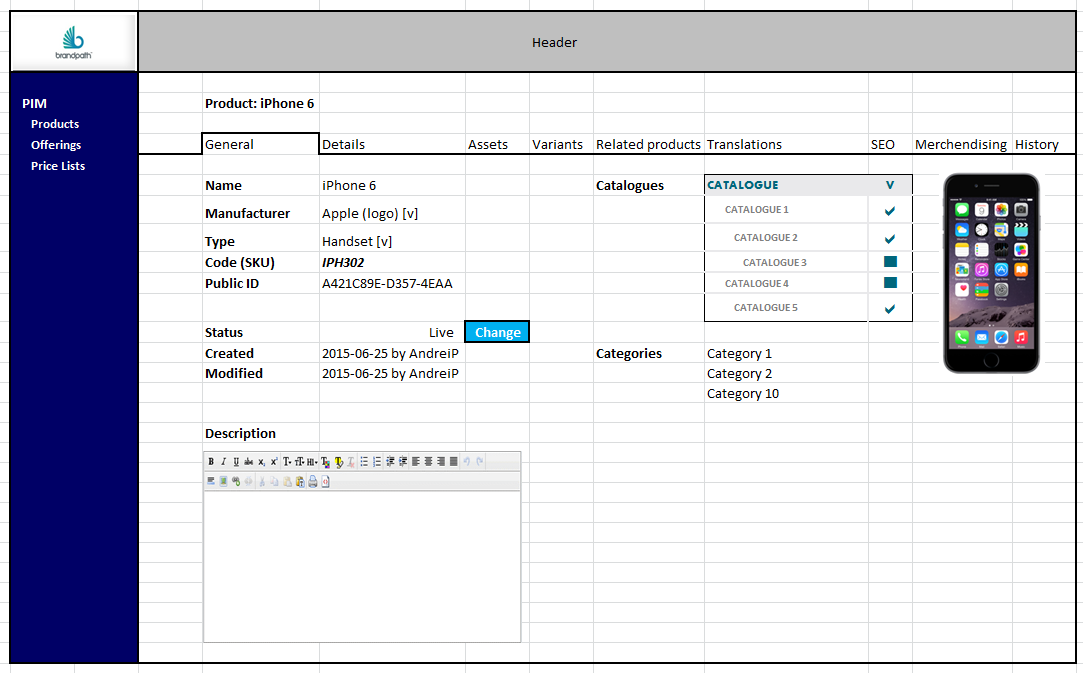
Products can be assigned to catalogues in two different ways:

1. Via individual product details page
2. Via Catalogue product listing page (NOT IN SCOPE FOR PHASE 1)

**Database operations**

Catalogue assignments are stored in**:** catalogue.product

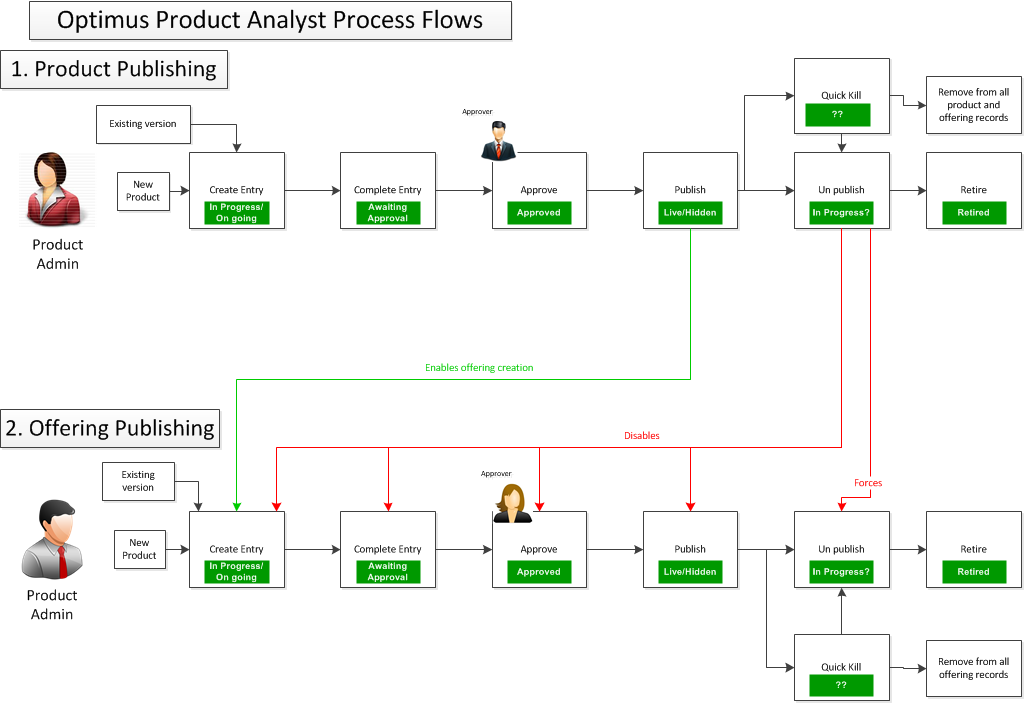
**Mockups**



### Product workflow and product history

**Overview**

Creating a product doesn’t mean it can automatically be used in offerings. Each product before it will be visible on the site needs to go through a pre-defined flow:



In future product publishing flows will be customizable and it will be controlled by BPM engine. However for now the flow will be controlled manually and will be limited to the below:



Each of the transition between states will result in different set of operations (see below for details). Some of the transition will require to save/update a version of the product.

**Business Rules**

Above diagrams reflect the flow that the product can go through. Product can remain in one of the following status:

* PUBLISHED
* RETIRED
* DRAFT (only for a version)
* ABORTED (only for a version)

Newly created product until published is a “version” and status of that version is set to DRAFT. After editing is completed Publisher can move the product to PUBLISHED status. Since that time product can be used to create offerings.

If editing of a product is needed Product Editor can move the product back to DRAFT status, but what does in fact is create another “version” of the product. Product remains PUBLISHED as it is already used (potentially) by offering. Once the DRAFT version is ready it can be PUBLISHED and it will override the existing PUBLISHED version. History will retain in the database for reference.

**Suggested approach**

Each of the status transformation should be a separate method in ProductManager:

* create()
  + An dummy (empty) product record should be created
  + Product version (index = 0) created in DRAFT mode referencing the dummy product
  + The only mandatory parameters are name, manufacturer, and product type, so the template for the properties can be displayed (publicId will be auto generated).
* publish()
  + validate the DRAFT version
  + overwrite the PUBLISHED data with the draft
  + update the version data
  + update the cache (to be discussed later)
  + update *draftVersionId* column to *NULL*
* enableEdit()
  + this should create a new DRAFT version (index+1) and copy all the data from PUBLISHED product.
  + update *draftVersionId* column to point to new version
* abortEdit()
  + change the DRAFT status of version to ABORTED
  + update *draftVersionId* column to *NULL*
* retire()
  + change the PUBLISHED status to RETIRE
  + retire all offerings that a product is used by
  + retire all media sets related to the product that are not assigned to any other product or offering
  + update the cache (to be discussed later)

Those operations need to be visible as buttons against a status field (see CHANGE button on mockups for Product View) on the main product tab. So for the particular statuses:

* DRAFT – There should be a button with Publish, Rollback (available only if “a” PUBLISHED version exist)
* ABORTED – this is to indicate DRAFT version that was rolled back. No buttons available and whole is read only.
* PUBLISHED – There should be a button with Edit label (That calls the enableEdit method) and Retire button (that call Retire())

Please bear in mind that this process in future will be extended heavily.

**Database operations**

Depending of the status/operation updating can happen to either “product” tables:

product.product, product.categoryMapping, product.propertyValue

or “version” tables:

product.version,product.versionCategoryMapping,product.versionPropertyValue

Structure of the tables is very similar. All operations on “DRAFT” versions are done against “version” tables, “product” tables are used to store published version.

Below table illustrates a typical create-publish-edit-publish flow:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Step 1** | **Create Product** | |  |  |  |
|  |  |  |  |  |  |
| **product.product** | |  | **product.version** |  |  |
| *productName* | *statusId* |  | *productName* | *statusId* | *versionId* |
| (no data) | |  | Phone | DRAFT | 1 |
|  |  |  |  |  |  |
| **Step 2** | **Publish product** | |  |  |  |
|  |  |  |  |  |  |
| **product.product** | |  | **product.version** |  |  |
| *productName* | *statusId* |  | *productName* | *statusId* | *versionId* |
| Phone | PUBLISHED |  | Phone | PUBLISHED | 1 |
|  |  |  |  |  |  |
| **Step 3** | **Edit product** | |  |  |  |
|  |  |  |  |  |  |
| **product.product** | |  | **product.version** |  |  |
| *productName* | *statusId* |  | *productName* | *statusId* | *versionId* |
| Phone | PUBLISHED |  | Phone | PUBLISHED | 1 |
|  |  |  | Mobile phone | DRAFT | 2 |
|  |  |  |  |  |  |
| **Step 4** | **Re-publish product** | |  |  |  |
|  |  |  |  |  |  |
| **product.product** | |  | **product.version** |  |  |
| *productName* | *statusId* |  | *productName* | *statusId* | *versionId* |
| Mobile phone | PUBLISHED |  | Phone | PUBLISHED | 1 |
|  |  |  | Mobile phone | PUBLISHED | 2 |

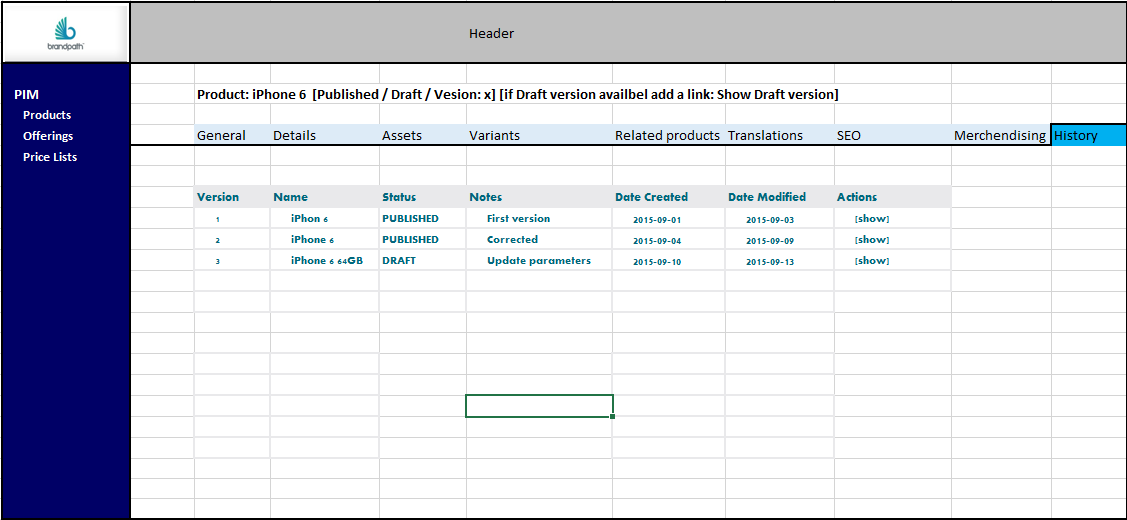
When product is created (Step 1) data will be stored in “version” tables. In Step 2 which is Publish operation status is updated in the “version” table and data is created/moved to product database. When product is then edited “DRAFT” (Step 3) entry needs to be created in “version” tables (no changes to “product” tables). Once the product is published again (Step 4) data should be copied from “version” to “product” tables as well as status updated to “PUBLISHED” (similar to Step 2).

Also product listing page needs to be updated to look through “version” tables when “DRAFT” status is selected.

**Mockups**

Next to the Product name in header there should be information of the currently visible version (i.e. Published / Draft / Version: xx). If the Product is PUBLISHED and Draft version is available a button [Show Draft version] should be visible next to the header and it should load the DRAFT version into the page. If a version other than the currently PUBLISHED version is currently loaded there should be a link [Show Published version].

Particular version of the product should be displayed in History tab in a table as per mockup. After clicking a [show] icon relevant version should be loaded into all tabs.



**Overview**

**Business Rules**

**Suggested approach**

**Database operations**

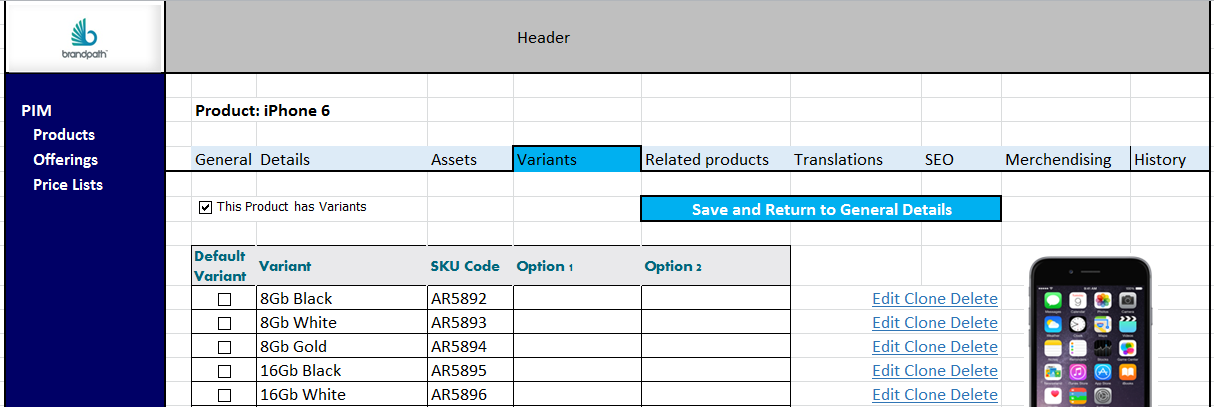
**Mockups**

## Product Variants

### View variants

**Overview**

Product Variant is one of the main components of PIM (Product Information Management). Product variants can be accessed in PIM 🡪 Products Listing 🡪 Product Details 🡪 Variants tab. Centralized data management is particularly well-suited for company websites, as documents, content and media objects such as product images can be linked with other business objects such as products.



**Business Rules**

It is considered an eCommerce best practice to create a single product with variants whenever possible since this creates ease of product management. Using variants prevents user from ending up with long lists of obsolete products, duplicated or misaligned products.

It is designed to assign images and descriptions to variants which are then used on customer receipts and displayed as sub items underneath the chosen top level product.

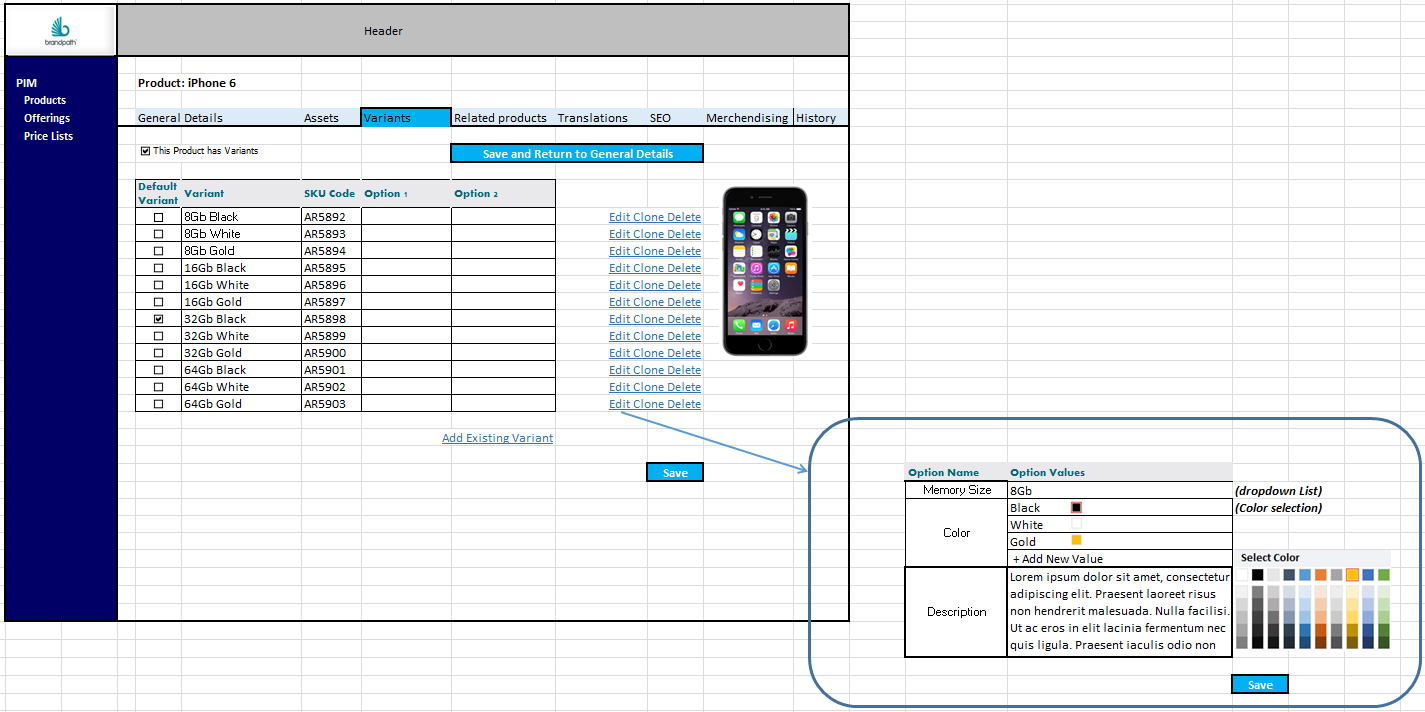
Different variants of same product could have different prices, so variants will be used to create different offerings.

A default Variant can be selected by user in order for this variant to be, for example, the first variant that appears on the website Product Listing page and Product Details page.

**Suggested approach**

**Database operations**

**Mockups**



### Create a variant / clone product

**Overview**

Any product can have variants to distinguish a product's different styles. If user does not have variants when creating the product, user can always add them later through PIM. Users will be able to create variants by adding new variants or cloning existing ones.

*Please see Assets section on instructions how to create or delete Option Names and Option Values.*

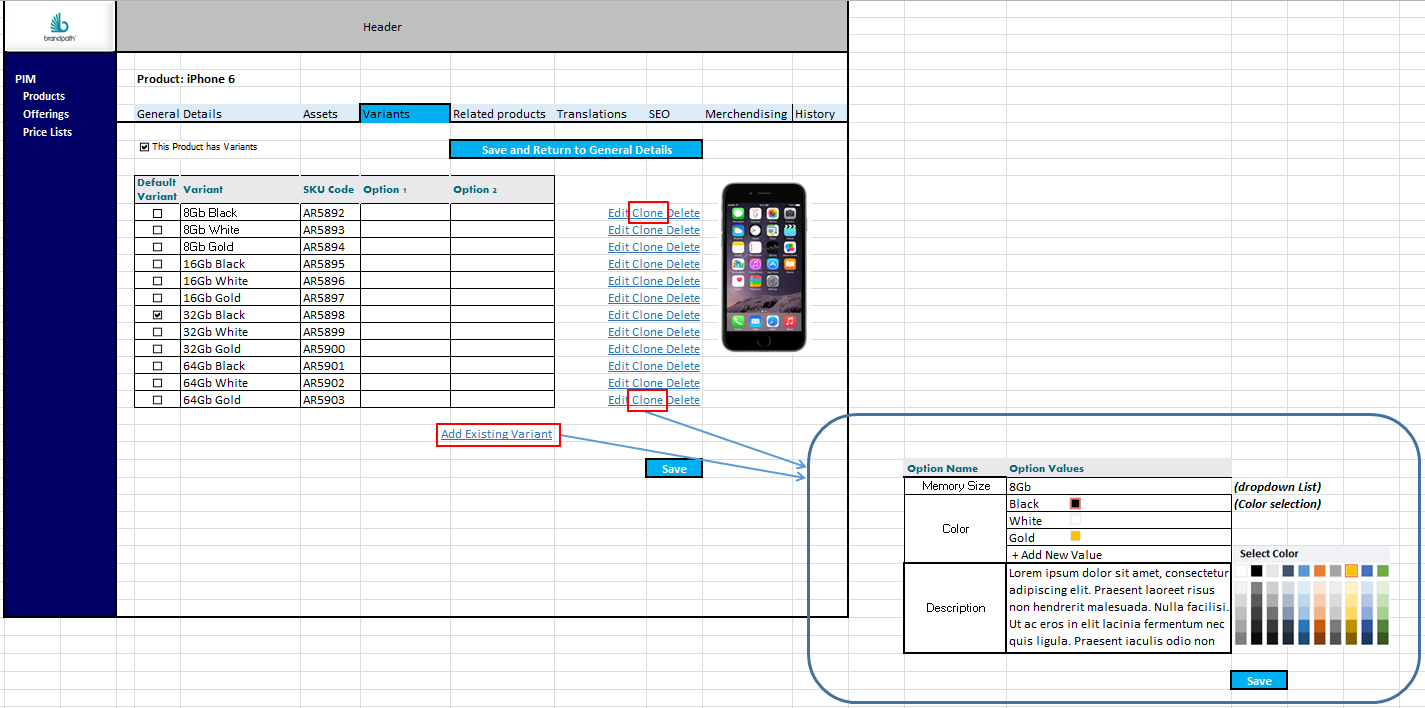
**Business Rules**

Flexibility is needed when handling products for different domains and with different properties. As products change and evolve, users will need to be able to add new variants to the products, in order to be on top of competition. To fasten the process of adding new variants on the online market, users will have the ability to add variants directly in PIM, instead of raising a ticket to development team. Also, to ease the process of adding variants and reduce the chances of user input error, cloning option will be available in PIM Variants section for each product.

**Suggested approach**

**Database operations**

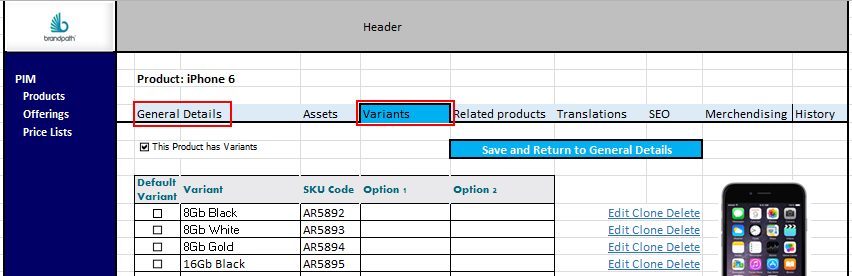
**Mockups**



### Base product handling

**Overview**

Base product handling and variant handling need to be connected to reduce the time of manual input and reduce the possibility of user error. Variants will copy most of the Base Product details automatically when created. User will be able to modify certain fields (eg: Description, SKU code).



**Business Rules**

Because it is used by multiple applications and the data is copied to all variants, product data needs to be flawless. An error in data can cause errors in all the applications that use it. For example, an incorrect name in the customer master might mean orders, bills to return, search engines not finding the correct product on the website and other issues.

General Details from each product are copied to each new variant created except for:

* SKU Code - will remain blank on the new variant.
* Default Variant option - will remain blank on the new variant.

eg: if user clones Default Variant, the new variant will have blank SKU Code and will not be default Variant. A product can only have one default variant.

In addition users need to be able to modify copied data like:

* Description – a different variant of same product might have different properties than the base product (eg: a Small T-Shirt can have slightly different material composition than a XXL T-Shirt).

**Suggested approach**

**Database operations**

**Mockups**

### Configure variants identifier

**Overview**

Admin Page will allow users to modify Option Names and Option Values for Variants for each Product Type.

**Business Rules**

In order to ease the product management, users should be able to modify the Variant Option Names and Values in a centralized way for each product type. For example, Accessories product type and Handsets product type will have different Option Names and Option Values:

* Accessories catalogue:
  + Item Weight
    - Free Text
  + Item model
    - Free Text
  + Noise Cancelling:
    - Yes
    - No
  + Integrated Mic:
    - Yes
    - No
* Handsets Catalogue:
  + Memory Size:
    - 16Gb
    - 32GB
    - 64GB
  + Color:
    - Black
    - White
    - Gold
  + Operating System:
    - IOS
    - Android
    - RIM (Blackberry)
  + Rear Camera
    - 5Mp
    - 8Mp
    - 12Mp
  + Front camera
    - No
    - 0.3 Mp
    - 1.2 Mp
  + Accelerometer
    - Yes
    - No

**Suggested approach**

**Database operations**

**Mockups**

## Product Enhancements and Usability

### Related offerings

### Quick offering creation

### Quick search

### Advanced search

### SEO

### Related products

## Features

### Create/Update feature

### Assign features to products

## 

## Product Translations

**Overview**

Product translations allows to translate product details in order to display product in multiple language. Areas that can be translated:

* Product Name
* Product Properties (only configured as translatable)
* Assets (future)

**Business Rules**

Translations should be enabled only to the languages specified for client of logged in user.

Base language is the main language of the system.

Translations should support versioning and be editable only when product is in “edit” mode (see product workflow).

**Suggested approach**

Products translations should be stored using the generic complex translations service provided in the translations module.

Translations are stored in a serialised XML format, products require the following XML data to be stored (suggestion):

<product>

<id>*id*</id>

<name>*name*</name>

<properties>

<property>

<id>*id*</id>

<value>*value*</value>

</property>

…

</properties>

</product>

A JAX-B annotated class should be created to represent this

## 

**Database operations**

Translations for version should be stored in the i18n.complex table against ProductVersionTranslation type. Once the product is published by user the translations should also be published (updated) to i18n.complex table against ProductTranslation type.

**Mockups**

## 

## Offering Management

### Basic Offering Management

### Offering listing

**Overview**

Offering listing page should display a list of products filtered according to specific criteria. When the user selects filters and clicks the search button a list of offerings that matches the filters chosen will be displayed.

**Business Rules**

Filtering criteria:

* **Pricelist** - dropdown with hierarchy of pricelists that user has permissions to with an option to add new one + option to view all offerings within pricelists that user has
* **Name –** Offering name – free text min 3 characters. Search product.name field
* **Manufacturer –** Multi-select searchable drop down (search manufacturers of products within offering)
* **Code –** Offering.code - free text – searches offering.code field
* **Type –** Offering type hierarchy drop down
* **Category** – Product category hierarchy drop down (search categories of products within offering)
* **Status** – Offering status dropdown – default should be only ‘LIVE’
* **Created –** Date selector to filter offering created after selected date.

Offerings should be displayed only if user has access to at least one pricelist that the offering is in or has “offering-admin” or “admin-pim” roles.

Results List:

* Thumbnail
* Code
* Name
* PublicId
* Price (minimum price)
* Type
* Status
* Created (date)
* Modified (date)
* Products quick view link
* Actions column (with view button)

The product quick view link should open a model with details of the products contained within the offering. The details displayed for each product should be:

* Thumbnail
* SKU (code)
* Name
* Type
* View link (should take the user to the view product details page for the chosen product)

**Suggested approach**

This will be a classic page where filters where results are displayed after Search button is clicked.

The complex part is the pricelist permission filtering: To determine if the user has access to a pricelist we need to use ApplicationSecurityService to determine allowed clients / channels / sites and list pricelist based on that.

**Database operations**

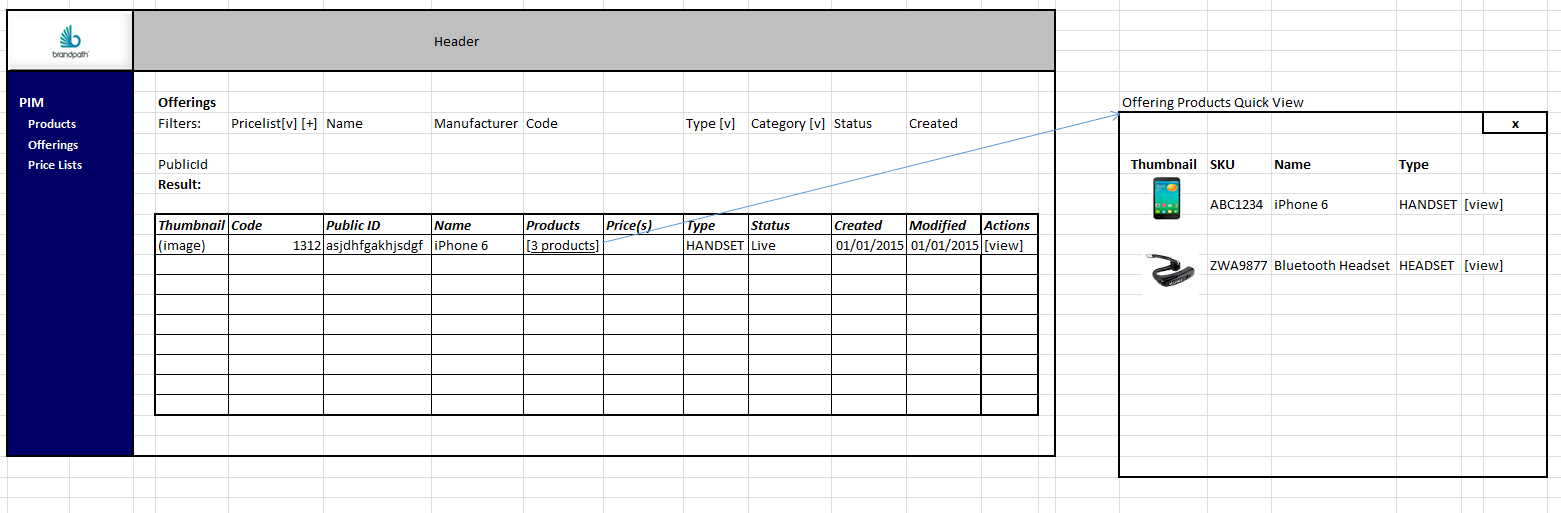
Tables that will be used:

For pricelist operations: offering.pricelist, offering.price

For filters: product.category,offering.product,offering.type,offering.status, offering.manufacturer,offering.offering

For results: media.asset,media.assetSetOffering,media.set (+ all above)

**Mockups**



### Offering viewing

**Overview**

Offering page shows the detail of one particular offering selected from offering listing. Data about offering is grouped in the tabs. Screen should be designed with a view that data on this page will be editable.

**Business Rules**

List of tabs:

* General - Display basic offering information (types, thumbnail) and description, price and pricelist information
* Details – Detailed properties describing offering
* Assets – Media asset management
* Variants
* Translations – Tab to manage translations of the offering to other languages
* SEO – Meta tags and related
* Merchandising – list of offerings, up-sells, cross-sells
* History – history of offering changes

**Suggested approach**

Offering page should be a normal page (not a modal window) and should display the data grouped by tabs. PF components should be used for tabs (as per OMS Order Details screen). Before the page is viewed there should be a check if a person has access to this offering (see offering listing logic). While building it the fact that this screen will be editable should be taken into consideration (see Offering Edit section)

General tab:

Data loaded from database. Status change button should remain inactive – it will be defined later in the document. Pricelists and Prices will be described later in the document. The description should be shown but edit box should be greyed out.

Details:

Details tab should display properties of the offering in two columns grouped by Property Type as per mock-up. It should handle situations that some properties can have multiple values.

All others tabs will be described in subsequent sections of the documents.

**Database operations**

For general tab: offering.pricelist, offering.price, offering.offering,,offering.type,offering.status

For image : media.asset,media.assetSetOffering,media.set

For details tab: offering.property, offering.propertyType, offering.propertyValue, template.offeringProperty

### Asset viewing / Editing

As per product asset viewing / editing but linking to offering via media.setOffering and media.verionSetOffering

### Offering and properties editing

**Overview**

Offering edit pages should allow edition of all of the data on the relevant tab if the offering is in “DRAFT” mode. All the controls should change to be editable (edit boxes, drop downs). To approve the changes user should click the Save button to confirm the changes.

**Business Rules**

If an offering was PUBLISHED already there will be set of rules that restricts on what we can do with a offering i.e. you can’t change the SKU code or public ID, more rules might follow up.

**Suggested approach**

When the offering is In DRAFT more all relevant controls should become editable and “SAVE” button should be visible.

There should be an Offering Manager EJB service that will be responsible for save operation.

When offering is In DRAFT mode properties in Details tab needs to show relevant editable controls (i.e. Drop downs, Calendar type, masked inputs for Regex validations – taking into account template.offeringProperty table). Data should be validated before saving.

The lifecycle of property value data is defined in ‘Offering workflow and offering history’

Database operations

Data needs to be saved to: offering.offering, offering.propertyValue.

Data for versions to be saved to: offering.version, offering.versionPropertyValue

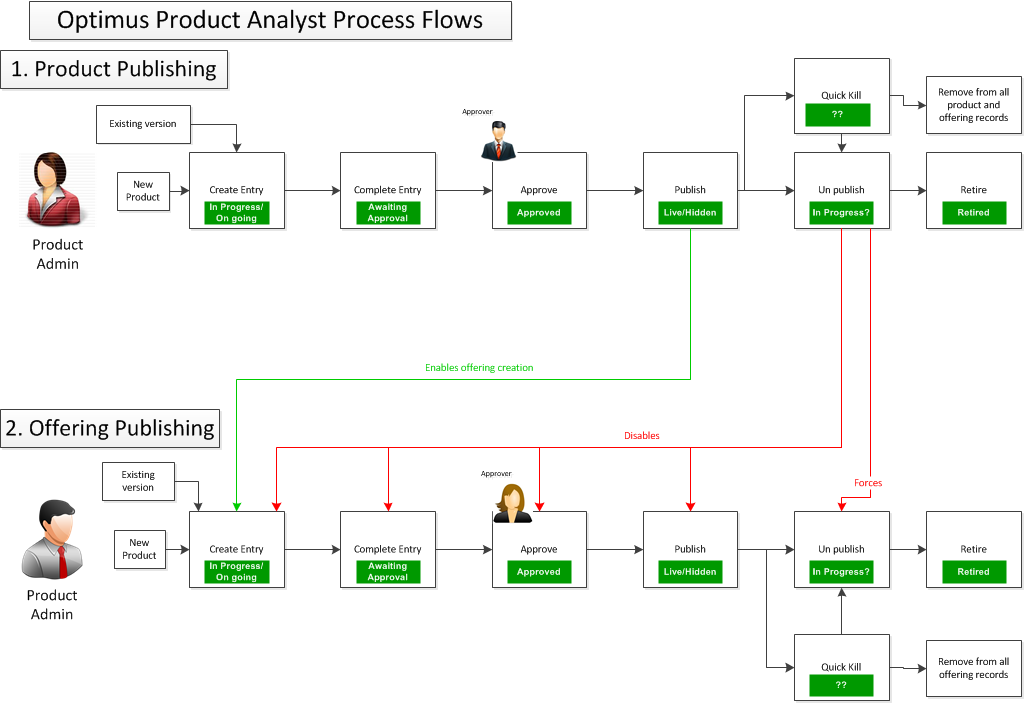
**Mockups**

As per Offering View

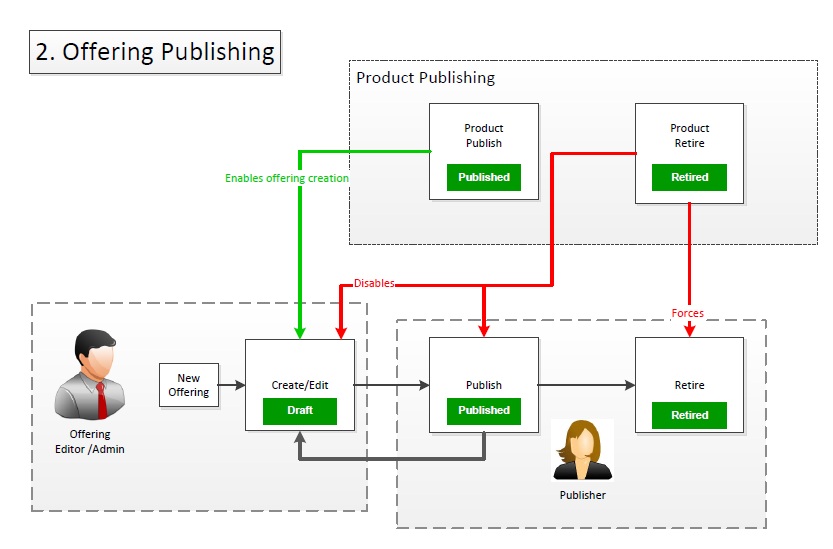
### Offering workflow and offering history

**Overview**

Offerings are dependent on the state of products; only published products can be added to an offering:



In future offering publishing flows will be customizable and it will be controlled by BPM engine. However for now the flow will be controlled manually and will be limited to the below:



Each of the transition between states will result in different set of operations (see below for details). Some of the transitions will save/update a version of the offering.

**Business Rules**

Above diagrams reflect the flow that the offering can go through. Offerings can be in one of the following status:

* PUBLISHED
* RETIRED
* DRAFT (only for a version)
* ABORTED (only for a version)

A newly created offering, until published, is a “version” and status of that version is set to DRAFT. After editing is completed the Publisher can move the offering to PUBLISHED status.

If editing of an offering is needed the Offering Editor can move the offering back to DRAFT status, which creates another “version” of the offering. Offering remains PUBLISHED as it is already used (potentially) by the implemented site. Once the DRAFT version is ready it can be PUBLISHED and it will override the existing PUBLISHED version. Each version will be retained for auditing/reference.

**Suggested approach**

Each of the status transformation should be a separate method in OfferingManager:

* create()
  + An dummy (empty) offering record should be created
  + Offering version (index = 0) created in DRAFT mode referencing the dummy offering
  + The only mandatory parameters are name and type, so the template for the properties can be displayed (publicId will be auto generated).
* publish()
  + validate the DRAFT version
  + overwrite the PUBLISHED data with the draft
  + update the version data
  + update the cache (to be discussed later)
  + update *draftVersionId* column to *NULL*
* enableEdit()
  + this should create a new DRAFT version (index+1) and copy all the data from PUBLISHED offering.
  + update *draftVersionId* column to point to new version
* abortEdit()
  + change the DRAFT status of version to ABORTED
  + update *draftVersionId* column to *NULL*
* retire()
  + change the PUBLISHED status to RETIRE
  + retire all media sets related to the offering that are not assigned to any other offering or offering
  + update the cache (to be discussed later)

Those operations need to be visible as buttons against a status field (see CHANGE button on mockups for Offering View) on the main offering tab. So for the particular statuses:

* DRAFT – There should be a button with Publish, Rollback (available only if “a” PUBLISHED version exist)
* ABORTED – this is to indicate DRAFT version that was rolled back. No buttons available and whole is read only.
* PUBLISHED – There should be a button with Edit label (That calls the enableEdit method) and Retire button (that call Retire())

Please bear in mind that this process in future will be extended heavily.

**Database operations**

Depending of the status/operation updating can happen to either “offering” tables:

Offering.offering, offering.price, offering.propertyValue

or “version” tables:

offering.version, offering.versionPropertyValue

Structure of the tables is very similar. All operations on “DRAFT” versions are done against “version” tables, “offering” tables are used to store published version.

Below table illustrates a typical create-publish-edit-publish flow:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Step 1** | **Create Offering** | |  |  |  |
|  |  |  |  |  |  |
| **offering.offering** | |  | **offering.version** |  |  |
| *offeringName* | *statusId* |  | *offeringName* | *statusId* | *versionId* |
| (no data) | |  | Phone | DRAFT | 1 |
|  |  |  |  |  |  |
| **Step 2** | **Publish Offering** | |  |  |  |
|  |  |  |  |  |  |
| **offering.offering** | |  | **offering.version** |  |  |
| *offeringName* | *statusId* |  | *offeringName* | *statusId* | *versionId* |
| Phone | PUBLISHED |  | Phone | PUBLISHED | 1 |
|  |  |  |  |  |  |
| **Step 3** | **Edit offering** | |  |  |  |
|  |  |  |  |  |  |
| **offering.offering** | |  | **offering.version** |  |  |
| *offeringName* | *statusId* |  | *offeringName* | *statusId* | *versionId* |
| Phone | PUBLISHED |  | Phone | PUBLISHED | 1 |
|  |  |  | Mobile phone | DRAFT | 2 |
|  |  |  |  |  |  |
| **Step 4** | **Re-publish offering** | |  |  |  |
|  |  |  |  |  |  |
| **offering.offering** | |  | **offering.version** |  |  |
| *offeringName* | *statusId* |  | *offeringName* | *statusId* | *versionId* |
| Mobile phone | PUBLISHED |  | Phone | PUBLISHED | 1 |
|  |  |  | Mobile phone | PUBLISHED | 2 |

When the offering is created (Step 1) the data will be stored in “version” tables. In Step 2 which is Publish operation status is updated in the “version” table and data is created/moved to offering database. When offering is then edited “DRAFT” (Step 3) entry needs to be created in “version” tables (no changes to “offering” tables). Once the offering is published again (Step 4) data should be copied from “version” to “offering” tables as well as status updated to “PUBLISHED” (similar to Step 2).

Also offering listing page needs to view “version” data from the offering.versionView database view.

**Mockups**

Next to the Offering name in header there should be information of the currently visible version (i.e. Published / Draft / Version: xx). If the Offering is PUBLISHED and Draft version is available a button [Show Draft version] should be visible next to the header and it should load the DRAFT version into the page. If a version other than the currently PUBLISHED version is currently loaded there should be a link [Show Published version].

Particular version of the offering should be displayed in History tab in a table as per mockup. After clicking a [show] icon relevant version should be loaded into all tabs.

**[ADD MOCKUP HERE]**

### Price List assignments

To be available for sale an offering must have at least one price. Each price is assigned to a pricelist.

## Offering Variants

### View variants

### Create a variant / clone offerings

## Pricing / Pricelists

### Price lists listing

### Price list editing

### Bulk updates for price lists

## Offering Usability

### Price lists quick update

## Offering Translations

## Reference data management

## Workspace

## Caching

### General caching mechanism

### Exposing products details

### Exposing offerings (with pricelists)

### Extending exposed data

## CDN

## Offering API

## Product API

## Vouchers