Data Integrity Report - Group 5

— Preface —

Each of the models (Product, Item, StorageUnit, etc...) have their own singleton `Vault` class which makes sure they are valid with respect to other models of their kind, and manages relationships between models.

For example, a new StorageUnit will check to see if it's unique by calling the `validateNew` or `validateModify` methods on its pointer to the StorageUnitVault instance. The vault will check to see if the name is unique, along with other constraints, and will return a Result object indicating the success of the verification. If a model does not meet the data constraints, it cannot be successfully validated.

Each instance of a model contains a `_valid` member variable which is by default false, and is set to true only when the model has been successfully validated. Models can only be saved if they are first validated.

In our data organization scheme, each model instance is assigned an ID and added to a map in its associated vault. Rather than a parent node holding pointers to its children as is suggested in the functional spec, each model holds the ids of its parents. If I were desire access to a product beneath a specific StorageUnit, I'd call `find("RootParentId = foo")` which would return to me a copy of the found product.

— Integrity Constraints from the Data Dictionary —

Product Container

The data constraints for ProductContainer are enforced by the children of this superclass.

Storage Unit

CONSTRAINT	IMPLEMENTED	TESTED
Name must be non empty	StorageUnit.validate	StorageUnitTest.testValidate
Name must be unique	StorageUnit.validate	StorageUnitTest.testValidate

Product

CONSTRAINT	IMPLEMENTED	TESTED
creationDate must equal the earliest entry date for any item of the product.	TODO WITH CONTROLLER	TODO WITH CONTROLLER
Barcode must be non-empty	Product.validate	ProductTest.testValidate
Description must be non-empty	Product.validate	ProductTest.testValidate
Amount must be non-zero	Product.setSize	ProductTest.testSetSize
Amount must be limited to 1 when the unit is "count"	Size.Validate	ProductTest.testValidate
Shelf life must be non-negative	Product.setShelfLife	ProductTest.testShelfLife
3-Month supply must be non-negative	Product.set3MonthSupply	ProductTest.testSet3MonthSupply
ParentId and RootParentId must be non-empty	TODO in controller	TODO in controller

Item

CONSTRAINT	IMPLEMENTED	TESTED
creationDate must equal the earliest entry date for any item of the product.	TODO WITH CONTROLLER	TODO WITH CONTROLLER

Product Group

CONSTRAINT	IMPLEMENTED	TESTED
3 Month supply cannot be negative. If the unit is count, the amount must be 1.	Size.validate	ProductGroupTest.testSet3MonthSupply

— Additional Integrity Constraints from the Functional Spec —

Adding Items

CONSTRAINT	IMPLEMENTED	TESTED
When a new Item is added to the system, it is placed in a particular Storage Unit (called the "target Storage Unit"). The new Item is added to the same Product Container that contains the Item's Product within the target Storage Unit. If the Item's Product is not already in a Product Container within the target Storage Unit, the Product is placed in the Storage Unit at the root level.		
New Items are added to the Product Container within the target Storage Unit that contains the Item's Product. If the Item's Product is not already in the Storage Unit, it is automatically added to the Storage Unit at the top level before the Items are added.		

Moving / Transferring Items

CONSTRAINT	IMPLEMENTED	TESTED
An Item is contained in exactly one Product Container at a time (until it is removed, at which point it belongs to no Product Container at all).		
When an Item is dragged into a Product Container, the logic is as follows: Target Product Container = the Product Container the user dropped the Item on Target Storage Unit = the Storage Unit containing the Target Product Container If the Item's Product is already in a Product Container in the Target Storage Unit Move the Product and all associated Items from their old Product Container to the Target Product Container Else Add the Product to the Target Product Container Move the selected Item from its old Product Container to the Target Product Container		
When an Item is transferred into a Storage Unit, it is added to the Product Container within the target Storage Unit that contains the Item's Product. If the Item's Product is not already in the Storage Unit, it is automatically added to the Storage Unit at the top level before the Item is transferred.		

Removing Items

CONSTRAINT	IMPLEMENTED	TESTED
When an Item is removed, **1.** The Item is removed from its containing Storage Unit. **2.** The Exit Time is stored in the Item. **3.** The Item is retained for historical purposes (i.e., for calculating statistics and reporting).		

Putting Products in a Product Container

CONSTRAINT	IMPLEMENTED	TESTED
A Product may be in any number of Storage Units. However, a Product may not be in multiple different Product Containers within the same Storage Unit at the same time. That is, a Product may appear at most once in a particular Storage Unit.		
When a Product is dragged into a Product Container, the logic is as follows: Target Product Container = the Product Container the user dropped the Product on Target Storage Unit = the Storage Unit containing the Target Product Container If the Product is already contained in a Product Container in the Target Storage Unit Move the Product and all associated Items from their old Product Container to the Target Product Container Else add the Product to the target Product Container.		

Deleting Products

CONSTRAINT	IMPLEMENTED	TESTED
A Product may be deleted from a Product Container only if there are no Items of the Product remaining in the Product Container.		
A Product may be deleted from the system only if there are no Items of the Product remaining in the system.		