Case study

A simple microservice written in spring boot that provides RESTFul APIs to manage a simple product catalog.

# Design

## Entities and their relationship

We have the following entities

### Category

Represents a product category or a discount category. A category can be a super category (root) or a subcategory of a category.

A category belongs to one parent but can have sub categories in a hierarchical structure. For super categories parent is null.

### Item

Represents a SKU (item). Items are leaf nodes in the category hierarchy. A category can have zero or more items. But an Item can not have any child nodes.

An item can have multiple categories (parent nodes).

### Discount

Discounts can be applied to a category or an item. Currently only percentage discount types are supported.

### Catalog

Catalog is a collection of super categories present in the system. It is automatically derived when we add one or more super categories in the system.

The above entities can be represented in the form of a Tree structure, super-categories as root nodes, categories as child nodes at various levels and Items as leaves. Items can have multiple parent nodes. We can represent them in the form of a `composite design pattern` (with an enhancement to support multiple parents to leaf nodes)

<<interface>>

CatalogComponent

Item

Catagory

## Price calculation

The main logic to calculate total percentage of an item is nothing but knowing all possible paths of the item node to the root node and traversing the paths to sum up the discounts at various nodes. To do that we store “path” of every node in hierarchical format similar to file system paths. (*/rootnode-code/subnode1-code/subnode2-code/…/item1-code).* Since item nodes can have multiple parent nodes, they contain a collection of paths. The calculation of total discount is a chain of responsibility. Every node in the path checks if it has a discount applied, adds it and traverses to next node in the path until we reach the end of the path.

Another thing we need to make sure while traversing the nodes is not to add same discount twice. It is possible that we may visit the same node visited before based on the hierarchy and we should handle duplicates. We used **java.util.Set** to store discounts at various levels that has inbuilt feature to avoid duplicates .

## Database

In this simple catalog service, we used an open source in memory database called **Nitrite** (<https://www.dizitart.org/nitrite-database/>) to store sample data. It is simple to use key-value/document and object store, doesn’t need any server setup and suits well for quick prototypes. However in real world scenarios we can easily extend the design to use robust databases like MongoDB / cassandra etc.

## Microservices:

The simple catalog service is developed on Spring boot framework. It has RESTFul APIs to manage the catalog. We can test the service in “testMode” that loads seed data for quick testing of APIs without the need to add categories.

### **REST APIs supported**

The webservice runs on port 8083 (configurable) http://localhost:8083/catalogws is the root of webservice.

#### **List catalog**

Lists all catalog items (i.e. scatagories that are marked as supercategory appear here automatically) *API Path*: GET /catalogws/catalog

*Sample output*:

[{"code":"cat-baby","label":"Baby","description":"Baby products"}]

#### **Add category**

*API Path*: POST /catalogws/category *Sample Request Body*:

{

"type": "category",

"label": "Sports",

"description": "Sports products",

"superCategory": true

}

*Sample output*:

{

"type": "category",

"id": "5e4200fa-de59-46ca-85fe-7f3b51eeef1f",

"code": "cat-sports",

"label": "Sports",

"description": "Sports products",

"discount": null,

"children": null,

"superCategory": true,

"path": "/cat-sports"

}

#### **Update category**

*API Path*: PUT /catalogws/category/<id> Updates an existing category (To be added)

#### **Delete category**

Deletes a category (To be added) *API Path*: DELETE /catalogws/category/<id> Deletes an existing category and all its children (To be added)

#### **List category**

Lists all categories *API Path*: GET /catalogws/category

#### **Search category**

Supports text search as well as filtering by category code

#### **Search by query**

*API Path*: GET /catalogws/category?q=Boy

#### **Filter by category code**

*API Path*: GET /catalogws/category?code=cat-boys

#### **Get category by Id**

*API Path*: GET /catalogws/category/<id>

#### **Add item**

*API Path*: POST /catalogws/item *Sample Request Body*:

{

"type": "item",

"label": "Baby Soap",

"description": "Baby soap product"

}

#### **Update item**

*API Path*: PUT /catalogws/item/<id> Updates an existing item (To be added)

#### **Delete item**

Deletes a item (To be added) *API Path*: DELETE /catalogws/item/<id> Deletes an existing item

#### **List item**

Lists all items *API Path*: GET /catalogws/item

#### **Search item**

Supports text search as well as filtering by item code

#### **Search by query**

*API Path*: GET /catalogws/item?q=car

#### **Filter by item code**

*API Path*: GET /catalogws/item?code=sku-battery-toy-car

#### **Get item by Id**

*API Path*: GET /catalogws/item/<id>

## Installing and running

Clone simple-catalog-ws and goto the root folder Build using ./mvnw clean package Run using java -jar target/simple-catalog-ws-0.0.1-SNAPSHOT.jar

If you want to load sample data while the service starts you can pass loadTestDb arguement as follows java -jar target/simple-catalog-ws-0.0.1-SNAPSHOT.jar testMode