Ben-Jerry

Building an API for Ben & Jerry's fans

Objects handled: JSON array of below structure

```
{
  "name": "Vanilla Toffee Bar Crunch",
  "image_closed": "/files/live/sites/systemsite/files/flavors/products/us/pint/open-closed-pints/vanilla-toffee-landing.png",
  "image_open": "/files/live/sites/systemsite/files/flavors/products/us/pint/open-closed-pints/vanilla-toffee-landing-open.png",
  "description": "Vanilla Ice Cream with Fudge-Covered Toffee Pieces",
  "story": "Vanilla What Bar Crunch? We gave this flavor a new name to go with the new toffee bars we’re using as part of our commit
  "sourcing_values": [
    "Non-GMO",
    "Cage-Free Eggs",
    "Fairtrade",
    "Responsibly Sourced Packaging",
    "Caring Dairy"
  ],
  "ingredients": [
    "cream",
    "skim milk",
    "liquid sugar",
    "water",
    "sugar",
    "coconut oil",
    "egg yolks",
    "butter",
    "vanilla extract",
    "almonds",
    "cocoa (processed with alkali)",
    "milk",
    "soy lecithin",
    "cocoa",
    "natural flavor",
    "salt",
    "vegetable oil
    "guar gum",
    "carrageenan"
  ],
  "allergy_info": "may contain wheat, peanuts and other tree nuts",
  "dietary_certifications": "Kosher",
  "productId": "646"
}
```

Tasks Implemented

- Implemented a REST API with CRUD functionality with PosgressSQL as DB
- Authenicated using JWT token
- Documentation done using Swagger (// Swagger json is not created properly at the time of writing)
- Extensive Testing is done using Post Main, Go Test and manual testing

Architecural Decision

Table Definition is given below:

Product Table will hold the string fields Ingredients Table will hold (Product ID, Ingredient ID) SourcingValue Table will hold (Product ID, Sourcing Value ID) IngredientIndex Table will (Ingredient ID, Value) SourcingValue Table will (Sourcing Value ID, Value)

Design Choices Considered

Efficient Storage and Retrieval of values based on the Values (Ingredients , Sourcing Values) by keeping in Separate Tables

PreProcess of Values in Ingredients :

"butter (cream","salt)" - Instead of saving 2 values , mapped to Single value in table "butter (cream salt)

"liquid sugar (sugar","water)" - mapped to "liquid sugar (sugar water)

Database Used: Posgress SQL (Used PostgreSQL as a Service , Elephant SQL) - DB instance is up and running - Testing is possible by running application locally

How Ingredients / Sourcing Values are updated since it is kept in two tables

Old Values : Array of Ingredients present in DB New Values : Array of Ingredients given to the endpoint

Compare Old Values and New Values :

Make a Decision on Which Values to be Deleted Make a Decision on Which Values to be Inserted Make a Decision on Which Values not touched

Example :

Old Values : "ingredients": [*"cream", "skim milk", "liquid sugar", "water",* **"sugar", "coconut oil"]**

New Values : "ingredients": [*"cream", "skim milk", "liquid sugar", "water",* **"black sugar", "coco powder"**]

Rows Untouched during Update : ["cream" , "skim milk" , "liquid sugar", "water"]

Rows to be Deleted: ["sugar", "coconut oil"]

Rows to be Inserted: ["black sugar", "coco powder"]

If the indexes are available for black sugar and coco powder in the ingredients index table , update only ingredients table

If the indexs are not available for black sugar and coco powder in the ingredients index table , create entries for black sugar and coco powder in the ingredients table and then update the ingredients table

---

Table Definition

Users	Products
id	id (PK)
email	name
password	image_open
	| image_close
	| description
	| story
	| allergy_info
	| dietary_certifications

ingredientsindex	sourcingvalueindex
id (PK)	id (PK)
value (text)	value (text)

ingredients	sourcing_values
id	id
product_id	product_id
value_id	value_id
PRIMARY KEY (product_id, value_id)	PRIMARY KEY (product_id, value_id)
FOREIGN KEY (product_id) REFERENCES products (id)	FOREIGN KEY (product_id) REFERENCES products (id)
FOREIGN KEY (value_id) REFERENCES ingredientsindex (id)	FOREIGN KEY (value_id) REFERENCES sourcingvalueindex (id)

API Definition

List of APIs supported

GET ALL Products

[GET] /products

Response : JSON Array

While testing it took 1.5 min to get all the values

GET Product by id

[GET] /products/{:id}

Request Parameter : id

Response : JSON element with same structure as example

CREATE Product by id

[POST] /products/{:id}

Request Parameter: id

Request Body : JSON (with same structure as example) # productID field not supported # new ID generated

Response : JSON { "productID" : value } value - ID of newly created element

UPDATE Product by id

[PUT] /products/{:id}

Request Parameter: id

Request Body : JSON (with same structure as example)

Response: 0 - No change , 1 - Element updated

DELETE Product by id

[DELETE] /products/{:id}

Request Parameter: id

Response: 0 - No change , 1 - Element updated

Additional APIs provided for Each Values :

Read Product Name for Product ID

[GET] /products/{:id}/name

Update Product Name for Product ID

[PUT] /products/{:id"}/name

Request Body: { "name" : value }

Read Product Image_open for Product ID

[GET] /products/{:id}/image_open

Update Product Image_open for Product ID

[PUT] /products/{:id"}/image_open

Request Body: { "image_open" : value }

Read Product Image_closed for Product ID

[GET] /products/{:id}/image_closed

Update Product Image_open for Product ID

[PUT] /products/{:id"}/image_closed

Request Body: { "image_closed" : value }

Read Product Description for Product ID

[GET] /products/{:id}/description

Update Product Description for Product ID

[PUT] /products/{:id"}/description

Request Body: { "description" : value }

Read Product Story for Product ID

[GET] /products/{:id}/story

Update Product Story for Product ID

[PUT] /products/{:id"}/story

Request Body: { "story" : value }

Read Product Dietary Certification for Product ID

[GET] /products/{:id}/diet

Update Product Dietary Certification for Product ID

[PUT] /products/{:id"}/diet

Request Body: { "dietary_certifications" : value }

Read Product Allergy Info for Product ID

[GET] /products/{:id}/allergy

Update Product Allergy Info for Product ID

[PUT] /products/{:id"}/allergy

Request Body: { "allergy_info" : value }

Read Product Ingredients for Product ID

[GET] /products/{:id}/ingredients Response : { "ingredients" : Array of Strings }

Update Product Ingredients for Product ID

[PUT] /products/{:id"}/ingredients

Request Body: { "ingredients" : Array of Strings }

Read Product Sourcing Value for Product ID

[GET] /products/{:id}/sourcingvalue Response : { "sourcing_value" : Array of Strings }

Update Product Ingredients for Product ID

[PUT] /products/{:id"}/ingredients

Request Body: { "sourcing_value" : Array of Strings }

Testing

Using PostMan -- Added postman collection to the zip Using Go Test Manual Testing