Table of Contents

Reedsy's (fictional) Merchandising Store - Bilal Aslam

1. Description

A store consists of multiple products. Using the API, you can list all the existing products and their attributes. Also, you can update the price for the existing products.

Users can send a list of products (codes and quantities) and can determine the total price for the list of products. Price of product will be discounted depending on the quantity that the user is purchasing.

Note: I have built a complete generic solution that will work for any number of products and any variable discounts that you wish to apply on the product quantities. [More detail in upcoming sections.]

2. Running the Project

1. Extract the zip file containing the code.
2. The project requires **Rails (6.1.4.1)** and **Ruby (2.7.0)**
3. After extracting the project run the below commands in order:
    a. **bundle install**
    b. **rails db:create**
    c. **rails db:migrate**
    d. **rails db:seed**
    e. **rails server**
4. The project will start running locally. You can start testing each API using the cURL commands.
5. Run the **rspec** command to test all cases.

Note: The seed file will create the products and discounts according to the scenario described in the requirements.
https://github.com/reedsy/challenges/blob/main/ruby-on-rails-engineer-v2.md

3. Design Decisions and Constraints

Model Classes

- **Product** (id, code, name, price): Each product has a unique code. The code column of Product has also been indexed.
- **Discount** (id, *product_id*, quantity_range, percentage): quantity_range is an **int8range** field that is used to determine the discount percentage. For example quantity between (11-20 for a TSHIRT) gives 2% discount. Also, a rule is defined such that a product can only have one discount percentage for a one quantity range.

Note: You can add more products and define more discount rules in the DB or edit the existing ones. The solution is generic and is not dependent on existing or specified rules.

Business Rules

- A product can have many discounts with different quantity_range set.
- A discount belongs to a product.

API Design

I have made the restful API for this assessment in rails using my API boilerplate that I made long ago.

- **API Main Components:**
    - ***Respondable*:** It handles all the Restful CRUD operations. The code is generic. All you need is to make any resource controller and define its parameters.
    - ***Error Handler*:** The error handler will cover all types of errors during Restful API calls.
    - **API Response:** I have built single template of a serialized hash in Product model for list of responses. Due to limited use cases in assignment, I did not use any JSON API response builder gem. However, I usually use https://github.com/jsonapi-serializer/jsonapi-serializer

Note: Using my Rails API boilerplate, all the CRUD operations of any resource can be implemented within seconds. However, I have only generated routes for the APIs required in the technical assessment. You can add more routes for remaining CRUD operations in **api_v1.rb** file. Adding the routes is enough to run the CRUD for Restful resources.

4. Testing Requirements

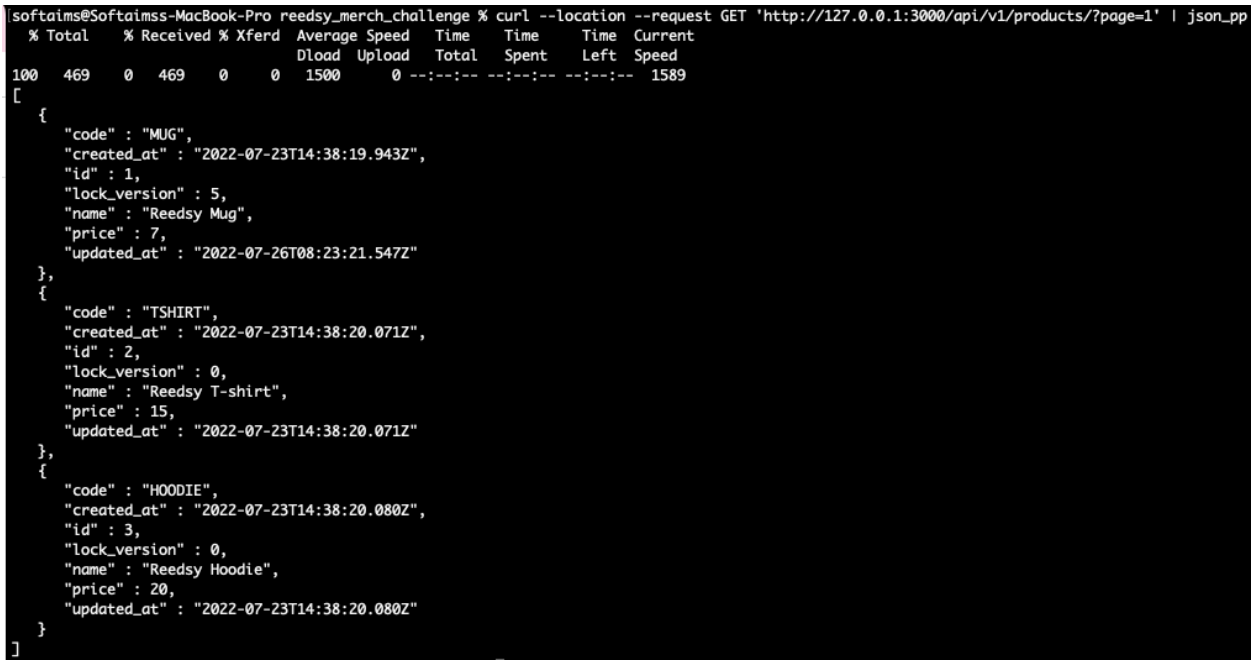
Requirement 01 - API that lists existing items in the store with attributes

It uses the **ProductsController#index** method and I am also using pagination to improve API performance.

- **cURL command:**

  curl --location --request GET 'http://127.0.0.1:3000/api/v1/products/?page=1'

- **Response:**



Requirement 02 - API that updates the price of the product

It uses the **ProductsController#update** method. The edge case is handled using optimistic locking on the products table. [More details in upcoming sections]

- **cURL command:**

  **curl --location --request PUT 'http://127.0.0.1:3000/api/v1/products/1' **
  **--header 'Content-Type: application/json' **
  --data-raw '{
  "product":{
  "price": 7
  }

}'

- **Response:**

```
softaims@Softaimss-MacBook-Pro reedsy_merch_challenge % curl --location --request PUT 'http://127.0.0.1:3000/api/v1/products/1' \
--header 'Content-Type: application/json' \
--data-raw '{
        "product":{
            "price": 7
        }
}' | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   185    0   150  100    35   1931    450 --:--:-- --:--:-- --:--:--  3083
{
   "code" : "MUG",
   "created_at" : "2022-07-23T14:38:19.943Z",
   "id" : 1,
   "lock_version" : 5,
   "name" : "Reedsy Mug",
   "price" : 7,
   "updated_at" : "2022-07-26T08:23:21.547Z"
}
```

Requirement 03 & 04 - API that checks the price of a given list of items and provide scenario based discounts

It uses the **ProductsController#list_price** method. The user has to provide the product codes and quantity they require. Discount is applied on the basis of discount business rules that are stored in discounts table in database.

- **cURL command:**

```
curl --location --request GET 'http://127.0.0.1:3000/api/v1/products/price' \
--header 'Content-Type: application/json' \
--data-raw '{
   "products": {
      "items": [
         {
            "code": "MUG",
            "quantity": 10
         },
         {
            "code": "TSHIRT",
            "quantity": 20
         },
         {
            "code": "HOODIE",
            "quantity": 1
         }
      ]
   }
}'
```

- **Response:** [Response is explained below in Algorithms section]

```
softaims@Softaimss-MacBook-Pro reedsy_merch_challenge % curl --location --request GET 'http://127.0.0.1:3000/api/v1/products/price' \
--header 'Content-Type: application/json' \
--data-raw '{
    "products": {
        "items": [
            {
                "code": "MUG",
                "quantity": 10
            },
            {
                "code": "TSHIRT",
                "quantity": 20
            },
            {
                "code": "HOODIE",
                "quantity": 1
            }
        ]
    }
}' | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   694    0   361  100   333  13395  12356 --:--:-- --:--:-- --:--:-- 46266
{
   "items" : [
      {
         "code" : "MUG",
         "discount_percentage" : 2,
         "discounted_price" : 6.86,
         "id" : 1,
         "original_price" : 7,
         "quantity" : 10
      },
      {
         "code" : "TSHIRT",
         "discount_percentage" : 30,
         "discounted_price" : 10.5,
         "id" : 2,
         "original_price" : 15,
         "quantity" : 20
      },
      {
         "code" : "HOODIE",
         "discount_percentage" : null,
         "discounted_price" : null,
         "id" : 3,
         "original_price" : 20,
         "quantity" : 1
      }
   ],
   "total_price" : 298.6
}
```

5. Algorithms

Discount Calculation

- A request is sent to **ProductsController#list_price** method.
- All the products along with their discounts are extracted using the item codes sent by the user.
- A **ReceiptGenerator** service is called which will prepare the receipt i.e., product pricing list along with their discounts.
- For each item code sent by user, we see if the item code is valid and if it is, we check if there are discounts available for the product.

- If a product exists in the database, its **sale_price** method is called from its Model.
- The **sale_price** method checks if there is any discount associated with product, it returns the discounted single unit price or the actual single unit price if there is no discount.
- Once all the receipt is prepared by the **ReceiptGenerator** service, the following response is sent to the user.
    - A hash consisting of **items** array and **total_price**
    - Each item consists of:
        - **id**: product id
        - **code**: product code
        - **quantity:** product quantity request by the user.
        - **original_price:** original price per unit for the product.
        - **discounted_price:** discounted price per unit for the product.
        - **discounted_percentage:** percentage discount that will be applied on original price.
    - **total_price:** consists of the sum of the total price for all the list items after applying discounts if there are any.

6. Test Cases

I have used the **RSpecs** gem to test the API requests and model validations. Below are the test coverage details:

Models Test Coverage

- **Product**
    - Validate presence of code, name, price during product creation.
    - Validate numericality of price while creating or updating the product.
    - Validate uniqueness of code while creating the product.
    - Factory bot object creation test.
    - Product association with discounts test.
    - Product sale price test with different quantity range to test discounted and non-discounted price.

- **Discount**
    - Factory bot object creation test.
    - Discount association with product.
    - Validate presence of quantitiy_range and percentage while creating a discount.

API Requests Test Coverage

- **GET api/v1/products/**
    - Check response for a valid request.
    - Check length of returned product list responses.

- **PUT /api/v1/products/:id**
    - Check for valid price and product id.
    - Check for invalid price.
    - Check for concurrent product price update.
    - Check with missing params.

- **GET /api/v1/products/price**
    - Check for total price calculation with discounted prices.
    - Check for total price calculation with undiscounted prices.

7. Developer Notes

- I have tried to fulfill the requirements in the most minimalist way possible.
- If anything is missing from the requirements or any additional functionality is required, please let me know.
- **Email:** aslam.bilal0011@gmail.com

The End