Junior Backend Developer Task for Monk Commerce 2024

(Coupons Management API for an E-commerce Website)

Objective: Build a RESTful API to manage and apply different types of discount coupons (cart-wise, product-wise, and BxGy) for an e-commerce platform, with the ability to easily add new types of coupons in the future.

Note: There are going to be many cases to be implemented. **DO NOT** try to implement everything, just make sure that the limitations and assumptions are well documented.

Instructions for Candidates (PLEASE READ CAREFULLY):

The most important part of this task is **thinking of as many cases(like coupon constraints, subtypes etc.) as possible**. Clearly list all the use cases you can think of in the **readme** file, including those you have implemented and those that are difficult to implement given the time constraint. **The majority of your score will be based on the quality, variety and relevance of the cases you consider (regardless whether they are implemented or not)**.
Following this, your task will be assessed with the next highest priorities being:

- Correct implementation of the cases you choose to solve.
- Optimising the solution for performance and efficiency.
- Writing clean, readable, and maintainable code.

Write any limitations and assumptions into the **readme** file. ***There is no need to solve for every case***—just ensure the **assumptions and limitations for correct functionality** are clearly mentioned.

Consider the following while working on the task:

- Different scenarios for applying cart-wise, product-wise, and BxGy coupons.
- Edge cases and potential issues with coupon applicability.
- Assumptions made to ensure correct working of the system.
- Any limitations of the current implementation and suggestions for improvement.
- You can design your own schema as the above schema is just for overview.
- Upload the project to a public **git** repository.
- Must add a readme file and explain **cases**, assumptions and anything you want to add.

Requirements:

1. **Coupon Types:**
    - **Cart-wise:** Apply a discount to the entire cart if the total amount exceeds a certain threshold.
    - **Product-wise:** Apply a discount to specific products.
    - **BxGy:** “Buy X, Get Y” deals with a repetition limit and can be applicable to a set of products (e.g., Buy 3 of Product X or Product Y, get 1 of Product P and Product Q free and so on).

2. **API Endpoints:**
    - **POST /coupons:** Create a new coupon.
    - **GET /coupons:** Retrieve all coupons.
    - **GET /coupons/{id}:** Retrieve a specific coupon by its ID.
    - **PUT /coupons/{id}:** Update a specific coupon by its ID.
    - **DELETE /coupons/{id}:** Delete a specific coupon by its ID.
    - **POST /applicable-coupons:** Fetch all applicable coupons for a given cart and calculate the total discount that will be applied by each coupon.
    - **POST /apply-coupon/{id}:** Apply a specific coupon to the cart and return the updated cart with discounted prices for each item.

3. **Coupon Structure:**
    - Each coupon should have an ID, type (cart-wise, product-wise, BxGy), discount details, and conditions for applicability.
    - The implementation should be designed to easily add new types of coupons in the future.

4. **Database:**
    - Use any database you are comfortable with(in-memory, mySQL, mongoDb or anything else).
    - Store coupon details and conditions.

5. **Error Handling:**
    - Implement basic error handling (e.g., coupon not found, invalid input, conditions not met).

6. **Documentation:**
    - **README File:** Document the cases and limitations in the README file. This should include:
    - **Implemented Cases:** Describe the cases you have implemented and how they work.
    - **Unimplemented Cases:** List any cases you thought of but could not implement, along with reasons.
    - **Limitations:** Outline the limitations of your current implementation.
    - **Assumptions:** List any assumptions you made to ensure the correct working of the system.

Bonus:

- Implement unit tests.
- Add expiration dates for coupons.

Examples of Coupon Cases

1. **Cart-wise Coupons:**
    - **10% off on carts over Rs. 100**
        - Condition: Cart total > 100
        - Discount: 10% off the entire cart
2. **Product-wise Coupons:**
    - **20% off on Product A**
        - Condition: Product A is in the cart
        - Discount: 20% off Product A
3. **BxGy Coupons:**
    - Buy a specified number of products from one array and get a specified number of products from another array for free.
        - Example: b2g1 - Buy 2 products from the “buy” array (e.g., [X, Y, Z]) and get 1 product from the “get” array (e.g., [A, B, C]) for free.
    - Scenarios:
        - If the cart has products X, Y, and A, then ‘A’ would be free.
        - If the cart has products X, Z, and C, then ‘C’ would be free.
        - If the cart has products X, A, and B, the b2g1 coupon is not applicable as there are not 2 products from the “buy” array.
    - Repetition Limit: If the repetition limit is 3, the coupon can be applied 3 times.
        - If the cart has 6 products from the “buy” array and 3 products from the “get” array, the coupon can be applied 3 times. I.e. for **b2g1**, buying 6 products from [X, Y, Z] would result in getting 3 products from [A, B, C] for free.
        - If the cart has products [X, X, X, Y, Y, Y] (6 items from the “buy” array) and products [A, B, C] or [A,B], then [A, B, and C] or [A,B] would be free.

Example Payloads and Responses

1. **POST /coupons**

```
{
  "type": "cart-wise",
  "details": {
    "threshold": 100,
    "discount": 10
  }
}
```

```
{
  "type": "product-wise",
  "details": {
    "product_id": 1,
    "discount": 20
  }
}
```

```
{
  "type": "bxgy",
  "details": {
    "buy_products": [
      {"product_id": 1, "quantity": 3},
      {"product_id": 2, "quantity": 3}
    ],
    "get_products": [
      {"product_id": 3, "quantity": 1}
    ],
    "repition_limit": 2
  }
}
```

2. **POST /applicable-coupons**

```
{"cart": {
    "items": [
      {"product_id": 1, "quantity": 6, "price": 50}, // Product X
      {"product_id": 2, "quantity": 3, "price": 30}, // Product Y
      {"product_id": 3, "quantity": 2, "price": 25}  // Product Z
    ]
  }}
```

Response:

```
{
  "applicable_coupons": [
    {
      "coupon_id": 1,
      "type": "cart-wise",
      "discount": 40 // 10% of 400
    },
    {
      "coupon_id": 3,
      "type": "bxgy",
      "discount": 50 // Buy 6 of Product X or Y, Get 2 of Product Z Free
(2x $25)
```

```
    }
  ]
}
```

3. **POST /apply-coupon/{id}**

```
{
  "cart": {
    "items": [
      {"product_id": 1, "quantity": 6, "price": 50}, // Product X
      {"product_id": 2, "quantity": 3, "price": 30}, // Product Y
      {"product_id": 3, "quantity": 2, "price": 25}  // Product Z
    ]
  }
}
```

Response:

```
{
  "updated_cart": {
    "items": [
     {"product_id": 1, "quantity": 6, "price": 50,"total_discount": 0},
// discount on 3 quantity of product x
     {"product_id": 2, "quantity": 3, "price": 30, "total_discount": 0},
     {"product_id": 3, "quantity": 4, "price": 25, "total_discount": 50}
// 2 extra Product Z free
    ],
    "total_price": 490,
    "total_discount": 50, // On the entire cart
    "final_price": 440
  }
}
```

This task will help assess your ability to handle different coupon logics, implement CRUD operations, and ensure the correct application of discounts based on various conditions, including fetching and applying specific coupons to the cart, updating the cart's prices accordingly, and **designing the system for future extensibility.**