Admin Dashboard

To simplify manual testing, I have implemented a basic admin dashboard that allows easy management of the system. The dashboard includes CRUD operations for managing Products, Countries, and Currencies.

Price List Assumptions

By default, I assumed that the base price's currency_code would be in USD. For example, if no active price list is found, the system will return the price as 100 USD instead of just 100.

NOTES

● GET /api/products

For this endpoint, I assumed that the price list would be applicable when the current date falls between the start_date and end_date of the price list.

If no order parameter is provided, or if an invalid one is provided, the system will return the products ordered by creation date, from newest to oldest.

● GET /api/products/{id}

For this endpoint, if the date is provided in the request, I will filter the price list based on the provided date. If the date is not provided, I will automatically filter the price list based on today's date to retrieve the applicable price list.

PriceService Class Overview
The PriceService class is responsible for calculating and returning the applicable price for a given product based on various conditions, such as date, country, and currency. It provides a method getApplicablePrice, which performs the following tasks:

- **Filters Price Lists: It checks each price list associated with the product to determine whether it is valid based on the provided date, country, and currency.**
- **Date Validation: The price list is considered valid if the start_date is before or on the given date, and the end_date is after or on the given date.**
- **Country Validation: If a country code is provided, the price list must match the country or be null (which means it applies to all countries).**

- **Currency Validation: Similar to the country check, the price list must either match the specified currency or be null to apply to all currencies.**
- **Price Selection: Once the price lists are filtered, the system selects the one with the highest priority (lowest priority number), or defaults to the product's base price if no applicable price list is found.**
- **Fallback: If no price list matches the criteria, the method returns the product's base price and defaults the currency code to 'USD'.**

This service ensures that the correct price is returned based on the given conditions.

Use of Laravel Features

● Requests

I utilized Laravel's built-in request handling to manage errors effectively. This ensures proper validation and error management across the application, such as when a code is not found in the database, or an invalid date format is provided.

● Resources

I used Laravel Resources to format the responses, ensuring a consistent data structure that is easy for the frontend to process.

● Traits

To simplify communication with the frontend and ensure the responses are standardized, I used traits to consolidate response handling logic.

Performance Optimization

To optimize performance, I added indexes on the price_lists table to enhance query speed. These indexes improve database query performance. The following indexes were created for optimization:

1. **Index on product_id, country_code, and currency_code to speed up lookups when filtering by these fields.**
2. **Index on start_date and end_date to optimize queries that check if a price list is active for a given date range.**

Testing Coverage

I have implemented test cases to cover all possible scenarios, ensuring the system behaves correctly under different conditions. These are **feature tests**, validating the integration between models and API endpoints.

You can find the test file at: tests/Feature/PriceListTest.php.