Code Documentation

Settings ENV

Important variables to include in .env file. It is already included in provided env

OPEN_EXCHANGE_RATES_APP_ID=TEST

This environment variable holds the API key used to authenticate requests to the Open Exchange Rates service. In production, replace the test with the actual API key to enable live exchange rate fetching.

APP_BASE_CURRENCY=USD

Sets the base currency for all exchange rate calculations within the application. All fetched or stored exchange rates are relative to this currency. In this case, USD is used as the default base for consistent financial operations.

Key strengths provided in this project

The following strategies were applied throughout the project to ensure quality, maintainability, and performance

1. **Architectural Design**
2. **Validation & Data Reliability**
3. **Performance Optimization**
4. **Automation via Artisan Commands**
5. **Testing Strategy**
6. **Others**

1. Architectural Design

- **API Versioning**

  Implements versioning e.g **/api/v1/invoices** *to ensure backward compatibility and flexibility as the API evolves over time. This allows new versions to be introduced without breaking existing client integrations.*
- **Repository Pattern**

  Abstracts database queries into repository classes, promoting separation of concerns and making the codebase more maintainable, testable, and flexible.
- **Service Pattern**

  Encapsulates business logic in dedicated service classes, keeping controllers lightweight and focused on handling HTTP requests and responses.

- **Helpers**

  Contains globally accessible functions for reusable logic that doesn't belong to any specific class, helping reduce code duplication and improve consistency.

- **Enum Helpers**

  Provides utility methods for working with PHP Enums, such as retrieving labels, values, or formatting enum-based responses for APIs and forms.

- **Accessors and Mutators**

  Uses Eloquent model accessors and mutators to format attributes when retrieving or storing them in the database, improving data consistency and readability.

- **API Resource**

  Transforms models into structured and standardized JSON responses using Laravel's JsonResource, making the API output cleaner and more maintainable.

- **API Throttling**

  Implements request rate limiting to protect the API from excessive traffic, and potential denial-of-service attacks. By using Laravel's throttle middleware and custom rate limiters, the system regulates the number of requests allowed within a given timeframe.

2. Validation & Data Reliability

- **Used Form Request Classes for Validation**

  Utilized Laravel's custom Form Request classes to encapsulate validation logic, keeping controllers clean and ensuring that input data is validated consistently and robustly before reaching the business logic layer.

- **Currency Validation with API and Caching**

  Fetched all active currencies from an external API and stored them in the database. Implemented caching to reduce API calls and improve performance by validating currency codes against the cached data rather than making repeated requests.

- **Prevent Modification of Exchange Rate with Existing Invoices**

  Added validation logic to disallow changes to exchange rates if they are already associated with issued invoices. This ensures the integrity and consistency of financial records and prevents retroactive data changes.

- **Validation for Valid Currency Inputs**

  Ensured that user-provided currency codes or symbols are checked against the list of supported and active currencies. This prevents invalid or unsupported currency values from being processed in transactions.

- **Validation for Missing Exchange Rate**

  Ensured that user provides exchange rate is existing otherwise will throw an error.

- **GraphQL Integration Using Rebing Package**

  Implemented a GraphQL architecture in Laravel using the Rebing package, defining custom types, queries, and mutations to data interaction and improve API flexibility.

3. Performance Optimization

- **Caching the currencies Table (TTL: 1 Day)**

  The list of active currencies from the currencies table is cached to reduce frequent database queries and

external API calls. This significantly improves performance, especially for validations and conversions that rely on currency data. The cache is set to expire every 24 hours (Time To Live: 1 day), ensuring the data remains reasonably fresh while minimizing unnecessary overhead.

- **Seeders**

  The CurrencySeeder class automates the process of populating the currencies table by retrieving a complete list of global currencies from the Open Exchange Rates API. This eliminates the need to manually enter each currency, ensuring that all currency codes and names are accurately inserted or updated in the database.

4. Automation via Artisan Commands

- **Fetch Exchange Rates**

  This custom Artisan command retrieves the latest exchange rates from a free third-party API, using USD as the base currency. The fetched rates are then processed and stored for further use within the application. This helps keep currency conversions up-to-date for accurate financial calculations.

  Command:

  ```
  php artisan exchange:fetch
  ```

- **Sync Active Currencies**

  This command synchronizes the list of active currencies by fetching them from an external source (such as an API), then saving or updating them in the local database. It ensures that the system maintains an accurate and current list of supported currencies for validation and transactional purposes.

  Command:

  ```
  php artisan currencies:sync
  ```

5. Testing Strategy

- **Feature Testing**

  Feature tests are used to test the full flow of a feature from the API endpoint down to the database. These tests ensure that your application behaves as expected from the user's perspective.

  - test_create_invoice Feature Test

    Validates the full invoice creation process through the API. It ensures that all required fields are handled correctly and that the invoice is saved in the database.

    ***Command**:*

    ```
    php artisan test –filter=InvoiceUnitTest
    ```

  - test_create_company Feature Test

    Tests the company creation endpoint to confirm that the API can successfully store new company data with proper validations.

    ***Command**:*

    ```
    php artisan test –filter=CompanyApiTest
    ```

- test_create_exchange_rate Feature Test

  Verifies the API functionality for creating new exchange rates and ensures that validations and storage logic are working correctly.

  Command:

  ```
  php artisan test –filter=ExchangeRateApiTest
  ```

❖ **Unit Testing**

- test_get_latest_rate Unit Test

  Validates that the method correctly retrieves the most recent exchange rate.

- test_convert_to_base_currency Unit Test

  Ensures that conversion logic to the base currency is accurate.

  Command:

  ```
  php artisan test –filter=InvoiceUnitTest
  ```

6. Others

❖ **Search for companies and invoices**

❖ **Include and sort parameters for companies and invoices**

❖ **Pagination**

❖ **GraphQL Types, Mutation and Query approach**

❖ **Laravel Pint Formatter**