Home Assignment

> *Please add the necessary documentation related to code in README.md file*

The service should be developed using Java for the server-side implementation and either PostgreSQL or MySQL as the persistent database.

What needs to be implemented:

1. **Internal Interface (with data team):**
    - Implement two separate API endpoints to receive and store:
        - Shopper personalized product lists (JSON object containing `shopperId` and an array of `productId` and `relevancyScore` for each product)
        - Product metadata (JSON object containing `productId`, `category`, and `brand` for each product)
    - Design and implement a well-structured and normalized database schema to store the shopper personalized product lists and product metadata efficiently.
    - Develop logic to parse the received JSON data and persist it in the database, utilizing lambda expressions and streams for concise and readable code.
    - **Use Spring Validation for input validation and data integrity.**
    - Implement proper error handling and logical validations for robust and reliable data ingestion.
    - Use Lombok annotations (e.g., `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`) to reduce boilerplate code in DTOs and entity classes.
2. **External Interface (with eCommerce):**
    - Implement a high-performance, real-time GET API endpoint to retrieve a shopper's personalized product list.
    - Support filtering by `category`, `brand`, `productId`, and limiting the number of results using the `limit` query parameter.
    - Consider supporting additional filtering options based on product attributes (e.g., price range, rating) to enhance functionality and usability.
    - Implement additional options like sorting by `relevancyScore`, `category`, `brand`, or other product attributes, ordering (`asc` or `desc`), and pagination (`page` and `size`).
    - Ensure correct JPA object mapping and utilize Spring Data Specifications, JPA criteria queries, or JPA native queries for flexible and powerful query building, leveraging lambda expressions for concise query definitions.
    - Implement caching mechanisms (e.g., Redis, Memcached) or other performance optimization techniques to meet real-time performance requirements.
    - Follow industry-standard URL conventions and best practices for designing RESTful APIs.
3. **Database Design:**

    - Implement a well-structured and normalized database schema with separate tables for shoppers, products, and shopper-product associations (shelves or personalized product lists).
    - Enforce uniqueness constraints and foreign key constraints at the database level to ensure referential integrity.
    - Consider indexing columns used in filtering and sorting queries for efficient querying.
4. **Best Practices:**
    - Keep the code clean and have a organized project structure that adheres to industry best practices and coding conventions (e.g., separate packages for controllers, services, repositories, DTOs).
    - Follow consistent naming conventions throughout the codebase for improved readability and maintainability.
    - Use good naming convetions for the endpoints
    - Separate concerns by having dedicated classes or components for input validation, error handling, and data mapping.
    - Utilize features like streams and lambda expressions for more concise and readable code throughout the codebase, including data processing, mapping, and querying.
    - Use Lombok annotations to reduce boilerplate code in DTOs and entity classes.
    - Implement proper logging using a logging framework like SLF4J or Log4j.

Dos:

- Design a well-structured and normalized database schema.
- Enforce uniqueness constraints and foreign key constraints at the database level.
- Implement separate API endpoints for receiving shopper personalized product lists and product metadata.
- Utilize Spring Validation for input validation and data integrity.
- Implement proper error handling and logical validations for robust and reliable data ingestion.
- Implement caching mechanisms or other performance optimization techniques for real-time performance.
- Follow industry-standard URL conventions and best practices for designing RESTful APIs.
- Ensure correct JPA object mapping and utilize Spring Data Specifications, JPA criteria queries, or JPA native queries for flexible and powerful query building.
- Maintain a clean and organized project structure that adheres to industry best practices and coding conventions.
- Follow consistent naming conventions throughout the codebase.
- Separate concerns by having dedicated classes or components for input validation, error handling, and data mapping.
- Utilize features like streams and lambda expressions for more concise and readable code.
- Use Lombok annotations to reduce boilerplate code.
- Implement proper logging using a logging framework like SLF4J or Log4j.

Don'ts:

- Don't mix different types of data (shopper personalized product lists and product metadata) in the same API endpoint.
- Don't fetch unnecessary data or perform inefficient database operations when retrieving the shopper's personalized product list.
- Don't mix concerns by combining different responsibilities (e.g., input/output APIs, DTOs, and service classes) in the same class or module.
- Don't introduce unnecessary complexity in DTOs or mapping logic.
- Don't perform inefficient database operations or fetch unnecessary data.
- Don't duplicate validation logic throughout the codebase, as it can lead to inconsistencies and maintenance issues.

Good to have:

- Consider implementing integration tests to ensure end-to-end functionality.
- Containerize the application using Docker for easy deployment and scalability.