Technical Documentation & Justifications

This document details the technical decisions and choices made for this e-commerce backend. The primary goals of these decisions are to ensure scalability, maintainability, where this code to be used in a production environment.

1. Caching Strategy for Product Listings

A caching layer is implemented for specific read heavy endpoints to reduce database load and improve response times for users

Endpoint Cached:

- **GET /products**: The endpoint that lists all available products.

Technical Justifications:

- **Why caching:** The full product catalog is requested frequently by users browsing the site in a production enviroment. However, the product data itself (name, price, description) does not change frequently, at least not compared to the request to view them. Caching this data significantly reduces the number of repetitive read queries to the database, leading to faster page loads.
- **Caching method:** We use a DB store like **Redis**. Redis is chosen for its cheap plans and high availability in the cloud.
- **Justification for Time-to-Live (TTL):**
    - A TTL of 5 minutes is set for the product list cache.
    - This TTL functions as a good fallback method. While we have logic to actively clear the cache when a product is created, updated, or deleted, the TTL makes sure that that if this invalidation fails, users would see the updated product list within 5.
- **Cache Invalidation:** The cache is cleared whenever a write operation occurs on the products table , like create, modify or delete, so that users aren’t shown old data.

2. Authentication with JWT

For user authentication and authorization, we have implemented a stateless mechanism using JSON Web Tokens (JWT).

Technical Justifications:

- **Scalability:** JWTs are stateless. After a user logs, the server generates a JWT containing the user's ID and role. This token is sent to the client and included in the header of subsequent requests. The server does not need to store session information, making it easier to scale the application.
- **Reduced Database Lookups:** The user's role (user or admin) is directly in the JWT token. For protected endpoints that require role-based access control, the server can verify the

role from the token's signature without needing an extra database query on every single request.

3. Repository Pattern for Data Abstraction

The codebase is structured around a Repository Pattern as was used in previous weeks, where each repository class encapsulates the data access logic for a specific database entity and function.

Technical Justifications:

- **Separation of Logic:** This pattern cleanly separates the logic of the application from the details of data management. The rest of the application does not need to know how the data is stored or recalled. It simply calls methods like user_repo.add_user().
- **Maintainability & Testability:** Centralizing data logic in one place for each type makes the code easier to maintain and debug. It also simplifies unit testing, as the repositories can be easily mocked like was done in this project.
- **Data base migrations:** Should we ever need to switch our database from PostgreSQL to another, we would only need to update the repository classes. The main logic remains unchanged, as well as the endpoint logic.