Product Management Endpoints for a Retail/E-commerce Application

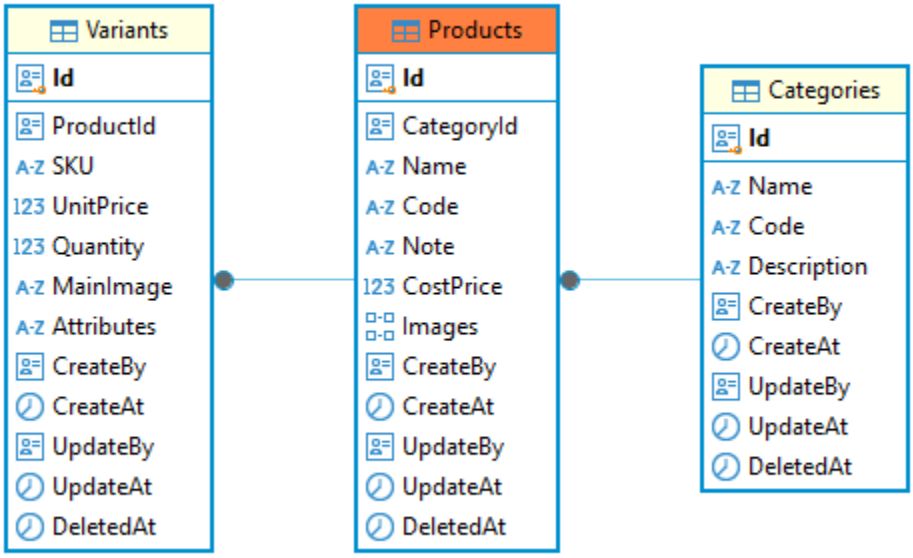
http://assignment-react-app.s3-website-ap-southeast-1.amazonaws.com/

Github source code: nhanne/ProductService

Approach

Understand the requirements, design the domain models, then build with Clean Architecture + DDD.

Database



- SQL (PostgreSQL) because product data needs ACID and consistent relationships.

- Flexible attributes stored in JSONB or Variant table → can add new product attributes anytime without schema changes.

- Split read/write DB later for performance. On read side, disable EF change tracking to speed up queries.

Tech Stack

- .NET 8, Entity Framework Core (Code First)

- MediatR (CQRS), FluentValidation (input validation)

- Serilog (logs), Autofac (DI)

- API Design: Standard RESTful CRUD, Swagger (API docs), Use DTOs for requests/responses.

- Validate: input with FluentValidation, Always return global exception on error (400/404/500).

Caching

I use Redis for caching, and basically there are two common strategies:

1. Lazy Loading (Cache-Aside)

- On read: check cache → if miss, load from DB → put result into cache.

- Cheap and simple, but may have stale data right after a write.

2. Write-Through

- Whenever the database is updated, I also update the cache immediately.

- Ensures the cache always has the latest data, but it increases write cost and latency.

My Approach: Hybrid

- Frequently updated or critical data → Write-Through (Keeps cache fresh and avoids stale reads).

- Less frequently updated data → Lazy Loading (Saves cost and still improves performance).

Concurrency

There are three ways to handle concurrent updates:

1. Optimistic Locking (use RowVersion)

- No lock, just check version when saving.

- Fast and scalable, but conflicts require retry.

2. Pessimistic Locking (SELECT … FOR UPDATE)

- Lock row until transaction finishes.

- No retry needed, but blocks other requests and reduces throughput.

3. Queue / FIFO Processing

- Put requests into a queue and process one by one in background.

- Handles huge traffic, avoids conflicts, but is eventually consistent.

Query Performance

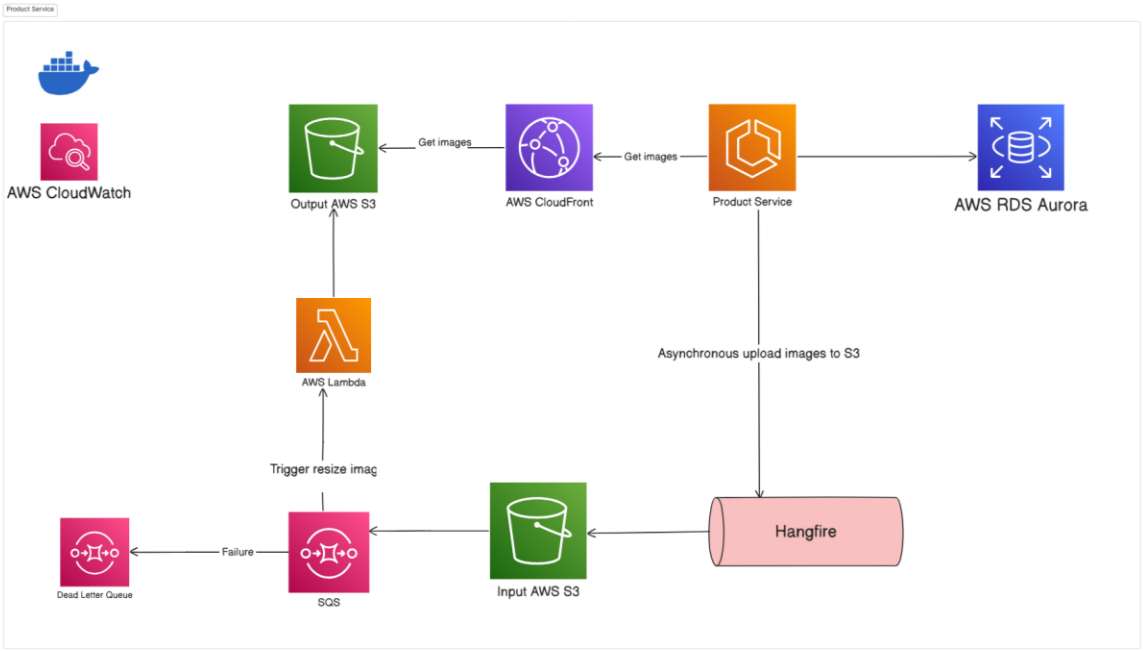
AsNoTracking(), indexes, and read-only repository to handle queries traffic.

Limitation

Even though I don't have too much time to implement full caching and concurrency logic, I clearly understand the best practices and the trade-offs for each approach (optimistic/pessimistic locking, queueing, lazy-loading cache vs write-through, etc.).

Future Imporve

Unit Testing
Write automated unit tests to cover all business logic and edge cases.
Image Handling
Move image storage to **Amazon S3** with signed URLs and proper lifecycle policies.
Monitoring & Search
Implement full logging/monitoring with Serilog + Kibana.
Build a search engine for products using **Elasticsearch**.
Deployment & Infrastructure
Currently: Client deployed to S3, API deployed as **Docker container on ECS**.
Future: Improve **networking, security groups, and standardize infrastructure** for better security and maintainability.



All diagram: https://app.eraser.io/workspace/ZuxJp0sPbJ4zaCz4Pmvn?origin=share