Product Verification Interview Task (FastAPI + MySQL + MongoDB)

1. Objective

Build one feature in this architecture style:

```
API -> Use Case -> Application Service -> Domain -> Repositories/Adapters
```

Feature:

- Create product (`pending_verification` by default)
- Verify product
- If checks pass -> `active` ; else -> `rejected`
- Emit domain events via a dummy dispatcher

Timebox: **6-8 hours** (single sitting) or **1 day max**.

2. What We Are Testing

- Can you follow layered architecture strictly?
- Can you use both MySQL and MongoDB intentionally?
- Can you model state transitions and domain events?
- Can you write clean tests around use cases?

3. Required Implementation

A. API Layer (FastAPI routers only)

Implement endpoints:

- `POST /api/v1/products`
- `POST /api/v1/products/{product_id}/verify`
- `GET /api/v1/products/{product_id}`

Rules:

- Router does request/response mapping only.
- Router must call Use Cases, not services/repositories directly.

B. Use Case Layer (separate, mandatory)

Create:

- `CreateProductUseCase`
- `VerifyProductUseCase`
- `GetProductUseCase`

Rules:

- Orchestrates workflow and transaction boundary (`UnitOfWork`).
- No HTTP concerns.
- No DB model mapping logic.

C. Application Service Layer

Create/update services for:

- product creation/update
- verification evaluation logic (or delegate to domain policy object)

D. Domain Layer

- Product entity with statuses: `pending_verification` , `active` , `rejected`
- Valid transition rules:
  - `pending_verification -> active`
  - `pending_verification -> rejected`
- Domain events:
  - `ProductCreatedPendingVerification`
  - `ProductVerificationCompleted`

E. Adapters/Repositories

- **MySQL**: product core record (`id` , `name` , `price` , `currency` , `status` , `created_at` , `updated_at`)
- **MongoDB**: verification/checklist document (`product_id` , `checks` , `reasons` , `verified_at`)

F. Dummy Event Dispatcher

Implement local dispatcher (no AWS):

- logs events to console and/or in-memory list
- injectable through DI
- test-friendly

4. Verification Rules (for assignment)

Pass only if:

- name present
- category present
- currency present
- price > 0
- stock_quantity >= 0
- At least 1 asset

If any fail:

- status = `rejected`
- reasons saved in Mongo

5. Deliverables

- Working code with the 3 endpoints
- Use Case layer implemented as separate folder/module
- MySQL + Mongo persistence wired
- Dummy dispatcher wired via DI
- Tests:
  - unit tests for verification decision logic
  - integration test for create -> verify flow
- `ASSIGNMENT_NOTES.md`:
  - architecture choices
  - what is stored in MySQL vs MongoDB
  - assumptions and tradeoffs

6. Hard Constraints (Fail if violated)

- No business logic in routers
- No repository calls directly from routers
- Use cases must exist as a distinct layer
- No skipping tests
- Status must start as `pending_verification`

7. Scoring Rubric (100)

- Layering correctness (API/UseCase/Service/Domain separation): 30
- Functional correctness + state transitions: 25
- MySQL/Mongo integration quality: 15
- Event dispatcher design + DI: 10
- Test quality: 15
- Code clarity/docs: 5