Technical Internship Case Assignment

Context:

As part of our tech team, you can shape your learning path by focusing on frontend, backend, or full-stack development. This assignment shows us how you think and communicate your decisions. We're not looking for perfection — we evaluate your reasoning and approach.

Your Assignment:

Choose one of the two tracks — **Frontend (Track A)** or **Backend (Track B)**. If you have extra time and interest, you're welcome to complete both, but this is entirely optional!

What to Submit:

1. **Track selection**
   a. Tell us which track(s) you chose.
2. **Source code**
   a. A link to your **GitHub/GitLab repository** (preferred), or
   b. A **zip file** with your project.
3. **Instructions to run**
   a. A short `README.md` with:
      i. How to install dependencies.
      ii. How to run the app or backend.
      iii. Any configuration we need.
4. Loom **links** *(~30–50 minutes; Loom free tier: ~5 min per recording, up to 25 recordings)*
   a. One video at the start (your plan and approach)
   b. One video at the end (walkthrough of what you built + next steps)
   c. Optional: 1–2 short progress videos
   d. Explain: Your decisions, where you intentionally cut corners, what you would improve with more time.

Additional informations:

1. Time Management (Guideline):
   a. Track(s): each ~1–2 hours
   b. If you run out of time, simply stop and use your final Loom to explain what's done and what you'd do next. We care about your thinking, not perfection!
2. We'll look at:
   a. Clarity of thinking
   b. Trade-off awareness (and explicit "I cut this corner because…")
   c. Code structure and readability
   d. Use of chosen tech stack
   e. Basic UX (Track A) or API design / schema design (Track B)

Track A: (Frontend) – Mini E-commerce Store

Goal: build a small e-commerce frontend using: https://fakestoreapi.com/
Tech Stack: Nuxt, Vue, Svelte, Angular, Next, React, or even vanilla JS, etc.

A1. Product overview page

Create a page that:

- Fetches a list of products from the Fake Store API.
- Shows a **grid or list** of products with at least:
  - Product image
  - Title
  - Price
  - Short or truncated description
- Includes **search**:
  - At least search by **product title**.
  - You decide whether search is:
    - Client-side filtering, or
    - Implemented via API calls (if applicable).

Nice-to-have **(not required):**

- Basic **filters** (category, price range, etc.).
- Proper **loading** and **error** states.

A2. Product detail page

When a user clicks a product:

- Navigate to a **detail page** (e.g. `/products/:id`).
- Show at least:
  - Product image
  - Title
  - Price
  - Full description
  - Category
- Make sure:
  The URL is **shareable** (opening the link shows that product).
  - You handle **loading** and **error** cases (e.g. product not found).

A3. (Bonus) Cart

Optional but appreciated:

- Allow users to **add products to a cart** from the overview or detail pages.
- Cart should:
    - Show cart items and quantities.
    - Allow updating quantity / removing items.
    - Show a **cart total**.
- Implementation options:
    - In-memory state (e.g. React/Vue store).
    - Or persisted in localStorage.

A full checkout is **<u>not</u>** required.

Track 2: (Backend) – E-commerce Schema & Products API

Goal: design a simple ecommerce backend and expose a `GET /products` endpoint.
Tech Stack: Django, Node/Express, Nest, Laravel, Flask, Rails, etc.

B1. Database schema for an ecommerce store

Design a small relational schema for an ecommerce system.

At minimum, include tables for:

- `products`
- `categories`
- `customers` (or `users`)
- `orders`
- `order_items` (line items)

You can provide this as:

- SQL `CREATE TABLE` statements, or
- A diagram (e.g. ERD), or
- A structured text description (with fields, types, relationships).

We'll look at:

- Clear **relationships** (e.g. product–category, order–order_items).
- Reasonable **normalization** (no obvious duplication).
- Sensible fields (e.g. price, currency, timestamps, etc.).

B2. Products API endpoint

Implement a `GET /products` endpoint that:

- Returns a list of products from your schema (or a mocked equivalent).
- Supports some basic **query parameters**, for example:
    - `?search=...`
    - `?category=...`
    - `?page=...&page_size=...` (pagination)

Data source options:

- Real database (SQLite/Postgres/MySQL); or
- In-memory array with a clear schema; or
- ORM models with seed data.

In your Loom, briefly explain:

- Why you designed the schema this way.
- How the endpoint works, including parameters and response shape.
- What you’d add for a real production system (e.g. validation, auth, logging, tests)