Assessment for Backend Engineer

Thank you for your interest in joining our team. We appreciate the time and effort you've taken to apply. To proceed with the evaluation, please choose **one** of the tasks listed and complete it according to the instructions provided.

Our average review time is **5 days**, and we kindly ask that you **carefully follow all task guidelines** to ensure a smooth assessment process.

E-commerce Ordering & Payment System

1. Project Overview

Build a backend system for managing users, products, orders, and payments with support for multiple payment providers (Stripe, bKash). The system must expose APIs for user registration, product management, order creation, checkout, and payment verification.

2. Requirements

2.1 Functional Requirements

2.1.1 User Management

- Users can register and log in.
- User data stored in the **Users** table.
- Email must be unique.
- Users can view their own orders and payments.

2.1.2 Product Management

- Admin can create, update, delete products.
- Product fields:
  id (PK), name, sku (unique), description, price, stock, status (active/inactive), timestamps

- Users can view product lists and details.

2.1.3 Order Management

- Orders belong to users.
- Fields:
  id, user_id (FK → Users.id), total_amount, status (pending, paid, canceled), timestamps
- An order consists of multiple products.
- Requires an **OrderItems** table:
  Id, order_id (FK), product_id (FK), quantity, price, subtotal

2.1.4 Payment System

Payment handled via **Stripe** and **bKash**.

Stripe

- Create payment intent.
- Confirm payment.
- Webhook for payment updates.
- Store:
  - provider = "stripe"
  - transaction_id = payment_intent_id
  - status = pending/success/failed

bKash

- Checkout API integration.
- Execute payment.
- Query payment.
- Store:
  - provider = "bkash"

- transaction_id = bkash_payment_id
- status = pending/success/failed

2.1.5 Payment Table

Id, order_id (FK), provider (stripe/bkash), transaction_id (unique), status, raw_response (JSON), timestamps

2.1.6 Order Flow

1. User selects products → creates order.
2. User chooses the payment provider.
3. The system initiates payment.
4. Provider confirms or fails payment.
5. Order status updates accordingly.
6. Stock is reduced after successful payment.

2.2 Core Design & Algorithm Requirements

2.2.1 OOP Requirement

Use OOP classes (User, Product, Order, Payment) in User Management, Product Management, Order Management, and Payment System to organize logic and support future extensions.

2.2.2 Data Structure Requirement

Use relational tables and indexed fields in Users, Products, Orders, OrderItems, Payments, and Categories for efficient querying and hierarchical relationships.

2.2.3 Algorithm Requirement

Implement deterministic algorithms in Order Management to calculate totals and subtotals, and in Product Management to safely reduce stock after payment.

2.2.4 Design Pattern Requirement

Implement strategy pattern in the Payment System to switch between Stripe, bKash, and future providers without modifying core order logic.

2.2.5 DFS + Caching Requirement

Use DFS in Product Recommendation / Category Hierarchy to traverse the category tree for recommending related products efficiently, and cache the category tree in **Redis/memcached** to minimize database calls and speed up repeated traversals.

3. Non-Functional Requirements

- Clean REST API design.
- Use migrations for DB.
- Proper data validation.
- Secure storage of API keys.
- Logging & error handling.
- Scalable schema for adding more payment providers later.

4. Deliverables

4.1 Documentation

- System architecture diagram.
- ERD (Users, Products, Orders, OrderItems, Payments).
- API documentation (Postman/Swagger).
- Payment flow diagrams (Stripe & bKash).

4.2 Code Deliverables

- Backend project (Node.js / Express.js/ Django/ FastApi — depends on your choice).
- Seeders for admin user and sample products.

4.3 Payment Integrations

- Stripe integration (test + live mode).
- bKash integration (sandbox + live).
- Webhook handlers for each.

4.4 Testing

- Unit tests for models.
- API tests for authentication, orders, and payments.
- Webhook test cases.

4.5 Deployment

- Environment configuration guide.
- Frontend on Vercel
- Backend running locally via ngrok
- Docker deployment (for backend + DB)