Full Stack Developer Test Task

Objective

Build a simplified **Full-Stack e-commerce system** with a **Laravel RESTful API** backend and a **React.js Frontend** interface

---

Key Requirements

1. Backend – Laravel (API)

Models & Relationships

- Create two main models: `Product` and `Order`.
- Each `Order` should store which products were purchased (many-to-many relationship).

API Endpoints

- `GET /products`: List all products with **pagination** and **filtering** (by name, price range, and optionally category).
- `POST /orders`: Place an order with validation (e.g., product availability, quantity in stock).
- `GET /orders/{id}`: Show detailed order view (products, quantities, total).

Features

- Use **Eloquent** for model interaction.
- Add **validation** for both product and order creation.
- Use **caching** for the `GET /products` endpoint.
- Implement **events** and **listeners** for the "Order Placed" event, where an email notification would be sent to the admin (no need to actually send the email, just trigger the event).
- Implement **basic search** and efficient **pagination**.
- Use **Laravel Sanctum** for authentication and protect order endpoints.

2. Frontend – React.js (UI)

Build a clean, responsive frontend that communicates with the Laravel API.

Do not use Inertia.js for communication between frontend and backend.

Screens to Include

- **Login Page**
- **Product & Order Page (combined functionality)**
    - Search products by name
    - Filter by price range and category
    - Paginate through product listings
    - For each product:
    - Input desired quantity
    - Add to current order
    - View and manage current selected items in a "Cart" or "Order Summary" section
    - Submit order
- **Order Details Page**
    - View items in the order, quantities, total cost

Frontend Requirements

- Use **Material UI (preferred)** or another CSS framework.
- Must be fully **responsive** and tested on common screen sizes.
- Use **React functional components** and hooks.
- Organize code cleanly and maintain separation of concerns.
- Use standard routing (e.g., React Router) for navigation.

Figma Ui Link: Design Here

Security Considerations

- Use **Sanctum** for authentication.
- Restrict order-related endpoints to **authenticated users**.
- Sanitize and validate all inputs to avoid common vulnerabilities (e.g., SQL injection).

Deliverables

Submit a single Laravel-based full-stack project where the React frontend is integrated within the Laravel application (e.g., served from `public/` or a dedicated `resources/js` folder using Vite or similar).

Project Structure

- **A complete Laravel project including:**
  - Migrations, models, controllers, routes
  - Laravel Sanctum for authentication
  - `.env.example` and setup instructions
- **The React frontend as part of the Laravel codebase (not a separate app)**
  - Either inside `resources/js/` (preferred if using Vite)
  - Or compiled into `public/` if using a separate build step
  - No usage of Inertia.js — use direct REST API calls from React
- **Responsive, clean UI using Material UI or any CSS framework**

API Documentation

- **Include inline documentation or a `README.md` describing:**
  - API endpoints
  - Setup instructions
  - Authentication flow
  - How to run both backend and frontend

Time Tracking

- **Before starting, record your estimated time**
- **On submission, include:**
  - Estimated time
  - Actual time taken
  - Be transparent — we value clarity and code quality over speed

Submission Format

- **Submit a single zipped folder containing:**
  - The unified Laravel + React project
  - All source code and the compiled frontend (`public/` or build folder)
  - Do not include `node_modules` or `vendor` directories