Laravel Technical Assessment

Objective

The goal of this assessment is to **develop a mini e-commerce application** with core functionalities and well-defined **API endpoints** for a **website, mobile app, and admin dashboard**. The solution should adhere to **best practices**, including **MVC architecture, RESTful API design, authentication, role-based access control (RBAC), dependency injection, localization.** The application must be **scalable, maintainable, and optimized for performance**, ensuring **security, efficient database management, and seamless user experience** across multiple platforms.

---

1. Core Functionalities & API Endpoints

1.1 User Management

Attributes:

Name , email, email_verified_at , password ,image, is admin

Features:

- Email verification (Send Email to user email contain link to verify)
- Role-based access control (RBAC) for **admin and customers**.
- Authentication using Google Account

Endpoints:

• POST `/api/register` – Register a new user.

• POST `/api/login` – Authenticate a user and return a token.

• GET `/api/user` – Retrieve authenticated user details.

• POST `/api/logout` – Log out a user.

1.2 Category Management

Attributes:

Name , Description , Image , Status (active , not active)

Features:

- CRUD operations for categories.
- API Resource Transformers for structured responses.

Endpoints:

POST `/api/categories` – Create a new category

GET `/api/categories` – Retrieve all categories

GET `/api/categories/{id}` – Retrieve category by ID

PUT `/api/categories/{id}` – Update category details

DELETE `/api/categories/{id}` – Delete a category

1.3 Product Management

Attributes:

Name , Description , Images (List), Price, discounted_price (if product has discount),

quantity (available in stock) ,Status (active , not active)

Features:

- Each Product belong to **one or more** category
- CRUD operations for products.
- API Resource Transformers for structured responses.
- apply pagination

Endpoints:

POST `/api/products` – Create a new product

GET `/api/products` – Retrieve all products

GET `/api/products/{id}` – Retrieve product by ID

PUT `/api/products/{id}` – Update product details

DELETE `/api/products/{id}` – Delete a product

1.4 Cart Management

Features:

- Use a **cart package** like gloudemans/shoppingcart.

Endpoints:

POST `/api/cart` – Add a product to the cart

GET `/api/cart` – Retrieve all cart items

PUT `/api/cart/{id}` – Update cart item quantity

DELETE `/api/cart/{id}` – Remove a product from the cart

POST `/api/cart/clear` – clear the cart

1.5 Order and Checkout

Attributes:

order number, List of product in the order , delivery address, user information , order status ("pending", "shipped", "delivered”), payment method, payment status (paid , not_paid)

Features:

- Since the cart is maintained on the server, **order creation will automatically use the cart contents**, eliminating the need to send products in the order request body.
- add status for the order "pending", "shipped", "delivered”
- add delivery address (city name, address name and building_number)
- enable payment, Integration with **Stripe & PayPal** for secure payments.

Endpoints:

POST `/api/orders` – Create a new order.

GET `/api/orders` – Retrieve all orders for a user

GET `/api/orders/{id}` – retrieve order details

PUT `/api/cart/{id}/status` – update order status

---

2. Project Requirement

2.1 Validation

• Implement comprehensive validation for all input fields
• Ensure data integrity (e.g., positive prices, valid status values)
• Return structured error messages following REST standards

2.2 Database Architecture

• Implement MySQL with proper schema design
• Optimize through strategic indexing

• Establish robust table relationships
• Ensure data consistency and referential integrity

2.3 API Documentation
• Document all endpoints with detailed specifications
• Include request/response formats and status codes
• Provide practical usage examples
• Add clear setup and testing instructions

2.4 Request Handling
• Implement RESTful CRUD operations
• Support multiple client platforms (web, mobile)
• Handle authentication and authorization
• Implement proper error handling and logging

2.5 Resource Management
• Design RESTful resource endpoints
• Map resources to database entities
• Implement proper resource relationships
• Ensure efficient resource serialization

2.6 Localization Management

The API should support language selection using the **Accept-Language** header. When making a request:

• **Accept-Language: EN** → The response should return content in **English**.

• **Accept-Language: AR** → The response should return content in **Arabic**.

• The API should return only the requested language's data, **without duplicating fields** (e.g., no name_en and name_ar together in the response).

• If the Accept-Language header is not provided, the API should **default to English**.

• This ensures efficient localization without unnecessary data redundancy.

---

2. Submission Requirement

- If you notice areas for improvement in the project, feel free to make changes. However, please **explain your reasoning** so we understand the modifications.
- **Implementing a server-side cart is a significant advantage** 💡. While it's optional, it's highly recommended. If you choose not to implement it, ensure

that products are included in the **Create Order API** request.

- Upload the project to **GitHub** and provide a **well-structured repository**.
- Include a **Postman collection** for testing API endpoints.
- Share the **GitHub repository link** and **Google Drive link (containing the Postman collection)** via email.

✅ **Deliverables:**

- **GitHub Repository** – Well-documented code with a clear project structure.
- **Postman Collection** – API testing files for easy validation.
- **Submission Email** – Send both links (GitHub & Google Drive) to the provided email.
- **Submission Email Address:** dev-submissions@it-trendco.com with subject line "backend-dev"

Human Resources Department

IT-TrendCO
Consulting and Operation