

# Full Stack Developer Assignment - Mini E-Commerce Platform

## Objective

The goal of this assignment is to evaluate your **end-to-end full-stack development skills**, including backend architecture, frontend/mobile development, database design, API development, deployment, and code quality.

You are required to build a **small but complete e-commerce application** where users can browse products, add them to a cart, and place orders. An admin dashboard should allow order management. No payment gateway integration is required.

---

## Tech Stack (Mandatory)

You are expected to use the following technologies:

### Backend

- Python
- Django
- Django Rest Framework (DRF)
- PostgreSQL
- Jinja templating (for admin dashboard)

### Frontend / Mobile App

- Flutter (Android APK required)

### DevOps & Tooling

- AWS (EC2 – free tier allowed)
  - Git & GitHub
- 

## Functional Requirements

### 1. User Application (Flutter)

#### Guest Usage & Authentication

- App should allow **guest users** to browse products and add items to cart without login
- When placing an order:
- User must either **log in / register**, OR
- Proceed as **guest checkout** by providing:
  - Full name

- Email address
- Guest order details should be stored with the order

## Product Listing

- Display a list of products
- Each product should show:
  - Product image
  - Name
  - Price
  - Short description
  - Stock availability

## Cart Management

- Add products to cart
- Update quantity of products in cart
- Remove products from cart
- Cart should support **multiple products at once**

## Order Placement

- Place an order from the cart
- No payment gateway is required
- Validate:
  - Product stock availability
  - Minimum order amount (configurable)

## Coupon Application (App)

- Input field to apply coupon code at checkout
- Show:
  - Applied discount
  - Updated payable amount
- Handle coupon errors gracefully (expired, not applicable, minimum value not met)

---

## 2. Backend (Django + DRF)

### APIs

Create REST APIs for:

- User registration & login (basic authentication or token-based)
- Product listing
- Cart management
- Order creation
- Coupon validation

### Database Models (Suggested)

- User
- Product
- Cart
- CartItem
- Order
- OrderItem

- Coupon

You are free to enhance or optimize the schema as needed.

---

### 3. Admin Dashboard (Django + Jinja)

Create a simple admin dashboard using **Django views + Jinja templates** (do not rely only on Django Admin).

#### Admin Features

- Login for admin
- Manage products (add, edit, delete)
- View orders
- Update order status:
  - Pending
  - Shipped
  - Cancelled
- View applied coupons per order

#### Coupon Management (Admin)

- Create new coupons with:
- Coupon code
- Discount type (percentage / flat)
- Discount value
- Validity period (start & end date)
- Minimum cart value
- Usage limit (optional)
- Coupon applicability:
  - Applicable to **all products**, or
  - Applicable to **specific product(s)** only
- Enable / disable coupons

---

## Deployment Requirements

### Backend Deployment

- Deploy Django backend on **AWS EC2 (Free Tier)**
- Configure:
  - Gunicorn
  - Nginx (preferred)
  - PostgreSQL
- Environment variables for secrets

### Mobile App

- Build a **release APK** for the Flutter app
- App should point to the deployed backend APIs

---

## Repository & Submission Guidelines

### GitHub

- Create **separate repositories** (or clearly separated folders) for:
- Backend (Django)
- Frontend (Flutter)
- Use meaningful commit messages
- Include a proper `README.md` with:
  - Project overview
  - Tech stack
  - Setup instructions
  - API documentation (basic)

### Submission

Provide the following: - GitHub repository link(s) - Deployed backend URL - Flutter APK download link

---

## Evaluation Criteria

You will be evaluated on: - Code quality & structure - API design & data modeling - Use of Django & DRF best practices - Flutter UI & state management - Proper use of Git - Deployment correctness - Edge case handling (cart, coupons, stock, orders)

---

## Bonus (Optional)

- Dockerized setup
  - Swagger / OpenAPI documentation
  - Order history screen in app
  - Clean UI/UX
- 

## Timeline

- Expected completion time: **2 weeks**
- 

## Notes

- Focus on **clarity, correctness, and completeness**, not over-engineering.
- This assignment simulates a real-world startup-style project — keep it practical and clean.

Good luck 