“SHOP MAJUU GET IT LOCAL” / SOKONI MAJUU PAYMENT SYSTEM

I will be building a secure system for local users that will help a user be able to use payment options like MPESA , STRIPE and EBAY to make even international purchases and not rely on a third party for that.

My key motivation for this project is to simplify the process of ordering items in Amazon which has proven a challenge due to the fact that most users lack international credit cards or visa cards thus relying on relatives/family abroad to order and ship for them despite the improvement that the international logistic industry has taken.
Upon completion of this project ordering stuff through both local and international stores will be made easier than before thus boosting sales for the mentioned online stores that will be using that API.

Features of the System

- **User management:**

Registration, login, 2FA (Two-Factor Authentication)
Roles: Admin, Merchant, Customer
KYC verification (optional but recommended)

- **API Key management**

Issue and manage API keys per merchant

Usage limits and key rotation

- **Transaction Management**

Payment initiation
Status tracking (pending, completed, failed, refunded)
Refunds and chargebacks
Webhooks for payment events

- **Security**

HTTPS enforcement
JWT or OAuth2 for API authentication
Input validation and CSRF/XSS protection
IP whitelisting, rate limiting

- **Audit Logging**

Log all critical operations: transactions, login, key access, etc.

- **Payment Method Integration**

Support for cards, bank transfers, wallets, or mock payments

- **Dashboard (optional frontend or API-based)**

Transaction history
Account management
Reporting tools

Multi-Payment Support

MPESA (via Daraja API or direct integrations)
Stripe for card-based international payments
PayPal/Proxy support for eBay (if no direct integration)

- **Merchant Integration**

API key per store (merchant)
Webhook URL registration for order updates
Store-level transaction records

- **Customer-Facing API**

Tokenized checkout
Wallet or mobile money integration
International/local payment selection

- **KYC for Merchants**

Email verification + business registration info
API key activation after review

- **Currency Conversion (Optional)**

Auto-fetch rates using a currency API (e.g., exchangerate-api.com)

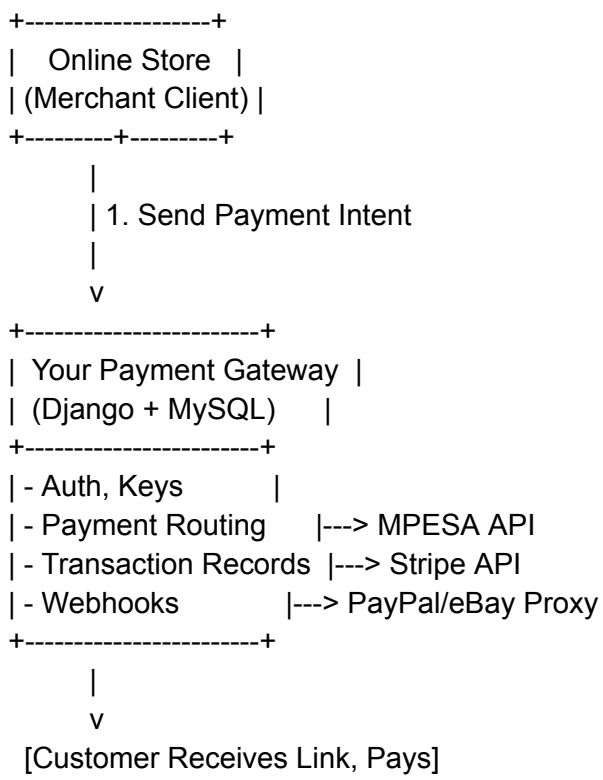
- **Mobile-First Support**

Optimized JSON APIs for frontend/mobile integration

- **International Order Proxying (Optional Later Stage)**

Ability to proxy-buy from Amazon using stored credentials/API automation

Architecture Overview (Simplified)



App	Description
`users`	User and merchant management
`payments`	Core payment logic (Stripe, MPESA, PayPal)
`merchants`	Store data, API keys, webhook URLs
`transactions`	Log of all payment actions
`webhooks`	Delivery + retry system for notifying stores
`currencies`	Currency rates and conversion support
`audit`	Logs and sensitive action tracking

📦 Resources for the Stack

MPESA Integration

- Safaricom Daraja API Docs
- Use `requests` or `httpx` for API calls

Stripe

- Stripe Python Docs
- Use `stripe` Python SDK

eBay (Indirect)

- eBay Buy API
- If needed, fallback to PayPal SDK

📆 4-Week Work Plan

Week	Theme	Key Deliverables
1	Project Setup + Merchant Onboarding	Django project setup, DRF, JWT Auth, API keys, KYC
2	MPESA + Stripe Integration	MPESA STK Push, Stripe PaymentIntent, Payment models
3	Transactions + Webhooks	Store transactions, webhook retry system, merchant dashboard
4	Testing + Documentation	End-to-end testing, Dockerize, Postman collection, deployment

BE Capstone Part 2 : Design Phase

DESIGN OF DATABASE SCHEMA

Main Entities :

1. User

2. OrderRequest
3. Item
4. Payment
5. CurrencyRate
6. Warehouse
7. Shipment (optional, for tracking later)
8. TransactionLog

ERD STRUCTURE TEXT FORMAT:

User
- id (PK)
- full_name
- email
- phone
- created_at

OrderRequest
- id (PK)
- user_id (FK → User)
- item_url
- status (e.g., pending, paid, processing, shipped)
- created_at

Item
- id (PK)
- order_id (FK → OrderRequest)
- name
- price_original (in source currency)
- currency
- quantity
- price_converted (in local currency)

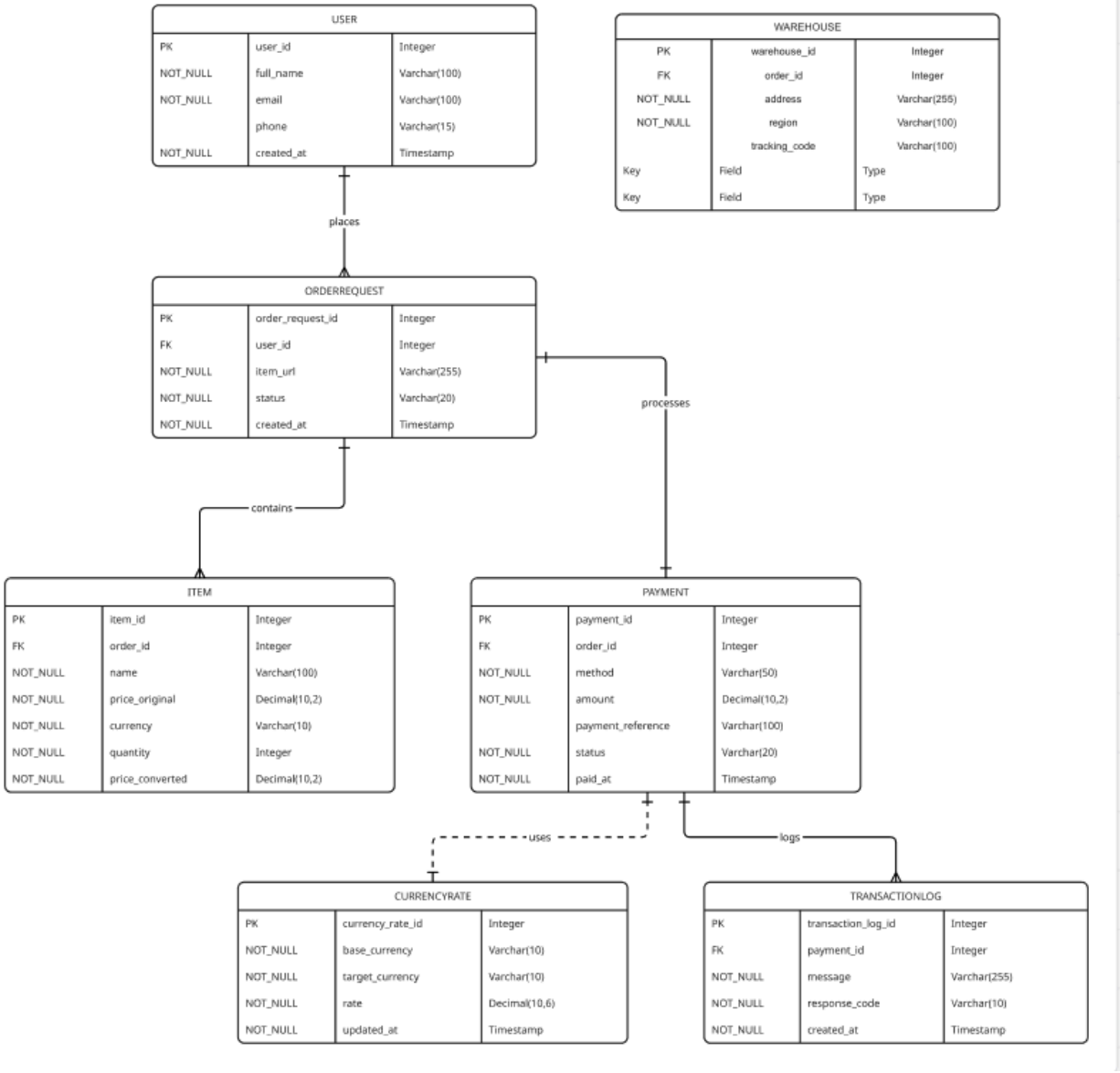
Payment
- id (PK)

- order_id (FK → OrderRequest)
- method (e.g., MPESA, VISA)
- amount
- payment_reference
- status
- paid_at

CurrencyRate
- id (PK)
- base_currency
- target_currency
- rate
- updated_at

Warehouse
- id (PK)
- order_id (FK → OrderRequest)
- address
- region
- tracking_code

TransactionLog
- id (PK)
- payment_id (FK → Payment)
- message
- response_code
- created_at



📦 High-Level Data Flow (Context Level DFD):

[User]
|
v
Submit Item URL + Quantity
|
v
[System]
|
├─> Scrape/Search External Store (get price, name)
├─> Fetch CurrencyRate

└─> Calculate Total + Forex Fee
|
v
Show Total Price to User
|
v
[User Pays] → [Payment Gateway (M-Pesa / Visa)]
|
v
[System Confirms Payment] → Update OrderRequest → Issue Warehouse Address

✅ PART 3: API Endpoints

🔐 Auth (if needed)

```
POST /api/auth/register
POST /api/auth/login
GET /api/user/profile
```

📥 Order Requests

```
POST /api/orders
Body: {
  "item_url": "https://example.com/product/123",
  "quantity": 1
}

GET /api/orders/{order_id}
GET /api/orders (list user orders)
```

🔍 Price Lookup & FX Calculation

```
POST /api/orders/{order_id}/calculate
→ retrieves price, converts with forex, returns total
```

```
GET /api/currency-rates
```

💳 Payments

```
POST /api/orders/{order_id}/pay
Body: {
  "method": "mpesa" | "visa"
}

POST /api/payments/mpesa/callback
POST /api/payments/visa/webhook
GET  /api/payments/{payment_id}
```

🏠 Warehouse Management

```
GET /api/orders/{order_id}/warehouse
```

📦 (Optional) Shipments

```
GET /api/orders/{order_id}/shipment
```