Smart Stock Inventory Optimization for Retail Stores

G. JYOTSNA SAI & P. VARSHA

BACKEND PRESENTATION

Agenda

- 1. Project Overview
- 2. Architecture
- 3. Environment & Libraries
- 4. Detailed Backend Code (models, serializers, views, urls)
- 5. API examples & Postman
- 6. Tests, Migrations & Deployment
- 7. Security & Best Practices
- 8. Future Work

Project Overview

- Goal: Build a robust Django backend that automates inventory tracking and optimization for retail stores.
- Key capabilities:
- • CRUD APIs for products and stock movements
- • Low-stock detection & restock suggestions
- • Clean separation of concerns (models → serializers → views)
- • Ready for ML-based forecasting in future

System Architecture

- Frontend (React/Vite) <--REST API--> Django Backend <--ORM--> Database (Postgres)

- Backend components:
- • Models: data schema
- • Serializers: validation & transformation
- • Views/ViewSets: HTTP endpoints
- • Signals/tasks: notifications & async jobs

Environment & Libraries

- Python 3.11+
- Django 5.x
- Django REST Framework
- django-cors-headers
- psycopg2-binary (Postgres)
- pytest / Django test framework
- celery (optional for async tasks)
- redis (broker for celery)

- Dev tools: Postman, pgAdmin, Docker (optional)

API Examples (Postman / curl)

- Example API calls:
- GET all products:
- GET /api/products/
- Create product (POST):
- POST /api/products/
- body:
  {"sku":"P001","name":"Pen","category":1,"price":10.5}

Transactions, Tests & Migrations

- • Use database transactions for multi-step updates (atomic)
- • Write pytest / Django tests for models, serializers, and views
- • Example: test stock decrement on OUT movement
- • Run migrations: python manage.py makemigrations && python manage.py migrate

- Testing tips:
- - Use factories (factory_boy)
- - Use APIClient from rest_framework.test

Deployment & Scaling

- • Use PostgreSQL in production
- • Serve with Gunicorn + Nginx
- • Use environment variables for secrets
- • Use Docker for reproducible deployments
- • Consider Celery + Redis for async tasks (notifications, reports)
- • Enable logging and monitoring (Sentry, Prometheus)

Security & Best Practices

- • Do not commit SECRET_KEY or DB passwords — use env vars
- • Use HTTPS in production
- • Implement authentication (JWT) and permissions
- • Rate-limit sensitive endpoints
- • Validate all inputs and use serializer validators
- • Use prepared DB queries via ORM to avoid SQL injection

Future Enhancements — ML & Analytics

- • Add demand forecasting model (ARIMA / Prophet / LSTM)
- • Use historical sales to predict reorder quantities
- • Build dashboard endpoints for analytics
- • A/B test restock strategies
- • Add role-based access and audit logs

Thank You