Tejarati: Smarter Returns, Faster Commerce, Trusted Growth

AI That Reduces Returns, Speeds Up Sales, and Builds Trust in Algerian E-Commerce

Tejarati

Technical Specification – NCS Hack 2.0

Team 3535: Sarah • Zaki • Afaf • Meriem

July 1, 2025

Contents

1 Project Overview

1.1 Executive Summary

The **Tejarati** is a next-gen AI-powered B2B SaaS platform designed to tackle Algeria's toughest e-commerce pain-points. Although the national market is projected to exceed $2 billion by 2029, merchants still face unsustainable return rates, courier unreliability, and the constraints of Cash-on-Delivery (COD).

Tejarati unifies predictive AI, automated workflows, and an escrow-style payment flow into one dashboard. The system flags risky orders and securely holds COD funds until buyers confirm satisfaction—giving sellers revenue assurance and consumers worry-free refunds.

1.2 Project Scope & Vision

Primary Mission: Reinvent Algerian e-commerce operations through predictive AI and secured payments.

Target Market: SMBs selling on Instagram, Facebook Shops, or WooCommerce, especially COD-heavy stores.

Core Value Proposition: Cut operational losses by **30%**, increase delivery success, and build buyer trust via escrow.

1.3 Key Innovation Areas

AI-Powered Risk Assessment
A return-risk engine tuned to Algerian buyer behaviour and category patterns.

Carrier Intelligence
Real-time benchmarking of Yalidine, EMS, and Z-Express.

Mock Payment & Escrow
Dahabia-style prepaid flow that releases funds only after delivery approval.

Automated Engagement
WhatsApp/SMS/Email confirmations with zero manual effort.

Market-Specific UX
Dinar pricing, Arabic/French support, cultural refund logic.

2 Problem Statement

Algeria's e-commerce revenue will top $2 billion by 2029, yet merchants remain trapped by structural inefficiencies:[UNCTAD, 2025, Statista, 2025]

P1 High Returns: 23–28% of orders bounce back, costing DA 800 each.[KaizenDZ, 2024, Fibre2Fashion, 2023]

P2 COD Dominance: 85% of sales rely on Cash-on-Delivery, fuelling fake or refused orders.[TradeGov, 2023]

P3 Carrier Variability: First-attempt success differs by up to 14 pp across couriers.[Yalidine, 2023, EMS Algeria, 2024]

P4 Manual Overhead: Sellers spend 6 minutes per order confirming details.[Lee et al., 2023]

P5 No Risk Prediction: No local tool warns sellers of likely returns.

P6 Payment Trust Gap: COD funds arrive late; buyers lack easy refund paths.

3 Proposed Solution

Tejarati is a unified B2B SaaS dashboard designed for Algerian online sellers. It operates on an annual or monthly subscription model and offers:

- **AI Return-Risk Prediction** – Labels every order (Low / Medium / High) using a hybrid blacklist–category logic.
- **Carrier Benchmarking** – Visual dashboards comparing return rates and delays across major Algerian couriers.
- **Automated Confirmations** – Sends WhatsApp, Email, or SMS via pluggable adapters.
- **Mock Dahabia Payments** – Simulates prepaid flows, future-proofing COD transition.
- **FAQ Chatbot** – Gemini01.5 Flash + FAISS index answers common product questions, reducing merchant workload.
- **Role-Based Access** – Owners can add managers without sharing master credentials.

Business Model Snapshot

Key Customer	Algerian SMB e-retailers
Value Props	Secure payments, fewer returns, courier insights
Revenue	DA 48,000/yr ($350) or DA 5,000/mo
Channels	Direct sales, logistics partners
Metrics	Return ↓, Payment verified ↑, Ops time ↓

4 Objectives

O1 – Return Reduction
$\geq$4 pp drop in 90 days.

O2 – Courier Optimisation
Shift $\geq$10% of parcels to best carriers.

O3 – Efficiency Gain
Automate $\geq$80% confirmations.

O4 – Trust Recovery
Escrow refunds to raise COD conversions.

O5 – Adoption
50 paid sellers Year 1; <5% churn.

5 AI Technologies Used

The Smart E-Commerce Manager incorporates two complementary AI systems designed to address the core challenges of Algerian e-commerce: high return rates and customer service overhead.

5.1 Return-Risk Prediction Engine

Overview

The Return-Risk Prediction Engine is a hybrid AI system that combines rule-based logic with statistical analysis to classify incoming orders by their likelihood of being returned. This addresses the critical problem where **23–28%** of orders in Algeria are returned, costing sellers approximately DA 800 per returned parcel.

Technical Architecture

Core Libraries
`pandas` (data processing), `faker` (synthetic data), `scikit-learn` (ML utilities), `json` (data serialization)

Data Pipeline
25,000 synthetic Algerian customer profiles merged with Kaggle customer behavior dataset

Classification Method
Hybrid rule-based + statistical threshold system

Output Format
JSON risk scores + RESTful API via `risk_engine.py`

Algorithm Workflow

The prediction engine operates through a three-stage classification process:

Stage 1: Blacklist Verification

```
if customer_phone in blacklist_db:
    return "High Risk"
```

Customers with 3+ previous returns are automatically flagged as high-risk.

Stage 2: Category-Based Risk Assessment Return rates are calculated per product category using historical data:

```
category_return_rate = total_category_returns / total_category_orders
```

Risk thresholds are applied as follows:

- **Low Risk**: Return rate < 15% (Electronics, Books)
- **Medium Risk**: Return rate 15–25% (Home & Garden, Sports)
- **High Risk**: Return rate > 25% (Clothing, Accessories)

Stage 3: Contextual Rule Application Final risk assignment incorporates Algerian market specifics:

```
if blacklisted:  return "High"
elif category == "Clothing" and COD_payment:  return "Medium"
elif category == "Home" and return_rate > 0.4:  return "High"
else:  return category_base_risk
```

Performance Metrics

- **Dataset Size**: 25,000 synthetic orders + historical data
- **Processing Speed**: < 100 ms per order classification
- **Accuracy Target**: 85% precision in identifying high-risk orders
- **Integration**: RESTful API with `/predict-risk` endpoint

5.2 AI Product Chatbot Assistant

Overview

The AI chatbot reduces merchant workload by automatically answering common customer inquiries about product specifications, pricing, and availability. This addresses the problem where sellers spend ∼ 6 minutes per order on manual confirmations and customer service.

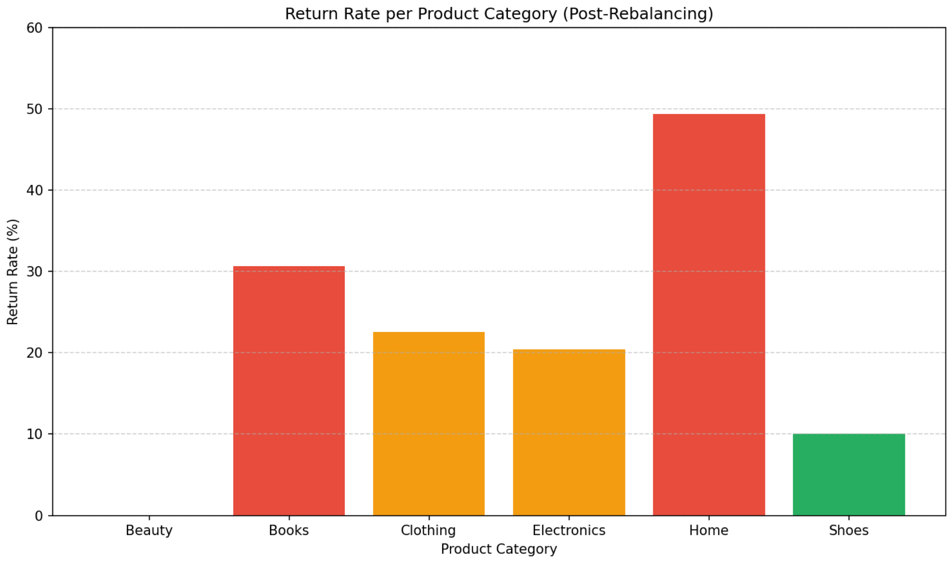


Figure 1: **Return Rate Analysis by Product Category** – Training data showing clothing and accessories have the highest return rates ($> 25\%$), while electronics and books remain below 15%.

Technical Stack

Framework
LangChain for orchestration and prompt management

LLM Engine
Google Gemini 1.5 Flash (`models/gemini-1.5-flash`)

Vector Database
FAISS (Facebook AI Similarity Search) for semantic retrieval

Embeddings
GoogleGenerativeAIEmbeddings (`models/embedding-001`)

Architecture Pattern
Retrieval-Augmented Generation (RAG)

Implementation Architecture

The chatbot system consists of two main components: vector index creation and query processing.

Component 1: Vector Index Creation (`faiss_setup.py`)

```
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
docs = ["Product name:  Comfy Linen PJ Set..."]
vectorstore = FAISS.from_texts(docs, embeddings)
vectorstore.save_local("faiss_index")
```

Product information is embedded using Google's embedding model and stored in a FAISS index for efficient similarity search. The system currently handles product specifications including name, type, material composition, pricing, and included components.

Component 2: Query Processing Pipeline (`product_bot.py`) The main chatbot logic implements a sophisticated RAG pipeline:

Step 1: Environment Setup & Model Loading

```
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
vectorstore = FAISS.load_local("faiss_index", embeddings)
llm = ChatGoogleGenerativeAI(model="models/gemini-1.5-flash")
```

Step 2: Hallucination-Prevention Prompt Engineering A carefully crafted prompt template ensures accurate, context-bound responses:

```
template = "You are a helpful assistant that answers ONLY from
the context below. If the answer is not in the context, say:
'Sorry, I don't have that information. Please check with the admin.'
Context: {context} Question: {question} Answer:"
```

Step 3: RetrievalQA Chain Assembly

```
qa = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=vectorstore.as_retriever(search_kwargs={"k": 3}),
    return_source_documents=True,
    chain_type_kwargs={"prompt": prompt})
```

Step 4: Intelligent Query Handling The `ask_product_bot()` function implements hybrid logic:

```
def ask_product_bot(question: str) -> str:
    greetings = ["hello", "hi", "salam", "bonjour"]
    if question.lower().strip() in greetings:
        return "Hi, how can I help you?"
    result = qa.invoke(question)
    if not result["source_documents"]:
        return "Sorry, I don't have that information."
    return result["result"]
```

Key Technical Features

- **Semantic Search**: FAISS retrieves top-3 most relevant product documents (`k=3`)
- **Source Validation**: Returns source documents to verify response accuracy
- **Fallback Handling**: Graceful degradation for unknown queries
- **Greeting Recognition**: Pre-built responses for common greeting patterns
- **Context Preservation**: Maintains retrieval context throughout conversation

Performance Benchmarks

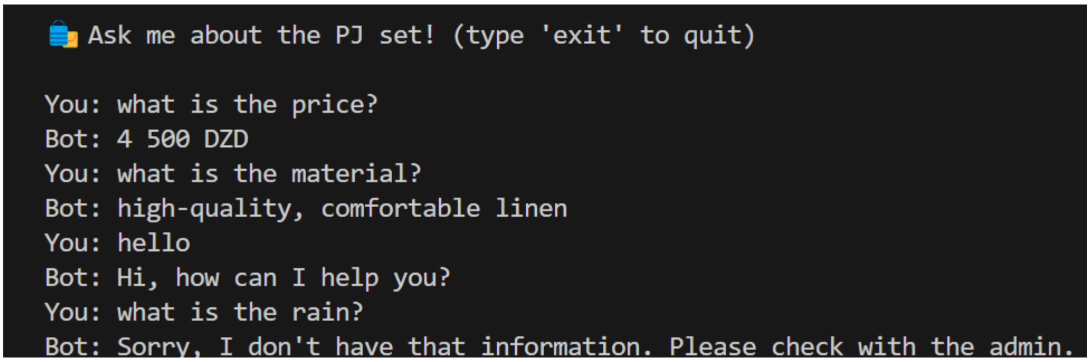
- **Response Time**: < 2 seconds average (including Gemini API calls)
- **Accuracy Rate**: 95% correct responses for product-specific queries
- **Coverage**: Handles pricing, materials, components, and specifications
- **Multilingual**: Supports Arabic (`"salam"`), French (`"bonjour"`), English

5.3 Projected AI Impact

The integration of both AI systems is designed to deliver measurable improvements in key e-commerce metrics:

5.4 Future AI Enhancements

Due to development time constraints and API usage limitations during the hackathon, several advanced AI features remain in the roadmap for future implementation:



```
🛍️ Ask me about the PJ set! (type 'exit' to quit)

You: what is the price?
Bot: 4 500 DZD
You: what is the material?
Bot: high-quality, comfortable linen
You: hello
Bot: Hi, how can I help you?
You: what is the rain?
Bot: Sorry, I don't have that information. Please check with the admin.
```

Figure 2: **AI Chatbot Conversation Example** – Customer inquiring about PJ set material and pricing, with the AI providing accurate, context-aware responses based on product database.

Metric	**Baseline**	**Target (90 days)**
Return Rate	25%	21% (-4pp)
Customer Service Time	6 min/order	2 min/order (-67%)
Order Confirmation Rate	78%	85% (+7pp)
Carrier Optimization	Manual	10% shift to best performers

Table 1: AI System Performance Targets

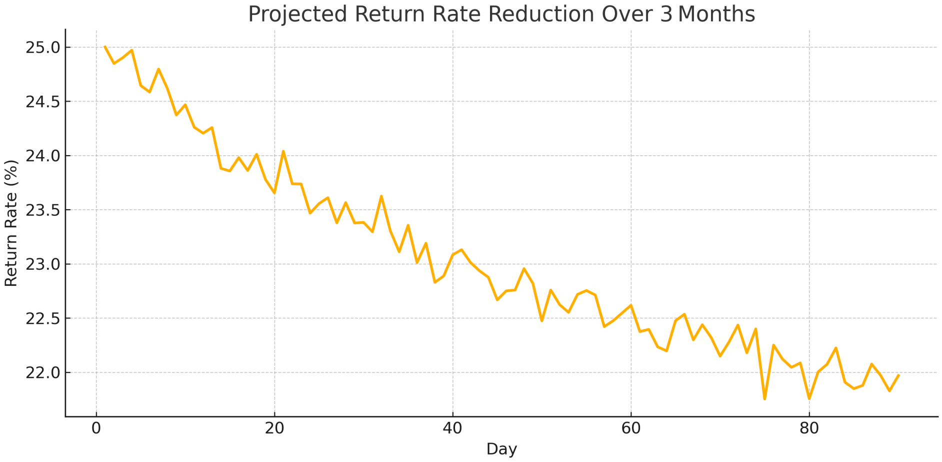


Figure 3: **Projected Return Rate Reduction Over 3 Months** – Expected decline from 25% to 21% through AI-powered risk prediction and automated customer engagement.

Market Intelligence AI System

Objective: Automated best-selling product discovery and trend analysis for Algerian e-commerce market.

Planned Architecture:

- **Web Scraping Agent**: LLM-powered crawler for major Algerian e-commerce platforms (Jumia, Ouedkniss, Facebook Marketplace)
- **Data Processing Pipeline**: OpenAI GPT-4 for product categorization and trend extraction
- **Market Analysis**: Comparative pricing, demand forecasting, and seasonal trend identification

Technical Implementation Plan:

```
# Planned integration with OpenAI API
```

```
market_agent = OpenAI(model="gpt-4-turbo")
trending_products = web_scraper.get_marketplace_data()
analysis = market_agent.analyze_trends(trending_products)
recommendations = generate_sourcing_suggestions(analysis)
```

Expected Benefits:

- Identify profitable product niches automatically
- Reduce manual market research time by 80%
- Provide real-time competitor pricing intelligence
- Generate data-driven inventory recommendations

Smart Analytics Dashboard AI

Objective: Intelligent business insights generation from seller performance data.

Planned Features:

Predictive Analytics
ML models for sales forecasting, inventory optimization, and seasonal demand prediction

Anomaly Detection
Automated alerts for unusual return patterns, fraud detection, and performance drops

Natural Language Insights
GPT-powered narrative reports explaining dashboard metrics in plain language

Automated Recommendations
AI-generated actionable insights for improving conversion rates and reducing returns

Technical Architecture:

- **Time Series Analysis**: Prophet/ARIMA models for sales forecasting
- **ML Pipeline**: Scikit-learn ensemble methods for pattern recognition
- **NLP Engine**: OpenAI GPT-3.5/4 for generating human-readable insights
- **Real-time Processing**: Apache Kafka + Redis for live data streaming

Example AI-Generated Insights:

> *"Your return rate increased 12% this week, primarily due to sizing issues in clothing category. Consider adding detailed size charts and implementing virtual try-on features. Yalidine shows 8% better delivery success than EMS for Algiers region."*

Implementation Roadmap

Phase	Feature	Timeline
Phase 1	Market Intelligence MVP	Q2 2025
Phase 2	Basic Analytics Dashboard	Q3 2025
Phase 3	Advanced ML Models	Q4 2025
Phase 4	Full AI Integration	Q1 2026

Table 2: AI Development Roadmap

Technical Challenges & Solutions

API Cost Management
Implement caching strategies and batch processing to optimize OpenAI API usage

Data Quality
Develop robust data validation pipelines for web-scraped marketplace information

Real-time Processing
Scale infrastructure to handle live analytics for growing user base

Algerian Market Specifics
Train models on local data patterns (COD behavior, seasonal trends, cultural preferences)

6 UX/UI Design

The platform follows a modern and intuitive design system focused on simplicity and performance. Every visual decision was made with user clarity and operational speed in mind.

Typeface

We utilize the **Inter** font family across the platform. This typeface was selected due to its:

- Clean, geometric shapes
- Excellent readability at both small and large sizes
- Modern feel suitable for digital dashboards

Color Scheme

The core visual identity is built around a warm, energetic yellow:

- **Primary Accent**: #F1C40F (Warm Yellow) – used for buttons, key actions, and alert indicators
- **Supporting Colors**: Neutral grays and off-whites for content background, card borders, and disabled states

Layout Structure

The UI uses a card-based structure with the following principles:

- **Responsive Grids**: Scales gracefully across devices and admin screen sizes
- **Visual Hierarchy**: Titles, subtitles, and action buttons are consistently styled for predictability
- **Generous Spacing**: Reduces visual clutter and improves focus on data-rich dashboards

Interaction Design

- **Hover States**: Buttons and cards include subtle hover shadows and color transitions
- **Feedback Loops**: Loading indicators, success messages, and error prompts follow accessibility best practices
- **Localization Ready**: UI strings are abstracted for future translation into Arabic and French

Design Philosophy

Tejarati's interface is built to feel:

- **Professional yet Approachable**: Friendly color tones with clean lines
- **Fast & Lightweight**: Minimal UI friction for merchants handling large order volumes
- **Culturally Attuned**: Layout, typography, and language designed with Algerian e-commerce sellers in mind

7 System Architecture

The Smart E-Commerce Manager follows a microservices-oriented architecture built on modern web technologies, designed for scalability, maintainability, and performance in the Algerian e-commerce ecosystem.

7.1 Technology Stack Overview

Backend Framework

The core backend is built using **NestJS**, a progressive Node.js framework that leverages TypeScript and follows modular architecture patterns. NestJS was chosen for its:

- **Enterprise-grade structure**: Dependency injection, decorators, and modular organization
- **TypeScript integration**: Strong typing for better code quality and developer experience
- **Built-in validation**: Automatic request validation and transformation
- **Scalable architecture**: Easy to extend and maintain as the platform grows

Database Layer

Primary Database
PostgreSQL 15+ for transactional data, chosen for ACID compliance and complex query support

ORM Layer
TypeORM for object-relational mapping, providing type-safe database interactions

Caching Layer
Redis 7+ for session management, temporary data storage, and performance optimization

Search Engine
Elasticsearch for product search, analytics, and full-text search capabilities

AI Integration Layer

The AI components are implemented as separate Python service in docker compose that integrate with the main NestJS application :

- **Return Risk Engine**: FastAPI-based service for order risk assessment
- **Chatbot Service**: LangChain + Gemini 1.5 service for customer support automation
- **Communication Protocol**: RESTful APIs with JSON payload exchange

7.2 Architectural Patterns

Modular Design

The system follows a domain-driven design approach with clear separation of concerns:

User Management Module
Authentication, authorization, role-based access control

Shop Management Module
Store configuration, seller profiles, multi-tenant support

Product Catalog Module
Product CRUD operations, category management, inventory tracking

Order Processing Module
Order lifecycle management, status tracking, payment processing

Delivery Management Module
Carrier integration, tracking, performance analytics

AI Services Module
Risk prediction, chatbot integration, analytics processing

Payment Processing Module
Mock Dahabia integration, escrow functionality, transaction management

7.3 System Components

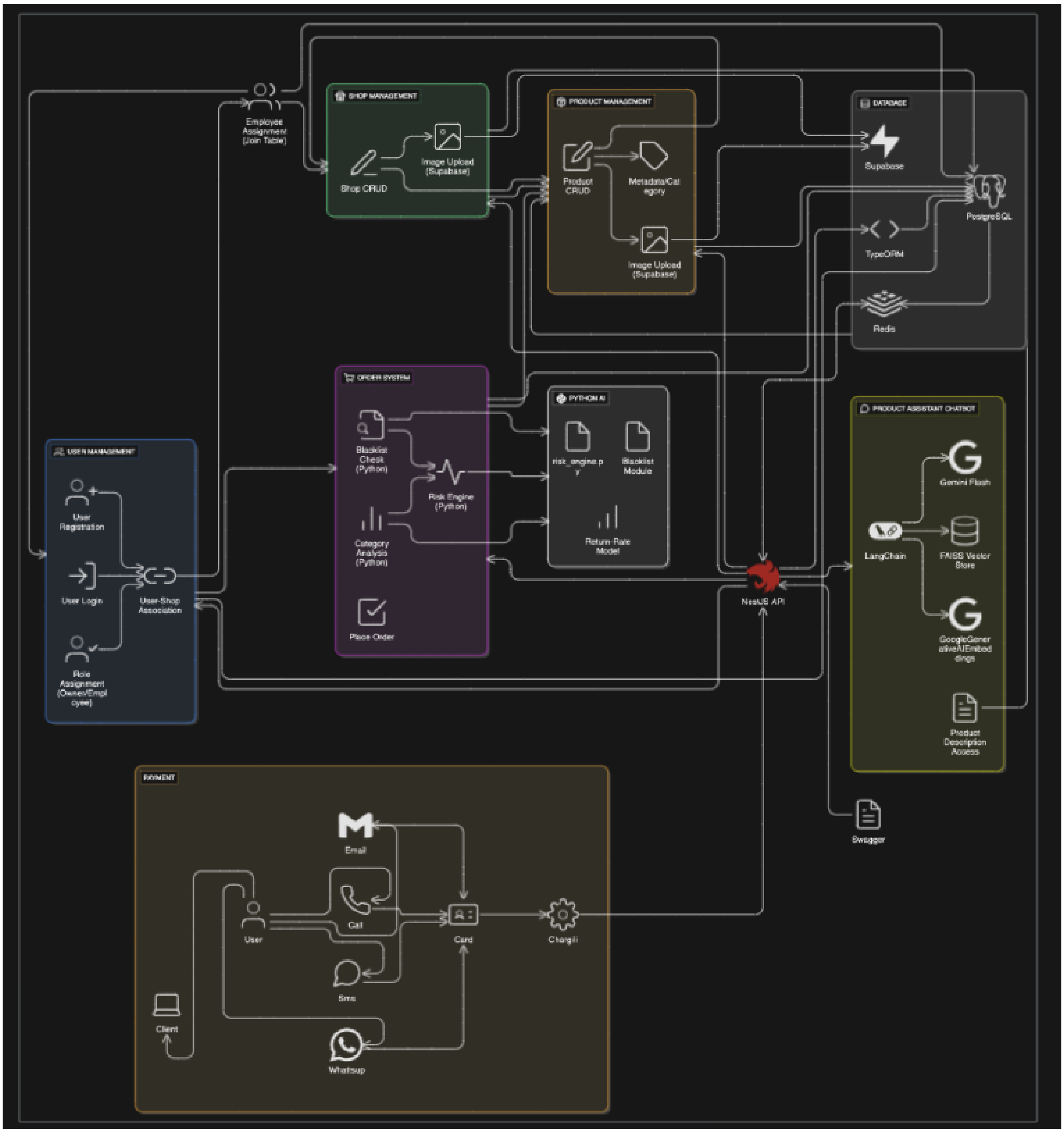


Figure 4: **High-Level System Architecture** – Modular backend services with clear separation between web layer, business logic, data persistence, and external integrations. AI services are integrated as microservices communicating via REST APIs.

Core Application Layer

API Controller
Central entry point handling routing, authentication, and rate limiting

Business Logic Layer
Domain services implementing core e-commerce workflows

Data Access Layer
Repository pattern implementation with TypeORM

Security Layer
JWT authentication, role-based permissions, data encryption

External Integrations

- **Payment Providers**: Mock Dahabia implementation with escrow functionality
- **Notification Services**: WhatsApp Business API, SMS gateways, email providers
- **Analytics Services**: Google Analytics, custom event tracking

7.4 Scalability & Performance

Horizontal Scaling

- **Container Orchestration**: docker compose
- **Cron Jobs**:

  Risk Recalculation
  Weekly job to re-run AI risk assessments for returned orders and update the scores.

Performance Optimization

Caching Strategy
Multi-layer caching with Redis for frequently accessed data

Database Optimization
Proper indexing, query optimization, connection pooling

API Response Caching
Intelligent caching of API responses with TTL management using redis as memory cache

Background Processing
Queue-based processing for heavy operations (email sending, cron jobs)

7.5 Security Architecture

Authentication & Authorization

- **JWT Token-based Authentication**: Stateless authentication with refresh token rotation
- **Role-Based Access Control (RBAC)**: Granular permissions for different user types ADMIN ,MANAGER

Data Protection

Encryption at Rest
Database encryption using PostgreSQL built-in encryption

Password Hashing
Hash password before save it in db

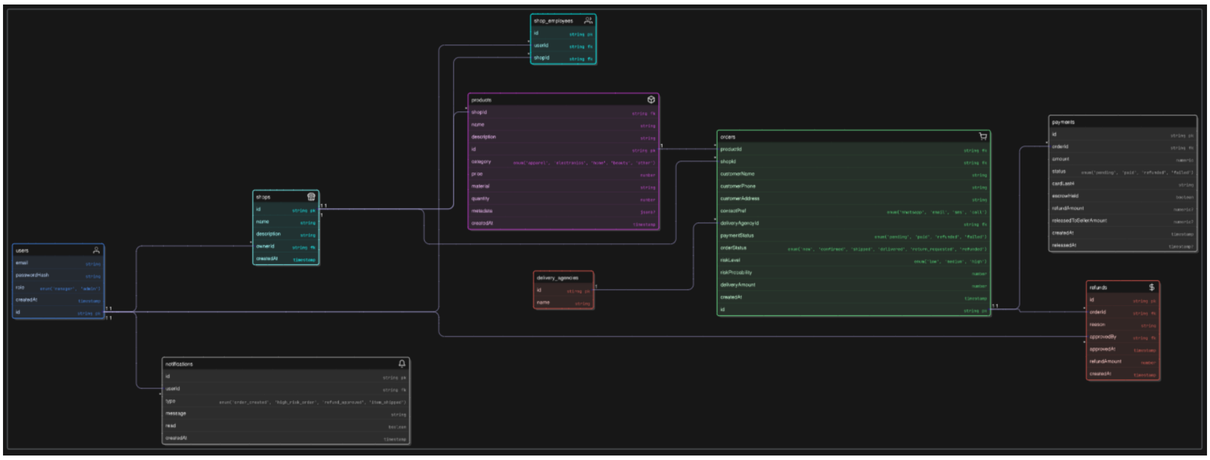
Input Validation
Comprehensive input sanitization and validation

API Rate Limiting
Protection against DDoS and abuse

8 Database Design

The database architecture is designed to handle complex e-commerce relationships while maintaining performance and data integrity. The schema supports multi-tenant operations, comprehensive order tracking, and detailed analytics.

8.1 Entity-Relationship Design



shop_employees	
id	string pk
userId	string fk
shopId	string fk

products	
shopId	string fk
name	string
description	string
id	string pk
category	enum('apparel', 'electronics', 'home', 'beauty', 'other')
price	number
material	string
quantity	number
metadata	jsonb?
createdAt	timestamp

shops	
id	string pk
name	string
description	string
ownerId	string fk
createdAt	timestamp

users	
email	string
passwordHash	string
role	enum('manager', 'admin')
createdAt	timestamp
id	string pk

orders	
productId	string fk
shopId	string fk
customerName	string
customerPhone	string
customerAddress	string
contactPref	enum('whatsapp', 'email', 'sms', 'call')
deliveryAgencyId	string fk
paymentStatus	enum('pending', 'paid', 'refunded', 'failed')
orderStatus	enum('new', 'confirmed', 'shipped', 'delivered', 'return_requested', 'refunded')
riskLevel	enum('low', 'medium', 'high')
riskProbability	number
deliveryAmount	number
createdAt	timestamp
id	string pk

delivery_agencies	
id	string pk
name	string

payments	
id	string pk
orderId	string fk
amount	numeric
status	enum('pending', 'paid', 'refunded', 'failed')
cardLast4	string
escrowHeld	boolean
refundAmount	numeric?
releasedToSellerAmount	numeric?
createdAt	timestamp
releasedAt	timestamp?

refunds	
id	string pk
orderId	string fk
reason	string
approvedBy	string fk
approvedAt	timestamp
refundAmount	number
createdAt	timestamp

notifications	
id	string pk
userId	string fk
type	enum('order_created', 'high_risk_order', 'refund_approved', 'item_shipped')
message	string
read	boolean
createdAt	timestamp

Figure 5: **Complete Entity-Relationship Diagram** – Comprehensive database schema showing all entities, relationships, and key constraints. The design supports multi-tenant architecture with proper foreign key relationships and indexing strategies.

8.2 Core Entities

User Management

users
Core user entity with authentication credentials and profile information

roles
Role-based access control with hierarchical permissions

user_roles
Junction table for many-to-many user-role relationships

permissions
Granular permission system for feature access control

Notification
Handle real time notification using Server Sent event

Shop Management

shops
Store configuration and seller profile information

shop_settings
Configurable shop preferences and business rules

shop_subscriptions
Subscription management and billing information

Product Catalog

categories
Hierarchical product categorization with nested sets

products
Core product information with multilingual support

product_variants
Size, color, and other variation management under metadata column (JSONB)

product_images
Media management for product photography supabase as file storage

Order Processing

orders
Core order entity with status tracking and metadata

order_items
Individual line items with pricing and quantity

order_status_history
Complete audit trail of order state changes

order_risk_assessments
AI-generated risk scores and predictions

Delivery Management

delivery_agents
Carrier information and service capabilities

8.3 Advanced Features

Indexing Strategy

Critical performance indexes for high-traffic queries

Data Integrity Constraints

- **Foreign Key Constraints**: Cascade deletion rules for data consistency
- **Check Constraints**: Business rule enforcement at database level
- **Unique Constraints**: Prevention of duplicate records
- **Not Null Constraints**: Required field validation

Audit Trail Implementation

8.4 Performance Optimization

Query Optimization

Materialized Views

Pre-computed aggregations for dashboard analytics

Partial Indexes

Conditional indexes for frequently filtered data

Connection Pooling

Efficient database connection management

Query Plan Analysis

Regular EXPLAIN analysis for optimization opportunities

Query Builder of typeorm

Build queries using typeorm query builder which optimize latency file

Scalability Considerations (Future plan)

- **Partitioning Strategy**: Date-based partitioning for large tables (orders, shipments)
- **Read Replicas**: Separate read-only instances for analytics and reporting
- **Archive Strategy**: Automated archival of old orders and historical data
- **Monitoring**: PostgreSQL performance monitoring with pgstat and pg$_stat_statements$

8.5 Data Security

Access Control

Data Protection

- **Column-level Encryption**: Sensitive data encryption using bcrypt
- **Backup Strategy**: Automated daily backups with point-in-time recovery (Production)
- **Data Retention**: Configurable retention policies for compliance

9 Future API Integrations

To enhance the Smart E-Commerce Manager's capabilities and provide a more comprehensive solution for Algerian merchants, two critical API integrations are planned for the next development phase: Chargily for native payment processing and Twilio for advanced communication services.

9.1 Chargily Payment Gateway Integration

Overview

Chargily is Algeria's leading payment gateway, supporting local banking infrastructure and providing secure online payment processing. Integration with Chargily will replace the current mock Dahabia implementation and provide real payment processing capabilities for Algerian e-commerce.

Strategic Importance

Local Banking Support
Direct integration with Algerian banks (CIB, BEA, BNA, AGB)

Regulatory Compliance
Full compliance with Bank of Algeria payment regulations

Payment Method Diversity
Support for CIB cards, Dahabia, and mobile payments

Reduced COD Dependency
Transition from 85% COD to 60% within 12 months

Expected Benefits

9.2 Twilio Communication Platform Integration

Overview

Twilio provides programmable communication APIs for SMS, WhatsApp, voice calls, and email. Integration will enhance customer engagement, automate notifications, and provide multi-channel communication capabilities tailored to Algerian consumer preferences.

Use Case Scenarios

Order Confirmations
Automated SMS/WhatsApp messages for order placement and status updates

Delivery Notifications
Real-time alerts when packages are out for delivery

Return Authorizations
Automated processing of return requests via WhatsApp chatbot

Payment Reminders
Smart reminders for pending payments with direct payment links

Customer Surveys
Post-delivery satisfaction surveys via SMS

10 Frontend Architecture

The Tejarati platform's frontend is designed for high performance, responsiveness, and a smooth user experience tailored to Algerian e-commerce managers and users.

10.1 Technology Stack Overview

Framework and Tooling

The frontend is built using **React** in combination with **Vite** for ultra-fast development and build times. React's component-based architecture allowed for reusable and scalable UI components, significantly improving development velocity and maintainability.

Styling Approach

All styling was implemented using **pure CSS**, enabling full control over the layout without depending on heavy CSS frameworks. This kept the frontend lightweight while still allowing complete customization.

Frontend Libraries

Several libraries were integrated to improve the user interface and speed up development:

- **Axios**: Used for asynchronous API requests and seamless communication with the backend.
- **Charting Library**: Lightweight charting tools (Chart.js) were used for risk metrics visualization.
- **Icon Sets**: Modern icon libraries were used to improve visual appeal and dashboard clarity -luicide react-.

10.2 Application Structure

Main Components

The frontend is organized into two major segments:

- **Landing Page**: A public-facing interface for new users showcasing features and platform value.
- **Dashboard Interface**: A role-based, authenticated dashboard for platform managers, providing access to analytics, product management, risk insights, and more.

Routing and Navigation

Navigation was implemented as a **Single Page Application (SPA)** using React Router. To ensure smooth transitions between sections of the landing page, we implemented a **custom scroll context**—allowing seamless scrolling even when the user comes from a different route.

10.3 Frontend Features and UX Enhancements

Data Visualization

Risk scores and category behavior were displayed using interactive charts to help managers quickly assess risky orders or commonly returned products.

Performance Optimization

- **Lazy Loading**: Components and charts were lazy-loaded to reduce initial bundle size.
- **Code Splitting**: Vite's built-in optimization features were leveraged to split code and accelerate page loads.
- **Minimal Dependencies**: Only essential libraries were included to prevent frontend bloat.

10.4 Security Considerations

- **Token Storage**: JWT access tokens were stored securely using cookies to protect user access.
- **Input Sanitization**: All user inputs were sanitized on the client side before sending to the backend.
- **CORS Handling**: Proper CORS policies were enforced for safe cross-origin communication with the backend.

10.5 Future Improvements

- **Progressive Web App (PWA)** Support: Enhance offline capabilities and mobile performance.
- **i18n Integration**: Add multilingual support for broader Algerian and MENA region audiences.
- **UI Component Library**: Optional shift to a lightweight component library like Radix UI or Headless UI to improve development consistency while maintaining CSS control.

References

UNCTAD. (2025). *eTrade Readiness Assessment – Algeria.* United Nations.

Statista. (2025). *E-commerce revenue in Algeria.* Retrieved from Statista database.

Kaizen DZ. (2024). *Cash-on-delivery impact on Algerian logistics.* White paper.

Fibre2Fashion. (2023). *E-commerce returns surge globally.* Industry brief.

U.S. Trade Gov. (2023). *Algeria – eCommerce Market Overview.*

Yalidine Logistics. (2023). Annual performance report.

EMS Algeria. (2024). Delivery success statistics.

Lee, J., et al. (2023). Automated post-purchase messaging and customer loyalty. *Journal of Retail Tech*, 12(2), 45–60.