CHAPTER FOUR

SYSTEM DESIGN AND IMPLEMENTATION

4.1 Objectives of the Design

The primary objective of this project is to design and implement an AI-driven Product Alert Management System with Predictive Analytics that enables organizations to proactively monitor, predict, and respond to product-related issues across the supply chain. The system addresses critical gaps in traditional reactive alert management by leveraging machine learning algorithms trained on historical product data, quality metrics, and supply chain patterns to forecast potential issues before they impact operations.

The specific design objectives are:

1. Proactive Alert Generation: To implement comprehensive predictive analytics capabilities that forecast product-related issues including stockouts, quality defects, expiration risks, and supply chain disruptions before they occur, enabling preventive action rather than reactive responses.
2. Intelligent Alert Prioritization: To develop machine learning models capable of automatically scoring and prioritizing alerts based on severity, business impact, likelihood, and confidence measures using Random Forest classifiers and time-series forecasting models.
3. Modular System Architecture: To design a layered architecture separating data ingestion, feature engineering, predictive analytics, alert generation, and user presentation into independent, maintainable components following object-oriented principles established in Chapter 3.
4. Accessibility and Usability: To create a web-based system accessible across devices with intuitive dashboards suitable for product managers, quality assurance teams, and inventory analysts, with simplified deployment using SQLite for development and PostgreSQL-ready architecture for production.

4.2 Control Centre/Main Menu

The Control Centre represents the system's primary user interface, implemented as a modern web-based dashboard providing centralized access to product monitoring, alert management, and predictive analytics capabilities. The interface follows contemporary design principles emphasizing clarity and intuitive navigation.

The dashboard comprises several functional panels:

Landing Page: Serves as the entry point with authentication controls for login and registration. Features dynamic content highlighting system capabilities including real-time monitoring, predictive analytics, and intelligent alert prioritization.

User Dashboard: The primary operational interface displaying real-time product monitoring status, active alerts organized by severity, and key performance indicators including total products under monitoring, active alerts by priority level, stock health distribution, and recent prediction confidence trends. Color-coded alert cards provide immediate visual feedback with critical alerts highlighted prominently.

Alert Management Panel: Displays comprehensive alert information with advanced filtering by category (quality, supply, demand, expiration, defects), severity level, date range, and status. Each alert shows title, category badge, severity indicator, confidence score, timestamp, and associated product information.

Product Inventory Overview: Presents all products under monitoring with real-time inventory levels, reorder point indicators, and stock health status. Supports search functionality by SKU, name, or manufacturer. Visual indicators show whether stock levels are critical, warning, or healthy.

Predictive Analytics Visualization: Displays forecasting results through interactive charts including demand forecast line charts, anomaly detection scatter plots, risk assessment distributions, and time-series trends. Chart controls allow time horizon adjustments and data export.

Fig. 4.1: System Landing Page

Fig. 4.2: User Dashboard Overview

4.3 The Submenus/Subsystems

The system comprises several interconnected subsystems following OOADM principles with clearly defined interfaces:

1. Data Ingestion and Processing Subsystem: Implements data collection from ERP systems, IoT sensors, quality databases, and supplier feeds. Supports real-time streaming and batch processing with validation, cleansing, transformation, and synchronization. Implements ETL pipelines aggregating data into the centralized warehouse.
2. Feature Engineering Subsystem: Transforms raw data into machine learning features through time-series extraction (lag features, rolling statistics, seasonality), quality metric aggregation (defect rates, trends), supplier performance calculation (reliability scores, lead time variability), and inventory health indicators (turnover rates, stockout risk). Outputs 30-40 normalized features.
3. Machine Learning and Predictive Analytics Subsystem: Comprises specialized components:
4. Anomaly Detection Engine using Isolation Forest algorithms
5. Risk Classification System employing Random Forest classifiers
6. Alert Generation and Decision Engine with composite scoring, SLA management, escalation logic, and deduplication
7. Database Management Subsystem: Handles persistent storage using SQLite for development with PostgreSQL capability for production. Implements SQLAlchemy ORM with transactional operations, query optimization, and backup procedures.
8. Web API Subsystem: Implements RESTful endpoints using FastAPI with automatic documentation, authentication enforcement, and multiple endpoints for auth, products, alerts, predictions, and health monitoring.
9. Notification and Escalation Subsystem: Manages multi-channel alert delivery through email, SMS, and in-app notifications with configurable rules, escalation workflows, and comprehensive logging.
10. User Management and Authentication Subsystem: Provides JWT-based authentication with OAuth2 password grant flow, password hashing using bcrypt, token validation, role-based access control (admin, manager, analyst, operator), and session management.

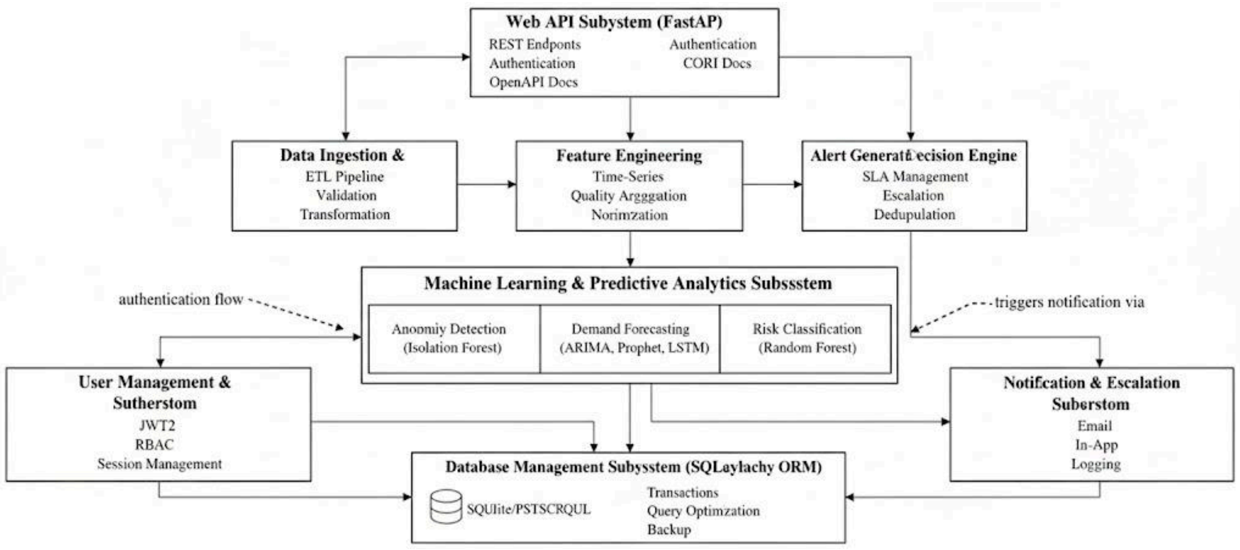
Web API Subystem (FastAP)
REST Endponts
Authentication
OpenAPI Docs
Authentication
CORI Docs
Data Ingestion &
ETL Pipeline
Validation
Transformation
Feature Engineering
Time-Series
Quality Argggation
Norimzation
Alert GeneratEecision Engine
SLA Management
Escalation
Dedupulation
Machine Learning & Predictive Analytics Subssstem
Anoomiy Detection (Isolation Forest)
Demand Forecasting (ARIMA, Prophet, LSTM)
Risk Classification (Random Forest)
authentication flow
triggers notification via
User Management & Sutherstom
JWT2
RBAC
Session Management
Notification & Escalation Suberstom
Email
In-App
Logging
Database Management Subysstem (SQLaylachy ORM)
SQUlite/PSTSCRQUL
Transactions
Query Optimzation
Backup

Fig. 4.3: System Subsystem Architecture

4.4 System Specifications

4.4.1 Database Development Tool

The system employs SQLite as the database management system for development, providing a lightweight, file-based solution requiring minimal configuration. Database interactions are managed through SQLAlchemy ORM, a Python toolkit providing declarative schema definitions, type-safe query builders, and database-agnostic abstractions facilitating future migration to PostgreSQL for production deployments.

4.4.2 Database Design and Structure

The database schema comprises fifteen interconnected tables designed to support comprehensive product monitoring, alert management, and predictive analytics:

Table 4.1: users

Field	Type	Description
user_id	String (UUID)	Unique user identifier
username	String (Unique)	User login name
email	String (Unique)	User email address
hashed_password	String	Bcrypt hashed password

full_name	String	User's full name
role	Enum	User role: admin, manager, analyst, operator
notification_preferences	JSON	Communication channel preferences
created_at	DateTime	Account creation timestamp
is_active	Boolean	Account status flag

Table 4.2: products

Field	Type	Description
product_id	String (UUID)	Unique product identifier
sku	String (Unique)	Stock keeping unit code
name	String	Product name
category	String	Product category
manufacturer	String	Manufacturer name
current_stock	Integer	Current inventory quantity
reorder_point	Integer	Reorder threshold
safety_stock	Integer	Minimum safety stock level
unit_price	Float	Product unit cost
last_updated	DateTime	Last inventory update time
metadata	JSON	Additional flexible attributes

Table 4.3: alerts

Field	Type	Description
alert_id	String (UUID)	Unique alert identifier
product_id	String (Foreign Key)	Associated product
title	String	Alert title/summary
category	Enum	Category: quality, supply, demand, expiration, defects
severity	Enum	Severity: critical, high, medium, low
confidence_score	Float	Prediction confidence (0.0-1.0)
alert_score	Float	Composite priority score
priority_level	String	Priority: P1, P2, P3, P4
sla_deadline	DateTime	Response deadline
status	Enum	Status: open, acknowledged, resolved, closed
created_at	DateTime	Alert creation timestamp
assigned_to	String (Foreign Key)	Assigned user ID
fingerprint	String	Deduplication hash

Table 4.4: predictions

Field	Type	Description
prediction_id	String (UUID)	Unique prediction identifier
product_id	String (Foreign Key)	Associated product

prediction_type	Enum	Type: demand_forecast, risk_assessment, anomaly
model_version	String	Model identifier used
prediction_date	DateTime	Prediction generation time
forecast_horizon	Integer	Days ahead for forecast
predicted_value	Float	Forecasted quantity or score
confidence_interval_lower	Float	Lower confidence bound
confidence_interval_upper	Float	Upper confidence bound
confidence_score	Float	Prediction confidence
risk_level	Enum	Risk: low, medium, high, critical

Table 4.5: inventory_metrics

Field	Type	Description
metric_id	String (UUID)	Unique metric identifier
product_id	String (Foreign Key)	Associated product
timestamp	DateTime	Metric observation time
stock_quantity	Integer	Inventory count at timestamp
demand_quantity	Integer	Demand observed
turnover_rate	Float	Inventory turnover metric
days_of_inventory	Integer	Days until stockout

stockout_risk_score	Float	Computed stockout probability

Table 4.6: quality_metrics

Field	Type	Description
metric_id	String (UUID)	Unique metric identifier
product_id	String (Foreign Key)	Associated product
batch_number	String	Manufacturing batch identifier
inspection_date	DateTime	Quality inspection timestamp
defect_count	Integer	Number of defects found
defect_rate	Float	Defects per unit inspected
passed_inspection	Boolean	Overall pass/fail status

Table 4.7: supplier_risk

Field	Type	Description
risk_id	String (UUID)	Unique risk record identifier
supplier_name	String	Supplier organization name
product_id	String (Foreign Key)	Associated product
assessment_date	DateTime	Risk assessment timestamp
reliability_score	Float	Performance score (0-100)
lead_time_avg	Integer	Average delivery time in days

overall_risk_level	Enum	Overall: low, medium, high, critical

Table 4.8: workflow_cases

Field	Type	Description
case_id	String (UUID)	Unique case identifier
case_number	String (Unique)	Human-readable case number
alert_id	String (Foreign Key)	Originating alert
priority	String	Priority level: P1, P2, P3, P4
status	Enum	Status: open, investigating, resolved, closed
assigned_to	String (Foreign Key)	Current assignee user ID
created_at	DateTime	Case creation timestamp
resolution_summary	Text	Resolution description

Additional supporting tables include notifications, model_performance, alert_history, demand_patterns, expiration_tracking, system_logs, and feedback_loops with similar structured designs supporting comprehensive system functionality.

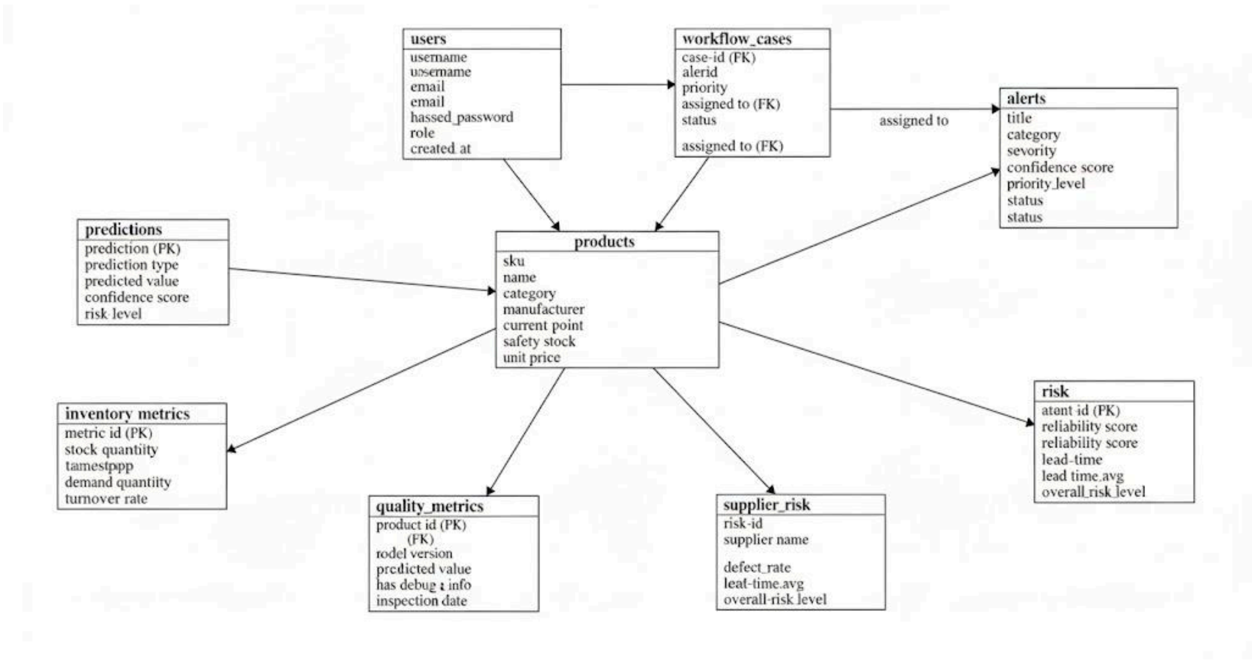

Fig. 4.4: Database Schema ERD

4.4.3 Mathematical Specification

The system employs several mathematical formulations:

Shannon Entropy for anomaly detection: $H(X) = -\Sigma P(x_i) \log_2 P(x_i)$

where $P(x_i)$ represents the probability of observing demand level x_i. High entropy indicates erratic, unpredictable patterns.

Random Forest Classification with majority voting: $\hat{y} = \text{mode}\{h_1(x), h_2(x), ..., h_{□}(x)\}$

where h_i represents individual decision trees and x is the feature vector.

Composite Alert Score: Alert_Score = w_1·Severity + w_2·Likelihood + w_3·Impact + w_4·Confidence

with default weights (0.35, 0.25, 0.30, 0.10) emphasizing severity and business impact.

Isolation Forest Anomaly Score: s(x,n) = 2^(-E(h(x))/c(n))

where scores approaching 1.0 indicate anomalies.

4.4.4 Program Module Specification

Data Ingestion Module: Handles inputs from ERP systems, quality databases, and supplier platforms with validation, sanitization, and rate limiting.

Feature Engineering Module: Transforms raw data through time-series extraction, quality aggregation, supplier performance calculation, and inventory health features with standardization.

Predictive Analytics Module: Orchestrates ML inference with model loading, preprocessing, prediction generation, confidence computation, and result formatting.

Alert Generation Module: Processes predictions with condition evaluation, composite scoring, severity assignment, priority classification, SLA deadline calculation, fingerprint generation, and enrichment.

Output Module: Formats outputs as JSON responses, HTML emails, CSV exports, and dashboard aggregations with response caching.

4.4.5 Input/Output Format Input Specifications:

Product Data Input (JSON):

```
POST /api/products
{
  "sku": "PROD-2024-001",
  "name": "Medical Antibiotics",
  "category": "Pharmaceuticals",
  "current_stock": 500,
  "reorder_point": 150,
  "safety_stock": 50
}
```

Authentication Input (OAuth2):

```
POST /api/auth/login
username=analyst_user&password=secure_password
```

Output Specifications:

Alert Response:

```
{
  "alert_id": "uuid-string",
  "title": "Critical Stock Level",
  "severity": "critical",
  "confidence_score": 0.92,
  "alert_score": 78.5,
  "priority_level": "P1",
  "sla_deadline": "2025-10-08T18:30:00Z",
  "recommendations": ["Initiate emergency procurement"]
}
```

Prediction Response:

```
{
  "prediction_type": "demand_forecast",
  "forecast_horizon": 30,
  "predictions": [{
    "date": "2025-10-09",
    "predicted_demand": 45,
    "confidence_interval": [38, 52]
  }]
}
```

Fig. 4.5: Alert Dashboard Display

4.4.6 Algorithm

Algorithm 1: Feature Extraction from Product Data

Input: product_id, time_window (days)

Output: feature_vector

```
1. Retrieve product record by product_id
2. Extract basic features (current_stock, reorder_point, safety_stock)
3. Query inventory_metrics within time_window:
   - Compute demand statistics (mean, std, trend)
   - Compute supply statistics (mean, reliability)
   - Calculate turnover_rate, days_of_inventory
4. Query quality_metrics:
   - Compute defect_rate, quality_trend
5. Query supplier_risk:
   - Extract reliability_score, lead_time_avg
6. Generate time-series features (lag_7d, lag_14d, rolling_mean_7d)
7. Extract seasonality (day_of_week, month, holiday_proximity)
8. Normalize numerical features to [0,1]
9. Encode categorical features
10. Return feature_vector
```

```
Algorithm 2: Alert Generation and Prioritization
Input: product_id, prediction_results, anomaly_results
Output: alert, alert_score

1. Determine alert conditions (stockout, quality, expiration, anomaly)
2. For each triggered condition, create alert instance
3. Calculate severity score (0-100) based on conditions
4. Calculate likelihood from prediction confidence
5. Calculate business impact from revenue_at_risk
6. Extract confidence from predictions
7. Compute alert_score = (0.35×Severity + 0.25×Likelihood + 0.30×Impact + 0.10×Confidence)
8. Assign severity level (critical/high/medium/low)
9. Assign priority (P1/P2/P3/P4) and calculate SLA deadline
10. Generate fingerprint for deduplication
11. Check for duplicate alerts and merge if exists
12. Enrich with recommendations
13. Assign to appropriate user
14. Save alert and log creation
15. Return (alert, alert_score)
```

```
Algorithm 3: Demand Forecasting with Ensemble Models
Input: product_id, forecast_horizon
Output: forecast_results

1. Retrieve historical demand data (365 days)
2. Preprocess time series (handle missing values, remove outliers)
```

3. Generate ARIMA forecast with confidence intervals
4. Generate Prophet forecast with seasonality
5. Generate LSTM forecast with prediction variance
6. Ensemble predictions: weighted_avg = $(0.3\times\text{ARIMA} + 0.4\times\text{Prophet} + 0.3\times\text{LSTM})$
7. Compute ensemble confidence intervals
8. Calculate confidence scores from interval widths
9. Format results with date, value, intervals, confidence
10. Return forecast_results

4.4.7 Data Dictionary

Field	Type	Description	Example
product_id	UUID	Unique product identifier	"a7f3c8d2-..."
sku	String	Stock keeping unit code	"PROD-2024-001"
alert_id	UUID	Unique alert identifier	"b8e4d3f1-..."
severity	Enum	Alert severity level	"critical"
confidence_score	Float	Prediction confidence (0-1)	0.87
alert_score	Float	Composite priority score	78.5
priority_level	String	Priority classification	"P1"
prediction_type	Enum	Prediction category	"demand_forecast"
forecast_horizon	Integer	Days ahead for forecast	30

risk_level	Enum	Risk classification	"medium"
defect_rate	Float	Defects per unit	0.023
reliability_score	Float	Supplier performance	87.5
turnover_rate	Float	Inventory turnover	8.5
stockout_risk_score	Float	Stockout probability	0.15

4.5 System Flowchart

The system flowchart illustrates the complete analysis workflow from data ingestion through alert generation and notification The flowchart demonstrates parallel processing with feature extraction, anomaly detection, and forecasting occurring concurrently. Error handling is incorporated at critical stages with a caching layer optimizing performance.

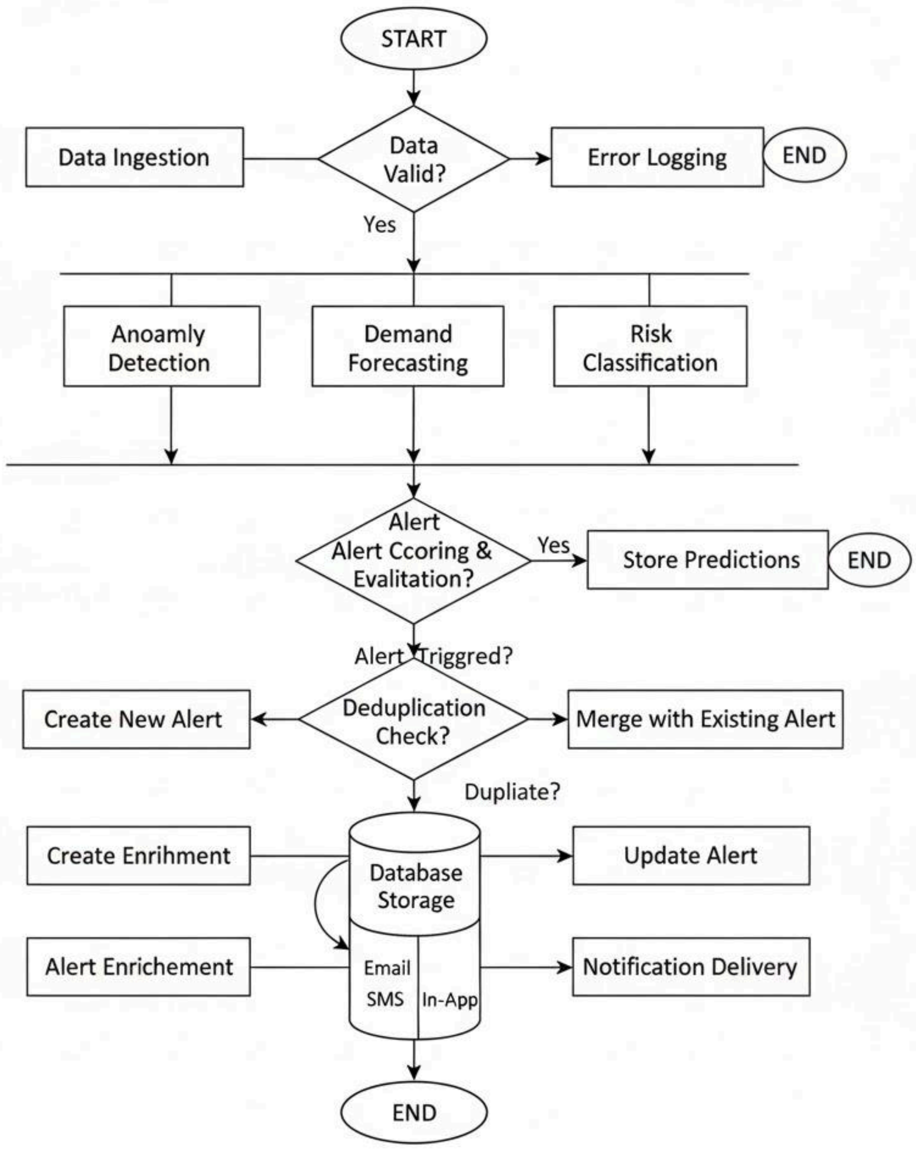

Fig. 4.6: System Flowchart

4.6 Object Diagrams

The object diagram illustrates runtime relationships between key entities during alert generation. The diagram demonstrates object-oriented design principles with Product, FeatureExtractor, PredictiveModel, AlertGenerator, and NotificationService objects maintaining distinct responsibilities with clear interfaces.

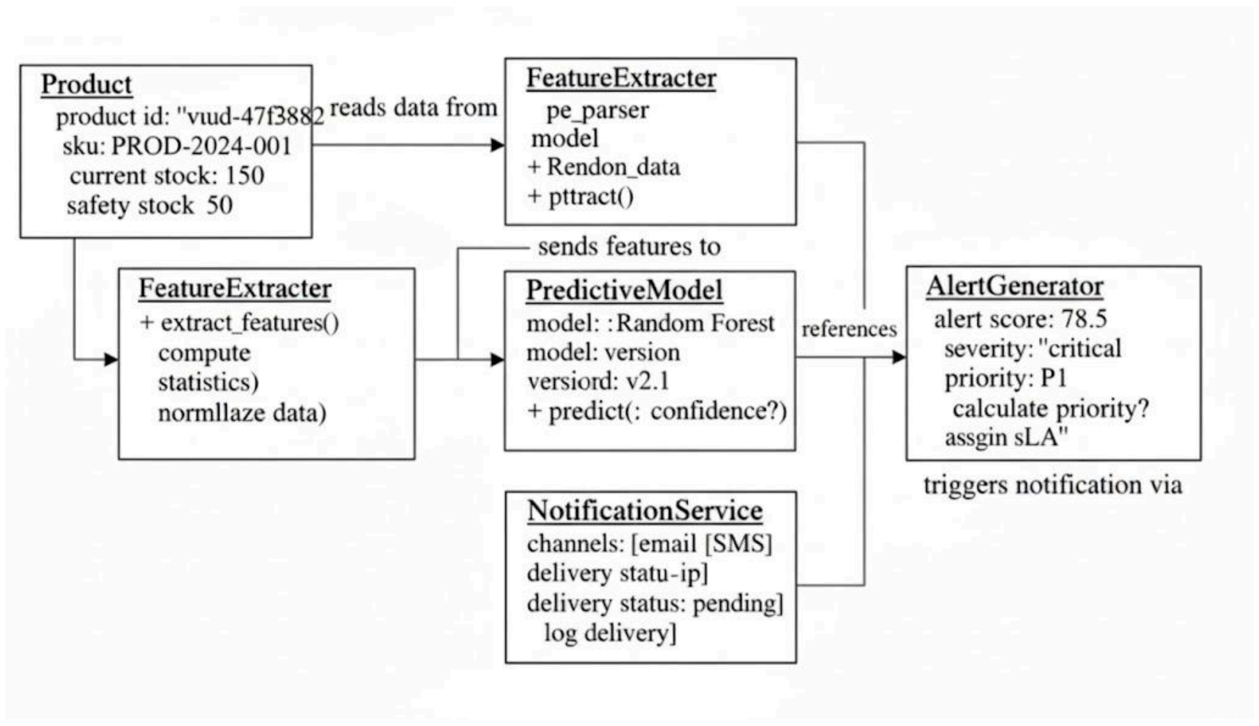

Fig. 4.7: Object Diagram

4.7 System Implementation

4.7.1 Proposed System Requirements

4.7.1.1 Hardware Requirements Development Environment:

Processor: Intel Core i5 (10th gen) or AMD Ryzen 5 equivalent

Memory: Minimum 8GB RAM, recommended 16GB

Storage: 100GB available disk space

Display: 1920×1080 resolution or higher

Deployment Environment:

Server with 4+ CPU cores

16GB+ RAM

SSD storage

Network bandwidth for concurrent connections

4.7.1.2 Software Requirements Core Stack:

Python 3.10+, Node.js 18+, SQLite 3.35+, Git, Docker (optional)

Python Dependencies:

fastapi==0.104.1, sqlalchemy==2.0.23, scikit-learn==1.3.2

pandas==2.1.3, numpy==1.26.2, prophet==1.1.5

python-jose==3.3.0, passlib==1.7.4, joblib==1.3.2
Frontend Dependencies:
next==14.0.3, react==18.2.0, typescript==5.3.2
tailwindcss==3.3.5, recharts==2.10.3, axios==1.6.2

4.7.2 Program Development

4.7.2.1 Choice of Programming Environment

Backend: Python with FastAPI for high-performance async APIs, scikit-learn for ML, Prophet for forecasting, SQLAlchemy for database abstraction.
Frontend: Next.js 14 with React 18 and TypeScript for type-safe component development, Tailwind CSS for styling, Recharts for visualizations.
Database: SQLAlchemy ORM with SQLite for development, PostgreSQL-ready for production with Alembic migrations.
ML Pipeline: scikit-learn Pipeline with joblib serialization for model persistence and fast inference.
Authentication: JWT with OAuth2, python-jose for token handling, passlib with bcrypt for password hashing.
Version Control: Git with structured branching and conventional commits.

4.7.3 Training

User Training: Quick start guide, feature interpretation guide, dashboard navigation tutorials with annotated screenshots and video demonstrations.
Administrator Training: Model retraining procedures, system maintenance guides, user management procedures, troubleshooting handbooks.
Educational Materials: Theoretical foundations, laboratory exercises, and case studies for academic integration.

4.7.4 Documentation

User Documentation: Comprehensive user manual, FAQ document covering common questions.
Technical Documentation: API reference with OpenAPI specs, architecture documentation, database schema reference with ERD.
Development Documentation: Setup instructions, code organization overview, testing guide.

Research Documentation: Experiment reproduction guide, model performance documentation with training records.

4.7.5 System Conversion

4.7.5.1 Changeover Procedures

Direct Conversion: Immediate replacement on cutover date. Minimal transition duration but no fallback.

Parallel Conversion: Simultaneous operation of old and new systems for validation (30-90 days). Conservative approach enabling confidence building.

Pilot Conversion: Gradual phased rollout starting with limited scope, progressively expanding.

Phased Conversion: Staged capability deployment (Phase 1: basic alerts, Phase 2: forecasting, Phase 3: quality analytics, Phase 4: full predictive).

4.7.5.2 Recommended Procedure

Parallel conversion recommended for 60 days providing validation through empirical comparison, stakeholder confidence building, risk mitigation with fallback options, user familiarization period, iterative improvement based on feedback, and thesis artifact generation.