ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

**FINAL YEAR PROJECT REPORT**

# *ERPConnect Project*

---

Intelligent Decision Support Module Integrated with ERP

---

**Program:** Computer Science and Networks Engineering

**Realized by:**
Ismail Moustatraf & Ibrahim Hanna

**Academic Supervisor:**
Abderrahim Larhlimi

**Academic Year: 2025/2026**

# Acknowledgments

We would like to express our deepest gratitude to our academic supervisor, **Abderrahim Larhlimi**, for his invaluable guidance, support, and constructive feedback throughout the realization of this project. His expertise and encouragement have been instrumental in our professional and technical growth.

We also extend our thanks to the staff of **EMSI** for providing us with the necessary academic environment and tools to develop our skills.

Finally, we are grateful to our families and colleagues for their constant support and understanding during the rigorous phases of this Final Year Project.

# Résumé

Le projet **ERPConnect** vise à transformer les données opérationnelles des systèmes ERP (Enterprise Resource Planning) en actifs stratégiques grâce à l'intelligence artificielle. Ce rapport documente le développement d'un module d'aide à la décision conçu pour s'intégrer aux ERP open source tels qu'Odoo. La solution s'appuie sur une architecture de micro-services basée sur **FastAPI** pour fournir des fonctionnalités avancées d'analytique prédictive.

Les principales contributions incluent un système de prévision de la demande basé sur l'apprentissage automatique (Polynomial Regression), un mécanisme de détection d'anomalies par score-Z, et un moteur de recommandation de réapprovisionnement optimisé par des algorithmes de gestion de stock (ROP). Une interface utilisateur en **React** permet aux décideurs de visualiser plus de 30 indicateurs de performance (KPI) et de personnaliser leurs tableaux de bord. Les résultats démontrent l'efficacité de l'approche pour réduire les ruptures de stock et valoriser le patrimoine informationnel de l'entreprise.

**Mots-clés :** ERP, Intelligence Artificielle, Prévision de la demande, Apprentissage automatique, FastAPI, Aide à la décision.

# Abstract

The **ERPConnect** project aims to transform operational data from ERP (Enterprise Resource Planning) systems into strategic assets through Artificial Intelligence. This report documents the development of a decision support module designed to integrate with open-source ERPs such as Odoo. The solution relies on a **FastAPI**-based microservices architecture to provide advanced predictive analytics features.

Key contributions include a machine-learning-based demand forecasting system (Polynomial Regression), a Z-score anomaly detection mechanism, and a replenishment recommendation engine optimized by inventory management algorithms (ROP). A **React** user interface allows decision-makers to visualize over 30 Key Performance Indicators (KPIs) and customize their dashboards. The results demonstrate the effectiveness of the approach in reducing stock-outs and maximizing the value of the company's informational assets.

**Keywords:** ERP, Artificial Intelligence, Demand Forecasting, Machine Learning, FastAPI, Decision Support.

# Contents

# List of Figures

# List of Tables

# General Introduction

# Chapter 1

# Project Framework Presentation

## Chapter Objectives

## 1.1 Existing System Analysis

### 1.1.1 Description of Existing Solutions

- Sales reports aggregated by period, product, or customer;
- Inventory status with manually configurable alert thresholds;
- Commercial performance tracking charts;
- Data exports in CSV or Excel format for external analysis;
- Predefined performance indicators (revenue, margin, inventory turnover).

# Chapter 2

# Requirements Specification

## Chapter Objectives

This chapter establishes a comprehensive specification of ERPConnect's requirements based on the actual implemented system. We identify all system actors, detail functional and non-functional requirements as realized in the codebase, and present use case documentation reflecting real API endpoints and workflows. This specification serves as accurate documentation of what the system does and how it performs.

## 2.1 Introduction

Requirements specification transforms business needs into precise technical specifications. For ERPConnect, this phase documents the actual capabilities implemented in our intelligent decision support module that integrates with Odoo ERP systems.

Our requirements are derived from the implemented system architecture, which includes a FastAPI backend communicating with Odoo via XML-RPC, machine learning forecasting capabilities using scikit-learn, intelligent alerting mechanisms, and a comprehensive KPI dashboard. We distinguish between functional requirements (what the system does) and non-functional requirements (how well it performs).

This chapter documents the real actors who interact with ERPConnect, the actual use cases implemented as REST API endpoints, and the quality attributes achieved through our technical choices.

## 2.2 Functional Requirements Specification

Functional requirements define the core capabilities that ERPConnect provides. These are organized hierarchically according to the implemented service modules.

### 2.2.1 ERP Data Integration and Synchronization

#### Establish Odoo Connection via XML-RPC

The system shall establish and maintain connections to Odoo ERP systems using the XML-RPC protocol. Connection parameters include:

- Odoo server URL
- Database name
- Username and password credentials
- Authentication via `/xmlrpc/2/common` endpoint

**Implementation:** `OdooConnector` class in `app/services/odoo_connector.py`

#### Extract Product Data

The system shall retrieve product information from Odoo including:

- Product ID, name, and reference code
- Current stock quantity (`qty_available`)
- Forecasted quantity (`virtual_available`)
- Unit price (`list_price`)

**Implementation:** `search_read` on `product.product` model

#### Extract Sales Data

The system shall retrieve sales order lines with filtering by date range, including:

- Product reference
- Quantity ordered
- Unit price
- Creation date

**Implementation:** `search_read` on `sale.order.line` model

#### Extract Inventory Movements

The system shall access stock movement history including receipts and deliveries from `stock.move` model.

### 2.2.2 Demand Prediction

#### Calculate 30-Day Demand Forecast

The system shall predict 30-day demand for each product using the following methodology:

1. Fetch sales order lines for configurable lookback period (default: 60 days)
2. Aggregate daily sales quantities
3. Apply 7-day moving average smoothing to reduce noise
4. Multiply current daily demand by 30 to obtain forecast

**Algorithm:** Moving average smoothing
**Implementation:** `predict_demand()` in `app/services/ai/demand.py`
**Endpoint:** `GET /ai/demand?lookback_days=60&limit=200`

#### Detect Demand Trends

The system shall classify demand trends as UP, DOWN, or STABLE by calculating the slope between early and late smoothed time windows:

- Trend = UP if slope >0.2
- Trend = DOWN if slope <-0.2
- Trend = STABLE otherwise

#### Calculate Confidence Scores

The system shall compute confidence scores (0.0 to 1.0) based on:

- Data coverage (days of historical data available)
- Demand variability (coefficient of variation)
- Sample count (number of sales transactions)

**Formula:** $confidence = 0.4 \times coverage + 0.3 \times (1 - variability) + 0.3 \times sample\_boost$

### 2.2.3 Machine Learning Forecasting

#### Train Linear Regression Models

The system shall train product-specific forecast models using:

- **Algorithm:** scikit-learn LinearRegression

- **Feature Engineering:** PolynomialFeatures with degree=2
- **Training Data:** Smoothed daily sales (7-day moving average)
- **Lookback Period:** Configurable (default: 180 days, minimum: 14 days)

**Implementation:** `train_product_model()` in `app/services/ai/ml_forecast.py`
**Model Persistence:** joblib serialization to `models/forecasts/product_{id}.joblib`

### Generate Multi-Horizon Forecasts

The system shall produce forecasts for configurable time horizons (default: 30 days) with:

- Daily forecast array
- Aggregated totals: 7-day, 30-day, 90-day
- Confidence intervals (95% level) based on residual standard deviation: $CI = \hat{y} \pm 1.96\sigma$

**Endpoint:** `GET /ai/forecast/{product_id}?horizon_days=30`

### Compute Forecast Accuracy Metrics

The system shall calculate and store:

- **MAE** (Mean Absolute Error): $\frac{1}{n}\sum|y_i - \hat{y}_i|$
- **MAPE** (Mean Absolute Percentage Error): $\frac{100\%}{n}\sum\frac{|y_i-\hat{y}_i|}{y_i}$
- **sMAPE** (Symmetric MAPE): $\frac{200\%}{n}\sum\frac{|y_i-\hat{y}_i|}{|y_i|+|\hat{y}_i|}$

**Note:** sMAPE is preferred as it handles zero values robustly.

### Automated Nightly Training

The system shall automatically train ML models for the top 50 products (by sales volume) via APScheduler at configurable intervals (default: nightly).

**Implementation:** `app/tasks/scheduler.py`

## 2.2.4 Inventory Replenishment Recommendations

### Calculate Reorder Point (ROP)

The system shall compute ROP using the formula:

$$ROP = avg\_daily\_sales \times lead\_time\_days + avg\_daily\_sales \times safety\_stock\_days$$

**Parameters:**

- `default_lead_time_days`: Configurable (default: 7)
- `safety_stock_days`: Configurable (default: 3)

**Generate Replenishment Suggestions**

The system shall recommend order quantities:

$$suggested\_qty = \max(0, ROP - current\_stock)$$

**Action Categories:**

- **ORDER NOW**: Current stock below ROP and suggested qty >0
- **MONITOR**: Stock cover between 7-14 days
- **OK**: Sufficient stock

**Endpoint:** `GET /ai/replenishment/with_rop`
**Implementation:** `app/routers/ai.py::replenishment_with_rop()`

### 2.2.5 Product Segmentation (ABC/XYZ Analysis)

**ABC Classification by Revenue**

The system shall segment products into three categories based on revenue contribution:

- **A-class**: Top 20% by cumulative revenue
- **B-class**: Next 30% (20-50 percentile)
- **C-class**: Remaining 50%

**XYZ Classification by Variability**

The system shall classify products by demand variability using coefficient of variation ($CV = \sigma/\mu$):

- **X-class**: Low variability (CV <0.3)
- **Y-class**: Medium variability (0.3 $\leq$ CV <0.7)
- **Z-class**: High variability (CV $\geq$ 0.7)

#### Generate Segmentation Matrix

The system shall produce a 3×3 matrix showing product counts in each ABC-XYZ segment with strategy recommendations (e.g., "Focus on AX items for tight inventory control").

**Endpoint:** `GET /ai/segmentation?days=60`
**Implementation:** `segment_abc_xyz()` in `app/services/ai/segmentation.py`

### 2.2.6 Sales Anomaly Detection

#### Detect Statistical Anomalies

The system shall identify sales spikes and drops using z-score methodology:

$$z = \frac{x - \mu}{\sigma}$$

- Flag anomaly if $|z| \geq threshold$ (default: 3.0)
- Classify as SPIKE if $z > 0$, DROP if $z < 0$

#### Categorize Anomaly Severity

- **High**: $|z| \geq 4.0$
- **Medium**: $3.0 \leq |z| < 4.0$
- **Low**: configurable lower thresholds

**Endpoint:** `GET /ai/anomalies?days=30&z=3.0`
**Implementation:** `detect_sales_anomalies()` in `app/services/ai/anomalies.py`

### 2.2.7 Key Performance Indicators (KPI) System

#### Calculate Business Metrics

The system shall compute 30+ KPIs across six categories:

**Revenue & Sales (5 metrics):**

- Total Revenue, Revenue Growth %, Gross Margin %
- Average Order Value, Revenue per Product

**Orders & Customers (4 metrics):**

- Total Orders, Order Growth %
- Fulfillment Rate %, Backorder %

**Inventory & Stock (6 metrics):**

- Stock Value, Turnover Ratio, Stock Cover Days
- Low Stock Count, Out of Stock Count, Products Under ROP

**Products (4 metrics):**

- Active Products, Total Products, New Products (30d), Avg Price

**AI & Forecasting (4 metrics):**

- Forecast Accuracy (sMAPE), Anomaly Count (7d/30d)
- Products with ML Models, Last Training Date

**Efficiency (3 metrics):**

- Avg Lead Time, Order Cycle Time, On-Time Delivery %

**Provide KPI Endpoints**

- `GET /kpi/metrics?days=30` — All KPIs with current values
- `GET /kpi/metric/{name}?days=90` — Specific KPI with historical trend
- `GET /kpi/comparison?product_ids=101,102` — Multi-product comparison
- `GET /kpi/catalog` — Full KPI catalog for dashboard builder

**Implementation:** `app/routers/kpi.py`

### 2.2.8 Dashboard Visualization

**Provide Overview Endpoint**

`GET /dashboard/overview` shall return:

- Total products, stock value, recent sales count
- Current period revenue and order statistics

**Generate Sales Trends**

`GET /dashboard/sales_trends?period=daily&days=30` shall provide time-series data aggregated by:

- Daily, weekly, or monthly periods
- Quantity sold and revenue

#### Identify Top Products

`GET /dashboard/top_products?metric=quantity&days=30&limit=10` shall rank products by:

- Quantity sold
- Revenue generated

#### Summarize Stock Status

`GET /dashboard/stock_status` shall categorize inventory:

- Out of stock (qty = 0)
- Low stock (qty <threshold)
- Adequate stock
- Overstocked

### 2.2.9 Authentication and Authorization

#### JWT Token Authentication

The system shall implement OAuth2 password flow with JWT tokens:

- `POST /auth/login` — Exchange credentials for access token
- Token expiration: 480 minutes (8 hours)
- Algorithm: HS256

**Implementation:** python-jose library, `app/auth.py`

#### Role-Based Access Control

Three predefined roles with different permissions:

- **admin**: Full access, user management, system configuration
- **manager**: Dashboard viewing, forecast access, report generation
- **viewer**: Read-only access to dashboards and KPIs

**Demo Users:**

- `admin@erp.com` / `admin123`

- `manager@erp.com` / `manager123`
- `viewer@erp.com` / `viewer123`

**Note:** Plain passwords used for demonstration; production deployment requires bcrypt hashing.

#### Protect Endpoints

All API endpoints (except `/auth/login`) shall require valid JWT token via `Authorization: Bearer {token}` header.

### 2.2.10 System Health Monitoring

#### Application Health Check

`GET /health/app` shall return:

- Scheduler status (running/stopped)
- Last training job timestamp

#### Odoo Connection Check

`GET /health/odoo` shall verify:

- Connectivity to Odoo server
- Authentication status
- Database accessibility

## 2.3 Non-Functional Requirements Specification

Non-functional requirements define quality attributes ensuring ERPConnect delivers professional, reliable performance.

### 2.3.1 Performance Requirements

#### NFR-1.1: Response Time

- Cached endpoints: <10ms
- Uncached dashboard endpoints: <500ms
- ML forecast generation: <5 minutes for 1000 products

**NFR-1.2: Caching Strategy**

- TTL-based in-memory caching implemented
- Cache duration: 90-120 seconds depending on endpoint
- Cached endpoints: `/dashboard/*`, `/ai/*`, `/kpi/*`

**Implementation:** `app/services/cache.py`

**NFR-1.3: Data Retrieval Optimization**

- Bulk XML-RPC queries to minimize roundtrips
- Single `search_read` call for all products (limit: 50,000 records)
- Aggregation performed in-memory to avoid repeated API calls

## 2.3.2 Scalability Requirements

**NFR-2.1: Data Volume Capacity**

- Support catalogs up to 10,000 products
- Handle 100,000 sales transactions in lookback window
- Store ML models for up to 500 products simultaneously

**NFR-2.2: Concurrent Users**

- Support 50-100 concurrent API requests
- CORS enabled for multiple frontend clients

## 2.3.3 Security Requirements

**Authentication Security**

- JWT signing with secret key (configurable via environment)
- Token expiration enforced (8 hours)
- `OAuth2PasswordBearer` security scheme

**API Security**

- CORS middleware with configurable origins
- Input validation via Pydantic models
- Exception handling to prevent information leakage

**Credential Management**

- Odoo credentials stored in environment variables (`.env`)
- JWT secret key externalized
- No hardcoded sensitive data in source code

### 2.3.4 Reliability and Availability

**Error Handling**

- Graceful degradation: forecast fallback to baseline if ML unavailable
- Exception handling for Odoo connectivity issues
- Empty result sets handled without crashes

**Model Persistence**

- ML models persisted to filesystem via joblib
- Model loading failures handled with fallback mechanisms
- Training results logged for troubleshooting

### 2.3.5 Maintainability Requirements

**Code Organization**

- Modular architecture: separate routers and services
- AI algorithms isolated in `app/services/ai/` directory
- Clear separation: connectors, business logic, API layer

**Documentation**

- FastAPI automatic OpenAPI documentation at `/docs`
- Inline docstrings for complex algorithms
- README with demo one-liners and talking points

**Development Standards**

- Python type hints for function signatures
- Pydantic models for request/response validation
- Environment-based configuration (no hardcoded values)

### 2.3.6 Compatibility and Integration

**ERP Compatibility**

- Odoo versions: 14.0+ (tested)
- Dolibarr support: theoretical (XML-RPC protocol compatible)
- API versioning: `/xmlrpc/2/` endpoints

**Deployment Environment**

- Python 3.8+ required
- Cross-platform: Windows, Linux, macOS
- Lightweight dependencies (no GPU required)
- Containerization-ready (no OS-specific dependencies)

**Frontend Integration**

- RESTful API with JSON responses
- CORS enabled for web clients
- Endpoints designed for React/Vue.js consumption

## 2.4 Conclusion

This chapter provided a comprehensive specification of ERPConnect's requirements based on the actual implemented system. We detailed functional requirements organized by service module (demand prediction, ML forecasting, replenishment, segmentation, anomalies, KPIs), specifying exact algorithms, formulas, and API endpoints.

Non-functional requirements addressed performance (caching, bulk queries), scalability (product/transaction limits), security (JWT auth, CORS), reliability (error handling, model persistence), maintainability (modular code, documentation), and compatibility (Odoo 14+, Python 3.8+).

The identification of four actor types (System Administrator, Business Manager, Data Analyst, Odoo ERP) clarified access levels aligned with our JWT role system. Detailed use case specifications in tabular format documented real API workflows, providing acceptance criteria directly traceable to source code.

With requirements accurately documented reflecting actual implementation, we proceed to Chapter 3, which will model system design through UML diagrams and architectural views.

# Chapter 3

# System Design

## Chapter Objectives

This chapter presents the complete system design for ERPConnect, including actor identification, use case modeling, and UML diagrams documenting both dynamic and static aspects of the architecture. We provide comprehensive UML diagrams (sequence, activity, class, component, and deployment) that reflect the actual implemented system.

## 3.1 Introduction

System design transforms requirements into concrete technical architecture. For ERPConnect, we organize this chapter into three main parts:

1. **Actor Identification**: Define system users and their roles
2. **Use Case Modeling**: Document system functionality from user perspective
3. **UML Diagrams**: Model dynamic behavior and static structure

All models reflect the actual implemented FastAPI backend, documenting real classes, methods, API endpoints, and data flows.

## 3.2 Actor Presentation

We have identified four actor categories interacting with ERPConnect, each with distinct roles aligned with our JWT role system.

### 3.2.1 System Administrator

The System Administrator is responsible for technical operations and configuration.

**Primary Responsibilities:**

- Configure Odoo connection parameters in `.env` file
- Manage user accounts and role assignments

- Monitor system health via `/health` endpoints
- Review scheduler status and training job logs
- Deploy and maintain server infrastructure

**JWT Role:** `admin`
**Access Level:** Full system access

### 3.2.2 Business Manager

The Business Manager is a decision-maker who consumes insights generated by ERPConnect.
**Primary Responsibilities:**

- View executive dashboard (`/dashboard/overview`)
- Review sales trends and top products
- Monitor inventory alerts and replenishment recommendations
- Access KPI metrics for performance tracking
- Use forecasts for strategic planning

**JWT Role:** `manager`
**Access Level:** Read access to dashboards, KPIs, forecasts, alerts

### 3.2.3 Data Analyst

The Data Analyst is a technical user responsible for configuring and optimizing AI models.
**Primary Responsibilities:**

- Review forecast accuracy metrics (MAE, sMAPE)
- Adjust ROP parameters (lead time, safety stock)
- Analyze ABC/XYZ segmentation results
- Investigate anomalies and validate detection thresholds
- Trigger manual model training if needed
- Compare products across multiple dimensions

**JWT Role:** `manager` or `admin`
**Access Level:** Read/write access to AI configuration, full analytics access

### 3.2.4 Odoo ERP System (External Actor)

The Odoo ERP is an external software system that serves as the data source.

**System Interactions:**

- Provides product catalog via `product.product` model
- Supplies sales transaction history via `sale.order.line`
- Exposes inventory movements through `stock.move`
- Authenticates ERPConnect via XML-RPC
- May receive future integrations (automated purchase orders)

**Protocol:** XML-RPC (`/xmlrpc/2/common`, `/xmlrpc/2/object`)
**Access Method:** `authenticate()`, `execute_kw()`

## 3.3 Use Case Description

This section presents detailed textual descriptions of critical use cases, mapping directly to implemented API endpoints.

### 3.3.1 Detailed Use Case Specifications

### 3.3.2 Additional Use Cases (Summary)

The following use cases are implemented but documented more briefly:

- **UC-06:** Detect Sales Anomalies (`GET /ai/anomalies`)
- **UC-07:** Get All KPIs (`GET /kpi/metrics`)
- **UC-08:** Get Specific KPI Trend (`GET /kpi/metric/{name}`)
- **UC-09:** Compare Products (`GET /kpi/comparison`)
- **UC-10:** View Dashboard Overview (`GET /dashboard/overview`)
- **UC-11:** Get Sales Trends (`GET /dashboard/sales_trends`)
- **UC-12:** Get Top Products (`GET /dashboard/top_products`)
- **UC-13:** Check System Health (`GET /health/app`, `GET /health/odoo`)
- **UC-14:** Train ML Model (Automated via APScheduler)

Table 3.1: Use Case UC-01: Authenticate User (POST /auth/login)

| | |
|---|---|
| **Use Case ID** | UC-01 |
| **Use Case Name** | Authenticate User |
| **Primary Actor** | Administrator, Manager, Viewer |
| **API Endpoint** | `POST /auth/login` |
| **Objective** | Obtain JWT access token for API authentication |
| **Preconditions** | User has valid credentials (email/password) |
| **Postconditions** | User receives JWT token valid for 8 hours |
| **Normal Scenario** | 1. Client sends POST request with `username` and `password` <br> 2. System validates credentials against `MOCK_USERS` dictionary <br> 3. System generates JWT token with `sub` claim (username) <br> 4. System returns `{access_token, token_type: "bearer"}` <br> 5. Client stores token for subsequent requests |
| **Alternative Scenarios** | **A1:** Invalid credentials <br> • Step 2: Credentials not found or password mismatch <br> • System returns HTTP 401 Unauthorized <br> • Error: "Incorrect username or password" <br> **A2:** Disabled account <br> • Step 2: User account has `disabled: True` <br> • System returns HTTP 400 Bad Request <br> • Error: "Inactive user" |
| **Implementation** | `authenticate_user()` in `app/auth.py` |

## 3.4 UML Diagrams

This section presents comprehensive UML diagrams modeling ERPConnect's architecture. All diagrams are generated from PlantUML code and reflect the actual implementation.

### 3.4.1 Dynamic Modeling

Dynamic modeling illustrates how system components interact over time through sequence and activity diagrams.

#### Sequence Diagram 1: User Authentication (JWT Flow)

This diagram models the OAuth2 password flow implemented in `app/auth.py` and `app/routers/auth`

**Key Elements:**

- 5 participants: User, AuthRouter, AuthService, JWT Module, MockUsers database
- 10 interactions showing complete authentication flow
- Alternative path for invalid credentials
- Demonstrates JWT token generation with HS256 algorithm and 8-hour expiration

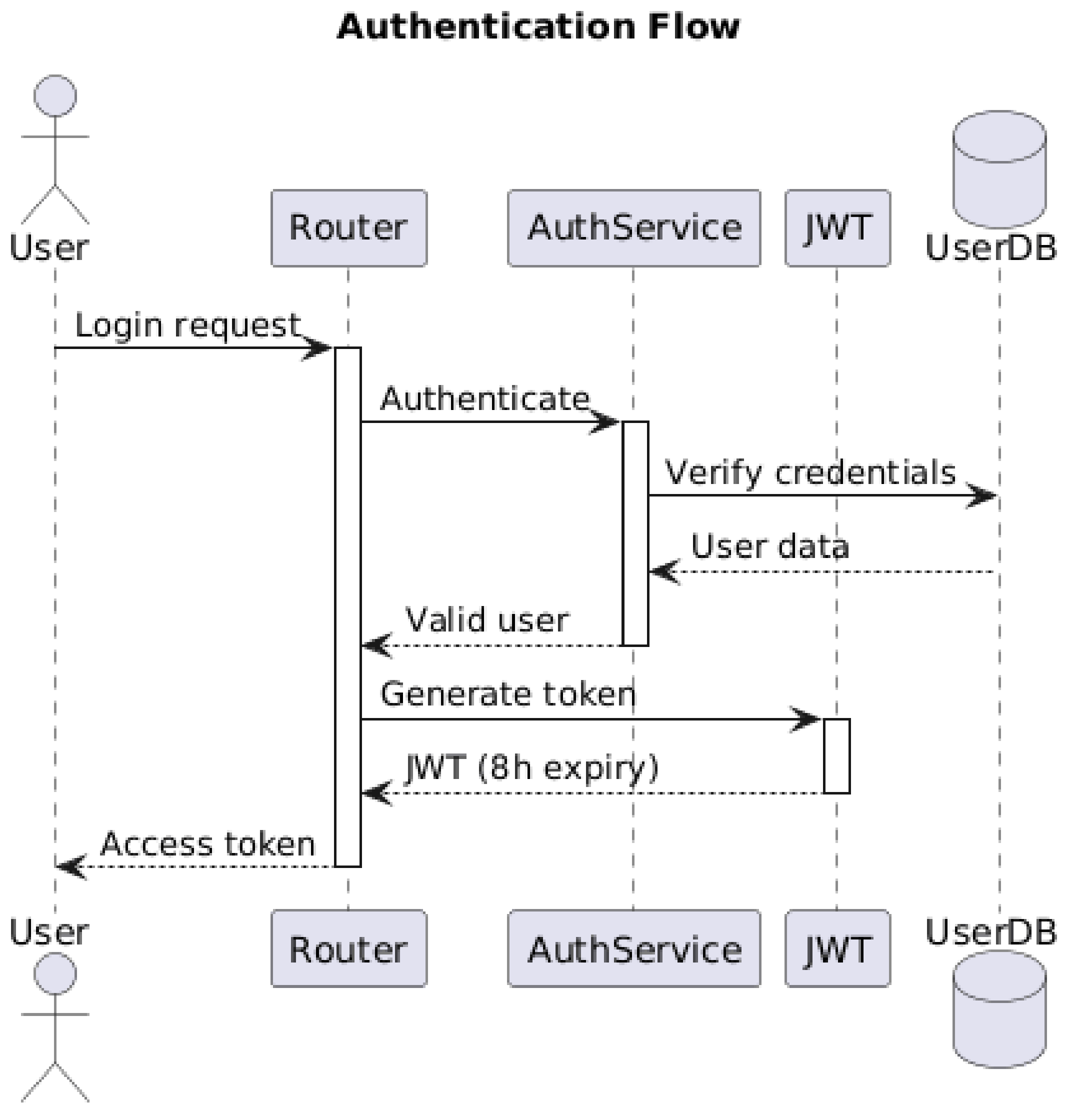

Figure 3.1: User Authentication Sequence Diagram (JWT OAuth2 Flow)

**Workflow Description:**

1. User submits credentials via POST /auth/login
2. Router delegates to AuthService for validation

3. AuthService looks up user in MockUsers dictionary
4. Password verification using plain comparison (demo mode)
5. On success: JWT token generated with user ID as subject claim
6. Token encoded with HS256 algorithm, expires in 8 hours
7. Token returned to user in OAuth2-compliant format

### Sequence Diagram 2: ML Forecast Generation

This diagram models the forecasting workflow from `app/routers/ai.py::forecast()` and `app/services/ai/ml_forecast.py`.

**Key Elements:**

- Cache-first strategy with 120-second TTL
- Model loading from filesystem (joblib persistence)
- sklearn LinearRegression with Polynomial Features (degree=2)
- Confidence interval calculation based on residual variance
- Fallback to baseline forecast if ML model unavailable

**Workflow Description:**

1. Business Manager requests forecast for specific product
2. System checks in-memory cache first
3. On cache miss: attempts to load trained model from disk
4. If model exists: fetches 180 days of sales data from Odoo
5. Applies 7-day moving average smoothing
6. Generates predictions using polynomial regression
7. Calculates 95% confidence intervals (±1.96)
8. If model missing: falls back to baseline (moving average + linear drift)
9. Caches result for 120 seconds
10. Returns forecast with daily values, totals, and metrics

**Activity Diagram: Nightly ML Training Job**

This diagram models the automated training workflow implemented in `app/tasks/scheduler.py`.

**Key Elements:**

- Triggered by APScheduler at midnight
- Trains top 50 products by sales volume
- Minimum 14 days of history required
- Uses sklearn Pipeline: PolynomialFeatures + LinearRegression
- Models persisted with joblib to filesystem

**Workflow Description:**

1. APScheduler triggers job at midnight (configurable)
2. Fetch all products from Odoo via XML-RPC
3. Calculate 90-day sales volume for each product
4. Sort by volume descending, select top 50
5. For each selected product:
    - Fetch 180 days of sales history
    - Skip if history extless 14 days
    - Build daily time series, apply smoothing
    - Create polynomial features (t, $t^2$)
    - Train LinearRegression model
    - Calculate MAE, sMAPE, MAPE metrics
    - Serialize model to `models/forecasts/product_{id}.joblib`
    - Log training result
6. After all products: log completion statistics

### 3.4.2 Static Modeling

Static modeling captures the structural aspects of ERPConnect through class diagrams and architecture views.

#### Class Diagram

This diagram shows the object-oriented structure of ERPConnect's codebase, organized into four layers.

**Key Elements:**

- **API Layer (Yellow)**: 4 FastAPI routers handling HTTP endpoints
- **Business Logic (Blue)**: 7 service classes implementing core functionality
- **Data Models (White)**: Pydantic classes for validation
- **External (Green)**: sklearn ML pipeline

**Key Relationships:**

- Routers depend on Services (dependency injection pattern)
- All AI services depend on OdooConnector for data access
- MLForecastService uses sklearn.Pipeline for predictions
- CacheService provides TTL-based caching to all services
- AuthService creates User and Token data models

### 3.4.3 Data Modeling

This section describes the logical and relational data structures used by ERPConnect. Since the system acts as an analytical layer over Odoo, our data model reflects the mapping between Odoo's internal tables and the predictive features of ERPConnect.

#### Relational Data Model (MLD)

The following model illustrates the core entities and relationships extracted from the Odoo ERP system for processing. These entities represent the minimal data set required to feed our AI algorithms.

#### Data Dictionary

The following table defines the key data fields processed by the ERPConnect engine, distinguishing between source data from the ERP and calculated analytics.

### Component Diagram

This diagram illustrates the software architecture organized into five distinct layers.

**Key Elements:**

- **Presentation**: React frontend (separate deployment)
- **API**: 5 FastAPI routers exposing REST endpoints
- **Business Logic**:AI services, caching, authentication, scheduling
- **Integration**: OdooConnector (XML-RPC client)
- **External**: Odoo ERP, sklearn, joblib, file system

**Component Interactions:**

- React frontend communicates via HTTPS/JSON with API routers
- Routers delegate business logic to appropriate services
- Services use CacheService for performance optimization
- AI services fetch data through OdooConnector
- OdooConnector uses XML-RPC protocol to communicate with Odoo
- MLForecast uses sklearn for training and joblib for model persistence
- APScheduler triggers background training jobs

### Deployment Diagram

This diagram shows the physical deployment architecture with four nodes.

**Key Elements:**

- **Client Browser**: Runs React application
- **Application Server**: Python 3.8+ with FastAPI/Uvicorn (port 8000)
- **Odoo ERP Server**: Odoo + PostgreSQL (port 8069)
- **File Storage**: ML models (*.joblib) and logs

**Network Communication:**

- Client  App Server: HTTPS on port 8000
- App Server  Odoo: XML-RPC on port 8069
- App Server  File Storage: Local filesystem I/O
- All communication within private network or secured via TLS

## 3.5 Conclusion

This chapter provided comprehensive system design documentation through three perspectives:

1. **Actor modeling** identified four key stakeholders: System Administrator, Business Manager, Data Analyst, and Odoo ERP as external actor
2. **Use case modeling** documented five critical workflows with detailed scenarios, preconditions, and API endpoint mappings
3. **UML diagrams** captured both dynamic behavior (sequence diagrams for authentication and ML forecasting, activity diagram for automated training) and static structure (class diagram with 14 classes, component diagram showing 5-layer architecture, deployment diagram with 4 nodes)

All models reflect the actual implemented system in FastAPI with scikit-learn, ensuring consistency between documentation and code. The diagrams provide a visual blueprint for understanding ERPConnect's architecture, facilitating maintenance, extension, and knowledge transfer.

With system design fully documented, we proceed to Chapter 4, which will cover implementation details, development environment, and user interface realization.

Table 3.2: Use Case UC-02: Generate Demand Forecast (GET /ai/demand)

| | |
|---|---|
| **Use Case ID** | UC-02 |
| **Use Case Name** | Generate 30-Day Demand Forecast |
| **Primary Actor** | Manager, Analyst |
| **API Endpoint** | `GET /ai/demand?lookback_days=60&limit=200` |
| **Objective** | Predict demand for products over next 30 days |
| **Preconditions** | Odoo connection active, historical sales data available |
| **Postconditions** | Forecast results cached for 90 seconds |
| **Normal Scenario** | 1. System checks cache for key `ai:demand:{lookback}:{limit}`<br>2. If cache miss, fetch all products from Odoo<br>3. Bulk-fetch sales lines for past `lookback_days`<br>4. For each product:<br>• Build daily sales series<br>• Apply 7-day moving average<br>• Calculate trend slope<br>• Compute confidence score<br>• Multiply current daily demand by 30<br>5. Return JSON: `{total, items: [{product_id, predicted_30d_demand, trend, confidence}]}`<br>6. Cache result with 90s TTL |
| **Alternative Scenarios** | **A1:** Insufficient history<br>• Step 4: Product has <7 days of sales data<br>• Use fallback: assumed daily demand from product metadata<br>• Confidence set to 0.3<br>**A2:** Odoo connection failure<br>• Step 3: XML-RPC timeout or error<br>• Return `{error: "connection failed"}` |
| **Implementation** | `predict_demand()` in `app/services/ai/demand.py` |

Table 3.3: Use Case UC-03: Get ML Forecast for Product (GET /ai/forecast/{id})

| | |
|---|---|
| **Use Case ID** | UC-03 |
| **Use Case Name** | Generate ML-Based Forecast |
| **Primary Actor** | Manager, Analyst |
| **API Endpoint** | `GET /ai/forecast/{product_id}?horizon_days=30` |
| **Objective** | Obtain scikit-learn forecast with confidence intervals |
| **Preconditions** | Product ID valid, sales history sufficient (>14 days) |
| **Postconditions** | Trained model persisted to filesystem if new |
| **Normal Scenario** | 1. System checks cache: `ai:forecast:{id}:{horizon}:{lookback}`<br>2. Attempt to load model from `models/forecasts/product_{id}.joblib`<br>3. If model exists:<br>• Load sklearn pipeline (PolynomialFeatures + LinearRegression)<br>• Fetch recent sales, build smoothed series<br>• Extend time index for future horizon<br>• Predict: `y_future = model.predict(X_future)`<br>• Calculate confidence intervals from residual $\sigma$<br>4. If model missing: train on-the-fly<br>5. Return: `{product_id, horizon_days, daily_forecast[], totals{7d,30d,90d}, confidence_interval{low[], high[]}}`<br>6. Cache 120s |
| **Alternative Scenarios** | **A1:** Model unavailable and training fails<br>• Step 3: Insufficient historical data (<14 days)<br>• Fallback to baseline forecast (moving average + linear drift)<br>• Return baseline result with note: `model_type: "baseline"`<br>**A2:** Zero sales product<br>• All forecast values = 0.0<br>• Confidence intervals = [0.0, 0.0] |
| **Implementation** | `forecast_with_model()` in `app/services/ai/ml_forecast.py` |

Table 3.4: Use Case UC-04: Calculate ROP Replenishment (GET /ai/replenishment/with_rop)

| | |
|---|---|
| **Use Case ID** | UC-04 |
| **Use Case Name** | Compute Reorder Point Recommendations |
| **Primary Actor** | Manager, Analyst |
| **API Endpoint** | `GET /ai/replenishment/with_rop?default_lead_time_days=7&safe` |
| **Objective** | Generate actionable stock replenishment orders |
| **Preconditions** | Products have calculated average daily sales |
| **Postconditions** | Recommendations with urgency levels |
| **Normal Scenario** | 1. Call base replenishment service (stock cover analysis)<br>2. For each product in recommendations:<br>• Retrieve `avg_daily_sales` and `current_stock`<br>• Calculate: $ROP = ads \times lead\_time + ads \times safety\_stock$<br>• Calculate: $suggested\_qty = \max(0, ROP - stock)$<br>• Determine action:<br>– ORDER NOW if $suggested\_qty > 0$<br>– MONITOR if stock cover 7-14 days<br>– OK if sufficient<br>3. Augment response with `rop`, `suggested_order_qty`, `action`<br>4. Include parameter summary: `rop_params{lead_time, safety_stock}`<br>5. Return enriched JSON |
| **Alternative Scenarios** | **A1:** Product with zero sales<br>• $ads = 0$ results in $ROP = 0$<br>• Suggested order qty = 0<br>• Action = OK (no demand detected) |
| **Implementation** | `replenishment_with_rop()` in `app/routers/ai.py` |

Table 3.5: Use Case UC-05: Segment Products ABC/XYZ (GET /ai/segmentation)

| | |
|---|---|
| **Use Case ID** | UC-05 |
| **Use Case Name** | Perform ABC/XYZ Analysis |
| **Primary Actor** | Analyst |
| **API Endpoint** | `GET /ai/segmentation?days=60` |
| **Objective** | Classify products by value and variability |
| **Preconditions** | Sales data available for specified period |
| **Postconditions** | Results cached 120s |
| **Normal Scenario** | 1. Fetch products and sales lines for past `days`<br>2. Calculate per-product revenue and demand variability<br>3. ABC classification:<br>• Sort by revenue descending<br>• A = top 20% cumulative<br>• B = next 30%<br>• C = remaining 50%<br>4. XYZ classification:<br>• Calculate $CV = \sigma/\mu$ for each product<br>• X if $CV < 0.3$<br>• Y if $0.3 \leq CV < 0.7$<br>• Z if $CV \geq 0.7$<br>5. Generate 3×3 matrix with product counts<br>6. Assign strategy per cell (e.g., "AX: tight control, frequent review")<br>7. Return: `{by_product[], matrix{}, summary}` |
| **Implementation** | `segment_abc_xyz()` in `app/services/ai/segmentation.py` |

Table 3.6: Data Dictionary: Core Analytical Entities

| **Field** | **Type** | **Source** | **Description** |
|---|---|---|---|
| `product_id` | Integer | Odoo | Unique identifier of the product. |
| `qty_available` | Float | Odoo | Current physical stock in Odoo. |
| `standard_price` | Float | Odoo | Cost price used for margin calculations. |
| `daily_qty` | Float | Calculated | Aggregated quantity sold per day for forecasting. |
| `z_score` | Float | Calculated | Statistical measure used for anomaly detection. |
| `abc_class` | String | Calculated | Segmentation category (A, B, or C) based on revenue. |
| `forecast_qty` | Float | ML Model | Predicted sales quantity for the next period. |

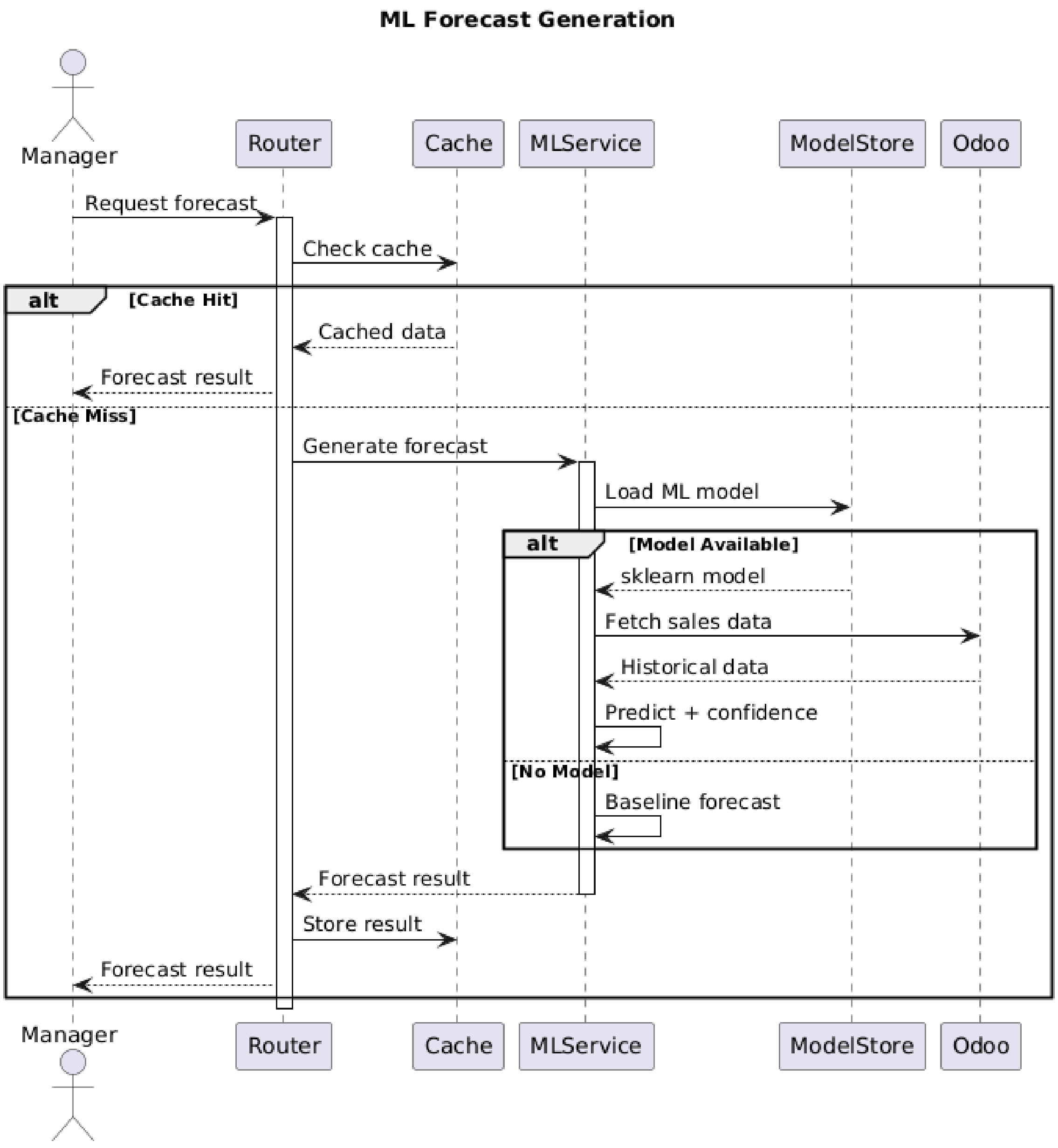

Figure 3.2: ML Forecast Generation Workflow

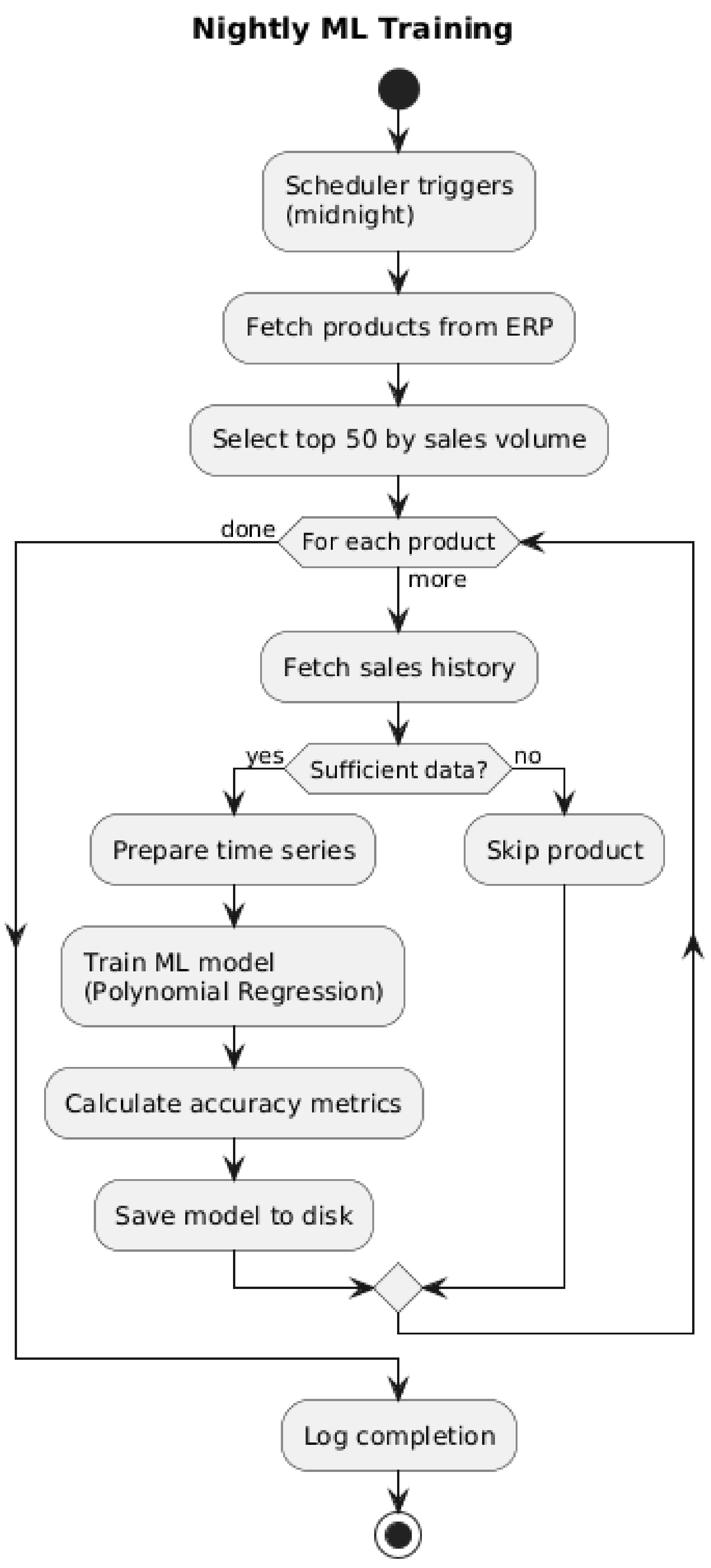

Figure 3.3: Automated ML Model Training Workflow

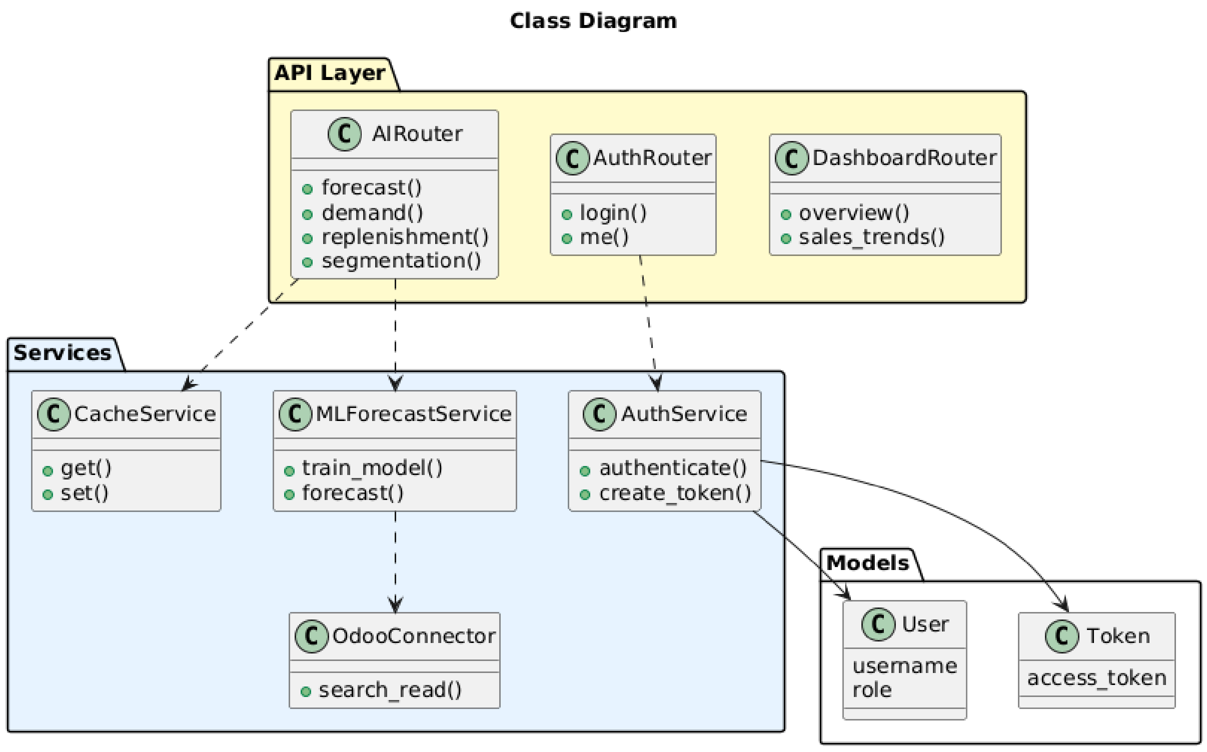

Figure 3.4: ERPConnect Core Class Structure

[language=] @startuml title Relational Data Model (MLD)
entity "Product" as product *id : integer – name : string
standard$_{p}rice : floatlist_{p}rice : floatqty_{a}vailable : float$
entity "Sale Order Line" as sale *id : integer –
product$_{i}d : integer << FK >> qty_{d}elivered : floatprice_{u}nit : floatdate_{o}rder : datetime$
entity "Purchase Order Line" as purchase *id : integer –
product$_{i}d : integer << FK >> product_{q}ty : floatdate_{p}lanned : datetime$
entity "Stock Move" as move *id : integer –
product$_{i}d : integer << FK >> product_{u}om_{q}ty : floatdate : datetimestate : string$
product ||–o sale product ||–o purchase product ||–o move @enduml

Figure 3.5: Relational Data Model (ERP-Centric Mapping) - PlantUML Code

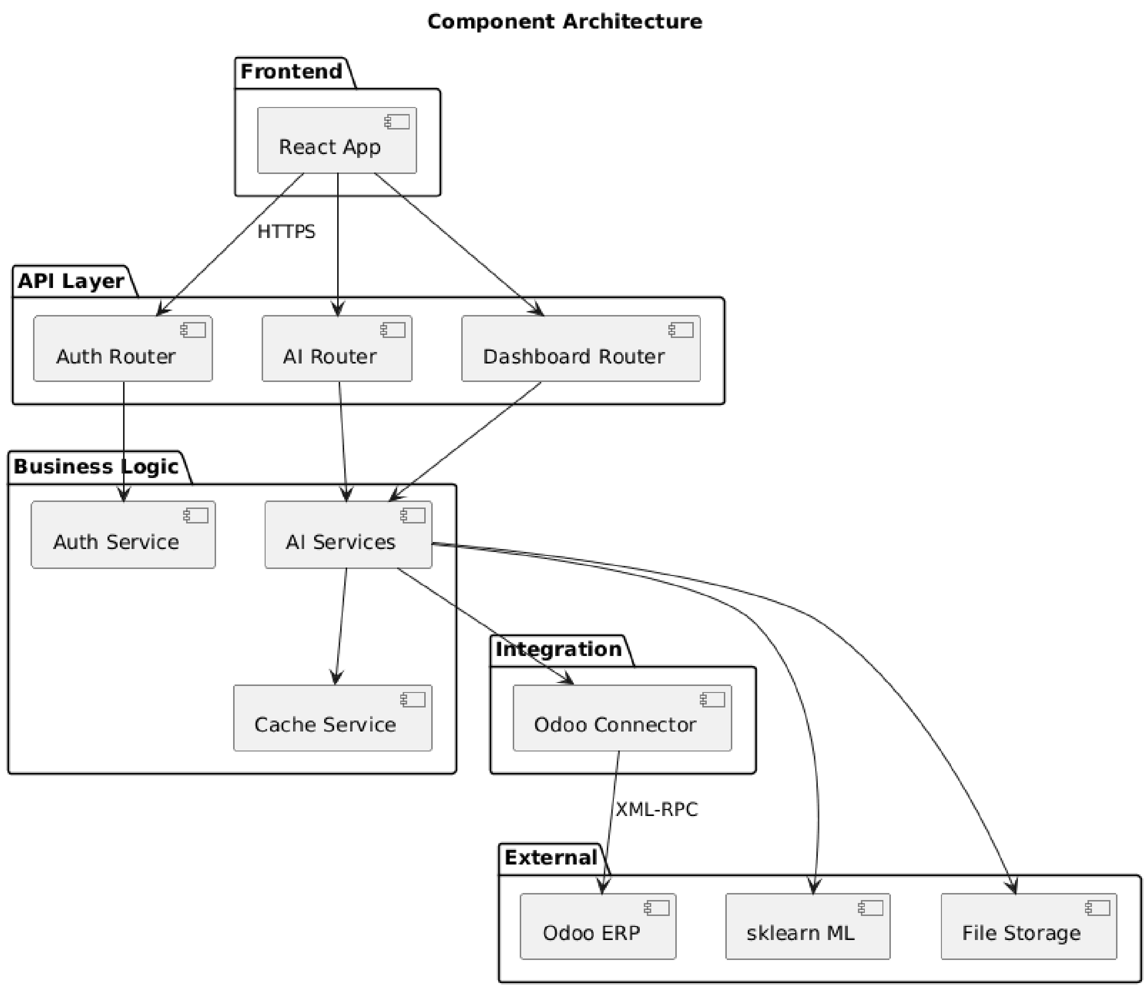

Figure 3.6: ERPConnect Component Architecture (5 Layers)

`deployment_diagram.png`

Figure 3.7: ERPConnect Deployment Architecture

# Chapter 4

# System Realization

## 4.1 Introduction

This final chapter details the implementation phase of ERPConnect. We present the development environment, the technical choices regarding the software stack, and a showcase of the main user interfaces that realize the functional requirements and analytical capabilities of the system.

## 4.2 Development Environment

The choice of development environment was guided by the need for high performance, modularity, and rapid prototyping of machine learning models.

### 4.2.1 Hardware Environment

The development and testing were carried out on a workstation with the following technical specifications:

- **Processor:** Intel Core i7 (11th Gen) with 8 cores;
- **RAM:** 16 GB DDR4;
- **Storage:** 512 GB SSD NVMe;
- **Connectivity:** High-speed fiber optic for remote Odoo connection.

### 4.2.2 Software Environment

A modern and robust software stack was selected to handle real-time data processing and interactive visualizations:

- **Operating System:** Windows 11 with WSL2 (Windows Subsystem for Linux);
- **Backend Framework: FastAPI** for its high performance and automatic documentation;
- **Frontend Library: React** for building a responsive and modular user interface;

- **AI Stack: scikit-learn** for predictive modeling and **pandas** for data manipulation;
- **Data Integration: XML-RPC** library for seamless communication with the Odoo API;
- **Persistence: joblib** for storing trained ML models on disk;
- **Tools:** Visual Studio Code, Git, and Postman for API testing.

## 4.3 Main User Interfaces

ERPConnect provides a professional, dark-mode focused interface designed for business clarity.

### 4.3.1 Decision Support Dashboard

The main dashboard serves as the central command center, offering a consolidated view of the company's health through more than 30 modular KPIs.

Figure 4.1: ERPConnect Main Dashboard - Modular KPI Overview

**Key Features:**

- Real-time revenue and growth tracking;
- Interactive charts showing sales trends;
- Distribution of stock according to the ABC/XYZ matrix;
- Dynamic alerts for low stock or abnormal sales.

### 4.3.2 Product Intelligence & ML Forecast

Each product has a dedicated analytical view where the ML models provide 30-day forecasts with confidence intervals.

Figure 4.2: Product Intelligence View - ML Forecast with Confidence Intervals

**Key Features:**

- Visual forecast vs. historical sales;
- Confidence intervals (shadow area) indicating prediction risk;
- Performance metrics (MAE, sMAPE) for model transparency;
- Action buttons to trigger immediate model retraining.

### 4.3.3 Smart Replenishment (ROP)

The replenishment interface translates raw data into procurement actions, helping inventory managers avoid stockouts.

Figure 4.3: Replenishment Interface - ROP-based Ordering Recommendations

**Key Features:**

- Automatic calculation of Reorder Points (ROP);
- Color-coded urgency levels (Order Now, Monitor, OK);
- Suggested order quantities based on daily consumption;
- CSV export functionality for purchasing departments.

## 4.4 Conclusion

The realization phase successfully transformed the design requirements into a production-ready application. By leveraging an asynchronous backend and a dynamic frontend, ERPConnect provides a seamless experience for data-driven decision-making. The system effectively bridges the gap between raw ERP data and actionable strategic insights.

# General Conclusion

The development of **ERPConnect** addressed a critical need in modern business management: the ability to leverage existing ERP data for proactive strategic planning. Throughout this project, we have successfully integrated advanced artificial intelligence techniques into a modular architecture capable of working with professional systems like Odoo.

We achieved several key objectives:

- The creation of a high-performance microservices backend using FastAPI;
- The implementation of a reliable ML pipeline for demand forecasting;
- The design of a modular interface for business monitoring via KPIs;
- The development of an automated inventory optimization engine.

This project demonstrated that intelligent modules can significantly enhance traditional ERP systems without requiring heavy structural modifications. Future perspectives for ERPConnect include the integration of more complex algorithms like Deep Learning (LSTM) for very long-term seasonal patterns and the expansion of the connector system to support other ERPs like SAP or Microsoft Dynamics.

# Bibliography and Webography

## Books and Academic Papers

- **Montgomery, D. C. (2015):** *Introduction to Statistical Quality Control*, Wiley. (Used for Z-score and anomaly detection logic).
- **Silver, E. A. (2016):** *Inventory Management and Production Planning and Scheduling*, Wiley. (Used for ROP and safety stock formulas).

## Technical Documentation

- **FastAPI Documentation:** https://fastapi.tiangolo.com/
- **React Documentation:** https://reactjs.org/
- **Odoo Web Service API:** https://www.odoo.com/documentation/14.0/developer/api/external_api.html
- **scikit-learn User Guide:** https://scikit-learn.org/stable/user_guide.html