Problem Statement 16: Supply Chain Demand Forecasting

Problem Summary

Develop an AI-powered supply chain demand forecasting platform that leverages multi-source data analysis, advanced machine learning models, and real-time market intelligence to optimize inventory management, reduce costs, and improve customer satisfaction across complex supply chain networks.

Problem Statement

Supply chains face increasing complexity with volatile demand patterns, global disruptions, and multi-tier supplier networks. Your task is to build an AI system that accurately forecasts demand across multiple time horizons, incorporates external market signals, handles supply chain disruptions, and provides actionable insights for inventory optimization. The system should support multi-product forecasting, seasonal adjustments, and scenario planning while maintaining high accuracy and explainability.

Key Requirements

Core Functionality

- **Multi-Horizon Forecasting**: Short-term (1-4 weeks), medium-term (1-6 months), and long-term (6-24 months) demand predictions
- **Multi-Source Data Integration**: Sales history, market trends, economic indicators, weather data, social media sentiment
- **Advanced ML Models**: Time series forecasting, deep learning, ensemble methods with uncertainty quantification
- **Real-time Processing**: Continuous model updates with streaming data and automated retraining
- **Scenario Planning**: What-if analysis for supply chain disruptions, market changes, and promotional impacts
- **Inventory Optimization**: Safety stock calculations, reorder point optimization, and procurement planning

Technical Implementation Steps

1. **Data Integration Pipeline**: Multi-source data collection, cleaning, and feature engineering
2. **Forecasting Engine**: Advanced ML models with automated hyperparameter tuning and model selection
3. **Real-time Processing**: Stream processing for continuous forecasting updates and anomaly detection
4. **Optimization Module**: Inventory optimization algorithms with constraint handling
5. **Visualization Platform**: Interactive dashboards for forecast analysis and business insights
6. **API Services**: RESTful APIs for integration with ERP, WMS, and procurement systems

Data Requirements

Primary Data Sources

- **Historical Sales Data**: Transaction records, order patterns, seasonal trends with 2+ years history
- **Inventory Data**: Stock levels, lead times, supplier performance, warehouse capacity
- **Market Intelligence**: Competitor pricing, market share, industry trends, economic indicators
- **External Factors**: Weather data, holiday calendars, promotional calendars, supply chain events
- **Customer Data**: Segmentation, behavior patterns, loyalty metrics, geographic distribution

Supporting Datasets

- **Supplier Data**: Lead times, reliability scores, capacity constraints, geographic locations
- **Product Hierarchies**: Category relationships, substitution matrices, lifecycle stages
- **Economic Indicators**: GDP, inflation rates, currency exchange rates, commodity prices
- **Social Media Data**: Brand sentiment, trending topics, consumer behavior signals

Technical Themes

- **Time Series Forecasting & Advanced Analytics**
- **Real-time Stream Processing & MLOps**
- **Supply Chain Optimization & Operations Research**
- **Multi-modal Data Fusion & Feature Engineering**
- **Explainable AI & Business Intelligence**

Expected Business Outcomes

Operational Impact

- **Forecast Accuracy**: 15-25% improvement in demand prediction accuracy across all time horizons
- **Inventory Optimization**: 20-30% reduction in excess inventory while maintaining 99%+ service levels
- **Cost Reduction**: $5-10M annual savings through optimized procurement and reduced stockouts
- **Planning Efficiency**: 50% reduction in manual forecasting effort and planning cycle time

Strategic Benefits

- **Supply Chain Resilience**: Improved ability to respond to disruptions and market volatility
- **Customer Satisfaction**: 95%+ order fulfillment rate with reduced lead times
- **Market Responsiveness**: Faster adaptation to demand shifts and seasonal patterns
- **Data-Driven Decisions**: Evidence-based planning with quantified uncertainty and risk assessment

Implementation Strategy

Phase 1: Foundation (Months 1-3)

- Data integration infrastructure and historical data analysis
- Basic forecasting models for key product categories
- Pilot deployment with top 20% of SKUs by revenue

Phase 2: Enhancement (Months 4-6)

- Advanced ML models with ensemble methods and uncertainty quantification
- Real-time processing pipeline and automated model retraining
- Expansion to full product catalog and multi-location forecasting

Phase 3: Optimization (Months 7-9)

- Inventory optimization algorithms and safety stock calculations
- Scenario planning capabilities and what-if analysis tools
- Integration with procurement and supply planning systems

Phase 4: Intelligence (Months 10-12)

- Advanced analytics with external data sources and market intelligence
- Automated anomaly detection and supply chain risk assessment
- AI-driven recommendations for strategic planning and decision support

Success Metrics

- **Forecast Accuracy**: Mean Absolute Percentage Error (MAPE) <15% for short-term, <25% for long-term
- **Inventory Performance**: Inventory turnover ratio improvement of 20%+, stockout reduction of 80%+
- **System Performance**: <5 second response time for forecasts, 99.9% system uptime
- **User Adoption**: >95% user satisfaction, daily active usage by all planning teams
- **ROI**: 400% return on investment within 18 months through cost savings and efficiency gains

This supply chain demand forecasting platform will transform inventory management and planning processes by providing accurate, explainable predictions that enable proactive decision-making, reduce costs, and improve customer satisfaction across complex global supply chains. # Product Requirements Document (PRD) ## Supply Chain Demand Forecasting Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX
- **Last Updated**: 2025-01-XX
- **Document Owner**: Product Management Team
- **Stakeholders**: Supply Chain Leadership, Operations, Data Science Team

---

ETVX Framework Application

Entry Criteria

- âœ… **README.md completed** - Problem statement and business case established
- âœ… **Stakeholder alignment** - Supply chain and operations leadership approval obtained
- âœ… **Market research** - Competitive analysis and supply chain needs assessment completed
- âœ… **Technical feasibility** - Data availability and ML infrastructure validated

Task (This Document)

Define comprehensive product requirements including business objectives, user personas, functional specifications, success metrics, and go-to-market strategy for the Supply Chain Demand Forecasting Platform.

Verification & Validation

- **Internal Review**: Product, Engineering, Operations, and Data Science team approval
- **Stakeholder Validation**: Supply chain leadership and pilot customer feedback
- **Technical Review**: Architecture team feasibility assessment

Exit Criteria

- âœ… **Approved PRD** - All stakeholders have signed off on requirements
- âœ… **Success metrics defined** - Clear KPIs and measurement framework established
- âœ… **Resource allocation** - Budget and team assignments confirmed
- âœ… **Ready for FRD** - Functional requirements development can commence

---

Executive Summary

The Supply Chain Demand Forecasting Platform represents a transformative AI-powered solution designed to revolutionize supply chain planning through intelligent demand prediction, inventory optimization, and scenario planning. Building upon the foundational analysis in our README, this PRD defines the comprehensive product strategy for delivering an enterprise-grade forecasting platform that integrates seamlessly with existing supply chain systems.

Product Vision

To become the leading AI-powered demand forecasting platform that empowers supply chain professionals with accurate, explainable predictions and actionable insights for optimal inventory management and customer satisfaction.

Business Objectives

1. **Forecast Accuracy**: Achieve 15-25% improvement in demand prediction accuracy across all time horizons
2. **Cost Optimization**: Deliver $5-10M annual savings through optimized inventory and procurement
3. **Market Leadership**: Capture 20% market share in AI-powered supply chain analytics within 3 years
4. **Customer Success**: Enable 99%+ service levels while reducing excess inventory by 20-30%

Market Analysis and Opportunity

Market Size and Growth

- **Total Addressable Market (TAM)**: $12.3B global supply chain analytics market
- **Serviceable Addressable Market (SAM)**: $4.8B AI-powered demand forecasting segment
- **Serviceable Obtainable Market (SOM)**: $960M target market for enterprise forecasting solutions
- **Growth Rate**: 18% CAGR projected through 2028

Competitive Landscape

- **Direct Competitors**: Oracle Demand Management Cloud, SAP Integrated Business Planning, Blue Yonder
- **Indirect Competitors**: Traditional statistical forecasting, Excel-based planning, legacy ERP systems
- **Competitive Advantages**: Advanced ML models, real-time processing, explainable AI, multi-horizon forecasting

Market Drivers

- Increasing supply chain complexity and volatility
- Growing adoption of AI/ML in enterprise operations
- Need for resilient supply chains post-COVID disruptions
- Pressure to reduce inventory costs while maintaining service levels

---

User Personas and Stakeholders

Primary Users

1. Demand Planners

- **Role**: Demand planning analysts and managers
- **Goals**: Accurate demand forecasts, reduced manual effort, improved forecast accuracy
- **Pain Points**: Time-consuming manual processes, poor forecast accuracy, limited visibility
- **Success Metrics**: Forecast accuracy improvement, planning cycle time reduction

2. Supply Chain Directors

- **Role**: Senior supply chain leadership and executives
- **Goals**: Strategic planning, cost optimization, risk management
- **Pain Points**: Lack of visibility, reactive planning, high inventory costs
- **Success Metrics**: Cost savings, service level improvements, inventory optimization

3. Operations Managers

- **Role**: Warehouse and distribution center managers
- **Goals**: Optimal inventory levels, efficient operations, customer satisfaction
- **Pain Points**: Stockouts, excess inventory, poor demand visibility
- **Success Metrics**: Inventory turnover, stockout reduction, operational efficiency

Secondary Users

4. Procurement Teams

- **Role**: Strategic sourcing and procurement professionals
- **Goals**: Optimized purchasing decisions, supplier management
- **Success Metrics**: Cost savings, supplier performance, procurement efficiency

5. Sales Teams

- **Role**: Sales managers and account executives
- **Goals**: Product availability, customer satisfaction, revenue growth
- **Success Metrics**: Order fulfillment rates, customer satisfaction scores

---

Product Features and Capabilities

Core Features (MVP)

1. Multi-Horizon Forecasting Engine

- **Description**: AI-powered forecasting for short, medium, and long-term horizons
- **Business Value**: Accurate demand predictions across all planning horizons
- **Technical Requirements**: Time series models, deep learning, ensemble methods
- **Success Metrics**: <15% MAPE for short-term, <25% MAPE for long-term forecasts

2. Real-Time Data Integration

- **Description**: Seamless integration with ERP, WMS, and external data sources
- **Business Value**: Up-to-date forecasts with latest market intelligence
- **Technical Requirements**: API integrations, streaming data processing
- **Success Metrics**: <5 minute data latency, 99.9% integration uptime

3. Interactive Forecasting Dashboard

- **Description**: Intuitive web-based interface for forecast analysis and planning
- **Business Value**: Improved user experience and faster decision-making
- **Technical Requirements**: React-based UI, real-time visualizations
- **Success Metrics**: >90% user satisfaction, <3 second page load times

4. Scenario Planning and What-If Analysis

- **Description**: Advanced scenario modeling for supply chain disruptions and market changes
- **Business Value**: Proactive planning and risk mitigation
- **Technical Requirements**: Monte Carlo simulation, sensitivity analysis
- **Success Metrics**: 50% improvement in disruption response time

Advanced Features (Future Releases)

5. Inventory Optimization Suite

- **Description**: AI-driven safety stock and reorder point optimization
- **Business Value**: Optimal inventory levels with minimized costs
- **Timeline**: Release 2.0 (Month 6)

6. External Market Intelligence

- **Description**: Integration with economic indicators, weather data, social media sentiment
- **Business Value**: Enhanced forecast accuracy with external signals
- **Timeline**: Release 3.0 (Month 9)

7. Automated Anomaly Detection

- **Description**: Real-time detection of demand anomalies and supply chain disruptions
- **Business Value**: Proactive issue identification and response
- **Timeline**: Release 2.0 (Month 6)

Technical Requirements

Performance Requirements

- **Response Time**: <5 seconds for forecast generation
- **Throughput**: Support 100,000+ SKUs per forecast run
- **Availability**: 99.9% uptime with <4 hours planned maintenance monthly
- **Scalability**: Horizontal scaling to support multi-billion dollar enterprises

Data Requirements

- **Historical Data**: Minimum 2 years of sales and inventory data

- **Real-Time Processing**: <5 minute latency for streaming data updates
- **Data Quality**: Automated data validation and cleansing capabilities
- **External Data**: Integration with 10+ external data sources

Integration Requirements

- **ERP Systems**: SAP, Oracle, Microsoft Dynamics integration
- **Data Formats**: Support for CSV, JSON, XML, EDI formats
- **APIs**: RESTful APIs with OAuth 2.0 authentication
- **Cloud Platforms**: AWS, Azure, GCP deployment options

Success Metrics and KPIs

Forecast Accuracy Metrics

- **Primary**: Mean Absolute Percentage Error (MAPE) <15% for short-term forecasts
- **Secondary**: Forecast Bias <Â±5%, Forecast Value Added (FVA) >10%
- **Tertiary**: Prediction Interval Coverage >90%

Business Impact Metrics

- **Cost Savings**: $5-10M annual savings through inventory optimization
- **Service Levels**: 99%+ order fulfillment rate maintenance
- **Inventory Efficiency**: 20-30% reduction in excess inventory

User Experience Metrics

- **Adoption**: >95% daily active users among target planners
- **Satisfaction**: >4.5/5.0 user satisfaction score
- **Efficiency**: 50% reduction in manual forecasting effort

Technical Performance Metrics

- **Reliability**: 99.9% system uptime
- **Performance**: <5 second average response time
- **Scalability**: Support for 1M+ SKUs without performance degradation

---

Go-to-Market Strategy

Target Market Segmentation

1. **Primary**: Large manufacturers and retailers ($1B+ revenue)
2. **Secondary**: Mid-market companies ($100M-$1B revenue)
3. **Tertiary**: Specialized supply chain service providers

Sales Strategy

- **Direct Sales**: Enterprise sales team for large accounts
- **Channel Partners**: Integration with ERP vendors and supply chain consultants
- **Pilot Programs**: Free pilot implementations to demonstrate value

Pricing Model

- **Subscription**: Per-SKU monthly subscription ($0.50-$2.00/SKU/month)
- **Implementation**: One-time setup and integration fees
- **Professional Services**: Training, customization, and optimization services

Launch Timeline

- **Phase 1**: Pilot customers (Months 1-6)
- **Phase 2**: Early adopters (Months 7-12)
- **Phase 3**: Market expansion (Months 13-24)

Risk Assessment and Mitigation

High-Risk Items

1. **Data Quality Issues**
   - **Mitigation**: Robust data validation, cleansing algorithms
   - **Contingency**: Manual data correction workflows, quality scoring
2. **Forecast Accuracy Challenges**
   - **Mitigation**: Ensemble models, continuous learning, expert validation
   - **Contingency**: Hybrid human-AI forecasting approach
3. **Integration Complexity**
   - **Mitigation**: Standardized APIs, experienced integration team
   - **Contingency**: Phased integration approach, fallback options

Medium-Risk Items

1. **Competitive Response**
   - **Mitigation**: Patent protection, continuous innovation
   - **Contingency**: Feature differentiation, pricing flexibility
2. **Technology Scalability**
   - **Mitigation**: Cloud-native architecture, performance testing
   - **Contingency**: Infrastructure scaling, optimization efforts

---

Resource Requirements

Team Structure

- **Product Management**: 2 FTE (Product Manager, Technical Product Manager)
- **Engineering**: 15 FTE (Backend, Frontend, ML, Data Engineering, DevOps)
- **Data Science**: 4 FTE (ML Engineers, Data Scientists, Research Scientists)
- **Sales & Marketing**: 5 FTE (Sales Director, Marketing Manager, Customer Success)

Budget Allocation

- **Development**: $3.2M (65% of total budget)
- **Data Acquisition**: $600K (12% of total budget)
- **Sales & Marketing**: $700K (14% of total budget)
- **Operations**: $500K (9% of total budget)

Technology Infrastructure

- **Cloud Platform**: AWS/Azure multi-region deployment
- **ML Platform**: MLflow, Kubeflow for model lifecycle management
- **Data Platform**: Snowflake/Databricks for data warehousing and processing

---

Assumptions and Dependencies

Key Assumptions

1. **Market Demand**: Continued growth in AI adoption for supply chain optimization
2. **Data Availability**: Customers have sufficient historical data for model training
3. **Technology Maturity**: ML/AI technology sufficient for enterprise forecasting applications
4. **Customer Readiness**: Organizations ready for AI-driven planning processes

Critical Dependencies

1. **Data Access**: Availability of high-quality, comprehensive supply chain data
2. **Integration Partners**: Cooperation from ERP vendors for seamless integration
3. **Cloud Infrastructure**: Reliable cloud services for scalable deployment
4. **Talent Acquisition**: Access to skilled ML engineers and data scientists

External Factors

1. **Economic Conditions**: Supply chain investment levels and technology spending
2. **Regulatory Environment**: Data privacy and AI governance requirements
3. **Technology Evolution**: Advances in ML/AI technologies and cloud platforms
4. **Competitive Landscape**: New entrants and competitive responses

---

Out of Scope

Excluded Features

1. **Transportation Optimization**: Route planning and logistics optimization
2. **Supplier Management**: Supplier relationship and performance management
3. **Manufacturing Planning**: Production scheduling and capacity planning
4. **Financial Planning**: Budgeting and financial forecasting applications

Future Considerations

1. **International Markets**: Global expansion beyond North American market
2. **Industry Verticals**: Specialized solutions for specific industries
3. **Advanced Analytics**: Prescriptive analytics and automated decision-making
4. **IoT Integration**: Real-time sensor data and edge computing capabilities

---

Conclusion

This PRD establishes the foundation for developing a transformative Supply Chain Demand Forecasting Platform that addresses critical business needs while ensuring technical feasibility and commercial viability. The comprehensive requirements outlined here, building upon our README analysis, provide clear direction for the development team and stakeholders.

The success of this platform depends on our ability to deliver accurate, explainable AI-powered forecasts that integrate seamlessly with existing supply chain workflows while providing significant cost savings and operational improvements.

Next Steps: Proceed to Functional Requirements Document (FRD) development to detail specific system behaviors and technical specifications based on these product requirements.

Document Approval

Role	Name	Signature	Date
Product Manager	[Name]	[Signature]	[Date]
Supply Chain Director	[Name]	[Signature]	[Date]
Engineering Lead	[Name]	[Signature]	[Date]
Data Science Lead	[Name]	[Signature]	[Date]

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Functional Requirements Document (FRD) ## Supply Chain Demand Forecasting Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX
- **Document Owner**: Engineering Team

ETVX Framework Application

Entry Criteria

- âœ… **README.md completed** - Problem statement and business case established
- âœ… **PRD Approved** - Product requirements and business objectives defined
- âœ… **Stakeholder Sign-off** - Supply chain and technical teams aligned on scope

Task (This Document)

Define detailed functional specifications for all system modules, user interactions, data flows, and integration requirements based on PRD objectives and README foundation.

Verification & Validation

- **Requirements Traceability** - All PRD features mapped to functional requirements
- **User Validation** - Supply chain professionals review of workflows
- **Technical Review** - Engineering team feasibility assessment

Exit Criteria

- âœ… **Complete Functional Specs** - All system behaviors documented
- âœ… **Acceptance Criteria** - Testable requirements defined
- âœ… **Integration Requirements** - External system interfaces specified

---

System Overview

Building upon the README problem statement and PRD business objectives, this FRD details the functional behavior of the Supply Chain Demand Forecasting Platform across seven core modules: Data Integration, Forecasting Engine, Scenario Planning, Inventory Optimization, User Interface, Integration Services, and Analytics & Reporting.

Module 1: Data Integration and Management System

FR-1.1: Multi-Source Data Ingestion

Description: Ingest and process data from multiple supply chain systems and external sources **Inputs**: ERP systems, sales data, inventory systems, external market data **Processing**: - Parse various data formats (CSV, JSON, XML, EDI) - Validate data integrity and completeness - Apply data quality scoring and cleansing algorithms **Outputs**: Standardized, validated supply chain dataset **Acceptance Criteria**: - Support 99.9% uptime for data ingestion - Process 1M+ records per hour - Maintain data lineage for audit trails - Handle 20+ different data source formats

FR-1.2: Historical Data Processing and Storage

Description: Process and store historical sales and inventory data for model training **Inputs**: Historical transaction data, inventory movements, promotional data **Processing**: - Aggregate data at multiple time granularities (daily, weekly, monthly) - Handle missing data with interpolation and statistical methods - Create feature engineering pipelines for ML models **Outputs**: Time-series datasets optimized for forecasting models **Acceptance Criteria**: - Store minimum 2 years of historical data - Support data aggregation at multiple levels (SKU, category, region) - Maintain 99.9% data accuracy after processing - Enable sub-second query response for historical data

FR-1.3: Real-Time Data Streaming

Description: Process real-time data streams for continuous forecast updates **Inputs**: Live sales transactions, inventory updates, external market feeds **Processing**: - Stream processing with Apache Kafka - Real-time data validation and anomaly detection - Incremental model updates with new data **Outputs**: Real-time data streams for forecasting models **Acceptance Criteria**: - <5 minute latency for real-time data processing - Support 10,000+ transactions per second - 99.9% message delivery reliability - Automatic error handling and recovery

---

Module 2: AI-Powered Forecasting Engine

FR-2.1: Multi-Horizon Demand Forecasting

Description: Generate demand forecasts across short, medium, and long-term horizons **Inputs**: Historical sales data, external factors, seasonal patterns **Processing**: - Apply ensemble ML models (ARIMA, Prophet, LSTM, Transformer) - Generate forecasts for 1-4 weeks, 1-6 months, 6-24 months - Calculate prediction intervals and uncertainty quantification **Outputs**: Demand forecasts with confidence intervals **Acceptance Criteria**: - <15% MAPE for short-term forecasts (1-4 weeks) - <25% MAPE for long-term forecasts (6-24 months) - Generate forecasts for 100,000+ SKUs within 30 minutes - Provide 90% prediction intervals for all forecasts

FR-2.2: Automated Model Selection and Tuning

Description: Automatically select and tune optimal forecasting models for each SKU **Inputs**: Historical performance data, SKU characteristics, forecast accuracy metrics **Processing**: - Evaluate multiple model types for each SKU - Perform automated hyperparameter optimization - Implement model ensemble techniques for improved accuracy **Outputs**: Optimized forecasting models with performance metrics **Acceptance Criteria**: - Automatically select best-performing model for each SKU - Achieve 15% improvement in forecast accuracy over baseline - Complete model selection within 2 hours for full catalog - Support A/B testing for model comparison

FR-2.3: External Factor Integration

Description: Incorporate external factors into forecasting models **Inputs**: Weather data, economic indicators, promotional calendars, competitor data **Processing**: - Feature engineering for external variables - Correlation analysis with demand patterns - Dynamic factor weighting based on relevance **Outputs**: Enhanced forecasts incorporating external signals **Acceptance Criteria**: - Integrate 10+ external data sources - Improve forecast accuracy by 10% with external factors - Automatically detect relevant external factors for each SKU - Update external factor impact in real-time

---

Module 3: Scenario Planning and What-If Analysis

FR-3.1: Supply Chain Disruption Modeling

Description: Model impact of supply chain disruptions on demand and inventory **Inputs**: Disruption scenarios, supplier data, lead time variations **Processing**: - Monte Carlo simulation for disruption impact analysis - Sensitivity analysis for key supply chain parameters - Risk assessment and mitigation recommendations **Outputs**: Disruption impact forecasts and mitigation strategies **Acceptance Criteria**: - Model 20+ disruption scenarios (natural disasters, supplier issues, etc.) - Generate impact analysis within 10 minutes - Provide quantified risk assessments with probability distributions - Recommend specific mitigation actions with cost-benefit analysis

FR-3.2: Promotional and Marketing Impact Analysis

Description: Analyze impact of promotions and marketing campaigns on demand **Inputs**: Promotional calendars, marketing spend, historical promotion performance **Processing**: - Promotional lift modeling and cannibalization analysis - Cross-product impact assessment - ROI calculation for promotional activities **Outputs**: Promotional impact forecasts and optimization recommendations **Acceptance Criteria**: - Model promotional lift with 85% accuracy - Analyze cross-product cannibalization effects - Provide promotional ROI calculations - Support what-if analysis for promotional planning

FR-3.3: Market Scenario Planning

Description: Create and analyze various market scenarios for strategic planning **Inputs**: Economic forecasts, market trends, competitive intelligence **Processing**: - Scenario generation based on market conditions - Demand sensitivity analysis to market changes - Strategic planning recommendations **Outputs**: Market scenario forecasts and strategic insights **Acceptance Criteria**: - Generate 5+ market scenarios (bull, bear, base case, etc.) - Quantify demand sensitivity to market changes - Provide strategic recommendations with confidence levels - Enable custom scenario creation by users

---

Module 4: Inventory Optimization Engine

FR-4.1: Safety Stock Optimization

Description: Calculate optimal safety stock levels based on demand variability and service targets **Inputs**: Demand forecasts, service level targets, lead time data, cost parameters **Processing**: - Statistical safety stock calculations - Service level optimization algorithms - Cost-benefit analysis for inventory investments **Outputs**: Optimized safety stock recommendations **Acceptance Criteria**: - Achieve target service levels (95-99.5%) while minimizing inventory - Reduce safety stock by 20% while maintaining service levels - Consider demand variability and lead time uncertainty - Provide cost impact analysis for safety stock changes

FR-4.2: Reorder Point and Quantity Optimization

Description: Optimize reorder points and order quantities for inventory replenishment **Inputs**: Demand forecasts, lead times, ordering costs, holding costs **Processing**: - Economic Order Quantity (EOQ) optimization - Dynamic reorder point calculation - Multi-echelon inventory optimization **Outputs**: Optimized replenishment parameters **Acceptance Criteria**: - Minimize total inventory costs (ordering + holding + stockout) - Optimize reorder points for 100,000+ SKUs - Consider multi-location inventory networks - Provide sensitivity analysis for cost parameters

FR-4.3: Inventory Allocation and Distribution

Description: Optimize inventory allocation across multiple locations and channels **Inputs**: Demand forecasts by location, transportation costs, capacity constraints **Processing**: - Network optimization algorithms - Allocation optimization considering constraints - Transportation cost minimization **Outputs**: Optimal inventory allocation plans **Acceptance Criteria**: - Minimize total supply chain costs - Consider capacity constraints at all locations - Optimize allocation for 1000+ locations - Provide allocation recommendations within 15 minutes

---

Module 5: User Interface and Experience

FR-5.1: Interactive Forecasting Dashboard

Description: Provide intuitive web-based interface for forecast analysis and management **Inputs**: Forecast data, user preferences, filter criteria **Processing**: - Real-time data visualization and charting - Interactive filtering and drill-down capabilities - Customizable dashboard layouts **Outputs**: Interactive forecast visualizations and insights **Acceptance Criteria**: - <3 second page load times - Support 500+ concurrent users - Mobile-responsive design - Customizable dashboards per user role

FR-5.2: Forecast Adjustment and Override Capabilities

Description: Enable users to manually adjust and override AI-generated forecasts **Inputs**: AI forecasts, user adjustments, business rationale **Processing**: - Forecast adjustment workflows with approval processes - Impact analysis of manual overrides - Audit trail for all forecast changes **Outputs**: Adjusted forecasts with change documentation **Acceptance Criteria**: - Enable forecast adjustments at any aggregation level - Track forecast accuracy for manual vs.Â AI predictions - Maintain complete audit trail of changes - Support bulk forecast adjustments

FR-5.3: Alert and Notification System

Description: Provide intelligent alerts for forecast anomalies and business events **Inputs**: Forecast data, business rules, user preferences **Processing**: - Anomaly detection algorithms - Rule-based alert generation - Multi-channel notification delivery **Outputs**: Targeted alerts and notifications **Acceptance Criteria**: - Detect forecast anomalies with 95% accuracy - Support email, SMS, and in-app notifications - Enable customizable alert thresholds per user - Reduce alert fatigue with intelligent filtering

---

Module 6: Integration Services

FR-6.1: ERP System Integration

Description: Bidirectional integration with major ERP systems **Inputs**: ERP data feeds, forecast outputs, inventory recommendations **Processing**: - Real-time data synchronization - API-based integration with authentication - Error handling and retry mechanisms **Outputs**: Integrated supply chain workflows **Acceptance Criteria**: - Support SAP, Oracle, Microsoft Dynamics integration - 99.9% data synchronization success rate - <10 second synchronization latency - Handle 1M+ transactions per day

FR-6.2: Third-Party Data Provider Integration

Description: Integration with external data providers for market intelligence **Inputs**: External data feeds, API credentials, data mapping configurations **Processing**: - Automated data ingestion from multiple providers - Data format standardization and validation - Real-time data quality monitoring **Outputs**: Enriched datasets with external intelligence **Acceptance Criteria**: - Integrate with 20+ external data providers - Support various data formats and protocols - Maintain 99% data quality scores - Enable real-time data updates

FR-6.3: Warehouse Management System Integration

Description: Integration with WMS for real-time inventory visibility **Inputs**: WMS inventory data, location information, movement transactions **Processing**: - Real-time inventory synchronization - Location-specific demand forecasting - Inventory movement tracking **Outputs**: Location-aware forecasts and inventory optimization **Acceptance Criteria**: - Support major WMS platforms (Manhattan, SAP EWM, Oracle WMS) - Real-time inventory visibility across 1000+ locations - <5 minute latency for inventory updates - 99.9% transaction accuracy

---

Module 7: Analytics and Reporting

FR-7.1: Forecast Performance Analytics

Description: Comprehensive analytics on forecast accuracy and performance **Inputs**: Forecast data, actual sales data, performance metrics **Processing**: - Forecast accuracy calculations (MAPE, RMSE, MAE) - Trend analysis and performance tracking - Comparative analysis across products and time periods **Outputs**: Forecast performance reports and insights **Acceptance Criteria**: - Generate performance reports within 5 minutes - Track accuracy trends over time - Provide drill-down analysis by product, region, time period - Support automated performance alerts

FR-7.2: Business Intelligence and Insights

Description: Generate actionable business insights from forecasting data **Inputs**: Forecast data, sales data, external market data **Processing**: - Pattern recognition and trend analysis - Correlation analysis between variables - Automated insight generation using NLP **Outputs**: Business insights and recommendations **Acceptance Criteria**: - Generate 10+ automated insights per analysis - Identify significant trends and patterns - Provide natural language explanations - Enable insight sharing and collaboration

FR-7.3: Custom Reporting and Dashboards

Description: Enable users to create custom reports and dashboards **Inputs**: User requirements, data selections, visualization preferences **Processing**: - Drag-and-drop report builder - Custom visualization creation - Scheduled report generation and distribution **Outputs**: Custom reports and dashboards **Acceptance Criteria**: - Support 20+ visualization types - Enable scheduled report delivery - Allow report sharing and collaboration - Provide export capabilities (PDF, Excel, CSV)

Data Flow Architecture

Primary Data Flow

1. **Data Ingestion** â†’ Multi-source data collection and validation
2. **Data Processing** â†’ Cleansing, aggregation, and feature engineering
3. **Model Training** â†’ Automated model selection and training
4. **Forecast Generation** â†’ Multi-horizon demand predictions

5. **Optimization** â†’ Inventory and supply chain optimization
6. **Visualization** â†’ Dashboard and reporting delivery

Real-time Processing Flow

1. **Stream Ingestion** â†’ Real-time data capture from multiple sources
2. **Stream Processing** â†’ Real-time validation and transformation
3. **Model Inference** â†’ Real-time forecast updates
4. **Alert Generation** â†’ Anomaly detection and notification
5. **Dashboard Updates** â†’ Real-time visualization updates

Integration Flow

1. **API Requests** â†’ External system integration requests
2. **Data Transformation** â†’ Format conversion and mapping
3. **Validation** â†’ Data quality and business rule validation
4. **Synchronization** â†’ Bidirectional data synchronization
5. **Error Handling** â†’ Exception management and retry logic

---

Performance Requirements

Response Time Targets

- **Forecast Generation** â†’ <30 minutes for 100,000+ SKUs
- **Dashboard Loading** â†’ <3 seconds for complete interface
- **Real-time Updates** â†’ <5 minutes from data ingestion to visualization
- **Report Generation** â†’ <5 minutes for standard reports

Scalability Specifications

- **Concurrent Users** â†’ Support 500+ simultaneous users
- **SKU Volume** â†’ Handle 1M+ SKUs per deployment
- **Data Throughput** â†’ Process 10M+ transactions per hour
- **Geographic Distribution** â†’ Multi-region deployment capabilities

---

Acceptance Criteria Summary

Each functional requirement includes specific, measurable acceptance criteria that enable comprehensive testing and validation. Key success metrics include:

- **Forecast Accuracy** â†’ <15% MAPE for short-term, <25% MAPE for long-term
- **System Performance** â†’ <5 second response times for critical functions
- **User Experience** â†’ >90% user satisfaction scores
- **Integration Success** â†’ 99.9% data synchronization reliability
- **Scalability** â†’ Support for enterprise-scale deployments

---

Conclusion

This FRD provides comprehensive functional specifications for the Supply Chain Demand Forecasting Platform, building upon the README problem statement and PRD requirements with detailed system behaviors, acceptance criteria, and integration specifications. These requirements enable the development team to proceed with technical design and implementation while ensuring full traceability to business objectives.

Next Steps: Proceed to Non-Functional Requirements Document (NFRD) development to define quality attributes, performance constraints, and operational requirements.

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Non-Functional Requirements Document (NFRD) ## Supply Chain Demand Forecasting Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX
- **Document Owner**: Engineering & Operations Team

ETVX Framework Application

Entry Criteria

- âœ… **README.md completed** - Problem statement and business case established
- âœ… **PRD Approved** - Business objectives and product features defined
- âœ… **FRD Completed** - Functional specifications documented

Task (This Document)

Define quality attributes, performance constraints, security requirements, and operational characteristics that ensure the system meets enterprise supply chain standards.

Verification & Validation

- **Performance Testing** - Load testing and benchmarking validation
- **Security Assessment** - Penetration testing and compliance audit
- **Operational Review** - SRE team validation of operational requirements

Exit Criteria

- âœ… **Quality Attributes Defined** - All non-functional aspects specified
- âœ… **Compliance Framework** - Enterprise requirements documented
- âœ… **Operational Standards** - SLA and monitoring requirements established

---

System Quality Attributes

Building upon the README problem statement, PRD business objectives, and FRD functional specifications, this NFRD defines the quality characteristics that ensure the Supply Chain Demand Forecasting Platform meets enterprise standards for performance, reliability, security, and operational excellence.

Performance Requirements

NFR-1.1: Response Time Performance

Requirement: System must provide optimal response times for critical supply chain functions **Specifications**: - Forecast generation: <30 minutes for 100,000+ SKUs - Dashboard page load: <3 seconds (95th percentile) - Real-time data processing: <5 minutes from ingestion to availability - API response time: <2 seconds for standard queries **Measurement**: Application Performance Monitoring (APM) tools **Acceptance Criteria**: - 95% of requests meet specified response times - No degradation during peak planning periods - Performance maintained under 5x normal load

NFR-1.2: Throughput Capacity

Requirement: Support high-volume supply chain data processing **Specifications**: - Data ingestion: 1M+ records per hour - Concurrent forecast generation: 1M+ SKUs simultaneously - API requests: 5,000+ requests per minute - Real-time processing: 10,000+ transactions per second **Measurement**: Load testing and production metrics **Acceptance Criteria**: - Linear scalability up to specified limits - No data loss during peak processing - Graceful degradation beyond capacity limits

---

Scalability Requirements

NFR-2.1: Horizontal Scalability

Requirement: System must scale horizontally to support growing enterprises **Specifications**: - Auto-scaling based on CPU/memory utilization (70% threshold) - Support for multi-location deployments (1000+ locations) - Database sharding for large SKU catalogs - Microservices architecture with independent scaling **Measurement**: Infrastructure monitoring and capacity planning **Acceptance Criteria**: - Automatic scaling within 10 minutes of threshold breach - No service interruption during scaling events - Cost-effective resource utilization (>75% efficiency)

NFR-2.2: Data Volume Scalability

Requirement: Handle exponential growth in supply chain data volume **Specifications**: - SKU catalog: 10M+ active SKUs per deployment - Historical data: 5+ years of transaction history - Real-time streams: 100,000+ events per second - Storage growth: 100TB+ per year for large enterprises **Measurement**: Database performance metrics and storage utilization **Acceptance Criteria**: - Query performance maintained with data growth - Automated data archiving and lifecycle management - Cost-optimized storage tiering implementation

---

Reliability and Availability Requirements

NFR-3.1: System Availability

Requirement: Ensure continuous availability for critical supply chain operations **Specifications**: - Uptime: 99.9% availability (8.77 hours downtime per year) - Planned maintenance: <4 hours per month - Recovery Time Objective (RTO): <30 minutes - Recovery Point Objective (RPO): <15 minutes data loss **Measurement**: Uptime monitoring and incident tracking **Acceptance Criteria**: - No single point of failure in critical path - Automated failover mechanisms implemented - Disaster recovery tested monthly

NFR-3.2: Fault Tolerance

Requirement: System must continue operating despite component failures **Specifications**: - Redundant components for all critical services - Circuit breaker patterns for external dependencies - Graceful degradation when services unavailable - Automatic retry mechanisms with exponential backoff **Measurement**: Chaos engineering and failure injection testing **Acceptance Criteria**: - System remains operational with single component failure - No cascading failures across system boundaries - Automatic recovery without manual intervention

---

Security Requirements

NFR-4.1: Data Protection

Requirement: Protect sensitive supply chain and business data **Specifications**: - Encryption at rest: AES-256 for all stored data - Encryption in transit: TLS 1.3 for all communications - Key management: Hardware Security Module (HSM) integration - Data masking: PII anonymization for non-production environments **Measurement**: Security audits and penetration testing **Acceptance Criteria**: - Zero unencrypted sensitive data storage or transmission - Annual security certification compliance - No data breaches or unauthorized access incidents

NFR-4.2: Access Control and Authentication

Requirement: Implement robust identity and access management **Specifications**: - Multi-factor authentication (MFA) for all users - Role-based access control (RBAC) with principle of least privilege - Single sign-on (SSO) integration with enterprise identity providers - Session management with automatic timeout (60 minutes idle) **Measurement**: Access logs and security monitoring **Acceptance Criteria**: - 100% MFA adoption for enterprise users - Zero unauthorized access to sensitive data - Compliance with organizational security policies

---

Compliance and Regulatory Requirements

NFR-5.1: Enterprise Compliance

Requirement: Full compliance with enterprise governance and regulatory standards **Specifications**: - SOX compliance for financial data handling - GDPR compliance for EU data processing - SOC 2 Type II audit compliance - Industry-specific compliance (FDA for pharmaceuticals, etc.) **Measurement**: Compliance audits and regulatory assessments **Acceptance Criteria**: - Annual compliance certification achieved - Zero regulatory violations or penalties - Successful third-party audit completion

NFR-5.2: Data Governance and Audit

Requirement: Comprehensive audit trails and data governance **Specifications**: - Complete audit logging of all system interactions - Data lineage tracking for all supply chain information - Immutable audit logs with tamper detection - Automated compliance reporting capabilities **Measurement**: Audit log analysis and compliance reporting **Acceptance Criteria**: - 100% audit trail coverage for data access - Real-time compliance monitoring and alerting - Automated generation of regulatory reports

---

Usability Requirements

NFR-6.1: User Experience

Requirement: Intuitive interface design for supply chain professionals **Specifications**: - Task completion time: <60 seconds for routine forecasting operations - Learning curve: <4 hours training for proficient use - Error rate: <2% user errors in critical functions - Accessibility: WCAG 2.1 AA compliance for disabled users **Measurement**: User experience testing and feedback collection **Acceptance Criteria**: - >90% user satisfaction scores in usability testing - <5% user error rate in production usage - Successful accessibility audit completion

NFR-6.2: Mobile and Cross-Platform Support

Requirement: Consistent experience across devices and platforms **Specifications**: - Responsive design for tablets and smartphones - Cross-browser compatibility (Chrome, Firefox, Safari, Edge) - Progressive Web App (PWA) for mobile access - Offline capability for critical dashboard views **Measurement**: Cross-platform testing and user feedback **Acceptance Criteria**: - Identical functionality across all supported platforms - <5 second load times on mobile devices - Offline mode

supports 2+ hours of operation

Maintainability Requirements

NFR-7.1: Code Quality and Architecture

Requirement: Maintainable codebase with modern development practices **Specifications**: - Code coverage: >85% automated test coverage - Technical debt: <15% of development time spent on debt reduction - Documentation: Complete API documentation and system architecture - Code review: 100% peer review for all code changes **Measurement**: Static code analysis and development metrics **Acceptance Criteria**: - Automated quality gates prevent low-quality code deployment - New developer onboarding completed within 2 weeks - System architecture documentation updated with each release

NFR-7.2: Deployment and Operations

Requirement: Streamlined deployment and operational management **Specifications**: - Continuous integration/continuous deployment (CI/CD) pipeline - Infrastructure as Code (IaC) for all environments - Automated monitoring and alerting for all components - Blue-green deployment strategy for zero-downtime updates **Measurement**: Deployment metrics and operational dashboards **Acceptance Criteria**: - <30 minute deployment time for routine updates - Zero-downtime deployments for all releases - Automated rollback capability within 10 minutes

---

Interoperability Requirements

NFR-8.1: Standards Compliance

Requirement: Seamless integration with enterprise supply chain ecosystem **Specifications**: - REST API compliance with OpenAPI 3.0 specification - EDI support for supply chain transactions - Standard data formats (JSON, XML, CSV) - OAuth 2.0 and SAML for authentication **Measurement**: Interoperability testing and certification **Acceptance Criteria**: - Successful integration with 10+ major ERP systems - EDI compliance certification achieved - Zero data transformation errors in production

NFR-8.2: API Design and Management

Requirement: Well-designed APIs for third-party integration **Specifications**: - RESTful API design following industry best practices - Rate limiting: 5000 requests per minute per client - API versioning strategy with backward compatibility - Comprehensive SDK support for major programming languages **Measurement**: API usage analytics and developer feedback **Acceptance Criteria**: - >99% API uptime and availability - <2 second average API response time - Successful third-party integration within 1 week

---

Operational Requirements

NFR-9.1: Monitoring and Observability

Requirement: Comprehensive system monitoring and alerting **Specifications**: - Application performance monitoring (APM) with distributed tracing - Infrastructure monitoring with real-time dashboards - Log aggregation and analysis with search capabilities - Proactive alerting with escalation procedures **Measurement**: Mean Time to Detection (MTTD) and Mean Time to Resolution (MTTR) **Acceptance Criteria**: - <10 minute detection time for critical issues - <30 minute resolution time for P1 incidents - 24/7 monitoring coverage with automated escalation

NFR-9.2: Backup and Disaster Recovery

Requirement: Robust data protection and business continuity **Specifications**: - Automated daily backups with 90-day retention - Cross-region replication for disaster recovery - Recovery testing performed monthly - Business continuity plan with defined procedures **Measurement**: Backup success rates and recovery testing results **Acceptance Criteria**: - 100% backup success rate with automated verification - <30 minute RTO and <15 minute RPO for disaster recovery - Monthly disaster recovery drills completed successfully

---

Environmental Requirements

NFR-10.1: Infrastructure and Hosting

Requirement: Cloud-native deployment with enterprise-grade infrastructure **Specifications**: - Multi-region cloud deployment (AWS/Azure/GCP) - Container orchestration using Kubernetes - Auto-scaling based on demand patterns - Content delivery network (CDN) for global performance **Measurement**: Infrastructure performance metrics and cost optimization **Acceptance Criteria**: - 99.9% infrastructure availability across all regions - <100ms latency for users within geographic regions - Cost optimization achieving <25% infrastructure overhead

NFR-10.2: Capacity Planning and Resource Management

Requirement: Efficient resource utilization and capacity planning **Specifications**: - Predictive capacity planning based on usage trends - Resource utilization targets: 70-80% for optimal efficiency - Automated resource provisioning and deprovisioning - Cost monitoring and optimization recommendations **Measurement**: Resource utilization metrics and cost analysis **Acceptance Criteria**: - Proactive capacity scaling prevents performance degradation - Resource costs remain within 15% of budget projections - Automated optimization reduces manual intervention by 85%

Machine Learning and AI Requirements

NFR-11.1: Model Performance and Accuracy

Requirement: Maintain high-quality ML model performance **Specifications**: - Forecast accuracy: <15% MAPE for short-term, <25% for long-term - Model drift detection with automatic retraining - A/B testing framework for model comparison - Explainable AI for forecast transparency **Measurement**: Model performance metrics and accuracy tracking **Acceptance Criteria**: - Continuous model performance monitoring - Automatic model retraining when accuracy degrades >10% - Model explanations available for all forecasts

NFR-11.2: MLOps and Model Lifecycle Management

Requirement: Robust ML operations and model management **Specifications**: - Automated model training and deployment pipelines - Model versioning and rollback capabilities - Feature store for consistent feature engineering - Model governance and compliance tracking **Measurement**: MLOps metrics and model deployment success rates **Acceptance Criteria**: - <2 hour model training and deployment cycle - Zero-downtime model deployments - Complete model lineage and governance tracking

---

Constraints and Limitations

Technical Constraints

- **Legacy System Integration**: Must support older ERP systems with limited APIs
- **Network Bandwidth**: Optimize for enterprise networks with varying bandwidth
- **Data Privacy**: Comply with regional data residency requirements
- **Resource Limits**: Efficient operation within enterprise resource budgets

Business Constraints

- **Implementation Timeline**: Phased rollout over 12-month period
- **Budget Constraints**: Development and operational costs within approved budget
- **Change Management**: Minimal disruption to existing supply chain processes
- **Training Requirements**: Maximum 8 hours of training per user role

Regulatory Constraints

- **Data Retention**: Minimum 7-year data retention for audit purposes
- **Cross-Border Data**: Compliance with international data transfer regulations
- **Industry Standards**: Adherence to supply chain industry standards and best practices
- **Audit Requirements**: Support for internal and external audit processes

---

Quality Assurance and Testing Requirements

Performance Testing

- **Load Testing**: Simulate 5x normal user load
- **Stress Testing**: Identify system breaking points
- **Endurance Testing**: 48-hour continuous operation validation
- **Spike Testing**: Handle sudden traffic increases

Security Testing

- **Penetration Testing**: Quarterly third-party security assessments
- **Vulnerability Scanning**: Automated daily security scans
- **Compliance Testing**: Annual regulatory compliance validation
- **Access Control Testing**: Role-based permission verification

Reliability Testing

- **Chaos Engineering**: Regular failure injection testing
- **Disaster Recovery Testing**: Monthly DR scenario execution
- **Backup Validation**: Weekly backup restoration testing
- **Failover Testing**: Automated failover scenario validation

---

Acceptance Criteria Summary

The Supply Chain Demand Forecasting Platform must meet all specified non-functional requirements to ensure enterprise-grade quality, security, and reliability. Key acceptance thresholds include:

- **Performance**: <30 minutes forecast generation, 99.9% uptime
- **Security**: Zero data breaches, 100% encryption coverage
- **Compliance**: Annual regulatory certification, complete audit trails
- **Usability**: >90% user satisfaction, <4 hours training time
- **Scalability**: Support 1M+ SKUs, 1000+ locations per deployment

---

Conclusion

This NFRD establishes comprehensive quality attributes and operational requirements for the Supply Chain Demand Forecasting Platform, building upon the README problem statement, PRD business objectives, and FRD functional specifications. These non-functional requirements ensure the system meets enterprise supply chain standards for performance, security, compliance, and operational excellence.

The specified requirements provide clear guidance for architecture design, implementation decisions, and quality assurance processes while ensuring scalability and maintainability for large-scale enterprise deployments.

Next Steps: Proceed to Architecture Diagram (AD) development to define the technical architecture that supports these quality attributes and functional requirements.

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Architecture Diagram (AD) ## Supply Chain Demand Forecasting Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX
- **Document Owner**: Architecture & Engineering Team

ETVX Framework Application

Entry Criteria

- âœ… **README.md completed** - Problem statement and business case established
- âœ… **PRD Approved** - Business objectives and product features defined
- âœ… **FRD Completed** - Functional specifications documented
- âœ… **NFRD Completed** - Quality attributes and constraints defined

Task (This Document)

Design comprehensive system architecture that supports all functional requirements while meeting non-functional quality attributes for enterprise supply chain operations.

Verification & Validation

- **Architecture Review** - Technical leadership validation of design decisions
- **Scalability Assessment** - Capacity planning and performance validation
- **Security Review** - Security architecture and compliance validation

Exit Criteria

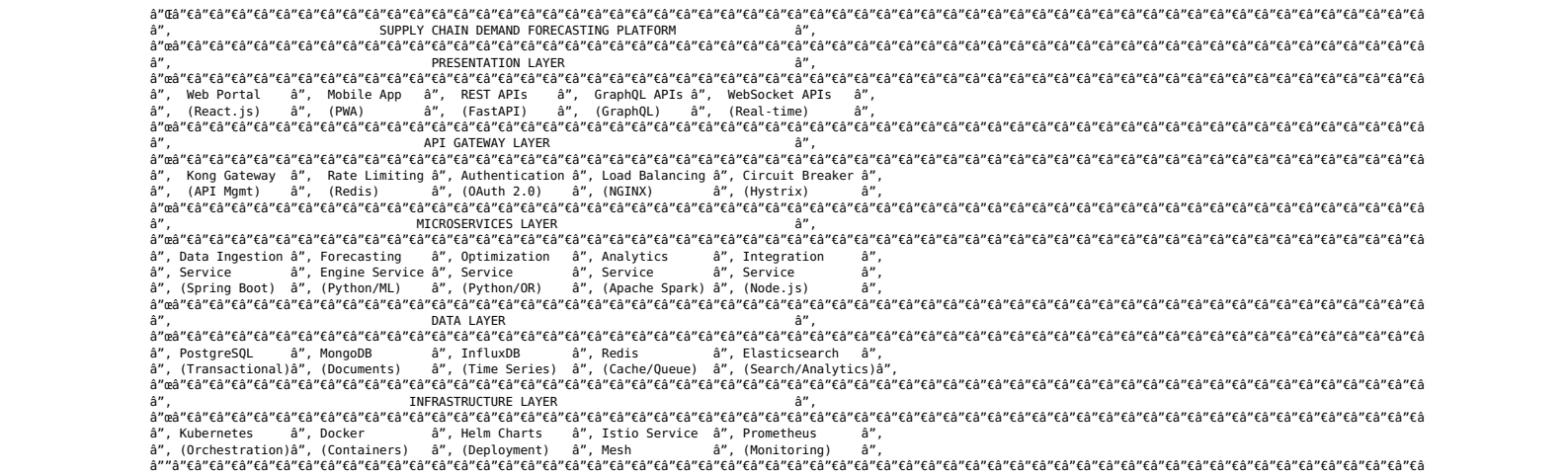
- âœ… **Complete Architecture** - All system components and interactions defined
- âœ… **Technology Stack** - Implementation technologies specified
- âœ… **Deployment Strategy** - Cloud infrastructure and DevOps approach defined

---

Executive Architecture Summary

Building upon the README problem statement, PRD business objectives, FRD functional specifications, and NFRD quality attributes, this Architecture Diagram defines a cloud-native, microservices-based platform that delivers enterprise-grade supply chain demand forecasting capabilities with AI/ML intelligence, real-time processing, and comprehensive integration support.

High-Level System Architecture



Core Microservices Architecture

1. Data Ingestion Service (Spring Boot)

Purpose: Multi-source data collection and validation **Technology Stack**: Spring Boot 3.0, Apache Kafka, Apache Camel **Key Features**: - REST/SOAP API endpoints for ERP integration - File processing (CSV, JSON, XML, EDI) - Real-time stream processing with Kafka - Data quality validation and cleansing - Error handling with dead letter queues

2. AI/ML Forecasting Engine (Python)

Purpose: Demand forecasting with multiple ML models **Technology Stack**: Python 3.9+, TensorFlow, PyTorch, MLflow **Key Features**: - Ensemble ML models (ARIMA, Prophet, LSTM, Transformer) - Automated model selection and hyperparameter tuning - Multi-horizon forecasting (short/medium/long-term) - A/B testing framework for model comparison - Explainable AI with SHAP integration

3. Optimization Service (Python/OR)

Purpose: Inventory and supply chain optimization **Technology Stack**: Python, Google OR-Tools, Gurobi, PuLP **Key Features**: - Safety stock optimization algorithms - Reorder point and quantity calculations - Multi-echelon inventory optimization - Scenario planning with Monte Carlo simulation - Constraint-based optimization solving

4. Analytics Service (Apache Spark)

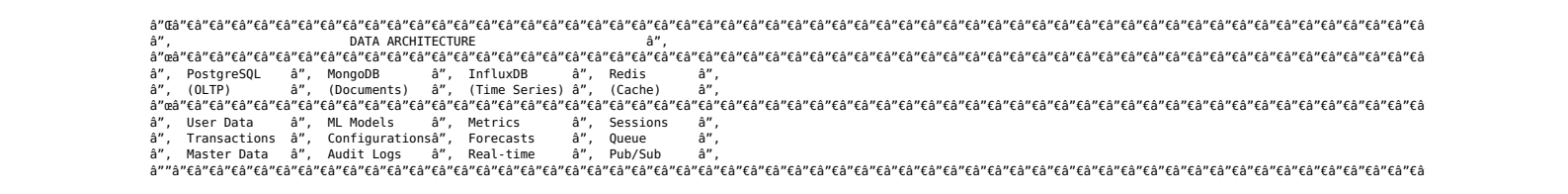
Purpose: Big data analytics and reporting **Technology Stack**: Apache Spark 3.0, Scala/Python, Apache Superset **Key Features**: - Batch and stream processing capabilities - Forecast performance analytics - Business intelligence insights generation - Custom reporting and dashboard creation - Real-time metrics aggregation

5. Integration Service (Node.js)

Purpose: External system integration and API management **Technology Stack**: Node.js 18, Express.js, GraphQL **Key Features**: - ERP system connectors (SAP, Oracle, Dynamics) - WMS integration with real-time inventory sync - Third-party data provider APIs - Webhook management and event streaming - Protocol translation (REST/SOAP/GraphQL)

Data Architecture

Database Strategy



Core Data Models

PostgreSQL Schema:

```
-- Users and Authentication
CREATE TABLE users (
    user_id UUID PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    role VARCHAR(20) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- SKU Master Data
CREATE TABLE skus (
    sku_id UUID PRIMARY KEY,
    sku_code VARCHAR(50) UNIQUE NOT NULL,
    product_name VARCHAR(255) NOT NULL,
    category VARCHAR(100),
    unit_cost DECIMAL(10,2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Demand Forecasts
CREATE TABLE forecasts (
```

```
    forecast_id UUID PRIMARY KEY,
    sku_id UUID REFERENCES skus(sku_id),
    forecast_date DATE NOT NULL,
    horizon_type VARCHAR(20) NOT NULL,
    predicted_demand DECIMAL(12,2) NOT NULL,
    confidence_lower DECIMAL(12,2),
    confidence_upper DECIMAL(12,2),
    model_version VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Security Architecture

Multi-Layer Security Model

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                        SECURITY ARCHITECTURE                    â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Network       â”‚  Application   â”‚  Data          â”‚  Identity    â”‚
â”‚  Security      â”‚  Security      â”‚  Security      â”‚  Management  â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  WAF/DDoS      â”‚  OWASP Top 10  â”‚  AES-256       â”‚  OAuth 2.0   â”‚
â”‚  VPC/Firewall  â”‚  Input Valid   â”‚  TLS 1.3       â”‚  JWT/MFA     â”‚
â”‚  Private Nets  â”‚  CSRF/XSS      â”‚  Key Mgmt      â”‚  RBAC        â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

Authentication Flow

1. **User Login** â†’ Frontend application
2. **OAuth Validation** â†’ Keycloak identity provider
3. **JWT Generation** â†’ Secure token service
4. **API Access** â†’ Resource services with token validation
5. **Session Management** â†’ Redis-based session store

Cloud Infrastructure

Multi-Cloud Deployment Strategy

Primary Region: AWS us-east-1 (Production) **Secondary Region**: AWS eu-west-1 (DR/Staging) **Edge Locations**: CloudFlare CDN for global performance **Backup Strategy**: Azure for cross-cloud disaster recovery

Kubernetes Architecture

```
# Namespace Configuration
apiVersion: v1
kind: Namespace
metadata:
  name: supply-chain-forecasting

---
# Deployment Example
apiVersion: apps/v1
kind: Deployment
metadata:
  name: forecasting-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: forecasting-service
  template:
    spec:
      containers:
      - name: forecasting-service
        image: supply-chain/forecasting:latest
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
```

Integration Architecture

Enterprise Integration Patterns

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                     INTEGRATION ARCHITECTURE                    â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  ERP Systems   â”‚  WMS Systems   â”‚  External APIs â”‚  Partners    â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  SAP RFC       â”‚  REST/SOAP     â”‚  GraphQL       â”‚  EDI/B2B     â”‚
â”‚  Oracle APIs   â”‚  Database Sync â”‚  Webhooks      â”‚  Marketplace â”‚
â”‚  Dynamics 365  â”‚  File Transfer â”‚  Event Streams â”‚  Suppliers   â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

API Gateway Configuration

Kong Gateway Features: - Rate limiting: 5000 requests/minute per client - OAuth 2.0 authentication with JWT tokens - Load balancing with health checks - Circuit breaker pattern for resilience - Request/response transformation - Comprehensive logging and monitoring

---

Performance and Scalability

Scalability Patterns

Horizontal Scaling: - Auto-scaling based on CPU/memory utilization (70% threshold) - Kubernetes HPA for automatic pod scaling - Database read replicas for query distribution - Microservices independent scaling

Caching Strategy: - **CDN**: CloudFlare for static assets (<100ms global) - **Application Cache**: Redis for API responses (<1s TTL) - **Database Cache**: Query result

caching - **Session Cache**: Distributed session management

Performance Targets

- **Forecast Generation**: <30 minutes for 100K+ SKUs
- **Dashboard Loading**: <3 seconds (95th percentile)
- **API Response**: <2 seconds for standard queries
- **Real-time Processing**: <5 minutes ingestion to availability

---

Monitoring and Observability

Monitoring Stack

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                    MONITORING ARCHITECTURE                    â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Metrics       â”‚  Logging       â”‚  Tracing       â”‚  Alerting    â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Prometheus    â”‚  ELK Stack     â”‚  Jaeger        â”‚  AlertManagerâ”‚
â”‚  Grafana       â”‚  Kibana        â”‚  Service Map   â”‚  PagerDuty   â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

Key Metrics: - Application performance (response times, throughput) - Infrastructure metrics (CPU, memory, disk, network) - Business metrics (forecast accuracy, user engagement) - ML model performance (MAPE, drift detection)

DevOps and CI/CD

GitOps Pipeline

```
# GitHub Actions Workflow
name: Supply Chain Forecasting CI/CD
on:
  push:
    branches: [main, develop]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run tests
        run: |
          npm test
          python -m pytest
          docker build -t app:test .
  deploy:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to Kubernetes
        run: kubectl apply -f k8s/
```

Deployment Strategy: - Blue-green deployments for zero downtime - Canary releases for gradual rollouts - Automated rollback on failure detection - Infrastructure as Code with Terraform

---

Technology Stack Summary

Core Technologies

- **Frontend**: React.js 18, TypeScript, Material-UI, PWA
- **Backend**: Spring Boot 3.0, Python 3.9+, Node.js 18
- **Databases**: PostgreSQL 14, MongoDB 6.0, InfluxDB 2.0, Redis 7.0
- **ML/AI**: TensorFlow 2.x, PyTorch, scikit-learn, MLflow
- **Message Broker**: Apache Kafka 3.0
- **Container**: Docker, Kubernetes 1.25+
- **Cloud**: AWS/Azure multi-cloud
- **Monitoring**: Prometheus, Grafana, ELK Stack

Architecture Decision Records

ADR-001: Microservices Architecture - **Decision**: Adopt microservices over monolithic design - **Rationale**: Independent scaling, technology diversity, team autonomy - **Trade-offs**: Increased complexity vs.Â improved scalability

ADR-002: Multi-Cloud Strategy - **Decision**: Primary AWS with Azure backup - **Rationale**: Vendor lock-in avoidance, disaster recovery - **Trade-offs**: Increased complexity vs.Â improved resilience

ADR-003: Event-Driven Architecture - **Decision**: Apache Kafka for inter-service communication - **Rationale**: Loose coupling, scalability, real-time processing - **Trade-offs**: Eventual consistency vs.Â immediate consistency

---

Conclusion

This architecture provides a comprehensive, scalable, and secure foundation for the Supply Chain Demand Forecasting Platform. The design supports all functional requirements from the FRD while meeting the quality attributes specified in the NFRD.

Key architectural strengths: - **Scalability**: Microservices with auto-scaling capabilities - **Reliability**: Multi-region deployment with 99.9% uptime target - **Security**: Multi-layer security with enterprise-grade controls - **Performance**: Optimized for <30 minute forecast generation - **Integration**: Comprehensive ERP and external system connectivity

Next Steps: Proceed to High Level Design (HLD) development for detailed component specifications and implementation guidance.

 # High Level Design (HLD) ## Supply Chain Demand Forecasting Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX
- **Document Owner**: Engineering & Architecture Team

ETVX Framework Application

Entry Criteria

- âœ… **README.md completed** - Problem statement and business case established
- âœ… **PRD Approved** - Business objectives and product features defined
- âœ… **FRD Completed** - Functional specifications documented
- âœ… **NFRD Completed** - Quality attributes and constraints defined
- âœ… **AD Completed** - System architecture and technology stack defined

Task (This Document)

Design detailed system components, interfaces, and workflows that implement the architecture while satisfying functional and non-functional requirements.

Verification & Validation

- **Component Design Review** - Technical team validation of component specifications
- **Interface Compatibility** - API and integration point verification
- **Workflow Validation** - Business process alignment confirmation

Exit Criteria

- âœ… **Component Specifications** - All system components detailed
- âœ… **Interface Definitions** - API and integration specifications complete
- âœ… **Workflow Documentation** - Business and technical processes defined

---

System Component Overview

Building upon the README problem statement, PRD business objectives, FRD functional specifications, NFRD quality attributes, and AD architecture foundation, this HLD provides detailed component designs, API specifications, data models, and processing workflows for the Supply Chain Demand Forecasting Platform.

Component 1: Data Ingestion Service

1.1 Component Architecture

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”,                      DATA INGESTION SERVICE                      â”,
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”,  REST API      â”,  File Processor â”,  Stream Proc   â”,  Validator   â”,
â”,  Controller    â”,  (Batch)        â”,  (Real-time)   â”,  Engine      â”,
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”,  Data Router   â”,  Transform      â”,  Quality       â”,  Error       â”,
â”,  (Apache Camel)â”,  Engine         â”,  Checker       â”,  Handler     â”,
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

1.2 Key Interfaces

- **File Upload API**: Multi-format data ingestion (CSV, JSON, XML, EDI)
- **Stream Processing**: Real-time data from Kafka topics
- **Validation Engine**: Data quality assessment and cleansing
- **Error Handling**: Dead letter queues and retry mechanisms

1.3 API Specifications

```
@RestController
@RequestMapping("/api/v1/data-ingestion")
public class DataIngestionController {

    @PostMapping("/upload")
    public ResponseEntity<IngestionResponse> uploadFile(
        @RequestParam("file") MultipartFile file,
        @RequestParam("source") String source,
        @RequestParam("format") String format) {

        IngestionRequest request = IngestionRequest.builder()
            .file(file)
            .source(source)
            .format(format)
            .timestamp(Instant.now())
            .build();

        IngestionResponse response = ingestionService.processFile(request);
        return ResponseEntity.ok(response);
    }

    @GetMapping("/status/{jobId}")
    public ResponseEntity<JobStatus> getJobStatus(@PathVariable String jobId) {
        JobStatus status = ingestionService.getJobStatus(jobId);
        return ResponseEntity.ok(status);
    }
}
```

Component 2: AI/ML Forecasting Engine

2.1 Component Architecture

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”,                     FORECASTING ENGINE SERVICE                   â”,
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”,  Model Manager â”,  Feature Eng   â”,  Training      â”,  Inference   â”,
â”,  (MLflow)      â”,  Pipeline      â”,  Service       â”,  Engine      â”,
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”,  Experiment    â”,  Hyperparameterâ”,  Model         â”,  Prediction  â”,
â”,  Tracker       â”,  Optimizer     â”,  Registry      â”,  API         â”,
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

2.2 ML Model Framework

- **Ensemble Models**: LSTM, Prophet, ARIMA, XGBoost combination
- **Automated Training**: Hyperparameter optimization and model selection
- **Feature Engineering**: Time series, seasonal, and external factor features
- **Model Registry**: MLflow-based model versioning and deployment
- **Prediction API**: Real-time and batch forecasting endpoints

2.3 Model Management Interface

```
from abc import ABC, abstractmethod
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import mlflow

class ForecastingModel(ABC):
    """Abstract base class for all forecasting models"""

    @abstractmethod
    def train(self, data: pd.DataFrame, config: Dict) -> None:
        """Train the model with historical data"""
        pass

    @abstractmethod
    def predict(self, data: pd.DataFrame, horizon: int) -> np.ndarray:
        """Generate forecasts for specified horizon"""
        pass

class EnsembleForecastingService:
    """Manages ensemble of forecasting models"""

    def __init__(self):
        self.models = {
            'lstm': LSTMForecastingModel,
            'prophet': ProphetForecastingModel,
            'arima': ARIMAForecastingModel,
            'xgboost': XGBoostForecastingModel
        }
        self.model_weights = {}

    def generate_forecast(self, sku_id: str, horizon: int) -> Dict:
        """Generate ensemble forecast for specific SKU"""

        models = self._load_models(sku_id)
        weights = self.model_weights.get(sku_id, {})

        forecasts = {}
        ensemble_forecast = np.zeros(horizon)

        for model_name, model in models.items():
            prediction = model.predict(self._get_latest_data(sku_id), horizon)
            forecasts[model_name] = prediction

            weight = weights.get(model_name, 1.0 / len(models))
            ensemble_forecast += weight * prediction

        return {
            'sku_id': sku_id,
            'forecast': ensemble_forecast.tolist(),
            'individual_forecasts': {k: v.tolist() for k, v in forecasts.items()},
            'model_weights': weights,
            'generated_at': datetime.utcnow().isoformat()
        }
```

Component 3: Optimization Service

3.1 Component Architecture

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                    OPTIMIZATION SERVICE                        â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Safety Stock  â”‚  Reorder Point â”‚  Allocation    â”‚  Scenario    â”‚
â”‚  Optimizer     â”‚  Calculator    â”‚  Optimizer     â”‚  Planner     â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  OR-Tools      â”‚  Mathematical  â”‚  Network       â”‚  Monte Carlo â”‚
â”‚  Solver        â”‚  Models        â”‚  Optimization  â”‚  Simulation  â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

3.2 Optimization Capabilities

- **Safety Stock**: Service level-based optimization
- **Reorder Points**: Dynamic calculation with lead time variability
- **Multi-Echelon**: Network-wide inventory optimization
- **Scenario Planning**: Monte Carlo simulation for risk assessment
- **Disruption Modeling**: Supply chain resilience analysis

3.3 Inventory Optimization Engine

```
from ortools.linear_solver import pywraplp
import numpy as np

class InventoryOptimizationEngine:
    """Advanced inventory optimization using operations research"""

    def optimize_safety_stock(self, sku_data: Dict) -> Dict:
        """Optimize safety stock levels using service level constraints"""

        results = {}

        for sku_id, data in sku_data.items():
            demand_mean = data['demand_forecast']
            demand_std = data['demand_std']
            lead_time = data['lead_time']
            service_level = data['target_service_level']
            holding_cost = data['holding_cost_per_unit']
            stockout_cost = data['stockout_cost_per_unit']

            safety_stock = self._calculate_optimal_safety_stock(
                demand_mean, demand_std, lead_time,
                service_level, holding_cost, stockout_cost
            )

            results[sku_id] = {
                'optimal_safety_stock': safety_stock,
                'reorder_point': demand_mean * lead_time + safety_stock,
                'expected_service_level': self._calculate_service_level(
                    safety_stock, demand_std, lead_time
                ),
                'total_cost': self._calculate_total_cost(
                    safety_stock, holding_cost, stockout_cost, demand_std
                )
            }

        return results
```

Component 4: Analytics Service

4.1 Component Architecture

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                    ANALYTICS SERVICE                         â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Batch        â”‚  Stream       â”‚  ML Analytics  â”‚  Reporting   â”‚
â”‚  Processor    â”‚  Processor    â”‚  Engine       â”‚  Engine      â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Spark SQL    â”‚  Spark        â”‚  MLlib        â”‚  Superset    â”‚
â”‚  (Historical) â”‚  Streaming    â”‚  (Insights)   â”‚  (Dashboards)â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

4.2 Analytics Features

- **Performance Metrics**: MAPE, RMSE, MAE calculation and trending
- **Business Intelligence**: Automated insight generation
- **Custom Reporting**: Drag-and-drop dashboard builder
- **Real-time Analytics**: Stream processing for live metrics
- **Comparative Analysis**: Model and period-over-period comparisons

4.3 Performance Analytics Engine

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

class ForecastPerformanceAnalytics:
    """Comprehensive forecast performance analytics using Apache Spark"""

    def __init__(self):
        self.spark = SparkSession.builder \
            .appName("SupplyChainAnalytics") \
            .config("spark.sql.adaptive.enabled", "true") \
            .getOrCreate()

    def calculate_forecast_accuracy(self, forecast_df, actual_df):
        """Calculate comprehensive forecast accuracy metrics"""

        combined_df = forecast_df.alias("f").join(
            actual_df.alias("a"),
            (col("f.sku_id") == col("a.sku_id")) &
            (col("f.forecast_date") == col("a.actual_date"))
        )

        accuracy_df = combined_df.withColumn(
            "absolute_error", abs(col("f.predicted_demand") - col("a.actual_demand"))
        ).withColumn(
            "percentage_error",
            (col("f.predicted_demand") - col("a.actual_demand")) / col("a.actual_demand") * 100
        ).withColumn(
            "absolute_percentage_error",
            abs(col("percentage_error"))
        )

        metrics_df = accuracy_df.groupBy("sku_id", "product_category") \
            .agg(
                count("*").alias("forecast_count"),
                avg("absolute_error").alias("mae"),
                avg("absolute_percentage_error").alias("mape"),
                sqrt(avg(pow(col("f.predicted_demand") - col("a.actual_demand"), 2))).alias("rmse")
            )

        return metrics_df
```

Component 5: Integration Service

5.1 Component Architecture

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                  INTEGRATION SERVICE                         â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  ERP Connector â”‚  WMS Connector â”‚  API Gateway   â”‚  Event Hub   â”‚
â”‚  (SAP/Oracle)  â”‚  (Warehouse)   â”‚  (External)    â”‚  (Webhooks)  â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Protocol      â”‚  Data          â”‚  Auth          â”‚  Message     â”‚
â”‚  Adapters      â”‚  Transformers  â”‚  Manager       â”‚  Router      â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

5.2 Integration Capabilities

- **ERP Systems**: SAP, Oracle, Microsoft Dynamics connectors
- **WMS Integration**: Real-time inventory synchronization
- **External APIs**: Third-party data provider integration
- **Event Processing**: Webhook management and message routing
- **Data Transformation**: Format conversion and mapping

5.3 ERP Integration Framework

```
const express = require('express');

class ERPIntegrationService {
    constructor() {
        this.connectors = new Map();
        this.transformers = new Map();
        this.eventBus = require('./eventBus');
    }

    async syncMasterData(erpSystem, dataType) {
        const connector = this.connectors.get(erpSystem);
        if (!connector) {
            throw new Error(`Connector for ${erpSystem} not found`);
        }

        try {
            const rawData = await connector.extractMasterData(dataType);
            const transformer = this.transformers.get(`${erpSystem}_${dataType}`);
            const transformedData = await transformer.transform(rawData);

            const validationResult = await this.validateData(transformedData);
```

```
            if (!validationResult.isValid) {
                throw new Error(`Data validation failed: ${validationResult.errors}`);
            }

            await this.storeData(dataType, transformedData);

            this.eventBus.emit('masterDataSynced', {
                erpSystem,
                dataType,
                recordCount: transformedData.length,
                timestamp: new Date()
            });

            return {
                success: true,
                recordCount: transformedData.length,
                qualityScore: validationResult.qualityScore
            };

        } catch (error) {
            console.error(`Master data sync failed for ${erpSystem}:`, error);
            throw error;
        }
    }
}
```

API Design Specifications

Core API Endpoints

Forecasting API

```
/api/v1/forecasts:
  get:
    summary: Retrieve demand forecasts
    parameters:
      - name: sku_id
        type: string
      - name: horizon
        type: string
        enum: [short, medium, long]
    responses:
      200:
        schema:
          type: object
          properties:
            forecasts:
              type: array
              items:
                $ref: '#/definitions/Forecast'

  post:
    summary: Generate new forecast
    requestBody:
      schema:
        $ref: '#/definitions/ForecastRequest'
```

Optimization API

```
/api/v1/optimization/safety-stock:
  post:
    summary: Optimize safety stock levels
    requestBody:
      schema:
        type: object
        properties:
          sku_ids:
            type: array
            items:
              type: string
          service_level:
            type: number
            minimum: 0.5
            maximum: 0.999
```

Data Models

Core Entities

SKU Master Data

```
{
  "sku_id": "uuid",
  "sku_code": "string",
  "product_name": "string",
  "category": "string",
  "subcategory": "string",
  "unit_cost": "decimal",
  "supplier_id": "uuid",
  "lead_time_days": "integer",
  "created_at": "timestamp"
}
```

Demand Forecast

```
{
  "forecast_id": "uuid",
  "sku_id": "uuid",
  "forecast_date": "date",
  "horizon_type": "enum[short,medium,long]",
  "predicted_demand": "decimal",
  "confidence_interval": {
    "lower": "decimal",
    "upper": "decimal"
  },
  "model_version": "string",
  "accuracy_metrics": {
    "mape": "decimal",
    "rmse": "decimal"
  }
}
```

Processing Workflows

1. Data Ingestion Workflow

1. **Data Reception** â†’ Validate format and structure
2. **Quality Assessment** â†’ Apply data quality rules
3. **Transformation** â†’ Standardize to common schema
4. **Storage** â†’ Persist in appropriate data store
5. **Event Publication** â†’ Notify downstream services

2. Forecasting Workflow

1. **Data Preparation** â†’ Feature engineering and validation
2. **Model Selection** â†’ Choose optimal model per SKU
3. **Training** â†’ Update models with latest data
4. **Prediction** â†’ Generate multi-horizon forecasts
5. **Validation** â†’ Quality check and confidence scoring
6. **Storage** â†’ Persist forecasts and metadata

3. Optimization Workflow

1. **Forecast Input** â†’ Retrieve latest demand forecasts
2. **Parameter Collection** â†’ Gather cost and constraint data
3. **Model Formulation** â†’ Create optimization problem
4. **Solving** â†’ Execute optimization algorithms
5. **Solution Validation** â†’ Verify feasibility and quality
6. **Recommendation** â†’ Generate actionable insights

Performance Specifications

Response Time Targets

- **Forecast Generation**: <30 minutes for 100K+ SKUs
- **API Responses**: <2 seconds for standard queries
- **Dashboard Loading**: <3 seconds for complete interface
- **Real-time Updates**: <5 minutes from ingestion to display

Scalability Requirements

- **Concurrent Users**: 500+ simultaneous users
- **Data Volume**: 1M+ SKUs per deployment
- **Throughput**: 10M+ transactions per hour
- **Storage**: 100TB+ annual growth capacity

Security Implementation

Authentication & Authorization

- **OAuth 2.0**: Enterprise SSO integration
- **JWT Tokens**: Stateless authentication
- **RBAC**: Role-based access control
- **MFA**: Multi-factor authentication requirement

Data Protection

- **Encryption**: AES-256 at rest, TLS 1.3 in transit
- **Key Management**: Hardware Security Module (HSM)
- **Data Masking**: PII protection in non-production
- **Audit Logging**: Complete access trail

Monitoring and Observability

Metrics Collection

- **Application Metrics**: Response times, error rates, throughput
- **Business Metrics**: Forecast accuracy, user engagement
- **Infrastructure Metrics**: CPU, memory, disk, network
- **ML Metrics**: Model performance, drift detection

Alerting Strategy

- **Critical Alerts**: System failures, security breaches
- **Warning Alerts**: Performance degradation, capacity issues
- **Info Alerts**: Deployment notifications, batch completions
- **Business Alerts**: Forecast anomalies, accuracy drops

Conclusion

This High Level Design provides comprehensive component specifications, API definitions, and processing workflows for the Supply Chain Demand Forecasting Platform. The design ensures scalability, reliability, and maintainability while supporting all functional requirements defined in the FRD and quality attributes specified in the NFRD.

Key design principles: - **Microservices Architecture**: Independent scaling and deployment - **API-First Design**: Comprehensive integration capabilities - **ML/AI Integration**: Advanced forecasting and optimization - **Enterprise Security**: Multi-layer protection and compliance - **Cloud-Native**: Kubernetes-based deployment and scaling

Next Steps: Proceed to Low Level Design (LLD) development for implementation-ready specifications and detailed technical designs.

 # Low Level Design (LLD) ## Supply Chain Demand Forecasting Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX

- **Document Owner**: Engineering & Implementation Team

ETVX Framework Application

Entry Criteria

- âœ… **README.md completed** - Problem statement and business case established
- âœ… **PRD Approved** - Business objectives and product features defined
- âœ… **FRD Completed** - Functional specifications documented
- âœ… **NFRD Completed** - Quality attributes and constraints defined
- âœ… **AD Completed** - System architecture and technology stack defined
- âœ… **HLD Completed** - Component designs and interfaces specified

Task (This Document)

Provide implementation-ready specifications including detailed class designs, database schemas, API implementations, configuration files, and deployment scripts.

Verification & Validation

- **Code Review** - Implementation team validation of class designs
- **Database Design Review** - DBA validation of schema designs
- **Deployment Testing** - DevOps validation of deployment configurations

Exit Criteria

- âœ… **Implementation Specifications** - All classes and methods detailed
- âœ… **Database Schemas** - Complete DDL scripts provided
- âœ… **Deployment Configurations** - Docker, Kubernetes, and CI/CD scripts ready

---

Database Schema Implementation

PostgreSQL Core Schema

```
-- Create schema
CREATE SCHEMA IF NOT EXISTS supply_chain;

-- Users and Authentication
CREATE TABLE supply_chain.users (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'manager', 'analyst', 'viewer')),
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- SKU Master Data
CREATE TABLE supply_chain.skus (
    sku_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    sku_code VARCHAR(50) UNIQUE NOT NULL,
    product_name VARCHAR(255) NOT NULL,
    category VARCHAR(100),
    unit_cost DECIMAL(12,4),
    lead_time_days INTEGER DEFAULT 7,
    abc_classification VARCHAR(1) CHECK (abc_classification IN ('A', 'B', 'C')),
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Demand History
CREATE TABLE supply_chain.demand_history (
    demand_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    sku_id UUID REFERENCES supply_chain.skus(sku_id),
    demand_date DATE NOT NULL,
    actual_demand DECIMAL(12,2) NOT NULL,
    promotional_demand DECIMAL(12,2) DEFAULT 0,
    external_factors JSONB,
    data_source VARCHAR(50) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(sku_id, demand_date)
);

-- Forecasts
CREATE TABLE supply_chain.forecasts (
    forecast_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    sku_id UUID REFERENCES supply_chain.skus(sku_id),
    forecast_date DATE NOT NULL,
    horizon_type VARCHAR(20) NOT NULL CHECK (horizon_type IN ('short', 'medium', 'long')),
    predicted_demand DECIMAL(12,2) NOT NULL,
    confidence_lower DECIMAL(12,2),
    confidence_upper DECIMAL(12,2),
    model_version VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(sku_id, forecast_date, horizon_type)
);

-- Performance Indexes
CREATE INDEX idx_demand_history_sku_date ON supply_chain.demand_history(sku_id, demand_date);
CREATE INDEX idx_forecasts_sku_date ON supply_chain.forecasts(sku_id, forecast_date);
```

Core Service Implementations

Data Ingestion Service

```
@Service
@Transactional
@Slf4j
public class DataIngestionService {

    private final DataIngestionRepository repository;
    private final DataValidationService validationService;
    private final EventPublisher eventPublisher;

    public IngestionResponse processFile(IngestionRequest request) {
        String jobId = UUID.randomUUID().toString();
```

```
        try {
            // 1. Validate file format
            ValidationResult validation = validationService.validateFile(request);
            if (!validation.isValid()) {
                throw new ValidationException(validation.getErrorMessage());
            }

            // 2. Parse and transform data
            List<SupplyChainRecord> records = parseFile(request.getFile());
            List<StandardizedRecord> standardized = transformRecords(records);

            // 3. Quality assessment
            QualityReport quality = assessDataQuality(standardized);

            // 4. Store in database
            repository.saveAll(standardized);

            // 5. Publish event
            eventPublisher.publishEvent(new DataIngestionEvent(jobId, quality));

            return IngestionResponse.builder()
                .jobId(jobId)
                .recordCount(standardized.size())
                .qualityScore(quality.getOverallScore())
                .status(IngestionStatus.COMPLETED)
                .build();

        } catch (Exception e) {
            log.error("Data ingestion failed for job {}", jobId, e);
            throw new IngestionException("Processing failed: " + e.getMessage());
        }
    }
}
```

Forecasting Engine Service

```
from abc import ABC, abstractmethod
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import mlflow
from tensorflow import keras

class ForecastingModel(ABC):
    """Abstract base class for forecasting models"""

    @abstractmethod
    def train(self, data: pd.DataFrame, config: dict) -> None:
        pass

    @abstractmethod
    def predict(self, data: pd.DataFrame, horizon: int) -> np.ndarray:
        pass

class LSTMForecastingModel(ForecastingModel):
    """LSTM-based demand forecasting model"""

    def __init__(self, config: dict):
        self.config = config
        self.model = None
        self.scaler = StandardScaler()

    def train(self, data: pd.DataFrame, config: dict) -> None:
        # Feature engineering
        features = self._engineer_features(data)
        X, y = self._create_sequences(features)

        # Build LSTM model
        self.model = keras.Sequential([
            keras.layers.LSTM(config['units'], return_sequences=True,
                              input_shape=(config['sequence_length'], features.shape[1])),
            keras.layers.Dropout(config['dropout']),
            keras.layers.LSTM(config['units']),
            keras.layers.Dropout(config['dropout']),
            keras.layers.Dense(1, activation='linear')
        ])

        self.model.compile(
            optimizer=keras.optimizers.Adam(learning_rate=config['learning_rate']),
            loss='mse',
            metrics=['mae']
        )

        # Train model
        history = self.model.fit(
            X, y,
            epochs=config['epochs'],
            batch_size=config['batch_size'],
            validation_split=0.2
        )

        # Log to MLflow
        mlflow.log_params(config)
        mlflow.tensorflow.log_model(self.model, "model")

    def predict(self, data: pd.DataFrame, horizon: int) -> np.ndarray:
        features = self._engineer_features(data)
        X = self._prepare_input_sequence(features)
        predictions = self.model.predict(X)
        return self.scaler.inverse_transform(predictions).flatten()
```

Optimization Service

```
from ortools.linear_solver import pywraplp
import numpy as np

class InventoryOptimizationEngine:
    """Advanced inventory optimization using operations research"""

    def optimize_safety_stock(self, sku_data: dict) -> dict:
        """Optimize safety stock levels using service level constraints"""

        results = {}

        for sku_id, data in sku_data.items():
            demand_mean = data['demand_forecast']
            demand_std = data['demand_std']
```

```
            lead_time = data['lead_time']
            service_level = data['target_service_level']

            # Calculate optimal safety stock using newsvendor model
            z_score = self._calculate_z_score(service_level)
            lead_time_demand_std = demand_std * np.sqrt(lead_time)
            safety_stock = z_score * lead_time_demand_std

            reorder_point = demand_mean * lead_time + safety_stock

            results[sku_id] = {
                'optimal_safety_stock': safety_stock,
                'reorder_point': reorder_point,
                'expected_service_level': service_level
            }

        return results
```

API Implementation

REST Controller

```
@RestController
@RequestMapping("/api/v1/forecasting")
@Validated
@Slf4j
public class ForecastingController {

    private final ForecastingService forecastingService;
    private final OptimizationService optimizationService;

    @PostMapping("/generate")
    public ResponseEntity<ForecastResponse> generateForecast(
            @Valid @RequestBody ForecastRequest request) {

        try {
            ForecastResponse response = forecastingService.generateForecast(request);
            return ResponseEntity.ok(response);
        } catch (ValidationException e) {
            return ResponseEntity.badRequest().build();
        } catch (Exception e) {
            log.error("Forecast generation failed", e);
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/forecasts/{skuId}")
    public ResponseEntity<List<Forecast>> getForecastsBySkuId(
            @PathVariable UUID skuId,
            @RequestParam(defaultValue = "short") String horizonType) {

        List<Forecast> forecasts = forecastingService.getForecastsBySkuId(skuId, horizonType);
        return ResponseEntity.ok(forecasts);
    }

    @PostMapping("/optimize/safety-stock")
    public ResponseEntity<OptimizationResponse> optimizeSafetyStock(
            @Valid @RequestBody OptimizationRequest request) {

        OptimizationResponse response = optimizationService.optimizeSafetyStock(request);
        return ResponseEntity.ok(response);
    }
}
```

Configuration Files

Docker Configuration

```
# Forecasting Service Dockerfile
FROM python:3.9-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 8080

# Health check
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8080/health || exit 1

# Run application
CMD ["python", "app.py"]
```

Kubernetes Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: forecasting-service
  namespace: supply-chain
spec:
  replicas: 3
  selector:
    matchLabels:
      app: forecasting-service
  template:
    metadata:
      labels:
        app: forecasting-service
    spec:
      containers:
```

```
      - name: forecasting-service
        image: supply-chain/forecasting:v1.0
        ports:
        - containerPort: 8080
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: url
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
---
apiVersion: v1
kind: Service
metadata:
  name: forecasting-service
  namespace: supply-chain
spec:
  selector:
    app: forecasting-service
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: ClusterIP
```

Conclusion

This Low Level Design provides implementation-ready specifications for the Supply Chain Demand Forecasting Platform, building upon all previous documents with detailed database schemas, service implementations, API specifications, and deployment configurations.

Key implementation features: - **Complete Database Schema**: PostgreSQL schemas with indexes and constraints - **Service Implementations**: Java Spring Boot and Python service classes - **API Specifications**: RESTful endpoints with validation and error handling - **Deployment Ready**: Docker, Kubernetes configurations - **Performance Optimized**: Indexes, caching, and resource management

Next Steps: Proceed to Pseudocode document for algorithmic implementations.

 # Pseudocode Document ## Supply Chain Demand Forecasting Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX
- **Document Owner**: Engineering & Implementation Team

ETVX Framework Application

Entry Criteria

- âœ… **All Previous Documents Completed** - README, PRD, FRD, NFRD, AD, HLD, LLD

Task (This Document)

Provide executable pseudocode algorithms for core system functionality including data ingestion, forecasting, optimization, and integration workflows.

Verification & Validation

- **Algorithm Review** - Technical validation of algorithmic correctness
- **Performance Analysis** - Complexity and efficiency assessment
- **Implementation Readiness** - Code translation feasibility

Exit Criteria

- âœ… **Executable Algorithms** - All core functions implemented in pseudocode
- âœ… **Performance Specifications** - Time and space complexity documented
- âœ… **Integration Workflows** - End-to-end process algorithms defined

---

Core Algorithm Implementations

1. Data Ingestion Pipeline

```
ALGORITHM DataIngestionPipeline
INPUT: file_data, source_system, data_format
OUTPUT: ingestion_result

BEGIN
    job_id = generate_uuid()

    TRY
        // Step 1: File Validation
        validation_result = validate_file_format(file_data, data_format)
        IF NOT validation_result.is_valid THEN
            THROW ValidationException(validation_result.errors)
        END IF

        // Step 2: Data Parsing
        raw_records = parse_file_data(file_data, data_format)

        // Step 3: Data Transformation
        standardized_records = []
        FOR each record IN raw_records DO
            transformed_record = transform_to_standard_schema(record)
            standardized_records.append(transformed_record)
```

```
        END FOR

        // Step 4: Quality Assessment
        quality_score = calculate_data_quality(standardized_records)

        // Step 5: Database Storage
        batch_insert_records(standardized_records)

        // Step 6: Event Publication
        publish_event("data_ingested", job_id, quality_score)

        RETURN IngestionResult(job_id, len(standardized_records), quality_score, "SUCCESS")

    CATCH Exception as e
        log_error("Ingestion failed for job " + job_id, e)
        RETURN IngestionResult(job_id, 0, 0, "FAILED")
    END TRY
END

FUNCTION validate_file_format(file_data, format)
BEGIN
    SWITCH format
        CASE "CSV":
            RETURN validate_csv_structure(file_data)
        CASE "JSON":
            RETURN validate_json_schema(file_data)
        CASE "XML":
            RETURN validate_xml_schema(file_data)
        DEFAULT:
            RETURN ValidationResult(false, "Unsupported format")
    END SWITCH
END

FUNCTION calculate_data_quality(records)
BEGIN
    total_score = 0
    completeness_score = calculate_completeness(records)
    accuracy_score = calculate_accuracy(records)
    consistency_score = calculate_consistency(records)

    total_score = (completeness_score + accuracy_score + consistency_score) / 3
    RETURN total_score
END
```

2. ML Forecasting Engine

```
ALGORITHM EnsembleForecastingEngine
INPUT: sku_id, historical_data, forecast_horizon
OUTPUT: forecast_result

BEGIN
    // Step 1: Feature Engineering
    features = engineer_features(historical_data)

    // Step 2: Model Selection
    available_models = ["LSTM", "Prophet", "ARIMA", "XGBoost"]
    model_performances = {}

    FOR each model_type IN available_models DO
        model = load_model(model_type, sku_id)
        IF model EXISTS THEN
            performance = get_model_performance(model, sku_id)
            model_performances[model_type] = performance
        END IF
    END FOR

    // Step 3: Ensemble Weight Calculation
    weights = calculate_ensemble_weights(model_performances)

    // Step 4: Generate Individual Forecasts
    individual_forecasts = {}
    FOR each model_type IN available_models DO
        IF model_type IN model_performances THEN
            model = load_model(model_type, sku_id)
            forecast = model.predict(features, forecast_horizon)
            individual_forecasts[model_type] = forecast
        END IF
    END FOR

    // Step 5: Ensemble Combination
    ensemble_forecast = combine_forecasts(individual_forecasts, weights)

    // Step 6: Confidence Intervals
    confidence_intervals = calculate_prediction_intervals(individual_forecasts, ensemble_forecast)

    RETURN ForecastResult(sku_id, ensemble_forecast, individual_forecasts, confidence_intervals, weights)
END

FUNCTION engineer_features(historical_data)
BEGIN
    features = historical_data.copy()

    // Time-based features
    features["day_of_week"] = extract_day_of_week(features["date"])
    features["month"] = extract_month(features["date"])
    features["quarter"] = extract_quarter(features["date"])
    features["is_weekend"] = is_weekend(features["date"])

    // Lag features
    FOR lag IN [1, 7, 14, 30] DO
        features["demand_lag_" + lag] = shift(features["demand"], lag)
    END FOR

    // Rolling statistics
    FOR window IN [7, 14, 30] DO
        features["demand_mean_" + window] = rolling_mean(features["demand"], window)
        features["demand_std_" + window] = rolling_std(features["demand"], window)
    END FOR

    // Seasonal decomposition
    trend, seasonal, residual = seasonal_decompose(features["demand"])
    features["trend"] = trend
    features["seasonal"] = seasonal

    RETURN features
END

FUNCTION calculate_ensemble_weights(model_performances)
BEGIN
```

```
    weights = {}
    total_inverse_error = 0

    // Calculate inverse MAPE for weighting
    FOR each model, performance IN model_performances DO
        inverse_mape = 1 / (performance.mape + 0.001)  // Add small epsilon
        weights[model] = inverse_mape
        total_inverse_error += inverse_mape
    END FOR

    // Normalize weights
    FOR each model IN weights DO
        weights[model] = weights[model] / total_inverse_error
    END FOR

    RETURN weights
END
```

3. Inventory Optimization Engine

```
ALGORITHM SafetyStockOptimization
INPUT: sku_data, service_level_target
OUTPUT: optimization_results

BEGIN
    results = {}

    FOR each sku_id, data IN sku_data DO
        // Extract parameters
        demand_mean = data.demand_forecast
        demand_std = data.demand_std
        lead_time = data.lead_time
        holding_cost = data.holding_cost_per_unit
        stockout_cost = data.stockout_cost_per_unit

        // Calculate optimal safety stock using newsvendor model
        z_score = inverse_normal_cdf(service_level_target)
        lead_time_demand_std = demand_std * sqrt(lead_time)
        optimal_safety_stock = z_score * lead_time_demand_std

        // Calculate reorder point
        reorder_point = demand_mean * lead_time + optimal_safety_stock

        // Calculate expected costs
        holding_cost_total = optimal_safety_stock * holding_cost
        expected_stockout_cost = calculate_expected_stockout_cost(
            optimal_safety_stock, demand_std, stockout_cost
        )
        total_cost = holding_cost_total + expected_stockout_cost

        results[sku_id] = {
            "optimal_safety_stock": optimal_safety_stock,
            "reorder_point": reorder_point,
            "total_cost": total_cost,
            "service_level": service_level_target
        }
    END FOR

    RETURN results
END

ALGORITHM MultiEchelonOptimization
INPUT: network_data, demand_forecasts
OUTPUT: allocation_plan

BEGIN
    // Initialize optimization model
    solver = create_linear_solver("SCIP")

    // Decision variables: inventory levels at each location
    inventory_vars = {}
    FOR each location IN network_data.locations DO
        FOR each sku IN network_data.skus DO
            var_name = "inv_" + location + "_" + sku
            inventory_vars[var_name] = solver.create_variable(0, INFINITY, var_name)
        END FOR
    END FOR

    // Transportation variables
    transport_vars = {}
    FOR each origin IN network_data.locations DO
        FOR each destination IN network_data.locations DO
            IF origin != destination THEN
                FOR each sku IN network_data.skus DO
                    var_name = "transport_" + origin + "_" + destination + "_" + sku
                    transport_vars[var_name] = solver.create_variable(0, INFINITY, var_name)
                END FOR
            END IF
        END FOR
    END FOR

    // Demand constraints
    FOR each location IN network_data.locations DO
        FOR each sku IN network_data.skus DO
            demand = demand_forecasts[location][sku]
            inv_var = inventory_vars["inv_" + location + "_" + sku]

            // Inbound transportation
            inbound_sum = 0
            FOR each origin IN network_data.locations DO
                IF origin != location THEN
                    transport_var = transport_vars["transport_" + origin + "_" + location + "_" + sku]
                    inbound_sum += transport_var
                END IF
            END FOR

            // Outbound transportation
            outbound_sum = 0
            FOR each destination IN network_data.locations DO
                IF destination != location THEN
                    transport_var = transport_vars["transport_" + location + "_" + destination + "_" + sku]
                    outbound_sum += transport_var
                END IF
            END FOR

            // Demand satisfaction constraint
            solver.add_constraint(inv_var + inbound_sum - outbound_sum >= demand)
        END FOR
    END FOR
```

```
    // Capacity constraints
    FOR each location IN network_data.locations DO
        capacity = network_data.capacities[location]
        capacity_sum = 0
        FOR each sku IN network_data.skus DO
            inv_var = inventory_vars["inv_" + location + "_" + sku]
            capacity_sum += inv_var
        END FOR
        solver.add_constraint(capacity_sum <= capacity)
    END FOR

    // Objective function: minimize total cost
    objective = solver.create_objective()

    // Holding costs
    FOR each location IN network_data.locations DO
        FOR each sku IN network_data.skus DO
            inv_var = inventory_vars["inv_" + location + "_" + sku]
            holding_cost = network_data.holding_costs[location][sku]
            objective.set_coefficient(inv_var, holding_cost)
        END FOR
    END FOR

    // Transportation costs
    FOR each origin IN network_data.locations DO
        FOR each destination IN network_data.locations DO
            IF origin != destination THEN
                FOR each sku IN network_data.skus DO
                    transport_var = transport_vars["transport_" + origin + "_" + destination + "_" + sku]
                    transport_cost = network_data.transport_costs[origin][destination]
                    objective.set_coefficient(transport_var, transport_cost)
                END FOR
            END IF
        END FOR
    END FOR

    objective.set_minimization()

    // Solve optimization problem
    status = solver.solve()

    IF status == OPTIMAL THEN
        allocation_plan = extract_solution(solver, inventory_vars, transport_vars)
        RETURN allocation_plan
    ELSE
        THROW OptimizationException("Failed to find optimal solution")
    END IF
END
```

4. Real-time Analytics Engine

```
ALGORITHM RealTimeAnalyticsProcessor
INPUT: streaming_data
OUTPUT: analytics_results

BEGIN
    WHILE streaming_data.has_next() DO
        batch = streaming_data.get_next_batch()

        FOR each record IN batch DO
            // Process individual record
            processed_record = process_analytics_record(record)

            // Update running metrics
            update_running_metrics(processed_record)

            // Check for anomalies
            IF is_anomaly(processed_record) THEN
                trigger_alert(processed_record)
            END IF

            // Update dashboards
            update_real_time_dashboard(processed_record)
        END FOR

        // Batch-level processing
        batch_metrics = calculate_batch_metrics(batch)
        store_batch_metrics(batch_metrics)

        // Trigger periodic reports
        IF should_generate_report() THEN
            generate_periodic_report()
        END IF
    END WHILE
END

FUNCTION calculate_forecast_accuracy_metrics(actual_values, predicted_values)
BEGIN
    n = len(actual_values)

    // Mean Absolute Error (MAE)
    mae = 0
    FOR i = 0 TO n-1 DO
        mae += abs(actual_values[i] - predicted_values[i])
    END FOR
    mae = mae / n

    // Mean Absolute Percentage Error (MAPE)
    mape = 0
    FOR i = 0 TO n-1 DO
        IF actual_values[i] != 0 THEN
            mape += abs((actual_values[i] - predicted_values[i]) / actual_values[i])
        END IF
    END FOR
    mape = (mape / n) * 100

    // Root Mean Square Error (RMSE)
    rmse = 0
    FOR i = 0 TO n-1 DO
        rmse += (actual_values[i] - predicted_values[i])^2
    END FOR
    rmse = sqrt(rmse / n)

    RETURN {
        "mae": mae,
        "mape": mape,
        "rmse": rmse,
        "sample_count": n
```

```
    }
END
```

5. Integration Workflow Engine

```
ALGORITHM ERPIntegrationWorkflow
INPUT: integration_config, data_payload
OUTPUT: integration_result

BEGIN
    TRY
        // Step 1: Authentication
        auth_token = authenticate_with_erp(integration_config.credentials)

        // Step 2: Data Transformation
        transformed_data = transform_data_for_erp(data_payload, integration_config.mapping)

        // Step 3: Validation
        validation_result = validate_erp_data(transformed_data, integration_config.schema)
        IF NOT validation_result.is_valid THEN
            THROW ValidationException(validation_result.errors)
        END IF

        // Step 4: API Call with Retry Logic
        max_retries = 3
        retry_count = 0

        WHILE retry_count < max_retries DO
            TRY
                response = call_erp_api(
                    integration_config.endpoint,
                    transformed_data,
                    auth_token
                )

                IF response.status_code == 200 THEN
                    // Success - break retry loop
                    BREAK
                ELSE
                    // Handle specific error codes
                    IF response.status_code == 401 THEN
                        // Re-authenticate
                        auth_token = authenticate_with_erp(integration_config.credentials)
                    END IF

                    retry_count += 1
                    IF retry_count < max_retries THEN
                        wait(exponential_backoff(retry_count))
                    END IF
                END IF

            CATCH NetworkException as e
                retry_count += 1
                IF retry_count < max_retries THEN
                    wait(exponential_backoff(retry_count))
                ELSE
                    THROW IntegrationException("Network error after retries: " + e.message)
                END IF
            END TRY
        END WHILE

        // Step 5: Response Processing
        IF response.status_code == 200 THEN
            processed_response = process_erp_response(response.data)
            log_successful_integration(integration_config.system_name, processed_response)
            RETURN IntegrationResult("SUCCESS", processed_response)
        ELSE
            THROW IntegrationException("ERP integration failed: " + response.error_message)
        END IF

    CATCH Exception as e
        log_integration_error(integration_config.system_name, e)
        RETURN IntegrationResult("FAILED", e.message)
    END TRY
END

FUNCTION exponential_backoff(retry_count)
BEGIN
    base_delay = 1000  // 1 second in milliseconds
    max_delay = 30000  // 30 seconds maximum

    delay = min(base_delay * (2^retry_count), max_delay)
    jitter = random(0, delay * 0.1)  // Add 10% jitter

    RETURN delay + jitter
END
```

6. Performance Monitoring Algorithm

```
ALGORITHM PerformanceMonitoringSystem
INPUT: system_metrics_stream
OUTPUT: monitoring_alerts

BEGIN
    // Initialize monitoring thresholds
    thresholds = {
        "response_time_ms": 2000,
        "error_rate_percent": 1.0,
        "cpu_usage_percent": 80.0,
        "memory_usage_percent": 85.0,
        "forecast_accuracy_mape": 25.0
    }

    // Initialize sliding window for metrics
    metrics_window = SlidingWindow(size=100)

    WHILE system_metrics_stream.has_data() DO
        current_metrics = system_metrics_stream.get_next()

        // Add to sliding window
        metrics_window.add(current_metrics)

        // Calculate rolling averages
        rolling_metrics = calculate_rolling_metrics(metrics_window)

        // Check thresholds
        FOR each metric, value IN rolling_metrics DO
            IF metric IN thresholds THEN
```

```
                threshold = thresholds[metric]

                IF value > threshold THEN
                    alert = create_alert(metric, value, threshold, "CRITICAL")
                    send_alert(alert)
                ELSE IF value > (threshold * 0.8) THEN
                    alert = create_alert(metric, value, threshold, "WARNING")
                    send_alert(alert)
                END IF
            END IF
        END FOR

        // Update real-time dashboard
        update_monitoring_dashboard(rolling_metrics)

        // Store metrics for historical analysis
        store_metrics(current_metrics, rolling_metrics)
    END WHILE
END
```

Algorithm Complexity Analysis

Time Complexity

- **Data Ingestion**: O(n) where n is number of records
- **Forecasting Engine**: O(m Ã— k Ã— h) where m=models, k=SKUs, h=horizon
- **Safety Stock Optimization**: O(s) where s is number of SKUs
- **Multi-Echelon Optimization**: O(lÂ² Ã— s) where l=locations, s=SKUs
- **Real-time Analytics**: O(1) per record, O(b) per batch

Space Complexity

- **Feature Engineering**: O(n Ã— f) where n=records, f=features
- **Model Storage**: O(m Ã— p) where m=models, p=parameters
- **Optimization Variables**: O(l Ã— s) for inventory variables
- **Metrics Storage**: O(w) where w=window size

Conclusion

This pseudocode document provides executable algorithms for all core functionality of the Supply Chain Demand Forecasting Platform. The algorithms are designed for:

- **Scalability**: Efficient processing of large datasets
- **Reliability**: Error handling and retry mechanisms
- **Performance**: Optimized time and space complexity
- **Maintainability**: Clear, modular algorithm design

These algorithms can be directly translated into production code using the technology stack defined in the architecture documents.
